Development and Evaluation of Interactive Courseware for Visualization of Graph Data Structure and Algorithms

Thomas E. Beutel
Nova Southeastern University, tom_beutel@yahoo.com

Development and Evaluation of Interactive Courseware for Visualization of Graph Data Structures and Algorithms

by

Thomas E. Beutel

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

School of Computer and Information Sciences
Nova Southeastern University

1997

We hereby certify that this dissertation, submitted by Thomas E. Beutel, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

[illegible] 05/16/97

Gertrude W. Abramson, Ed.D
Chairperson of Dissertation Committee
Date.

[illegible] 5/19/97

Michael J. Laszlo, Ph.D.
Dissertation Committee Member
Date

[illegible] 05/16/97

Junping Sun, Ph.D.
Dissertation Committee Member
Date

Approved: [illegible] 5/29/97

Edward Lieblein, Ph.D.
Dean, School of Computer and Information Sciences
Date

School of Computer and Information Sciences
Nova Southeastern University

1997

Certification Statement

I hereby certify that this dissertation constitutes my own product and that the words or ideas of others, where used, are properly credited according to accepted standards for professional publications.

Signed Thomas E. Beutel

Thomas E. Beutel

Abstract

The primary goal of this dissertation was to develop and pilot test interactive, multimedia courseware which would facilitate learning the abstract structures, operations, and concepts associated with graph and network data structures in Computer Science. Learning objectives and prerequisites are presented in an introduction section of the courseware and a variety of learning activities are provided including tutorials, animated demonstrations, interactive laboratory sessions, and self-tests. Courseware development incorporated principles and practices from software engineering, instructional design, and cognitive learning theories. Implementation utilized an easy-to-use authoring tool, NeoBook Professional (1994), to create the overall framework and the user interfaces, and Microsoft QuickBASIC 4.5 (1990) to program the interactive animated demonstrations and laboratory exercises. A major emphasis of the courseware is the use of simple interactive, animated displays to demonstrate the step-by-step operation of graph and network algorithms such as depth-first traversal, breadth-first traversal, shortest path, minimum spanning tree and topological ordering.

Acknowledgements

Successful completion of any significant piece of work is never a solitary effort. The knowledge, wisdom and example of others who have worked before us; the support of colleagues, peers, family and friends; and the opportunities and gifts given us by God all come together to help us achieve our goals.

In particular, I would like to thank Dr. Trudy Abramson for her example as an educator and educational technologist, and her help and encouragement as professor and Dissertation Chairperson. As a Computer Scientist, I have truly appreciated the support, ideas and contributions of Dr. Mike Laszlo and Dr. Junping Sun, the Computer Scientists on my Dissertation committee. Dr. Steve Terrell and Dr. Marlyn Littman challenged me during courses in research, educational theory, multimedia, and networks. The reading, writing, and thinking these two professors assigned provided valuable experience for the dissertation process.

Mount Vernon Nazarene College supported my doctoral work financially and in other ways which made it easier to balance working and full-time graduate work. Dr. E. LeBron Fairbanks, college President, and Dr. Jack Anderson, Academic Dean, have been very encouraging and supportive throughout the entire doctoral program. The Computer Science students at Mount Vernon Nazarene College have shown understanding and patience as I balanced teaching with research and writing.

I would also like to acknowledge my parents, Mr. and Mrs. Frederick Beutel. Their belief in the value of education and the importance of doing your best helped to equip me for this difficult task. Most of all, I thank my wife Wendy, and my sons, Danny and Adam, who have sacrificed in many ways to allow me to pursue my graduate work and complete this dissertation.

Table of Contents

List of Tables

List of Figures

Chapter I

Introduction

The idea that technology can be used to improve education is not a new one. Noblitt (1995) points out that blackboards and books, as well as computers, constitute a level of educational technology and that new technologies are incorporated into the educational process whenever they improve teaching and learning. The use of computer-based instruction (CBI) has been found to improve motivation and achievement (Podell, Kaminsky & Cuismano, 1993), to increase retention and decrease learning time (Galbreath, 1994), and to encourage active student learning and higher-order cognitive activities (Weiss, 1994). In a study conducted at Wright State University in Dayton, Ohio, students indicated that computer-assisted presentations "make classes more interesting, exciting", "make class more organized", and "make...information clearer" (Sammons, 1995, p.68). Smith and Debenham (1993) have developed interactive instructional software with the goals of reducing educational costs and increasing access to educational materials.

Despite the many benefits of CBI which have been demonstrated or presumed, the technology has not had a revolutionary impact on education (Moore, 1994; Solomon, 1994). There are a number of possible reasons including financial costs, time costs, resistance to change, and others (Solomon, 1994). The development and evaluation of an interactive, animated courseware package for learning about graphs and networks in Computer Science described in this report is intended to demonstrate both the effectiveness of appropriate CBI and an effective methodology for design and development of CBI applications. Use of the developed courseware will enhance professional practice in Computer Science education as well as student learning of abstract structures, operations and concepts. Rigorous design and development of the courseware demonstrates what is needed to produce instructional software which meets measurable learning objectives.

Problem Scope

The problem to be addressed is the development of instructional software which facilitates learning abstract structures, operations, and concepts by Computer Science students. To insure effectiveness, the software not only deals with subject matter content accurately and appropriately, but was developed according to established software engineering and instructional design principles, utilizes current educational technology effectively, and is guided by cognitive learning theories. To accomplish the development in a reasonable time development tools were used which facilitated the development process.

Abstractions in Computer Science

As a relatively young discipline, Computer Science has struggled to develop effective and efficient methodologies as well as its theoretical foundations. The ever-increasing use of computer technology in business, education, entertainment, and most fields of endeavor, has challenged Computer Scientists to improve the way in which computer software is designed and developed. According to Naps and Pothering (1992), “In the late 1950’s and early 1960’s there was a widely held belief that designing effective software systems was something akin to an occult art” (p.46). The result of this approach was often failure or, at best inefficient and costly development efforts.

The solution to these problems was to adopt methodologies from the field of engineering. In general, an engineer begins the design process by determining the needs of the ultimate user, then proceeds through a series of designs, beginning with a purely conceptual, or abstract, model and culminating in a detailed design (Naps & Pothering, 1992). The focus at early stages of the design on abstract representations of the end product allows the engineer to concentrate on the essential features and to experiment with a variety of implementations.

All nontrivial computer programs are concerned with two types of entities: data structures and algorithms (Naps & Pothering, 1992; Riley, 1990). The term data structures refers to how various types and pieces of data are organized within computer memory. Algorithms are the operations which manipulate data and produce results. One aspect of applying engineering methods to computer programming is to view data structures and algorithms as abstractions, or conceptual models. This allows the software designer to focus on the essential elements of the data structures and algorithms and to experiment with various implementations, in much the same way as the engineer does with an engineering design. The goal in both instances is to find the most effective and efficient implementation for the problem at hand.

In addition to the implementation benefits of approaching data structures and algorithms as abstractions, it is necessary to do so because the data structures and algorithms do not have a physical existence of their own. The ultimate implementation of both data structures and algorithms in a computer is manifested in electronic components and circuits. Aho and Ullman (1992) note that while most scientists deal with the world as it exists, "Computer scientists, on the other hand, must create abstractions of real-world problems that can be represented and manipulated inside a computer" (p.1). Thus, even the data structures and algorithms which programmers seek to implement are abstractions of the underlying computer architecture (Naps & Pothering, 1992). By working with abstractions of these entities, Computer Scientists are really working with abstractions of abstractions.

Difficulty in Learning Abstract Structures, Operations and Concepts

Often, abstractions and conceptual models are difficult to grasp. An example of this from everyday life would be viewing two paintings, one abstract, one realist. Most people can easily grasp what is being portrayed in the realist painting, but have difficulty with the abstract. The same can be true with Computer Science students, especially Freshman and

Sophomore college students confronting abstract data types and algorithms for the first time. When one realizes that these entities are abstractions of abstractions, the difficulty is compounded.

Authors of traditional Computer Science textbooks recognize this problem. In the preface to a current Computer Science text, the authors point out that the text contains "extensive figures and graphic documentation: These allow students to visualize the effect of algorithms on data" (Naps & Nance, 1995, p.xxiii). The use of figures is intended to make real both the abstract data structures and the abstract algorithms. The text *Introduction to Algorithms* by Cormen (1990) provides a thorough coverage of algorithms, including almost 200 pages on graphs and networks alone. The text makes extensive use of sequences of diagrams to demonstrate the effect of an algorithm on a particular data structure.

While dealing with abstractions in Computer Science is a problem in general, the difficulties escalate when students are exposed to complex, logical data structures and their associated algorithms. Simple data structures typically define the relationship of one element to another in terms of physical ordering. A familiar example would be a grocery list in which items are usually written one above the other in columnar form. The simplest equivalent Computer Science structure is a one-dimensional array which is a linear, physically-ordered structure.

A logical data structure, on the other hand, is one in which the relationship of one element of the data structure to another is not physically determined. Instead, each item includes information about which other items are related to it. The structure of these logical data structures is more complex and harder to visualize than that of simpler structures. Examples of such structures include linked lists, binary trees, and graphs and networks. Because the structures themselves are more complex, the algorithms which manipulate these structures are also more complex. Thus, the general problem of dealing with abstractions is made more difficult by having to deal with complex abstractions.

Researchers and practitioners emphasize the difficulty of learning foreign concepts. "This is a particularly challenging task in science and engineering disciplines where the subjects and concepts are often complex, multidimensional processes and phenomena" (Aukstakalnis & Mott, 1996, p.14). These authors discuss the use of computer simulation and visualization techniques as tools to enhance student learning of complex, abstract concepts, structures, and operations in science and engineering.

Overview of Graphs and Networks

Leonhard Euler, born in Switzerland, "was the most prolific mathematician of his generation" (Miller, Heeren, & Hornsby, 1990, p.99). Among his many original contributions to the field of mathematics is the formalization of the study of graphs and networks. Euler's solution of a popular problem, the Koenigsburg Bridge problem, marks the beginning of the formal study of graphs and networks (Miller, Heeren, & Hornsby, 1990).

A primary emphasis in Computer Science is that of organizing data items and the relationships among items. According to Aho and Ullman (1992), "graphs are a versatile model for organizing data" (p. 435). They define a graph as "a set of points (called nodes) connected by lines (called edges)." Naps and Pothering (1992) state that "graphs are the most general data structures" (p.421), because they can be used to represent not only one-to-one and one-to-many relationships, but also the many-to-many relationships found in a wide variety of problems.

To clarify these concepts, consider a traditional nuclear family with two or more children. There is a one-to-one relationship between the husband and wife. Each husband has one wife and each wife has one husband. A one-to-many relationship exists between each parent and their children. Each child has only one parent, but each parent has more than one child. If there are at least three children, then there is a many-to-many relationship among the siblings. Each child has more than one (many) siblings.

Graphs and networks can be used to solve problems in transportation, communication, scheduling and many other areas. In general, any problem which consists of entities of some sort which are related to one another in some way can be represented by a graph. The entities are represented as nodes and the relationships as edges connecting the nodes. Networks differ from graphs in that each edge has an associated weight, which indicates the cost of traversing the edge between two nodes. An example would be the distance between two cities.

Nodes in a graph or network may be directly connected by an edge. If there is not a direct connection, there may be a path, a sequence of edges, which indirectly connects two nodes. A familiar analogy would be that of a direct flight between two cities as opposed to a route which involves a connection at some intermediate city.

Solutions to graph and network problems are carried out by applying appropriate algorithms. Common graph and network algorithms include depth-first traversal, breadth-first traversal, shortest path, minimum spanning tree, and topological sort (Naps & Pothering, 1992). The traversal algorithms generally process each node in a graph or network in a particular order. Shortest path algorithms determine the minimum cumulative edge weight between two nodes; whereas the minimum spanning tree represents the minimum total edge weight to connect all nodes in some fashion. An example of the latter would be to find the least amount of network cable to connect a number of computers, where there would not be direct connections among all computers on the network.

There are a number of techniques for implementing graphs and networks in computer programs. A popular method is that of the adjacency matrix (Naps & Pothering, 1992; Riley, 1990). An adjacency matrix can be thought of as a two-dimensional array, or table, listing all nodes both horizontally and vertically. Nodes which are directly connected are indicated by placing a nonzero value in the array element which corresponds to the two nodes.

Elements of Educational Technology

Educational technology encompasses a variety of disciplines and areas of interest including multimedia, hypermedia, interactivity, and authoring. Multimedia is a catchword of the nineties. There seem to be as many definitions of multimedia as there are books and articles about multimedia. Definitions of multimedia include a variety of key elements. Galbreath (1992) insists that user interactivity must be included in any definition of multimedia. Buford (1994) says, "It is the simultaneous use of data in different media forms (voice, video, text, animations, etc.) that is called multimedia" (p.2).

Sweeters (1994) claims that, “Computer-based learning (CBL) has been enhanced by the addition of multimedia capabilities” (p.47). Multimedia features can be incorporated into tutorials, educational databases, simulations, and educational games (Sweeters, 1994). The incorporation of multimedia technology into teaching and learning environments empowers students who do not learn well using traditional methodologies (Noblitt, 1995).

In 1945, Vannevar Bush, science advisor to President Franklin Roosevelt, conceived of a collection of scientific literature which could be accessed in a nonsequential manner, thus facilitating research (Maddux, 1992; Myers, 1994). More recently, this concept has come to be known as hypertext. In a hypertext system users may "skip around" to learn about a related topic or to go into greater depth in some portion of a topic. They are not constrained to proceed through the presented material in a strictly sequential manner.

Information in a hypertext system is stored in a network of nodes, each node containing an appropriate unit of information. Links between various nodes form a network structure. Users of a hypertext system can navigate from node to node according to the links provided by the hypertext designer (Lanza, 1991). When information is stored and accessible in forms other than text, such as audio, graphic images, animation or video, the term hypermedia is normally used, rather than hypertext (Maddux, 1992). A hypermedia system allows nonsequential access to a body of multimedia data.

Educational applications of hypermedia include information retrieval systems and computer-based constructivisitic learning environments (Jonassen & Grabinger, 1993). Lennon and Maurer (1994) discuss the use of hypermedia systems in lecturing situations. In general, the interactivity and the potential for learners to choose their own paths through a body of material are seen as strengths of hypermedia-based learning systems, allowing learning to be more student-centered and encouraging active student participation in the learning process (Johnson & Grover, 1993).

The appeal of multimedia and the promise of interactive, hypermedia systems might be of little practical consequence except for the fact that authoring systems which allow educators to develop high-quality, high-level instructional software are becoming increasingly available. Entry-level multimedia authoring tools allow developers to create presentations and simple interactive applications without programming (Lynch, 1995). Hypermedia learning programs which motivate learners and "promote application, analysis, synthesis, and evaluation" can be developed with commercially available authoring systems (Abramson, 1993, p.6). Lynch (1995) describes the assembly of a variety of authoring tools into an authoring environment which is oriented to the skills of the developer and to the types of multimedia packages being developed.

Issues Related to the Development of Instructional Software for Computer Science

The need for techniques and tools for dealing with complex abstractions in Computer Science is well understood. Nevertheless, Computer Scientists are like shoemakers whose children go barefoot, they often work with less than ideal design and development tools to develop computer-based applications. Shneiderman (1992) observes, "There is a great thirst for knowledge, software tools, design guidelines, and testing techniques" (p.34) with respect to user interface design. This single subfield of Computer Science is representative of other areas.

Lack of Computer-based Educational Applications in Computer Science

Computer Scientists need to understand the benefits of applying computer-based educational technology to their own field, and to see examples of effective courseware. An informal examination of several educational software catalogs showed that the majority of commercial educational software is for K-12 rather than for the college level. In addition, those products which are appropriate for college students cover chemistry, mathematics, and other areas, but there are virtually no titles for Computer Science education.

Hirschbuhl and Faseyitan (1994) confirm this observation, stating, "Unfortunately, current studies indicate that the use of computers for instructional purposes has not yet hit the mainstream of university education" (p.64). Likewise, Heterick (1994) says, "Information technology is everywhere, but not enough of it is in our colleges and universities." Twigg (1995) asserts, "despite the potential offered by such materials, many in our community have noted the failure of most institutions--even those who have worked hard to create them--to integrate their use into their academic programs." There does not seem to be, at this time, a significant market for instructional software at the college level (Twigg, 1996).

The majority of software which is available for use in Computer Science courses can be categorized as tools. Algorithms animators (Wilson, Aiken, & Katz, 1996) are a prime example. While these software products are helpful, they are not instructional software. Like microscopes in Biology, they help the student do a task or observe behavior. However, such activities are not guided by built-in educational objectives or designed according to instructional design principles.

Lack of High-quality, High-level Computer-based Educational Applications

Although courseware is lacking in Computer Science more than in some academic areas, effective courseware beyond the drill-and-practice level is not widely available in any field. Johnson and Grover (1993) report, "For more than a decade of classroom microcomputer use, teachers, reviewers, critics, and software evaluators have consistently reported that although there are some excellent programs, the overall quality of interactive instruction is relatively poor" (p.5).

Articles in popular magazines such as *Time* and *Byte* as well as educational journals continue to decry the lack of effective, high-level educational software. Wallis (1995) observes, "Computers are indeed everywhere in American schools, but they are generally used as little more than electronic workbooks for drill" (p.49). Johnson (1995), who quotes Wallis, says "she shares a common concern about the lack of evidence indicating that a learning revolution has taken place" (p.1).

Courseware should be learner-centered, encouraging active participation by learners (Johnson & Grover, 1993). Abramson (1993) encourages courseware development which promotes higher-level learning including application, analysis, synthesis, and evaluation and asserts that, "Traditional applications such as drill-and-practice programs, and those in which pressing the mouse replaces turn-the-page, do not use the power and waste the glory of hypermedia" (p.6).

Courseware Design and Development Often Does not Follow Established Practice

An important aspect of interactive, multimedia courseware is the user interface. Although much research has been done in the area of human-computer interface and guidelines for effective user interface design have evolved (Shneiderman, 1992), few of these results have been applied to the development of educational software (Jones, Farquhar & Surrey, 1995). Along these same lines, traditional courseware does not incorporate newer learning theories based on cognitive models, particularly as they impact the user interface of courseware applications (Jones et al., 1995).

Inadequate user interface design of courseware is only one aspect of a larger design and development problem. Often courseware applications are created with little regard for proven instructional design or software development practices. There is a need to follow established software engineering techniques in the development of courseware. These techniques begin with determining user requirements, and proceed through design, implementation, testing, and distribution (Luther, 1994). In addition to following established software engineering practice, developers must utilize instructional design practices and techniques (Ehrlich & Reynolds, 1992). A key element of instructional design is determining learner needs and educational objectives (Ehrlich & Reynolds, 1992; Gagne`, Briggs & Wager, 1992).

Development of Multimedia Courseware Requires Multiple Skills and Tools

Design and development of effective interactive courseware is a difficult task. Even with appropriate authoring tools available, utilizing appropriate guidelines for user interface design, and employing applicable software engineering and instructional design principles, the task of developing effective courseware requires diverse skills and abilities (Ehrlich & Reynolds, 1992). Solomon (1994), says that, “Developing quality multimedia courseware is too difficult for 98% of all faculty” (p.83). A development team or subject matter expert who is educated in educational technology is needed to produce good quality applications.

In contemplating the development of interactive courseware for learning about graphs and networks in Computer Science, several implementation difficulties became apparent. A major problem in learning complex, abstract structures is the learner's difficulty in visualizing the dynamic operations which manipulate these structures. Therefore, necessary parts of the courseware are dynamic elements which allow the learner to see the operations "in action."

Authoring packages which are intuitive and easy-to-use, are not powerful enough to incorporate the types of interactive animations described above. More powerful authoring systems are often difficult to use (Lynch, 1995). For a developer with a Computer Science background, it is easier to program interactive, dynamic displays than to learn a new authoring system. The difficulty in this approach is that programming attractive, easy-to-use user interfaces is not an easy task. The solution to this impasse is to combine the use of an easy-to-use authoring tool and a programming language, using the authoring system to create the user interface and overall courseware structure, and the programming language to create the interactive animations. Thus, the courseware is a hybrid application capitalizing on the strengths of authoring systems and traditional programming languages.

Multimedia Courseware Development is Time-Consuming

Another significant issue relating to the production of instructional software is the time required to carry out the design and development. Solomon (1994) observes, "Even for that rare individual with all of the necessary talents to make a program a success, it takes too much time" (p.83). Developing courseware which is more than a simple slide show or drill-and-practice application requires significant design and development time (Sammons, 1995).

Some argue that CBI saves teachers time, since students can work independently. However, if courseware is to be developed by teachers, the time required for design and development far outweighs any time saved during instruction (Stoddart & Niederhauser, 1993).

Goals and Significance of Research

Development of interactive, multimedia courseware for learning about complex, abstract data structures and algorithms in Computer Science will enhance professional practice and improve understanding in the area of interactive courseware design and development, and will facilitate meeting specified learning objectives in the area of Computer Science education. These goals are accomplished by: (1) applying the concepts and practices of software engineering, instructional design, and cognitive learning theory in the development of interactive, multimedia courseware; (2) effectively using authoring tools and conventional programming languages in courseware development; and (3) conducting formative evaluations of the courseware throughout the development process.

Learning Objectives

The goal of this dissertation was to develop courseware which will provide interactive, animated graphical representations of graphs and networks which facilitate learning the abstract structures, operations and concepts related to graph and network data structures and related algorithms by Computer Science students. Specific learning objectives for the completed courseware implementation are given in Table 1.

Table 1: Courseware Learning Objectives

1. The student will **define the following terms** related to graphs and networks:

    a. acyclic graph
    b. adjacency matrix
    c. breadth-first traversal
    d. cycle
    e. degree
    f. depth-first traversal
    g. digraph
    h. edge
    i. graph
    j. minimal spanning tree
    k. network
    l. node
    m. path
    n. topological ordering

2. The student will **explain the following concepts** related to graphs and networks:

    a. A graph or network is comprised of a set of nodes and edges.
    b. Traversing a graph or network using a breadth-first technique.
    c. Traversing a graph or network using a depth-first technique.
    d. The use of an adjacency matrix as a means of implementing a graph or network data structure.
    e. Finding the shortest path between two nodes in a network.
    f. Finding the minimum spanning tree of a network.
    g. Performing a topological ordering of the nodes in a network.

3. The student will **implement an adjacency matrix** as a means of implementing a graph or network.

4. The student will **implement common primitive operations** for graphs and networks including:

    a. creating a graph/network
    b. adding an edge
    c. removing an edge
    d. adding a node
    e. performing a depth-first traversal
    f. performing a breadth-first traversal.

5. The student will **implement additional operations** for networks including:

    a. shortest path
    b. minimum spanning tree
    c. topological ordering.

6. The student will recognize and describe problems amenable to solution using graphs and networks.

7. The student will design a computer program which uses graphs or networks to solve a specific problem.

Surveys were conducted to determine student attitudes about the usability and effectiveness of the courseware. Accomplishment of learning objectives will typically be

2. The student will **explain the following concepts** related to graphs and networks:

    a. A graph or network is comprised of a set of nodes and edges.
    b. Traversing a graph or network using a breadth-first technique.
    c. Traversing a graph or network using a depth-first technique.
    d. The use of an adjacency matrix as a means of implementing a graph or network data structure.
    e. Finding the shortest path between two nodes in a network.
    f. Finding the minimum spanning tree of a network.
    g. Performing a topological ordering of the nodes in a network.

3. The student will **implement an adjacency matrix** as a means of implementing a graph or network.

4. The student will **implement common primitive operations** for graphs and networks including:

    a. creating a graph/network
    b. adding an edge
    c. removing an edge
    d. adding a node
    e. performing a depth-first traversal
    f. performing a breadth-first traversal.

5. The student will **implement additional operations** for networks including:

    a. shortest path
    b. minimum spanning tree
    c. topological ordering.

6. The student will recognize and describe problems amenable to solution using graphs and networks.

7. The student will design a computer program which uses graphs or networks to solve a specific problem.

Surveys were conducted to determine student attitudes about the usability and effectiveness of the courseware. Accomplishment of learning objectives will typically be determined by examinations and homework assignments as the courseware is used in conjunction with normal classroom presentations and assignments.

Using Software Engineering Methodology in Courseware Development

A number of different software engineering models are used in the development of computer software. These include the build-and-fix model, the waterfall model, the rapid prototyping model, the incremental model, and the spiral model (Schach, 1993). Despite differences in these models, all incorporate certain broad phases: (1) requirements analysis, (2) specification, (3) planning, (4) design, (5) implementation, and (6) integration. Appropriate documentation is produced in each phase.

Researchers and practitioners in the area of the design and development of CBI also define various phases of the development process. Luther (1994) identifies the following design and development phases of courseware authoring: (1) concept, (2) design, (3) obtaining materials, (4) assembly, (5) testing, and (6) distribution. Alessi and Trollip (1991) organize CBI design and development into the areas of (1) preparation, (2) design, (3) flowcharting, (4) storyboarding, (5) programming and support materials preparation, and (6) evaluation.

There is not a one-to-one correspondence among the various methodologies, either in terms of specific activities or in terms of the sequence of activities, although similarities exist. The comparison of these three methodologies shown in Table 2 demonstrates a rough correspondence among the three methodologies.

Table 2: Comparison of Design and Development Methodologies

Software Engineering	**Authoring**	**CBI Development**
Requirements Analysis, Specification, Planning	Concept, Obtaining Materials	Preparation
Design	Design	Design, Flowcharting, Storyboarding
Implementation, Integration	Assembly, Testing, Distribution	Programming, Support Materials, Evaluation

Some similarities among the methodologies are apparent while others require probing into the details of each phase of each model. For example, the preparation phase of Alessi and Trollip (1991) includes the individual activities of (1) determining needs and goals, (2) collecting resources, (3) learning content, and (4) generating ideas for the most effective way of presenting the instruction to meet the needs and goals.

Compare these activities with the first and third phases described in Luther (1994): concept and obtaining materials. In the concept phase objectives and the type of application are defined. Individual media units including text, graphics, and other formats which may be used are collected. The activities in both methodologies are similar.

In the software engineering methodology user needs are assessed in the requirements phase, and the functional characteristics of the application are defined in the specification phase. Again, the basic goals, as well as some of the specific activities, of these phases are similar to the other methods.

There are differences among these methodologies, some of which are significant. One difference in the first stage of development is the explicit identification of a planning activity in the software engineering method. All development methodologies should incorporate a planning phase in which specific development activities are identified and schedules and budgets are developed. Neither Luther (1994) nor Alessi and Trollip (1991) specifically discusses the need for project planning. Developers should be aware of the need for planning in the early stages of the development process.

Another apparent difference among the models is that testing or evaluation, explicitly presented in the authoring and CBI development methodologies, is not identified separately in the software engineering method. Schach (1993) emphasizes throughout his discussion of the software development process that testing should be an integral part of each phase of development. This is an important principle which promotes early discovery and correction of fallacies, errors, or misunderstandings. If evaluation is not conducted until the application is completed, errors made in the requirements analysis phase or design phase will have propagated throughout the implementation. They are likely to have a significant impact on the implementation and result in costly and time-consuming corrections. Appropriate evaluation should be incorporated throughout the development process, not deferred until implementation is complete.

Integrating Instructional Design Techniques into Courseware Development

In addition to following established principles and techniques of software engineering, courseware development must incorporate the practices of instructional design. Lanza (1991) says that, "The apparent ease of development and production of hypertext instructional software could lead developers to disregard methodological and theoretical design issues" (p.18). A well-designed instructional unit, whether hypermedia or some other format, must communicate learning objectives, provide opportunities to practice techniques, and allow for assessment as well as presenting information and examples (Sweeters, 1994).

Ehrlich & Reynolds (1992) describe a "design model [which] is derived from traditional instructional system design models" (p.282). The focal point of instructional design is to begin with the needs and goals of the intended learners. Gagne`, Briggs, and Wager (1992) describe the stages in designing and developing instructional systems, not necessarily computer-based, as beginning with "analysis of needs, goals, and priorities" (p.5).

Another important aspect of instructional design is that of defining the desired learning outcomes. These should be stated in measurable form, so that learners and teachers know when the outcomes have been accomplished (Ehrlich & Reynolds, 1992; Gagne`, Briggs & Wager, 1992). The implications for courseware developers are that the courseware must be designed to satisfy the needs and goals of the learner, and should effectively enable the learner to reach stated measurable learning outcomes.

Alessi and Trollip (1991) base their development process on an expository model of education which consists of four parts: (1) presenting information, (2) guiding the student, (3) practicing by the student, and (4) assessing student learning. While the elements of this model focus on certain important aspects of learning, they omit other important aspects of instructional design. Gagne`s events of instruction enumerate nine separate elements essential to effective learning (Sweeters, 1994). These are listed in Table 3. The model proposed by Alessi and Trollip corresponds to events 4 through 8 of Gagne``s model (practice incorporates both eliciting expected performance and feedback). The other events, gaining learner attention, stating objectives, recalling prerequisites, and enhancing retention are not dealt with.

While it is true that computer-based instruction is not necessarily intended to stand alone, it is possible and desirable to consciously and effectively incorporate more of Gagne``s model into courseware.

Table 3: Gagne``s Events of Instruction

1. Gain learner attention
2. State objectives
3. Recall prerequisites
4. Present information
5. Provide learning guidance
6. Elicit expected performance
7. Provide feedback
8. Assess performance
9. Enhance retention

Applying Cognitive Learning Theories to Guide Design

Current educational learning theories focus on cognitive models rather than behaviorist models to explain the learning process (Lanza, 1991; Cortinovis, 1992). According to cognitive learning theory, learning is an active process whereby knowledge is built upon existing knowledge. Knowledge is structured, with the structure of an individual learner's knowledge being added to and modified as new knowledge is acquired. The network structure of hypermedia information resembles the structure of knowledge, facilitating the "mapping of a knowledge domain onto the cognitive structures of learners" (Lanza, 1991, p.19). The implication for courseware developers is to include hypermedia-style structures where they facilitate learning.

Johnson and Grover (1993) call for learner-centered design inherent in cognitive models of learning, and claim that traditional CBI, designed according to behaviorist theories, ignores the active learner. Abramson (1993) emphasizes that courseware should support learning outcomes which are at higher levels of cognition including application, analysis, synthesis, and evaluation.

The use of a hypermedia model, learner interactivity, and other elements of cognitive learning theory such as constructivist activities, must be tempered with an awareness of potential problems. For example, despite the wealth of research which demonstrates the benefits of hypermedia-based instruction, there are many significant problems also reported in the literature. Chief among these are the problems related to navigating through the network of hypermedia links and nodes. Studies show that with the freedom to select their own path, users can easily develop the feeling of being "lost", of not knowing where they are in the network of information (Roselli, 1991; Staninger, 1994; Tolhusrt, 1992).

It is not enough to "jump on the cognitive bandwagon" in developing interactive, multimedia courseware. Developers must be aware of potential problem areas and intentionally incorporate features which promote learning while avoiding techniques which retard learning.

Facilitating Development by Combining Authoring and Programming

Development of interactive, multimedia courseware requires a set of appropriate tools tailored to the type of applications being developed and to the individual skills and work style of the developer (Luther, 1994). Different authoring tools are based on different metaphors such as a slideshow, book, timeline, windows, or icons (Luther, 1994). The underlying metaphor of the authoring tool affects the format and style of the courseware product. Not all metaphors are appropriate for interactive, higher-level courseware. For example, the slide-show metaphor is basically a page-turning model, suitable for accompanying a live presentation or for a stand-alone information kiosk, but not for interactive learning at the higher levels of cognition.

Lynch (1995) reports on a number of entry-level multimedia authoring tools which can be used to produce professional quality courseware without the need for scripting, the authoring term for programming. Nevertheless, he notes that, "All five are based on the familiar slide-show metaphor now used in most presentation software" (p.10). These, then, may not be useful in designing interactive, higher-level courseware.

The full gamut of possible authoring tools includes traditional programming languages, authoring languages and authoring systems (Maddux, 1992). Traditional programming languages such as BASIC, Pascal, and others have the advantage that "every aspect of the creative process is under the direct and complete control of the author" (Maddux, 1992, p.8). That is, the developer is not restricted to a particular metaphor, particular types of media units, or any other restriction which the creators of an authoring tool may have built in to their product. The disadvantage to using a traditional programming language derives from the same point, not only are the developers not restricted, they are completely on their own, with no help being provided by the development tool.

Older authoring languages such as PILOT or Hypercard are similar to traditional programming languages except that they are oriented specifically towards courseware development (Maddux, 1992). While these languages aid in courseware development, they still require an aptitude for programming.

Authoring systems tend to be more structured than authoring languages, providing a framework for instructional material, with the developer filling in the content. Tools are provided to create screens; specify interactivity, for example, buttons; and incorporate content in various forms such as text, images, and animations. The biggest disadvantage to these easy-to-use systems is their lack of flexibility (Maddux, 1992).

familiar slide-show metaphor now used in most presentation software" (p.10). These, then, may not be useful in designing interactive, higher-level courseware.

The full gamut of possible authoring tools includes traditional programming languages, authoring languages and authoring systems (Maddux, 1992). Traditional programming languages such as BASIC, Pascal, and others have the advantage that "every aspect of the creative process is under the direct and complete control of the author" (Maddux, 1992, p.8). That is, the developer is not restricted to a particular metaphor, particular types of media units, or any other restriction which the creators of an authoring tool may have built in to their product. The disadvantage to using a traditional programming language derives from the same point, not only are the developers not restricted, they are completely on their own, with no help being provided by the development tool.

Older authoring languages such as PILOT or Hypercard are similar to traditional programming languages except that they are oriented specifically towards courseware development (Maddux, 1992). While these languages aid in courseware development, they still require an aptitude for programming.

Authoring systems tend to be more structured than authoring languages, providing a framework for instructional material, with the developer filling in the content. Tools are provided to create screens; specify interactivity, for example, buttons; and incorporate content in various forms such as text, images, and animations. The biggest disadvantage to these easy-to-use systems is their lack of flexibility (Maddux, 1992).

One problem in developing effective courseware which incorporates instructional design principles and follows appropriate cognitive learning theories is that no one tool or type of tool lends itself to the task. A development process which utilizes both authoring systems and traditional programming languages can realize the benefits of both types of tools: the ease of creating screens and a basic framework afforded by the authoring system and the flexibility and power afforded by the traditional programming language.

Effectiveness of Interactive, Animated Courseware

As work in the areas of the natural sciences, engineering and mathematics has shown, the use of animation and interactive displays can enhance learning (Aukstakalnis & Mott, 1996; Kaplan, 1992). Park (1994) discusses the theoretical basis, strategic applications, and instructional conditions for use of dynamic visual displays.

The concepts discussed by Park (1994) which are relevant to the visualization of abstract data structures and the operation of associated algorithms include (1) an illustration aid, (2) the facilitator of mental model formation, and (3) a visual anchor for understanding abstract concepts and processes. As an illustration aid, "in a complex system, motion can be used to highlight critical features and their relation(s) to other components that might not be easily grasped or attended" (p.23). Relative to formation of a mental model, "The graphical representation of invisible system functions or behaviors is particularly important to help the student form a mental model" (p.23). Finally, computer-controlled animation allows abstract concepts to be observed in a concrete way.

In addition to the direction provided above in the use of animation, Shyu and Brown (1993) have found that interactive visualizations improve learning and learning attitudes more than the use of static diagrams in non-procedural learning tasks.

Development Schedule

Development of a courseware package which covers the content described above, meets the specified learning objectives, and is facilitated by the use of appropriate development tools required approximately six months, with the developer working 15-20 hours per week. Table 4 presents an approximate schedule.

Table 4: Approximate Development Schedule

Month(s)	**Development Activity**
Oct. 1996	Requirements Analysis
Nov.-mid Dec. 1996	Specifications and Planning
mid-Dec. 1996-mid Jan. 1997	Design
mid Jan. 1997-Feb. 1997	Implementation
Mar. 1997	Formative Evaluation

Limitations and Delimitations

The goal of this development was to demonstrate a methodology for developing effective instructional software which meets specified measurable learning objectives. Nevertheless, the results are limited in certain respects.

The focus of the development was on subject matter which is abstract in nature. The effectiveness of the courseware does not necessarily imply that similar courseware applied to learning more concrete or procedural content will be effective in the same ways. The emphasis on simple animations to facilitate understanding of algorithm operation is applicable primarily to dynamic processes. Typically, these types of processes are difficult to learn using static aids.

While the overall structure of the courseware is not linear and does allow learner control to be exercised in terms of instructional sequence, activities and presentations within topical units is structured, and an optimum sequence of units is suggested to the learner. Thus, the courseware does not pretend to be an exploratory or reference-style package. A few hypermedia-style features are provided and used where appropriate, but generally within a structure intended to guide the student through appropriate activities which facilitate learning.

The algorithm animation feature of the courseware was developed specifically for algorithms associated with graphs and networks. The algorithms which are supported include depth-first traversal, breadth-first traversal, shortest path, minimum spanning tree, and topological ordering. While the animations are general in the sense of supporting any graph or network specified in the program or by a student, they do support other graph and network algorithms or algorithms for other abstract structures. Development of a general algorithm animator was not the goal of this project. Rather, the incorporation of an animation tool into an instructional software package developed according to software engineering and instructional design principles and in accordance with cognitive learning theories was the aim.

Evaluation of the courseware was limited to formative evaluation. Individuals and a small group of Computer Science students evaluated the usability and effectiveness of the courseware. Evaluation did not include experimental studies of the effect of the courseware on student achievement.

Summary

Computer Science students, particularly at the Freshman and Sophomore level, often find it difficult to grasp the abstract structures, operations, and concepts frequently encountered in the discipline of Computer Science. The design of computer software, like the design of a bridge or building, often begins with an abstract model of what the user needs. Working with abstractions allows the Computer Scientist to experiment with various implementations to find the most effective and efficient one. In addition, the data structures and algorithms commonly used in computer programs are themselves abstractions of their actual implementation in the underlying computer architecture.

To facilitate learning the abstract structures, operations, and concepts associated with graphs and networks, an interactive, animated courseware package was developed. Development was carried out in accordance with established software engineering and instructional design principles, and guided by the application of cognitive learning theories. To facilitate the development, an authoring system was used to implement the overall courseware structure as well as the primary user interface. Animations, interactive laboratory sessions, and self-tests were implemented using a traditional programming language. Compiled programs have been interfaced to the user screens to present an integrated application.

Current multimedia and hypermedia technology has been utilized where appropriate for enhancing student learning. The courseware is intended to be used in conjunction with normal lectures, homework assignments, and other learning activities.

structures and algorithms commonly used in computer programs are themselves abstractions of their actual implementation in the underlying computer architecture.

To facilitate learning the abstract structures, operations, and concepts associated with graphs and networks, an interactive, animated courseware package was developed. Development was carried out in accordance with established software engineering and instructional design principles, and guided by the application of cognitive learning theories. To facilitate the development, an authoring system was used to implement the overall courseware structure as well as the primary user interface. Animations, interactive laboratory sessions, and self-tests were implemented using a traditional programming language. Compiled programs have been interfaced to the user screens to present an integrated application.

Current multimedia and hypermedia technology has been utilized where appropriate for enhancing student learning. The courseware is intended to be used in conjunction with normal lectures, homework assignments, and other learning activities.

Chapter II

Review of Literature

Introduction

The primary goal of this development effort was to produce interactive, multimedia courseware which will allow Computer Science students to meet specific, measurable learning objectives related to graph and network data structures and algorithms. To insure success, the development has followed proven software engineering methodology, integrated instructional design techniques into the development process, and applied appropriate cognitive learning theories to the courseware design.

A knowledge of the benefits and pitfalls of employing multimedia and other computer-based technology, such as hypermedia and animation, in instructional software is crucial to the development of effective instructional software. In addition, application of existing guidelines in human-computer interface (HCI) design, as well as enhancements to these principles based on cognitive learning theories are essential, as is a knowledge of available development tools such as various authoring systems.

Graphs and Networks

Graph theory, a subfield of mathematics, had its origins in the solution of a mathematical problem popular in the early 1700's known as the Koenigsburg Bridge Problem. Leonhard Euler, a Swiss mathematician, heard of the problem while engaged in the court of Catherine the Great (Miller, Heeren, & Hornsby, 1990).

In Euler's day, the Pregal River flowed through the town of Koenigsburg, around the island Kneiphof. Just past the island, the river forked. Seven bridges connected the river banks to the island and each other as shown in Figure 1. The problem was this. Could a person, starting at any of the four land areas indicated as **a** through **d**, travel across each of the bridges once and only once, visiting each of the land areas and end his journey where he began? No one was able to solve the problem.

Euler represented the problem as a graph, a mathematical abstraction consisting of nodes and edges. The nodes represented the land areas and the edges represented the bridges, as shown in Figure 2. Using this abstraction, Euler was able to formulate rules which provided a solution to the problem (Miller, Heeren, & Hornsby, 1990).

Figure 1: Koenigsburg Bridge Problem

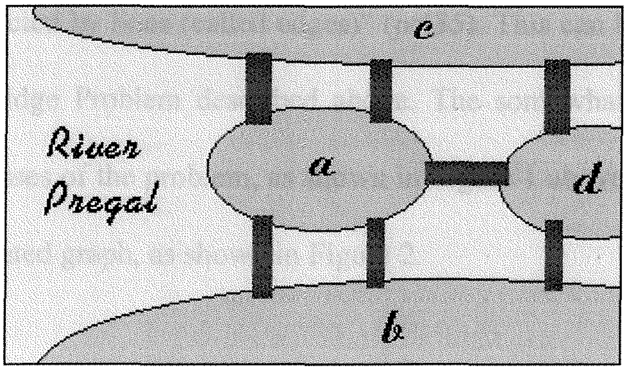
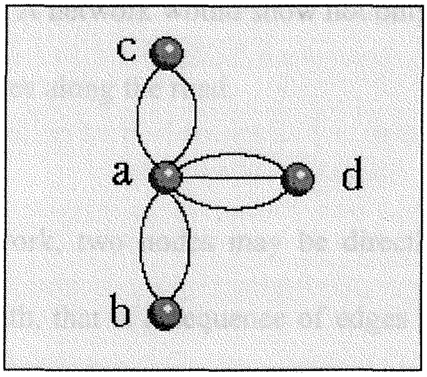


Figure 2: Graph Representing Koenigsburg Bridge problem



Concepts and Terminology

Graphs are abstract mathematical constructs which can be used to represent a wide variety of problems (Aho & Ullman, 1992; Naps & Porthering, 1992). In particular, graphs are useful in representing entities and relationships among entities. Thus, graphs can be applied to problems in communication, transportation, or interrelationships among people (Miller, Heerden, & Hornsby, 1990).

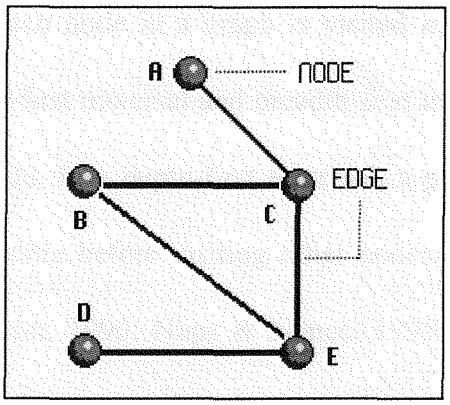
Aho and Ullman (1992) assert that a graph "has a powerful visualization as a set of points (called nodes) connected by lines (called edges)" (p.435). This can be seen in the case of the Koenigsburg Bridge Problem described above. The somewhat complex system of bridges and land masses of the problem, as shown in Figure 1 above, is represented quite simply by the associated graph, as shown in Figure 2.

Networks are an extension of graphs in which each edge has an associated weight (Naps & Pothering, 1992). For example, a graph could be used to show whether two cities were connected directly by a road. A network would show not only the connection, but also the distance between the two cities along the road.

In either a graph or a network, two nodes may be directly connected by an edge or indirectly connected by a path, that is a sequence of edges from the start node, through intermediate nodes, to a destination node (Aho & Ullman, 1992). Figure 3 shows a simple graph. In this graph there is a direct connection between nodes **A** and **C** since there is an edge connecting these two nodes. While there is not a direct connection between nodes **A** and **B**, there is a path from **A** to **C** and from **C** to **B**.

Graphs may be classified as cyclic or acyclic. In a cyclic graph at least one path is a cycle; that is, following the path from its source results in returning to the source. The graph is Figure 3 is a cyclic graph with the path connecting nodes **B**, **C**, and **E** constituting a cycle. A graph which has no cycles is called an acyclic graph (Aho & Ullman, 1992; Naps & Pothering, 1992).

Figure 3: A simple graph

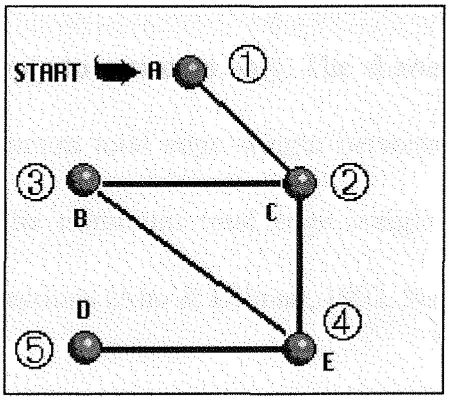


Graphs such as the one in Figure 3 are called undirected graphs since the edges which connect the nodes are assumed to be bi-directional. For example, if the graph in Figure 3 represented a system of roads connecting five cities, two-way traffic would be allowed on each of the roads. In certain problems, edges connecting nodes are unidirectional; that is travel between two nodes can occur in only one direction. The edges are then represented with arrows indicating the direction of travel and the graphs are called directed graphs, or digraphs (Aho & Ullman, 1992; Naps & Pothering, 1992).

The degree of a node in a graph or network is defined as the number of edges connected to the node. In the graph of Figure 3, node **A** has a degree of one since there is only one edge connected to it, while node **C** has a degree of three. For a directed graph the number of incoming edges constitutes the indegree of a node, while the number of outgoing edges constitutes the outdegree of a node (Naps & Pothering, 1992).

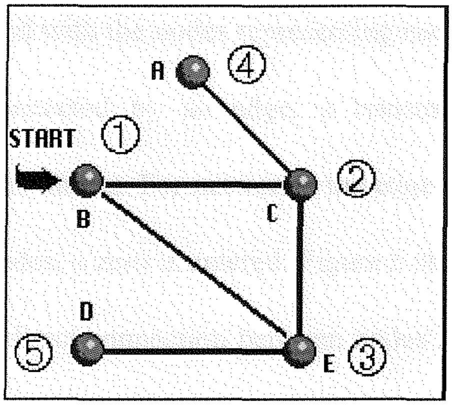
A common operation associated with a graph or network is that of a traversal. A traversal is an operation in which each node in a graph is visited in some order. Two common traversal methods are depth-first traversal and breadth-first traversal. Both methods begin at a designated starting node. In a depth-first traversal, a path is followed as far away from the start node as possible before visiting other nodes connected to the start node (Cormen, Leiserson & Rivest, 1990; Naps & Nance, 1995; Naps & Pothering, 1992). Figure 4 illustrates a depth-first traversal of the graph from Figure 3. Starting at node **A**, the traversal proceeds to node **C**, then to node **B**, then to node **E**, and finally to node **D**. In this example, the entire graph is traversed in a single path starting at node **A**.

Figure 4: Depth-first Traversal



In a breadth-first traversal each node directly connected to a given node is visited before traveling along a path further from the node (Naps & Pothering, 1992). Figure 5 shows a breadth-first traversal. Starting at node **B**, node **C** is visited, then node **E**, both directly connected to node **B**. Since there are now no other nodes directly connected to node **B** the traversal proceeds to follow the first path, that to node **C**, arriving at node **A**. Finally, the path to node **D** is taken completing the traversal.

Figure 5: Breadth-first Traversal

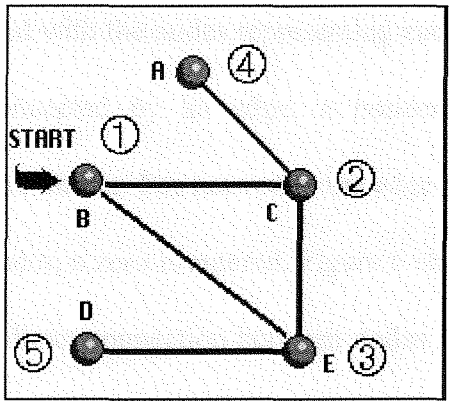


Several advanced operations are typically presented in undergraduate study of graphs and networks. These include shortest path determination, minimum spanning tree and topological ordering. All apply to networks only. The shortest path algorithm is used to find the path with the minimum total edge weight between two nodes. The minimum spanning tree determines the minimum total edge weight to connect all nodes, not necessarily with direct connections (Aho & Ullman, 1992; Naps & Pothering, 1992).

In certain applications, such as project scheduling, a network may be used to represent precedence relationships, such as which tasks must be completed before other tasks. A topological ordering of the nodes of the graph will indicate the optimum order for completing the tasks (Aho & Ullman, 1992; Naps & Pothering, 1992).

traversal proceeds to follow the first path, that to node **C**, arriving at node **A**. Finally, the path to node **D** is taken completing the traversal.

Figure 5: Breadth-first Traversal



Several advanced operations are typically presented in undergraduate study of graphs and networks. These include shortest path determination, minimum spanning tree and topological ordering. All apply to networks only. The shortest path algorithm is used to find the path with the minimum total edge weight between two nodes. The minimum spanning tree determines the minimum total edge weight to connect all nodes, not necessarily with direct connections (Aho & Ullman, 1992; Naps & Pothering, 1992).

In certain applications, such as project scheduling, a network may be used to represent precedence relationships, such as which tasks must be completed before other tasks. A topological ordering of the nodes of the graph will indicate the optimum order for completing the tasks (Aho & Ullman, 1992; Naps & Pothering, 1992).

Implementation

A common technique for representing a graph is the adjacency matrix (Aho & Ullman, 1992; Cormen, Leiserson & Rivest, 1990; Naps & Pothering, 1992). A square, two-dimensional matrix is formed with the nodes representing each row and column. For each pair of nodes directly connected by an edge, a nonzero value is entered in the corresponding element of the matrix. For networks, the edge weight is entered. If there is not an edge between two nodes, a zero is entered. Figure 6 show the adjacency matrix for the graph of Figure 3. The direct connection between nodes **A** and **C** is presented by a 1 in row **A**, column **C**. Similarly, the lack of an edge between nodes **A** and **B** is indicated by a 0 in row **A**, column **B**.

Figure 6: Adjacency Matrix for Graph in Figures 3

$$\begin{array}{c|ccccc}
 & \mathbf{A} & \mathbf{B} & \mathbf{C} & \mathbf{D} & \mathbf{E} \\
\hline
\mathbf{A} & 0 & 0 & 1 & 0 & 0 \\
\mathbf{B} & 0 & 0 & 1 & 0 & 1 \\
\mathbf{C} & 1 & 1 & 0 & 0 & 1 \\
\mathbf{D} & 0 & 0 & 0 & 0 & 1 \\
\mathbf{E} & 0 & 1 & 1 & 1 & 0
\end{array}$$

Learning Theory

Learning, whether by an individual for a personal reason, or by a group for a common goal, always has some objective. Even if learning is engaged in for its own sake, there is a goal, namely to learn. Given that learning is goal-driven, educators have, throughout time, sought to understand the purpose and process of learning to improve the degree of learning, reduce the time needed to learn, and increase retention. Thus, learning theories abound.

If instructional methodology and the use of technology in instruction are to be effective, they must take into account current learning theories. Educators must also be receptive to new theories and be sure to adopt new methodologies which demonstrate measurable benefit.

Historical Development of Learning Theory

The earliest learning theories date back to Pre-Christian times. Aristotle, a student of Plato, believed that education was necessary for the development of good individuals and a good society. In his writing, *Politics,* Aristotle asserts that "virtue and goodness in the state are not a matter of chance but the result of knowledge and purpose" (Aristotle, 1968, p.3).

John Dewey, an educator in the late 1800's, had a major impact on learning theories of that time. Dewey's *My Pedagogic Creed* was based on his belief that the success of democracy depended on education, both by the family and by the school (Sullivan, 1971). Dewey (1968), believed that "all education proceeds by participation of the individual in the social consciousness of the race" (p.51). He believed that learning was a social experience, that there should not be a set curriculum, and that the ultimate goal of education was the improvement of society.

Similarly, Thorndike (1971), writing in the early 1900's, believed that education was needed to change people. He says, "The need for education arises from the fact that what is is not what ought to be" (p.12). Thorndike also believed that the primary goal of education was to teach societal and behavioral values to individuals, in particular, "Education should make human beings wish each other well, should increase the sum of human energy and happiness and decrease the sum of discomfort of the human beings that are or will be, and should foster the higher, impersonal pleasures" (p.13).

While there were differences among these early learning theories, there was one common thread, an idea made famous by the American psychologist B.F. Skinner, that of behaviorism. Aristotle, Dewey, and Thorndike all believed that the result of education was a behavioral change in the individual. Skinner specialized in an instructional methodology in which learning was seen as producing an appropriate response to a given stimulus and reinforced by rewarding proper behavioral responses. Skinner (1971) describes this process by saying, "Once we have arranged the particular type of consequence called a reinforcement, our techniques permit us to shape up the behavior of an organism almost at will" (p.23).

democracy depended on education, both by the family and by the school (Sullivan, 1971). Dewey (1968), believed that "all education proceeds by participation of the individual in the social consciousness of the race" (p.51). He believed that learning was a social experience, that there should not be a set curriculum, and that the ultimate goal of education was the improvement of society.

Similarly, Thorndike (1971), writing in the early 1900's, believed that education was needed to change people. He says, "The need for education arises from the fact that what is is not what ought to be" (p.12). Thorndike also believed that the primary goal of education was to teach societal and behavioral values to individuals, in particular, "Education should make human beings wish each other well, should increase the sum of human energy and happiness and decrease the sum of discomfort of the human beings that are or will be, and should foster the higher, impersonal pleasures" (p.13).
While there were differences among these early learning theories, there was one common thread, an idea made famous by the American psychologist B.F. Skinner, that of behaviorism. Aristotle, Dewey, and Thorndike all believed that the result of education was a behavioral change in the individual. Skinner specialized in an instructional methodology in which learning was seen as producing an appropriate response to a given stimulus and reinforced by rewarding proper behavioral responses. Skinner (1971) describes this process by saying, "Once we have arranged the particular type of consequence called a reinforcement, our techniques permit us to shape up the behavior of an organism almost at will" (p.23).

While much of Skinner's experimental work was carried out with mice or pigeons, he proposed the adoption of this behavioral change methodology in educating children. Contending that the classroom teacher would not be able to adequately provide the necessary reinforcements, he suggested a teaching machine to do the task (Skinner, 1971). This suggestion represents an early integration of technology and educational theory and foreshadowed computerized programmed learning systems.

Gagne` (1992) has made significant contributions to the development of instructional design. His work, to a large degree, is based on Skinner's, being fundamentally behaviorist-oriented. In the most recent edition of his classic instructional design text he writes, "Changes in the behavior of human beings and in their capabilities for particular behaviors take place following their experience within certain identifiable situations. These situations stimulate the individual in such a way as to bring about the change in behavior" (p.6).

Modern Learning Theories

While still influential, behaviorist learning theories are being replaced by cognitive learning theories. There are several fundamental differences between the two types of theories. Behaviorism emphasizes the transfer from the teacher to a passive learner. The curriculum is rigid, and learners have little control over the learning process.

Cognitive learning theories are based on information processing models (McGilly, 1994) and emphasize the learner's active participation in the learning process. Several manifestations of cognitive theory are popular including collaborative learning, constructivism, and situated learning. Each of these models approaches the cognitive process according to a particular instructional methodology: collaboration with group work, constructivism with the learner's unique contribution to knowledge, and situated learning with environment-based learning experiences.

An important aspect of cognitive learning theories is that knowledge is viewed as an interconnected network of ideas, concepts and facts (Lanza, 1991; McGilly, 1994). A structure of interconnected knowledge has implications for learning and for the design of instructional systems. Gagne` (1971) indicates that at least some types of knowledge are hierarchically structured. He hypothesizes that the knowledge needed to perform a type of task may be subdivided into subcategories of more basic knowledge. This process is repeated on the subcategories until one arrives at knowledge which is considered to be fundamental. The result is a hierarchical arrangement of categories of knowledge.

A major influence in current learning theory is that of constructivism in which learners construct knowledge out of their own experiences (Hollis, 1991). Privateer and MacCrate (1992) emphasize the benefits of collaborative, constructivistic techniques in a college-level humanities course. Constructivism fits in well with the idea that knowledge is an interconnected network of information. McGilley (1994) describes the process of elaboration in which learning consists of adding new information to an individual's existing network of information.

Constructivism is often seen in contrast to objectivism. The objectivist view assumes that "the world is real, it is structured, and that structure can be modeled for the learner" (Jonassen, 1991, p.28). In contrast, constructivism "claims that reality is more in the mind of the knower, that the knower constructs a reality or at least interprets it based upon his/her experiences" (Jonassen, 1991, p.29).

Problem-based learning and situated learning are two applications of constructivism. Savery and Duffy (1995) describe a problem-based learning approach used by Barrows. Medical students, working in groups, are given a diagnostic problem with which they are unfamiliar. The students do appropriate research and propose an answer to the problem. The instructor evaluates the solution and, if it is not sufficient, requires the students to do additional research. Students learn by tackling a "real-life" problem and by working together to increase their knowledge. The authors state, "We cannot talk about what is learned separately from how it is learned" (p.31).

Instructional principles derived from constructivism are: (1) learning activities should be anchored to a larger task or problem, (2) learners should be supported in developing ownership for a problem, (3) tasks should be authentic, (4) tasks and activities should reflect the complexity of the actual situation in which the learner will eventually operate, (5) allow the learner to be responsible for the process for developing the solution, (6) the environment should support and challenge the learner, (7) encourage learners to consider alternative views, and (8) allow and encourage the learner to reflect on the solution and the process of arriving at the solution (Savery & Duffy, 1995).

While certain aspects of constructivism are appealing, there are also difficulties with this approach which must be taken into account when applying constructivistic theories. Jonassen (1991) believes that "evaluation of learning from constructivistic environments is perhaps the most difficult issue related to constructivism" (p.28). Evaluation of learning according to the traditional objectivist model is criterion-referenced. That is, specific, measurable learning objectives are developed for instruction, and learner achievement is measured by determining to what degree the goals have been met. Jonassen contends that evaluation must be more goal-free in constructivistic instruction. "If possible, it is the process of knowledge acquisition that should be evaluated, rather than a product" (Jonassen, 1991, p.30).

Not all educators would be willing to agree that the process of instruction is more important than the product. In addition to philosophical differences about the validity of goal-free evaluation, there are practical problems. Jonassen (1991) says "that using a single set of criteria for assessing the quality of outcomes is unacceptable" (p.31). He also contends that, "Constructivistic evaluation requires a panel of reviewers, each with a meaningful perspective from which to evaluate the outcomes" (p.31).

Another area of concern is that constructivistic learning often places high, perhaps unrealistically high, demands on the learner in the areas of cognitive load, metacognitive activities, and appropriate learner attitudes (Perkins, 1991). Constructivistic learning often focuses on context-rich environments in which learners can discover and construct knowledge. These environments are necessarily complex. The complexity places a high cognitive load on learners (Perkins, 1991).

In addition to cognitive complexity "typical constructivistic instruction asks learners to play more of the task management role than in conventional instruction" (Perkins, 1991, p.20). That is, learners are seen to be in control, or more in control, of their own learning which demands appropriate metacognitive skills in monitoring and evaluating learning. Not all learners are equipped for this increased demand and not all teachers provide the support needed for learners to succeed in managing their own learning (Perkins, 1991). Wilson (1995) says that "as the teacher relinquishes control over the content, pacing, and specific activities, students need corresponding increases in decision and performance support" (p.28).

Finally, while constructivist theory claims that learners are more responsible for their own learning and construct their own knowledge, not all learners see the learning process in this light. Often learners are used to "being taught" and assume all teachers will teach them what they need to know. Learners do not always "buy in" to the constructivistic approach (Perkins, 1991).

Philosophically, cognitive learning theories tend to be humanistic. Emphasis is on the self-actualization of the learner. Knowledge is constructed by learners, with learners defining reality from their own point of view. The process is experiential (Hollis, 1991). Traditionally instructional design has been largely based on behaviorist theories and has taken a scientific, problem-solving approach to learning. Hollis (1991) sees the possibility of bridging the gap between humanist theory-based learning theories and instructional design. He points out that, while its orientation differs dramatically from that of cognitive theories, instructional design is really a theory of instruction whereas humanistic educational approaches are theories of learning. Instructional design is primarily concerned with the application of learning theories and is free to combine elements of various theories.

An important aspect of constructivistic learning is that of situated cognition or learning environments. McLellan (1994) explains that the theory of situated cognition "is based upon the notion that knowledge is contextually situated and is fundamentally influenced by the activity, context, and culture in which it is used" (p.7). She emphasizes that learning should take place in an actual environment or a rich simulated environment which is as much like the actual environment as possible.

Wilson (1995) says, "A learning environment is a place where people can draw upon resources to make sense out of things and solve problems" (p.26). He emphasizes the need for appropriate information resources and tools within the learning environment, and that learners be given adequate support and guidance. Learning environments should be collaborative "communities of learners [who] work together on projects and learning agendas, supporting and learning from one another, as well as from the physical environment" (Wilson, 1995, p.27).

Technology and Multimedia

Overview of EducationalTechnology and Multimedia

Buford (1994) defines multimedia as "the simultaneous use of data in different forms (voice, video, text, animations, etc.)" (p.2). Galbreath (1992) insists that user interactivity via a computer must be included in any definition of multimedia. Multimedia applications exist in entertainment, home shopping, healthcare, science and engineering, and education (Buford, 1994). Buford believes that "multimedia documents, presentations, mail, games, and other applications will be the common denominator by which people and organizations communicate, work, and play together" (p.24).

Multimedia is one of the latest forms of technology to break upon the education scene. Education has embraced technology of all sorts over the years, with the goal of improving teaching and learning (Noblitt, 1995). Multimedia applications can provide individualized instruction, and provide information in forms other than text such as graphics, animations, and audio (Galbreath, 1994). There is a wide range of educational applications which can incorporate multimedia technology including tutorials, educational databases, simulations, and educational games (Sweeters, 1994).

Educational Technology Applications

Applications of educational technology vary widely in style and complexity. At one end of the spectrum is the computer-aided presentation. Primarily aimed at enhancing classroom presentations, this type of application is similar to a slide show or use of an overhead projector. Sammons (1995) describes a project at Wright State University in Dayton, Ohio in which selected faculty were provided with equipment and software for developing and delivering multimedia presentations. Faculty were trained on equipment use as well as on development techniques. The project focused on large, general education courses.

Student reaction to the use of multimedia for classroom presentations was highly favorable. Students felt that the computer-aided presentations "organized and supported the content", made presentations "more interesting and helped with understanding of material", and "helped them pay attention and clarified information" (Sammons, 1995, p.67). According to Sammons, the students "overwhelmingly supported continued use of the computer in the classroom" (p.69).

Applications which integrate course content and learning objectives, educational technology, and learner interactivity to provide a learning environment are at the other end of the spectrum from computer-aided presentations. Thurber, Macy and Pope (1991) developed interactive, multimedia instructional software for use in a college-level humanities course. The final product, NewBook, incorporated graphics, text, hypertext links, and several types of learner interactivity. The goal was to involve learners, in a constructivistic way, in the learning process. “Readers would, in the act of reading, create and eventually write another kind of small ‘book’ that would be a record of their reading and their own decision-making about what is or is not important” (Thurber, et al, 1991, p.58).

In the Odyssey Project, Privateer and MacCrate (1992) developed an interactive, computer-based application. Like NewBook, the Odyssey Project used graphics, text, and hypertext links to provide a rich information resource and overall course structure. Lecture notes were included as well as a “‘smart’ syllabus, a time management system” that insures students keep current in their work (Privateer & MacCrate, 1992, p.77). The application also incorporates a word processor to facilitate student note-taking and recording of thoughts and observation.

The Odyssey disk was used in conjunction with a cooperative learning technique which “encourages the use of higher-order, cognitive thinking strategies” (Privateer & MacCrate, 1992, p.78). Thus, the instructional software was integrated with cognitive strategies to encourage and support higher-order learning.

Computer-based simulations represent another type of educational technology application. “Computer-based simulations are efficient, effective, highly motivational, serve the need for individualization and enhance the transfer of learning” (DeNardo & Pyzdrowski, 1994, p.126). DeNardo and Pyzdrowski (1994) conducted a study of the effectiveness of using simulation to teach computer architecture at the college level. Results of the study showed that students felt “the simulators helped them to understand the concepts relating to the specific machine demonstrated in the simulators, helped them understand the concepts relating to computer architecture, and made learning about computer architectures more concrete” (p.134).

Podell, Kaminsky and Cusimano (1993) used computers and computer software as tools to enhance student motivation in ninth-grade physical science instruction. Rather than developing new software, the researchers implemented a science computer laboratory consisting of networked computers equipped with electronic probes. In addition, students were given access to spreadsheet templates, graphing programs, word processing programs and commercial instructional software for physical science. An experimental group “received instruction in physical science through computer-based and hands-on activities” (Podell, et al, 1993, p.68). Results of the study showed that the use of computer technology enhanced achievement and had a positive effect on attendance, subsequent enrollment in science courses, and toward computers.

Technology has not had a Wide-spread Impact on Education

Whether behaviorist or cognitive, learning theories encourage the use of technology to supplement and enhance instruction. Skinner (1971) suggested the use of a teaching machine to provide the proper reinforcements needed to ensure desired behavioral change in learners. Cognitive theories emphasize the information processing aspects of learning with the implication that applying information processing technology to instruction should prove beneficial. McGilley (1994) claims that applying the principles of cognitive science to education can have profound benefits.

Despite the almost universal support for the use of educational technology, in particular the use of computers in education, widespread benefits have not materialized. Wallis (1995) notes that "computers are indeed everywhere in American schools, but they are generally used as little more than electronic workbooks for drill, or as places for kids to play games during 'free choice' periods" (p.49).

The situation in higher education, particularly in American colleges and universities, is even worse. There are at least two major problems. First, there is not a significant amount of high-quality instructional software available for higher education. Second, computers are not used effectively for support of higher-order learning.

Ely (1993) observes that "very little software exists that is oriented to higher education alone" (p.53). Heterick (1994) agrees. He says, "Information technology is everywhere, but not enough of it is in our colleges and universities." Significant development of computer-based, interactive, instructional software for colleges and universities has not occurred as it has for K-12 education (Twigg, 1996). Twigg (1995) notes that "mainstream faculty (read, most people) want 'proven applications of compelling value' in order to change the way they go about their work, and these applications simply do not exist at the college level."

Where computers are being used, they are often not used effectively to enhance instruction. Surveys show that the majority of college faculty use computers to do word processing, spreadsheets or for other noninstructional uses (Ely, 1993; Heterick, 1994; Solomon, 1994).

In the Fall semester of 1996, the author assigned students in an introductory computer class at Mount Vernon Nazarene College the task of surveying college faculty and office personnel about computer usage. The survey instrument, the Technology Fact Find Questionnaire, is included in Appendix A. Computer use was high for faculty and staff. 91% of the 116 college personnel surveyed used their computer daily. Word processing was reported as an application for 97% of faculty and 90% of staff personnel. The primary educational applications for faculty were preparing materials (76%), presentations (55%), and electronic gradebooks (38%). Less than 25% of the faculty indicated using the computer for instructional applications such as tutorials, simulations, and drill-and-practice. These results, while restricted to a single, small college, are basically in agreement with the findings of others including Solomon (1994).

Computer usage in K-12 education is often ineffective as well. Caftori (1994) reports on research conducted at Old Orchard Junior High School in Skokie, Ill. Students were observed using educational software during free time after lunch. The results of the study indicated that students generally did not use the software as intended by the software designers, and that they frequently treated the programs like arcade games, missing the intended educational objectives. Caftori (1994) attributes the problems, at least partially, to incorrect use of the software. "Software documentation indicated that many programs were designed for a classroom environment with trained teacher guidance" (p.63). Other factors contributing to the ineffectiveness of the software were: (1) insufficient time, students were allowed only 20 minutes; and (2) distraction caused by some of the attractive features of the software .

Solomon (1994) attributes the lack of significant computer use in higher education to a number of factors including the prevailing culture in colleges and universities, financial costs, lack of standards, and the demand for multiple skills and significant time for faculty to develop their own software.

The prevailing culture in colleges and universities does not encourage innovation. In research institutions "there is not sufficient credit toward tenure and promotion for such activities"; while "in teaching institutions the teaching load is so high (normally 15 or 18 contact hours per semester) that there is insufficient time" (Solomon, 1994, p.82).

Computers, especially multimedia systems, are expensive and colleges are already under economic pressure. Costs have risen at a rate greater than twice the rate of inflation (Solomon, 1994) and the tax base needed to support education is eroding (Smith & Debenham, 1993). Multimedia instructional materials are seen as supplemental and, therefore, nonessential in the face of budgetary constraints (Solomon, 1994).

Development of high-quality instructional software for higher education is impeded by the general lack of technical standards for software, especially for various media such as graphics, audio, and video (Solomon, 1994). One solution is for faculty to develop their own software on their own equipment for their own classes. While there are instances of successful developments of this kind, for many, development is not an option. Solomon (1994) claims that the task is "too difficult for 98% of all faculty" and that "most people who begin the process of developing multimedia courseware give up soon into the process" (p.83). Multiple skills are needed to develop multimedia software (Erlich & Reynolds, 1992; Solomon, 1994) and development requires vast amounts of time (Solomon, 1994).

Using Technology Effectively

Technology, including the computer, has not had a wide-spread impact on education, either in K-12 education or at the college level. Nevertheless, there have been successes and there are things which can be done to promote the effective use of technology.

A good starting point is for educators to use existing, well-done instructional software and other technology effectively. The ineffective use of educational software at Old Orchard Junior High described by Caftori (1994) can be corrected relatively easily. A teacher, familiar with the operation of the software, the intended educational objectives, and the proper use of the package could provide a training session at the time the software is purchased. Students should use the software in the intended setting, such as the classroom with guidance from a teacher. Students should use the software for an appropriate amount of time, long enough to allow sufficient exploration, experience, problem-solving and synthesis of facts and concepts to occur (Caftori, 1994).

Software designers must avoid the use of attractive graphics, animations, audio, and other multimedia features if they tend to distract students from the primary task of learning. Software should "adopt the pedagogical methods that teachers use in a hands-on environment or for manipulatives: Remind students of what the goal is and point out inconsistencies in students' actions" (Caftori, 1994, p.65). Designers must design features into the software which provide immediate feedback and individual guidance in much the same way as a teacher would.

Littauer (1994) discusses considerations for the effective use of instructional software. Two vital points are that software is often incomplete and the teacher's role is different when using instructional software. First, teachers must realize that in many instances the "usual considerations in lesson planning - questions of content, organization and evaluation - are not addressed as clearly and respectively in instructional software" as they are in textbooks and other traditional materials. (p.53). The implication is that the teacher must be present when students are using software and may have to provide additional guidance or elaboration.

Second, the teacher's role in the classroom is often different when students are using instructional software. Students learn as information is presented, as they encounter environments and experiences provided by the software, and possibly as they contribute to the information being presented by notetaking, journaling or in other ways. “Where the teacher was a lecturer, he or she is now a facilitator” (Littauer, 1994, p.53). To be effective in this new role the teacher must become thoroughly familiar with the software before incorporating it into lesson plans.

The project at Wright State University to encourage faculty to incorporate technology into classroom instruction focused on the use of computer-aided presentations in large lecture classes (Sammons, 1995). While overall reactions of faculty and students were positive, several ideas for the proper use of technology in lecture situations emerged. Screens should not be overcrowded with information, lettering should be large enough to read at a distance, sufficient color contrast must be used, and screen content should generally be an outline, providing the main points and highlights.

If presentations simply incorporate text or a few images, the use of a computer is unnecessary. The power of the machine is not being utilized. Students at Wright State indicated that, “In addition to text, they would like to have sound, pictures, maps, diagrams, animation and humor. Many feel that the use of the computer as a glorified overhead projector is a waste” (Sammons, 1995, p.68). Students were also concerned that classes with high technology use not become depersonalized, that faculty be knowledgeable in the use of equipment, and that presentations should not rely on flashy features, but should provide sufficient content.

A major problem discussed in the preceding section was that of the lack of significant instructional software at the college and university level. Twigg (1995) makes several suggestions aimed at rectifying this situation. Her emphasis is on increasing the amount of instructional software which is available for higher education. Colleges should join together to write specifications for needed software and request software developers to bid on development. By collaborating, institutions would demonstrate to developers that there is an adequate market for college-level instructional software and provide a guarantee that developed software would be purchased and used. "Courses or subject areas enrolling a large number of post-secondary students or serving as a pre-requisite for further study in many areas should be the focus of our effort." The idea is to develop instructional software where it can be widely used and have the greatest institutional impact.

Twigg (1996) also suggests that developers look at the final market for instructional software not as the institution, but as the student. She argues that instructional software is like other learning materials, such as textbooks, or other learning tools, such as calculators. Typically these are purchased directly by students. Institutions could work with vendors to provide software to students at reasonable prices.

The effective use of educational technology is a multifaceted problem. Not only must appropriate instructional software and other technology be made available, but faculty and administration must be encouraged to use it and given the opportunity to see the benefits which it can provide.

Faculty in colleges and universities might be more willing to use instructional software if they knew that they could receive needed help and support. Willis (1993) notes that "Technology-assisted innovations are particularly sensitive to inadequate support" (p.26). He suggests that software vendors provide "school-based consultants who spend a significant amount of their time in schools working with teachers" (Willis, 1993, p.26). Other possibilities for support suggested by Willis include an "electronic hotline and technical support as well as normal phone and E-mail contact" (p.26).

Other factors which are believed to promote faculty use of educational technology include adequate training; peer interaction and support; time to experiment with the technology; voluntary participation as opposed to imposed use; and administrative cooperation and support.

Perhaps the most important issue related to the effective use of technology in education is to be sure that the technology is used to promote higher-order learning. Abramson (1993) emphasizes "computer-based instructional resources that promote application, analysis, synthesis, and evaluation" (p.6). She also comments that, "Traditional applications such as drill-and-practice programs, and those in which press-the-mouse replaces turn-the-page, do not use the power and waste the glory of hypermedia" (p.6).

Stoddart and Niederhauser (1993) contend that the mere infusion of technology into education will not change instruction. Instead, the important factor is how technology is used. They believe that technology should incorporate and support cognitive learning theories if it is to be effective.

Computers can be used to mediate the interaction of learners with natural or social phenoma (Dede, 1995). Software simulates environments which learners explore and manipulate, allowing them to discover facts and principles and to construct new knowledge. Dede (1995) thinks that this use of computer technology can be extended to the point that "learners can immerse themselves in distributed, synthetic environments, becoming 'avatars' who vicariously collaborate and learn-by-doing, using virtual artifacts to construct knowledge" (p.46).

The COMPUTER TUTOR software described by Smith and Debenham (1993) is intended to provide increased effectiveness and accessibility of instruction. Their approach was to develop "interactive computer software that would simulate a personal tutor and help students master course materials at their own pace" (p.71). A primary design criterion was that the software would incorporate cognitive learning theory. Students who took classes in which COMPUTER TUTOR was used believed that "the software improves their learning", and that "using the computer to learn key terms and concepts is a more efficient way of learning than reading the text alone" (p.73).

The major factors responsible for the lack of a wide-spread impact of computer technology on education can be addressed. Developers can be encouraged to develop software for colleges and universities acting in collaboration. Adequate training and support, as well as the support of administration, can promote faculty usage of educational technology. Most importantly, technology, particularly computer-based technology, must be used in ways that capitalize on the strengths and capabilities of the technology and incorporate cognitive learning theories.

Hypermedia and Interactivity

Hypermedia in its various forms is becoming widely used in a variety of learning situations. From encyclopedic databases to on-line help to computer-based instruction programs, hypermedia is one of the major new technologies in education as well as other applications (Ambrose, 1991; Cortinovis, 1992; Lanza, 1991; Myers & Burton, 1994; Park, 1991).

The term hypermedia refers to a system of presenting text, graphics, sound, animation or video-based information in such a way that the user of the medium may proceed in a nonlinear fashion (Maddux, 1992; Myers & Burton, 1994). For example, hypertext, the forerunner of the more general hypermedia, is characterized by the reader choosing the path through a document rather than proceeding strictly from beginning to end. Myers and Burton (1994) compare hypertext to "*Star Trek*'s transporter, allowing the user to 'beam' to any coordinates in a given hyperspace of information" (p.10).

Throughout the hypertext document certain keywords are highlighted. By selecting a keyword, the reader may branch to text which describes or explains that word or concept, or which is related to the concept in some way. Hypermedia simply extends the concept of hypertext to information in forms other than pure text, such as graphics, animation or sound (Myers & Burton, 1994).

The concept of hypermedia was first conceived by Vannevar Bush, science advisor to Franklin Roosevelt (Maddux, 1992; Myers & Burton, 1994). Bush envisioned a database of scientific literature capable of non-sequential access which would facilitate research. This system, called memex, was based on the principle of association. "The selection of any item could allow the immediate selection of another. By forming associations at will, the user could build a custom-made trail through the material" (Myers & Burton, 1994, p.11).

In the 1960's, Ted Nelson, a student at Harvard University, conceived of an electronic system which could represent the linkages he perceived between the units of knowledge he was studying in various courses (Paske, 1990). Nelson embarked on a project called Xanadu, with the goal of linking together a large body of human knowledge and providing access to this knowledge on-line and in real time (Myers & Burton, 1994). The hypermedia concept was popularized with the inclusion of Hypercard on Apple

Macintosh personal computers (Ambrose, 1991). "This software allows everyday users to create their own hypermedia products" (Myers & Burton, 1994, p.15).

While hypermedia seems to be a promising medium for educational applications, there are a number of problems associated with it (Park, 1991; Roselli, 1991; Wulfekuhle, 1994). Although hypertext and hypermedia have been the topic of research and many journal articles, important questions still remain unanswered about the effectiveness of hypermedia in various learning situations and for learners with different learning styles (Ambrose, 1991; Burwell, 1991; Jonassen & Grabinger, 1993; Moore, 1994; Myers & Burton, 1994).

Perceived Advantages of Hypermedia

Studies related to the use of hypertext, hypermedia and related technologies, such as interactive videodiscs, claim numerous advantages in instructional settings. Hypertext and hypermedia are effective in a variety of learning situations including delivery of traditional forms of CBI such as tutorials, drills and simple simulations; information retrieval systems; and in complex, constructivistic environments (Jonassen & Grabinger, 1993).

Hypermedia systems incorporate cognitive learning theories. Lanza (1991) says that "Both the representation of knowledge and its processing - inherent in the hypertextual approach to instruction - conform to the newer cognitive models of the learning/teaching process" (p.19). Other researchers argue that traditional CBI is machine-centered, rather than learner-centered because traditional CBI is based on B. F. Skinner's behaviorist theories rather than on newer cognitive theories (Johnson & Grover, 1993). "Hypertext is the most appropriate technology for supporting constructivistic learning environments because acquiring knowledge from hypertext requires the user to engage in constructivisitc learning processes" (Jonassen & Grabinger, 1993, p.26).

These ideas are supported by researchers who believe that hypermedia promotes nonlinear thinking (Wei, 1991), allows learners to access and organize information according to their own cognitive needs (Park, 1991), and that the interconnectedness of knowledge is supported by the structure of hypermedia (Ambrose, 1991; Staninger, 1994). Ambrose (1991) says that "the hypermedia format is likely to encourage thinking, speculation, and personal judgements on the part of the learner" (p.52). Learners are able to employ new strategies for learning, and to focus on the relationships among units of knowledge rather than simply accumulating isolated facts (Ambrose, 1991).

Hypermedia supports the concept of the active learner, allowing learners to pursue paths through information which meet their interests or are appropriate for their individual style of learning. Johnson and Grover (1993) describe a hypermedia model, the hypertutor system, which includes a "highly intuitive, default structure within an instructional environment having a high degree of learner control" (p.7). Learners who need more structure can follow the default path, while those who have less need of structure are free to navigate through the content more freely.

Cognitive Flexibility Theory holds that learners should recognize the interconnectedness of knowledge and develop new cognitive methods to organize complex information (Staninger, 1994). Some research tends to show that, consistent with Cognitive Flexibility Theory, the use of hypermedia actually does encourage the development of higher-level cognitive learning skills. Weiss (1994) contends that hypermedia systems "tend to encourage active processing on the part of students and support higher-order thinking" (p.31).

Cortinovis (1992) cites several current trends in training, all of which would seem to be consistent with the hypermedia approach. These are "reference-based training", the change from being a "teaching culture" to being a "learning culture", and the integration of classic behaviorist theories with cognitive theories. Cortinovis defines "reference-based training" as "getting the right information when and where needed" (p.47). This is certainly in line with hypermedia which allows the user to proceed nonlinearly through information as needed rather than enforcing a strictly sequential path through possibly irrelevant information. The idea of being a "learning culture" rather than a "teaching culture" embodies the idea that the learner is involved in and, at least partially, in control of, the process rather than the teacher. This, too, is consistent with the hypermedia approach as well as with cognitive learning theories.

A number of studies in specialized settings show that hypermedia is an effective instructional tool. In particular, hypertext has been used to improve humanities learning, the key being the flexibility of encountering the text and interacting with the text (Thurber, Macy & Pope, 1991). Privateer and MacCrate (1992), using their Odyssey project as the basis, claim that the use of hypertext is "an excellent way of presenting information that has applications across all disciplines" (p.78). Another area in which some researchers have found benefit in the use of hypermedia is that of cooperative learning. Maddux (1992) found that when hypermedia is used in conjunction with cooperative learning, the two approaches, both of which encourage and depend on higher-order cognitive thinking strategies, support one another.

In a different vein, hypermedia is judged to be an effective means of developing CBI lessons by teachers, rather than by professional developers (Maddux, 1992). Wulfekuhle (1994) concludes that CBI programs developed using hypermedia are effective because they are interactive and employ multiple media formats which stimulate and hold learners' interests.

Finally, a common benefit claimed for the hypermedia approach is that serendipitous learning may occur as learners navigate their own path through the hypermedia information (Staninger, 1994).

Problems Related to Hypermedia

Despite the wealth of research which demonstrates the benefits of hypermedia-based instruction, there are many significant problems also reported in the literature. Chief among these are problems related to navigating through the network of hypermedia links and nodes. Studies show that with the freedom to select their own path, users can easily develop the feeling of being lost, of not knowing where they are in the network of information (Roselli, 1991; Staninger, 1994; Tolhurst, 1992; Wei, 1991). Staninger (1994) observes, "For the novice user, the unfamiliar organization and retrieval structure of hypertext requires a period of adjustment and a new attitude toward information acquisition" (p.51).

The navigational problem is worse with larger hypermedia systems, which may be the most useful systems since a larger quantity of knowledge is linked together (Park, 1991). The worsening of the navigational problem with large systems may render such systems ineffective (Rivlen, Botafugo & Schneiderman, 1994).

In her text on the design of CBI systems, Steinberg (1991) notes, "Students often view each display as a separate entity and fail to perceive the continuity between displays. Consequently, the physical scheduling of related information must be planned with special care" (p.17). However, with a hypermedia system, the planned sequence intended by the designer need not be followed by the learner.

A number of researchers indicate concern over the possibility of learner distraction and cognitive overload with users of hypermedia systems. That is, due to the sheer number of choices and possible paths, the learner can experience too much information to process at one time (Roselli, 1991). Roselli also says that "cognitive overload gives rise to a further problem, which can be defined as 'conceptual disorientation', which occurs when the user loses sight of what really interests him" (p.42). Cognitive overload occurs because there is a limit to the amount of information that a person is able to process in a given amount of time (Steinberg, 1991).

The navigational problem and cognitive overload problem described above are both related to the issue of learner-control in instructional systems. Perhaps a more pertinent concern is that learner-control itself is not necessarily effective with all types of learners. Some students learn better with learner-control while others learn better with more structure and guidance (Burwell, 1991). Research has not shown learner-control to be effective because (1) learners do not know enough about what they are learning to make good decisions about sequence, and (2) metacognitive skills may not be adequate to allow learners to monitor and evaluate progress when dealing with a nonlinear presentation of information (Park, 1991).

Lanza (1991) contends that the hypermedia system designer must understand the cognitive learning styles and strategies of the intended learners to effectively set up the links of a hypermedia system. Learners with different cognitive styles may not be able to realize the same benefit from a given system as those for whom it was intended.

While the results described above indicate that learner-control may not be effective with all types of learners, there are also studies which show that learner-control may not be effective in certain content areas or learning situations. Knowledge in science tends to be hierarchical, one topic building on preceding topics. There must be a progression to the learning of the related material. Following other transverse paths; that is, those which cut across the hierarchy, tends to disrupt the natural order of the subject matter (Roselli, 1991).

One potential advantage of hypermedia claimed by researchers as described above was that hypermedia systems promote the development of higher-order cognitive skills. However, it may be that hypermedia requires higher-order cognitive skills for effective use (Wei, 1991; Roselli, 1991). In a hypermedia system the burden is on the learner to bring order to the seemingly chaotic collection of knowledge. If this is not done, there may be problems in comprehension (Staninger, 1994). Roselli (1991) says that "this kind of environment obliges the learner to make decisions continually and to assess constantly his state of progress, forcing him to apply higher-order intellectual powers" (p.42). A recent study shows that when readers are allowed to select the sequence of text, the sequences chosen are poor and incomplete (Rivlin, Botafugo & Schneiderman, 1994).

Considering all of the problems identified with the use of hypermedia, one may be tempted to question whether instruction is even an appropriate application of hypermedia. A number of researchers note that hypermedia was conceived of and developed primarily for reference applications rather than for instruction (Lanza, 1991). Sweeters (1994) relates various types of computer-based learning to Gagne`'s events of instruction. Hypermedia systems only fulfill two of the Gagne`'s nine required events of instruction. Crucial events such as providing learning guidance, performing feedback, and assessing performance are not inherently part of the hypermedia approach.

Finally, even claims of serendipitous learning are not wholeheartedly embraced by some experts. Sweeters (1994) says that, "although incidental learning may occur when one simply browses through a database, this is unlikely to meet major educational goals" (p.49). Similarly, Jonassen and Grabinger (1993) assert, "Browsing in a domain for which no properly developed schemata have yet been constructed, or no obvious need has been identified, is not likely to lead to satisfactory knowledge acquisition" (p.19).

Learning Abstract Content and Visualization

Computer-based technology may be used to facilitate and enhance learning in a variety of ways as described in the preceding sections. Nevertheless, the key element in the effective use of technology in instruction is that of appropriateness; that is, technology must be applied in ways which capitalize on its strengths, which enhance teaching or learning, not just for its own sake. Jonassen and Grabinger (1993) warn against using hypertext inappropriately. Boling (1994) discusses a number of issues relevant to the design of interactive, multimedia educational applications. Considerations such as screen size and resolution, use of audio and motion, and degree of user interactivity are issues which must be consciously dealt with (Boling, 1994). Park (1994) cautions that "animation should be incorporated only when the attributes are congruent to the learning task" (p.22).

Past Attempts to Facilitate Learning Abstract Structures, Operations and Concepts

Certain subject matter content is difficult to learn because it deals with concepts, structures or operations which are very complex, invisible to the naked eye, or are abstract representations of actual phenomena. This is frequently the case "in science and engineering disciplines where the subjects and concepts studied are often complex, multidimensional processes" (Aukstakalnis & Mott, 1996, p.14). Animations can be used to facilitate understanding in situations such as these. "The graphical representation of ideas can provide advantages over written formulas and theories. For instance, animated presentations allow students to actually see a physical law in action" (Armstrong & Loane, 1994, p.20).

Traditionally, authors of computer science texts have used a series of static diagrams to illustrate the effect of a particular algorithm acting on a data structure. In the introduction to their text, Naps and Nance (1995) note that an important feature of the text is that "algorithms are pictorially traced in a way that will bring them to life in the students' minds" (p.xxiii).

Current Multimedia Applications for Facilitating Learning Abstractions

Shyu and Brown (1993) cite studies in which the use of dynamic presentations was more effective than the use of static diagrams. While their own study of the effectiveness of dynamic presentations in the learning of a procedural task did not show a significant improvement of achievement, the results did indicate improved learning attitudes, as well as "supporting a pattern of higher task performance and shorter time for instruction" (Shyu & Brown, 1993, p.79).

Park (1994) claims a number of important benefits from the appropriate use of animation. In particular, graphical animations have been effectively used to "explicitly represent highly abstract and dynamic concepts in science", "to simulate functional behaviors of mechanical or electronic systems", and "to explicitly represent invisible flow of information" (Park, 1994, p.22).

These concepts have been put into practice, and the benefits of interactive, animated presentations have been demonstrated, by several educators. The use of multimedia courseware to provide visualization and animation was undertaken in the subject areas of chemistry, physics, and math at the University of Massachusetts at Lowell. Using animation tools, such as Authorware, animations were created to illustrate chemical processes, the effects of physical laws, and mathematical concepts (Kaplan, 1992). Kashef (1991), using computer-aided design (CAD) tools to create simulations, found that the "simulations enhance motivation, the transfer of learning and efficiency" (p.64).

Algorithm Animators in Computer Science

The effectiveness of animations in computer science may be inferred from the growing popularity of algorithm animation systems which are available. "It is assumed that such systems help students learn algorithms better than they could otherwise" (Wilson, Aiken

& Katz, 1996, p.75). Visual animation systems in common use include Balsa, Xtango, GAIGS, and FLAIR.

Projecting algorithm animations during a lecture can be an effective way of presenting the important features of algorithm operation. The speed of the animation can be controlled, the animation can be paused, and the effect of using different data sets can be easily observed by students (Rodger, 1996).

Studies indicate that algorithm animation systems are most effective when used to supplement and support lectures and other traditional instructional methods, rather than as stand-alone applications. "Learning is facilitated when animations are included as a part of an instructional context which also includes textual components and which features carefully focused task assignments designed to achieve specific learning objectives" (Wilson, Aiken & Katz, 1996).

Instructional Design and Multimedia

To be effective, instructional materials, including computer-based materials, must be designed according to proven principles and techniques of instructional design. In addition, traditional instructional design methods must be updated to incorporate considerations for new technologies and new learning theories (Erlich & Reynolds, 1992).

Instructional Design

Instructional design is a methodology which employs the systematic, purposeful organization of instructional experiences to promote learning. The emphasis is on the individual learner rather than on large groups, and the primary guiding assumption is that learning occurs best in systematic, structured environments (Gagne`, Briggs & Wager, 1992).

Five basic assumptions guide the process of instructional design. First, instructional design is intended primarily to aid the individual learner. Second, instructional design incorporates both short-term and long-term design phases. Long-term design focuses on the overall structure and goal of a curriculum or set of lessons while short-term design pertains to the immediate preparation of individual lessons. Third, instruction is most effective if it is systematically designed. "Unplanned and undirected learning, we believe, is very likely to lead to the development of many individuals who are in one way or another incompetent to derive personal satisfaction from living in society" (Gagne`, Briggs & Wager, 1992, p.5).

Fourth, the design and development of instructional materials must also be systematic, following systems engineering methodology. Finally, instruction must be designed in ways which incorporate current learning theories (Gagne`, Briggs & Wager, 1992).

Gagne, Briggs and Wager (1992) describe the internal processes of learning and relate these to external instructional events which they call the events of instruction. These include: (1) gaining learner attention, (2) stating objectives, (3) recalling prerequisites, (4) presenting information, (5) providing learning guidance, (6) eliciting expected performance, (7) providing feedback, (8) assessing performance, and (9) enhancing retention. Sweeters (1994) examines several CBI models including tutorials, hypertext and hypermedia systems, simulations, and educational games with respect to their incorporation of the events of instruction. Interestingly enough, tutorials, generally considered to be low-level instructional materials, incorporate more of the events of instruction than any other model. This seems to imply that appropriate tutorials ought to be a part of interactive, multimedia instructional software.

Instructional design incorporates analysis and design, development, and evaluation. In particular the process consists of the following activities: (1) defining instructional goals, (2) performing instructional analysis, (3) identifying learner characteristics, (4) specifying performance objectives, (5) determining criterion-referenced performance measures, (6) designing an instructional strategy, (7) collecting and developing instructional materials, (8) conducting formative evaluation, and (9) conducting summative evaluation (Gagne`, Briggs & Wager, 1992).

The process begins with defining the goals of the instruction which are derived from learner needs. This focus on needs and goals in at the heart of the design process (Erlich & Reynolds, 1992; Kozel, 1995).

Instructional analysis focuses on the "skills involved in reaching a goal" (Gagne`, Briggs & Wager, 1992, p.23). Task analysis can be used to determine the skills needed at each stage of carrying out a particular procedure, while information-processing analysis determines the mental operations which are involved. The goal is to understand the underlying requirements involved in a particular learning situation.

In addition to setting goals and analyzing instructional processes, the characteristics of individual learners must be determined. This includes identifying prior skills and knowledge, and necessary prerequisites. If this is not done, instruction may not be appropriate or effective for the intended learners (Gagne`, Briggs & Wager, 1992).

Since instruction is intended to meet an identified need or reach a specified goal, it is essential to know whether or not the objective has been accomplished. This requires the specification of measurable performance objectives. Objectives "should focus on learning outcomes and be stated in measurable terms" (Erlich & Reynolds, 1992, p.283). Specific, measurable performance objectives also facilitate planning and development of materials and instructional activities (Gagne`, Briggs & Wager, 1992). As indicated in the events of instruction, the performance objectives should be communicated to the learner.

Performance assessment is intended to determine whether and to what degree the learner has met the performance objectives (Erlich & Reynolds, 1992). In addition, performance measures can be used for learner placement. Performance measures should be criterion-referenced, that is they should be based on specific learning objectives (Gagne`, Briggs & Wager, 1992).

Designing an instructional strategy involves planning what resources and procedures learners will experience, the sequence of experiences, the degree and timing of guidance, and how specific activities contribute to accomplishing the performance objectives. At this point various learning theories can be accommodated, specifying teacher-led experiences for some activities, as well as learner-controlled activities for others. Associated with instructional strategy is the selection and development of appropriate materials to support and deliver instruction (Gagne`, Briggs & Wager, 1992; Luther, 1994).

Formative and summative evaluation are intended to determine the effectiveness of the instructional product. "Formative evaluation provides data for revising and improving instructional materials" (Gagne`, Briggs & Wager, 1992, p.30). Materials should be tested initially one-on-one, with a single learner and an evaluator working together. Next, materials are used by small groups of learners representative of the target audience, and finally, by an entire class in a field test (Gagne`, Briggs & Wager, 1992).

Once the materials have been revised and finalized, summative evaluation can be done. Summative evaluation is intended to determine the effectiveness of the instructional system as a whole. The instructional system is used in broad and varied circumstances representative of the target audience, and measures of effectiveness are recorded and analyzed (Gagne, Briggs & Wager, 1992).

Designing Interactive, Multimedia Instructional Systems

The principles and techniques of instructional design provide a good foundation for the design of interactive, multimedia instructional systems. However, there are significant differences between static, sequentially-oriented materials and the dynamic, nonlinear features of interactive multimedia and hypermedia systems (Park & Hannafin, 1993). Instructional design does not specifically address issues related to the use of technology in instruction.

Kozel (1995) describes a model for designing interactive multimedia courseware called the interactive experience model (IEM). "The IEM divides the design process into two parts: defining the goal/experience and constructing the interactive spiral" (Kozel, 1995, p.74). True to instructional design methodology, learning goals are identified first. As goals are identified, experiences are planned which will help the learner achieve the goals.

The key to the IEM proposed by Kozel is the second phase, constructing the interactive spiral. The spiral "has three components - interest, activity and resolution - that recur in succession until the user experience is complete" (Kozel, 1996, p.62).The idea is to introduce content to the learner gradually, gaining their attention, engaging in an appropriate activity, and pointing the way to the next stage. This process is repeated as the learner progresses through the content. Gagne``s events of instruction are evident as well as the instructional design model's concern for instructional strategy and systematic design. Kozel (1996) emphasizes that instructional software will be effective only if it is user-centered and learner interaction is meaningful.

The design of a hypertext or hypermedia system must take into account factors and considerations which are unique to these styles of document implementation. Sweeters (1994) says, "Almost any presentation can be made more interesting by using hypermedia" (p.49). Nevertheless, factors such as the document and page structure, learner control, navigational and other tools, presentation of material, and multimedia considerations must be intentionally considered.

Cates (1992) proposes fifteen principles to be used in the design of a hypermedia/multimedia product. These design principles relate to educational and user interface concerns as well as those which derive from the nature of the medium.

A hypermedia system is typically perceived as a document which consists of a number of related units of information of various types, presented in various media formats. Each unit of information is considered to be a node. Certain nodes are connected by links forming a tree-like structure or network of interconnected nodes. Hypermedia systems tend to be ill-structured and chaotic and thus can inhibit learning (Staninger, 1994).

To avoid confusion due to lack of structure, hypermedia systems should be designed to incorporate an overall document structure, careful design of individual nodes, and meaningful links between nodes (Staninger, 1994). Lanza (1991) maintains that "designers have to know 'how' students learn in order to set up links which favor flexible and intuitive navigation within the hypercourse environment" (p.20).

[illegible]

The degree of learner control with respect to sequence of presentation is related to the establishing of links in the hypermedia system. A balance must be maintained between the desire to accommodate different cognitive styles and the necessity to provide guidance and structure (Tolhurst, 1992). Johnson and Grover (1993) suggest the implementation of a default path through a document which learners can follow, yet still allowing deviation from the path as the user desires.

[illegible]

Nodes in a hypermedia network represent units of information or knowledge. The hypermedia network, itself, is representative of the network structure of knowledge assumed in cognitive learning theories (Ambrose, 1991; Lanza, 1991). To facilitate incremental development of knowledge and allow learner control to navigate the hypermedia network relatively freely each node should represent a single concept. While the learner has the freedom to navigate within the hypermedia network at will, the designer can and should guide the learning by including appropriate links among nodes (Lanza, 1991).

There should be a well-defined starting point, or root node, for a hypermedia document. The root node should provide general information and options for proceeding to other nodes. The user should be able to return directly to a root node from any other node in the hypermedia network (Lanza, 1991).

Because the user is not constrained to follow a predetermined path through a hypermedia document, navigational controls and aids must be provided. Navigational controls generally allow the user to move around in the document, following links between nodes. Such controls typically allow simple tasks like going forward or backward a single page in a document, or jumping to a table of contents page, an index page, or a glossary.

In addition to simple page-turning, navigational aids also provide learners with information about where they are within the document. Maps, bookmarks, and progress indicators are examples of additional navigational aids (Boling, 1994; Tolhurst, 1992).

A primary consideration in designing a hypermedia system is that of screen or page design. The typical computer display is not able to present the quantity of text or other material that can be presented on a printed page (Boling, 1994; Wei, 1991). Richards and others (1991) recommend the use of well-defined page structures and describe four such structures: simple page, tiled page, overlay page, and oversized page.

The use of multiple media formats in a hypermedia system can provide a rich learning environment. Nevertheless, the designer must be sure that media are used appropriately.

In particular, audio, video and animation formats, while dramatic, should only be used when the presentation is enhanced by their use (Boling, 1994).

Human-Computer Interface Design

Perhaps the most important consideration in the design of interactive, multimedia instructional software, as in any system, is the design of the user interface. Shneiderman (1992) says, "When an interactive system is well-designed, the interface almost disappears, enabling users to concentrate on their work, exploration or pleasure" (p.9).

The concept of user-friendliness is imprecise and must be succeeded by specific guidelines and principles for user interface design. Several important design considerations guidelines discussed by Shneideman (1992) are providing proper functionality, identification of user profiles, consistency, error detection and handling, avoiding cognitive overload, and minimizing user actions.

The user interface must provide capabilities for the user to perform all tasks necessary to the application. Providing proper functionality results from doing a task analysis as part of the user interface design. Tasks should be divided into categories according to frequency of use. Generally, tasks provided should relate to the application domain, minimizing the necessity for the user to master computer concepts to use an application (Shneiderman, 1992).

In most applications, users can be divided into three categories: novice, intermittent, and frequent. The needs of users in each of these groups is different. Novice users need instructions, frequent help and prompting. Intermittent users are typically knowledgeable about the system, but need reminders on specific details. Frequent users often wish to bypass lengthy sequences of operations, taking short-cuts to accomplish a task. The user interface design should make provision for all types of users.

Next to proper functionality, consistency is probably the most important consideration in user interface design. By providing consistent ways to do similar actions, communicating in consistent formats and locations, and responding in consistent ways to similar inputs, the user interface promotes confidence and a sense of control in the user. Users are not forced to figure out each new display or data entry when consistency is maintained (Shneiderman, 1992).

Operations should be rapid, incremental, and reversible, with the effects of operations being immediately visible (Shneiderman, 1992). The possibility of errors can be eliminated by presenting the user with valid alternatives. If errors occur, diagnostic messages should be clear and understandable, and give the user an explanation of how to remedy the condition.

Displays which are overcrowded, screens which require the user to remember information for action on subsequent screens, providing the user with too many options at one time all contribute to the problem of cognitive overload (Shneiderman, 1992; Roselli, 1991). The user interface design should attempt to minimize cognitive overload by simplifying displays and lists of alternatives, by retaining contextual information in the transition between screens, and generally by minimizing user actions where possible.

Authoring

There are a number of important considerations in developing, or authoring, instructional software. The process of authoring is of critical importance and should be based on established software engineering practices such as needs assessment, analysis, design, implementation, and evaluation (Mauldin, 1995). In addition, the authoring process must incorporate instructional design principles (Erlich & Reynolds, 1992) and cognitive learning theories (Park & Hannafin, 1993).

The selection and use of an appropriate authoring tool is key to the successful development of a hypermedia instructional unit. Mechanisms for including learner control, various types of presentation formats, and techniques for organizing and accessing the source data are important considerations (Wulfekuhle, 1994).

Learner control features must allow ease of use, and permit the learner to concentrate on educational tasks, not being distracted by complex or cumbersome controls. Under presentation considerations, Wulfekuhle (1994) includes the ability to add audio and visual elements, compatibility with common platforms, ability to use multimedia information with ease, and the capability to include user input such as test answers. Being able to handle a large amount of multimedia data and allowing flexible access to the data are source data organization concerns.

Creation of the software should be as simple as possible. Point-and-click interfaces for designing screens, including media units in nodes, and creating hyperlinks between nodes are significantly easier to use than mark-up languages, scripting or traditional programming. The authoring system should provide (or provide interfaces to) tools for creation and editing of media units of various types including text, graphic images, audio clips, and animations (Luther, 1993). Similarly, it should be possible to provide a link to external programs to allow the developer to write programs, if necessary, to supplement the features provided by the authoring system.

Chapter III

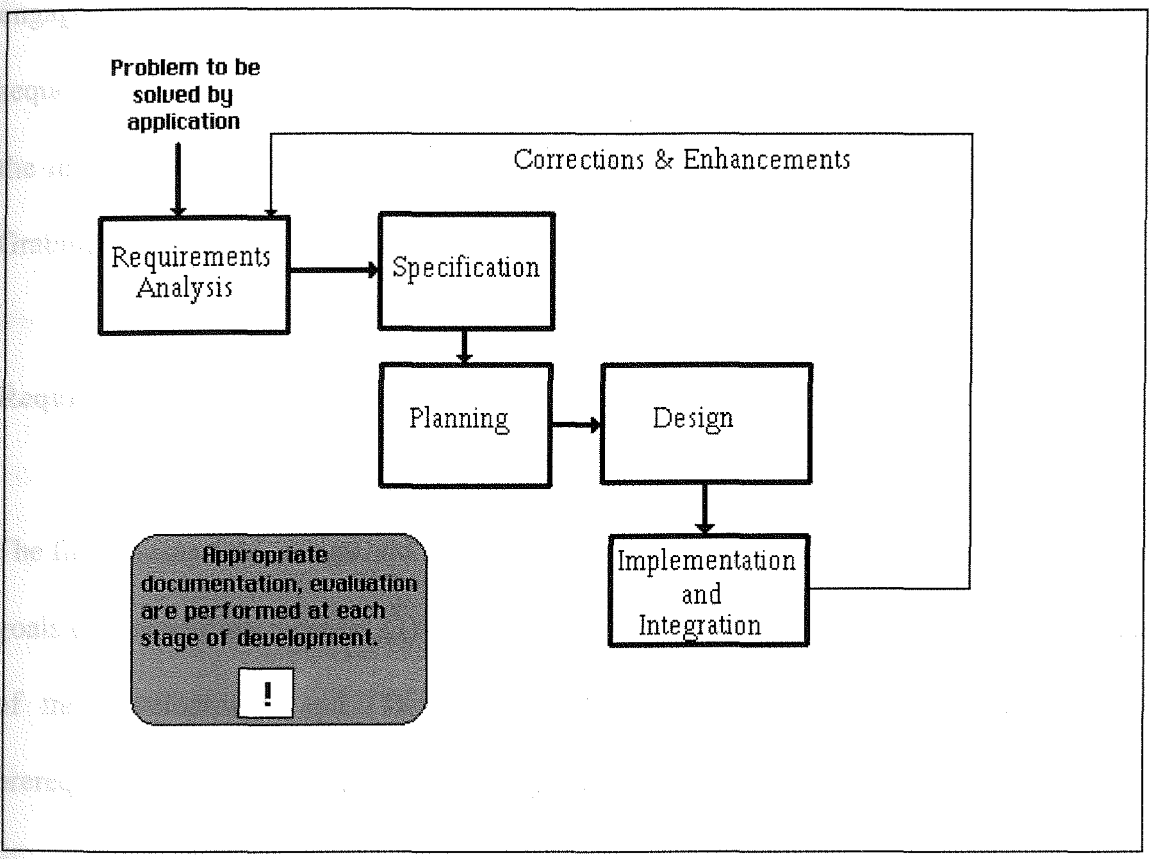
Methodology

Introduction

The primary goal of this dissertation has been the development and evaluation of interactive, multimedia software which incorporates animation to facilitate learning about abstract concepts, structures, and operations. The courseware was developed using proven methods of software engineering, incorporating instructional design principles. Cognitive learning theories guided the design to promote active, high-level learning.

The overall methodology was that of software engineering, a proven methodology for developing complex software systems. The major steps in the process included: (1) requirements analysis, (2) specification, (3) planning, (4) design, and (5) implementation and integration. Figure 7 illustrates the development process, emphasizing its iterative nature. Once the product has been completed, undiscovered errors or product enhancements cause the process to be repeated. The error or enhancement is analyzed, specifications are modified, plans are made to implement required changes, modifications are designed, and finally implemented. As noted in Figure 7 formative evaluation was conducted throughout the process to ensure correctness and usability of the resulting courseware (Schach, 1993).

Figure 7: Software Engineering Development Process


Problem to be solved by application
Corrections & Enhancements
Requirements Analysis
Specification
Planning
Design
Implementation and Integration
Appropriate documentation, evaluation are performed at each stage of development.
!

Instructional design considerations were incorporated into the software engineering methodology to tailor the general process to the design and development of educational software. Instructional design considerations include: (1) defining instructional goals, (2) identifying learner characteristics, (3) performing instructional analysis, (4) specifying performance objectives, (5) determining criterion-referenced performance measures, (6) designing an instructional strategy, (7) developing and collecting instructional materials, and (9) production of support materials (Gagne`, Briggs & Wager, 1992).

Cognitive learning theories suggest that learning occurs best when the learner is actively engaged; contributes to the learning process; has some control over the content, pace and sequence of instruction; and is presented with a network of information which resembles the interconnected structure of knowledge itself (Johnson & Grover, 1993; Jonassen & Grabinger, 1993; Staninger, 1994).

Requirements Analysis

The first phase of the design and development process was the requirements analysis. The goals of this activity were to: (1) define the problem to be addressed, (2) define the goals of the development, and (3) determine the characteristics of the users and any prerequisites for effective use of the final product.

In a conventional software development, much of the work during the requirements analysis phase in done jointly by the developer and a client. The client represents a group or organization which has a perceived problem. The client and the developer may belong to different organizations, to the same organization, or even be the same person. In this investigation, the developer was a Computer Science professor who has observed the difficulty Freshman and Sophomore Computer Science majors have in learning abstract structures, concepts, and operations.

Problem Definition

The procedures for problem definition in this development have been: (1) observation, (2) discussion with students, and (3) research into the experience of other Computer Science teachers. Problem definition was completed in identifying the dissertation topic. The problem has been observed over time in the difficulty students have understanding abstract structures, concepts, and operations presented in class and in implementing associated data structures and algorithms in homework assignments. Discussions with students confirm the difficulties as do discussions with colleagues teaching Computer Science courses. The problem is confirmed in writings by others both within Computer Science (Rodger, 1996; Wilson, Aiken, & Katz, 1996) and in other science and engineering disciplines (Aukstakalnis & Mott, 1996; Kaplan, 1992).

Development Goals

Having defined the problem, the next step was to specify development goals. In the case of instructional software, these goals relate to a number of areas including: (1) measurable learning objectives, (2) educational level, (3) delivery methodology, and (4) development methodology.

Instructional software development must have as its primary goal that of meeting specific, measurable learning objectives (Erlich & Reynolds, 1992; Kozel, 1995). Learning objectives are performance objectives, that is, they state learning outcomes in terms of activities the learner will perform to demonstrate that learning has occurred.

While the details of these learning objectives were determined in the next phase, specification, the need for the software to support attainment of specific learning objectives was acknowledged as a development goal.

In general, the content area to be covered may be oriented toward a particular age group or grade level or may be appropriate for various levels. In this development project, the developer determined for which age or grade levels the software is intended and specified prerequisite skills or knowledge which learners will need to use the software effectively. Traditional materials, including textbooks, workbooks, and audiovisuals were used to help determine appropriate level.

Consultation with practitioners and learners was used to determine the best delivery methodology. Literature review augmented personal experience and discussion with colleagues and students. Determination was made regarding the technology and the instructional features to be included in the courseware.

A final development goal related to the development itself. To be effective, instructional software must be developed according to established software engineering methodology, incorporate instructional design principles, and be guided by current learning theories (Erlich & Reynolds, 1992; Gagne`, Briggs & Wager, 1992; Stoddart & Neiderhauser, 1993). The methodology described in this chapter reflects this development goal.

Determining User Characteristics and Prerequisites

Instructional analysis was used to determine prerequisite skills and knowledge. Task analysis and information-processing analysis are two types of instructional analysis. Task analysis produces "a list of steps and the skills used at each step in a procedure" (Gagne`, Briggs & Wager, 1992, p.23). Information-processing analysis is used to determine the knowledge and conceptual understanding needed to learn new concepts.

Instructional analysis consisted of creating instructional curriculum maps (Alessi & Trollip, 1991) and noting skills and knowledge needed to attain each learning objective. To create the instructional curriculum map (ICM) a learning objective was decomposed into necessary skills and knowledge. Each requisite skill or unit of knowledge was then analyzed to determine its necessary skills and knowledge. The process was continued until a basic level of skills and knowledge was reached. This basic level consists of skills and knowledge which Sophomore Computer Science majors can be assumed to have.

User characteristics are summarized using a chart similar to the one shown in Figure 8 which is based on a similar chart in Alessi and Trollip (1991). The chart includes general student characteristics such as motivation, experience and interest, as well as specific attributes related to the area of learning abstract data structures and algorithms. Learner prerequisites determined by the instructional analysis replace the row labeled "Skills to implement data structures and algorithms."

Figure 8: User Characteristics for College Level Computer Science Students

		Level		Time to	Difficulty
	Low	Avg	High	Learn	to Learn
Year					
Experience					
Motivation					
Interest					
Computer Operation					
Courseware Familarity					
Skills to implement data structures & algorithms					

Specification

The purpose of the specification phase is to produce a specification document which describes the functional characteristics of the courseware product and associated documentation such as user's guides. The specification document is called by various names including the Requirements Specification, and the Functional Specification. The intent of the document is to clearly define what the end product is to be and what it is to do. The design at this stage is functional, that is, it concentrates on what functions are to be performed and describes them in ways that the end user can understand. The document is not a technical document in terms of implementation.

User interface designs were included in the specification document. The document was complete enough that the user could understand what the product would be, what it would look like and how it would operate. In a commercial software development, the specification document acts as a contract between the developer and the client. It defines what the client will receive and what the developer will deliver.

While the specification document clearly defines what the product is to be, it does not address how the implementation will be accomplished. The client is not generally concerned with this knowledge as long as the product does what it is supposed to do. Detailed design of the implementation is normally done in the design phase.

Learning Objectives

Specific, measurable learning objectives guide the overall design and development. Meeting the learning objectives is intended to solve the problem or meet the overall goal of the instructional experience. Gagne`, Briggs & Wager (1992) define a performance objective as "a precise statement of a capability that, if possessed by the learner, can be observed as a performance" (p.125). The emphasis of learning objectives is measurable, observable behavior (Kozel, 1995).

Abramson (1992) describes six categories of learning objectives: (1) knowledge objectives, (2) comprehension objectives, (3) application objectives, (4) analysis objectives, (5) synthesis objectives, and (6) evaluation objectives.

Knowledge objectives focus on the learner recalling or reciting learned information. Comprehension objectives are intended to demonstrate that the learner can use learned information. Applying abstract concepts, rules or procedures in particular situations is the focus of application objectives.

Analysis objectives center on distinguishing parts and breaking down ideas or models into constituent parts, while synthesis objectives involve building up new ideas or entities from basic elements. Finally, evaluation objectives demonstrate the learner's ability to make value judgments based on appropriate criteria.

Learning objectives have been stated using action verbs to describe desired learner performance. Objectives for the courseware include knowledge, comprehension, application, analysis, and synthesis objectives.

Overall Structure

An important part of the specification is defining the overall structure of the courseware. The overall structure defines the major parts of the courseware and their relationship to one another. Once the publication structure was specified, individual parts were specified in detail.

A multimedia courseware application typically consists of units linked together in some way: linearly, hierarchically, or network fashion. Each unit will consist of items which are unique to that unit such as graphic images, text and other media items. Typically there will also be some parts of each unit which are common to all or most of the units such as navigational controls.

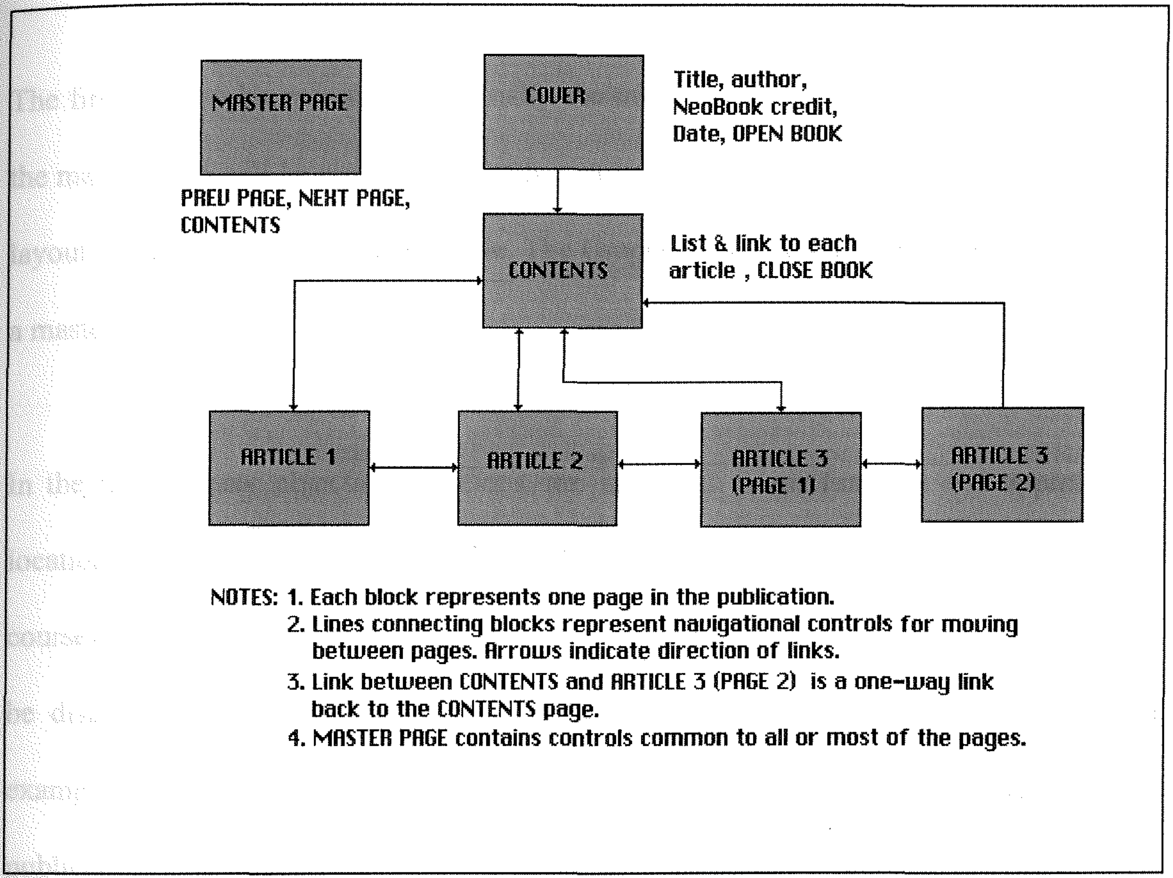
Documentation of the publication structure was done as a diagram showing the various units, their contents and their interrelationships, accompanied by an explanatory narrative. Figure 9 shows a sample overall structure diagram for a small electronic book application.

Several features of the structure diagram are worth noting. First, individual units of the document, in this case pages, are represented by rectangles. Each rectangle is labeled to indicate what part of the publication it represents. The completed publication must include a page for each rectangle in the structure diagram. Generally, notes within, next to or below a rectangle indicate the contents of the unit.

Lines between rectangles represent links. A link may be followed in the direction of an arrow. Navigational controls must be implemented to allow the user to navigate between units as indicated by the links in the structure diagram.

Those items which appear on all, or nearly all, units are shown as part of a "master page." In the figure, the PREV PAGE, NEXT PAGE, and CONTENTS buttons are indicated as being part of the master page.

Figure 9: Sample Overall Structure Diagram


MASTER PAGE
PREV PAGE, NEXT PAGE, CONTENTS
COVER
Title, author, NeoBook credit, Date, OPEN BOOK
CONTENTS
List & link to each article , CLOSE BOOK
ARTICLE 1
ARTICLE 2
ARTICLE 3 (PAGE 1)
ARTICLE 3 (PAGE 2)

NOTES: 1. Each block represents one page in the publication.
2. Lines connecting blocks represent navigational controls for moving between pages. Arrows indicate direction of links.
3. Link between CONTENTS and ARTICLE 3 (PAGE 2) is a one-way link back to the CONTENTS page.
4. MASTER PAGE contains controls common to all or most of the pages.

Master Page Layout

After the overall structure of the courseware was determined, the layout of each page, or type of page, was designed. The layout consists of a rough sketch of the page showing the approximate location of objects on the page. Page layout of this type is often referred to as storyboarding. With an interactive authoring system, such as NeoBook Professional (1994) which was used in this development project, it is sufficient to lay out roughly the areas for each of the objects since the objects can be moved and resized dynamically during implementation to achieve the best results.

The first page to be specified is normally the master page since the objects included on the master page will be present on all other pages. Figure 10 shows a sample master page layout for the electronic book example. The Graphs & Networks courseware does not use a master page.

In the figure, three objects, all buttons, are indicated. Their function and approximate location are indicated. These objects would typically be included on each page of the courseware application. Individual objects on specific pages which are not needed could be disabled by covering them with a filled rectangle or by some other means. For example, the NEXT PAGE button would not be needed on the last page of the publication. A solid rectangle could be placed over the button to hide and disable it.

Individual Page Layout

Once the master page had been designed, the layout of the individual pages was done. There were several different styles, or layouts, used for individual pages. The layout simply shows the approximate location and size of objects, the exact location and size were determined during the interactive authoring task. A sample page layout of the individual pages for the small electronic book is shown in Figure 11.

Figure 10: Sample Master Page Layout

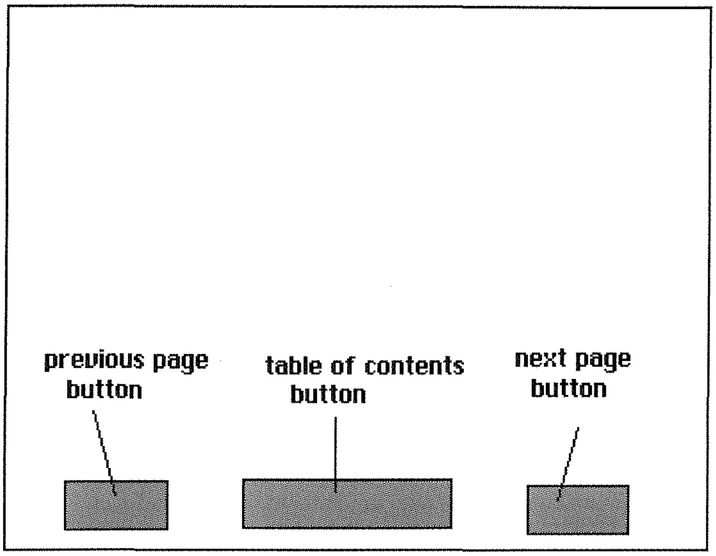
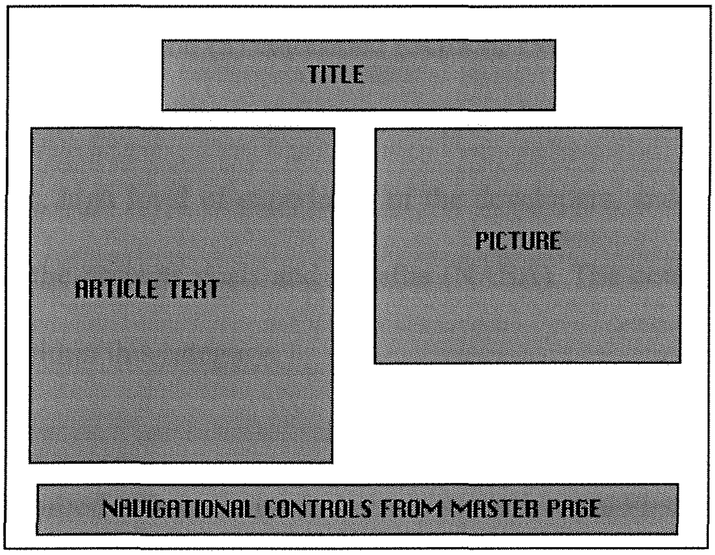


Figure 11: Sample Page Layout for Individual Pages



Planning

The planning phase of the software engineering process typically focuses on two main issues: (1) cost and (2) duration. Cost may include costs for equipment, tools and other materials needed to do the development, but is primarily concerned with labor cost, how many person-months of effort will be required. The duration is determined by allocating the required person-months of effort over time based on the number of development personnel and the scheduling of the various development tasks (Schach, 1993).

The method for determining development cost used for this project was the Intermediate Constructive Cost Model (COCOMO) as presented in Schach (1993). The model was developed by statistically analyzing the development effort required for a range of development projects. The model divides development projects into three groups: (1) organic, (2) semidetached, and (3) embedded (Schach, 1993). An organic development is characterized by its small size (less than 50,000 lines of code), relatively small development team, high level of experience of the developers, and high level of overall understanding of the projects goals and benefits (NASA). The courseware developed in this project falls within this category.

Estimating the required effort was carried out in several steps. First, the number of lines of program code to be produced was estimated. The estimate is stated in terms of KDSI or thousand delivered source instructions. Generally, this would be the number of lines of coding divided by one thousand. Estimates are typically made based on experience with similar projects. For this project, the estimate of KDSI was based on the approximate lines of coding for a prototype version of the software produced for the course DCTE 770/870 Courseware Design and Development at Nova Southeastern University during the Winter Term of 1996.

The estimate of KDSI was done in two parts corresponding to the two development tools being used: QuickBASIC 4.5 (1990) and NeoBook Professional (1994). The number of lines of QuickBASIC 4.5 code for the prototype courseware was extrapolated for the remaining modules to be programmed. A similar approach was used with the portions developed using NeoBook Professional although source line determination was modified due to the nature of development with NeoBook Professional.

The point-and-click development technique of NeoBook Professional (1994) produces a publication file. The lines in this file can be thought of as source instructions to the NeoBook Professional run-time package. Based on the developer's experience with programming and with developing NeoBook Professional applications, it is estimated that 10 lines of NeoBook Professional source code is equivalent to 1 line of programming, in terms of development effort. That is, it is ten times faster to develop a screen with NeoBook Professional than to program it directly. This is probably a conservative estimate. Therefore, the size of the NeoBook Professional publications were estimated based on the prototype, then divided by 10 to yield a KDSI figure which was added to that for the QuickBASIC 4.5 (1990) programming.

similar projects. For this project, the estimate of KDSI was based on the approximate lines of coding for a prototype version of the software produced for the course DCTE 770/870 Courseware Design and Development at Nova Southeastern University during the Winter Term of 1996.

The estimate of KDSI was done in two parts corresponding to the two development tools being used: QuickBASIC 4.5 (1990) and NeoBook Professional (1994). The number of lines of QuickBASIC 4.5 code for the prototype courseware was extrapolated for the remaining modules to be programmed. A similar approach was used with the portions developed using NeoBook Professional although source line determination was modified due to the nature of development with NeoBook Professional.

The point-and-click development technique of NeoBook Professional (1994) produces a publication file. The lines in this file can be thought of as source instructions to the NeoBook Professional run-time package. Based on the developer's experience with programming and with developing NeoBook Professional applications, it is estimated that 10 lines of NeoBook Professional source code is equivalent to 1 line of programming, in terms of development effort. That is, it is ten times faster to develop a screen with NeoBook Professional than to program it directly. This is probably a conservative estimate. Therefore, the size of the NeoBook Professional publications were estimated based on the prototype, then divided by 10 to yield a KDSI figure which was added to that for the QuickBASIC 4.5 (1990) programming.

Using the KDSI estimate a nominal development effort was calculated using the following formula which is given in Schach (1993):

$$\text{Nominal effort} = 3.2 \times (\text{KDSI})^{1.05} \text{ person-months}$$

After the nominal effort was calculated, it was adjusted using a set of multipliers which characterize the product, platform, development personnel, and project. Table 5 gives the factors and their associated multipliers used in determining the correction to the nominal effort.

Each characteristic, referred to as a cost driver in the table, was estimated for the courseware development. The fifteen factors were multiplied together to yield a single value which was then multiplied with the nominal effort to give the actual estimate of total development effort.

To help monitor progress, the overall development effort estimate was allocated to the various development phases. This allocation was done according to the percentages of total development shown in Table 6 based on information in Schach (1993).

Table 5: COCOMO Software Development Effort Multipliers (Schach, 1993).

	Rating					
Cost Drivers	Very Low	Low	Nom.	High	Very High	Extra High
Product Attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Database size		0.94	1.00	1.08	1.16	
Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
Execution time constraint			1.00	1.11	1.30	1.66
Main storage constraint			1.00	1.06	1.21	1.56
Virtual machine volatility*		0.87	1.00	1.15	1.30	
Computer turnaround time		0.87	1.00	1.07	1.15	
Personnel Attributes						
Analyst capabilities	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Programmer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience*	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project Attributes						
Use of modern programming practices	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

* For a given software product, the underlying virtual machine is the complex of hardware and software (operating system, database management system) it calls on to accomplish its task.

Table 6: Percentage of Development Effort for Development Phases

Phase	% of Total Development
Requirements Analysis	9
Specification	12
Planning	7
Design	18
Implementation/Integration	54

Finally, once the total development effort was determined and distributed to each phase of development, a schedule was determined. Since the development was undertaken by a single developer, all phases were performed in sequence, not allowing for two activities to be performed simultaneously. The hours/month which the developer planned to allocate to the development work was used to translate person-months into calendar months. The result was shown as a total development duration and allocated to the individual development phases as described above.

Evaluations of the courseware were scheduled into the development schedule as indicated in the section below on **Formative Evaluation**. Timing depended on the availability of faculty and students who performed the evaluations.

Design

In the specification phase the functionality of the courseware was defined. What the product will do and what elements will constitute the finished product were specified. Emphasis was on what will be done, not on how it would be done. In the design phase implementation details were specified. In the case of program development, the specific algorithms for accomplishing the specified functionality were defined and recorded in narrative form.

Design of the elements of the courseware which were implemented using NeoBook Professional (1994), primarily the user interface, consisted of specifying screen layouts and button actions.

Cognitive Learning Theory Design Considerations

A major concern of the design phase was the intentional use of various elements of cognitive learning theories to guide the design. The specific considerations are described below with the more general concerns described first, then those which are more specific.

1. The courseware incorporates various instructional models including: (1) tutorials using text and static graphic images, (2) simple animated demonstrations, (3) interactive, animated laboratory exercises, and (4) self-tests. The intention was to include a reasonably complete instructional experience with the courseware.

2. The courseware is designed to be a supplement to normal instructional activities. Despite the breadth of instructional activities included, the courseware is not intended to be used in a stand-alone manner. Rather, it is intended to be used as a supplement to normal instructional activities including reading, lectures, and programming assignments.

3. To facilitate use of the courseware and incorporation into the normal curriculum, an instructor guide is provided with the courseware. The goal is to promote proper use of the courseware.

4. The overall structure of the courseware is that of a hypermedia system. This structure is consistent with cognitive learning theories which contend that knowledge itself is organized as a network and that the use of hypermedia supports this structure (Ambrose, 1991; Lanza, 1991). The various units of content and activities are linked allowing relatively free navigation by the learner.

5. The root of the hypermedia network is a table of contents with links to all topics and online help. Each topic has a central point from which the learner can branch to a specific activity or return to the table of contents. The intent is to minimize learner disorientation and facilitate navigation.

6. The courseware allows the learner to navigate freely, but provides guidance regarding optimum sequencing.

7. Where appropriate, the courseware allows the learner to experiment with the effects of various algorithms to promote a better understanding of algorithm operation. Within some necessary constraints, such as screen space, learners are able to specify a data structure and watch an animated sequence as an algorithm acts on the data structure. This feature is similar to algorithm animators being used in Computer Science courses, with the added benefit of being incorporated in a courseware package which provides a relatively complete instructional environment and is designed to achieve specific,

8. The courseware provides extensive learner support features including online, context-sensitive help; recommended sequence of activities; and immediate feedback. The intent is to support the learner's metacognitive activities.

9. The user interface minimizes cognitive load by using consistent displays, navigational controls, and a familiar metaphor, that of a computer laboratory.

10. Screens were designed using a consistent format, a limited palette of colors, do not use scrolling, have labeled navigational controls, and are not overcrowded. Animations and other displays generated by the QuickBASIC 4.5 (1990) programs are presented in a window within a page created using NeoBook Professional (1994) to provide consistency and continuity.

11. The courseware is designed to minimize the possibility of learner errors by providing mouse-selectable options rather than using keyboard input.

12. Sound and color are used to draw attention to or emphasize important information, but are not over-used. Simple animation is used to make visible the dynamic operations of algorithms acting on data structures.

Implementation and Integration

Because of the hybrid nature of this development, implementation and integration deal with procedures for using the authoring tool, NeoBook Professional (1994), and with procedures for programming the QuickBASIC 4.5 (1990) programs.

NeoBook Professional (1994) was used to create the user interface, the overall structure of the courseware, the general online help, and the tutorial activities. NeoBook Professional was chosen as the authoring tool for several reasons including: (1) cost, (2) ease of use, (3) its ability to call an external program, and (4) its ability to compile applications for stand-alone execution.

QuickBASIC 4.5 (1990) was selected as the programming language for two specific reasons. First, changing the palette to match the one selected for the NeoBook Professional portion is difficult in Turbo Pascal 6.0 (1990), the other language considered for the implementation. Second, it is possible to start a compiled QuickBASIC 4.5 program in graphics mode without resetting the monitor screen. This is not possible in Turbo Pascal. For consistency of appearance it is desirable to have the QuickBASIC 4.5 programs appear to run within a window on the NeoBook-displayed screen. This cannot be done if the screen is reset when the program is run.

One aspect of the authoring activity using NeoBook Professional (1994) was the creation or location of the media units, such as text files and graphic images which were needed to produce the courseware. Creation and editing of graphic images was done using NeoPaint (1994), a companion product to NeoBook Professional.

A second aspect is that of assembling the media units onto pages and linking the pages according to the overall courseware structure. The assembly process is simplified by using NeoBook Professional since pages were created and buttons and other objects placed using a point-and-click interface. This allowed the author to create and alter each page dynamically during the authoring task to achieve the best results without having to program or write scripts.

Despite the ease of use of NeoBook Professional, there are techniques which were used to further simplify the assembly stage, and to promote consistency. These techniques include stub authoring, cloning buttons or other objects, cloning pages, and modular implementation.

Stub Authoring

The technique of stub authoring is derived from the equivalent practice in computer programming called stub programming. In stub authoring, the author creates all units, or pages, of the courseware, along with the navigational controls, but does not add the media content to the individual pages. The idea is to implement the overall structure and to verify that the overall structure is valid and that navigational controls are properly implemented. The content-less pages are the stubs. Once the entire structure has been implemented and checked, the author proceeds to add content a page at a time, confident that the overall structure is correct.

A useful variation on the technique of stub authoring is to create a dummy media unit of each type to be used in the publication: text, graphic, etc. Individual pages can be created with the dummy units as the structure is implemented and tested, then modified later to include the actual media units.

Cloning Buttons or Other Objects

As a page is implemented, either during stub authoring or during actual implementation, buttons which have a similar size will be duplicated rather than creating each button separately. This process of cloning buttons will save development time and ensure a consistent look for similar buttons. Once a duplicate button has been created, it was moved to the desired location. Attributes of the button such as the fill color and pattern; text font, color or pattern were altered as needed.

Cloning Pages

Duplicating entire pages can vastly simplify creation of a publication. If pages had the same structure, but different content, a new page was made by making a duplicate copy of the page and editing it as needed.

Modular Implementation

The courseware was implemented in a modular fashion. The basic concept was to divide the application into relatively small, coherent units and link the units together to create the complete application. There are several advantages.

Due to the hypermedia nature of the overall design, the courseware consists of many small units of information or activities. Implementation was simplified by dealing with individual units one at a time rather than the entire application. Also, to avoid the possibility that the NeoBook Professional (1994) portion would be too large to run in available memory, individual modules are called from a main module.

At various stages evaluation of the courseware indicated the need for modifications. Making these modifications was simplified by modularization. If a particular portion of the application needed to be modified, it was changed and recompiled. The entire application did not need to be recompiled as long as associated links were not changed.

By modularizing the implementation and using the technique of stub authoring, major stubs were implemented as separate modules. Initially, these stubs did not contain actual content. As a stub was completed, it was recompiled and the already existing link to it caused the completed unit to be displayed.

QuickBASIC Programming

QuickBASIC 4.5 (1990) was used to program the animated demonstrations, interactive laboratory exercises, the self-test activities, and online help associated with these elements of the courseware. The general development procedures were modular implementation, reuse of modules, and the use of abstract data types. Modular implementation was similar to the technique described above for the authoring process. The programs were divided into modular sections for ease of programming and maintenance. Wherever possible, code modules written for one program were reused in others saving development time and promoting reliability.

An abstract data type is a data structure which represents some entity and the set of associated operations which are valid for that structure. The use of abstract data types in the implementation facilitated reuse of entities within different portions of the program, and enhanced reliability of the software. The latter occurs because entities are manipulated only according to defined operations. The use of abstract data types is akin to object-oriented programming. QuickBASIC 4.5 (1990) does not support object-oriented programming.

Formative Evaluation

The purpose of formative evaluation is to "provide data for revising and improving instructional materials" (Gagne`, Briggs & Wager, 1992, p.30). In keeping with software engineering methodology, formative evaluation was conducted throughout the design and development process (Schach, 1993). The procedures used as well as any instrumentation are described in this section.

Evaluation of Requirements Analysis and Specification

The result of the requirements analysis and specification phases is the Functional Specification which documents the user requirements and the functional definition of the finished product. The format and contents of this document are described in the following section **Presentation of Results**.

Evaluation of the Functional Specification consisted of a review by a Computer Science faculty member at Mount Vernon Nazarene College who is familiar with the subject area and with the problems associated with learning abstract structures, concepts and operations. The reviewer read the document and noted comments on the document. Subsequent to the review, the developer met with the reviewer and reviewed the Functional Specification page by page to understand the comments.

After the review session, the developer modified and corrected the Functional Specification as needed to satisfy the reviewer. The developer and the reviewer then met and reviewed the corrections and modifications to determine if they were satisfactory. The process was repeated until the developer and the reviewer agreed on the content of the Functional Specification.

Evaluation of Design

The Design Specification, which contains the details of the courseware design, was produced during the design phase. This document underwent a review process similar to the one described for the Functional Specification, except that a review team consisting of two Computer Science faculty members, and one non-Computer Science major reviewed the document. Evaluation focused on how well the design met the functional requirements documented in the Functional Specification.

Evaluation of Implementation

As the courseware was being developed, completed portions were tested and evaluated by the developer to see that they conformed to the Design Specification. Modifications and corrections were made as needed.

Once the courseware was completed, it went through a series of evaluations. In each evaluation, one or more evaluators used the product, noted problems and comments, and completed a Courseware Evaluation Form. The Courseware Evaluation Form is included in Appendix B. The developer discussed the problems and comments noted with the evaluators and made necessary modifications.

The first evaluation was by a Computer Science faculty member, the second by two or three upper-class Computer Science majors who were familiar with the content of the courseware, and the final evaluation by a small group or class of Computer Science majors who were not familiar with the content of the courseware.

Presentation of Results

The development yielded two types of results: (1) the completed courseware and related documentation, and (2) evaluation results. The courseware is presented in the appendix of the dissertation document. Separate appendices are included for the Functional Specification, Design specification, Instructor's Guide, and QuickBASIC 4.5 (1990) program listings.

The Functional Specification is presented using the following format:

Cover Page including Project Name, Developer's Name, Date

Table of Contents

1.0 Problem Description and Goals

2.0 Instructional Analysis

3.0 User Characteristics and Prerequisites

4.0 Functional Definition

4.1 Overall Structure

4.2 Individual Unit Specifications

5.0 System Requirements

The Design Specification is presented using the format shown below.

Cover Page including Project Name, Developer's Name, Date

Table of Contents

1.0 Introduction

2.0 Overall Design

3.0 Individual Unit Designs

4.0 System Requirements

Summaries of evaluation results are presented in Chapter IV of the dissertation with complete results being included in an appendix. Modifications made to the courseware in response to evaluations are discussed and explained.

Reliability and Validity

The reliability and validity of the courseware development and evaluation are inherent in the methodology which was followed and insured by conducting evaluations throughout the development process. The developed courseware must enhance a learner's ability to meet specific, measurable learning objectives. This was accomplished by: (1) using proven software engineering development methodology, (2) incorporating instructional design principles, and (3) applying current cognitive learning theories.

Software engineering methodology has proven reliable in the design and development of software systems in general. This reliability carries over to development of educational courseware provided appropriate educational considerations are taken into account (Mauldin, 1995). The first of these is the use of the techniques and principles of instructional design. Instructional design techniques have proven reliable for the design and development of instructional materials which were not computer-based. Incorporating these principles into the various phases of the software engineering methodology combines two proven techniques.

Finally, to make use of the ability of the computer to provide higher level cognitive activities, practical considerations derived from cognitive theories of learning were incorporated into the design of the courseware.

Summary

The guiding methodology of this dissertation was that of software engineering. Courseware which will enhance student learning of abstract data structures and algorithms was developed and evaluated according to this methodology which includes requirements analysis, specification, planning, design, and implementation and integration. Evaluation was conducted throughout the development process.

Principles of instructional design were integrated into the development process to apply the development activities to the development of computer-based instructional materials. To insure that the courseware promotes higher-level cognitive activities such as application, analysis and synthesis, cognitive learning theories were used to guide the design.

The overall structure of the courseware is that of a hypermedia system. Online help, labeled navigational controls, consistent user interfaces, and other design features help prevent the learner from becoming disoriented in the hypermedia environment.

The courseware was developed using an interactive authoring system, NeoBook Professional (1994), and a conventional programming language, QuickBASIC 4.5 (1990). The authoring system facilitated development of the user interface and overall courseware structure, while the programming language was used to implement features not present in the authoring package including the animated demonstrations, interactive laboratory exercises, and self-tests.

Development practices such as modular implementation, reusability of code or other entities, and the use of abstract data types facilitated implementation and modification of the courseware. Evaluation of the Functional Specification, Design Specification and of the developed product helped to insure that the courseware met its development goals.

Chapter IV

Results

Introduction

As described in **Chapter I Introduction**, the goals of this dissertation were to enhance professional practice, improve understanding in the area of interactive courseware design and development, and to develop courseware which would facilitate meeting specified learning objectives. These goals were accomplished by (1) applying the concepts and practices of software engineering, instructional design, and cognitive learning theory in the development of interactive, multimedia courseware; (2) effectively using authoring tools and conventional programming languages in courseware development; and (3) conducting formative evaluations of the courseware throughout the development process.

The development yielded two types of results: (1) the completed courseware and related documentation, and (2) the results of user evaluations. The first of these demonstrates the effectiveness of using software engineering and instructional design techniques to carry out the development and the efficiency realized by using an authoring system and a conventional programming language as development tools. The second type of result, user evaluations, provided feedback about the usability and probable effectiveness of using the completed courseware in an instructional situation.

In addition to these two broad types of results, it is important to analyze the effects of applying elements of instructional design and applicable cognitive learning theories to the design of the courseware.

Courseware Development

The overall methodology of the development work was that of software engineering, a proven methodology for developing complex software systems. The major steps in the development process consisted of: (1) requirements analysis, (2) specification, (3) planning, (4) design, and (5) implementation and integration. The details of these development activities are presented in **Chapter III Methodology**.

Functional Specification

The primary result of the requirements analysis and specification tasks was the Functional Specification. This document presents a description of the problem to be solved, the development goals, an instructional analysis of the learning task, and a summary of the characteristics of intended users of the courseware. The document also includes an overall structure of the courseware and detailed functional descriptions of individual units of the courseware.

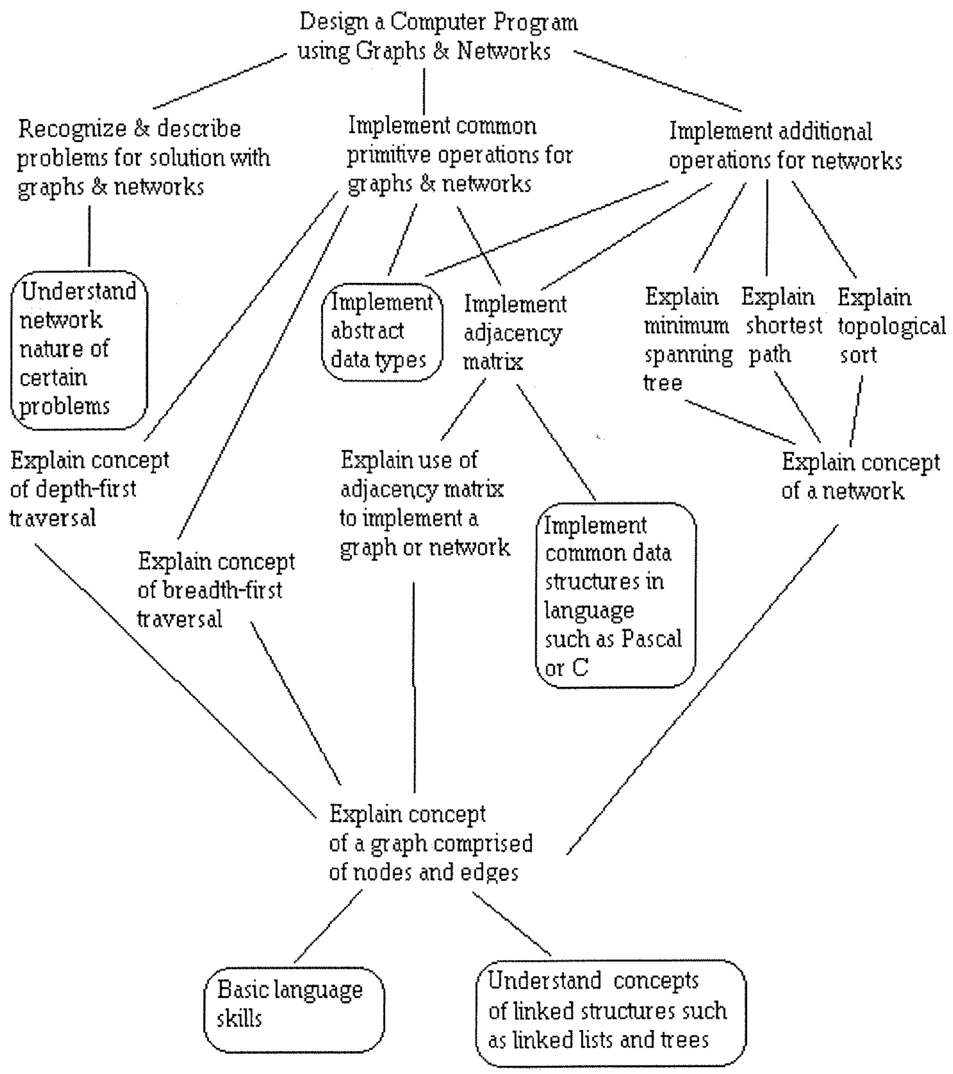
The Functional Specification presents the functional requirements of the courseware; that is, it concentrates on what functions the courseware is to perform. The emphasis is not on how these functions will be accomplished. The document includes user interface designs and a description of the content of various portions of the courseware. The complete Functional Specification is included in **Appendix C**.

Instructional Curriculum Map

An important part of the development was identifying what concepts and skills were requisite to accomplishing the learning objectives. Some of these concepts and skills were assumed to be prerequisites while others constituted intermediate learning objectives. An instructional curriculum map (ICM) was developed by starting with the primary overall objective of the courseware, being able to design a computer program which uses graphs and networks, and decomposing this overall objective into requisite learning objectives. The process was repeated until skills or knowledge which were needed were those assumed to be possessed by the intended users.

The ICM for the Graphs & Networks instructional software is shown in Figure 12 below and in Figure 1 of the Functional Specification. Items in round-cornered boxes represent skills or knowledge which are assumed prerequisites for the intended users. These items are commonly learned in introductory Computer Science courses prior to a student's first exposure to graphs and networks.

Figure 12: Instructional Curriculum Map for Courseware


Design a Computer Program using Graphs & Networks
Recognize & describe problems for solution with graphs & networks
Implement common primitive operations for graphs & networks
Implement additional operations for networks
Understand network nature of certain problems
Implement abstract data types
Implement adjacency matrix
Explain minimum spanning tree
Explain shortest path
Explain topological sort
Explain concept of depth-first traversal
Explain use of adjacency matrix to implement a graph or network
Implement common data structures in language such as Pascal or C
Explain concept of a network
Explain concept of breadth-first traversal
Explain concept of a graph comprised of nodes and edges
Basic language skills
Understand concepts of linked structures such as linked lists and trees

All of the other items in the ICM are addressed by the courseware. Starting at the bottom of the figure: explain concept of a graph comprised of nodes and edges, explain concept of breadth-first traversal, and explain concept of depth-first traversal are covered in the unit on Graphs & Digraphs.

Items relating to networks: explain concept of a network, explain minimum spanning tree, explain shortest path, and explain topological sort, are covered in the courseware unit on networks. Items in the ICM related to implementation depend on the basic units already mentioned and on the additional units on graph implementation and network implementation.

One final item, recognize and describe problems for solution with graphs and networks, is not covered as a separate topic, but is incorporated into various tutorials and laboratory sessions, primarily in the form of examples.

Learner Characteristics

In addition to analyzing the instructional characteristics of the courseware, the characteristics of the intended learners were determined. The technique was to summarize learner characteristics using a chart of the form shown in Figure 8 in **Chapter III Methodology**. This technique was patterned after a similar chart given in Alessi and Trollip (1991). Specific prerequisite skills determined from the instructional analysis and the primary learning objective were included in the chart in addition to certain basic characteristics such as year in college, motivation, etc. The chart of learner characteristics is presented in Table 7 below and in Table 2 of the Functional Specification.

Table 7: Learner Characteristics for Graphs & Networks Courseware

		Level		Time to	Difficulty
	Low	Avg	High	Learn	to Learn
Year	Soph	Junior	Senior		
Experience	Prereq[1]	Prereq[1]	Prereq[1]		
Motivation	High	High	High		
Interest	High	High	High		
Computer Operation	High	High	High		
Courseware Familarity	None	None	None	10-30 min	Easy
Basic Language Skills	Average	Average	Average		
Understand Concepts of Linked Structures	Low	Average	High		
Implement Common Data Structures	Average	High	High		
Implement Abstract Data Types	Average	High	High		
Understand the Network Nature of Certain Problems	Average	High	High		
Implement Graph & Network Data Structures and Algorithms	None	None	None	4 - 8 Hours	Difficult

1. Prereq. denotes completion of Computer Science I, Computer Science II, and File Processing

The chart rates potential learners, Sophomore, Junior, and Senior Computer Science majors, with respect to the mastery of prerequisite skills and knowledge. The first row of the chart shows the year, Sophomore, Junior, or Senior, of the learner corresponding to chart, all learners are assumed to have at least a minimum set of prerequisite skills and knowledge which would be normally attained by completing the first several courses in a typical Computer Science curriculum.

Since all of the intended learners are Computer Science majors, their motivation, interest, and ability to operate a computer are considered to be high. It is also assumed that the students have not used this particular courseware before, so their familiarity with it is rated as "none." Nevertheless, because the design follows instructional design principles and appropriate cognitive learning theories, it is estimated that students can master use of the courseware in a relatively short time, 10 - 30 minutes, and that this task is "easy."

All students were rated as having "average" basic language skills. However, Sophomore students were rated "low" in their ability to implement linked structures compared to "average" for Juniors and "high" for Seniors. This rating is based on the amount of experience students have typically had implementing linked structures in their various courses.

Sophomore students are assumed to be "average" in their ability to implement common data structures such as arrays and records; in their ability to implement abstract data types such as stacks and queues; and in their understanding of the network nature of certain problems. Juniors and Seniors, however, are rated "high" in these areas based on their greater experience.

The rating of "none" for all students with respect to implementing graph and network data structures and algorithms assumes that learners using the courseware are encountering these topics for the first time.

Design Specification

The design specification documents the overall and detailed design of the courseware. It includes a description of the overall design, the software structure of the final product, and detailed descriptions of all data structures, file formats and major software routines. The choice and use of color and the use of NeoBook Professional (1994) and QuickBASIC 4.5 (1990) in the development are described.

The animated demonstrations, laboratory sessions, and self-tests were implemented in a single QuickBASIC program. Specific demonstrations, laboratory sessions, and self-tests were implemented using data files which are processed by the program. To simplify development and maintenance, and to provide a more powerful program, abstract data types were implemented for user interface features such as buttons and regions, for graph and network abstractions, and for several supporting features such as queues and general purpose functions. The data structures and algorithms for these abstract data types are described in detail in section 3 of the Design Specification. The complete Design Specification is included in **Appendix D**.

User Interface Abstract Data Types

The user interface provided by the demonstration program was designed to provide convenient, intuitive operation. It consists of display regions and buttons which can be used for various functions depending on the specific application. Buttons are redefined as needed to support animated demonstration controls, laboratory session controls, and self-test actions. Fixed and pop-up screen regions are used for various functions such as demonstration narration, displaying help, and presenting lab problems or self-test questions. Figure 13 shows a screen capture of the demonstration program user display for the depth-first traversal laboratory session.

Notice that the user display includes several main regions. A description of the laboratory session is included in the top portion of the display, the text region. The lower left half of the display is a work region, used to display program- or user-generated graphs and networks. The graph shown in the figure was created interactively by a user. The lower right portion of the display includes two small regions for displaying information and special messages, and a set of buttons. Only certain buttons are active for the laboratory session depicted in the figure.

To facilitate implementation of the user display two abstract data types, ButtonType and Region, were defined. The ButtonType data type was used to implement three-dimensional buttons. The Region data type provides a window-like screen area which can be written to, cleared and manipulated in other ways. The specific operations provided for items of type ButtonType and Region are summarized in Table 8.

Figure 13: Demonstration Program User Display Screen

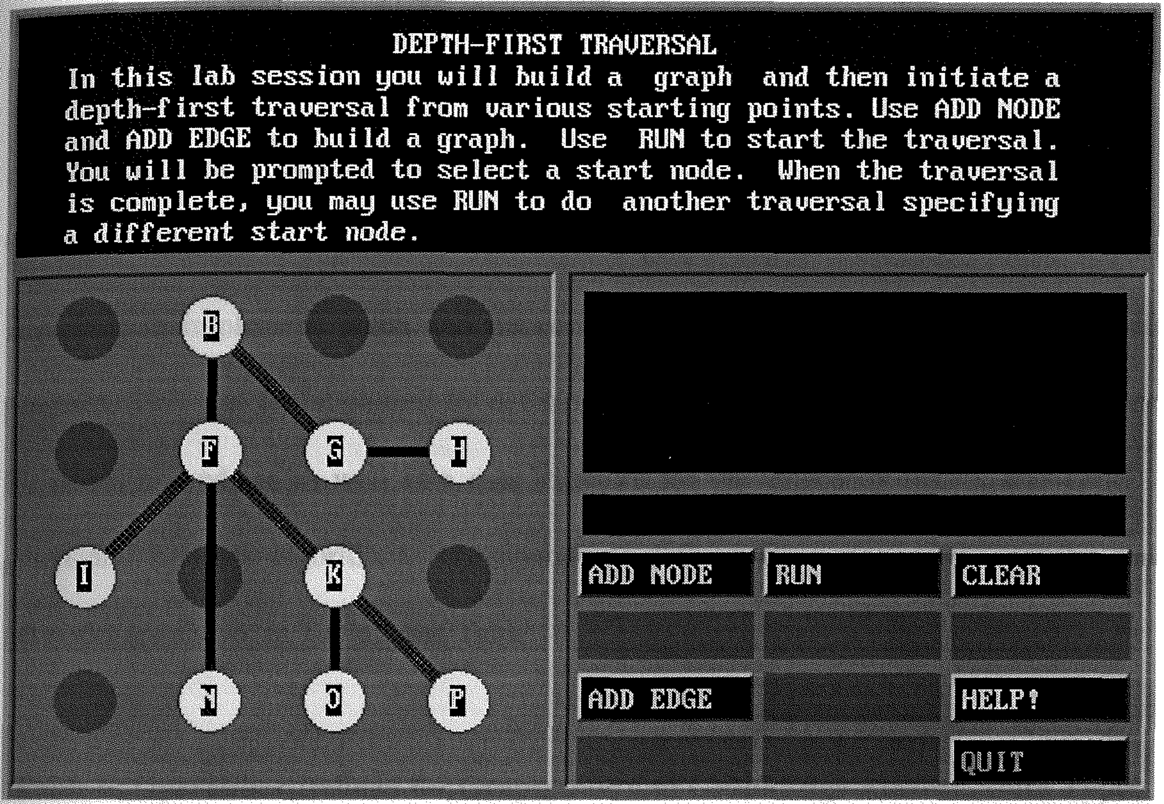

DEPTH-FIRST TRAVERSAL
In this lab session you will build a graph and then initiate a depth-first traversal from various starting points. Use ADD NODE and ADD EDGE to build a graph. Use RUN to start the traversal. You will be prompted to select a start node. When the traversal is complete, you may use RUN to do another traversal specifying a different start node.

Table 8: Operations Defined for User Interface Abstract Data Types

Operation	**Description**
ButtonAction	carries out a specified action when the button is pressed.
ButtonCreate	creates a button with specified attributes
ButtonDraw	displays a button
ButtonHide	hides a button and sets it to inactive
ButtonPress	displays a button so that it appears to be pressed in
ButtonUnhide	sets a button to active
RegionBorder	draws a three-dimensional border around a region
RegionClear	clears a region
RegionConfirm	displays a pop-up region used to confirm a user action
RegionCreate	creates a region with specified attributes
RegionPrint	prints text to a designated region

A detailed description of the data structures and the operations for user interface abstract data types is included in the Design Specification in **Appendix D**.

Graph and Network Abstract Data Types

To allow students to experiment with various graph and network algorithms, it was important to implement graph and network abstract data types in the demonstration program. This was accomplished by defining data structures and appropriate operations for nodes and edges, as well as a data structure for the adjacency matrix and various graph and network algorithms. The operations defined for nodes, edges, graphs and networks are shown in Table 9 and Table 10.

Table 9: Operations Defined for GraphNode and GraphEdge Abstract Data Types

Operation	**Description**
NodeCenter	returns the center of the circle which represents the node
NodeChangeColor	sets the display color of a node to a new value
NodeCreate	creates a node with specified attributes
NodeHide	hides a node on the user display
NodeShow	displays a node on the user display
NodeShowDimmed	displays a dimmed version of a node on the user display
EdgeAdd	adds a bi-directional edge to a graph
EdgeArrowHead	displays an arrowhead for a directed edge in a digraph
EdgeColorChange	sets the display color of an edge to a new value
EdgeCreate	creates an edge between specified nodes with a specified weight
EdgeDelete	removes an edge from a graph
EdgeFind	finds the edge associated with two specified nodes
EdgeHide	hides an edge on the user display
EdgeHighlight	highlights an edge on the user display
EdgeShow	displays an edge on the user display

Table 10: Operations Defined for Graph and Network Abstract Data Types

Operation	Description
GraphBreadthFirstTraversal	performs a breadth-first traversal of a graph
GraphDepthFirstTraversal	performs a depth-first traversal of a graph
GraphEmpty	returns TRUE if a graph is empty
GraphInit	initializes a graph
GraphShow	displays a graph on the user display
NetMinSpanTree	finds the minimum spanning tree of a network
NetShortestPath	finds the shortest path between two nodes
NetTopologicalSort	performs a topological sort of a network

Other data structures and algorithms which support the interactive, animated display of graphs and networks are described in detail in the Design Specification in **Appendix D**.

Planning

Planning is a crucial aspect in the development of any large software system, including instructional software. Courseware development typically requires a great deal of time. Therefore, it is important to estimate as accurately as possible the total time required and the duration of the development.

As described in **Chapter III Methodology**, the primary technique used for estimating the total development time was the Intermediate Constructive Cost Model (COCOMO) as presented in Schach (1993). According to the model the development of the Graphs & Networks courseware was classified as "organic." In the COCOMO model a project is considered to be organic if it is relatively small in size (less than 50,000 lines of code), is undertaken by a relatively small development team, and the developers have a high level of experience and a high level of understanding of the project (NASA).

Based on this classification, the relationship used to determine the nominal effort to complete the project is given by the following equation;

$$\text{Nominal effort} = 3.2 \times (\text{KDSI})^{1.05} \text{ person-months}$$

The term KDSI in the equation represents thousands of lines of delivered source instructions and is a measure of the overall size of the project.

The KDSI for this project were determined by estimating the total lines of source code based on the approximate lines of coding for a prototype version of the software produced for the course DCTE770/870 Courseware Design and Development at Nova Southeastern University during the winter Term of 1996. The estimate consisted of two parts, one for the portion generated using NeoBook Professional (1994) and one for the QuickBASIC 4.5 (1990) program. Table 11 shows the estimates for KDSI for the project. The resulting nominal effort is given by the equation following the table.

Table 11: KDSI Estimate for Graphs & Networks

Portion of Courseware	Estimated Source Lines
QuickBASIC program	4000
NeoBook publication	2250
Total Lines	6250
KDSI	6.25

Nominal effort = $3.2 \times (6.25)^{1.05} = 21.9$ person-months

The nominal effort estimate was modified using appropriate effort multipliers as defined in the COCOMO model. Table 5 in **Chapter III Methodology** gives the COCOMO effort multipliers. The actual multipliers used for this project are shown below in Table 12.

Table 12: COCOMO Effort Multipliers for Graphs & Networks

Effort Multiplier	Rating	Value
Product Attributes		
Required software reliability	Low	0.88
Database size	Low	0.94
Product complexity	Nominal	1.00
Computer Attributes		
Execution time constraint	Nominal	1.00
Main storage constraint	Nominal	1.00
Virtual machine volatility	Low	0.87
Computer turnaround time	Low	0.87
Personnel Attributes		
Analyst capabilities	Very High	0.71
Applications experience	Very High	0.82
Programmer capability	Very High	0.70
Virtual machine experience	High	0.90
Programming language experience	High	0.95
Project Attributes		
Modern programming practices	Very High	0.82
Use of software tools	Very High	0.83
Required development schedule	Nominal	1.00

Required software reliability was rated low because the impact of errors or operational problems is low compared to the need for accurate, reliable operation of a large database system or a process control system. In terms of COCOMO, this project was a relatively small, simple project, thus, database size and product complexity were rated low.

With respect to computer attributes, the software does not require highly time-critical operations, nor is a large storage capacity required. The platform on which the software was developed was expected to remain stable during the development. Turnaround time is not an issue at all since development was performed on a single user, interactive system. Therefore, the lowest rating possible for each of these elements was chosen.

Development was carried out by a single developer with approximately thirty years of analysis, applications, and programming language experience. The developer was also very familiar with the development platform and, as a Computer Science professor, has a high degree of programming experience. All personnel attributes were accordingly selected to be the highest available values.

Finally, the development was conducted using modern programming practices such as modular programming and abstract data types. Likewise the use of an interactive programming environment which supports structured programming and the use of an interactive authoring system constituted a high level of using software tools. These factors were rated as high as possible. The development schedule requirement was rated “nominal” since external pressure was not great.

The effort multipliers were then multiplied together to yield an overall effort. The result was then applied to the nominal effort to produce an actual estimate of the required development effort. The results of these calculations are shown below:

Overall Effort Multiplier = 0.148

Estimated Effort = 0.148 x 21.9 = 3.24 person-months (14 person-weeks)

It was further estimated that the developer was working 50% of a full-time schedule on the development project. The person-week estimate was doubled to 28 weeks to account for this.

In addition to the overall estimated effort, the portion of the development time for each phase of the project was estimated. The estimated effort was distributed among the project phases using the factors in Table 6 in **Chapter III Methodology**. A time log of actual development work was maintained throughout the development project to determine the actual development effort. The results for the estimated and actual development effort are shown below in Table 13.

Table 13: Estimated and Actual Effort for Project Phases

Phase	% of Total Development		Person-weeks	
	Estimated	Actual	Estimated	Actual
Requirements Analysis	9	16.5	2.5	4.0
Specification	12	12.4	3.4	3.0
Planning	7	6.2	2.0	1.5
Design	18	14.5	5.0	3.5
Implementation/Integration	54	50.4	15.1	12.2
Total	100	100.0	28.0	24.2

Actual effort was reasonably close to the estimate, although lower than projected overall. The slightly lower than projected design effort is attributed to the incorporation of some aspects of the design into the actual implementation. In particular, screen layouts were designed roughly since implementation with NeoBook Professional (1994) allowed exact locations and layouts to be determined interactively during screen creation.

Instructor's Guide

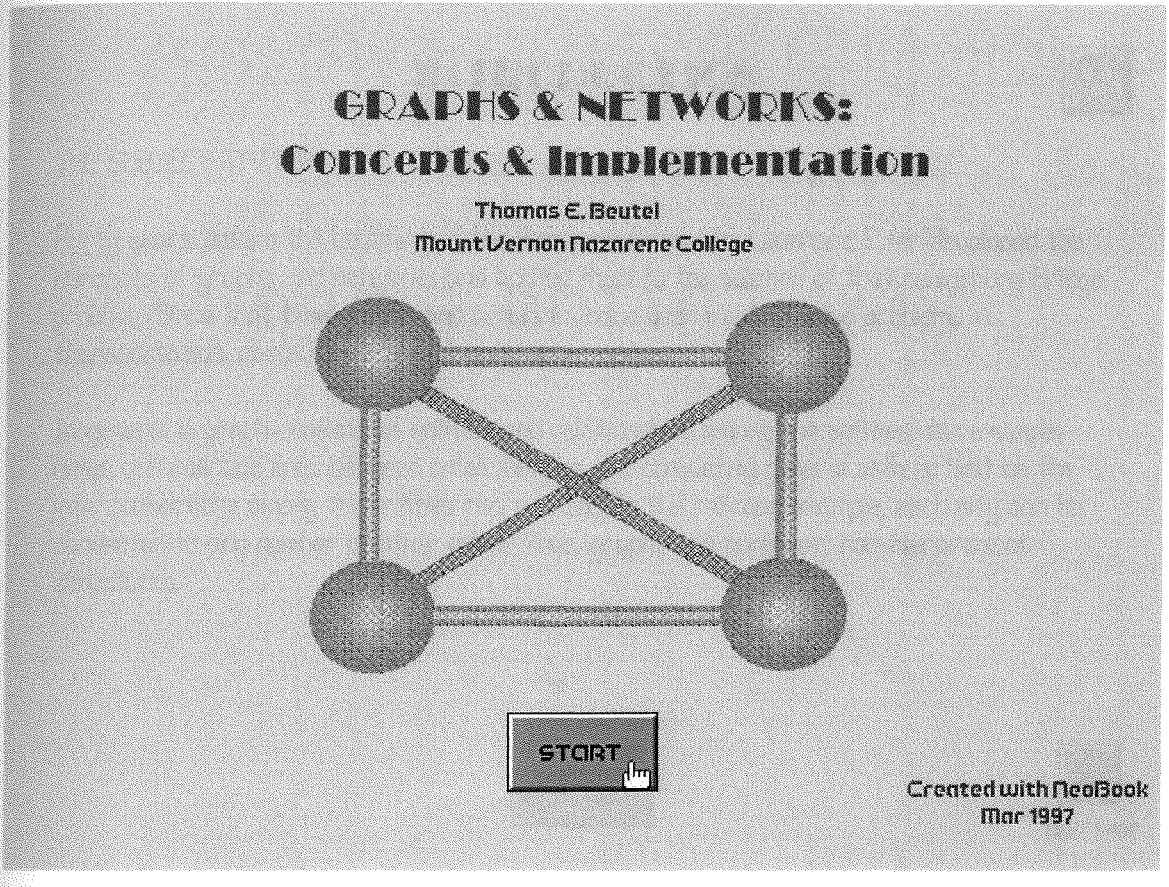
An Instructor's Guide was developed to accompany the courseware. This brief guide includes general information about the courseware as well as instructions for installing the software and system requirements. Suggested uses of the courseware within a normal Computer Science curriculum are described. Each of the basic instructional activities: tutorial, animated demonstration, laboratory sessions, and self-tests, are explained. The Instructor's Guide is presented in **Appendix E**.

Overview of Completed Software

The Graphs & Networks courseware provides instructional activities intended to help Computer Science students meet specific learning objectives related to graph and network data structures and algorithms. The courseware is intended to be used in conjunction with normal classroom lectures, programming and other assignments. The software provides brief tutorials; animated demonstrations; interactive, animated laboratory sessions; and self-tests. The animated demonstrations and interactive, animated laboratory sessions are intended to help the student understand the effects of various algorithms acting on graph and network data structures by presenting dynamic displays.

Figure 14 shows the title screen of the courseware. The learner begins a session by clicking on the START button with the mouse. The session begins with the Introduction in which the subject matter and the courseware are briefly described, the goals are presented, and prerequisites are explained. The first page of the Introduction which explains the subject matter is shown in Figure 15.

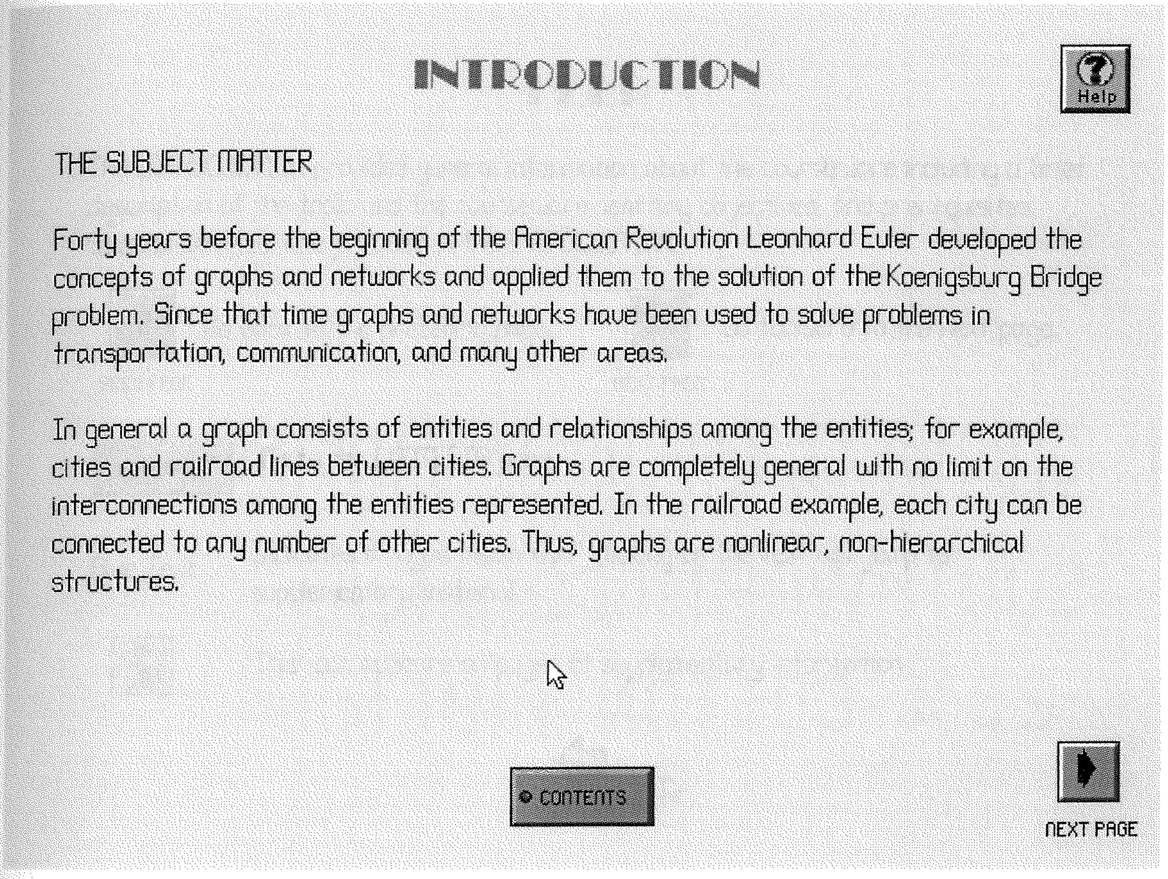
Figure 14: Courseware Title Page Screen


GRAPHS & NETWORKS:
Concepts & Implementation
Thomas E. Beutel
Mount Vernon Nazarene College

Created with NeoBook
Mar 1997

Context-sensitive help is available throughout the courseware. The Help icon appears in the upper right-hand corner of the screen as seen in Figure 15. Clicking on the Help icon brings up a help screen relevant to the current section of the courseware. For example, the help screen for the Introduction is shown in Figure 16.

Figure 15: First Page of the Courseware Introduction



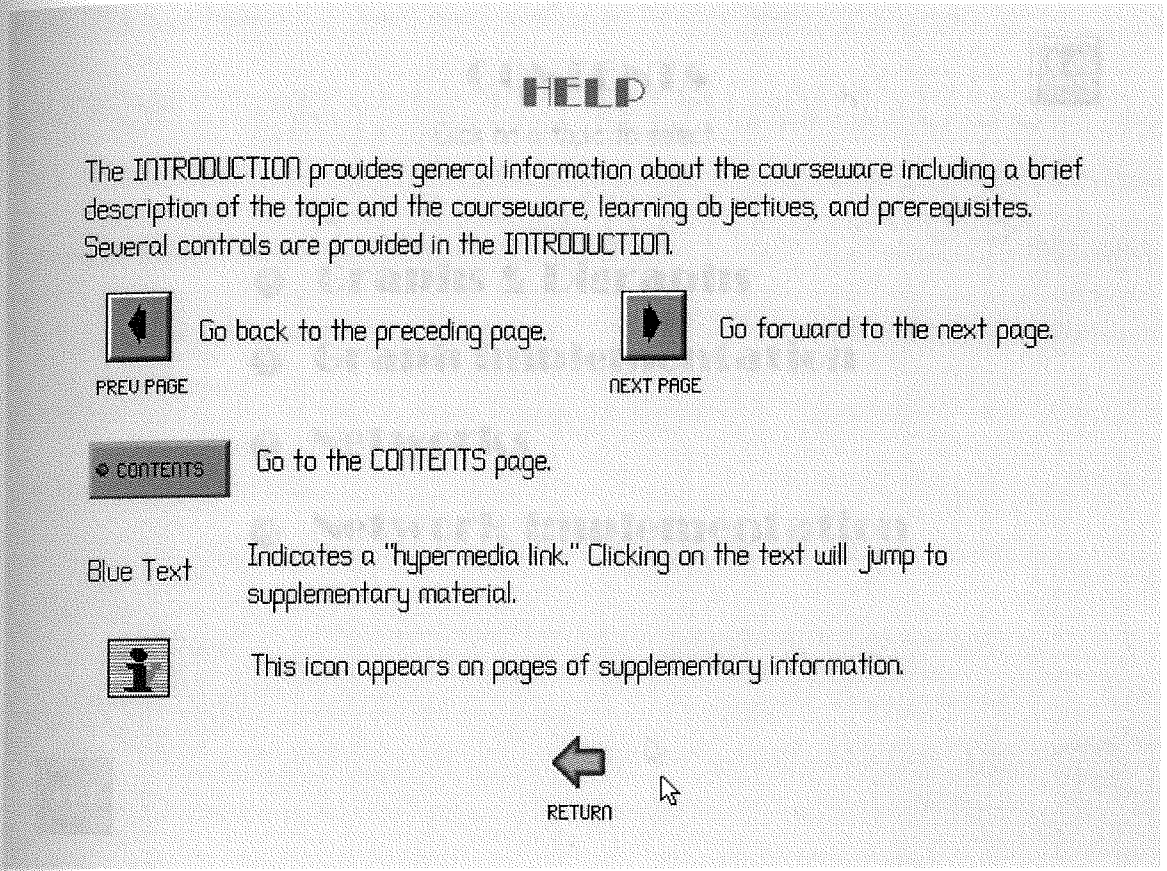
THE SUBJECT MATTER

Forty years before the beginning of the American Revolution Leonhard Euler developed the concepts of graphs and networks and applied them to the solution of the Koenigsburg Bridge problem. Since that time graphs and networks have been used to solve problems in transportation, communication, and many other areas.

In general a graph consists of entities and relationships among the entities; for example, cities and railroad lines between cities. Graphs are completely general with no limit on the interconnections among the entities represented. In the railroad example, each city can be connected to any number of other cities. Thus, graphs are nonlinear, non-hierarchical structures.

On each page of the Introduction, there is an icon at the bottom center of the screen labeled CONTENTS. Clicking on this icon will take the learner to the table of contents screen for the courseware. New users will probably want to read the entire Introduction; however, students who have used the courseware before can skip reading the Introduction by clicking on the CONTENTS icon.

Figure 16: Help Screen for Introduction


HELP

The INTRODUCTION provides general information about the courseware including a brief description of the topic and the courseware, learning objectives, and prerequisites. Several controls are provided in the INTRODUCTION.

PREV PAGE — Go back to the preceding page.

NEXT PAGE — Go forward to the next page.

CONTENTS — Go to the CONTENTS page.

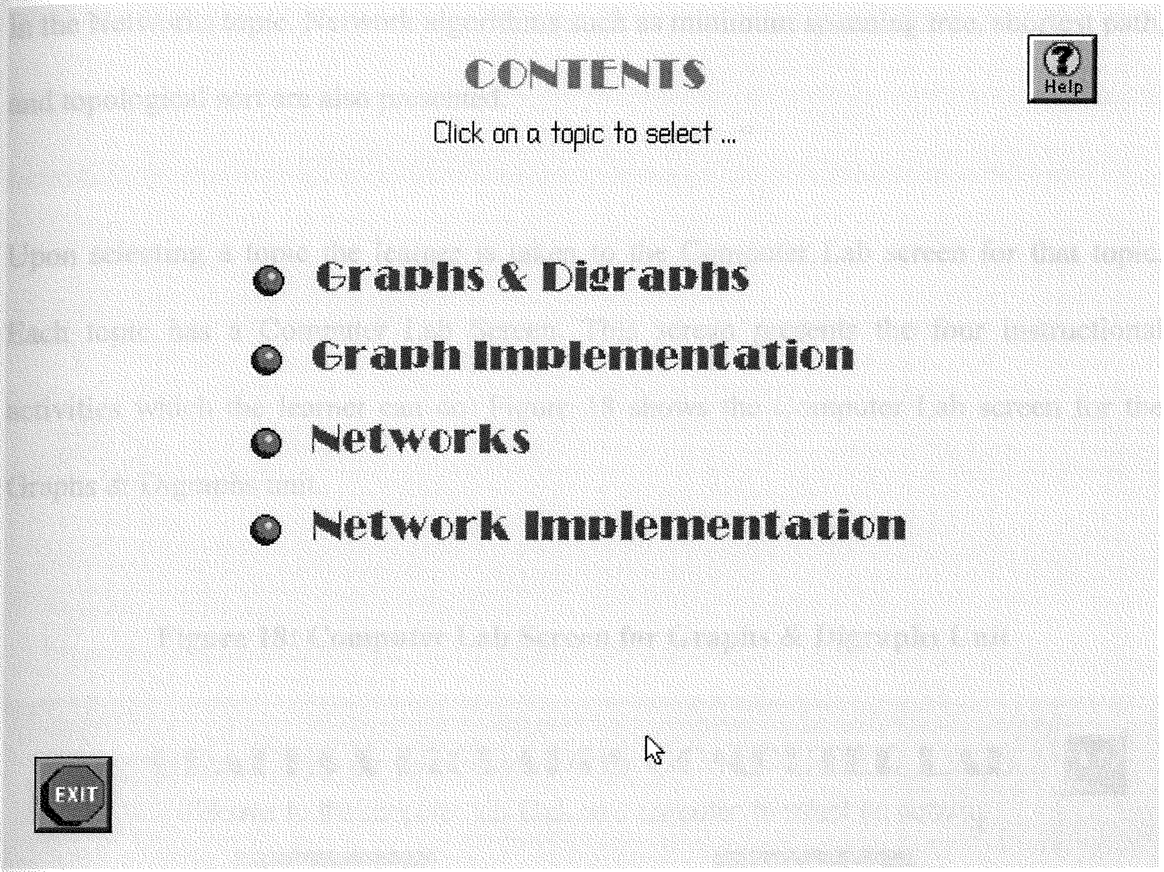
Blue Text — Indicates a "hypermedia link." Clicking on the text will jump to supplementary material.

This icon appears on pages of supplementary information.

RETURN

The table of contents page acts as the real starting point of the courseware. From this page the learner can select the topic to be studied. Upon completion of a topic the learner returns to the table of contents page, and each session is ended by returning to the table of contents page to exit. Figure 17 shows the table of contents page for the Graphs & Networks courseware.

Figure 17: Contents Page Showing Four Topical Units

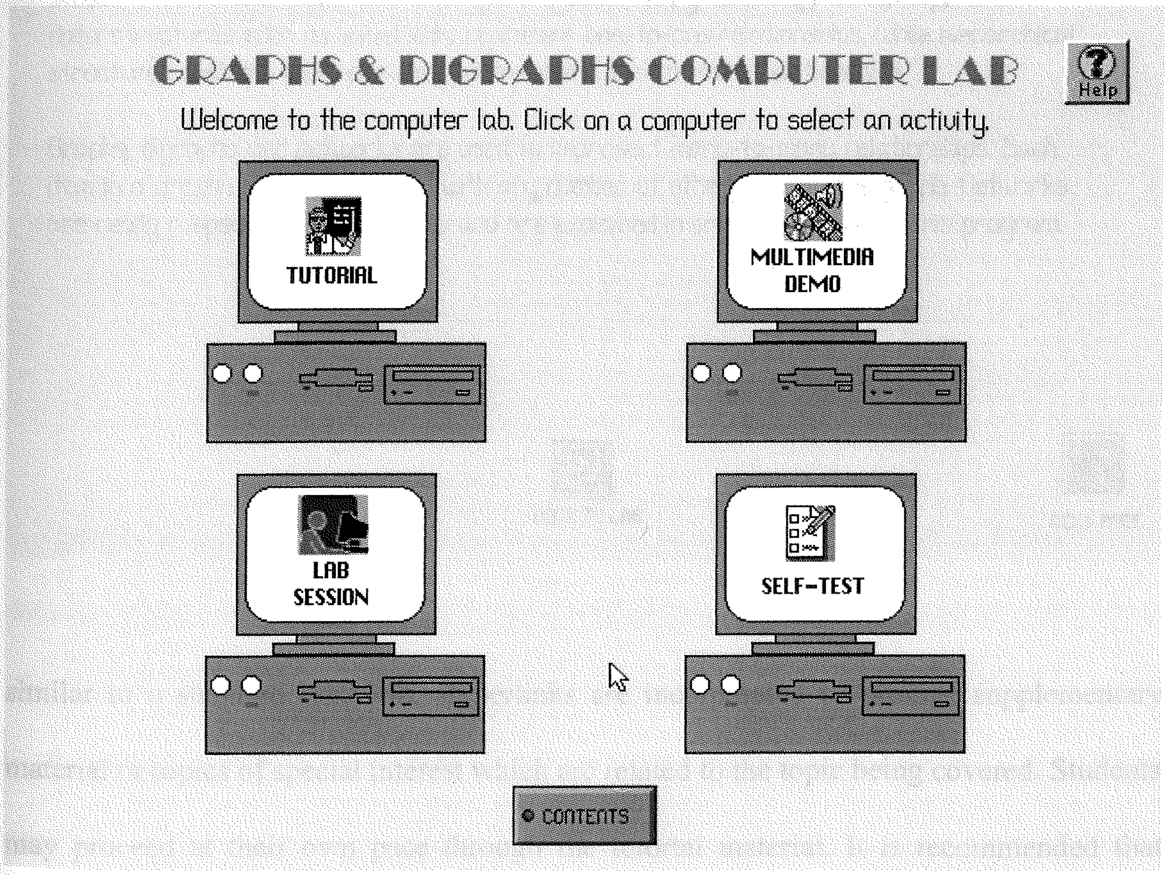


The learner can select a specific topic from the table of contents page. The first topic, Graphs & Digraphs, covers basic concepts and terminology related to graphs. Common graph algorithms, including depth-first traversal and breadth-first traversal are described and explained. Graph Implementation presents the concepts related to the data structures used to implement graphs, in particular the adjacency matrix. Sample Pascal coding for declaring the data structures is given.

Basic concepts related to networks and how networks are related to graphs are presented in the Networks topic. Network algorithms such as minimum spanning tree, shortest path, and topological sort are also presented.

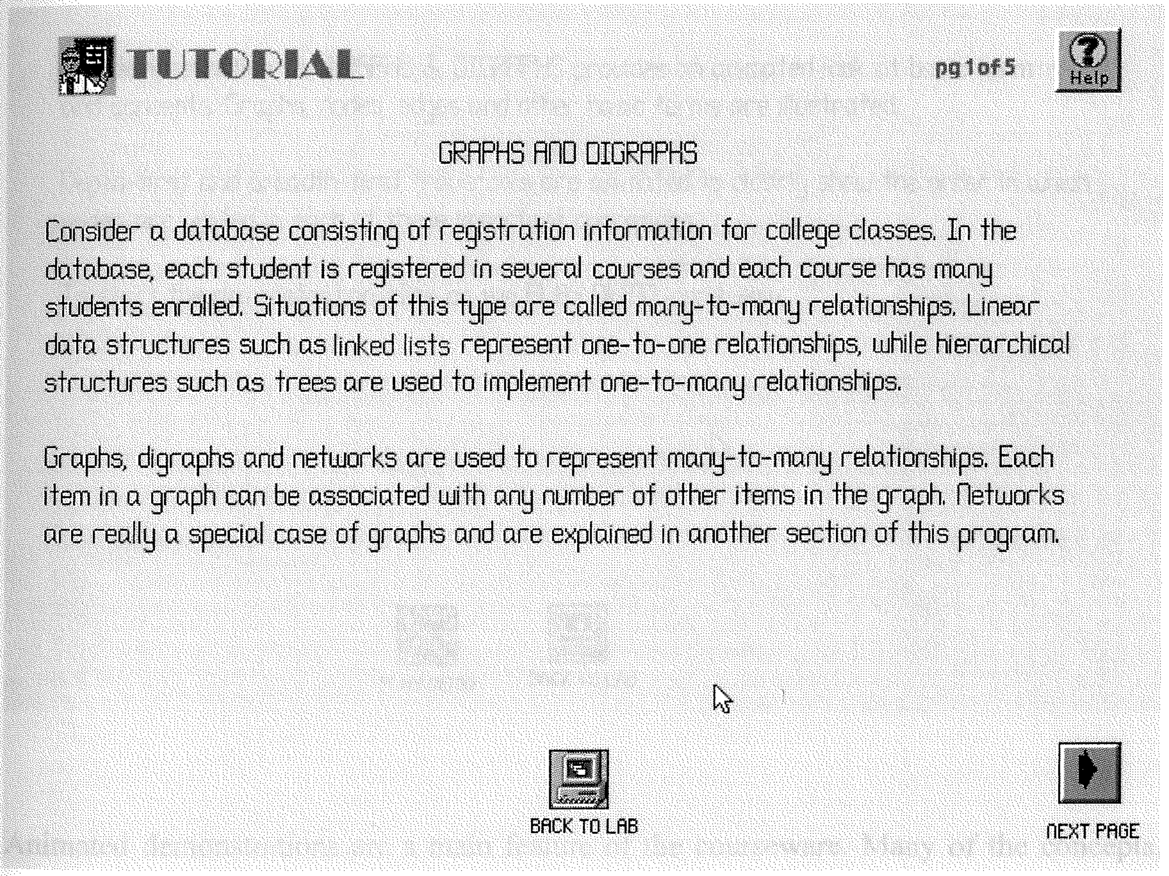
Upon selecting a topic the learner is taken to the Computer Lab screen for that topic. Each topic has a Computer Lab Screen. This screen presents the four instructional activities which the learner can do. Figure 18 shows the Computer Lab screen for the Graphs & Digraphs unit.

Figure 18: Computer Lab Screen for Graphs & Digraphs Unit


GRAPHS & DIGRAPHS COMPUTER LAB
Help
Welcome to the computer lab. Click on a computer to select an activity.
TUTORIAL
MULTIMEDIA DEMO
LAB SESSION
SELF-TEST
CONTENTS

From the Computer Lab screen, the learner can select tutorial, animated demonstration, laboratory session, or self-test. The tutorials provide the most basic instruction. Concepts, terminology, and examples are provided using text and graphic images in a manner

Figure 19: First Page of Graphs & Digraphs Tutorial



GRAPHS AND DIGRAPHS

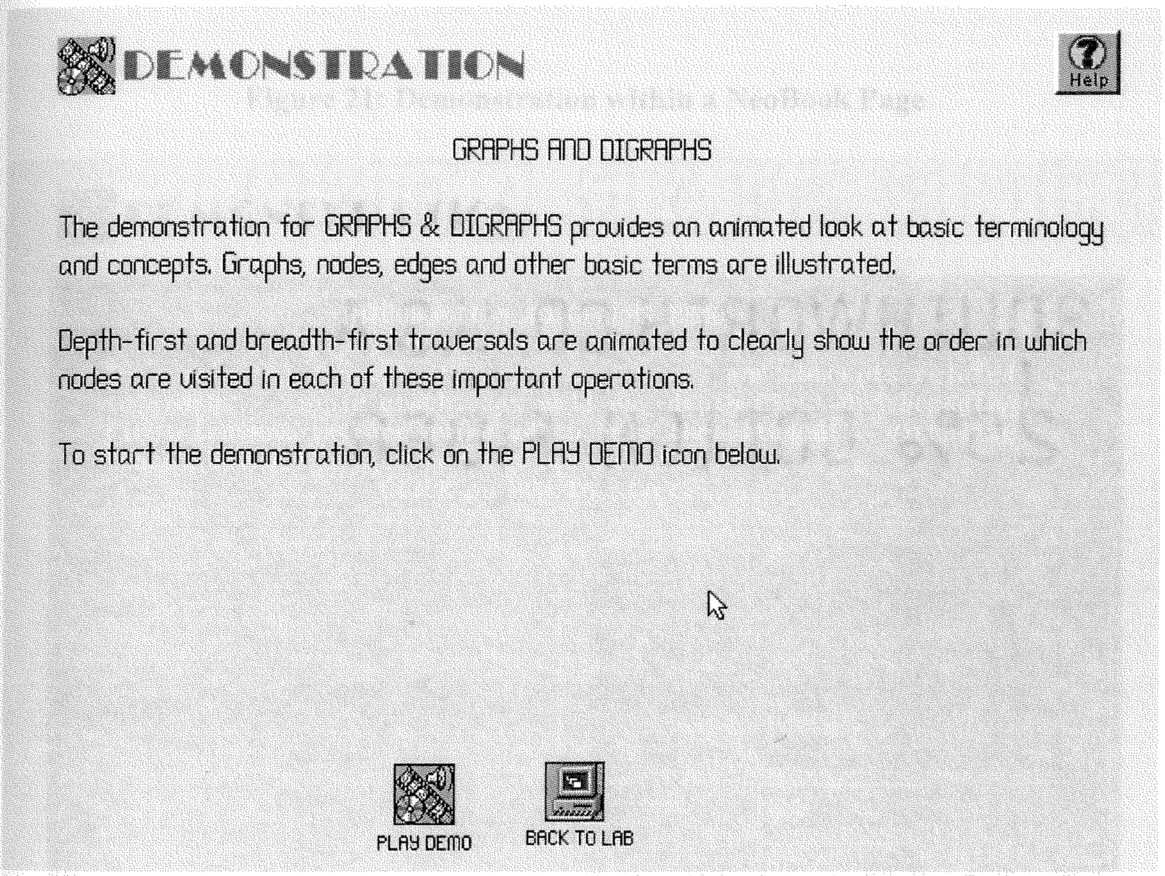
Consider a database consisting of registration information for college classes. In the database, each student is registered in several courses and each course has many students enrolled. Situations of this type are called many-to-many relationships. Linear data structures such as linked lists represent one-to-one relationships, while hierarchical structures such as trees are used to implement one-to-many relationships.

Graphs, digraphs and networks are used to represent many-to-many relationships. Each item in a graph can be associated with any number of other items in the graph. Networks are really a special case of graphs and are explained in another section of this program.

similar to a standard textbook. Hyperlinks are incorporated to provide supplementary material or topics of special interest which are related to the topic being covered. Students may proceed at their own pace through the tutorial material. It is recommended that students view the tutorial for a unit before going on to one of the other activities;

however, this is not strictly necessary. Figure 19 shows the first page of the tutorial for the Graphs & Digraphs unit.

Figure 20: Demonstration Opening Page



GRAPHS AND DIGRAPHS

The demonstration for GRAPHS & DIGRAPHS provides an animated look at basic terminology and concepts. Graphs, nodes, edges and other basic terms are illustrated.

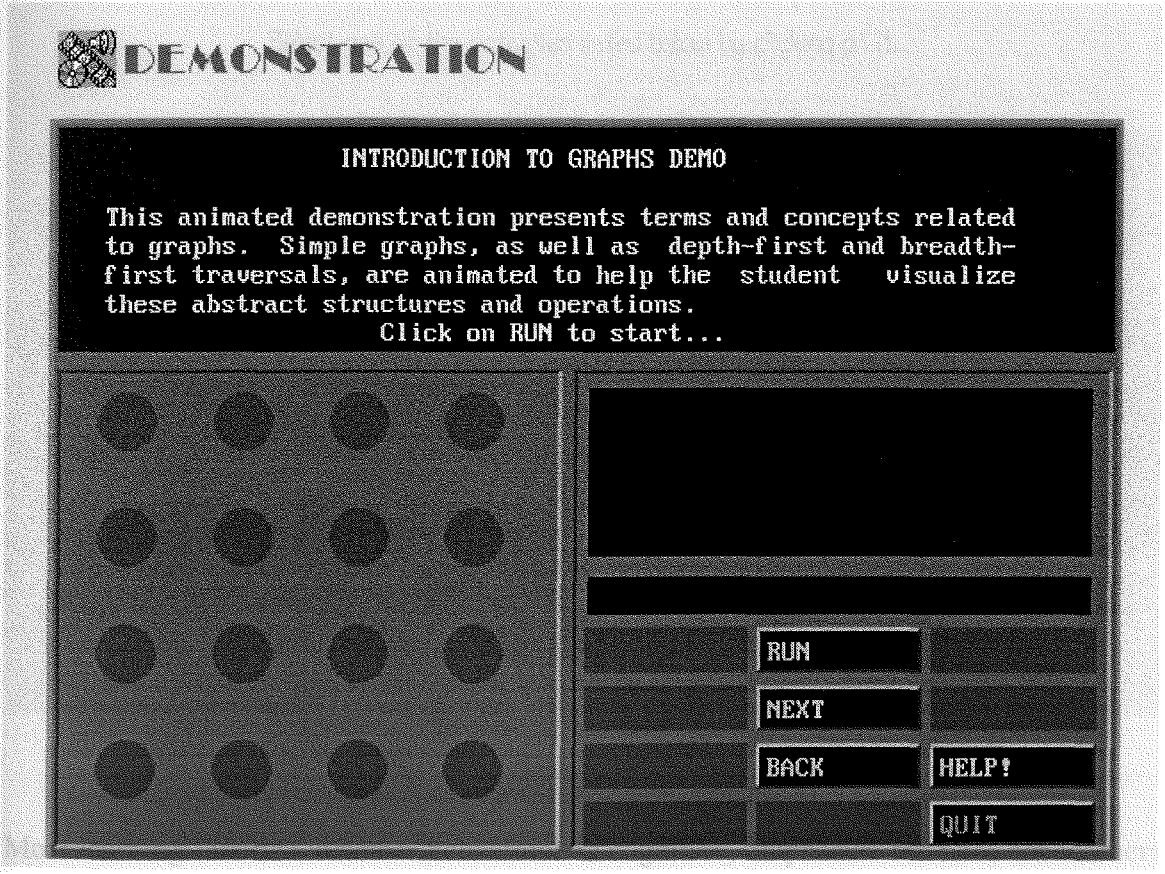
Depth-first and breadth-first traversals are animated to clearly show the order in which nodes are visited in each of these important operations.

To start the demonstration, click on the PLAY DEMO icon below.

Animated demonstrations are a main feature of the courseware. Many of the concepts, data structures, and algorithmic operations related to graphs and networks are difficult to visualize. Textbooks have traditionally used sequences of pictures with explanatory notes to illustrate these topics. The use of animation allows students to see the changes in a data structure resulting from the operation of an algorithm. Animations can be replayed as needed to study a specific topic.

When the learner selects an animated demonstration, the demonstration opening page shown in Figure 20 is presented. This page explains the purpose of the demonstration. The Play Demo icon at the bottom of the page is used to start the demonstration.

Figure 21: Demonstration within a NeoBook Page

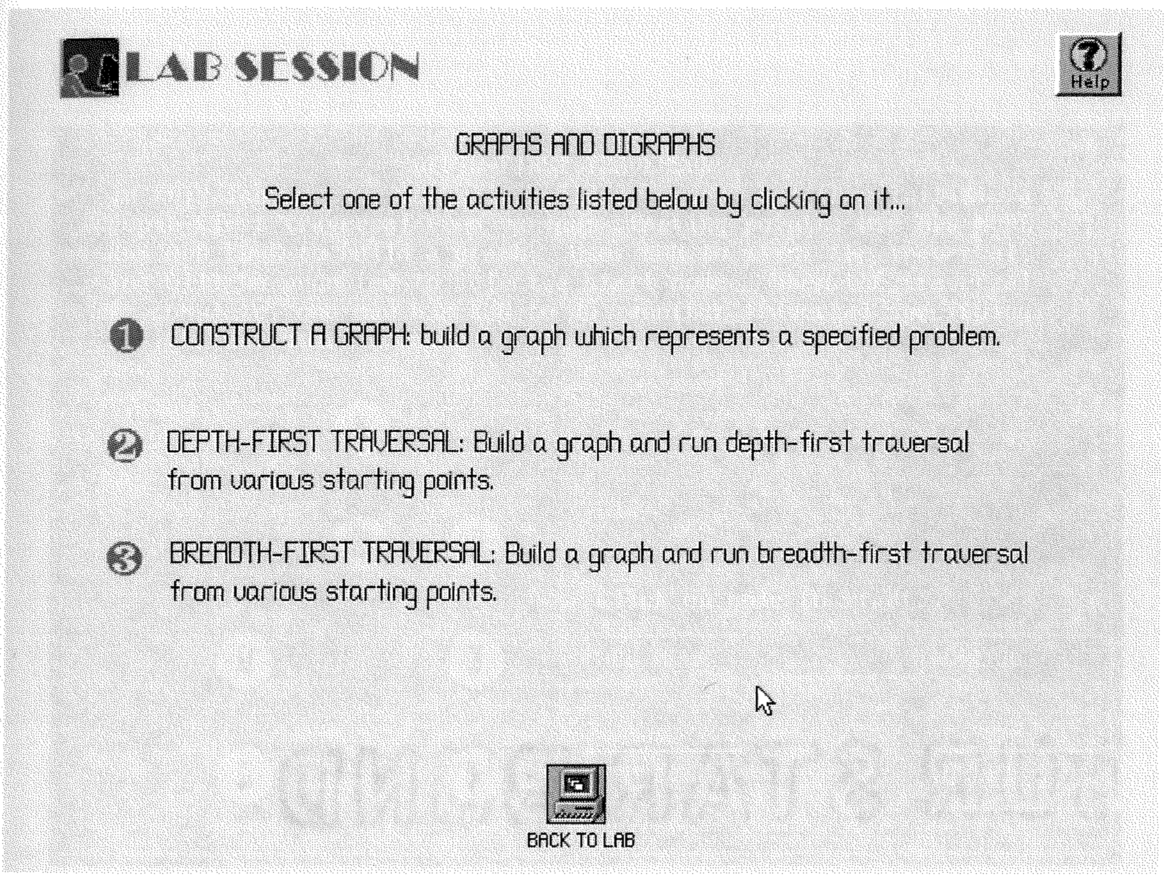


```
                INTRODUCTION TO GRAPHS DEMO

This animated demonstration presents terms and concepts related
to graphs.  Simple graphs, as well as  depth-first and breadth-
first traversals, are animated to help the  student   visualize
these abstract structures and operations.
                  Click on RUN to start...
```

The animated demonstration is implemented as a QuickBASIC 4.5 (1990) program. The output from this program is displayed in a portion of a demonstration screen presented by the NeoBook Professional (1994) publication. Figure 21 shows part of the animated demonstration for the Graphs & Digraphs unit as it is displayed within the demonstration page.

Figure 22: Lab Session Menu Page


LAB SESSION

Help

GRAPHS AND DIGRAPHS

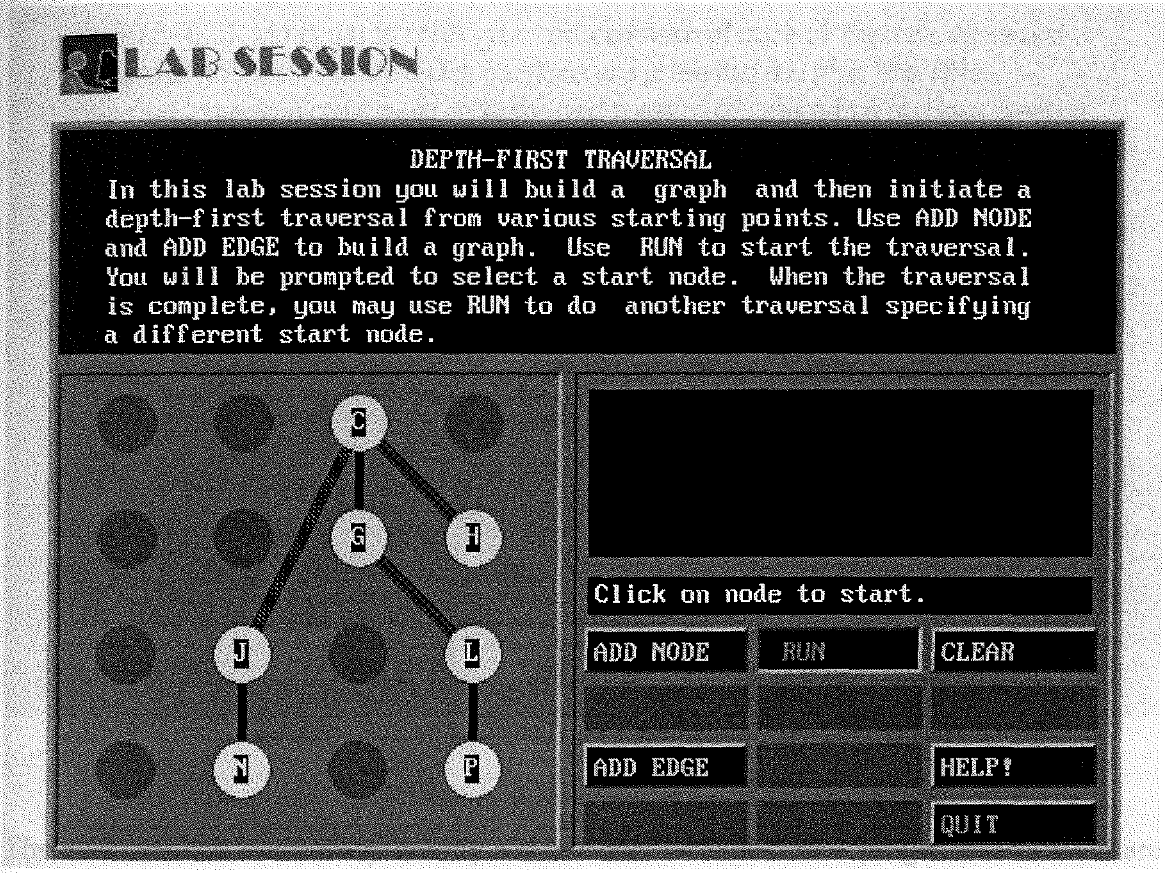
Select one of the activities listed below by clicking on it...

1. CONSTRUCT A GRAPH: build a graph which represents a specified problem.
2. DEPTH-FIRST TRAVERSAL: Build a graph and run depth-first traversal from various starting points.
3. BREADTH-FIRST TRAVERSAL: Build a graph and run breadth-first traversal from various starting points.

BACK TO LAB

Most learning theories, whether behavioral or cognitive, emphasize the need for learners to perform practical exercises to reinforce and clarify concepts and build skills. The interactive lab sessions in the courseware are intended to provide this type of learning activity. Students perform laboratory activities to solve specified problems or demonstrate phenomena. The activities are interactive. For example, one lab session in the first unit, Graphs & Digraphs, asks the student to build a graph of routes for a airline. Using a point-and-click interface the student adds edges to create a graph on screen.

Figure 23: Lab Session for Depth-First Traversal



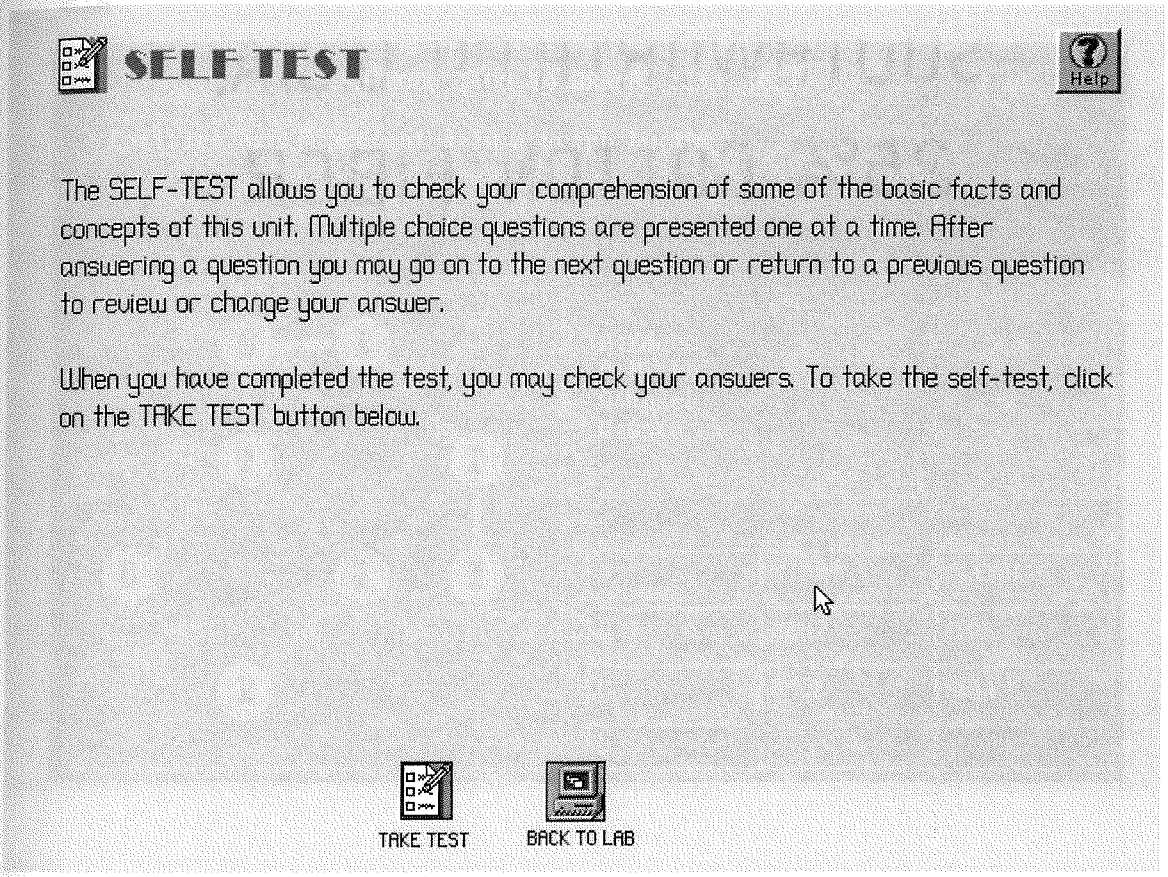
LAB SESSION

DEPTH-FIRST TRAVERSAL
In this lab session you will build a graph and then initiate a depth-first traversal from various starting points. Use ADD NODE and ADD EDGE to build a graph. Use RUN to start the traversal. You will be prompted to select a start node. When the traversal is complete, you may use RUN to do another traversal specifying a different start node.

Click on node to start.

Like the animated demonstration, a laboratory session begins with an opening page, and proceeds to a laboratory session page with a display from the QuickBASIC program running within a NeoBook Professional display. Figure 22 shows the laboratory session menu page for the Graphs & Digraphs unit, while Figure 23 presents a portion of the laboratory session for the depth-first traversal.

Figure 24: Self-Test Opening Page

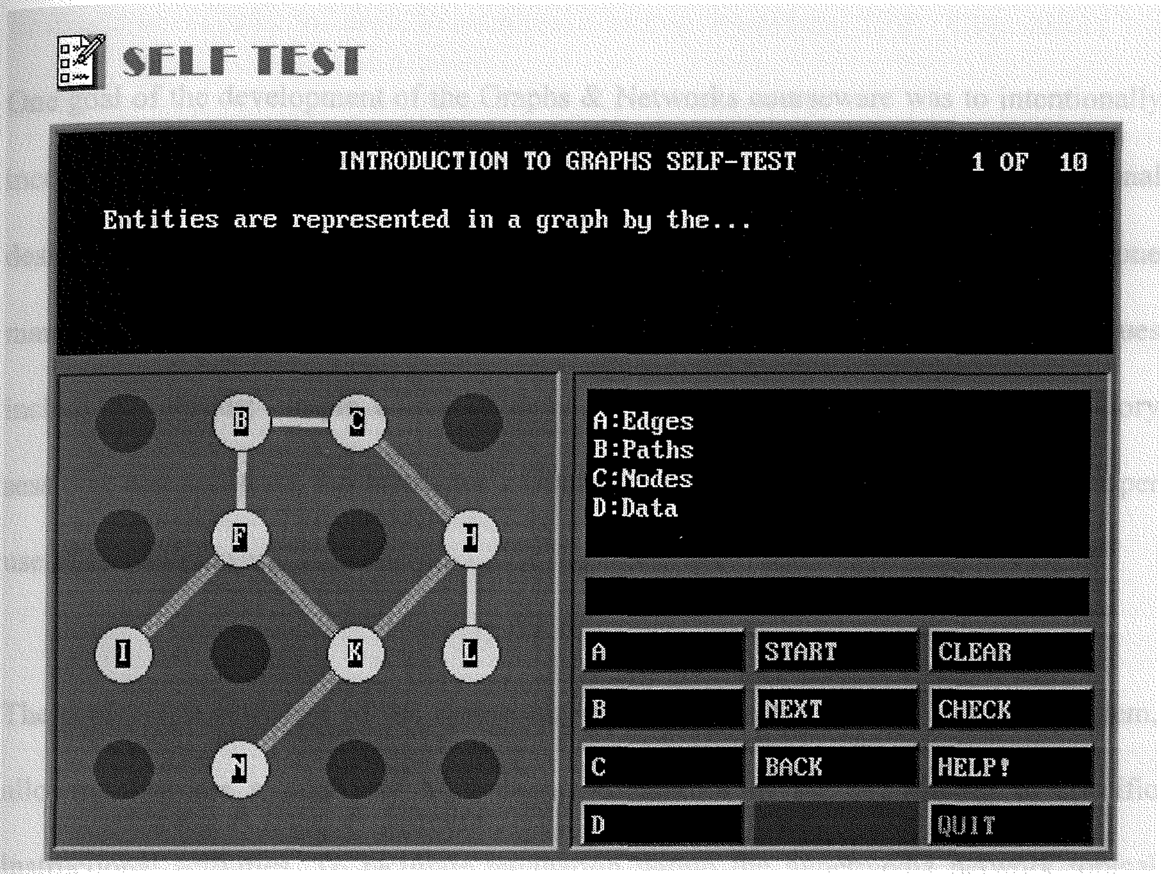


The SELF-TEST allows you to check your comprehension of some of the basic facts and concepts of this unit. Multiple choice questions are presented one at a time. After answering a question you may go on to the next question or return to a previous question to review or change your answer.

When you have completed the test, you may check your answers. To take the self-test, click on the TAKE TEST button below.

The self-tests are included primarily for students to check their progress. The tests are interactive and consist of multiple-choice questions. Students can review answers and change them as needed. Answers can be checked by the software. Incorrect answers are indicated and correct answers are displayed. Figure 24 shows the self-test opening page for the Graphs & Digraphs unit. A display of part of the self-test is shown in Figure 25.

Figure 25: Self-Test for Graphs & Digraphs



```
INTRODUCTION TO GRAPHS SELF-TEST          1 OF  10

Entities are represented in a graph by the...

A:Edges
B:Paths
C:Nodes
D:Data
```

Program Listing and Data Files

The animated demonstrations, interactive laboratory sessions, and self-tests were implemented using QuickBASIC 4.5 (1990). A general program which provides the interactive user display is customized for a particular session using a specified data file. **Appendix F** includes the program listing of the QuickBASIC program, ANILAB11.BAS. **Appendix G** contains listings of the data files used to specify the demonstrations, laboratory sessions, and self-tests.

Cognitive Learning Theories and Instructional Design Considerations

One goal of the development of the Graphs & Networks courseware was to intentionally incorporate features which take account of cognitive learning theories and instructional design considerations. While the courseware is not intended to be used in a stand-alone manner, an attempt was made to incorporate a variety of instructional techniques including tutorials; simple animated demonstrations; interactive, animated laboratory sessions; and self-tests. An Instructor's Guide is included to provide guidance on proper use of the courseware.

The topics and activities of the courseware are implemented as a hypermedia system, allowing learners a relatively high degree of control in the sequencing of specific instructional activities. To facilitate navigation within the hypermedia network topical units and instructional activities act as hyperlinks. Common controls such as requesting help, moving sequentially through a series of pages, and initiating an activity such as an animated demonstration are placed consistently on pages. The table of contents page acts as a "home page" from which the learner can select a topic and to which the learner can return when finished with a topic.

While the learner has a relatively high degree of user control, the courseware suggests an optimum sequencing. Learners who require a greater degree of structure can follow the recommended sequence. Screens have a similar appearance and use a small palette of colors to reduce distraction and cognitive overload. Even screens displayed by the

QuickBASIC 4.5 (1990) program are embedded in basic pages created with NeoBook Professional (1994) to provide continuity and a consistent look. User errors are minimized by providing mouse-selectable options.

Simple animation is used to make visible the dynamic operations of algorithms on the abstract structures of graphs and networks. Sound and color are used to convey information such as indicating which nodes in a graph have been visited during a traversal. Users can create structures interactively and experiment with the effects of various algorithms on these structures.

Courseware Evaluation

It is a characteristic of the software development process to conduct evaluations throughout the development process. A preliminary version of the software was reviewed by the dissertation committee in January along with the dissertation proposal. The primary feedback related to the need for detailed design specifications. These were developed and sent to the committee in February 1997.

Computer Science faculty at Mount Vernon Nazarene College also reviewed the software and made recommendations for changes and corrections. These included a minor content correction, and several operational issues. During implementation, the developer constantly ran tests, noted bugs or suggestions for improvements in a running log, and implemented modifications as needed.

Student/Faculty Evaluation of Completed Courseware

Following completion of the courseware, the software was evaluated by a small group of faculty and students at Mount Vernon Nazarene College. Two Computer Science faculty and six students, all of whom were Junior or Senior Computer Science majors, used the software for several hours and completed a questionnaire. The questionnaire was designed to measure the usability of the courseware and the users' attitudes towards the courseware. The evaluation was not intended to be an experimental study. It did not measure changes in achievement for students using the software. **Appendix B** contains a copy of the courseware evaluation form.

Overall response to the courseware was favorable. Questions 4 through 25 which had to do with operational and other features of the courseware all used a semantic differential scale to measure attitudes using bipolar adjectives such as "hard" versus "easy" and "very cluttered" versus "not cluttered." In all cases a rating of 1 represented an undesirable evaluation such as "hard" and a 5 represented a desirable evaluation, such as "easy." Table 14 shows the results of the evaluation by students and faculty. The last column of the table, Rating, indicates the weighted average of the evaluations for each question.

Table 14: Results of Student/Faculty Courseware Evaluation

Question	1	2	3	4	5	Rating
4. Rate the overall ease of use				6	2	4.250
5. Rate the overall appearance				7	1	4.125
6. How easy is it to navigate, e.g. to go from one activity or topic to another?				3	5	4.625
7. How easy is it to find the specific activity or topic you want, such as a demonstration or lab exercise?				4	4	4.500
8. Screens are ... (Easy to Read)			1	4	3	4.000
9. Screen layouts are ...(Not Cluttered)			1	4	3	4.000
10. Use of color, graphics and sound are ... (Helpful)			1	4	2	4.143
11. Content is accurate				1	7	4.875
12. Content has educational value				1	7	4.875
13. Prerequisites are clear			1	4	3	4.000
14. Purpose of courseware is clear				4	4	4.500
15. Courseware achieves its goal				1	7	4.875
16. Level of difficulty is appropriate for intended audience.				4	4	4.500
17. The courseware increased my motivation			3	1	3	4.000
18. Learners receive appropriate feedback			1	5	2	4.125
19. Learners have appropriate degree of control			2	2	4	4.250
20. Tutorials are helpful				3	5	4.625
21. Animated demonstrations are helpful				3	5	4.625
22. Interactive laboratory sessions are helpful				2	6	4.750
23. Self-tests are helpful				4	4	4.500
24. Supplementary materials are helpful			1	2	3	4.333
25. The courseware helped my learning			2	3	2	4.000

A number of conclusions can be derived from the results of the evaluation. These are discussed in **Chapter V Conclusions**.

In addition to the numerical results of the evaluation, most of the respondents included comments in the area provided on the courseware evaluation form. Comments included suggestions for modifications, corrections, and general reactions. Some of these were used to make immediate modifications to the courseware, while others were noted and are presented as suggestions for further work in **Chapter V Conclusions**.

Three items noted by several of the evaluators were considered serious enough to warrant software modification. A number of students and both faculty members raised a question related to the order of node selection during a traversal. In particular, they indicated that while the tutorial for Graphs & Digraphs showed the order of traversal, it did not explain why a given node was selected over another node when two nodes were adjacent to a parent node. The tutorial was modified to give a brief explanation and a further explanation was given in the Graph Implementation tutorial.

Another concern raised by several evaluators was that, on their machine, the check feature of the self-test ran through the questions too quickly. It was decided to modify the check feature to apply to the current question only. This not only solved the speed problem, but allows the user to check questions individually.

Finally, a question of content accuracy was raised by the two Computer Science faculty who evaluated the software. The first question of the self-test for Graphs & Digraphs originally stated "Information is represented in a graph by..." The correct answer was supposed to be "Nodes." The faculty members argued that both nodes and edges can be thought of as representing information. The edges, in particular, contain information about the relationships among nodes. The question was changed to read "Entities" instead of "Information."

Chapter V

Conclusion

Technology is a valuable tool which can be used to enhance teaching and learning. In particular, computer technology, with the seemingly endless variety of applications possible, promises to bring richness to the educational experience both in terms of content and in terms of supporting higher level learning. However, development of effective instructional software requires use of proven methodologies such as software engineering and instructional design, application of cognitive learning theories, and selection of proper development tools. The results described in the preceding chapter demonstrate the success which can be achieved from following these practices.

Conclusions

In drawing conclusions from the results of this development effort one should start with the evaluation of the courseware by Computer Science students and faculty. The attitudes and assessments of this group are important since these users represent the group of intended users. If student learning is to be enhanced, students must find the courseware helpful and easy to use. If faculty are to recommend the software to their students and use it in the classroom, they must consider the content to be accurate and the instructional activities to be helpful and supportive of learning.

Faculty/Student Evaluation

Table 14 in **Chapter IV Results** presents the ratings of the Graphs & Networks courseware by Computer Science students and faculty. A number of conclusions can be drawn by examining the data in this table.

First, the overall reaction of the evaluators was favorable. All items pertaining to ease of use and the effectiveness of the courseware were evaluated on a scale of one to five with one being the lowest (poorest) rating, and five being the highest (best) rating. Every item evaluated scored 4.000 or higher, the overall average rating being 4.385 out of 5.000. In all, 158 of the 171 total responses to questions 4 through 25 (92.4%) were either four or five.

With regard to the user interface, the evaluators rated the overall ease of use, appearance, ease of navigation, and ease of finding a particular activity high, the lowest rating being 4.125. Evaluators felt that screens were easy to read, not cluttered, and that the use of color, graphics, and sound were helpful.

The evaluation with regard to instructional effectiveness was also high. Evaluators rated the content and the educational value of the content very high (4.875). Similarly, evaluators indicated that the purpose of the courseware was clear (4.500), the prerequisites were clear (4.000) and that the courseware achieved its goal (4.875).

All instructional activities were rated as helpful. Despite the fact that a major emphasis of the courseware was on interaction and animation, even the static tutorials received a high rating (4.625). The animated demonstrations and interactive laboratory sessions were rated high with ratings of 4.625 and 4.750 respectively. Generally, students indicated that the courseware increased their motivation and helped their learning.

Students indicated in comments that the animations and interactive laboratories facilitated understanding. One student noted that, "The multimedia demonstration is very good" and that the "main menu with four options is very user friendly." A student who has not yet taken the advanced data structures course in which graphs and networks are presented stated, "This will be very helpful during the actual class." Another commented, "The real life examples and images were very helpful" and "I think it [the courseware] would help me in a Data Structures course."

Development Methodology

The methodology used in this courseware development was based on a conventional software development model incorporating instructional design principles and techniques, and guided by applicable cognitive learning theories. The products of the development were the functional specification, design specification, the courseware itself, and an Instructor's Guide. These items underwent evaluation throughout the development as well as evaluation by a small group of Computer Science students and faculty representative of the target audience. The fact that the products of the development were completed successfully, within the planned time frame, and were evaluated positively by users attests to the success of the methodology.

As described in **Chapter III Methodology** the development was planned to be carried out in specific phases including (1) requirements analysis, (2) specification, (3) planning, (4) design, and (5) implementation. The time required for the overall development and for each phase was estimated using the COCOMO model and proven estimation techniques and is given in Table 13 in **Chapter IV Results**. As shown in this table, the actual time spent was fairly close to the estimated time. The two greatest deviations from the estimated time were the requirements analysis and the design phases.

The longer time for the requirements analysis is attributed to the in-depth research needed to support the problem description and objectives for the dissertation process. In a typical application development, rightly or wrongly, requirements are usually analyzed for the local organization, not in terms of a more universal problem. The analogy would have been to focus only on the immediate learning objectives and needs of Mount Vernon Nazarene College Computer Science majors rather than on Computer Science majors in general. Investigating and describing the more general problem took a longer time.

The lower than projected time for design was a result of the selection of development tools, specifically, the use of NeoBook Professional (1994), a point-and-click authoring system, to develop the main user interfaces and overall application structure. The interactive nature of screen development made it unnecessary to specify exact locations of items on a screen. Instead, items were located and sized interactively during screen development.

The use of NeoBook Professional also contributed to an overall lower development time. The ease of creating and modifying screens, and of including buttons, graphic images and other items, reduced the time needed to do implementation of these parts of the courseware. Also, duplicating individual items and entire pages, easily done within the NeoBook Professional environment, simplified development and fostered consistency within the application.

Impact of Cognitive Learning Theories

The courseware design incorporated several key concepts derived from cognitive learning theories, including (1) the structuring of the courseware as a hypermedia system, (2) including the table of contents as a specific root node of the hypermedia system, (3) providing appropriate navigational controls, (4) incorporating online help, a recommended sequence of activities, mouse-selectable options, and appropriate feedback, (5) designing screens with a consistent, uncluttered format, a limited palette of colors, labeled controls, and a familiar metaphor for selecting instructional activities, and (6) the use of color, animation, and sound to draw attention to and emphasize important features.

Evaluations by faculty and students indicated that these features were effective. In particular, evaluators indicated that it was easy to navigate from one activity to another and that it was easy to find a specific activity or topic. They also agreed that the screens were easy to read, were not cluttered, and that the use of color, graphics, and sound were helpful. The helpfulness of the animated demonstrations and interactive laboratory sessions was rated very high. The use of a hypermedia structure was intended to provide freedom in learner control, particularly with regard to sequencing of topics and activities. The evaluation group indicated that learner control was appropriate.

Implications

The successful development of the Graphs & Networks courseware has implications for faculty development of instructional software and for the effectiveness of instructional software.

Faculty Development of Instructional Software

In **Chapter I Introduction** it was noted that the development of multimedia courseware requires multiple skills and tools and that development is time-consuming. The development of the Graphs & Networks courseware bears out these contentions, but also points to ways of reducing development demands.

For faculty to develop instructional software effectively and efficiently, they need to be aware of current learning theories and the potential of educational technology, as well as being knowledgeable in their content area. Teacher education programs for education majors, as well as in-service training, workshops, and continuing education courses for current faculty, should provide educational opportunities in the areas of learning theory, authoring, instructional design, and courseware development. Training in the use and effective application of specific design and development tools should also be provided on a regular basis.

Faculty who are interested in and capable of carrying out effective courseware development should be given time and encouragement to do so. Despite the power and ease of use of design and development tools, it still takes a significant amount of time to develop high quality, educationally effective instructional software. Likewise, faculty need access to appropriate tools such as interactive authoring systems, graphics, sound, and video editors, and computer systems capable of running these tools efficiently.

Effectiveness of Instructional Software

Several concerns regarding the lack of effective instructional software and the difficulties of learning abstract data structures and operations in Computer Science were identified at the early stages of the development effort. While there may be a proper place for drill-and-practice programmed instruction, this type of educational application neither utilizes the full capability of computer technology nor supports higher level learning activities such as analysis, synthesis, evaluation, and application.

Instructional software which consciously incorporates instructional design principles and applications of cognitive learning theories does provide higher level learning. The appropriate use of a hypermedia structure as well as the use of sound and animation can facilitate learning for most learners. The use of animation in the Graphs & Networks courseware made it easier to understand the way in which graph and network algorithms work. Evaluators of the software indicated that the animated demonstrations and interactive laboratory sessions were helpful and that they would be helpful in the classroom.

An important aspect of understanding complex or abstract structures and operations is being able to experiment with them and observe the results of various actions or decisions. The interactive, animated laboratory sessions provided in the Graphs & Networks courseware allow the learner to create structures, run algorithms, and observe the results. Within the constraints of the user display, the learner is free to create graph and network structures of their choosing. Algorithms may be run and rerun to promote a thorough understanding of their operation.

In addition to the direct educational benefits, evaluators indicated that the Graphs & Networks courseware increased their motivation to learn the content.

Recommendations

The recommendations related to this development fall into four broad categories: (1) those affecting professional practice in the development of instructional software, (2) those related to the use of the Graphs & Networks courseware, and (3) those related to enhancements and modifications of the courseware, and (4) further research using the Graphs & Networks courseware.

Recommendations for Development of Instructional Software

The methodology used in this development effort should serve as a model for the development of any significant instructional software application. While the main features of the methodology are not new, the integration of software development and instructional design techniques, and the application of relevant cognitive learning theories into a unified approach is unique. It is vital that the courseware developer carry out the proven development phases of software engineering: (1) requirements analysis, (2) specification, (3) planning, (4) design, and (5) implementation. Evaluation should be conducted as needed during each phase of development. Appropriate documents such as the functional specification, design specification, and user's guides should be produced as an integral part of the development.

Instructional design techniques such as defining learning objectives, performing task analysis, and defining learner characteristics must be incorporated into the software development methodology to customize the process for educational applications.

The design must take into account cognitive theories of learning and incorporate design features and an overall structure which support and facilitate learning. The overall structure should be that of a hypermedia network. Learner control options can be constrained if necessary, but should be left as open as possible to allow for different abilities and learning styles. Media other than static text and graphic images, such as sound and animation, should be incorporated to clarify complex structures and processes, or to draw attention to or emphasize certain aspects of the material.

Displays should be uncluttered, use few colors, and should be consistent with one another to avoid distraction or cognitive overload. Transitions between screens should be smooth with continuity of format or content also to minimize the possibility of cognitive overload on the part of the learner.

Finally, appropriate tools should be used to facilitate development. In particular, a powerful, easy-to-use authoring system should be used to implement at least the primary user displays and the overall application structure. Scripts or programs written with a traditional programming language such as BASIC, Pascal, or C, can be used to supplement the features of the authoring system if necessary.

Recommendations for Using the Graphs & Networks Courseware

The Graphs & Networks courseware was designed to be used in support of normal instructional activities including lectures and programming assignments. The courseware is not a stand-alone application. Within this context, the courseware can be used effectively in several ways.

First, the courseware can be used in the classroom to supplement lectures. The animated demonstrations and interactive laboratory sessions can be used to illustrate dynamic operations. The displays can be projected using a high-intensity overhead projector equipped with an LCD display panel. The self-tests can be used in class to check class comprehension or review for a test or quiz.

Second, students having difficulty with graph and network material can use the courseware for supplemental work. The tutorials and animated demonstrations can be used to provide another exposure to some of the content. The student can do the laboratory exercises to gain a better understanding, then check understanding using the self-tests.

Finally, the specific activities in the courseware can be assigned to all students to be completed on their own or during scheduled laboratory periods. The instructor could develop a laboratory guide with specific structures and problems for the students to create interactively and experiment with.

Recommendations for Enhancements and Modifications of the Courseware

The software engineering process is often referred to as the software development cycle or the software life cycle. The point is that the process is cyclic or iterative in nature.

Even when the product reaches the final phase, implementation, the discovery of previously-undiscovered bugs and ideas for additional features, or enhancements, causes the cycle to be repeated beginning at the requirements analysis phase. Several possible modifications and enhancements of the Graphs & Networks courseware are described in the following paragraphs.

First, there are other topics in Computer Science pertaining to data structures and algorithms which could be easily added to the application. Other linked structures such as linked lists and trees would be possible candidates for additional topical units. Especially in the case of trees, animated demonstrations of related algorithms such as the inorder, preorder, and postorder traversal of binary trees, and even adding nodes to and deleting nodes from binary search trees would be helpful to students.

Incorporation of new topics would be relatively easy given the modular design of the courseware. New screens, subroutines, and data files would need to be added with relatively little modification to existing software.

Second, additional features could be added to the courseware to enhance the instructional experience. Glossaries, search features, and other aids could be added to assist the student. Text input by students could be added to allow student note-taking, adding notes to existing topics and activities, essay questions on self-tests, and other constructivistic activities. Laboratory sessions could be enhanced by adding more problems and possibly by randomly selecting problems for students to do. The same could be done with self-tests, with questions being randomly selected from a pool of available questions.

Buttons could be added to the laboratory user display to allow nodes and edges to be removed as well as being added. The current version of the courseware only allows adding nodes and edges. The only way to remove an edge or a node is to clear the structure and start over.

In addition to enhancements, certain modifications might be desirable. The current version of the courseware is oriented around topics. It might be desirable to allow the laboratory sessions to be more general. For example, currently the user can select the depth-first traversal lab session in the Graphs & Digraphs unit. During the lab session the user can build various graph structures and run the depth-first traversal to observe the operation of the algorithm on these structures. To observe the operation of a different algorithm, for example the breadth-first traversal, on the same structure, the user must exit the lab session, select the breadth-first lab session, then reenter the structure and run the algorithm.

An alternative approach would be to have a general graph laboratory in which the user creates a graph structure then selects an algorithm to run. When the algorithm completes, a different algorithm could be selected for the same structure.

In the current version of the courseware there is no way to "back out" of the ADD NODE or ADD EDGE operation in an interactive laboratory session. The problem is not serious since the user can simply respecify an existing node or edge. However, adding the capability of canceling an operation would probably be a worthwhile modification.

Finally, directed graphs receive very little treatment in the current version. Additional material could be added to the Graphs & Digraphs tutorial, animated demonstration, laboratory session, and self-test to cover this subtopic more thoroughly.

Recommendations for Further Research

The primary goal of this development was to demonstrate a development methodology which can be used to produce effective, high-level instructional software efficiently. The courseware addresses the problem of learning abstract structures and operations. It was not a goal of the project to demonstrate experimentally that using the courseware would enhance student achievement.

Thus, one area of further work would be to conduct an experimental study which would investigate the effect of using the courseware on student achievement in learning the topics of graphs and networks.

More generally, the enhanced software development methodology could be used to develop other high-level instructional software and investigate the effect on achievement, motivation, or other factors related to student learning. The effect on learning of emphasizing or de-emphasizing various cognitive learning theory considerations such as constructivistic features or cognitive overload could be studied using modified versions of the courseware.

Finally, incorporating new tools and technology into the development process would produce interesting results. In particular, using HyperText Markup Language (HTML) to author the courseware as an internet or intranet application would facilitate delivery and increase accessibility. Features which could not be implemented directly using HTML could be incorporated using common gateway interface (cgi) programs in C, Perl or Java.

Summary

Computer Science, by its very nature, deals with abstractions. The algorithms and data structures used in computer programs are abstractions of real entities, relationships, and operations, and even the program coding is an abstraction of the underlying computer hardware. Because of this abstract nature of the discipline, Computer Science majors, particularly Sophomore and Junior students dealing with complex abstractions for the first, time find it difficult to understand the structures and operations they are studying.

Learning of complex abstractions can be facilitated by using appropriate instructional software; however, very little educational software exists at the college and university level, particularly in the area of Computer Science. In general, educational software tends to be aimed at lower levels of learning such as drill-and-practice, rather than at higher levels such as analysis, synthesis, evaluation, and application. Software which does exist is often ineffective because of poor design or improper use.

For instructional software to be effective it must be developed using the proven methodologies of software engineering and instructional design, and incorporate applicable considerations from cognitive learning theories. Selection and use of powerful and efficient development tools can reduce development time and effort.

Software engineering is a discipline which tends to change the activity of computer programming from a mysterious art to a rigorous science. The methodology is defined by a precise sequence of analysis, planning, design, and development phases, with evaluation and documentation being done throughout the process. The methodology emphasizes analysis, planning and design as a means of facilitating implementation and reducing the need for changes in the finished product. The process is considered to be iterative in nature allowing the product to be refined in a stepwise fashion.

Following the disciplined process of software engineering reduces overall development time, and produces software which is more reliable and easier to maintain. Incorporating techniques and principles from instructional design into the software development process customizes the process for educational applications. Instructional activities which address higher levels of learning can be incorporated by applying principles of cognitive learning theories.

Two primary difficulties arise in the development of high level instructional software. First, development requires a variety of skills and abilities including subject matter knowledge, familiarity with educational theory, knowledge of software engineering and instructional design methods, and proficiency with various development tools such authoring systems and programming languages. Second, development of courseware requires a significant amount of development time.

The first problem can be addressed by using teams of professionals to carry out development. Alternatively, faculty who wish to do development can be educated in the area of educational technology. In any case, the second factor, development time, can be reduced by following proper development procedures and using powerful, effective development tools. Authoring systems which require no programming and which offer appropriate development functions, design templates, and other design and development aids can reduce the difficulty and the time required to do development.

Development of the Graphs & Networks courseware utilized software engineering and instructional design techniques and incorporated applicable considerations from cognitive learning theory. An easy-to-use, yet relatively powerful authoring system, NeoBook Professional (1994) and a programming language, QuickBASIC 4.5 (1990), were used to carry out the development. NeoBook was used to create the main user displays, navigational controls, and overall courseware structure. Functions which could not be done easily in NeoBook were programmed in QuickBASIC and integrated into the NeoBook structure.

The courseware was organized as a hypermedia system allowing a high degree of learner control in sequencing of topics and instructional activities. For learners who need a greater degree of structure, a suggested sequence was provided. Simple animations were used to provide dynamic, visual presentations of the actions of graph and network algorithms on related data structures.

The development of the Graphs & Networks courseware was completed within the estimated time and produced all products specified including the functional specification, design specification, courseware, and Instructor's Guide. An important part of the specification phase was to define measurable learning objectives for the courseware. Instructional task analysis helped to define a hierarchy of instructional activities needed to meet the stated learning objectives and to identify prerequisite skills and knowledge assumed for the intended learners.

Evaluation of the courseware by a representative group of Computer Science faculty and students resulted in a overall favorable attitude and high ratings for the usability and instructional effectiveness of the courseware. In particular, the animated demonstrations and interactive laboratory sessions were viewed as very helpful.

The results of the project demonstrate the effectiveness of development which follows the proven techniques of software engineering and instructional design, and which incorporates applicable cognitive learning theory considerations. Also, the use of appropriate, powerful development tools is crucial to efficient development.

The Graphs & Networks software is designed to be used to supplement normal classroom lectures and programming assignments. The courseware can be used in class to demonstrate dynamically the operation of graph and network algorithms on related structures or outside of class in formal laboratories or self-study sessions.

Because of the modular design of the courseware, incorporation of other data structures and algorithms such as linked lists or binary trees could be easily accomplished, extending the usefulness of the courseware. Student aids such as glossaries, search features, and student input of notes, would enhance the educational value of the courseware.

Finally, it would be interesting to extend the research in several ways. Experimental studies should be conducted to determine quantitatively the degree to which this type of courseware enhances achievement, motivation, and other educational measures. Implementation using different platforms would extend the availability of the courseware. In particular, developing a version using HyperText Markup Language (HTML) for internet and intranet delivery would be useful.

Technology has always played an important role in education. Whenever new technology can be used to improve teaching and learning, it should be used to do so. The best parts of proven methods and new theories can be integrated to create effective instructional activities. Powerful tools can be used to facilitate development.

References

Abramson, G.W. (1993). Helping teachers create high-order, highly-motivating, hypermedia-based learning experiences, Part II: IBM Linkway. *SIGTC Connections*, *10*(1), 6-18.

Abramson, G.W. (1992). Learning objectives and lesson plans. Unpublished manuscript. William Paterson College of New Jersey.

Aho, A.V., & Ullman, J.D. (1992). *Foundations of computer science*. New York: W.H. Freeman and Company.

Alessi, S.M., & Trollip, S.R. (1991). *Computer-based instruction: Methods and development* (2nd ed.). Englewood Cliffs, NJ: Prentice-hall, Inc.

Ambrose, D. W. (1991). The effects of hypermedia on learning: A literature review. *Educational Technology*, *31*(12), 51-55.

Armstrong, T.C., & Loane, R.F. (1994). Educational software: A developer's perspective. *Tech Trends, 39*(1), 20-22.

Aristotle (1968). From Politics. In R. Foy (Ed.), *The world of education: Selected readings* (pp. 3-5). New York: The MacMillan Company.

Aukstakalnis, S., & Mott, M.W. (1996). Transforming teaching and learning through visualization. *Syllabus*, *9*(6), 14-16.

Barker, P. (1990). Designing interactive learning systems. *Educational and Training Technology International, 27*(2), 125-145.

Boling, E. (1994). Meeting the challenge of the electronic page: Extending instructional design skills. *Educational Technology*, *34*(7), 13-18.

Buford, J. F. K. (1994). *Multimedia Systems*. New York: ACM Press.

Burwell, L. B. (1991). The interaction of learning styles with learner control treatments in an interactive videodisk lesson. *Educational Technology*, *31*(3), 37-43.

Caftori, N. (1994). Educational effectiveness of computer software. *T.H.E. Journal, 22*(1), 62-65.

Cates, W. M. (1992). Fifteen principles for designing more effective instructional hypermedia/multimedia products. *Educational Technology*, *32*(12), 5-11.

Cormen, T.H., Leiserson, C.E., & Rivest, R. L. (1990). *Introduction to Algorithms*. Cambridge, MA: The MIT Press.

Cortinovis, R. (1992). Hypermedia for training: A software and instructional engineering model. *Educational Technology*, *32*(7), 47-51.

Dede, C. (1995). The evolution of constructivist learning environments: Immersion in distributed, virtual worlds. *Educational Technology, 35*(5), 46-52.

DeNardo, A.M., & Pyzdrowski, A.S. (1994). A study of the effectiveness of computer-based simulations in teaching computer architecture. *Computers in the Schools, 10*(1/2), 125-139.

Dewey, J. (1968). My pedagogic creed. In R. Foy (ed.), *The world of education: Selected readings* (pp. 49-58). New York: The MacMillan Company.

Ely, D. P. (1993). Computers in the schools and universities in the United States of America. *Educational Technology, 33*(7), 53-57.

Erlich, D., & Reynolds, L.(1992). Integrating instructional design and technology: A model and process for multimedia design. *Interactive Learning International, 8*(4), 281-289.

Gagne`, R.M. (1971). The acquisition of knowledge. In P.E. Johnson (Ed.), *Learning theory and practice* (pp. 288-301). New York: Thomas Y. Crowell Company, Inc.

Gagne`, R.M., Briggs, L.J., & Wager, W.W (1992). *Principles of instructional design* (4th ed.). Fort Worth, TX: Harcourt Brace Janovich College Publishers.

Galbreath, J. (1994). Multimedia in education: Because it's there? *Tech Trends, 39*(6), 17-20.

Galbreath, J. (1992). The educational buzzword of the 1990's: Multimedia, or is it hypermedia, or interactive multimedia, or..? *Educational Technology, 32*(4), 15-19.

Heterick, R.C., Jr. (1994). The shoemaker's children. *Educom Review, 29*(3). http://ivory.educom.edu/web/pubs/review/reviewArticles/29360.html

Hirschbuhl, J.J., & Faseyitan, S.O. (1994). Faculty uses of computers: Fears, facts & perceptions. *T.H.E. Journal, 21*(9), 64-65.

Hollis, W.F. (1991). Humanistic learning theory and instructional technology: Is reconciliation possible? *Educational Technology, 31*(11), 49-53.

Johnson, C. W., & Grover, P. A. (1993). Hypertutor therapy for interactive instruction. *Educational Technology*, *33*(1), 5-13.

Johnson, D.L. (1995). Where to look for the learning revolution. *Computers in the Schools*, *11*(4), 1-4.

Jonassen, D.H. (1991). Evaluating constructivistic learning. *Educational Technology, 31*(9), 28-33.

Jonassen, D. H., & Grabinger, R. S. (1993). Applications of hypertext: Technologies for higher education. *Journal of Computing in Higher Education*, *4*(2), 12-42.

Jones, M.G., Farquhar, J.D., & Surrey, D.W. (1995). Using metacognitive theories to design user interfaces for computer-based learning. *Educational Technology, 35*(4), 12-22.

Kaplan, H. (1992). Multimedia in lecture halls: Science & math visualizations. *T.H.E. Journal, 20*(5), 53-55.

Kashef, A.E. (1991). Visualization with CAD. *T.H.E. Journal, 19*(5), 64-66.

Kozel, K. (1996). The interactive experience model: Designing with the spiral. *Multimedia Producer*, *2*(1), 61-66.

Kozel, K. (1995). Crafting the user experience. *Multimedia Producer, 1*(11), 72-80.

Lanza, A. (1991). Some guidelines for the design of effective hypercourses. *Educational Technology*, *31*(10), 18-22.

Lennon, J., & Maurer, H. (1994). Lecturing technology: A future with hypermedia. *Educational Technology, 34*(4), 5-14.

Littauer, J. (1994). A "how-to" on using courseware in the classroom. *T.H.E. Journal, 22*(1), 53-54.

Luther, A. C. (1994). *Authoring Interactive Multimedia*. Boston: AP Professional.

Lynch, P. J. (1995). Entry-level multimedia authoring tools for education. *Syllabus*, *8*(8), 10-18.

Maddux, C. D. (1992). User-developed computer-assisted instruction: Alternatives in authoring software. *Educational Technology, 32*(4), 7-14.

Mauldin, M. (1995). Developing multimedia: A method to the madness. *T.H.E. Journal, 22*(7), 88-90.

McGilly, K. (1994). Cognitive science and educational practice: An introduction. In K. Gilly (Ed.) *Classroom lessons: Integrating cognitive theory and classroom practice* (pp. 2-21). Cambridge. MA: The MIT Press.

McLellan, H. (1994). Situated learning: Continuing the conversation. *Educational Technology, 34*(8), 7-8.

Miller, C.D., Heeren, V.E, & Hornsby, E.J., Jr. (1990). *Mathematical Ideas* (6th ed.). Glenview, IL: Scott, Foresman-Little, Brown.

Moore, D.M. (1994). The parable of the expensive ballpoint pen (revisited): Implications for hypermedia. *Computers in the Schools, 10*(1/2), 3-7.

Myers, R.J., & Burton, J.K. (1994). The foundations of hypermedia: Concepts and history. *Computers in the Schools, 10*(1/2), 9-20.

Naps, T.L., & Nance, D.W. (1995). *Introduction to computer science: Programming, problem-solving and data structures* (3rd alt. ed.). St. Paul, MN: West Publishing Company, Preface, xxiii.

Naps, T.L., & Pothering, G.J. (1992). *Introduction to data structures and algorithm analysis with Pascal* (2nd ed.). St. Paul, MN: West Publishing Co.

NASA: *Parametric Cost Estimating Reference Manual.* http://www.jsc.nasa.gov/bu2/COCOMO.html

NeoBook Professional [Computer software]. (1994). Bend, OR: NeoSoft Corporation.

NeoPaint [Computer software]. (1994). Bend, OR: NeoSoft Corporation.

Noblitt, J.S. (1995). Enhancing instruction with multimedia. *Syllabus, 8*(9), 28-30.

Park, I., & Hannafin, M. (1993). Empirically-based guidelines for the design of interactive multimedia. *Educational Technology, research and development, 41*(3), 63-85.

Park, O. (1994). Dynamic visual displays in media-based instruction. *Educational Technology, 34*(4), 21-25.

Park, O. (1991). Hypermedia: Functional features and research issues. *Educational Technology, 31*(8), 24-31.

Paske, R. (1990). Hypermedia: A brief history and progress report. *T. H. E. Journal, 18*(1), 53-56.

Perkins, D.N. (1991). What constructivism demands of the learner. *Educational Technology, 31*(9), 19-21.

Podell, D.M., Kaminshy, D., & Cusimono, V. (1993). The effects of a microcomputer laboratory approach to physical science instruction on student motivation. *Computers in the Schools, 9*(2/3), 65-73.

Privateer, P. M., & MacCrate, C. (1992). Odyssey project: A search for new learning solutions. *T. H. E. Journal, 20*(3), 76-80.

QuickBASIC 4.5 [Computer programming language]. (1990). Redmond, WA: Microsoft Corporation.

Riley, J.H., Jr. (1990). *Advanced programming and data structures using Pascal*. Boston, MA: PWS-Kent Publishing Co.

Rivlin, E., Botafugo, R. & Shneiderman, B. (1994). Navigating in hyperspace: Designing a structure-based toolbox. *Communications of the ACM, 37*(2), 87-96.

Rodger, S.H. (1996). Integrating animations into courses. *Integrating Technology into Computer Science Education* (pp. 72-74). New York: ACM Press.

Roselli, T. (1991). Control of user disorientation in hypertext systems. *Educational Technology, 31*(12), 42-46.

Sammons, M.C. (1995). Students assess computer-aided classroom presentations. *T.H.E. Journal, 22*(10), 66-69.

Savery, J.R., & Duffy, T.M. (1995). Problem-based learning: an instructional model and its constructivist framework. *Educational Technology, 35*(5), 31-38.

Schach, S.R. (1993). *Software engineering* (2nd ed.). Homewood, IL: Irwin.

Shneiderman, B. (1992). *Designing the user interface: Strategies for effective human-computer interaction* (2nd ed.). Reading, MA: Addison-Wesley.

Shyu, H., & Brown, S.W. (1993). A study of interactive learning: IVS and diagrams. *Computers in the Schools, 9*(4), 71-80.

Skinner, B.F. (1971). The science of learning and the art of teaching. In P.E. Johnson (Ed.), *Learning theory and practice*(pp.22-24). New York: Thomas Y. Crowell Company, Inc.

Smith, G., & Debenham, J. (1993). Automating university teaching by the year 2000. *T.H.E. Journal, 21*(1), 71-75.

Solomon, M.B. (1994). What's wrong with multimedia in higher education? *T.H.E. Journal, 21*(7), 81-83.

Staninger, S. W. (1994). Hypertext technology: educational consequences. *Educational Technology*, *34*(6), 51-53.

Steinberg, E. R. (1991). *Computer-assisted instruction: A synthesis of theory, practice, and technology.* Hillsdale, NJ: Lawrence Erlbaum.

Stoddart, T., & Neiderhauser, D. (1993). Technology and educational change. *Computers in the Schools, 9*(2/3), 5-22.

Sullivan, P. (1971). John Dewey's philosophy of education. In J.C. Stone & F.W. Schneider (Eds.), *Readings in the foundations of education* (2nd ed.): Vol. 2. Commitment to teaching (pp. 495-502). New York: Thomas Y. Crowell Company, Inc.

Sweeters, W. (1994). Multimedia electronic tools for learning. *Educational Technology, 43*(5), 47-52.

Thorndike, E.L. (1971). From the principles of teaching. In P.E. Johnson (Ed.), *Learning theory and practice* (pp.12-22). New York: Thomas Y. Crowell Company, Inc.

Thurber, B. D., Macy, G., & Pope, J. (1991). The book, the computer and the humanities. *T. H. E. Journal*, *19*(1), 57-61.

Tolhurst, D. (1992). A checklist for evaluating content-based hypertext computer software. *Educational Technology*, *32*(3), 17-21.

Turbo Pascal 6.0 [Computer programming language]. (1990). Scotts Valley, CA: Borland International.

Twigg, C.A. (1996). It's the student, stupid! *Educom Review, 31*(3). http://ivory.educom.edu/web/pubs/review/reviewArticles/31342.html

Twigg, C.A. (1995). A chicken-egg dilemma. *Educom Review, 30* (3). http://ivory.educom.edu/web/pubs/review/reviewArticles/30350.html

Wallis, C. (1995, Spring). The learning revolution. *Time*, 49-51.

Wei, C. (1991). Hypertext and printed materials: Some similarities and differences. *Educational Technology*, *31*(3), 51-53.

Weiss, J. (1994). Keeping up with the research. *Technology & Learning*, *14*(5), 30-36.

Willis, J. (1993). What conditions encourage technology use? It depends on the context. *Computers in the Schools, 9*(4), 13-32.

Wilson, B.G. (1995). Metaphors for instruction: Why we talk about learning environments. *Educational Technology, 35*(5), 25-30.

Wilson, J., Aiken, R., & Katz, I. (1996). Review of animation systems for algorithm understanding. *Integrating Technology into Computer Science Education* (pp. 75-77). New York: ACM Press.

Wulfekuhle, N. (1994). Selecting a hypermedia authoring program for CBT. *T. H. E. Journal*, *21*(7), 77-80.

Annotated Bibliography

Abramson, G. (1995). *Learning with Linkway: Tutorial & applications*. Cincinnati, OH: South-Western Educational Publishing.

This training manual focuses on the authoring system, Linkway, by IBM. Nevertheless, many principles are discussed throughout which can be generalized to the design and development of courseware. The tutorials which form the bulk of the book allow the reader to progress from simple to more complex implementations. The chapters on developing subject-rich software, and on design skills are useful, particularly with respect to the planning and design of the courseware product.

Abramson, G.W. (1993). Helping teachers create high-order, highly-motivating, hypermedia-based learning experiences, Part II: IBM Linkway. *SIGTC Connections*, *10*(1), 6-18.

In this practical and challenging article, Abramson makes a case for the use of well-designed and implemented interactive computer-based instruction, describes the contents and process for developing such applications, and gives examples in many subject areas. The author asserts that interactive multimedia instructional applications should aim at higher levels of learning including application, analysis, synthesis, and evaluation. They should not be computer-based page-turning exercises, but should allow the learner to select activities from a web of provided material.

The article gives a good summary of the contents of interactive multimedia packages including the superfolder, an organizing module of the application, and the various other folders or content/activity modules. Also, the specific topical examples help to show what can be done.

Abramson, G.W. (1992). Learning objectives and lesson plans. Unpublished manuscript. William Paterson College of New Jersey.

The emphasis of this article is on establishing higher-level learning objectives to allow technology to be applied effectively to instructional situations. Educational objectives are classified as (1) acquiring and recalling, (2) comprehending, (3) applying, (4) analyzing, (5) synthesizing, and (6) evaluating. Appropriate action verbs are associated with objectives in each category to produce performance objectives.

Aho, A.V., & Ullman, J.D. (1992). *Foundations of computer science*. New York: W.H. Freeman and Company.

Chapter 9 of this text by Aho and Ullman presents concepts and applications of graph data structures including cyclic and acyclic graphs and networks. The chapter begins with basic concepts and terminology including nodes and arcs, paths, cycles, acyclic graphs, and undirected graphs. Two main implementation techniques, adjacency matrices and adjacency lists are described in the next section. Common algorithms such as depth-first search, minimum spanning and shortest path are described and analyzed with regard to execution efficiency. The text is on the theoretical side and provides a thorough basis for understanding graph and network data structures and algorithms.

Alessi, S.M., & Trollip, S.R. (1991). *Computer-based instruction: Methods and development* (2nd ed.). Englewood Cliffs, NJ: Prentice-hall, Inc.

This is a classic text on the design and development of computer-based instruction applications. First the text covers various types of instructional applications, emphasizing those elements which are unique or important to each type of application. Tutorials, drills, simulations, instructional games, and tests are covered. Part 2 of the text presents a detailed explanation of various design and development activities including: preparation, design, flowcharting, storyboarding, programming and preparation of support materials, and evaluation. The last part covers a number of advanced topics. Computer-managed instruction, interactive video, and artificial intelligence applications are discussed.

In addition to the main material in the text there are several helpful appendices. One gives a thorough, although lengthy, summary for evaluating the quality of an application. Another appendix provides sample storyboard forms for use in designing screen layouts.

Ambrose, D. W. (1991). The effects of hypermedia on learning: A literature review. *Educational Technology*, *31*(12), 51-55.

In this article, Ambrose surveys both potentially positive and negative effects of the use of hypermedia learning systems on learning. One major emphasis is that hypermedia, with its web of links among units of knowledge and various media, is structurally similar to the semantic networks of knowledge itself.

Because of its nonlinear structure, hypermedia can encourage learners to explore, speculate, and form hypotheses. These types of activities are considered to be higher-level cognitive activities. Since hypermedia packages typically utilize multiple media types, including dynamic modes such as video, animation, and audio, they are able to attract and hold the learner's attention better than static media.

Problems associated with hypermedia are generally the "flip side" of the benefits. The web of nonlinear links in a hypermedia application can cause learner disorientation and can allow the learner to choose an ineffective sequence of material. The multimedia aspect can provide distractions if not used properly. The author emphasizes the need for research on the effects and effectiveness of hypermedia in learning situations.

Armstrong, T.C., & Loane, R.F. (1994). Educational software: A developer's perspective. *Tech Trends, 39*(1), 20-22.

Armstrong & Loane are two software developers. In this article, the authors discuss several aspects relating to the development of educational software including the advantages of educational software, state-of-the-art development techniques, and market trends. A primary benefit discussed is that of using graphical representation and simple animation to facilitate learning of laws or other processes which cannot be directly observed. Object oriented programming (OOP) techniques are described. The availability of computers to most learners and the ease of use of desktop computers running Windows or the Mac OS are presented. Finally, the authors describe a type of multimedia program, textbook accompaniments, which are gaining popularity. These programs tend to be interactive, and use text, graphics and simple animations to supplement traditional textbooks.

Aristotle (1968). From Politics. In R. Foy (Ed.), *The world of education: Selected readings* (pp. 3-5). New York: The MacMillan Company.

In this excerpt from Politics, Aristotle presents a case for educating for a virtuous life. In the democratic state of Aristotle's time, the belief was that each citizen should in turn govern and be governed. Education was a means of preparing people for their role as governor and as citizen, in particular for virtuous conduct based on rational principle.

Aristotle believed that education would develop the person's capability to rationally consider what was good and to pursue that which was best, not only for himself, but also for the society. He concludes by saying that some "in a vulgar spirit have fallen back on those (virtues) which promised to be more useful and profitable." That is, rather than pursuing the ideal best, they have pursued the immediately practical.

Aukstakalnis, S., & Mott, M.W. (1996). Transforming teaching and learning through visualization. *Syllabus*, *9*(6), 14-16.

Aukstakalnis & Mott present the case for using visualization techniques to facilitate learning difficult concepts, complex processes, and multidimensional structures. Examples come primarily from science and engineering where such concepts, processes, and structures are common.

The claims of the benefits of visualization techniques are based on cognitive psychology and "visual information processing" theories. Complex concepts and processes typically cannot be presented directly. However, when presented in symbolic or codified form, the mind must interpret the presented images before understanding can occur. This activity of interpretation retards learning. A dynamic or multidimensional visual model can often overcome this problem.

Barker, P. (1990). Designing interactive learning systems. *Educational and Training Technology International, 27*(2), 125-145.

This article focuses on multimedia, interactive learning systems used for individualized learning. The author discusses background considerations including the need to follow proven system engineering design and development practices, and the importance of establishing pedagogic objectives. Subject matter should be "open" to the use of technology. Metaphors are important in both the overall design as a unifying element and in aiding learning. Five paradigms are presented for utilizing media: (1) hypermedia paradigm, (2) reactive media paradigm, (3) surrogation, (4) learner-control paradigm, and (5) composite screen paradigm.

Boling, E. (1994). Meeting the challenge of the electronic page: Extending instructional design skills. *Educational Technology, 34*(7), 13-18.

Boling presents practical considerations and techniques for designing educational materials for delivery using the typical computer screen. She discusses the issues of the lack of physical presence, screen resolution and size, interactivity, and the use of motion and sound. The need for navigational controls and other "tools" to assist the learner in the use of the courseware is emphasized. Another important concern is the need for "chunks" of material to be able to stand alone in hypermedia type systems.

Brassard, G., & Bratley, P. (1996). *Fundamentals of Algorithmics*. Englewood Cliffs, NJ: Prentice-Hall.

This is a text on algorithms. The primary emphases are describing types of algorithms, general approaches to designing algorithms and analyzing the efficiency of various algorithms for performing a particular operation. Graphs, digraphs and networks are discussed in several places in the text. The data structures are introduced and formally defined in Chapter 5. Several graph algorithms including minimum spanning tree and shortest path, and graph applications such as scheduling, are discussed in a chapter on greedy algorithms. Chapter 9 focuses specifically on graphs including depth-first and bread-first search and applications to game-playing.

Buford, J. F. K. (1994). *Multimedia Systems*. New York: ACM Press.

Buford presents a wide range of topics relating to multimedia including uses of multimedia information; architectures for multimedia systems; various media technologies such as digital audio and video; operating system and system software support for multimedia; and multimedia communications systems.

Burwell, L. B. (1991). The interaction of learning styles with learner control treatments in an interactive videodisk lesson. *Educational Technology, 31*(3), 37-43.

Burwell discusses the interaction between learning styles and learner control used in computer-based learning applications, specifically those using a videodisk controlled by a computer. After summarizing the findings of other researchers and the lack of agreement in these findings, he describes an experimental study designed to focus on the interaction of learning style and learner control.

The results of Burwell's study demonstrate that field dependent learners learn most effectively when given control options and appropriate advice and guidance, while field independent learners do best when learning is guided by the program. Burwell also found that the groups using the programmed instruction performed better on post tests than those using a written study guide.

Bruner, J.S. (1971). After John Dewey, what?. In J.C. Stone & F.W. Schneider (Eds.), *Readings in the foundations of education* (2nd ed.) Vol. 2: Commitment to teaching, (pp. 495-502). New York: Thomas Y. Crowell Company, Inc.

The principles of John Dewey's pedagogic creed are reviewed and evaluated by Jerome Bruner in light of experience and current educational thinking. With regard to the education within and as an extension of the culture of the day, Bruner argues that education must also provide alternative views of the world and prepare and encourage the learner to explore these alternative views.

With regard to basing the curriculum and activities of education on the interests and capabilities of the child, Bruner says that "it is equally a mistake to sacrifice the adult to the child as to sacrifice the child to the adult." He asserts that the child can be exposed to new interests and develop new capacities.

The school must provide not only an extension to the community and home, but should also be a place of free exploration and exposure to new ideas and knowledge. Bruner proposes that excellence be a hallmark of schools, including excellence in teaching. Finally, in addition to Dewey's ideas, Bruner believes that knowledge is inherently worth mastering for its own right; not all learning has to be immediately useful. Also, while much learning may begin with concrete experiences, at least some learning goes on into more abstract realms.

Caffarella, R.S. (1993). Self-directed learning. In S.B. Merriam (Ed.), *An update on adult learning theory.* New Directions for Adult and Continuing Education, no. 57 (pp.25-35). San Francisco: Jossey-Bass.

Caffarella examines one of the major tenets of Knowles' theories of adult learning, that of self-direction. First, she discusses the underlying philosophical assumptions of self-direction, the primary assumption being that of humanism. The result of this philosophy is that learners are assumed to take primary responsibility for their learning and that the process of learning, which is centered on the learner, is more important than the content.

Other philosophies, progressivism, behaviorism, and critical theory, are also discussed. Critical theory, in particular, focuses on critical evaluation of institutions and culture and initiation of change.

Critics of the concept of self-direction include Boucouvalas who believes that self-direction must be coupled with concepts of interdependence and interconnectedness. Others argue that self-direction varies among learners and even from situation to situation depending on prior experience, learner confidence, and other factors.

Caftori, N. (1994). Educational effectiveness of computer software. *T.H.E. Journal, 22*(1), 62-65.

Caftori describes a study performed at Old Orchard Junior High School in which Junior High School students were observed using educational software during a 20 minute lunch period. Students were allowed to use the software in the computer lab rather than going to study hall. No guidance was given to students. Some students were interviewed as well as being observed. The primary conclusion of this informal study was that when educational software, even that which is considered to be good quality, is used improperly, educational objectives are usually not met. While this may seem obvious, it is likely that much educational software in use is not used with sufficient guidance or in other ways which were not intended.

Cates, W. M. (1992). Fifteen principles for designing more effective instructional hypermedia/multimedia products. *Educational Technology*, *32*(12), 5-11.

The premise of Cates' article is that technology does not revolutionize education, rather teachers and students do. The bulk of the article consists of presenting principles for designing effective multimedia or hypermedia instructional systems based on this premise. The suggested guidelines are practical rather than theoretical and emphasize support of current teaching practice and curricular emphases. Constructivistic considerations are included as well as cautions to use video and interaction in meaningful ways, not just for their own sake. The importance of appropriate and high-quality written materials to support the instructional software is stressed.

Chiou, G. (1993). Some potential areas of research and development in the space of computer-based learning. *Educational Technology, 33*(8), 19-23.

Chiou brings order to the diverse area of research and development associated with computer-based learning. Using a three-dimensional matrix, he organizes numerous research and development areas according to the considerations of user interface design, information technology, and learning concepts. The author calls for developing design rationales which are based on the principles of the contributing areas and that these rationales would be used as guidelines in developing computer-based learning materials.

Cormen, T.H., Leiserson, C.E., & Rivest, R. L. (1990). *Introduction to Algorithms.* Cambridge, MA: The MIT Press.

This is a thorough and weighty text on algorithms of various kinds. It includes five complete chapters on graph-related algorithms, almost 200 pages. Incorporated into the thorough discussion are numerous figures which attempt to show the operation of a given algorithm through a sequence of static images. While the depth of coverage is greater than will be presented in the courseware, the text makes an excellent reference to the topic of graphs and related algorithms. Depth-first search, breadth-first search, topological ordering, minimum spanning tree, shortest path, and several other major algorithms are presented.

Cortinovis, R. (1992). Hypermedia for training: A software and instructional engineering model. *Educational Technology, 32*(7), 47-51.

Cortinovis presents a model for the design and development of computer-based training applications which incorporates current educational theories and software engineering principles. In particular, he focuses on reference-based training and the integration of behaviorist and cognitive theories in the article.

The model consists of lessons comprised of instructional events, which are, in turn, made up of micro-instructional events, MIE's. The MIE's are modular units which can be arranged as needed to create the overall instructional event. The author presents models for a presentation MIE, a structured-test MIE and a job-aid MIE to illustrate the technique.

Dede, C. (1995). The evolution of constructivist learning environments: Immersion in distributed, virtual worlds. *Educational Technology, 35*(5), 46-52.

Chris Dede's article reads like a science fiction book, describing interactions via avatars and with knowbots in distributed virtual worlds. The application of computer technology to create virtual environments in which learners can participate marks a change in the way in which technology is used to implement constructivist theory-based learning. Instead of using technology to mediate the interaction of learners with constructivist environments, technology can now be used to create and populate these environments. In Dede's view, these environments are distributed simulations using networks to link geographically distant users.

Dede discusses various benefits and some drawbacks to this application of technology which extends the concepts of the "Dungeons and Dragons" style role-playing games. Interacting with others using avatars (simulated persona) which can be modified by the user allows people to interact in new ways and without old inhibitions. Being able to manipulate the physical laws of the environment provides first-hand experience with how these laws work.

DeNardo, A.M., & Pyzdrowski, A.S. (1994). A study of the effectiveness of computer-based simulations in teaching computer architecture. *Computers in the Schools, 10*(1/2), 125-139.

While this article focuses on simulation rather than multimedia or animation as such, it is important for three reasons: (1) it describes the use of computer-based instruction in Computer Science, (2) some of the characteristics of simulation are similar to those in animating abstract objects and processes, and (3) the results show improved student attitudes and understanding. Computer-based simulations involve the student and provide immediate feedback and repeatability difficult to achieve in traditional learning.

Dewey, J. (1968). My pedagogic creed. In R. Foy (ed.), *The world of education: Selected readings* (pp. 49-58). New York: The MacMillan Company.

This classic work by the educator John Dewey outlines his primary beliefs about education and learning. There are five primary areas which Dewey considers: (1) what education is, (2) what the school is, (3) the subject-matter of education, (4) the nature of method, and (5) the school and social progress.

Dewey believed that "all education proceeds by participation of the individual in the social consciousness of the race." The main idea here is that education is centered on the individual and their experiences in a social setting. School, for Dewey, existed as a social institution, as a part of "real life" not separate from it. He believed that it should be closely tied to home and community life and that the moral training provided in the home should be supported and extended in the school.

With regard to subject matter, Dewey believed that there should not be a set curriculum and that segregation of subject matter presented an unrealistic view to the learner. He felt that subject matter should spring from the interests and capabilities of the learner and that the "constructive activities" such as cooking, sewing, manual training, etc. were proper areas of study. Because the child is primarily active, rather than passive, Dewey believed that learning should be based in activities and that purely cognitive studies should be tied to appropriate uses of what was learned.

Finally, Dewey, like many others believed that education has, as its ultimate goal, the improvement of society. He felt that education would change people who in turn would change society and that this was more natural and effective than legislated changes. He claims that " the community's duty to education is, therefore, its paramount moral duty."

Ely, D. P. (1993). Computers in the schools and universities in the United States of America. *Educational Technology, 33*(7), 53-57.

This article presents summary statistics about the availability, use, and impact of computers in education in the United States. One telling fact reported is that very little software is available for post-secondary education. Another finding is that the most wide-spread use of computers in education is word processing; little direct application of computer technology to student learning is being used. While most public school systems, as well as most colleges and universities have computers for student use, the rationale for having this technology is more often based on social and vocational reasons than on pedagogic reasons. Ely concludes that on a national scale computer-based education has had little impact in the United States.

Erlich, D., & Reynolds, L.(1992). Integrating instructional design and technology: A model and process for multimedia design. *Interactive Learning International, 8*(4), 281-289.

Erlich & Reynolds describe an approach to the design and development of computer-based courseware. The emphasis of these authors' work is that of instructional design methodology. In particular, they seek to incorporate the use of multimedia into instructional design which traditionally has not dealt with these media.

The technique presented is straightforward, beginning with an analysis of learner needs and goals. Other considerations of the design and development process are: (1) learner characteristics, (2) topics/tasks, (3) objectives, (4) performance assessment, (5) instructional activities, (6) media/delivery systems and (7) resources. The emphasis on learner needs and goals is important and can be adapted to various learning theories.

Gagne', R.M. (1971). The acquisition of knowledge. In P.E. Johnson (Ed.), *Learning theory and practice* (pp. 288-301). New York: Thomas Y. Crowell Company, Inc.

In this article Gagne` presents the theory of a knowledge hierarchy, especially as it pertains to the concept of "productive learning." By productive learning, Gagne` means learning which enables the learner to perform a class of tasks rather than a single task. He hypothesizes that the knowledge needed to perform a type of task may be subdivided into subcategories of more basic knowledge. This process is repeated on the subcategories until one arrives at knowledge which is considered to be fundamental. The result is a hierarchical arrangement of categories of knowledge.

Gagne` then applies the theory to explain individual differences in learning. The idea is that different people enter a learning situation at different points in the knowledge hierarchy and, thus, require, more or less "training" to achieve the final learning goal. Progress through the knowledge hierarchy is accomplished by learners when they transfer knowledge from lower categories to higher ones with instructions being the only learning aid.

Gagne`, R.M., Briggs, L.J., & Wager, W.W (1992). *Principles of instructional design* (4th ed.). Fort Worth, TX: Harcourt Brace Janovich College Publishers.

This is a classic text on instructional design. Its importance to the design and development of computer-based learning applications lies its presentation of the principles of designing educational materials. In particular, the emphasis on the learner, the outcomes of instruction, defining performance objectives, analysis of the learning task, lesson design, and the discussion of individualized learning, are all important to courseware design and development.

Galbreath, J. (1994). Multimedia in education: Because it's there? *Tech Trends, 39*(6), 17-20.

Galbreath reviews the capabilities of multimedia instructional software, makes a case for using multimedia courseware, and examines the requirements for developing courseware. In particular, the author asserts that it is not the role of courseware to replace teachers, but, that courseware can assist teachers to provide individualized, student-centered learning. Galbreath also believes that users (teachers) can become developers if equipped with necessary development hardware and software. Three levels of hardware/software configurations are described.

Galbreath, J. (1992). The educational buzzword of the 1990's: Multimedia, or is it hypermedia, or interactive multimedia, or..? *Educational Technology, 32*(4), 15-19.

Galbreath covers two important aspects of multimedia in this article. First, he explores the meaning of the term multimedia. Second, he provides a summary of various multimedia and interactive technologies which are available today. These include: interactive video, digital platforms, CD-ROM, digital video interactive, compact disc-interactive and a few others.

Gay, L. R. (1992). *Educational research: Competencies for analysis and application* (3rd ed.). New York: Merrill.

Gay presents the essentials of doing educational research including basic statistical analysis. Important stages of research such as selecting and defining the problem, creating a research plan, selecting a sample and appropriate measuring instruments, are described. The text also discusses various types of research models, including historical, descriptive, correlational, causal-comparative, and experimental. Both descriptive and inferential statistics are presented. The text concludes with a description of the process for preparing and evaluating a research report.

Henry, M.J., & Southerly, T.W. (1994). A comparison of the language features of BASIC and Hypercard. *Computers in the Schools, 10*(1/2), 141-153.

The authors describe the main features of BASIC and Hypercard. Common commands in BASIC include INPUT, PRINT, READ/DATA, as well as looping constructs such as FOR/NEXT and branching statements such as GOTO and IF/THEN/ELSE. The capabilities of BASIC in the area of graphics and color are also presented. Hypercard features are described including stacks, backgrounds, and fields and the use of Go, Find, Push , and Pop commands is described.

The discussion centers around the use of one of these languages as a teaching tool. One emphasis of the article is that the choice of language may depend on the learning objectives. Various studies seem to indicate that Hypercard supports learning about software development while BASIC promotes the development of problem-solving skills.

Heterick, R.C., Jr. (1994). The shoemaker's children. *Educom Review, 29*(3). http://ivory.educom.edu/web/pubs/review/reviewArticles/29360.html

The main point of Heterick's article is that computer technology is not being used effectively in education as it is in other areas of society. While computers are used in banks, car engines, recording studios, factories, and other areas, the same colleges and universities which develop new technology are not integrating it into their instructional programs.

Hirschbuhl, J.J., & Faseyitan, S.O. (1994). Faculty uses of computers: Fears, facts & perceptions. *T.H.E. Journal, 21*(9), 64-65.

The authors conducted a study of college faculty to determine the main factors which affect the adoption of computer technology in their instruction. They found that personal attributes such as gender, rank, and research commitment do not affect the likelihood of adoption, whereas, the technological orientation of one's discipline did make a difference. Hirschbuhl and Faseyitan conclude that appropriate training be given to faculty to help them overcome fears of computers and to develop proficiency in computer use.

Hollis, W.F. (1991). Humanistic learning theory and instructional technology: Is reconciliation possible? *Educational Technology, 31*(11), 49-53.

Hollis examines the assumptions inherent in humanist learning theories and in instructional design theory. While not a learning theory per se, instructional design does exhibit a theory of education and draws on learning theories. Hollis draws a dichotomy between humanist theories and instructional design, the first being person-centered, the latter being more process-centered. In humanist theories, learning occurs relative to the learner, knowledge is relative, and the goal is self-actualization. Instructional design is systematic and assumes that appropriate methods must be used to produce desired outcomes.

The author centers on two aspects of learning to try to bring the two theories somewhat together. First, locus of control is examined. While humanist theories emphasize internal locus of control, traditional instructional design has emphasized external locus of control. Hollis says that this is changing with newer technologies.

Secondly, the ties between humanist theories and constructivism are explored. The key here is that it is learners who construct knowledge out of their own experiences. Again newer technologies allow constructivistic learning activities.

Horowitz, E., & Sahni, S. (1989). *Fundamentals of data structures in Turbo Pascal for the IBM PC*. Rockville, MD: Computer Science Press.

Terminology, concepts, and implementation techniques for graphs and networks are presented. Various implementation techniques are compared and algorithms are presented in Pascal. Depth first search, breadth first search, minimum spanning tree and shortest path algorithms are described and presented. Topological sorts and applications to critical path scheduling are also described. Overall the emphasis is fairly theoretical, but provides a solid foundation in the topic.

Jerome, M., & Lee, L. (1995). Multimedia presentation software comes of age. *Newmedia*, *5*(5), 61-75.

The authors present general characteristics of various presentation packages and then describe and compare several packages. Three categories of multimedia presentation software are described: programs which produce slide shows; programs which allow creation of learning materials and the incorporation of various media types; and programs which use a time-line metaphor to organize and synchronize materials and media. Important features of presentation packages include: (1) usability, (2) graphics, (3) outlining and charting, (4) media integration and editing, (5) synchronizing and ordering events, (6) transitions, animations, and glitz, (7) interactivity, and (8) runtime players and cross-platform capability.

Johnson, C. W., & Grover, P. A. (1993). Hypertutor therapy for interactive instruction. *Educational Technology*, *33*(1), 5-13.

This excellent, practical article describes a model and the characteristics of learner-centered computer-based, interactive instructional applications. The authors begin by highlighting two common deficiencies with the majority of current CBI products: (1) deficient message design models which emphasize the machine rather than the learner, and (2) deficient message design execution in which accepted practices of instructional design are overlooked. They acknowledge that early CBI products were primarily guided by behaviorist theories rather than cognitive theories of learning.

The hypertutor model incorporates (1) learner control, (2) consistent presentation formats using instructional design principles, (3)extensive, randomly selected examples and practice exercises, (4) frequent, varied, graded feedback, and (5) extensive, appropriate on-line, randomly-accessible, resources.

Johnson, D.L. (1995). Where to look for the learning revolution. *Computers in the Schools*, *11*(4), 1-4.

Johnson begins by emphasizing that educational technology has not had the widespread impact that many had predicted. He even quotes articles from the popular press which make the same point. He then goes on to say that perhaps we are expecting too much, too fast. Drawing the analogy to the inception of the automobile, Johnson points to Henry Ford as a risk taker and implementer of a new way of doing things, not just a user of new technology. The use of educational technology must be the same. Rather than looking for education-wide adoption of technology, we should point to those areas which have been successful and encourage others to take a risk. Then, eventually, when enough schools have obtained benefits and can demonstrate their success, others will follow suit.

Jonassen, D.H. (1991). Evaluating constructivistic learning. *Educational Technology, 31*(9), 28-33.

Jonassen pursues one of the most critical aspects of constructivist learning theory, that of assessing learning. The difficulty comes from the fact that constructivism, by its nature, does not recognize objective reality and, therefore, cannot use traditional criterion-based evaluation techniques.

Jonassen argues that it is the process of learning rather than the products of learning that should be evaluated in a constructivist learning situation. He also states that evaluation should be carried out by a panel of reviewers, each with a different and relevant perspective.

Jonassen, D. H., & Grabinger, R. S. (1993). Applications of hypertext: Technologies for higher education. *Journal of Computing in Higher Education*, *4*(2), 12-42.

Jonassen & Grabinger present a thorough analysis of the appropriate use of hypertext in educational systems. They begin by arguing against the indiscriminate use of hypertext and the assumption that linking materials together will automatically enhance student learning. The authors divide the acquisition of knowledge into three phases: introductory, advanced, and expert. They then describe appropriate hypertext-based systems for use at each stage. For example, simple hypertext-based tutorials are appropriate in introductory learning. At higher levels of learning hypertext may be used to support complex, constructivistic learning environments, environments which are need driven, provide learner-initiated interaction, and are conceptually and intellectually engaging.

Jones, M.G., Farquhar, J.D., & Surrey, D.W. (1995). Using metacognitive theories to design user interfaces for computer-based learning. *Educational Technology, 35*(4), 12-22.

The authors discuss considerations in the development of user interfaces for CBI applications which are based on metacognitive theories. After describing the state of user interface design, they proceed to explain the main elements of metacognition, the process by which learning is controlled and evaluated by the learner. Metacognitive processes are classified into two categories: 1) control of cognitive processes, and (2) monitoring of cognitive processes.

The article then discusses design considerations which are based on metacognitive processes, organizing the discussion around three questions relating to a CBI application: (1) What is it?, (2) How do I use it?, and (3) What do I know? Several practical issues are discussed. The idea of maps and other devices to let the users know where they are in the program is a useful one. Also, the idea that testing or self-evaluation may be more difficult in hypermedia type applications than in linear ones is a point to consider. When specific learning objectives are defined, there must be a way to be sure that the learner has worked through all necessary material.

Kaplan, H. (1992). Multimedia in lecture halls: Science & math visualizations. *T.H.E. Journal, 20*(5), 53-55.

Kaplan describes the development of computer-based, multimedia programs for use in large, freshman-level courses in chemistry, physics, and math. The key element was the use of images, both still and animated. The multimedia images accomplished the following: (1) captured the learners' attention, (2) made complex processes or those which cannot be directly observed visual, (3) enhanced readability and visibility in a large room through the use of a large screen projection system, (4) allowed dynamic manipulation of the display to explore various conditions.

Kashef, A.E. (1991). Visualization with CAD. *T.H.E. Journal, 19*(5), 64-66.

This article describes the use of a specific computer-based tool, a Computer Automated Design (CAD) system for enhancing instruction. The author reviews the debate regarding the use of CAD to teach engineering and science students, explains CAD and some of its uses, and points out educational benefits.

Among the benefits are helping students to understand structures by providing three dimensional visualizations, allowing students to manipulate visualizations and to experiment with them, providing an interesting and engaging way of studying visual topics, and allowing trial and error experimentation.

This article, although specifically aimed at CAD, provides more support for the use of interactive visualizations to teach complex structures and processes. The ability to make concrete things which are complex or impossible to view naturally simplifies the cognitive task of the learner. Being able to interact with the visualizations provides an experiential component to the instruction.

Kneller, G.F. (1971). Contemporary educational theories. In G.F. Kneller (Ed.), *Foundations of education* (3rd ed.) (pp. 231-251). New York: John Wiley & Sons. Inc.

George Kneller summarizes and critiques four educational theories which have had effects on American education. These include perennialism, progressivism, essentialism and reconstructionism.

Perennialism, essentially affirms the constancy of the most basic elements of life and human nature, emphasizing study of classic literature as a means of learning about these themes. The goal of education is to impart knowledge about eternal truths in preparation for life. The implication is that students need to be taught specified subjects, these primarily being grounded in classic literature, philosophy, history, and science.

Progressivism is essentially embodied in the theories of John Dewey and others who center education on the interests and capacities of the child, and do not prescribe a fixed subject matter. Progressivists claim that education is not preparation for life, but is, rather, life itself. They claim that change is the central theme in human affairs rather than constancy.

Essentialism is closer to perennialism than to progressivism, although essentialists are more willing to accept some of the contributions of the progressivists. The claim is that there are certain essential subjects children should learn, although these are not necessarily based in the classics as held by the perennialists. Essentialism claims that learning, by its nature, involves hard work, that initiative should lie with the teacher rather than with the learner, that there should be a prescribed subject matter, and mental discipline must be used in learning.

Recontructionism sees the school and education in general as leading the way in social reform. It holds that education should commit itself to the creation of a new social order, that the new society should be a genuine democracy, that social and cultural forces condition the learner and the school and that the teacher is to convince the student of the validity of recontructionism.

Kozel, K. (1996). The interactive experience model: Designing with the spiral. *Multimedia Producer*, *2*(1), 61-66.

The second phase of the interactive experience model (IEM) is described in this article, creating the interactive spiral. This spiral includes gaining interest, activity, and resolution. The aim is to engage the user thereby enhancing learning. Kozel discusses the theory of a "flow state" in which the user is completely engaged in what they are doing and the role of the spiral approach in encouraging this state. User options should be presented in ways which promise a benefit if exercised and the overall experience should allow for various levels of challenge.

Kozel, K. (1995). Crafting the user experience. *Multimedia Producer, 1*(11), 72-80.

This article is the first in a two part series on designing interactive, multimedia applications which "work." Kozel describes a model for thinking about and designing interactive, multimedia applications called the interactive experience model (IEM). The model consists of two broad phases: (1) define the goal and related user experience, and (2) develop the application using the interactive spiral. The first phase is discussed in this article. The author emphasizes the necessity of clearly defining the goal. For educational programs this would be the performance objectives. Then, the designer specifies what experience would enable the user to meet the intended goal. Incorporating appropriate experiences or simulated experiences is a type of situated learning and has been shown to be effective.

Lanza, A. (1991). Some guidelines for the design of effective hypercourses. *Educational Technology*, *31*(10), 18-22.

In this article the author emphasizes the importance of designing and implementing computer-based learning applications, in this case hypermedia, according to cognitive learning theories. She points out the danger of having new technology drive development of courseware, rather than following instructional design and cognitive theory guidelines.

Lanza describes the similarity between the network structure of hypercourses and the semantic network structure of knowledge according to cognitive theories. One implication of organizing hyperknowledge as a network and accessing it randomly through links is that a node, or unit of knowledge in a hypercourse, should deal with a single topic or concept and should stand alone. Links in the hypercourse will influence how students learn since access to knowledge is via links which are provided.

Lennon, J., & Maurer, H. (1994). Lecturing technology: A future with hypermedia. *Educational Technology, 34*(4), 5-14.

The focus of this article is on applying computer-based technology to lecture situations. Various traditional methods including the use of blackboards, white boards, overhead transparencies, and flip charts are analyzed to determine the advantages as well as the disadvantages of each. The goal, then is to incorporate the advantages of all methods into a specification for lecturing technology of the future. Advantages of future computer-based lecturing technologies include dynamic presentations, student interaction, and ease of providing all materials to learners.

Littauer, J. (1994). A "how-to" on using courseware in the classroom. *T.H.E. Journal, 22*(1), 53-54.

Littauer focuses on the fact that the role of the classroom teacher changes when instructional courseware is used. He briefly discusses seven considerations: (1) does the courseware basically consist of drills, (2) is the program project-oriented involving students in planning and problem solving, (3) are students grouped simply to help each other, (4) is the courseware interactive, (5) does the program contain all the data necessary to complete the assignment project, (6) what is the teacher's tolerance of "confusion", and (7) is the courseware sold with an unconditional, money-back guarantee?

Luther, A. C. (1994). *Authoring Interactive Multimedia.* Boston: AP Professional.

A general textbook on authoring interactive, multimedia applications. Topics discussed include the authoring process, authoring interfaces, authoring languages, and working with various types of media. Luther emphasizes the need to set up an authoring environment consisting of various authoring tools which are appropriate for the type of applications to be developed and to the skills and inclinations of the developer.

Lynch, P. J. (1995). Entry-level multimedia authoring tools for education. *Syllabus, 8*(8), 10-18.

Lynch presents a review of five authoring systems which are oriented toward beginning courseware developers. The author focuses on packages which facilitate development by using point-and-click style user interfaces and do not require scripting. Action!, Astound, Digital Chisel, mPOWER and Special Delivery are reviewed.

Maddux, C. D. (1992). User-developed computer-assisted instruction: Alternatives in authoring software. *Educational Technology, 32*(4), 7-14.

In this article, the author discusses various approaches which educators can take in developing custom courseware. Reasons for educators to do their own development are presented, and tools for doing the development are discussed. In particular, traditional programming languages, specialized authoring languages, authoring systems and authoring aids are described. The advantages and disadvantages of the various types of tools are examined.

Mauldin, M. (1995). Developing multimedia: A method to the madness. *T.H.E. Journal, 22*(7), 88-90.

Mauldin describes the software development process used at the Educational Technology Laboratory at the Medical University of South Carolina. It is generally a traditional software engineering model tailored to the development of educational materials. Goal specification and needs assessment are done early in the process, followed by analysis,

design, production, and implementation. The article is important in two respects: (1) the need for concentrating on goals, needs and analysis early in the development is emphasized, and (2) the need for formative evaluation at all stages of development is presented. This is seldom emphasized in development models except those of software engineering.

McFarland, R.D. (1995). Ten design points for the human interface to instructional multimedia. *T.H.E. Journal, 22*(7), 67-69.

The author's primary focus is on the design of the Human-Computer Interface (HCI) for multimedia programs. McFarland notes that one important characteristic of multimedia programs is their ability to engage the learner, and it is the human-computer interface which is fundamental in accomplishing this. He then briefly describes ten points to consider in the design of the HCI for a multimedia program.

McGilly, K. (1994). Cognitive science and educational practice: An introduction. In K. Gilly (Ed.) *Classroom lessons: Integrating cognitive theory and classroom practice* (pp. 2-21). Cambridge. MA: The MIT Press.

The idea of information processing is that humans learn and process information in ways similar to the information processing performed by digital computers. With regard to knowledge and knowledge types two ideas are important. First, knowledge tends to be of two types, declarative and procedural. Second, knowledge tends to be retained either as isolated and disconnected bits of information, or as connected, interrelated information. Memory is seen as existing in two types, working memory and long term memory.

Cognitive science methodologies of task analysis and verbal protocol analysis, as well as computer simulation are also described.

Several applications are briefly described which seem to support the claims of cognitive theory. The technique of reciprocal teaching is used to aid reading comprehension. Hypermedia is used to help students see and incorporate the interconnectedness of historical and literary knowledge. Using the concept of elaboration to build knowledge on existing knowledge is used in solving complex mathematical problems.

McLellan, H. (1994). Situated learning: Continuing the conversation. *Educational Technology, 34*(8), 7-8.

In this brief article, Hilary McLellan summarizes key aspects of situated learning. In particular, the theory of situated cognition assumes that knowledge is "contextually situated" and is influenced by the situations in which it is used. The theory identifies three types of learners: novice, expert and "just plain folks." The importance of this last category is that they exhibit some of the characteristics of experts in their normal situated environments, such as at work.

The situated learning model identifies the following components: (1) apprenticeship, (2) collaboration, (3) reflection, (4) coaching,(5) multiple practice, and (6) articulation. Most of these elements involve others and represent a social context for the learning. McLellan states that "stories" and technology are important in situated learning. Finally, knowledge must be learned in an appropriate context such as an actual work setting or a highly realistic "virtual" setting.

Miller, C.D., Heeren, V.E, & Hornsby, E.J., Jr. (1990). *Mathematical Ideas* (6th ed.). Glenview, IL: Scott, Foresman-Little, Brown.

This text by Miller is a mathematics text, not a Computer Science text. However, it provides some historical information about Leonhard Euler who pioneered graph theory, as well as some explanation of graphs and networks. The Koenigsbug Bridge problem is described.

Moore, D.M. (1994). The parable of the expensive ballpoint pen (revisited): Implications for hypermedia. *Computers in the Schools, 10*(1/2), 3-7.

Moore raises the question of whether hypermedia can be used to make a real contribution to education, or whether it is simply a new way of doing old things, one which is admittedly different, but not necessarily better. The author explains what hypermedia is, and discusses the opportunity for student-directed learning. He mentions the ideas of Papert in which students become developers of their own instruction.

Despite potential benefits, Moore cites several problems and drawbacks, including the fact that most existing hypermedia-based instructional programs are not created using instructional design and do not incorporate cognitive learning theories. Disorientation, distraction, human factors, and learner control are issues which need to be carefully addressed in hypermedia systems

Myers, R.J., & Burton, J.K. (1994). The foundations of hypermedia: Concepts and history. *Computers in the Schools, 10*(1/2), 9-20.

While hypermedia has been popular for several years, there still exist misconceptions and misunderstandings about what it is and what it can do. This article provides a good, thorough overview of hypermedia beginning with an explanation of the concept and related terminology. The historical roots are described including important contributors such as Bush, Englebart, Nelson, and Weyer. Hardware and some successful hypermedia systems are presented. The authors believe that Hypercard has done more to make hypermedia popular than any other factor since users can easily create their own hypermedia programs.

The authors also discuss design and implementation issues such as providing navigational aids, proper interface design, and the degree and type of learner control. Factors which may inhibit the use of hypermedia in education, development time and cost, hypermedia's frequent association with cooperative methods, and the change in the role of the teacher, are also presented.

Naps, T.L., & Nance, D.W. (1995). *Introduction to computer science: Programming, problem-solving and data structures* (3rd alt. ed.). St. Paul, MN: West Publishing Company, Preface, xxiii.

In the preface to this Computer Science text, the authors emphasize the use of "extensive figures and graphic documentation...to allow students to visualize the effect of algorithms on data." Additional explanation and examination of the text clearly indicates that a mechanism is needed to help students understand various abstract structures, processes and concepts presented in the text.

Naps, T.L., & Pothering, G.J. (1992). *Introduction to data structures and algorithm analysis with Pascal* (2nd ed.). St. Paul, MN: West Publishing Co.

Naps and Pothering provide a good, clear introduction to graph and network data structures and their associated algorithms. The use of diagrams and figures helps to clarify the presentation. Pascal implementations of data structures and algorithms demonstrate implementation techniques. Implementations are based on the concept of abstract data types which is used throughout the text. Depth first search, breadth first search, shortest path, minimum spanning tree, and topological ordering are all discussed.

NASA: *Parametric Cost Estimating Reference Manual.* http://www.jsc.nasa.gov/bu2/COCOMO.html

The reference provides a brief explanation of COCOMO and provides input fields to allow the user to perform COCOMO estimates online. Particularly helpful is the explanation of the three categories of software development used in the COCOMO model: organic, semidetached, and embedded.

Noblitt, J.S. (1995). Enhancing instruction with multimedia. *Syllabus*, *8*(9), 28-30.

This article links technology, in the broad sense, such as writing, books, images and digital computers to learning and teaching styles. Noblitt emphasizes that all technologies and methodologies have their proper place. The key advantage of the use of media (images) is visualization; the use of digital technology is interactivity. Interactive technology actively engages the learner.

Park, I., & Hannafin, M. (1993). Empirically-based guidelines for the design of interactive multimedia. *Educational Technology, research and development, 41*(3), 63-85.

Park & Hannafin try to bring order and method to the design of interactive multimedia courseware in this excellent article. In particular, they organize relevant research from psychology, pedagogy, and technology and derive principles for interactive multimedia design based on this research. While the content is too exhaustive to summarize, some of the areas considered are (1) improving learning, (2) integration of knowledge, (3) knowledge transfer, (4) feedback, (5) maintaining learner attention, (6) minimizing disorientation, (7) providing guidance, and (8) reducing metacognitive demands.

The twenty "implications" for design are practical, each being based on a corresponding principle derived from research. This is not only helpful in designing interactive multimedia systems, but also useful as a guide in evaluating systems. The presentation gives the designer a solid research base on which to build.

Park, O. (1994). Dynamic visual displays in media-based instruction. *Educational Technology, 34*(4), 21-25.

Park discusses the theoretical bases, strategic applications, and considerations for using dynamic visual displays (DVD's) in courseware applications. He traces the effectiveness of DVD's back to both behavioral and cognitive theories of learning. From a cognitive point of view, DVD's can "make complex cognitive tasks more concrete and easy to understand."

DVD's can be used as (1) an attention guide, (2) an illustration aid, (3) a knowledge representation means, (4) a facilitator of mental model formation, and (5) a visual analogy or reasoning anchor for understanding abstract and symbolic concepts, processes, and structures.

Park, O. (1991). Hypermedia: Functional features and research issues. *Educational Technology, 31*(8), 24-31.

In this article the author discusses three major topics relating to hypermedia research and application to education: (1) the desired functional features of hypermedia, (2) instructional applications, and (3) research issues and technical improvements. Park presents ten functional characteristics for hypermedia. Some which are particularly important to educational applications are the need for guidance in node selection, the need for a browser, the need for node selection using keywords, and interface capability with programming languages. The problems as well as the potential benefits of learner control inherent in hypermedia systems are discussed and research relative to learner-control is described.

Paske, R. (1990). Hypermedia: A brief history and progress report. *T. H. E. Journal, 18*(1), 53-56.

Paske provides a short summary of the key concepts of hypermedia. Hypermedia is defined and some of the history of hypermedia is presented. The overall structure of a hypermedia system is briefly described. Hypermedia is contrasted to interactive video and interactive multimedia. The author touches on the bandwidth demands of hypermedia systems delivered on a network and explains a little about ISDN.

Perkins, D.N. (1991). What constructivism demands of the learner. *Educational Technology, 31*(9), 19-21.

In this article, Perkins takes a critical look at the demands which constructivism places on the learner. The main areas considered are: cognitive complexity, task management, and "buying in."

Several factors contribute to the complexity of the cognitive load placed on the learner. First, students are generally asked to cope with complex situations. Second, phenomenaria are typically rich environments. And lastly, the conflict between learners' naive concepts and the concepts being learned may be dealt with by "facing" the conflict between the two, which may be difficult for learners.

Task management is a key part of learning. Constructivism expects more of the task management to be assumed by the learner than does a traditional learning situation. Finally, the learners may not wish to work in a constructivist environment. They may want teachers to "teach" them, rather than taking responsibility for their own learning.

Podell, D.M., Kaminshy, D., & Cusimono, V. (1993). The effects of a microcomputer laboratory approach to physical science instruction on student motivation. *Computers in the Schools, 9*(2/3), 65-73.

In this article Podell, Kaminsky & Cusimono describe a study which they conducted to determine the effect of the integration of computer technology into a secondary physical science course on factors related to motivation. The effects on attendance, attitudes toward science and toward computers, and student enrollment in subsequent science courses were studied. The researchers randomly assigned students to experimental or control groups from a population of all ninth-grade physical science students in a New York City high school. Students in the experimental groups used computer-based and hands-on activities while students in the control group received more traditional instruction, primarily lecture format and demonstrations.

The results of the study showed, that for the population studied, the use of computer-based instruction had a positive effect on student attitudes and behavior. Student attendance and enrollment in subsequent courses improved, and students' attitude toward computers was positively influenced. Also, science achievement was higher for the experimental group.

Privateer, P. M., & MacCrate, C. (1992). Odyssey project: A search for new learning solutions. *T. H. E. Journal, 20*(3), 76-80.

Privateer & MacCrate have developed an interesting and creative courseware application which supports the use of collaborative learning techniques in a humanities course. The authors found that students in large classes of a humanities course were reduced the role of transcribers and were not afforded opportunities for critical thinking and analysis which the instructors desired.

The courseware was a hypermedia application used in conjunction with collaborative techniques. The courseware provided an interdisciplinary knowledge base, navigational aids, course management tools, and an integrated word processor which allowed the students to contribute to the knowledge.

The authors seem to have done a good job of applying hypermedia technology to a course. The implementation was creative and effective. It is interesting that the courseware was not expected to stand on its own, but was supported by specific techniques, both in terms of group work and special lecture formats. This may partially account for the success of the application.

Riley, J.H., Jr. (1990). *Advanced programming and data structures using Pascal.* Boston, MA: PWS-Kent Publishing Co.

A Computer Science text, this book not only covers the concepts and terminology related to graphs and graph algorithms, but also includes Pascal code demonstrating the implementation of both the data structures and the algorithms. It provides a practical reference to the implementation of graphs and graph algorithms. Both the adjacency matrix and adjacency list techniques for implementing graphs are presented in detail.

Rodger, S.H. (1996). Integrating animations into courses. *Integrating Technology into Computer Science Education* (pp. 72-74). New York: ACM Press.

The author describes the use of algorithm animations in Computer Science courses. Animations are used in lectures to clarify the description of algorithm operation and to demonstrate visually the effects of algorithms on data structures. Animations are also used in laboratories. Students can run the animations with different data sets to see how the algorithms operate under different conditions. The animations are interactive, allowing students to pause and replay the program. Rodger also describes the use of the XTANGO animation tool which was used to create the animations.

Roselli, T. (1991). Control of user disorientation in hypertext systems. *Educational Technology, 31*(12), 42-46.

Roselli discusses problems associated with the use of hypertext (or hypermedia) systems for educational applications. These include the need for higher-order metacognitive skills, and possible disorientation and cognitive overload. Having described the problems the author suggests a means for overcoming these problems. The technique involves monitoring the user's navigational choices, and having the system make suggestions for "better" selections.

Sammons, M.C. (1995). Students assess computer-aided classroom presentations. *T.H.E. Journal, 22*(10), 66-69.

The article describes a project undertaken at Wright State University in Dayton, Ohio which was intended to assess and encourage faculty use of technology in their teaching. Faculty teaching large, lecture intensive, general education courses were the primary participants. The primary use of technology was to prepare computer-based presentations for use in lectures. One aspect of the study was an assessment by students of the faculty use of technology. The vast majority of students felt that using computer-based presentations supported course content, made lectures more organized, and facilitated seeing and reading instructor presentations and note-taking. The article presents general recommendations and comments regarding screen design, multimedia use, methodology, room design and layout, and hardware.

Savery, J.R., & Duffy, T.M. (1995). Problem-based learning: an instructional model and its constructivist framework. *Educational Technology, 35*(5), 31-38.

This article on constructivism emphasizes a problem-based learning approach, in particular, that used by Barrows. The authors begin by summarizing three principles of constructivism: (1) understanding is gained through interaction with the environment, (2) cognitive conflict provides a stimulus to learn, and (3) knowledge is created by social negotiation.

Instructional principles derived from constructivism are: (1) learning activities should be anchored to a larger task or problem, (2) learners should be supported in developing ownership for a problem, (3) tasks should be authentic, (4) tasks and activities should reflect the complexity of the actual situation in which the learner will eventually operate, (5) allow the learner to be responsible for the process for developing the solution, (6) the environment should support and challenge the learner, (7) encourage learners to consider alternative views, and (8) allow and encourage the learner to reflect on the solution and the process of arriving at the solution.

Schach, S.R. (1993). *Software engineering* (2nd ed.). Homewood, IL: Irwin.

This is a standard text used in Computer Science curricula for learning the discipline of software engineering. The scope and processes of software engineering are discussed as well as specific software development models, computer-assisted software engineering (CASE) tools, and testing principles. Each major phase of the development process is described in detail.

Schwier, R. A. (1993). Learning environments and interaction for emerging technologies: Implications for learner control and practice. *Canadian Journal of Educational Communication, 22*(3), 163-176.

The focus of Schwier's article is on levels of interactivity and learning environments. Three learning environments are described: prescriptive, democratic, and cybernetic. In prescriptive learning, basically all elements of learning, the content, sequence, activities, etc., are defined and learner interaction is restricted to reaction. Learners can be more proactive in democratic learning environments where they are allowed to make decisions and choices about sequence, content, and activities.

In cybernetic environments, learners work with adaptive computer-based systems. As in democratic systems, learners exercise a high degree of control; however, the system intelligently adapts to the learner's actions. Generally, learning is more constructivistic.

Shneiderman, B. (1992). *Designing the user interface: Strategies for effective human-computer interaction* (2nd ed.). Reading, MA: Addison-Wesley.

This is a classic text by the leading researcher in the field of human-computer interface (HCI) design. Shneiderman discusses a wide range of topics including: (1) human factors, (2) theories, principles, and guidelines for HCI, (3) menus, (4) command languages, (5) direct manipulation, (6) interaction devices, (7) response time and delay rate, (8) system messages, screen design, and color, and many others. Much of the focus of the text is on the benefit of direct manipulation and the creation of easy-to-use, intuitive interfaces.

Shyu, H., & Brown, S.W. (1993). A study of interactive learning: IVS and diagrams. *Computers in the Schools, 9*(4), 71-80.

Shyu & Brown discuss the possible advantages of using dynamic visual displays, especially in contrast to the use of static displays as in a textbook. They cite one study which seems to indicate that achievement was not enhanced by use of dynamic displays from an interactive videodisk system (IVS), however, another study showed significantly higher achievement for groups using dynamic means instead of static displays.

In addition, student reaction to the dynamic displays was more positive. The authors then present a study which they did comparing the use of dynamic visual images with that of static images in the learning of a procedural task. Their results do not show a significant difference in performance for learners using dynamic displays compared to those using static displays. However, students using dynamic displays did have more favorable attitudes about the instruction.

Skinner, B.F. (1971). The science of learning and the art of teaching. In P.E. Johnson (Ed.), *Learning theory and practice*(pp.22-24). New York: Thomas Y. Crowell Company, Inc.

B. F. Skinner describes recent (at the time) advances in learned behavior using appropriate reinforcements and schedules of reinforcements. In this article he claims two significant advances in techniques in the field of learning: (1) a better understanding and use of the Law of Effect, and (2) schedules of reinforcement which promote the maintenance of given behaviors for long periods of time.

After describing these techniques in experimental settings, Skinner moves on to their application to education. In particular, teaching elementary-level children mathematical skills is presented in terms of providing appropriate positive reinforcements. Based on the complexity of teaching mathematics, Skinner concludes that the classroom teacher cannot adequately provide the necessary reinforcements, and suggests a teaching machine to do the task. He concludes with an appeal for American education to use its resources to provide such machines in the schools.

Skinner's suggestion to use teaching machines, especially for learning subjects such as mathematics, certainly foreshadows the use of computers for drill-and-practice type applications. His emphasis on reinforcements, such as prompt positive feedback, is something a teaching machine or computer-based lesson can easily do.

Of course, Skinner views the process from a purely behaviorist view, provide proper stimuli and you will get proper responses. What is interesting is that in certain areas, such as drill and practice, the techniques associated with this view appear to work. By treating the learning process as a "black-box" and determining what inputs have to be entered to create desired outputs, it is not necessary to understand the actual processes of learning.

Smith, G., & Debenham, J. (1993). Automating university teaching by the year 2000. *T.H.E. Journal, 21*(1), 71-75.

Smith & Debenham assert that the traditional methods of delivering education at the college and university level must change in response to economic and other pressures. They propose a solution which is being developed at The University of Utah, that of using interactive, multimedia courseware to teach basic concepts and terminology, freeing faculty to focus on software development, lead seminars, and in general deal with higher level issues. The software created by the authors, COMPUTER TUTOR, is based on cognitive learning theories. Studies of classes using the software have shown a number of advantages in student attitude and performance.

Solomon, M.B. (1994). What's wrong with multimedia in higher education? *T.H.E. Journal, 21*(7), 81-83.

In this article, the author looks at the failure of various technologies to revolutionize education. Television and the microcomputer have not produced the revolutionary impact on education which was promised, and neither has multimedia says the author. Five factors which are making it difficult for multimedia to have a significant impact on education are: (1) the culture of higher education, (2) cost, (3) lack of standards, (4) multiple talents required to produce software and (5) the amount of time needed to produce software. Solomon asserts that only when these factors are overcome will multimedia be able to have the impact on education which many are hoping for.

Staninger, S. W. (1994). Hypertext technology: educational consequences. *Educational Technology, 34*(6), 51-53.

The author begins by explaining what hypertext is and how a hypertext application is structured using nodes and links. The nodes represent units of knowledge and may be comprised of information presented using various media. Staninger believes that the links are the vital part of the application and that a primary challenge for the designer is to determine and implement appropriate links.

According to the author, the main benefit of hypertext-type applications is that they allow the learner to pursue knowledge in ways which are appropriate to the ways in which people learn, specifically, in a nonlinear fashion. Cognitive Flexibility Theory claims that knowledge is interconnected and hypertext demonstrates and reinforces this.

Stanley, R. (1995). Steps, roads, funnels, galaxies: Metaphors for designing interactive presentations. *T.H.E. Journal, 22*(5), 57-61.

Stanley, an art instructor, presents a different view on the process of design of a multimedia program. He approaches the design process and the multimedia program as an artist, one not always comfortable with the rigid step-by-step approach demanded by most design and development methodologies. He maintains that in the early stage activities of selecting lessons, describing outcomes, doing an overall design, identifying materials, and exploring various pedagogic techniques should be done simultaneously, each being affected by the others until it becomes necessary to apply stricter discipline and work sequentially. His point is not that everyone should work in this manner, but that for some it may be more comfortable and productive than a strict sequential approach.

Steinberg, E. R. (1991). *Computer-assisted instruction: A synthesis of theory, practice, and technology.* Hillsdale, NJ: Lawrence Erlbaum.

This text is the second volume of a two-volume set, the first volume being, *Teaching Computers to Teach.* Unlike the first book which is primarily a how-to guide for developers of computer-assisted instruction (CAI), this volume focuses on the theoretical

foundations of design and development practices. Two important themes run throughout the book: (1) instructional interactions between people and computers are distinctive, and (2) despite this distinctiveness, there are features of CAI which are similar to other modes of instruction.

Stoddart, T., & Neiderhauser, D. (1993). Technology and educational change. *Computers in the Schools, 9*(2/3), 5-22.

The authors assert that, even with the use of computer-based instructional systems, education has changed little since "the inception of the common school." The primary thrust of the article is the comparison of Integrated Learning Systems (ILS's) and tool-based systems (TBS's). An ILS is primarily based on behaviorist learning theories and an objectivist view of learning. Knowledge is seen as primarily external to learners and the job of the teacher is to transmit this knowledge to students.

TBS's are predicated on cognitive theories, particularly constructivist theories which focus on knowledge being created by learners as they interact with the world around them. Thus, the TBS approach to computer-based instruction differs radically from the ILS approach and supports student-centered learning, and active student participation in the learning process. The authors also point out several barriers to adoption of TBS systems including resistance to change, lack of effective teacher training and cost in terms of both time and money.

Sullivan, P. (1971). John Dewey's philosophy of education. In J.C. Stone & F.W. Schneider (Eds.), *Readings in the foundations of education* (2nd ed.): Vol. 2. Commitment to teaching (pp. 495-502). New York: Thomas Y. Crowell Company, Inc.

Phyllis Sullivan summarizes John Dewey's Pedagogic Creed and gives a brief critique of Dewey's views. She begins by explaining the educational environment of the late 1800's and noting that "previous to this period, education was received from the home and some public institutions." Sullivan emphasizes that Dewey believed that the success of democracy depended on education both by the family and by the school.

Sullivan then reviews the five main principles in Dewey's pedagogic creed. First, education begins "by understanding a child's capacities, interests, habits, and instinct." In other words, education of children should be done in a way which recognizes that children are not little adults. Secondly, school must provide opportunities for learning within social settings, encouraging learning and discipline for the good of the social group, rather than for some external purpose.

The curriculum would be flexible, and the teacher's role would be to select appropriate activities geared to the children's interests and abilities. The author notes that Dewey has been criticized for not properly developing his concept of the curriculum. In her summary, the author relates some of the positive effects of Dewey's philosophy.

Sweeters, W. (1994). Multimedia electronic tools for learning. *Educational Technology, 43*(5), 47-52.

Sweeters describes and evaluates several types of instructional applications including (1) tutorials, (2) educational databases, (3) simulations, and (4) educational games as well as a model he calls learning nodes. Within the educational databases type are included hypertext and hypermedia applications.

The author examines each of the instructional applications in light of Gagne`s events of instruction. This provides a sound educational basis for analysis rather than focusing on technology issues. It is interesting that the tutorial method, which generally is losing favor, contains all of the events of instruction.

Thorndike, E.L. (1971). From the principles of teaching. In P.E. Johnson (Ed.), *Learning theory and practice* (pp.12-22). New York: Thomas Y. Crowell Company, Inc.

Written in the early 1900's(1906), this article takes an interesting view of education. First, education is intended to change people. Thorndike says that "The need for education arises from the fact that what is is not what ought to be." Second, the primary goals of education are to teach societal and behavioral values to individuals, in particular, "Education should make human beings wish each other well, should increase the sum of human energy and happiness and decrease the sum of discomfort of the human beings that are or will be, and should foster the higher, impersonal pleasures."

Finally, Thorndike sees the process as one largely of stimulus and response. The questions relating to teaching then become, what stimuli should I as a teacher provide to get the desired responses from the learners?

Thurber, B. D., Macy, G., & Pope, J. (1991). The book, the computer and the humanities. *T. H. E. Journal, 19*(1), 57-61.

The authors describe a computer-based learning application in the humanities and their rationale for developing the application. The application, NewBook, incorporates hypertext capabilities, as well as other forms of learner interactivity such as note-taking and writing essays. The goals were to engage the learner in a "reader-centered method of instruction, to embed the teaching of integrative, evaluative skills in the course material itself", and to "enable readers to have continuous evaluation of their own thought."

Tolhurst, D. (1992). A checklist for evaluating content-based hypertext computer software. *Educational Technology, 32*(3), 17-21.

Various criteria for evaluating hypertext or hypermedia educational systems are described. Factors discussed include: (1) user control vs. amount of guidance, (2)

navigational aids, (3) clarity of linking mechanisms, and (4) hypertext contexts. The author also discusses curriculum considerations in the use of hypertext.

Twigg, C.A. (1996). It's the student, stupid! *Educom Review, 31*(3). http://ivory.educom.edu/web/pubs/review/reviewArticles/31342.html

In this article, Twigg decries the lack of college-level instructional software and the opinion that their is not a market for such software. The primary fallacy of those who do not think that there is a market is that they are wrong about who the ultimate customers are, students. Twigg says that instructional software should be considered to be similar to learning materials such as textbooks, learning tools such as calculators, and learning services. Basing the potential demand on student need leads to the conclusion that there is a significant market for instructional software.

Twigg, C.A. (1995). A chicken-egg dilemma. *Educom Review, 30* (3). http://ivory.educom.edu/web/pubs/review/reviewArticles/30350.html

Twigg notes that, in general, there is not a great deal of good instructional software at the college level. Much of what is available consists of ad hoc learning programs created by faculty for their own courses. Twigg's suggestion is that a consortium of colleges and universities get together and create specifications for needed instructional software and agree to use it if developed. The specifications should be given to companies in the software development industry and paid for collectively by the colleges and universities. She even suggests some top-level specifications, including those based on achieving measurable learning outcomes.

Wallis, C. (1995, Spring). The learning revolution. *Time*, 49-51.

Wallis notes that the prophesied wide-spread impact of computer technology in education has not occurred. She goes on , however, to describe one particular school, the Dalton School, where computer-based technology has made a significant impact on education. Applications in history, literature, and science are described. The article gives a glimpse of the possibilities of effective CBI.

Wei, C. (1991). Hypertext and printed materials: Some similarities and differences. *Educational Technology*, *31*(3), 51-53.

Similarities and differences exist between printed materials and hypertext systems. The author points out that both media can be used for instruction; provide information; and use text, graphics, diagrams and pictures. However, there are fundamental differences between these two media.

In particular, hypertext systems are generally more complex, promote nonlinear thinking, emphasize cognitive information processing, and may be dynamic in content and structure.

Weiss, J. (1994). Keeping up with the research. *Technology & Learning, 14*(5), 30-36.

Key elements of current research in the area of computer-based learning are described in this article. Issues are reported from several major research efforts including Stanford Research Institute and Educational Development Corporation, consultant and researcher Saul Rockman, and researcher Henry Jay Becker. General topics include the impact of computer-based learning, factors affecting the implementation and effectiveness of computer-based learning, and factors affecting teacher use of computer-based learning.

Willis, J. (1993). What conditions encourage technology use? It depends on the context. *Computers in the Schools, 9*(4), 13-32.

Willis maintains that the primary issues relating to implementing computer technology in education today are different from those of the past. He claims that hardware selection is simpler due to fewer, more similar options. Training, while necessary, is more related now to how instructional software can and should be used than to how to use the equipment or the software. And finally, Willis believes that the technical quality, not necessarily the educational quality, of available software is better than in the past.

Several concrete suggestions for facilitating introduction of computer-based technology into the curriculum are given, including making educators aware of what is available, allowing them to explore new technologies, providing consultation for planning how to use technology in their classes, and providing training "just-in-time", that is, when it is needed and will be helpful.

Wilson, B.G. (1995). Metaphors for instruction: Why we talk about learning environments. *Educational Technology, 35*(5), 25-30.

The theory of situated cognition insists that all learning occurs in real situations, where the learner interacts with other people, employs appropriate "tools", and interacts with the environment. The claim is that learning cannot occur in "artificial" situations, such as formal schools.

The author cites research results as well as educational theoreticians such as Dewey and Kolb to bolster the case for situated learning. Dewey asserted that "All genuine education comes about through experience." Kolb's four-stage model of learning incorporates two stages, concrete experiences and active experimentation, which are obviously experiential.

The author claims that learning must be situated in "authentic activities" meaning those which require exactly the cognitive processes which one is trying to develop. The claim is that learning is not transferable from similar experiences.

Wilson, J., Aiken, R., & Katz, I. (1996). Review of animation systems for algorithm understanding. *Integrating Technology into Computer Science Education* (pp. 75-77). New York: ACM Press.

Wilson, Aiken & Katz describe several available packages which can be used to animate the operation of algorithms. The animations are dynamic visualizations of algorithm operation and the effect of the algorithms on data structures. These systems allow students to explore the operation of algorithms, often interactively. Several animation systems are briefly described including: Balsa, Xtango, GAIGS, and FLAIR. The authors note that evidence relating to the effectiveness of these packages is mostly anecdotal. Studies which have been done indicate that algorithm animations should be used as a supplement to traditional instruction, not in place of it.

Wulfekuhle, N. (1994). Selecting a hypermedia authoring program for CBT. *T. H. E. Journal*, *21*(7), 77-80.

This article, written by a program manager for a training organization, focuses on the use of hypermedia as a means of implementing computer-based training programs. Learner control, form of information processing and source data organization are the three functional areas on which the author concentrates. Each of these is elaborated to provide specific criteria to be used in the evaluation or design of a hypermedia computer-based training application. The author seems to have a high regard for the potential of hypermedia and believes that successful applications will custom build lessons to meet specific users' problems, resulting in reduced training costs and time.

Appendix A

Technology Fact Find Questionnaire

Technology Fact Find Questionnaire

The following questions are intended to explore the use of computer technology in educational, business, personal,and other situations. Please complete each question as accurately as possible. Thank you for your cooperation.

Interviewer ______________________________ Date ______________

Person interviewed ________________________ Position___________

PART I: COMPUTER USAGE AT WORK

SYSTEM INFORMATION (Check appropriate items)

Computer: IBM PC/compatible __386 __486 __Pentium __Other PC
Macintosh __68000 __ Power Mac __Other Mac
__DON'T KNOW

Operating System: __ DOS 5.0+ __Windows 3.1+ __Windows 95/NT
__Mac System 7.5 __Other
__DON'T KNOW

RAM: __1MB __2MB __4MB __8MB __More than 8MB __DON'T KNOW

Video: __VGA __SVGA __14"/256 color (Mac) __Other __DON'T KNOW

Other: __mouse __printer __sound card __CD-ROM __DON'T KNOW

Hard Disk size (fill in amount) ______ __DON'T KNOW

WORK USAGE INFORMATON (Check appropriate items as they apply to your computer usage at work.)

Frequency of use: __daily __weekly __less than once/week

Applications: __word processing __spreadsheets __database
__presentations __desktop publ. __Other

Internet Use: __World Wide Web __News groups __email
__download software __NOT APPLICABLE

Educational Use: __prepare materials __presentations __gradebook
__tutorials __simulations __drills
__Other __NOT APPLICABLE

How satisfied are you with the system you have at work?

__Very Satisfied __It's OK __Dissatisfied

Page 2

PART II: COMPUTER USAGE AT HOME

SYSTEM INFORMATION (Check appropriate items)

```
Computer:  IBM PC/compatible   __386     __486  __Pentium     __Other PC
           Macintosh           __68000   __ Power Mac         __Other Mac
           __DON'T KNOW

Operating System:   __ DOS 5.0+    __Windows 3.1+       __Windows 95/NT
                    __Mac System 7.5                    __Other
                    __DON'T KNOW

RAM:       __1MB  __2MB  __4MB  __8MB  __More than 8MB  __DON'T KNOW

Video:     __VGA  __SVGA __14"/256 color (Mac)  __Other __DON'T KNOW

Other:     __mouse __printer __sound card   __CD-ROM   __DON'T KNOW

Hard Disk size (fill in amount) ______   __DON'T KNOW
```

HOME USAGE INFORMATON (Check appropriate items as they apply to your computer usage at home.)

```
Frequency of use: __daily   __weekly   __less than once/week

Applications:     __word processing     __spreadsheets  __database
                  __presentations       __desktop publ. __Other

Internet Use:     __World Wide Web      __News groups   __email
                  __download software   __NOT APPLICABLE

Educational Use:  __prepare materials   __presentations __gradebook
                  __tutorials           __simulations   __drills
                  __Other               __NOT APPLICABLE
```

How satisfied are you with the system you have at work?

```
                  __Very Satisfied    __It's OK   __Dissatisfied
```

==

Thank You for your cooperation !

==

Appendix B

Courseware Evaluation Form

Courseware Evaluation Form

General Information

1. Reviewer : ______________________ 2. Date : ____________________

	CS Faculty	Non-CS Faculty	Fresh/ Soph	Junior/ Senior	
3. Classification	1	2	3	4	

Courseware User Interface

	Hard				Easy
4. Rate the overall ease of use	1	2	3	4	5
	Poor				Great
5. Rate the overall appearance	1	2	3	4	5
	Very Hard				Very Easy
6. How easy is it to navigate, e.g. to go from one activity or topic to another?	1	2	3	4	5
	Very Hard				Very Easy
7. How easy is it to find the specific activity or topic you want, such as a demonstration or lab exercise?	1	2	3	4	5
	Difficult to read				Easy to read
8. Screens are ...	1	2	3	4	5
	Very Cluttered				Not Cluttered
9. Screen layouts are ...	1	2	3	4	5
	Distracting				Helpful
10. Use of color, graphics and sound are ...	1	2	3	4	5

Instruction

	Strongly Disagree				Strongly Agree
11. Content is accurate.	1	2	3	4	5
12. Content has educational value.	1	2	3	4	5
13. Prerequisites are clear.	1	2	3	4	5

Courseware Evaluation Form (Page 2)

Instruction (cont.)

	Strongly Disagree				Strongly Agree
14. Purpose of courseware is clear.	1	2	3	4	5
15. Courseware achieves its goal.	1	2	3	4	5
16. Level of difficulty is appropriate for intended audience.	1	2	3	4	5
17. The courseware increased my motivation.	1	2	3	4	5
18. Learners receive appropriate feedback.	1	2	3	4	5
19. Learners have appropriate degree of control.	1	2	3	4	5
20. Tutorials are helpful.	1	2	3	4	5
21. Animated demonstrations are helpful.	1	2	3	4	5
22. Interactive laboratory sessions are helpful.	1	2	3	4	5
23. Self-tests are helpful.	1	2	3	4	5
24. Supplementary materials are helpful.	1	2	3	4	5
25. The courseware helped my learning.	1	2	3	4	5

Comments: Please write any additional comments about the courseware.

Appendix C

Functional Specification

Graphs & Networks: Concepts & Implementation

Instructional Software

Functional Specification

by

Thomas E. Beutel

January 1997
Revised March 1997

Table of Contents

List of Tables

List of Figures

1.0 Problem Description and Goals

1.1 Problem

The problem to be addressed is the development of instructional software which facilitates learning abstract structures, operations, and concepts by Computer Science students in the area of graphs and networks. To be effective, the software must not only deal with subject matter content accurately and appropriately, but must be developed according to established software engineering and instructional design principles, utilize current educational technology effectively, and be guided by cognitive learning theories. To accomplish the development in a reasonable time development tools must be used which facilitate the development process.

The ever-increasing use of computer technology in business, education, entertainment, and most fields of endeavor, has challenged Computer Scientists to improve the way in which computer software is designed and developed. In response, Computer Scientists have adopted methodologies from the field of engineering. In general, an engineer begins the design process by determining the needs of the ultimate user, then proceeds through a series of designs, beginning with a purely conceptual, or abstract, model and culminating in a detailed design. The focus at early stages of the design on abstract representations of the end product allows the engineer to concentrate on the essential features and to experiment with a variety of implementations.

One aspect of applying engineering methods to computer programming is to view data structures and algorithms as abstractions, or conceptual models. This allows the software designer to focus on the essential elements of the data structures and algorithms and to experiment with various implementations, in much the same way as the engineer does with an engineering design.

Often, abstractions and conceptual models are difficult to grasp, especially for Freshman and Sophomore college students confronting abstract data types and algorithms for the first time.

While dealing with abstractions in Computer Science is a problem in general, the difficulties escalate when students are exposed to complex, logical data structures and their associated algorithms. Simple data structures typically define the relationship of one element to another in terms of physical ordering. A logical data structure, on the other hand, is one in which the relationship of one element of the data structure to another is not physically determined. Instead, each item includes information about which other items are related to it.

The structure of these logical data structures is more complex and harder to visualize than that of simpler structures. Examples of such structures include linked lists, binary trees, and graphs and networks. Because the structures themselves are more complex, the algorithms which manipulate these structures are also more complex. Thus, the general problem of dealing with abstractions is made more difficult by having to deal with complex abstractions.

Researchers and practitioners emphasize the difficulty of learning foreign concepts and suggest the use of computer simulation and visualization techniques as tools to enhance student learning of complex, abstract concepts, structures, and operations in science and engineering.

1.2 Goals

The goal of this development is to produce courseware which will provide interactive, animated graphical representations of graphs and networks which facilitate learning the abstract structures, operations and concepts related to graph and network data structures and related algorithms by Computer Science students. Specific learning objectives for the completed courseware implementation when used to supplement normal instruction and assignments are given in Table 1.

Table 1: Courseware Learning Objectives

1. The student will **define the following terms** related to graphs and networks:

    a. acyclic graph
    b. adjacency matrix
    c. breadth-first traversal
    d. cycle
    e. degree
    f. depth-first traversal
    g. digraph
    h. edge
    i. graph
    j. minimal spanning tree
    k. network
    l. node
    m. path
    n. topological ordering

2. The student will **explain the following concepts** related to graphs and networks:

    a. A graph or network is comprised of a set of nodes and edges.
    b. Traversing a graph or network using a breadth-first technique.
    c. Traversing a graph or network using a depth-first technique.
    d. The use of an adjacency matrix as a means of implementing a graph or network data structure.
    e. Finding the shortest path between two nodes in a network.
    f. Finding the minimum spanning tree of a network.
    g. Performing a topological ordering of the nodes in a network.

3. The student will **implement an adjacency matrix** as a means of implementing a graph or network.

4. The student will **implement common primitive operations** for graphs and networks including:

   a. creating a graph/network
   b. adding an edge
   c. removing an edge
   d. adding a node
   e. performing a depth-first traversal
   f. performing a breadth-first traversal.

5. The student will **implement additional operations** for networks including:

   a. shortest path
   b. minimum spanning tree
   c. topological ordering.

6. The student will recognize and describe problems amenable to solution using graphs and networks.

7. The student will design a computer program which uses graphs or networks to solve a specific problem.

Accomplishment of learning objectives will be determined by normal examinations and homework assignments.

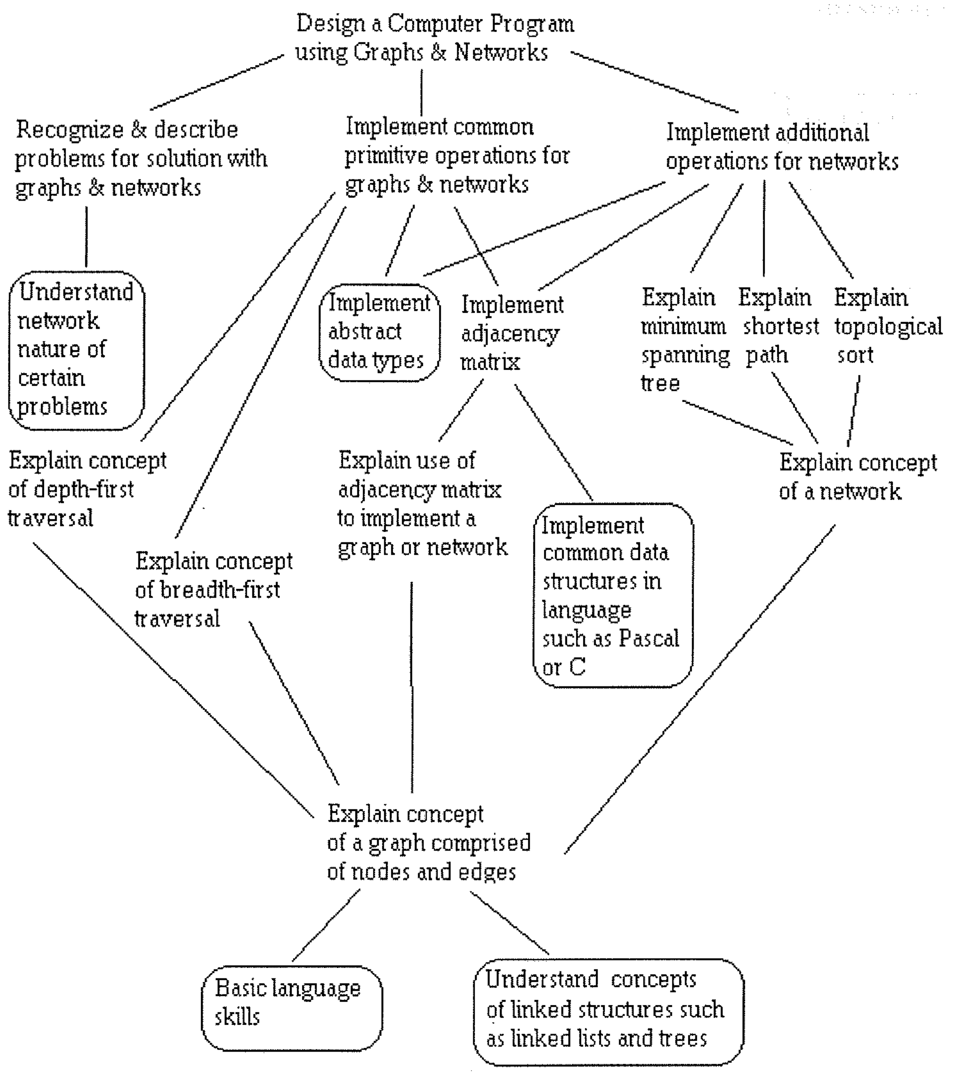
2.0 Instructional Analysis

Each of the learning objectives has certain prerequisite skills or knowledge which the learner must have to successfully achieve the objective. The objectives are not independent, but form a hierarchy of learning. To insure effective and efficient learning, instructional activities must take into account the interdependency of the learning objectives and the prerequisite skills and knowledge.

Instructional analysis is used to define the interdependencies among learning objectives and the prerequisites. An instructional curriculum map (ICM) is prepared by starting with the highest cognitive goal and decomposing that goal into its prerequisites. This process is then repeated on each prerequisite until the prerequisites which are identified are those assumed to be true for the intended learners.

For the Graphs & Networks courseware, the highest cognitive goal is the goal of designing and writing a computer program using graphs or networks to solve a problem. This goal involves application and problem-solving. Figure 1 shows the ICM for the Graphs & Network courseware. The items enclosed in round-cornered boxes represent basic skills and knowledge assumed for the intended learners, Sophomore Computer Science students.

Figure 1: Instructional Curriculum Map for Courseware


Design a Computer Program using Graphs & Networks
Recognize & describe problems for solution with graphs & networks
Implement common primitive operations for graphs & networks
Implement additional operations for networks
Understand network nature of certain problems
Implement abstract data types
Implement adjacency matrix
Explain minimum spanning tree
Explain shortest path
Explain topological sort
Explain concept of depth-first traversal
Explain concept of breadth-first traversal
Explain use of adjacency matrix to implement a graph or network
Implement common data structures in language such as Pascal or C
Explain concept of a network
Explain concept of a graph comprised of nodes and edges
Basic language skills
Understand concepts of linked structures such as linked lists and trees

3.0 User Characteristics and Prerequisites

Based on the instructional analysis of the preceding section, certain prerequisites can be identified for the intended learners, Sophomore Computer Science students. In addition, other characteristics are representative of this group of learners. Table 2 summarizes these characteristics and prerequisites.

Table 2: Learner Characteristics for Graphs & Networks Courseware

		Level		Time to	Difficulty
	Low	Avg	High	Learn	to Learn
Year	Soph	Junior	Senior		
Experience	Prereq[1]	Prereq[1]	Prereq[1]		
Motivation	High	High	High		
Interest	High	High	High		
Computer Operation	High	High	High		
Courseware Familarity	None	None	None	10-30 min	Easy
Basic Language Skills	Average	Average	Average		
Understand Concepts of Linked Structures	Low	Average	High		
Implement Common Data Structures	Average	High	High		
Implement Abstract Data Types	Average	High	High		
Understand the Network Nature of Certain Problems	Average	High	High		
Implement Graph & Network Data Structures and Algorithms	None	None	None	4 - 8 Hours	Difficult

1. Prereq. denotes completion of Computer Science I, Computer Science II, and File Processing

4.0 Functional Definition

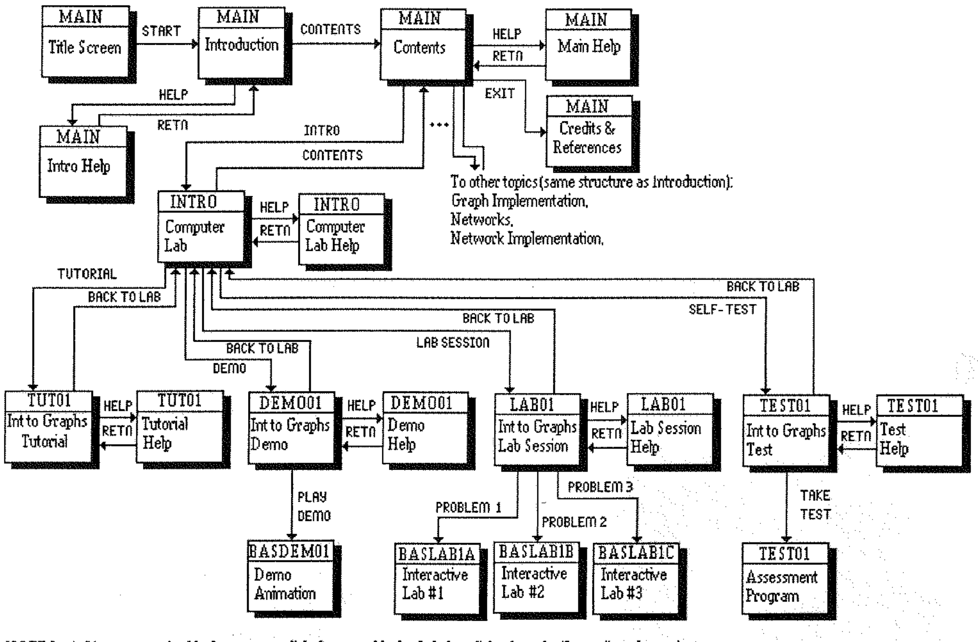
The courseware will consist of several types of instructional activities including tutorials, animated demonstrations, laboratory exercises, and self-tests. The emphasis will be on using the computer to provide learning experiences which are difficult to do in other ways and to facilitate learning of abstract structures, concepts and operations. Dynamic visualizations and interactivity will be important aspects of the courseware. In particular, the operation of algorithms such as depth-first traversal or shortest path on graph or network data structures will be demonstrated using simple animation. Laboratory exercises will allow the learner to build structures visually on the display, then run algorithms to observe the results.

4.1 Overall Structure

Functionally, the courseware will consist of five major parts: the introductory section, and four topical units covering graphs, graph implementation, networks, and network implementation. Each topical unit will incorporate four instructional activities: tutorial, animated demonstration, interactive laboratory sessions, and self-test. The overall structure of the Graphs & Networks courseware is shown in Figure 2.

The courseware is organized as a hypermedia system. Each block in the structure diagram represents a node in the hypermedia network, with the lines between the blocks representing links between nodes. In some cases, such as a tutorial, there may be multiple pages arranged linearly for presenting the content of the topic. There may also be additional pages or supplementary information which are linked in hypermedia fashion. These additional and supplementary pages are not shown in the structure diagram.

Figure 2: Overall Structure of Courseware



NOTES: 1. Lines connecting blocks represent links between blocks. Labels on links show the "button" used to activate the link.
2. Individual blocks may be implemented as multiple "pages" with primarily linear navagation using NEXT and PREV buttons.

The following sections describe each major unit of the courseware in detail, specifying what each unit will do. Implementation details are contained in the Design Specification.

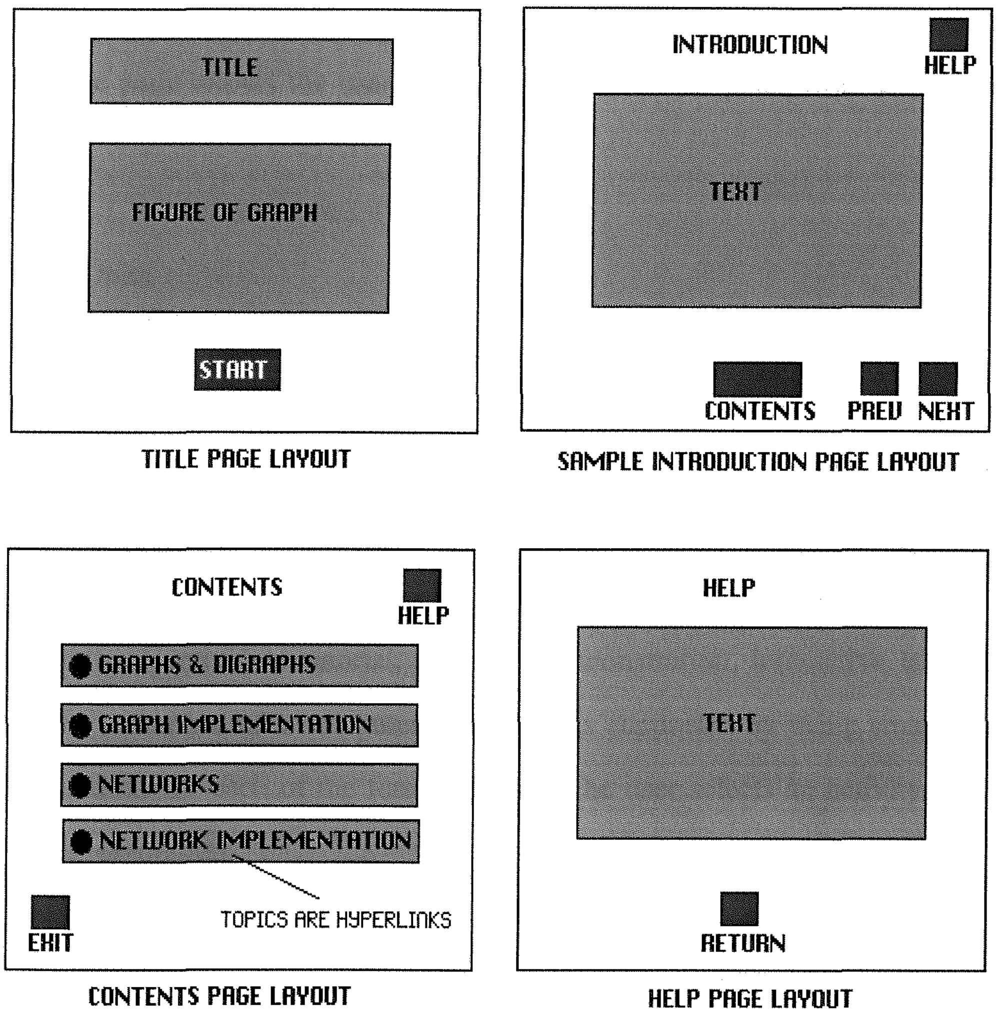
4.2 Individual Unit Specifications

4.2.1 Main Unit (Introductory Section)

The Main Unit serves as the starting point for use of the courseware. It includes a title page, introduction, and table of contents page. The table of contents page acts as the root node of the hypermedia network. The learner can access any of the four major units from the contents page. Similarly, each major unit is linked back to the contents page to provide a means for the learner to return to a known location in the hypermedia network.

The introduction will cover four main topics: (1) a brief explanation of the subject matter, (2) explanation of the courseware, (3) the learning objectives, and (4) prerequisites. Online help is accessible from all pages and explains the controls available to the learner. Figures 3 shows the page layout for the title page, a sample introduction page, the contents page and the main help page.

Figure 3: Page Layouts for the Main Unit



The Title Page will include the title of the courseware "Graphs & Networks: Concepts & Implementation", the author's name and college, a stylized image of a graph, an acknowledgment of the use of NeoBook, and the date. There will also be a start button on the page. Clicking on the start button will jump to the first page of the introduction.

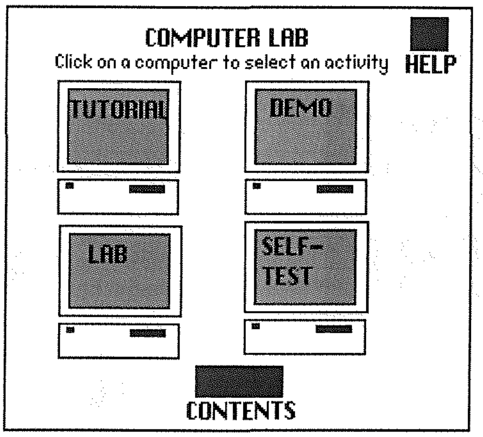
The introduction will consist of four pages as described above, with navigational controls for going to the next page or the previous page, help, or jumping directly to the contents page. Thus, the user who is familiar with the courseware need not go through the entire introduction section each time the software is used. The Help Page will include a description of the current section and any navigational controls. A return button at the bottom of the page allows the user to return directly to the place from which help was requested.

4.2.2 Topic Units

From the Contents Page the user can jump to one of four topic units. Each topic unit has the same configuration, although the content will differ according to the topic. On selection of a topic from the Contents Page, the user will follow a link to the Computer Lab page for the selected topic. The Computer Lab page acts as the starting point for all activities for a specific topic. From the Computer Lab page, the learner may select one of four instructional activities: tutorial, animated demonstration, interactive laboratory, or self-test. The metaphor of a computer laboratory is supported by using small images of computer stations for each of the four activities. The user selects an activity by clicking on one of the computers.

Figure 4 shows the layout of the Computer Lab page for each topic and Figure 5 shows the page layouts for each of the four instructional activities.

Figure 4: Page Layout for Computer Lab Page

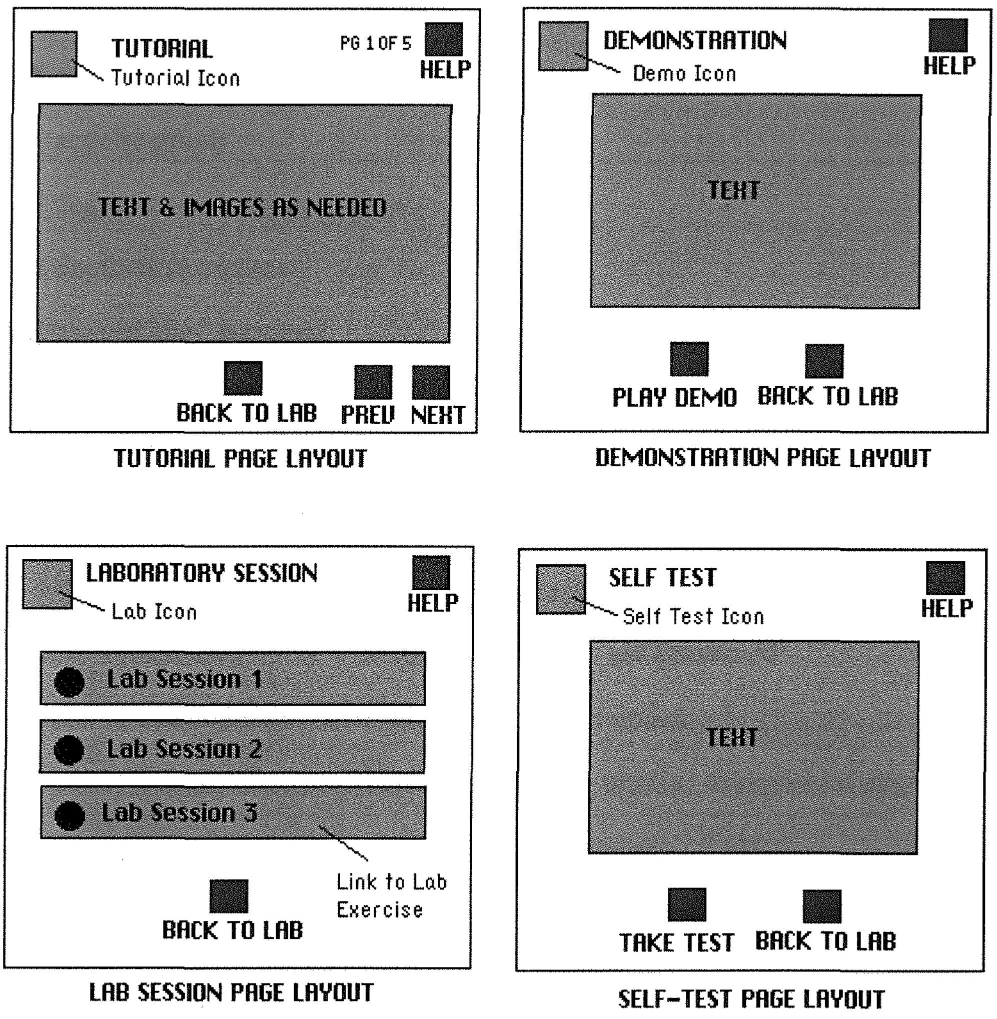


The tutorial unit will consist of pages having the format shown for the Tutorial Page in Figure 5. Each tutorial will consist of several pages containing text and images as needed to present the content related to the topic. In addition a help button, navigational controls for moving through the pages of the tutorial, and a button to return to the computer lab are included on the each page.

The Demonstration Page includes a help button, a button to return to the computer lab, a brief description of the animated demonstration, and a button to start the demonstration. Clicking on the Play Demo button will activate a QuickBASIC program which will present the animated demonstration.

Similarly, the Lab Session Page and the Self Test Page include controls to return to the computer lab, get help and activate the related instructional activities. The Lab Session page lists several laboratory exercises each of which is a hyperlink to the related interactive, animated QuickBASIC program. The Self Test page includes a button to allow the learner to Take Test. This button also links to a QuickBASIC program which presents the self test.

Figure 5: Page Layouts: Tutorial, Demonstration, Lab Session and Self-Test



The following sections describe the content of each of the topical instructional activities.

4.2.3 Graphs and Digraphs Tutorial

The Graphs and Digraphs tutorial presents general concepts and terminology relating to graphs. The following terms and concepts are covered:

- graph
- digraph
- node

- edge
- path
- cycle
- acyclic graph
- degree
- depth-first traversal
- breadth-first traversal

4.2.4 Graph Implementation Tutorial

The concept of using an adjacency matrix to represent a graph will be presented in this tutorial. Many of the terms and concepts from the Graphs and Digraphs tutorial are related to the adjacency matrix. The following topics are presented:

- use of a one-dimensional array to represent nodes
- use of a two-dimensional array (adjacency matrix) to represent edges
- symmetry of the adjacency matrix for an undirected graph
- relation of the adjacency matrix to paths, degree of a node, and traversals

4.2.5 Networks Tutorial

The basic concept of a network and what characteristics distinguish a network from a graph will be described. Common network algorithms will also be explained. Specific topics include:

- edge weight
- shortest path algorithm
- minimum spanning tree algorithm
- topological sort algorithm

4.2.6 Network Implementation Tutorial

Use of an adjacency matrix with edge weights as entries in the matrix will be presented as a means of implementing a network. A description of how the network algorithms use the information in the adjacency matrix to perform their operations will be discussed.

4.2.7 Graphs and Digraphs Demonstration

The animated demonstration for Graphs and Digraphs will present much of the same information as presented in the related tutorial activity. The primary difference between the two presentations is that the demonstration will incorporate simple animation. The use of animation will be particularly effective in showing the operation of algorithms such as the depth-first or breadth-first traversal on a particular graph.

The learner will be able to run the animation, proceed forward or backward through the pages of the animation, and terminate the animation at any time and return to the computer lab page. The animation will be implemented by a QuickBASIC program running within a window on the demonstration page of the courseware. Figure 6 shows the general layout of the demonstration page and the window for the animation. Figure 7 shows the design for the user display and controls for the demonstration which will be implemented in QuickBASIC.

Figure 6: Demonstration Page with Window for Animated Demo

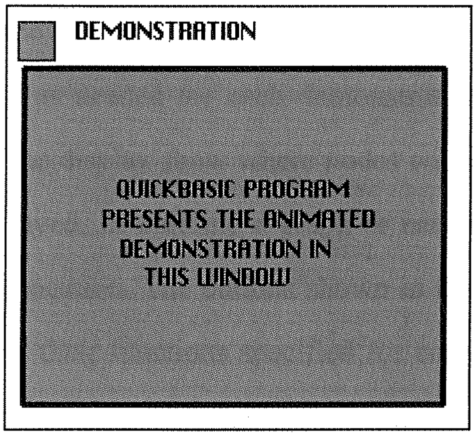
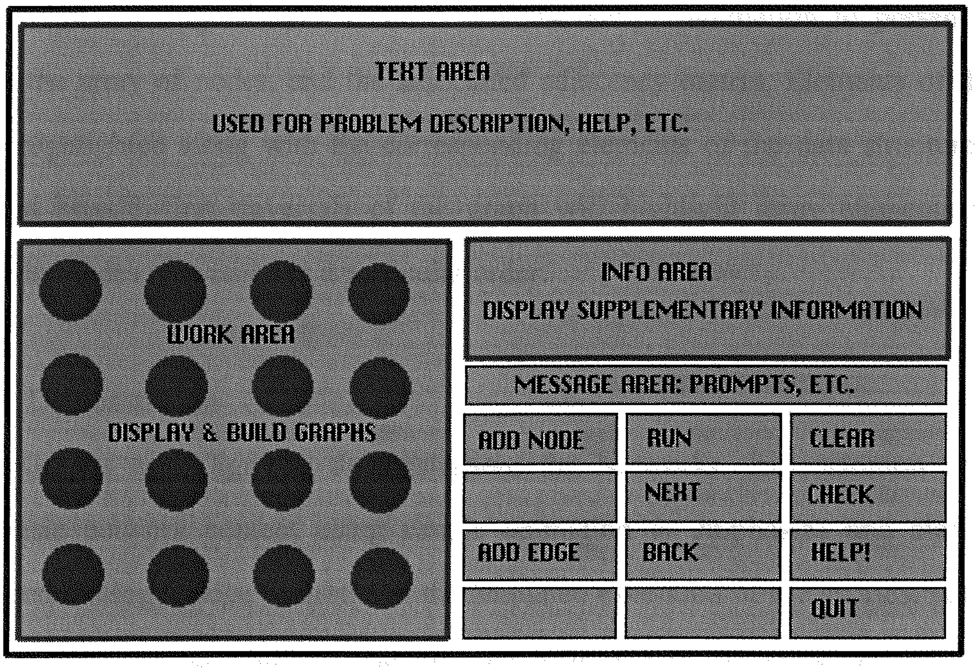

DEMONSTRATION
QUICKBASIC PROGRAM PRESENTS THE ANIMATED DEMONSTRATION IN THIS WINDOW

Figure 7: User Display and Controls for Lab/Demo Program


TEXT AREA
USED FOR PROBLEM DESCRIPTION, HELP, ETC.
WORK AREA
DISPLAY & BUILD GRAPHS
INFO AREA
DISPLAY SUPPLEMENTARY INFORMATION
MESSAGE AREA: PROMPTS, ETC.
ADD NODE
RUN
CLEAR
NEXT
CHECK
ADD EDGE
BACK
HELP!
QUIT

The user display shown in Figure 7 will be used for all demonstrations, lab sessions and self-tests. The various windows within the display, the text area, information area, and message area, will be used as needed for each demonstration, lab session or test. The circles in the work area of the display show where nodes will be displayed when a graph or network is built or displayed. In building a graph or network the student clicks on a circle to add a node at that location. The buttons shown in the lower right quarter of the display can be renamed and their functions specified for each session as needed. Using the same user interface for demonstrations, lab sessions, and tests and displaying the user display within a window of the main display will provide continuity and consistency for the user.

4.2.8 Graph Implementation Demonstration

The demonstration for Graph Implementation will use simple animation to present a simple graph, the array of nodes, and the associated adjacency matrix. Elements of the array will be highlighted along with the corresponding elements of the data structures. Depth-first and breadth-first traversals of the graph will highlight array elements to demonstrate why nodes are visited in a particular order.

4.2.9 Network Demonstration

As with the Graphs and Digraphs demonstration, the Networks demonstration will present the basic network content using simple animation to emphasize and clarify concepts and terminology. Edge weight, and the animated execution of the shortest path, minimum spanning tree, and topological sort algorithms will be presented in the demonstration.

4.2.10 Network Implementation Demonstration

The Network Implementation demonstration presents simple animations illustrating the use of the adjacency matrix to carry out the minimum spanning tree, shortest path and topological sort algorithms.

4.2.11 Graphs and Digraphs Lab Session

Three exercises will be provided in the Graphs and Digraphs Lab Session. In the first, the user will be given a simple problem and asked to build the corresponding graph. The nodes of the graph will be displayed in the work area of the user display. The student will use the Add Edge button to link the provided nodes appropriately to solve the given problem. The session allows students to check their answers when they think they have completed the problem.

The second exercise allows the student to build a graph then run the depth-first traversal algorithm on that graph from a specified starting node. The student can rerun the algorithm from different start nodes or may clear the graph and build a new one. The goal of the exercise is to help the student understand how a depth-first traversal works.

In the third exercise the student also builds a graph. Once the graph is built, the student can run the breadth-first traversal algorithm on that graph from a specified starting node. The student can rerun the algorithm from different start nodes or may clear the graph and build a new one. The goal of the exercise is to help the student understand how a breadth-first traversal works.

4.2.12 Graph Implementation Lab Session

In this lab session, the student will be presented with one laboratory exercise. The student is asked to build a graph and views the adjacency matrix as the graph is being built. The goal of the exercise is to help the student understand the concept of the adjacency matrix.

4.2.13 Network Lab Session

Four exercise are provided in the Network Lab Session. In general, they are similar to those in the Graph Lab Session. In the first, the student builds a network to solve a specified problem. The intent is to provide experience in using the lab display to build a network and to reinforce the concept of a network. In each of the other three exercises, the student will build a network, then run an algorithm on the network. The shortest path, minimum spanning tree and topological sort algorithms are each used in one of the three exercises.

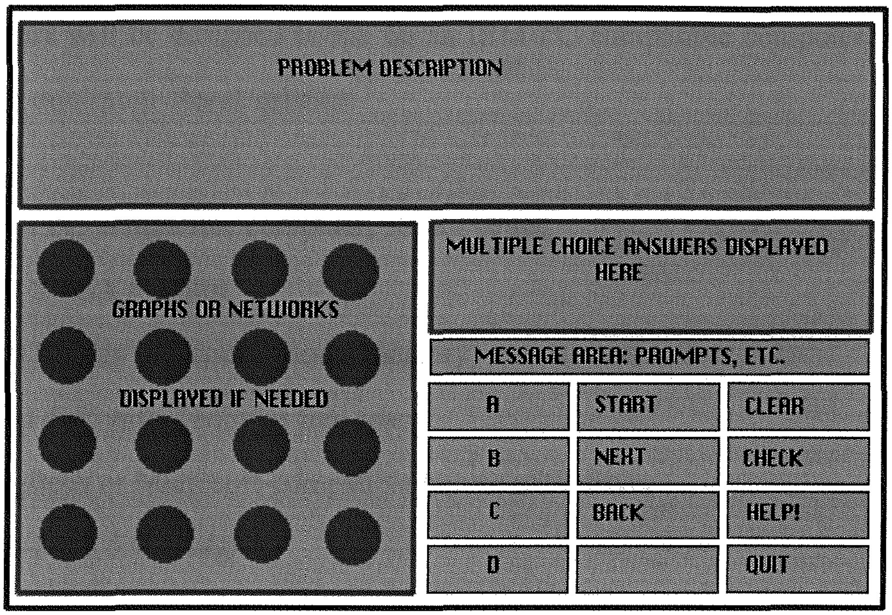
4.2.14 Network Implementation Lab Session

This lab session is similar to the Graph Implementation Lab Session. The student will either build a network and view the adjacency matrix.

4.2.15 Self Tests

The self test for a particular unit will consist of up to ten multiple choice questions covering the terms and concepts covered in the tutorial, animated demonstration, and laboratory session for the related topic. The user interface used for the interactive laboratory sessions and animated demonstrations will be used to present the test. Figure 8 shows how the various portions of the display and user controls will be used.

Figure 8: User Display and Controls for Self Tests



Notice in Figure 8 that several of the buttons have been redefined. Those at the left of the button area are labeled A, B, C, and D, and will be used to indicate multiple choice answers. The start button is used to begin the self-test or to restart at the beginning. The Next and Back buttons will advance to the next question or allow the student to return to the preceding question. The Check button is used to check answers to the questions. Incorrect answers will be indicated by changing the display to the question and displaying a message in the message area.

Some problems may use a graph or network as part of the question. These will be displayed in the work area.

5.0 System Requirements

The courseware will be designed to run on an IBM PC compatible computer which has the following minimum characteristics:

- IBM PC compatible computer, 386 or better
- 14" SVGA monitor
- 530 Kbytes free conventional memory
- hard disk with 2 Mbytes free space
- Logitech or Microsoft-compatible mouse with driver
- MS-DOS 5.0 or higher

The courseware is not designed to run under Windows. It may be run as a DOS application from Windows as long as all minimum requirements are met.

Appendix D

Design Specification

Graphs & Networks: Concepts & Implementation

Instructional Software

Design Specification

by

Thomas E. Beutel

February 1997
Revised March 1997

Table of Contents

List of Tables

List of Figures

1.0 Introduction

The Graphs & Networks instructional software is described in the document **Graphs & Networks: Concepts & Implementation Instructional Software Functional Specification** which describes the problem being addressed, the development goals, an instructional analysis and summary of user characteristics, and the functional description of the overall package and the individual units which comprise the package.

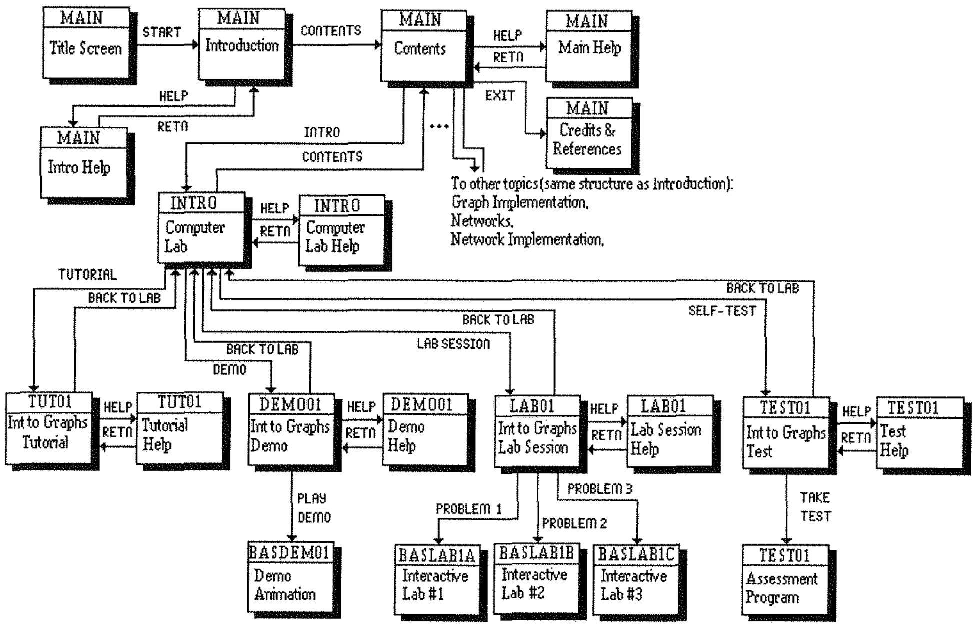
This document describes the design of the courseware including the details of the user interface design, the data structures and algorithms used to implement general features as well as graphs and networks, and the details of the individual instructional activities such as the tutorials, animated demonstrations, interactive lab sessions, and self-tests.

2.0 Overall Design

Functionally, the courseware will consist of five major parts: the introductory section, and four topical units covering graphs, graph implementation, networks, and network implementation. Each topical unit will incorporate four instructional activities: tutorial, animated demonstration, interactive laboratory sessions, and self-test. The overall functional structure of the Graphs & Networks courseware is shown in Figure 1.

The courseware is organized as a hypermedia system. Each block in the structure diagram represents a node in the hypermedia network, with the lines between the blocks representing links between nodes. In some cases, such as a tutorial, there may be multiple pages arranged linearly for presenting the content of the topic. There may also be additional pages or supplementary information which are linked in hypermedia fashion. These additional and supplementary pages are not shown in the structure diagram.

Figure 1: Overall Functional Structure of the Courseware



NOTES: 1. Lines connecting blocks represent links between blocks. Labels on links show the "button" used to activate the link.
2. Individual blocks may be implemented as multple "pages" with primarily linear navagation using NEXT and PREV buttons.

The courseware will be implemented in a manner which follows the overall structure as shown in Figure 1. Some portions of the courseware will be implemented with a courseware authoring system, NeoBook Professional, while others will be implemented using QuickBASIC, a compiler version of BASIC.

The authoring system will be used to create the overall courseware structure and the primary user interface. The reason for using the authoring system is the ease of creating screens which incorporate graphic images, text, interactive controls such as buttons and hyperlinks, and other interactive, multimedia features. While these features could be implemented with a traditional programming language, the overall courseware structure and user interface can be created much more quickly and easily with the authoring system.

NeoBook Professional was selected as the authoring system for several reasons including ease of use, support of all media types except video, the ability to call external programs, and the ability to compile stand-alone, royalty-free applications. QuickBASIC was selected primarily because its compiled programs would run within a window in the NeoBook-created screens without clearing the screen and the palette was more easily adapted to that selected for the NeoBook portion than for other languages such as Turbo Pascal. Also, NeoBook Professional is a DOS application, so windows-based languages such as Visual BASIC were not appropriate.

The implementation using NeoBook and QuickBASIC will include compiled NeoBook publications, compiled QuickBASIC programs, and data files used for lab sessions, interactive demonstrations, and self-tests. The specific implementation design is shown in Figure 2. In general, the structure in Figure 2 follows the functional design from Figure 1, although it emphasizes the implementation structure.

In Figure 2 square-edged boxes represent portions of the courseware which will be implemented using NeoBook, round-edged boxes represent portions to be implemented in QuickBASIC, and the boxes with a diagonal upper right corner represent data files. The following sections describe the design of the various parts of the courseware.

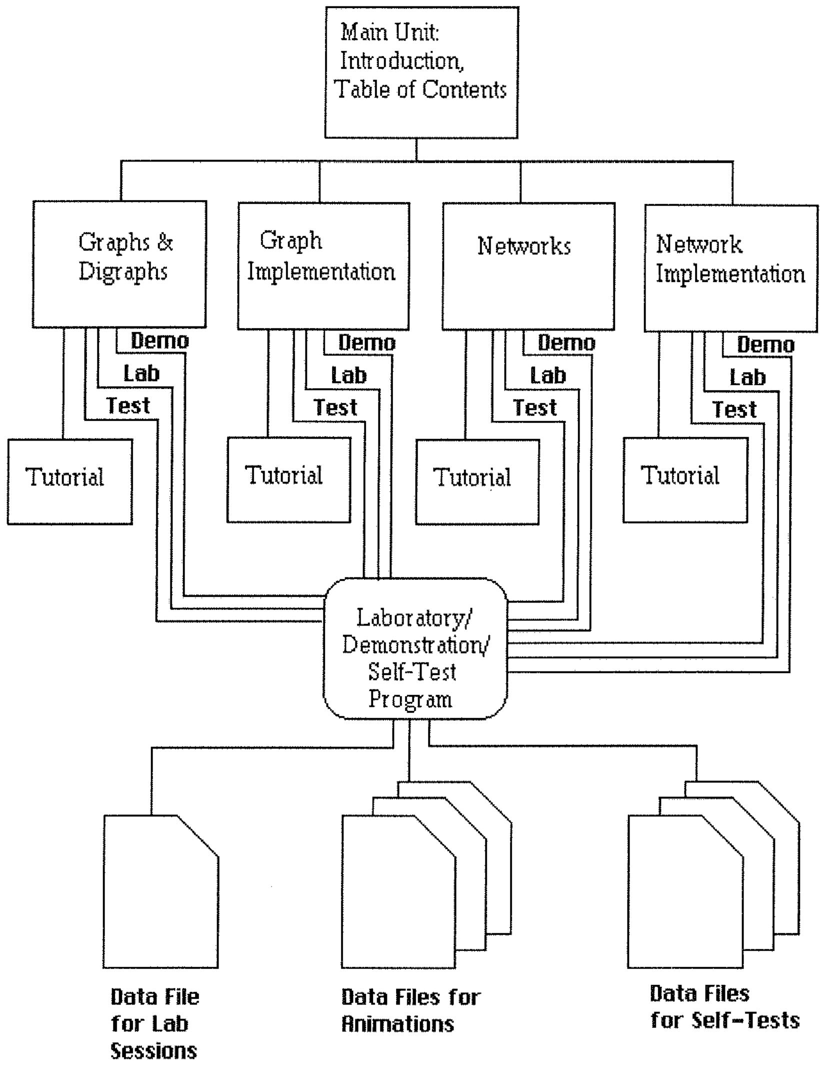
and user interface can be created much more quickly and easily with the authoring system.

NeoBook Professional was selected as the authoring system for several reasons including ease of use, support of all media types except video, the ability to call external programs, and the ability to compile stand-alone, royalty-free applications. QuickBASIC was selected primarily because its compiled programs would run within a window in the NeoBook-created screens without clearing the screen and the palette was more easily adapted to that selected for the NeoBook portion than for other languages such as Turbo Pascal. Also, NeoBook Professional is a DOS application, so windows-based languages such as Visual BASIC were not appropriate.

The implementation using NeoBook and QuickBASIC will include compiled NeoBook publications, compiled QuickBASIC programs, and data files used for lab sessions, interactive demonstrations, and self-tests. The specific implementation design is shown in Figure 2. In general, the structure in Figure 2 follows the functional design from Figure 1, although it emphasizes the implementation structure.

In Figure 2 square-edged boxes represent portions of the courseware which will be implemented using NeoBook, round-edged boxes represent portions to be implemented in QuickBASIC, and the boxes with a diagonal upper right corner represent data files. The following sections describe the design of the various parts of the courseware.

Figure 2: Courseware Implementation Structure


Main Unit: Introduction, Table of Contents
Graphs & Digraphs
Graph Implementation
Networks
Network Implementation
Demo
Lab
Test
Tutorial
Laboratory/ Demonstration/ Self-Test Program
Data File for Lab Sessions
Data Files for Animations
Data Files for Self-Tests

3.0 Individual Unit Designs

3.1 Main Unit (Introductory Section) and Other NeoBook-created Portions

All portions of the courseware to be created using NeoBook will be implemented according to the Functional Specification. The screen layouts including the use of text, images, buttons, and hypermedia links are given in the Functional Specification. Exact screen locations are not given for elements of screens because the screens will be implemented interactively using NeoBook. For example, a button can be created by clicking on the Button Tool in NeoBook, then dragging the button to its desired position and size. The position and size can be changed by selecting the button with a Selector Tool and redragging the button. Button color, text and action are indicated at the time the button is created.

Using an authoring tool such as NeoBook means that the functional design is adequate for doing the actual implementation of the portions which are being implemented with the authoring system. In addition to the screen layout specifications which indicate the placement of text, images, button, and hypermedia links, the overall structure diagram in Figure 1 of the preceding section indicates button actions by the links between boxes. For example, the link from the Title Screen box to the Introduction box is labeled START. This means that the START button indicated in the screen layout for the Title Page will have a button action which causes the Introduction Screen to be displayed.

3.1.1 Color Palette

The color palette used in the courseware was selected with several goals in mind. The use of color can be helpful in creating interest, for emphasis, and other uses. Nevertheless, if the use of color is overdone, it can be a distraction rather than a help. Color images which use alot of colors require more storage space and take longer to display than those which use fewer colors. For these reasons the color palette was restricted to sixteen colors. Many of the colors were selected simply because they produced a pleasant appearance. Others were selected with a specific purpose in mind. For example, to create the impression of three dimensional objects such as buttons, shades of gray, black and white were included. Table 1 gives the color palette to be used in the courseware.

Table 1: Courseware Color Palette

Color Number	Color	Special Use
0	Black	Background for text
1	Blue	
2	Yellow	Nodes, edges of graphs in the lab/demo/test program
3	Robin's Egg	
4	Red	
5	Light Gray	Light border color for 3D object
6	Medium Lt. Gray	
7	Medium Gray	Region Background Colors in program portion of courseware
8	Medium Dk. Gray	Dimmed Nodes on User Display, Dimmed Buttons (Inactive)
9	Dark Gray	Dark border color for 3D objects
10	Spring Green	
11	Light Red	Highlighted nodes, edges
12	Persimmon	Titles on pages in NeoBook-created portion of courseware
13	Cinnamon	
14	Vanilla	Main background in NeoBook-created portion of courseware
15	White	

3.2 Laboratory/Animated Demonstration/Self-test Program

The animated demonstrations, interactive laboratory sessions, and self-tests will be implemented using QuickBASIC. The decision to implement these features in this way was based on two factors: (1) NeoBook does not have the capability to implement these features easily, and (2) actual graph and network data structures and algorithms could be used to implement the demonstrations and lab sessions, making the experience more flexible and powerful in terms of what the student could do.

To simplify the programming, provide a consistent interface, and promote code reusability, it was decided to implement a single, customizable program. The program handles all user interface functions for the demonstrations, lab sessions, and self-tests using a common display and set of controls. Individual demonstrations, laboratory sessions or self-tests are called sessions and are implemented by customizing the program in two ways: (1) session-specific information is read from a sequential file, and (2) session-specific operations must be added to the program. These session-specific routines generally consist of custom actions for buttons, data structures and algorithms, and any auxiliary routines which might be needed.

Only one routine in the program must actually be changed to incorporate a new session. This routine consists of a large SELECT CASE statement which directs the program flow to the appropriate button handling. When a particular laboratory session, animated demonstration or self-test is selected on the NeoBook-created portion of the courseware, the laboratory/animation/self-test program is called with a command line parameter which indicates the particular session to be carried out. This parameter is used to access the appropriate data file and in the SELECT CASE statement to perform the correct button actions.

3.2.1 Main Logic

The main logic of the program will consist of some preliminary initialization, retrieving the session code from the command line, retrieving appropriate session setup data, creating the user display regions, and initializing the Node Pool, Edge Table, and graph. After these initializations, the title screen is displayed, the user display drawn, the session is set up, and the program goes into a loop retrieving user actions and carrying them out. The logic is shown below in pseudocode form.

```
Set screen mode to 640x480x16 colors
Set Color Palette
Retrieve Session Code from Command Line
If session is a lab then
        access lab data file and retrieve lab session data
else if session is a demonstration then
        access appropriate data file and retrieve demonstration session data
        create standard demonstration button set
        define standard demonstration node labels
        set up standard demonstration help
        initialize current frame to 0
else
        access appropriate data file and retrieve self-test session data
        create standard self-test button set
        define standard self-test node labels
        set up standard self-test help
        initialize current question to 0
create regions of user display
initialize Node Pool
initialize Edge Table
initialize graph adjacency matrix
display title screen
display user display
setup session
while not done
        get user choice
        if button pressed, carry out user choice
end while
display ending screen
```

3.2.2 Data Structures and Abstract Data Types

To facilitate implementation and reuse, a number of data structures and abstract data types are defined for the program. Some of these support the graphical user interface, such as the ButtonType type and Region type. Others implement the graph and network abstractions which are being demonstrated and studied. There are also data structures and routines which are used to implement the laboratory sessions, animated demonstrations, and self-tests.

3.2.3 User Interface Data Structures and Abstract Data Types

The data structures and abstract data types needed to implement the graphical user interface are defined in the following sections.

3.2.3.1 ButtonType Abstract Data Type

The ButtonType data type is used to implement a three-dimensional button for the user interface. A three-dimensional button is presented to the user by displaying a rectangle which has a light-colored border on the left and top sides and a dark-colored border on the right and bottom sides. This color scheme creates the impression of a rectangle which is raised from the plane of the screen. A ButtonType consists of the parameters listed below. The primitive data type of each parameter is given in parentheses following the description.

ButtonActive	: indicates if button is active in this session (boolean)
x1	: x coordinate of the top left corner of the button (integer)
y1	: y coordinate of the top left corner of the button (integer)
x2	: x coordinate of the bottom right corner of the button (integer)
y2	: y coordinate of the bottom right corner of the button (integer)
LightBorderColor	: color used for the light edge of a button (integer)
DarkBorderColor	: color used for the dark edge of a button (integer)
FaceColor	: color used for the face of the button (integer)
TextColor	: color for the button text (integer)
ButtonText	: text to appear on button (string)
ActionCode	: code specifying what action to take when button is pressed (integer)

The following operations can be performed on variables of type ButtonType. Operations are described using a procedure header format which gives the operation and the parameters needed by the procedure.

ButtonAction (B : ButtonType)

If a button is active, carry out the action specified in the field ActionCode.

ButtonCreate (B:ButtonType; x1,y1,x2,y2,Light,Dark,Face,TextCol:integer; ButtonText:string;Action,Active:integer)

Creates a button, B, with the specified parameters for location, colors, button text, action, and active status.

ButtonDraw (B:ButtonType)

Displays a button, B, using the parameters stored in the button's data structure including the location, colors, and button text.

ButtonHide (B:ButtonTpe)

Sets a button, B, to inactive. In active buttons are displayed as dimmed rectangles on the user display.

ButtonPress (B:ButtonType)

Displays a button with the light and dark colors swapped and the button text moved to the right and dimmed. This makes the button appear to be recessed into the plane of the screen and gives the impression that the button was pressed.

ButtonUnhide (B:ButtonType)

Sets a button, B, to active status. An active button is displayed as a three-dimensional object with the specified button text. When selected, the stored action will be carried out.

3.2.3.2 Region Abstract Data Type

A Region is a rectangular portion of the screen. It is used like a window to display certain information to the user. Regions can be fixed or can "popup" in response to an event. A region may have a three-dimensional border which makes it appear to be recessed into the plane of the screen. The user display for the laboratory/animation/self-test sessions consists of several regions as described in the Functional Specification. They include a Text Area, a Work Area, an Information Area, and a Message Area. A Region consists of the parameters listed below. The primitive data type of each parameter is given in parentheses following the description.

x1	: x coordinate of the top left corner of the Region (integer)
y1	: y coordinate of the top left corner of the Region (integer)
x2	: x coordinate of the bottom right corner of the Region (integer)
y2	: y coordinate of the bottom right corner of the Region (integer)
BGColor	: color used for the background of a Region (integer)

The following operations can be performed on variables of type Region. Operations are described using a procedure header format which gives the operation and the parameters needed by the procedure.

RegionBorder (R:Region)

Draws a three dimensional border around a region. The colors used for the border are the default BORDERDARK and BORDERLIGHT colors defined in constants at the beginning of the program.

RegionClear (R:Region)

Clears a Region, R, to the stored background color.

RegionConfirm (R:Region; Mesg:string; Response:boolean)

Pops up a region with a message and a place to click Yes or No in response to the message. Used generally to confirm an operation such as clearing a region or quitting the program. The message is one line and is printed in a fixed place in the region. It identifies the operation being confirmed. The additional message "ARE YOU SURE?" is added to the specified message. The response is returned.

RegionCreate (R:Region; x1,y1,x2,y2,BGColor:integer)

Creates a region, R, of type Region and stores the specified parameters including the location and background color.

RegionPrint (R: Region; RegionRow,RegionCol: integer; Mesg:string; Color:integer)

Prints the text in Mesg to the specified region, R. Printing occurs at the indicated row and column of the region. Within a region, rows and columns are numbered starting at 1.

3.2.3.3 Button Table

In addition to the abstract data types for buttons and regions described one other major data structure is used to support the graphical user interface, the Button Table. The Button

Table is a one-dimensional array with each element corresponding to a button on the user display. Each element, a ButtonType, includes relevant information such as the position of the associated button, whether the button is active or not, and the action code to indicate what action to take when the button is pressed. The Button Table is dimensioned for elements 1 to MAXBUTTONS. MAXBUTTONS is a defined constant.

3.2.4 Graph and Network Data Structures and Abstract Data Types

A graph consists of a set of nodes and a set of edges. The edges indicate connections between pairs of nodes. In general, nodes and edges for this application have attributes which include information used to display graphs on the user display. Nodes are displayed as circles with a label within the circle and edges are displayed as lines. In the laboratory/demonstration/self-test program, a complete set of nodes is created during program initialization. The set of nodes forms a Node Pool from which graphs can be constructed either according to information specified for a particular lab session or demonstration, or by the user. Nodes are initially not visible. When a node is added to a graph, the graph data structures are updated and the node is made visible on the user display.

Edges are created as needed and stored in a one dimensional array called the Edge Table. This structure supplements the adjacency matrix which also records connections between nodes. The information in the Edge Table encapsulates attributes of the edges such as the nodes being connected, and the color of the edge.

An adjacency matrix is used to record connections between nodes in a graph or network. The matrix is a two-dimensional array with each dimension ranging over all nodes in the Node Pool. Nodes connected by an edge are indicated by placing a nonzero value in the array element corresponding to the two nodes. For a graph a one is used to indicate an edge; for a network the value represents the weight of the edge. In both cases a zero indicates no connection.

The data structures and abstract data types needed to implement the graph and network abstractions are defined in detail in the following sections.

3.2.4.1 GraphNode Abstract Data Type

Nodes are represented by the data type GraphNode. A GraphNode is a node in a graph or network and has the attributes shown below. The primitive data type of each attribute is given in parentheses following the description.

Label	: identifier displayed in a node on the user display (character)
NodeColor	: the display color of the node on the user display (integer)
LabelColor	: the display color of the node label on the user display (integer)
NodeXCoord	: the x coordinate of the node's center (integer)
NodeYCoord	: the y coordinate of the node's center (integer)
NodeVisible	: indicates if node is visible (boolean)

The following operations can be performed on variables of type GraphNode. Operations are described using a procedure header format which gives the operation and the parameters needed by the procedure.

NodeCenter (Node: GraphNode; x,y:integer)

Given the index of the node in the Node Pool in the range 1..MAXNODES, return the x,y coordinates of the center of the center of the circle used to represent the node on the user display. This routine is used in creating nodes as initializing the Node Pool.

NodeChangeColor (Node: GraphNode; Color:integer)

Set the display color for a node to a new value. Used to change a node's display color for highlighting the node for some purpose such as indicating that the node has been visited during a traversal.

NodeCreate (Node:GraphNode; Label:String; NodeColor,LabelColor,x,y:integer)

Creates an entry in the Node Pool with the specified attributes. Nodes in the Node Pool are not part of a graph until a node is added to a graph and made visible. Therefore, the created node is initially set to be invisible.

NodeHide (Node:GraphNode)

Hides a GraphNode on the user display. The node is displayed dimmed to indicate its position but is not part of a graph. The node is shown at the coordinates defined when it was created. The NodeVisible attribute is also changed to FALSE.

NodeShow (NodeIndex: integer)

Displays a GraphNode on the user display. The node is displayed in the color stored for the node at the coordinates defined for the node when it was created. These coordinates are used to compute the row and column for displaying the node label. The NodeVisible attribute is also changed to TRUE.

NodeShowDimmed (Node:GraphNode)

Displays a dimmed version of a node in the work area of the user display. The user clicks on the dimmed version of the node to add it to a graph.

3.2.4.2 GraphEdge Abstract Data Type

Edges are represented by the data type GraphEdge. An edge in a graph is indicated in the adjacency matrix for the graph, but a table of edges is also kept to allow the current edge color to be stored. The edge color will change during traversals and other algorithmic manipulations. The entries in the Edge Table are of type GraphEdge and have the attributes shown below. The primitive data type of each attribute is given in parentheses following the description.

EdgeColor	: the display color of the edge on the user display (integer)
Node1	: Index in Node Pool of first node connected by edge (integer)
Node2	: Index in Node Pool of second node connected by edge (integer)

The following operations can be performed on variables of type GraphEdge. Operations are described using a procedure header format which gives the operation and the parameters needed by the procedure.

EdgeAdd (G:Graph; Node1,Node2,Weight: integer)

Adds a bi-directional edge to a graph by setting the appropriate elements in the adjacency matrix for the graph, G. Node1 and Node2 are the indexes in the Node Pool of the nodes to be connected and correspond to the row and column indexes in the adjacency matrix for the corresponding nodes. Weight is the edge weight.

EdgeArrowHead (Edge:GraphEdge)

Edge is a directed edge from Edge.Node1 to Edge.Node2 (fields in the GraphEdge data structure). The point of the arrowhead is first computed from the relative positions of Node1 and Node2, then the sides are drawn and the arrowhead is filled in.

EdgeChangeColor (Edge:GraphEdge; Color:integer)

Set the edge color to the specified value.

EdgeCreate (Edge:GraphEdge; EdgeColor,Node1,Node2,Weight:integer)

Adds an edge to the Edge Table. The current color and the two nodes connected by the edge are recorded in the entry. Edge is an available edge in the Edge Table found using the FindFreeEdge routine. Weight is the edge weight for a network edge.

EdgeDelete (G:Graph; Node1,Node2:integer)

Remove an edge from a graph, G, by setting the corresponding elements in the adjacency matrix to zero.

EdgeFind (EdgeTable: array of graphEdge; Node1,Node2,EdgeIndex:integer)

Given the indexes of two nodes, return the index in the EdgeTable which connects the two nodes. If the two specified nodes are not connected by an edge, return a 0 (FALSE).

EdgeFindFree (EdgeTable:array of GraphEdge; EdgeIndex:integer)

Unused entries in the EdgeTable are marked with a -1 in the EdgeColor field. This routine searches for the first unused entry and returns its index, EdgeIndex. Since the graph is restricted in size, the maximum number of edges are allowed for in the EdgeTable. Thus, the search cannot exceed the table size; it simply searches until an unused entry is found.

EdgeHide (Edge:GraphEdge)

Remove an edge from the Edge Table and clear it from the user display. Clearing is accomplished simply by redisplaying the graph without the removed edge.

EdgeHighlight(Edge:GraphEdge; NodePool:array of GraphNode; Color:integer)
Similar to EdgeShow, except the outer portion of the edge is displayed in the specified color to highlight the edge.

EdgeShow (Edge:GraphEdge; NodePool: array of GraphNode)
Make an edge visible on the user display. The indexes of the two nodes connected by the edge are kept in the fields Node1 and Node2. These are used to access the Node Pool to determine the screen locations of the nodes. Edges are drawn as a thick line consisting of five lines from the center of one node to the center of the other node. The nodes are then redisplayed.

3.2.4.3 Node Pool

The Node Pool is a one-dimensional array with each element corresponding to a GraphNode. There are MAXNODES nodes in the Node Pool, each node representing a displayable node in the work area of the user display. Node Pool entries are initialized at the start of the program to specify the location, color, associated label, etc. Initially, all nodes are hidden, that is, displayed as a dimmed image. As nodes are added to a graph, either by the user or by the program, the corresponding GraphNode is shown on the user display. The Node Pool keeps track of the state of individual nodes.

The following operations can be performed on the Node Pool. Operations are described using a procedure header format which gives the operation and the parameters needed by the procedure.

NodePoolInit (Nodes: array of GraphNode)
Initializes all entries of the Node Pool to default attributes. The label for each node is set according to a label string specified for the particular session. Colors are set to default

values, the center is computed using the NodeCenter routine and stored, and NodeVisible is set to FALSE.

3.2.4.4 Edge Table

The Edge Table is also a one-dimensional array, dimensioned to hold MAXEDGES entries, with each entry representing an edge in a graph. MAXEDGES represents the maximum number of edges possible for a graph displayed in the user area. When an edge is added to a graph by the user or by the program, an entry is made in the adjacency matrix used to represent the graph and the edge is recorded in the Edge Table.

The following operations can be performed on the Edge Table. Operations are described using a procedure header format which gives the operation and the parameters needed by the procedure.

EdgeTableInit (EdgeTable:array of GraphEdge)

Sets all entries in the EdgeTable to unused, by setting the edge color to -1.

3.2.4.5 Graph/Network Abstract Data Type

A graph or network is represented by a two-dimensional array called an adjacency matrix. and a list of nodes. Two nodes which are connected by an edge are represented by a nonzero value stored in the corresponding row/column elements of the array. For a network the value will be the edge weight. Note that there will be two such entries for each edge in an undirected graph. The adjacency matrix is dimensioned for the maximum number of nodes, MAXNODES.

The following operations can be performed on variables of type Graph. Operations are described using a procedure header format which gives the operation and the parameters needed by the procedure.

GraphBreadthFirstTraversal (G:Graph; StartIndex:integer)

Performs a breadth-first traversal on the graph, G, starting at the node having the index indicated by StartIndex.

GraphDepthFirstTraversal (G:Graph; StartIndex:integer)

Performs a depth-first traversal on the graph, G, starting at the node having the index indicated by StartIndex.

GraphEmpty (G:Graph):boolean

This function, returns a TRUE if a graph, G, is empty, or a FALSE if it is not.

GraphHighlightMatrix (G:Graph; Rwo,Col:integer)

Highlights a column in the adjacency matrix.

GraphInit (G:Graph)

Takes a graph, G, in an unknown state and initializes it to an empty graph in which all edge connections are set to zero

GraphSearchFrom (NodeIndex:integer)

This is the main routine of the depth-first traversal. The routine is recursive, using the system stack to process the graph in a Last In First Out manner.

GraphShow (NodePool:array of GraphNode; EdgeTable:array as GraphEdge)
Show the graph represented by the NodePool and EdgeTable on the user display. Nodes to be displayed are in the Node Pool with the NodeVisible attribute set to TRUE. Edges in the Edge Table are included if the EdgeColor is not -1 (indicates unused).

GraphShowMatrix (G:Graph)
Displays the adjacency matrix for the graph which is in the work area. A special window is used for this display.

NetAllIncluded (Visited:array of boolean)
Returns True if all elements of Visited are True, indicating that all nodes have been included by the algorithm processing the network.

NetFindMin(N:Graph; i,j:integer)
Given a network, N, return the indexes of the two nodes, i and j, such that I is included in the minimum spanning tree forthe network, j is not included, and for which the edge weight is a minimum.

NetMinSpanTree (N:Graph)
Finds the minimum spanning tree for the network, N, using Prim's algorithm. The spanning tree will be highlighted on the network on the user display.

NetShortestPath (N:Graph; Node1,Node2:integer)
Finds the shortest path between the two specified nodes in the network, N, using Dijkstra's Algorithm. The shortest path will be highlighted on the network on the user display.

NetShowShortestPath (Path: array of integer; Start, Dest:integer)

Recursively highlights the nodes and edges in the shortest path from the Start node to the Dest node as determined by the Shortest Path algorithm.

NetTopologicalSort (N:Graph)

Performs a topological ordering of the nodes in the network, N. The ordering will be displayed as a list in the information area of the user display.

3.2.4.6 Auxiliary Data Structures and Operations for Graphs and Networks

In addition to the main data structures and operations needed to implement graphs and networks, several auxiliary data structures and operations are also used. These are described below.

The Visited array is a one-dimensional array which is used to keep track of nodes in a graph or network which have been processed by some algorithm such as the depth-first traversal. A simple queue, implemented as a one-dimensional array with associated Head and Tail variables is used in doing the breadth-first traversal, along with a one-dimensional array which keeps track of which nodes have already been processed through the queue.

Several routines are used to manipulate the queue. These operations are described below using a procedure header format which gives the operation and the parameters needed by the procedure.

QueueAdd (Q:Queue; Head,Tail,Item:integer)

Adds an item to the queue, adjusting head and tail variables as needed.

QueueEmpty (Q:Head,Tail:integer)

A function which returns TRUE if a queue, Q, is empty, or FALSE, if it is not empty.

QueueInit (Head,Tail:integer)

Initialize a queue represented by its Head and Tail variables, to empty.

QueueRemove (Q:Queue; Head,Tail, Item:integer)

Remove an item from the Head of a Queue, Q, and return it in the variable, Item.

3.2.5 Application Data Structures and Operations

A number of data structures and operations are needed to implement the laboratory sessions, animated demonstrations, and self-tests. Each execution of the program implements a particular session, for example the animated demonstration for graphs and digraphs. The data structures and the operations described below support sessions in general. In addition, there will be session-specific operations which implement specific laboratory sessions, demonstrations, and self-tests.

3.2.5.1 Data Structures and Operations for General Session Support

The following data structures are used to implement the general session-handling capability of the program. Each data structure is described briefly.

Session	: holds the identifier for the particular session (string)
SessionDescr	: a one-dimensional array of strings which holds the description for the particular session. The array is dimensioned to hold the maximum number of lines available in the Text Area of the user display.
SessionHelp	: a one-dimensional array of strings which holds help information

	for the particular session. The array is dimensioned to hold one less than the maximum number of lines available in the Text Area of the user display to allow for a "Click to continue..." prompt.
NodeLabel	: a string of the labels to be used for the various nodes. The first character in the string is used for the node with index 1, etc.
NumFrames	: holds number of frames for an animated demonstration (integer).
Frame	: hold the number of the current frame in a demonstration (integer).
FrameText	: a two-dimensional array of strings which holds the lines of text for each frame in an animated demonstration. Each question may have up to the maximum number of lines available in the Text Area of the user display
NumQuestions	: holds the number of questions for a self-test (integer).
QuestionNum	: holds the number of the current question for a self-test (integer).
Question	: a two-dimensional array of strings which holds the lines of text for each question in a self-test. Each question may have up to the maximum number of lines available in the Text Area of the user display.
Answer	: a one-dimensional array of characters giving the correct answer for each question in a particular self-test.
NodeList	: a two-dimensional array used to specify a list of node indexes in two graphs. Used to implement examples and solutions in sessions.
NumNodes	: A two-dimensional array specifying the number of nodes in the lists in NodeList.
EdgeList	: a three-dimensional array used to specify a list of node pairs for indicating edges in two graphs. Used in conjunction with NodeList to implement examples and solutions in sessions.

NumEdges	: A two-dimensional array specifying the number of edges in the lists in EdgeList.

The following operations implement general session support. Operations are described using a procedure header format which gives the operation and the parameters needed by the procedure.

DemoGetSessionData (Code: string; Descr: array of string; NumFrames:integer; FrameTest: array of array of strings)

Retrieves session-specific data for an animated demonstration. The data for each demonstration is contained in a sequential file. The routine uses Code to access the correct file, reads the session description, reads the number of frames, and loads the frame text for each frame. If there are graphs used in the demonstration, the routine reads the data for the node list and edge list for up to two graphs.

DemoRun

Run an animated demonstration. Data for the demonstration, including the text for each frame is in global variables. The number of frames in the demo is in the variable NumFrames. The logic of each frame is implemented with a separate subroutine. DemoRun will call the first frame's subroutine and will exit when the subroutine returns. The user controls are used to progress through the demonstration. If the NEXT button is pressed, the Frame variable is incremented by one and DemoRun is called again. Using Frame, DemoRun calls the subroutine for the next frame.

Demoxxxx

Individual frames of each animated demonstration are implemented by specific subroutines. The are as follows:

DemoGim1 - DemoGim3: Frames 1 - 3 of the Graph Implementation demonstration.

DemoGraph1 - DemoGraph5: Frames 1 - 5 of the Graphs Concepts demonstration.

DemoNet1 - DemoNet4: Frames 1 - 4 of the Networks Concepts demonstration.

DemoNim1 - DemoNim5: Frames 1 - 5 of the Networks Implementation demonstration.

LabButtonService (B:ButtonType)

Services a button press by calling the appropriate subroutine depending on the button's ActionCode. This routine is application-specific. When new sessions are added to the program, appropriate action codes must be defined to indicate session-specific button actions, corresponding action routines must be added to the program, and the SELECT CASE statement in this routine must be modified to incorporate the new button action and action routine.

LabCheckBalsa (G:Graph)

Check button action for the Balsalines problem. Checks the graph built by the user against the correct solution.

LabCheckNet (G:Graph)

Check button action for the network errand problem. Checks the network built by the user against the correct solution.

LabDoAddEdge (NodePool:array of GraphNode)

Adds an edge to the graph and displays it on the user display.

LabDoAddNode (NodePool: array of GraphNode)

Adds a node to the graph and displays it on the user display.

LabDoClearGraph (NodePool:array of GraphNode;
EdgeTable:array of GraphEdge; G:Graph)

Resets the graph and clears it from the user display.

LabDoHelp

Displays help for a particular session in the Text Area and prompts the user to continue. When the user responds, the routine redisplays the associated session description.

LabDoMinSpan (G:Graph, NodePool: array of GraphNode, EdgeTable: array of GraphEdge)

Carry out the Minimum Spanning algorithm for the network, G.

LabDoShortestPath (G:Graph, NodePool: array of GraphNode, EdgeTable: array of GraphEdge)

Carry out the Shortest Path algorithm for the network, G.

LabDoTopSort(G:Graph, NodePool: array of GraphNode, EdgeTable: array of GraphEdge)

Carry out the Topological Sort algorithm for the network, G.

LabDoTraversal (G;Graph; NodePool: array of GraphNode)

Carries out a traversal laboratory session. Common actions are performed, then the traversal indicated by the laboratory session code is called.

LabDrawLabScreen

Draws the user display including all regions and buttons.

LabGetEdgeWeight (Weight: integer)

Show edge colors and prompt for user selection.

LabGetNodeSelection(NodePool:array of GraphNode; NodeIndex:integer)

Waits for the user to select a node in the work area and returns the index in the NodePool of the selected node.

LabGetSessionData (Code:string;Descr, Help:array of string; ButtonTable:array of ButtonType; Nodelabel:string)

Retrieves the appropriate data from the lab data file LABDATA.DAT and stores the data for use by the program. The session code is used to access the correct data in the file.

LabGetUserAction (Choice:integer)

Waits for a mouse click, then searches the Button Table to see which button, if any, was selected. Returns the index in the Button Table of the selected buton, or zero if an active button was not selected.

LabResetWorkRegion

Clears the work area of the user display and shows the dimmed nodes.

LabSetPalette (Colors: array of single precision)

Sets the palette to that used in the NeoBook portion of the courseware.

LabSetupBalsa

Display nodes for Balsa airlines laboratory session.

LabSetUpGraph (G:Graph; GraphNum:integer; NumNodes:array of integer; NodeList: array of integer; NumEdges: array of integer; EdgeList: array of integer)

Sets up a graph based on specified nodes and edges. GraphNum indicates which of the two lists of nodes and edges to use.

LabSetupMatrix (G:Graph)

Clear and display the matrix window and display the adjacency matrix for the graph, G.

LabSetupNet

Displays nodes for network errand laboratory session.

LabSetupSession (Session:string)

Performs any special actions at the start of a particular session. Session is used to determine which session is being initiated.

LabShowWeights

Displays the color key for edge weights for networks in the lower part of the work Area. The constants WGHTX1 and WGHTY1 are coordinates selected to simplify displaying the color boxes.

LabTheEnd

Prints a closing message for the session.

LabTitleScreen

Prints an opening title screen.

TestCheck (FrameText:string,Ans: array of string,NumFrames: integer)

Check user's answers to the self-test and indicated those which are not correct. Display the correct answer for incorrect problems.

TestClear (Ans: array of string, NumFrames: integer)

Clear user-entered answers for a self-test.

TestGetSessionData(Code:string, NumFrames:integer, Text, Ans:array of string, NumNodes:integer)

Retrieve data for a self-test from the data file.

TestRun

Run a user self-test. Data for the self-test, including the text for each question is in global variables. The number of questions in the self-test is in the variable NumFrames.

TestShowAns (Ans: array of string, Frame:integer)

Display multiple choice answers and the user's current answer (initialized to a space) in the Information Region of the display.

TestShowQuestion (Text: array of string, Frame:integer)

Display the question for the associated frame from the Text array in the Text Region of the display.

3.2.6 General Purpose Routines

The following routines are general in nature.

XColorValue (Red,Green, Blue: single precision)

This function returns a single precision value corresponding to the amount of red, green and blue which is indicated for a particular color.

XDelay (Seconds: single precision)

Delays the specified amount of time in seconds. The time is read from the system clock so that the delay should not be affected by CPU speed. The routine does not account for time passing midnight.

XPrintText (Row,Col:integr, Mesg:string; Color:integer)

Prints text at a specified row and column, and in a specified color.

3.2.7 Mouse Handling Routines

Mouse support is provided by a set of mouse handling routines written for a Microsoft or Logitech-compatible mouse driver.

Mouse (Fcn,Arg1,Arg2,Arg3:integer)

This routine provides the main interface to the mouse driver. A set of specific mouse routines are also provided to perform specific actions such as showing the mouse. These specific routines call this routine with appropriate values for the parameters and operate on the results which are returned.

The driver is called by doing a CALL interrupt for interrupt &H33. Before calling the driver, the mouse function (Fcn) is stored in the AX register, and other arguments as needed (Arg1...Arg3) are stored in registers BX, CX, and DX respectively. Results are returned in these same registers. Specific arguments and results are described in the specific routines which call this routine.

MouseButtonPressed

This routines calls the Mouse routine with a function 3 (Get Mouse Position and Status). The mouse driver returns the button status in BX, the current horizontal cursor position in CX, and the current vertical cursor position in DX. The button status is such that 1=left button pressed, 2=right button pressed, 4=middle button pressed, 0=no button pressed.

This routine simply checks for BX > 0 (some button pressed) or 0 (no button pressed) and returns the corresponding result.

MouseGet Pos (x,y:integer)

This routines calls the Mouse routine with a function 3 (Get Mouse Position and Status). The mouse driver returns the button status in BX, the current horizontal cursor position in CX, and the current vertical cursor position in DX. The button status is such that 1=left button pressed, 2=right button pressed, 4=middle button pressed, 0=no button pressed.

This routine returns the horizontal position (0<=x<640) and the vertical position (0<=y<200). Mouse coordinates are always in this range regardless of screen mode.

MouseHide

This routine calls Mouse with a function 2 (Hide Cursor). No results are returned.

MouseReset

This routine calls Mouse with a function 0 (Mouse Reset). The mouse driver checks for the existence of the mouse and resets it if found. If not found, the AX is set to -1. This routine does not check for mouse not found.

MouseShow

This routine calls Mouse with a function 1 (Show Cursor). No results are returned.

WaitForMouseClick (x,y:integer)

Calls MouseButtonPressed in a loop until TRUE is returned, then loops until MouseButtonPressed is FALSE, thus indicating that a button has been pressed and released. Calls MouseGetPos to get and return the current coordinates.

WaitForMousePress (x,y:integer)

Calls MouseButtonPressed in a loop until TRUE is returned, then calls MouseGetPos to get and return the current coordinates.

3.2.8 File Formats

Session-specific data for laboratory sessions, animated demonstrations and self-tests are stored in sequential files. Data for all laboratory sessions are stored in a single file

LABDATA.DAT. Data for animated demonstrations and self-tests are stored in specific files. The file names for these files consist of the session code concatenated with the extension ".DAT" Table 2 gives the session codes for all sessions.

Table 2: Session Codes for Laboratories, Demonstrations, and Self-Tests.

Code	**Type**	**Description**
GRAPHLAB	laboratory	Graph Unit, Lab 1: Build a graph
DEPTHLAB	laboratory	Graph Unit, Lab 2: Depth-first traversal
BREADTHLAB	laboratory	Graph Unit, Lab 3:Breadth-first traversal
NETLAB	laboratory	Network Unit, Lab 1: Build a network
PATHLAB	laboratory	Network Unit, Lab 2: Shortest Path
SPANLAB	laboratory	Network Unit, Lab 3: Min. Spanning Tree
SORTLAB	laboratory	Network Unit, Lab 4: Topological Sort
GIMP01LAB	laboratory	Graph Implementation Unit, Lab 1
NIMP01LAB	laboratory	Network Implementaton Unit, Lab 1
GRPHDEMO	demo	Graphs & Digraphs Demonstration
GIMPDEMO	demo	Graph Implementation Demonstration
NETDEMO	demo	Networks Demonstration
NIMPDEMO	demo	Networks Implementation Demonstration
GRPHTEST	self-test	Graphs & Digraphs Demonstration
GIMPTEST	self-test	Graph Implementation Demonstration
NETTEST	self-test	Networks Demonstration
NIMPTEST	self-test	Networks Implementation Demonstration

3.2.8.1 Lab Session Data File Format

The lab session data file, LABDATA.DAT, contains data for all lab sessions. Data for individual sessions is stored consecutively, with the data for each session having the format shown below.

```
Session Code (string)
Number of Lines of  Session Description (integer)
Description Line 1 (string)
          :
Description Line n (string)
Number of Lines of Help (integer)
Help Line 1 (string)
          :
Help Line n (string)
Node Labels (string)
Button 1 text,Text Color, x1, y1, x2, y2, action code, active
          :
Button 12 text,Text Color, x1, y1, x2, y2, action code, active
Number of nodes used
```

The number of lines of description is limited to MAXDESCR and the number of lines of help is limited to MAXHELP. In the button information x1,y1 are the coordinates of the top left corner of the button, and x2,y2 are the coordinates of the bottom right corner of the button.

3.2.8.2 Animated Demonstration Data File Format

Animated demonstrations are implemented using session-specific data and session-specific routines. The data for animated demonstrations can include a session description, text for each frame of the demonstration, and up to two graph specifications for graphs to be used in the demonstration. Each demonstration has a separate data file named XXXXXXXX.DAT, where XXXXXXXX is the same as the session code for the demonstration (see Table 2 above). The format of each animated demonstration data file is shown below.

```
Number of Lines of Description (integer)
Description Line 1 (string)
          :
Description Line n (string)
Number of Frames (integer)
Number of Lines of Text for Frame 1 (integer)
Text Line 1 for Frame 1 (string)
          :
Text Line n for Frame 1 (string)
Number of Lines of Text for Frame 2 (integer)
Text Line 1 for Frame 2 (string)
          :
Text Line n for Frame 2 (string)
          :
Number of Graphs (maximum of two)
Number of Nodes for Graph 1 (integer)
Node 1 Index (integer)
   :
Node n Index (integer)
Number of  Edges for Graph 1 (integer)
Node1 Index, Node 2 Index pair 1,EdgeWeight 1
          :
Node1 Index, Node 2 Index, pair n, EdgeWeight n
          :
Number of nodes used
```

Help will be the same for all demonstrations and is hard-coded. A standard button configuration is also used for each demonstration.

3.2.8.3 Self-Test Data File Format

Self-tests are implemented using session-specific data and a standard set of routines, buttons, and help. The data for self-tests can include a session description, questions, answers and correct answers for up to MAXQUESTIONS questions, and up to two graph specifications for graphs to be used in the self-test. Each self-test has a separate data file named XXXXXXXX.DAT, where XXXXXXXX is the same as the session code for the self-test (see Table 2 above). The format of each self-test data file is shown below.

```
Number of Lines of Description (integer)
Description Line 1 (string)
        :
Description Line n (string)
Number of Questions (integer)
Number of Lines of Text for Question 1 (integer)
Text Line 1 for Question 1 (string)
        :
Text Line n for Question 1 (string)
Text for Question 1, Answer 1 (string)
        :
Text for Question 1, Answer 4 (string)
        :
Correct Answer for Question 1 (character)
Number of Lines of Text for Question 2 (integer)
Text Line 1 for Question 2 (string)
        :
Number of Graphs (maximum of two)
Number of Nodes for Graph 1 (integer)
Node 1 Index (integer)
  :
Node n Index (integer)
Number of  Edges for Graph 1 (integer)
Node1 Index, Node 2 Index pair 1
        :
Node1 Index, Node 2 Index, pair n
        :
Number of nodes used
```

Questions are intended to be multiple choice.

4.0 System Requirements

The courseware will be designed to run on an IBM PC compatible computer which has the following minimum characteristics:

- IBM PC compatible computer, 386 or better
- 14" SVGA monitor
- 530 Kbytes free conventional memory
- hard disk with 2 Mbytes free space
- Logitech or Microsoft-compatible mouse with driver
- MS-DOS 5.0 or higher

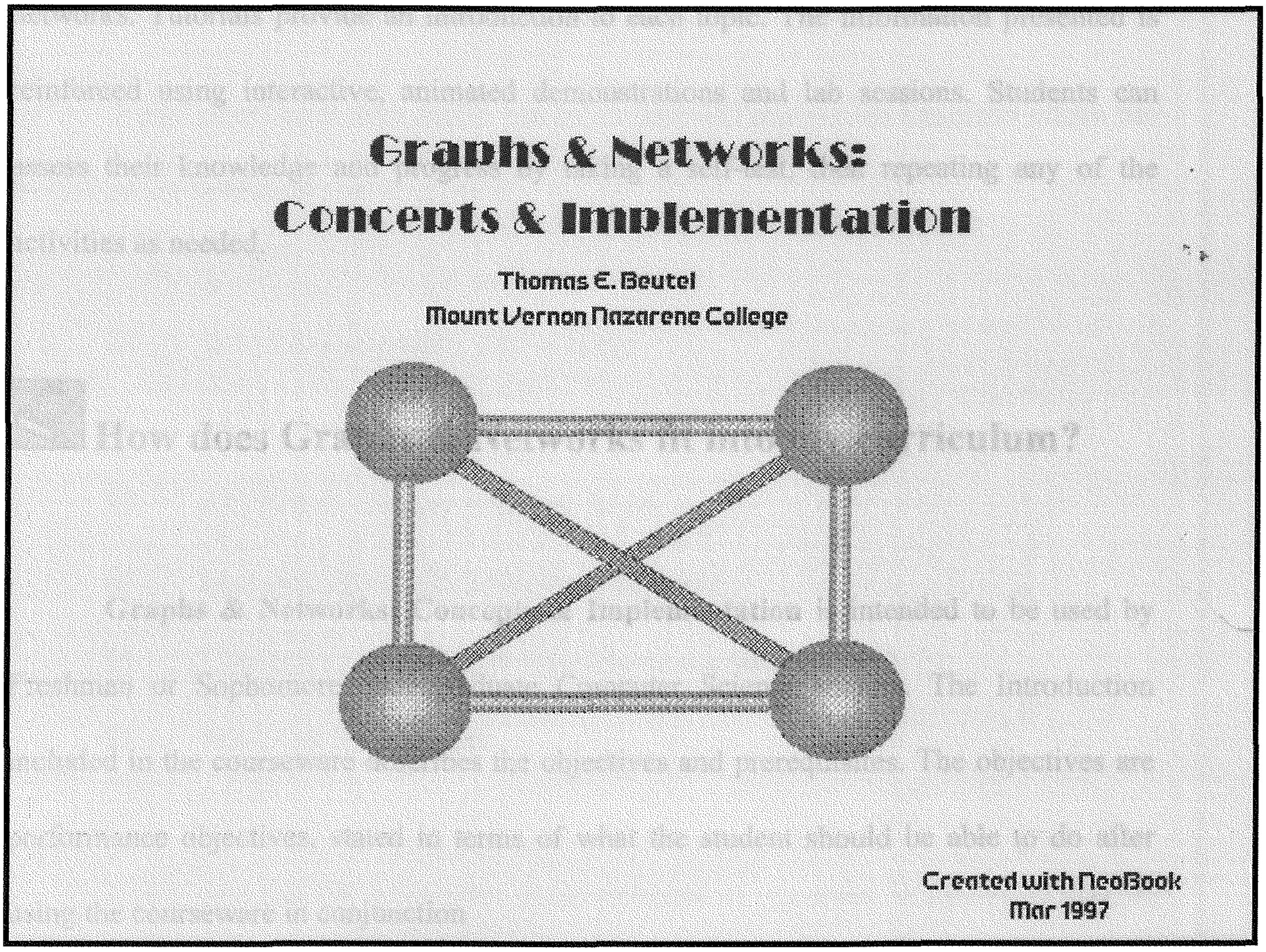
The courseware is not designed to run under Windows. It may be run as a DOS application from Windows as long as all minimum requirements are met.

Appendix E

Instructor's Guide

Graphs & Networks: Concepts & Implementation

Instructor's Guide



Graphs & Networks:
Concepts & Implementation

Thomas E. Beutel
Mount Vernon Nazarene College

Created with NeoBook
Mar 1997

What is the Graphs & Networks courseware?

Graphs & Networks: Concepts & Implementation is an interactive courseware product which combines text, color graphics, animation, and user interaction to present the concepts and implementation principles for the nonlinear data structures of graphs and networks. Tutorials provide an introduction to each topic. The information presented is reinforced using interactive, animated demonstrations and lab sessions. Students can assess their knowledge and progress by taking a self-test, then repeating any of the activities as needed.

How does Graphs & Networks fit into my curriculum?

Graphs & Networks: Concepts & Implementation is intended to be used by Freshman or Sophomore undergraduate Computer Science majors. The Introduction included in the courseware describes the objectives and prerequisites. The objectives are performance objectives, stated in terms of what the student should be able to do after using the courseware in conjunction

with other learning activities such as lectures and programming assignments.

Graphs & Networks is intended to supplement your normal instruction. It can be assigned to all students to facilitate their understanding of graph, digraph and network data structures and associated algorithms, or it can be used for those students who are having difficulty understanding these topics. You might even assign parts of the courseware prior to class sessions and devote the class time to clearing up lingering problems or elaborating on specific topics.

System Requirements

The courseware is designed to run on an IBM-PC compatible computer which has the following minimum characteristics:

- ☑ IBM PC-compatible, 386 or better
- ☑ 14” SVGA monitor
- ☑ 530 Kbytes free conventional RAM
- ☑ hard disk drive with 2 Mbytes free space
- ☑ Logitech or Microsoft-compatible mouse with driver
- ☑ MS-DOS 5.0 or higher

Please note that the courseware is not designed to run under Windows. It may be run as a DOS application from Windows as long as all minimum requirements are met.

Installing and Running Graphs & Networks

The software for **Graphs & Networks** is included on the floppy disk which accompanies this guide. It is a good idea to create a backup copy of the courseware disk before installing the software. To create a backup copy of the diskette, perform the following steps:

☑ Set the write protect tab on the courseware disk to ON.

☑ Use the DOS **diskcopy** command to copy the courseware disk to another floppy disk.

☑ The courseware disk includes an installation program which will **install Graphs & Networks** onto your hard drive. To run the installation program and install the courseware, proceed as follows:

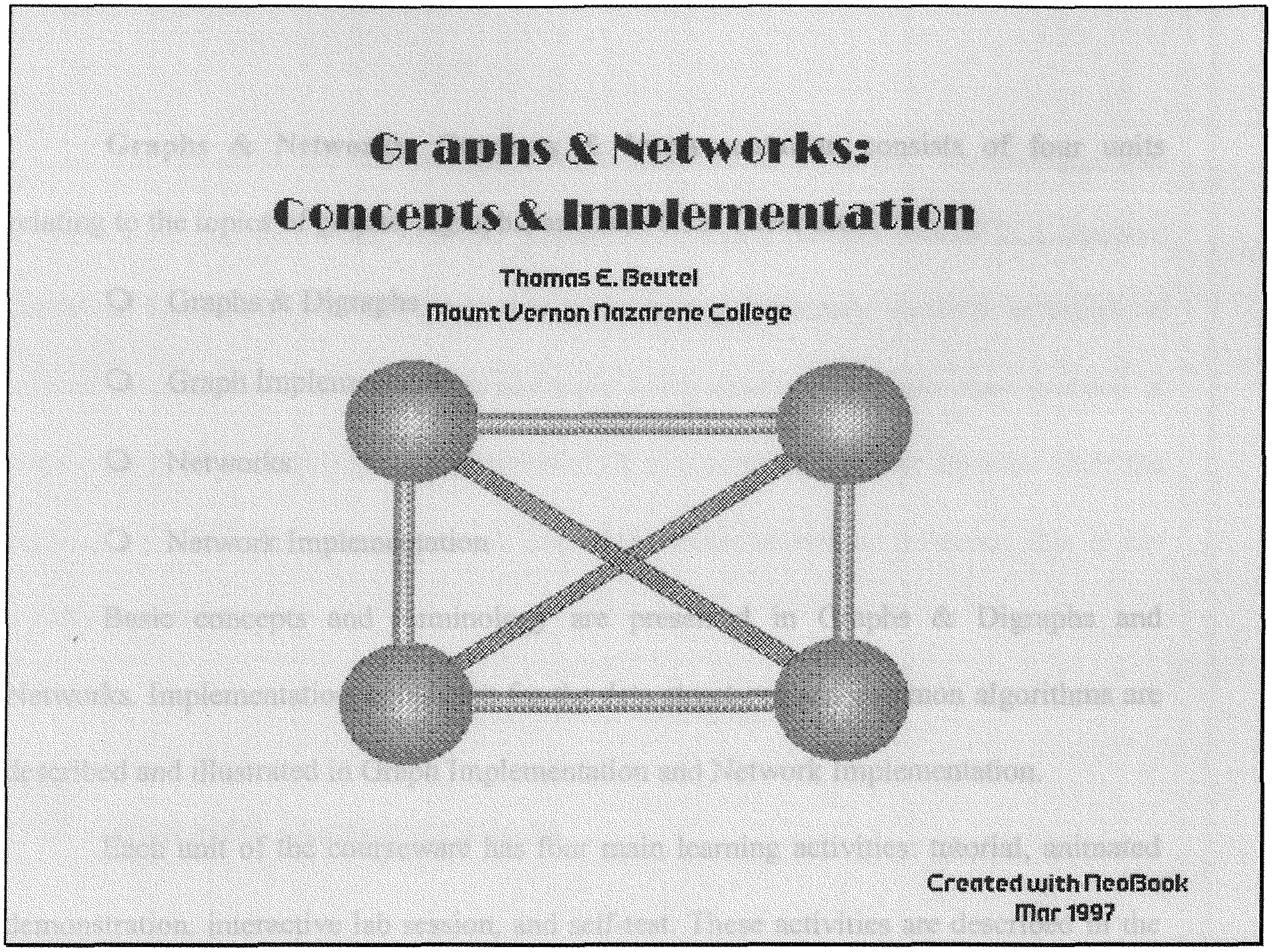
☑ Turn on your computer.

☑ Insert the distribution disk into **Drive A:**

☑ At the DOS prompt (usually C:>), type **A:INSTALL** and press **ENTER**

The installation program creates a directory on your hard drive called GRAPH-SW. All files for the courseware are placed in that directory. To start the courseware, change to the directory GRAPH-SW by typing the DOS command **cd c:\graph-sw** and pressing **ENTER**. Once in the proper directory, start the courseware by typing **graphs** and pressing **ENTER**. The **Graphs & Networks** opening screen shown in Figure 1 will be displayed. To begin, click on the START button.

Figure 1: Graphs & Networks Opening Screen



Graphs & Networks:
Concepts & Implementation

Thomas E. Beutel
Mount Vernon Nazarene College

Created with NeoBook
Mar 1997

Courseware Organization

Graphs & Networks: Concepts & Implementation consists of four units relating to the topics of graphs, digraphs, and networks. These are:

- Graphs & Digraphs
- Graph Implementation
- Networks
- Network Implementation

Basic concepts and terminology are presented in Graphs & Digraphs and Networks. Implementation techniques for the data structures and common algorithms are described and illustrated in Graph Implementation and Network Implementation.

Each unit of the courseware has four main learning activities: tutorial, animated demonstration, interactive lab session, and self-test. These activities are described in the following sections.

Tutorials

The tutorials provide the most basic instruction. Concepts, terminology, and examples are provided using text and graphic images in a manner similar to a standard textbook. Hyperlinks are incorporated to provide supplementary material or topics of special interest which are related to the topic being covered. Students may proceed at their own pace through the tutorial material. It is recommended that students view the tutorial for a unit before going on to one of the other activities; however, this is not strictly necessary.

Animated Demonstrations

Animated demonstrations are a main feature of the courseware. Many of the concepts, data structures, and algorithmic operations are difficult to visualize. Textbooks have traditionally used sequences of pictures with explanatory notes to illustrate these topics. The use of animation allows students to see the changes in a data structure or the operation of an algorithm as the change progresses. Animations can be replayed as needed to study a specific topic.

Interactive Lab Sessions

Most learning theories, whether behavioral or cognitive, emphasize the need for learners to perform practical exercises to reinforce and clarify concepts and build skills. The interactive lab sessions in the courseware are intended to provide this type of learning activity. Students perform laboratory activities to solve specified problems or demonstrate phenomena. The activities are interactive. For example, one lab session in the first unit, Graphs & Digraphs, asks the student to build a graph of routes for a airline. Using a point-and-click interface the student adds edges to create a graph on screen.

Self Tests

The self tests are included primarily for the student to check his/her own progress. The tests are interactive and multiple choice questions. Students can review answers and change them as needed. Answers can be checked by the software with incorrect answers indicated along with the correct answer.

Appendix F

QuickBASIC Program Listing

```
DECLARE SUB WaitForMouseClick (x%, y%)
DECLARE SUB TestGetSessionData (Code$, Descr$(), NumFrames%, FText$(),
ANS$(), ActNodes%)
DECLARE SUB TestCheck (FrameText$(), ANS$(), NumFrames AS INTEGER)
DECLARE SUB TestShowAns (ANS$(), Frame AS INTEGER)
DECLARE SUB TestShowQuestion (FrameText$(), Frame AS INTEGER)
DECLARE SUB TestRun ()
DECLARE SUB TestClear (ANS$(), NumFrames AS INTEGER)
DECLARE SUB DemoNim1 ()
DECLARE SUB DemoNim2 ()
DECLARE SUB DemoNim3 ()
DECLARE SUB DemoNim4 ()
DECLARE SUB DemoNim5 ()
DECLARE SUB GraphHighlightMatrix (G() AS INTEGER, row AS INTEGER, col AS
INTEGER)
DECLARE SUB DemoGim1 ()
DECLARE SUB DemoGim2 ()
DECLARE SUB DemoGim3 ()
DECLARE SUB DemoNet1 ()
DECLARE SUB DemoNet2 ()
DECLARE SUB DemoNet3 ()
DECLARE SUB DemoNet4 ()
DECLARE SUB NetTopologicalSort (N() AS INTEGER)
DECLARE SUB LabDoTopSort (Graph() AS INTEGER, Nodes() AS ANY, Edges() AS
ANY)
DECLARE FUNCTION XRadians! (Degrees AS SINGLE)
DECLARE SUB EdgeArrowHead (e AS ANY)
DECLARE SUB NetShowShortestPath (Path() AS INTEGER, START%, Dest%)
DECLARE SUB NetShortestPath (N() AS INTEGER, START%, Dest%)
DECLARE SUB LabDoShortestPath (G() AS INTEGER, Nodes() AS ANY, Edges()
AS ANY)
DECLARE SUB LabDoTraversal (Graph() AS INTEGER, Nodes() AS ANY)
DECLARE SUB NetMinSpanTree (N() AS INTEGER)
DECLARE SUB LabDoMinSpan (Graph() AS INTEGER, Nodes() AS ANY, Edges() AS
ANY)
DECLARE SUB LabDoDepth (Graph() AS INTEGER, Nodes() AS ANY)
DECLARE SUB LabDoBreadth (Graph() AS INTEGER, Nodes() AS ANY)
DECLARE SUB NetFindMin (N() AS INTEGER, Node1%, Node2%)
DECLARE FUNCTION NetAllIncluded% (Visited() AS INTEGER)
DECLARE SUB LabCheckNet (G() AS INTEGER)
DECLARE SUB LabSetupNet ()
DECLARE SUB LabShowWeights ()
DECLARE SUB LabGetEdgeWeight (Weight%)
DECLARE SUB LabSetupmatrix (G() AS INTEGER)
DECLARE SUB GraphShowMatrix (G() AS INTEGER)
DECLARE SUB EdgeHighlight (e AS ANY, Nodes() AS ANY, Kolor%)
DECLARE SUB NodePoolInit (Nodes() AS ANY)
DECLARE SUB LabSetupGraph (G() AS INTEGER, GNum%, NumNodes%(),
NodeList%(), NumEdges%(), EdgeList%())
DECLARE SUB LabResetWorkRegion ()
DECLARE SUB EdgeChangeColor (e AS ANY, Kolor%)
DECLARE SUB EdgeFind (Edges() AS ANY, Node1%, Node2%, EIndex%)
DECLARE SUB DemoGraph5 ()
DECLARE SUB DemoGraph4 ()
DECLARE SUB DemoGraph2 ()
DECLARE SUB DemoGraph3 ()
DECLARE SUB NodeCreate (N AS ANY, Label$, NColor%, LColor%, x%, y%)
DECLARE SUB DemoGraph1 ()
DECLARE SUB DemoRun ()
DECLARE SUB DemoGetSessionData (Code$, Descr$(), NumFrames%, FText$(),
ActNodes%)
```

```
DECLARE SUB GraphBreadthFirstTraversal (G() AS INTEGER, START%)
DECLARE SUB QueueAdd (Q() AS INTEGER, Head%, Tail%, Item%)
DECLARE SUB QueueRemove (Q() AS INTEGER, Head%, Tail%, Item%)
DECLARE SUB QueueInit (Head%, Tail%)
DECLARE FUNCTION QueueEmpty% (Head%, Tail%)
DECLARE FUNCTION GraphEmpty% (G() AS INTEGER)
DECLARE SUB NodeChangeColor (N AS ANY, Kolor%)
DECLARE SUB GraphDepthFirstTraversal (G() AS INTEGER, START%)
DECLARE SUB GraphSearchFrom (NodeIndex%)
DECLARE SUB LabSetupBalsa ()
DECLARE SUB LabSetupSession (Session$)
DECLARE SUB LabCheckBalsa (G() AS INTEGER)
DECLARE SUB GraphShow (Nodes() AS ANY, Edges() AS ANY)
DECLARE SUB EdgeAdd (G() AS INTEGER, Node1%, Node2%, Weight%)
DECLARE SUB LabButtonService (B AS ANY)
DECLARE SUB EdgeTableInit (Edges() AS ANY)
DECLARE FUNCTION XColorValue! (RED!, GREEN!, BLUE!)
DECLARE SUB RegionCreate (R AS ANY, x1%, y1%, x2%, y2%, BGColor%)
DECLARE SUB RegionClear (R AS ANY)
DECLARE SUB RegionPrint (R AS ANY, RegionRow%, RegionCol%, Mesg$,
Kolor%)
DECLARE SUB RegionBorder (R AS ANY)
DECLARE SUB GraphInit (G() AS INTEGER)
DECLARE SUB NodeCenter (NODE%, x%, y%)
DECLARE SUB ButtonAction (B AS ANY)
DECLARE SUB NodeShow (NIndex%)
DECLARE SUB EdgeShow (e AS ANY, Nodes() AS ANY)
DECLARE SUB EdgeCreate (e AS ANY, EColor%, Node1%, Node2%)
DECLARE SUB EdgeFindFree (Edges() AS ANY, EdgeIndex%)
DECLARE SUB NodeShowDimmed (N AS ANY)
DECLARE SUB ButtonPress (B AS ANY)
DECLARE SUB ButtonDraw (B AS ANY)
DECLARE SUB ButtonCreate (B AS ANY, x1%, y1%, x2%, y2%, Light%, Dark%,
Face%, TextCol%, ButtonText$, Action%, Active%)
DECLARE SUB LabSetPalette (Colors() AS SINGLE)
DECLARE SUB LabGetSessionData (Code$, Descr$(), SHelp$(), Button() AS
ANY, NLabel$, ActNodes%)
DECLARE SUB LabTitleScreen ()
DECLARE SUB LabDrawLabScreen ()
DECLARE SUB LabGetUserAction (choice%)
DECLARE SUB LabTheEnd ()
DECLARE SUB LabDoAddNode (Nodes() AS ANY)
DECLARE SUB LabDoAddEdge (Nodes() AS ANY)
DECLARE SUB LabDoClearGraph (Nodes() AS ANY, Edges() AS ANY, G() AS
INTEGER)
DECLARE SUB LabDoHelp ()
DECLARE SUB RegionConfirm (R AS ANY, Mesg$, Response%)
DECLARE SUB XDelay (Seconds!)
DECLARE SUB LabGetNodeSelection (Nodes() AS ANY, NodeIndex%)
DECLARE SUB XPrintText (row%, col%, Mesg$, Kolor%)
DECLARE SUB MouseHide ()
DECLARE SUB MouseShow ()
DECLARE SUB WaitForMousePress (x%, y%)
DECLARE SUB MouseReset ()
' ===================================================================
' Program Name : ANILAB11.BAS
' By           : T. Beutel
' Date         : Mar. 13, 1997
' Description  : This program provides an interactive laboratory and
' demonstration environment for working with graphs and networks. It
' is part of the courseware package on graphs and networks. The program
' is called from a NeoBook application which provides the user interface
' and overall structure of the courseware. NeoBook calls the program
' without clearing the screen so that the program seems to be running
' within the NeoBook application.
'
' The program provides a general purpose display consisting of display
' areas and user controls which can be used for the various animated
' demonstrations and laboratory exercises. When the NeoBook application
' calls the program, it passes an appropriate command line parameter
' to specify what lab or animation is to be done.
' ==========================================================================

'Boolean Constants
CONST TRUE = -1
CONST FALSE = 0

'Palette Colors: set to match selected palette in NeoBook application.
CONST Black = 0
CONST BLUE = 1
CONST YELLOW = 2
CONST ROBINSEGG = 3
CONST RED = 4
CONST LTGRAY = 5
CONST MEDLTGRAY = 6
CONST MEDGRAY = 7
CONST MEDDKGRAY = 8
CONST DKGRAY = 9
CONST SPRINGGREEN = 10
CONST LTRED = 11
CONST PERSIMMON = 12
CONST CINNAMON = 13
CONST VANILLA = 14
CONST WHITE = 15

'Default Colors
CONST TEXTKOLOR = VANILLA                'Color of text in Text Region
CONST NODEKOLOR = YELLOW                 'Color of node on user display
CONST EDGEKOLOR = YELLOW                 'Color of edge on user display
CONST DRAWKOLOR = DKGRAY                 'Color for drawing node outline
CONST LABELKOLOR = YELLOW                'Color of node label
CONST BUTTONLIGHT = LTGRAY               'Light color for 3D button edge
CONST BUTTONDARK = DKGRAY                'Dark color for 3D button edge
CONST BUTTONFACE = Black                 'Color for face of 3D button
CONST BORDERLIGHT = LTGRAY               'Light color for 3D border
CONST BORDERDARK = DKGRAY                'Dark color for 3D border
CONST HIGHLTCOLOR = LTRED                'Edge highlight color

'Start Coordinates for Nodes in Work Area: center of the first node
CONST STARTX = 68
CONST STARTY = 232

' Coordinates/Color for Demo area
CONST DEMOX1 = 28
CONST DEMOY1 = 68
CONST DEMOX2 = 615
CONST DEMOY2 = 474
CONST DEMOKOLOR = Black

'The user display (Demo Area) is organized into five regions as follows:
'
'       Text    : display lab problems, explanations, etc.
'       Work    : area where a graph is displayed, animated, etc.
'       Controls: buttons for controlling the display
'       Message : error or prompt messages
'       Info    : information related to the current activity
'
'In addition, there are two special regions used in the program.
'
'       Confirm : used for confirmation messages
'       Matrix  : used to display the adjacency matrix
'
'The constants given below, specify the location of each area for
'this program.

' Coordinates for Text Region
CONST TEXTX1 = 28
CONST TEXTY1 = 68
CONST TEXTX2 = 615
CONST TEXTY2 = 195

' Coordinates for Work Region
CONST WORKX1 = 30
CONST WORKY1 = 204
CONST WORKX2 = 308
CONST WORKY2 = 468

' Coordinates for Controls Region
CONST CTRLX1 = 316
CONST CTRLY1 = 204
CONST CTRLX2 = 612
CONST CTRLY2 = 468

' Coordinates for Message Region
CONST MESGX1 = 322
CONST MESGY1 = 316
CONST MESGX2 = 603
CONST MESGY2 = 340

' Coordinates for Info Region
CONST INFOX1 = 322
CONST INFOY1 = 212
CONST INFOX2 = 603
CONST INFOY2 = 308

' Coordinates for Confirm Region
CONST CONFX1 = 188
CONST CONFY1 = 132
CONST CONFX2 = 436
CONST CONFY2 = 236

' Coordinates for the Matrix Region
CONST MATRX1 = 322
CONST MATRY1 = 72
CONST MATRX2 = 603
CONST MATRY2 = 340

'Coordinates for small Matrix Region
CONST SMATX1 = 322
CONST SMATY1 = 196
CONST SMATX2 = 603
CONST SMATY2 = 340
```

```
' Coordinates for Edge Weight Key
CONST WGHTX1 = 120
CONST WGHTY1 = 448

'Lines of description and help
CONST MAXDESCR = 7
CONST MAXHELP = 6

'Maximum number of buttons, nodes, and edges for the user display.
CONST MAXBUTTONS = 12
CONST MAXNODES = 16
CONST MAXEDGES = 42
CONST NODERADIUS = 16                   'radius of a node
CONST MAXFRAMES = 10                    'separate pages in demo/test
CONST MAXWT = 100                       'high value for edge weight

' Action Codes for Buttons: codes specify possible actions for buttons.
' Not all actions will be used for a given lab/demo session. The
' actions for a given session will be specified in the session data.
CONST NOACTION = 0
CONST ADDNODE = 1
CONST DELNODE = 2
CONST ADDEDGE = 3
CONST DELEDGE = 4
CONST ANS.A = 5                 'multiple choice answers
CONST ANS.B = 6
CONST ANS.C = 7
CONST ANS.D = 8
CONST RUNDEPTH = 10             'depth-first traversal algorithm
CONST RUNBREADTH = 11           'breadth-first traversal algorithm
CONST RUNPATH = 12              'shortest path algorithm
CONST RUNSPAN = 13              'minimum spanning tree algorithm
CONST RUNSORT = 14              'topological sort algorithm
CONST RUNDEMO = 15              'animated demo
CONST STARTTEST = 16            'self-test controls
CONST NEXTQUES = 17
CONST PREVQUES = 18
CONST PAUSE = 20
CONST BACK = 21                 'go back one frame in demo
CONST CONTIN = 30
CONST FORWARD = 31              'go forward one frame in demo
CONST RESTART = 40
CONST CLRGRAPH = 50
CONST CLRBALSA = 51
CONST CLRMATRIX = 52            'clear graph & adjacency matrix display
CONST CLRTEST = 54              'clear answers on self-test
CONST CLRNET = 53               'clear network for errand problem
CONST CHECK = 60
CONST CHECKBALSA = 61           'check graph for Balsa airline problem
CONST CHECKNET = 62             'check network for errand problem
CONST CHECKTEST = 63            'check answers on self-test
CONST HELP = 70
CONST QUIT = 80

' -----------------------------------------------------------------
' Data Structure Declarations
' -----------------------------------------------------------------

' ButtonType: A ButtonType is a 3D button used to activate some program
' feature. A ButtonType consists of a LightBorderColor, DarkBorderColor,
' FaceColor, TextColor, ButtonText, top left corner (X1,Y1), bottom
' right corner (X2,Y2), an ActionCode, and an Active/Inactive flag.
' The ActionCode is used by the ServiceButton routine to call an
' associated subroutine.

TYPE ButtonType
   ButtonActive AS INTEGER              'actually "boolean"
   x1 AS INTEGER
   y1 AS INTEGER
   x2 AS INTEGER
   y2 AS INTEGER
   LightBorderColor AS INTEGER
   DarkBorderColor AS INTEGER
   FaceColor AS INTEGER
   TextColor AS INTEGER
   ButtonText AS STRING * 9
   ActionCode AS INTEGER
END TYPE

' GraphNode: A node in a graph consisting of a label, node color,
' label color, x and y coordinated for displaying the node, and a
' boolean flag indicating if the node is visible on the user display.

TYPE GraphNode
   Label AS STRING * 1
   NodeColor AS INTEGER
   LabelColor AS INTEGER
   NodeXCoord AS INTEGER
   NodeYCoord AS INTEGER
   NodeVisible AS INTEGER
END TYPE

' GraphEdge: Connects two GraphNodes. The connection is indicated
' in the adjacency matrix, but a table of edges is also kept to allow
' the current edge color to be stored. The edge color will change
' during traversals or other algorithmic manipulations.

TYPE GraphEdge
   EdgeColor AS INTEGER
   Node1 AS INTEGER
   Node2 AS INTEGER
END TYPE

' Region: A Region is a rectangular portion of the screen consisting of
' a RegionBGColor and a three-dimensional border. The location is
' specified by the coordinates of the top left corner (x1,y1) and the
' coordinates of the bottom right corner (x2,y2).

TYPE Region
   x1 AS INTEGER
   y1 AS INTEGER
   x2 AS INTEGER
   y2 AS INTEGER
   BGColor AS INTEGER
END TYPE

' ---------------------------------------------------------------------
' Variable Declarations
' ---------------------------------------------------------------------
' Among the variable declared below are three arrays which deserve
' some explanation: the Button Table, Node Pool and Edge Table.
'
```

```
' The Button Table is a one-dimensional array with each element
' corresponding to a button on the user display. Each element, a
' ButtonType, includes relevant information such as its position,
' colors, whether it is active or not, and an action code to indicate
' what action the ButtonService routine should initiate when the
' button is pressed.
'
' The Node Pool is a one-dimensional array with each element
' corresponding to a GraphNode. There are MAXNODES nodes in the
' Node Pool, each node representing a displayable node in the work
' area of the user display. These are initialized at the start of the
' program to specify the location, color, associated label, etc.
' Initially, all nodes are "hidden", that is, displayed as a dimmed
' image. As nodes are added to a Graph, either by the user or by
' the program, the corresponding GraphNode is "shown" on the user
' display. The Node Pool keeps track of the state of individual
' nodes.
'
' The Edge Table is also a one-dimensional array, with each element
' corresponding to an edge in a Graph. When an edge is added to a
' graph by the user or by the program, an entry is made in the
' adjacency matrix used to represent the Graph and the edge is
' recorded in the Edge Table. The main purpose of the edge table
' entry is to maintain the current color of the edge.

DIM Colors(0 TO 15)                        'Array of palette colors
DIM Button(1 TO MAXBUTTONS) AS ButtonType  'Button Table
DIM Nodes(1 TO MAXNODES) AS GraphNode      'pool of nodes
DIM Edges(1 TO MAXEDGES) AS GraphEdge      'edge table

' A Graph is represented by a two-dimensional array called and
' an adjacency matrix. Two nodes which are connected by an edge
' is represented by TRUE (-1) stored in the corresponding
' row/col elements of the array. (Note there will be two such
' entries for each edge since the graph is undirected.

DIM Graph(1 TO MAXNODES, 1 TO MAXNODES)  AS INTEGER

' The Visited array is used to keep track of nodes visited during
' the depth-first traversal.

DIM Visited(1 TO MAXNODES) AS INTEGER

' The InQueue array is used to keep track of nodes pending processing
' during the breadth-first traversal. Queue holds the indexes of nodes
' to be processed. Head and Tail indicate the head and tail of the
' queue.

DIM InQueue(1 TO MAXNODES) AS INTEGER
DIM Queue(1 TO MAXNODES) AS INTEGER
DIM Head AS INTEGER
DIM Tail AS INTEGER

' Regions on Lab Screen
DIM DemoRegion AS Region
DIM TextRegion AS Region
DIM MesgRegion AS Region
DIM InfoRegion AS Region
DIM WorkRegion AS Region
DIM ControlRegion AS Region
DIM ConfirmRegion AS Region
DIM MatrixRegion AS Region

' Session Data: stores all information needed to specify a session
' including the session code, description, help info. Button info.
' is stored in Button Table.

DIM Session$                        'code for current session
DIM SessionDescr$(1 TO MAXDESCR)
DIM SessionHelp$(1 TO MAXHELP)
DIM NodeLabel$                      'string containing node labels

' Demonstration Data: stores information for frames of a demo.
' The demo description and help are stored in the SessionDescr$
' and SessionHelp$ arrays above. A demo may also include up to
' two prespecified graphs. These are stored as lists of nodes
' and edges (node pairs).

DIM NumFrames AS INTEGER
DIM Frame AS INTEGER

DIM FrameText$(1 TO MAXFRAMES, 1 TO MAXDESCR)

'Self-test Data: Uses NumFrames, Frame and FrameText$ for questions.
'Ans$() holds multiple choice answers. Fifth entry holds a two-
'char string consisting of the correct answer followed by the user's
'answer.

DIM ANS$(1 TO MAXFRAMES, 1 TO 5)

DIM NumNodes(1 TO 2) AS INTEGER
DIM NumEdges(1 TO 2) AS INTEGER
DIM NodeList(1 TO 2, 1 TO MAXNODES) AS INTEGER

'EdgeList hold data for two graphs, each having up to MAXEDGES edges,
'with two nodes (last index 1 and 2) and an edge weight (last index 3).
DIM EdgeList(1 TO 2, 1 TO MAXEDGES, 1 TO 3) AS INTEGER

DIM DONE AS INTEGER                 'Boolean variable for main program loop
DIM i AS INTEGER
DIM ActualNodes AS INTEGER          'actual number of nodes used
DIM ShowMatrix AS INTEGER           'should adj. matrix be displayed
DIM Net AS INTEGER                  'TRUE if network being used
DIM Digraph AS INTEGER              'TRUE if digraph is being used

' ====================================================================
' Main Program
' ====================================================================
DEFINT A-Z                          'make integer the default data type
SCREEN 12                           '640x480x16 colors
LabSetPalette Colors!()             'Set palette to same as NeoBook
Session$ = COMMAND$                 'get session from command line

IF RIGHT$(Session$, 3) = "LAB" THEN     'lab session
  LabGetSessionData Session$, SessionDescr$(), SessionHelp$(), Button(),
NodeLabel$, ActualNodes

ELSEIF RIGHT$(Session$, 4) = "DEMO" THEN 'demo session
  DemoGetSessionData Session$, SessionDescr$(), NumFrames, FrameText$(),
ActualNodes

   ' Create standard button set for demos
   FOR i = 1 TO MAXBUTTONS
     READ Text$, TColor, x1, y1, x2, y2, Action, Active
     ButtonCreate Button(i), x1, y1, x2, y2, BUTTONLIGHT, BUTTONDARK,
 BUTTONFACE, TColor, Text$, Action, Active
   NEXT i

   ' Standard Node labels for demos
   NodeLabel$ = "ABCDEFGHIJKLMNOP"

   ' Standard Help for Demos
   SessionHelp$(1) = "                   CONTROLS FOR DEMONSTRATION"
   SessionHelp$(2) = ""
   SessionHelp$(3) = "            RUN            start demonstration"
   SessionHelp$(4) = "            NEXT           proceed to next frame
 of demo"
   SessionHelp$(5) = "            BACK           go back to preceding
 frame"
   SessionHelp$(6) = "            QUIT           terminate
 demonstration"

   Frame = 0

 ELSEIF RIGHT$(Session$, 4) = "TEST" THEN        'self-test
   TestGetSessionData Session$, SessionDescr$(), NumFrames, FrameText$(),
 ANS$(), ActualNodes

   ' Create standard button set for tests

   FOR i = 1 TO MAXBUTTONS                       'read past demo DATA
     READ Text$, TColor, x1, y1, x2, y2, Action, Active
   NEXT i
   FOR i = 1 TO MAXBUTTONS                       'read test DATA
     READ Text$, TColor, x1, y1, x2, y2, Action, Active
     ButtonCreate Button(i), x1, y1, x2, y2, BUTTONLIGHT, BUTTONDARK,
 BUTTONFACE, TColor, Text$, Action, Active
   NEXT i

   ' Standard Node labels for tests
   NodeLabel$ = "ABCDEFGHIJKLMNOP"

   ' Standard Help for tests
   SessionHelp$(1) = "                     CONTROLS FOR SELF-TEST"
   SessionHelp$(2) = "    START: start self-test            A: select
 answer A"
   SessionHelp$(3) = "    NEXT : go to next question        B: select
 answer B"
   SessionHelp$(4) = "    BACK : go to previous question    C: select
 answer C"
   SessionHelp$(5) = "    CHECK: check answer to question   D: select
 answer D"
   SessionHelp$(6) = "    QUIT : terminate self-test"

   Frame = 0

 END IF

 'Create Lab Screen Regions
 RegionCreate DemoRegion, DEMOX1, DEMOY1, DEMOX2, DEMOY2, DEMOKOLOR
 RegionCreate TextRegion, TEXTX1, TEXTY1, TEXTX2, TEXTY2, DEMOKOLOR
 RegionCreate WorkRegion, WORKX1, WORKY1, WORKX2, WORKY2, MEDGRAY
 RegionCreate MesgRegion, MESGX1, MESGY1, MESGX2, MESGY2, DEMOKOLOR
```

```
RegionCreate InfoRegion, INFOX1, INFOY1, INFOX2, INFOY2, DEMOKOLOR
RegionCreate ControlRegion, CTRLX1, CTRLY1, CTRLX2, CTRLY2, DEMOKOLOR
RegionCreate ConfirmRegion, CONFX1, CONFY1, CONFX2, CONFY2, DEMOKOLOR

IF Session$ = "GIMPDEMO" OR Session$ = "NIMPDEMO" THEN
   RegionCreate MatrixRegion, SMATX1, SMATY1, SMATX2, SMATY2, DEMOKOLOR
ELSE
   RegionCreate MatrixRegion, MATRX1, MATRY1, MATRX2, MATRY2, DEMOKOLOR
END IF

NodePoolInit Nodes()
EdgeTableInit Edges()
GraphInit Graph()

LabTitleScreen
LabDrawLabScreen
LabSetupSession Session$

DONE = FALSE
WHILE NOT DONE                    'loop until user quits session
  choice = 0
  LabGetUserAction choice
  IF (choice <> 0) THEN ButtonAction Button(choice)
WEND
LabTheEnd
CLS

' ==================================================================
' Data for default buttons for animated demos.
' ==================================================================
DATA "ADD NODE",5,322,346,412,370,0,0
DATA "DEL NODE",5,322,378,412,402,0,0
DATA "ADD EDGE",5,322,410,412,434,0,0
DATA "DEL EDGE",5,322,442,412,466,0,0
DATA "RUN",5,418,346,508,370,15,-1
DATA "NEXT",5,418,378,508,402,31,-1
DATA "BACK",5,418,410,508,434,21,-1
DATA "RESTART",5,418,442,508,466,0,0
DATA "CLEAR",5,514,346,604,370,0,0
DATA "CHECK",5,514,378,604,402,0,0
DATA "HELP!",10,514,410,604,434,70,-1
DATA "QUIT",11,514,442,604,466,80,-1
' ==================================================================
' Data for default buttons for self-tests.
' ==================================================================
DATA "A",5,322,346,412,370,5,-1
DATA "B",5,322,378,412,402,6,-1
DATA "C",5,322,410,412,434,7,-1
DATA "D",5,322,442,412,466,8,-1
DATA "START",5,418,346,508,370,16,-1
DATA "NEXT",5,418,378,508,402,17,-1
DATA "BACK",5,418,410,508,434,18,-1
DATA "RESTART",5,418,442,508,466,0,0
DATA "CLEAR",5,514,346,604,370,54,-1
DATA "CHECK",5,514,378,604,402,63,-1
DATA "HELP!",10,514,410,604,434,70,-1
DATA "QUIT",11,514,442,604,466,80,-1

' ==================================================
' Subroutines: Subroutines are organized alphabetically by QuickBASIC.
' So...subroutines are grouped into groups of related operations by
' starting each subroutine name with a key identifier, such as Button
' for all button-related routines. Routines starting with X are general
' purpose routines.
' ==========================================================
' ==========================================================
' ButtonAction: If a button is active, carry out the action specified
' in the ActionCode.
' ==========================================================
SUB ButtonAction (B AS ButtonType)
  IF (B.ButtonActive = TRUE) THEN
    ButtonPress B
    LabButtonService B
    ButtonDraw B
  END IF
END SUB

' ------------------------------------------------------------------
' ButtonCreate: Creates a ButtonType B with the specified parameters.
' ------------------------------------------------------------------
SUB ButtonCreate (B AS ButtonType, x1, y1, x2, y2, Light, Dark, Face,
TextCol, ButtonText$, Action, Active)
  B.ButtonActive = Active
  B.x1 = x1
  B.y1 = y1
  B.x2 = x2
  B.y2 = y2
  B.LightBorderColor = Light
  B.DarkBorderColor = Dark
  B.FaceColor = Face
  B.TextColor = TextCol
  B.ButtonText$ = ButtonText$
  B.ActionCode = Action
END SUB

' ------------------------------------------------------------------
' ButtonDraw: Displays a button using the parameters stored in the
' button's data structure.
' ------------------------------------------------------------------
SUB ButtonDraw (B AS ButtonType)
  MouseHide
  IF B.ButtonActive = TRUE THEN
    LINE (B.x1, B.y1)-(B.x2, B.y1), B.LightBorderColor
    LINE (B.x1 + 1, B.y1 + 1)-(B.x2 - 1, B.y1 + 1), B.LightBorderColor
    LINE (B.x1, B.y1)-(B.x1, B.y2), B.LightBorderColor
    LINE (B.x1 + 1, B.y1 + 1)-(B.x1 + 1, B.y2 - 1), B.LightBorderColor
    LINE (B.x2, B.y1)-(B.x2, B.y2), B.DarkBorderColor
    LINE (B.x2 - 1, B.y1 + 1)-(B.x2 - 1, B.y2 - 1), B.DarkBorderColor
    LINE (B.x2, B.y2)-(B.x1, B.y2), B.DarkBorderColor
    LINE (B.x2 - 1, B.y2 - 1)-(B.x1 + 1, B.y2 - 1), B.DarkBorderColor
    LINE (B.x1 + 2, B.y1 + 2)-(B.x2 - 2, B.y2 - 2), B.FaceColor, BF
    ButtonLength = INT((B.x2 - B.x1) \ 8)
    ButtonHeight = INT((B.y2 - B.y1) \ 16)
    row = INT(B.y1 \ 16) + ButtonHeight \ 2 + 2
    col = INT(B.x1 \ 8) + (ButtonLength - LEN(B.ButtonText$)) \ 2 + 1
    XPrintText row, col, B.ButtonText$, B.TextColor
  ELSE
    LINE (B.x1, B.y1)-(B.x2, B.y2), MEDDKGRAY, BF
  END IF
  MouseShow
END SUB
```

```
' ------------------------------------------------------------------
' ButtonHide: Sets the Button Active field to false so that the
' button will not be displayed.
' ------------------------------------------------------------------
SUB ButtonHide (B AS ButtonType)
  B.ButtonActive = FALSE
END SUB

' ------------------------------------------------------------------
' ButtonPress: Displays a button with the light and dark colors
' swapped and the Button text moved to the right to make it appear
' that the button has been pressed in.
' ------------------------------------------------------------------
SUB ButtonPress (B AS ButtonType)
  MouseHide
  LINE (B.x1, B.y1)-(B.x2, B.y1), B.DarkBorderColor
  LINE (B.x1 + 1, B.y1 + 1)-(B.x2 - 1, B.y1 + 1), B.DarkBorderColor
  LINE (B.x1, B.y1)-(B.x1, B.y2), B.DarkBorderColor
  LINE (B.x1 + 1, B.y1 + 1)-(B.x1 + 1, B.y2 - 1), B.DarkBorderColor
  LINE (B.x2, B.y1)-(B.x2, B.y2), B.LightBorderColor
  LINE (B.x2 - 1, B.y1 + 1)-(B.x2 - 1, B.y2 - 1), B.LightBorderColor
  LINE (B.x2, B.y2)-(B.x1, B.y2), B.LightBorderColor
  LINE (B.x2 - 1, B.y2 - 1)-(B.x1 + 1, B.y2 - 1), B.LightBorderColor
  LINE (B.x1 + 2, B.y1 + 2)-(B.x2 - 2, B.y2 - 2), B.FaceColor, BF
  ButtonLength = INT((B.x2 - B.x1) \ 8)
  ButtonHeight = INT((B.y2 - B.y1) \ 16)
  row = INT(B.y1 \ 16) + ButtonHeight \ 2 + 2
  col = INT(B.x1 \ 8) + (ButtonLength - LEN(B.ButtonText$)) \ 2 + 2
  XPrintText row, col, B.ButtonText$, MEDGRAY
  MouseShow
END SUB

' ------------------------------------------------------------------
' ButtonUnhide: Sets the Button Active field to True so that the
' button will be displayed.
' ------------------------------------------------------------------
SUB ButtonUnhide (B AS ButtonType)
  B.ButtonActive = TRUE
END SUB

' ------------------------------------------------------------------
' GetSessionData:Retrieve session data from file.
' ------------------------------------------------------------------
SUB DemoGetSessionData (Code$, Descr$(), NumFrames, FText$(), ActNodes)
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER

  FileName$ = Code$ + ".DAT"
  CLOSE #1
  OPEN FileName$ FOR INPUT AS #1
  INPUT #1, NumDescr                'how many descr lines?
  FOR i = 1 TO NumDescr             'get description
     INPUT #1, Descr$(i)
  NEXT i
  FOR i = NumDescr + 1 TO MAXDESCR 'fill in blank lines
     Descr$(i) = ""
  NEXT i
  INPUT #1, NumFrames
  FOR i = 1 TO NumFrames           'get text for each frame
```

```
      INPUT #1, NumLines
      FOR j = 1 TO NumLines
       INPUT #1, FText$(i, j)
      NEXT j
   NEXT i
   INPUT #1, NumGraphs
   FOR i = 1 TO NumGraphs
      INPUT #1, NumNodes(i)
      FOR j = 1 TO NumNodes(i)
       INPUT #1, NodeList(i, j)
      NEXT j
      INPUT #1, NumEdges(i)
      FOR j = 1 TO NumEdges(i)
       INPUT #1, EdgeList(i, j, 1), EdgeList(i, j, 2), EdgeList(i, j, 3)
      NEXT j
   NEXT i
   INPUT #1, ActNodes
   CLOSE #1
END SUB

' ----------------------------------------------------------------
' DemoGim1: First frame animation for Graph Implementation animated
' demonstration. The use of the adjacency matrix to represent the
' edge connections of a graph is presented.
' ----------------------------------------------------------------
SUB DemoGim1
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED WorkRegion AS Region
SHARED NodeLabel$
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER
SHARED ShowMatrix AS INTEGER

  Digraph = FALSE
  LabResetWorkRegion
  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphInit Graph()
  LabSetupmatrix Graph()
  FOR i = 1 TO NumNodes(1)
    N = NodeList(1, i)
    NodeShow N: SOUND 440, 1: XDelay (1)
  NEXT i
  XDelay (2)
  ShowMatrix = TRUE
  GraphInit Graph()
  FOR i = 1 TO NumEdges(1)
    EdgeShow Edges(i), Nodes(): SOUND 880, 1
    EdgeAdd Graph(), Edges(i).Node1, Edges(i).Node2, 1
    XDelay (2)
  NEXT i

END SUB

' ----------------------------------------------------------------
' DemoGim2: Second frame animation for Graph Implementation animated
' demonstration. The relationship of the degree of a node to the
' adjacency matrix is presented.
' -----------------------------------------------------------------
'
SUB DemoGim2
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED WorkRegion AS Region
SHARED NodeLabel$
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER

  Digraph = FALSE
  GraphInit Graph()
  LabSetupmatrix Graph()
  LabResetWorkRegion
  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphShow Nodes(), Edges()
  NodeChangeColor Nodes(1), LTRED
  NodeShow 1
  SOUND 440, 1
  XDelay (1)
  EdgeFind Edges(), 1, 2, EdgeIndex
  EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
  SOUND 880, 1
  GraphHighlightMatrix Graph(), 2, 1
  XDelay (2)
  EdgeFind Edges(), 1, 6, EdgeIndex
  EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
  SOUND 880, 1
  GraphHighlightMatrix Graph(), 6, 1
  XDelay (2)
END SUB

' ==============================================================
' DemoGim3: Third frame animation for Graph Implementation animated
' demonstration. The relationship of a path to the adjacency matrix
' is presented.
'
' ==============================================================
SUB DemoGim3
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED WorkRegion AS Region
SHARED NodeLabel$
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER

  Digraph = FALSE
  GraphInit Graph()
  LabSetupmatrix Graph()
  LabResetWorkRegion
  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphShow Nodes(), Edges()
  NodeChangeColor Nodes(1), LTRED
  NodeChangeColor Nodes(6), LTRED
  NodeChangeColor Nodes(5), LTRED
  EdgeFind Edges(), 1, 6, EdgeIndex
  EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
  SOUND 440, 1
  GraphHighlightMatrix Graph(), 1, 6
  XDelay (2)
  EdgeFind Edges(), 5, 6, EdgeIndex
  EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
  SOUND 440, 1
  GraphHighlightMatrix Graph(), 6, 5
  XDelay (2)

END SUB

' ------------------------------------------------------------------
' DemoGraph1: First frame animation for Graphs & Digraphs animated
' demonstration. The terms NODE and EDGE are presented.
' ------------------------------------------------------------------
SUB DemoGraph1
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED TextRegion AS Region
SHARED WorkRegion AS Region
SHARED NodeLabel$
SHARED Graph() AS INTEGER

  LabResetWorkRegion
  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  FOR i = 1 TO NumNodes(1)
    N = NodeList(1, i)
    NodeShow N: SOUND 440, 1: XDelay (1)
  NEXT i
  XDelay (1)
  RegionPrint TextRegion, 4, 20, "and EDGES.", TEXTKOLOR
  XDelay (1)
  FOR i = 1 TO NumEdges(1)
    EdgeShow Edges(i), Nodes(): SOUND 880, 1: XDelay (1)
  NEXT i
  XDelay (1)
END SUB

' ------------------------------------------------------------------
' DemoGraph2: presents direct connection of nodes
' ------------------------------------------------------------------
SUB DemoGraph2
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED Graph() AS INTEGER

  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphShow Nodes(), Edges()
  XDelay (1)
  EdgeFind Edges(), 6, 7, EIndex
  NodeChangeColor Nodes(6), LTRED
  NodeChangeColor Nodes(7), LTRED
  EdgeHighlight Edges(EIndex), Nodes(), HIGHLTCOLOR
  SOUND 880, 1
  XDelay (2)
END SUB

' ------------------------------------------------------------------
' DemoGraph3: Shows a path connecting two nodes.
' ------------------------------------------------------------------
SUB DemoGraph3
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED Graph() AS INTEGER

  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphShow Nodes(), Edges()
  XDelay (2)
  EdgeFind Edges(), 6, 11, E1
  EdgeFind Edges(), 10, 11, E2
  NodeChangeColor Nodes(6), LTRED
  NodeChangeColor Nodes(10), LTRED
  NodeChangeColor Nodes(11), LTRED
  EdgeHighlight Edges(E1), Nodes(), HIGHLTCOLOR: SOUND 880, 1: XDelay
(2)
  EdgeHighlight Edges(E2), Nodes(), HIGHLTCOLOR: SOUND 880, 1: XDelay
(2)
END SUB

' ------------------------------------------------------------------
' DemoGraph4: Shows the depth-first traversal for a graph.
' ------------------------------------------------------------------
SUB DemoGraph4
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED NodeLabel$
SHARED Graph() AS INTEGER

  LabSetupGraph Graph(), 2, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphShow Nodes(), Edges()
  XDelay (2)
  GraphDepthFirstTraversal Graph(), 2
END SUB
```

```
' ----------------------------------------------------------------
' DemoGraph5: Shows a breadth-first traversal for a graph.
' ----------------------------------------------------------------
SUB DemoGraph5
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED NodeLabel$
SHARED Graph() AS INTEGER

  LabSetupGraph Graph(), 2, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphShow Nodes(), Edges()
  XDelay (2)
  GraphBreadthFirstTraversal Graph(), 2
END SUB

' ----------------------------------------------------------------
' DemoNet1: First frame animation for Networks animated demonstration.
' The network similarity to a graph and the term EDGE WEIGHT are
' presented.
' ----------------------------------------------------------------
SUB DemoNet1
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED WorkRegion AS Region
SHARED NodeLabel$
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER

  Digraph = FALSE
  LabResetWorkRegion
  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphShow Nodes(), Edges()
  LabShowWeights

END SUB

' ========================================================
' DemoNet2: Show Minimum Spanning Tree for a network.
' ========================================================
SUB DemoNet2
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED NodeLabel$
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER

  Digraph = FALSE
  LabSetupGraph Graph(), 2, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphShow Nodes(), Edges()
  LabShowWeights
  XDelay (2)
  NetMinSpanTree Graph()

END SUB

' ======================================================
' DemoNet3: Show Shortest Path algorithm for a network.
' ======================================================
SUB DemoNet3
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED NodeLabel$
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER

  Digraph = FALSE
  LabSetupGraph Graph(), 2, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphShow Nodes(), Edges()
  LabShowWeights
  XDelay (2)
  NodeChangeColor Nodes(1), LTRED
  NodeShow 1: SOUND 440, 1: XDelay (1)
  NetShortestPath Graph(), 1, 8

END SUB

' ======================================================
' DemoNet4: Show Topological Sort algorithm for a network.
' ======================================================
SUB DemoNet4
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED NodeLabel$
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER

  Digraph = TRUE
  LabSetupGraph Graph(), 2, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  GraphShow Nodes(), Edges()
  LabShowWeights
  XDelay (2)
  NetTopologicalSort Graph()
  Digraph = FALSE

END SUB

' ======================================================
' DemoNim1: First frame of the Network Implementation Demo.
' Illustrates the use of the adjacency matrix for networks.
' ========================================================
SUB DemoNim1
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED WorkRegion AS Region
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER
SHARED Net AS INTEGER

  Digraph = FALSE
  Net = TRUE
  LabResetWorkRegion
  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  LabSetupmatrix Graph()
  GraphShow Nodes(), Edges()
  LabShowWeights
END SUB

' ========================================================
' DemoNim2: Begin sequence relating adjacency matrix to
' implementation of minimum spanning tree algorithm.
' ========================================================
SUB DemoNim2
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED WorkRegion AS Region
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER
SHARED Net AS INTEGER

  Digraph = FALSE
  Net = TRUE
  LabResetWorkRegion
  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  LabSetupmatrix Graph()
  GraphShow Nodes(), Edges()
  LabShowWeights
  NodeChangeColor Nodes(1), LTRED
  NodeShow 1
  SOUND 440, 1
  XDelay (1)

END SUB

' ========================================================
' DemoNim3: Continue sequence showing use of the adjacency
' matrix in implementing the minimum spanning tree algorithm.
' ========================================================
SUB DemoNim3
```

```
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED WorkRegion AS Region
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER
SHARED Net AS INTEGER

  Digraph = FALSE
  Net = TRUE
  LabResetWorkRegion
  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  LabSetupmatrix Graph()
  GraphShow Nodes(), Edges()
  LabShowWeights
  NodeChangeColor Nodes(1), LTRED
  NodeChangeColor Nodes(5), LTRED
  NodeShow 1
  EdgeFind Edges(), 1, 5, EdgeIndex
  EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
  SOUND 880, 1
  XDelay (2)

END SUB

' ========================================================
' DemoNim4: Continue sequence showing use of the adjacency
' matrix in implementing the minimum spanning tree algorithm.
' ========================================================
SUB DemoNim4
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER
SHARED WorkRegion AS Region
SHARED Graph() AS INTEGER
SHARED Digraph AS INTEGER
SHARED Net AS INTEGER

  Digraph = FALSE
  Net = TRUE
  LabResetWorkRegion
  LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
  LabSetupmatrix Graph()
  GraphShow Nodes(), Edges()
  LabShowWeights
  NodeChangeColor Nodes(1), LTRED
  NodeChangeColor Nodes(5), LTRED
  NodeChangeColor Nodes(2), LTRED
  EdgeFind Edges(), 1, 5, EdgeIndex
  EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
  EdgeFind Edges(), 5, 2, EdgeIndex
  EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
  SOUND 880, 1
```

```
   XDelay (2)

 END SUB

 ' ==========================================================
 ' DemoNim5: Continue sequence showing use of the adjacency
 ' matrix in implementing the minimum spanning tree algorithm.
 ' ==========================================================
 SUB DemoNim5
 SHARED Nodes() AS GraphNode
 SHARED Edges() AS GraphEdge
 SHARED NumNodes() AS INTEGER
 SHARED NumEdges() AS INTEGER
 SHARED NodeList() AS INTEGER
 SHARED EdgeList() AS INTEGER
 SHARED WorkRegion AS Region
 SHARED Graph() AS INTEGER
 SHARED Digraph AS INTEGER
 SHARED Net AS INTEGER

   Digraph = FALSE
   Net = TRUE
   LabResetWorkRegion
   LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
 EdgeList()
   LabSetupmatrix Graph()
   GraphShow Nodes(), Edges()
   LabShowWeights
   NodeChangeColor Nodes(1), LTRED
   NodeChangeColor Nodes(5), LTRED
   NodeChangeColor Nodes(2), LTRED
   NodeChangeColor Nodes(6), LTRED
   EdgeFind Edges(), 1, 5, EdgeIndex
   EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
   EdgeFind Edges(), 5, 2, EdgeIndex
   EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
   EdgeFind Edges(), 2, 6, EdgeIndex
   EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
   SOUND 880, 1
   XDelay (2)

 END SUB

 ' ---------------------------------------------------------------
 ' DemoRun: Run an animated demonstration. Data for the demo, including
 ' the text for each "frame" is in global variables. The number of
 ' frames in the demo is in the variable NumFrames. Each frame is
 ' implemented with a separate subroutine.
 ' ---------------------------------------------------------------
 SUB DemoRun
 SHARED TextRegion AS Region
 SHARED MatrixRegion AS Region
 SHARED Graph() AS INTEGER
 SHARED FrameText$()
 SHARED Frame AS INTEGER
 SHARED NumFrames AS INTEGER
 SHARED Session$
 SHARED Nodes() AS GraphNode
 SHARED Edges() AS GraphEdge

   RegionClear TextRegion
   FOR i = 1 TO MAXDESCR
     RegionPrint TextRegion, i, 4, FrameText$(Frame, i), TEXTKOLOR
   NEXT i
   RegionPrint TextRegion, 1, 64, STR$(Frame) + " OF" + STR$(NumFrames),
 TEXTKOLOR
   XDelay (2)
   IF Session$ = "GRPHDEMO" THEN
     SELECT CASE Frame
     CASE 1
       DemoGraph1
     CASE 2
       DemoGraph2
     CASE 3
       DemoGraph3
     CASE 4
       DemoGraph4
     CASE 5
       DemoGraph5
     END SELECT
   END IF
   IF Session$ = "NETDEMO" THEN
     SELECT CASE Frame
     CASE 1
       DemoNet1
     CASE 2
       DemoNet2
     CASE 3
       DemoNet3
     CASE 4
       DemoNet4
     END SELECT
   END IF
   IF Session$ = "GIMPDEMO" THEN
     SELECT CASE Frame
     CASE 1
       DemoGim1
     CASE 2
       DemoGim2
     CASE 3
       DemoGim3
     END SELECT
   END IF
   IF Session$ = "NIMPDEMO" THEN
     SELECT CASE Frame
     CASE 1
       DemoNim1
     CASE 2
       DemoNim2
     CASE 3
       DemoNim3
     CASE 4
       DemoNim4
     CASE 5
       DemoNim5
END SELECT
   END IF

END SUB

' ----------------------------------------------------------
' EdgeAdd: adds a bidirectional edge to a graph or network by
' setting the appropriate elements in the adjacency matrix. Weight
' is the edge weight. If Digraph=TRUE a directed edge is added.
' ------------------------------------------------------------------
SUB EdgeAdd (G() AS INTEGER, Node1, Node2, Weight)
SHARED ShowMatrix AS INTEGER
SHARED Digraph AS INTEGER

  G(Node1, Node2) = Weight      'edge from Node1 to Node2
  IF NOT Digraph THEN
     G(Node2, Node1) = Weight
  END IF
  IF ShowMatrix THEN GraphShowMatrix G()
END SUB

' ------------------------------------------------------------------
' EdgeArrowHead: Draw arrowhead to indicate a directed edge.
' ------------------------------------------------------------------
SUB EdgeArrowHead (e AS GraphEdge)
SHARED Nodes() AS GraphNode
DIM Theta AS SINGLE
DIM TanTheta AS SINGLE

   Node1 = e.Node1
   Node2 = e.Node2
   x1 = Nodes(Node1).NodeXCoord
   y1 = Nodes(Node1).NodeYCoord
   x2 = Nodes(Node2).NodeXCoord
   y2 = Nodes(Node2).NodeYCoord

   'convert to normal coordinates
   y1 = 480 - y1
   y2 = 480 - y2

   'compute angle theta
   IF (x1 = x2) AND (y1 > y2) THEN      'pos y axis
      Theta = XRadians!(90)
   ELSEIF (x1 = x2) AND (y1 < y2) THEN  'neg y axis
      Theta = XRadians!(270)
   ELSEIF (y1 = y2) AND (x1 > x2) THEN  'pos x axis
      Theta = XRadians!(0)
   ELSEIF (y1 = y2) AND (x1 < x2) THEN  'neg x axis
      Theta = XRadians!(180)
   ELSEIF (x1 > x2) AND (y1 > y2) THEN  'quad I
      Theta = ATN((y1 - y2) / (x1 - x2))
   ELSEIF (x1 < x2) AND (y1 > y2) THEN  'quad II
      Theta = XRadians!(180) + ATN((y1 - y2) / (x1 - x2))
   ELSEIF (x1 < x2) AND (y1 < y2) THEN  'quad III
      Theta = XRadians!(180) + ATN((y1 - y2) / (x1 - x2))
   ELSE                                 'quad IV
      Theta = ATN((y1 - y2) / (x1 - x2))
   END IF

   'compute x,y, intersection of the edge & node; also, pt of arrowhead
   x = x2 + NODERADIUS * COS(Theta)
   y = y2 + NODERADIUS * SIN(Theta)

   'compute (x3,y3) and (x4,y4), the "corners" of the arrowhead
   x4 = x + 10 * COS(Theta - 3.141593 / 6)
   y4 = y + 10 * SIN(Theta - 3.141593 / 6)
   x3 = x + 10 * COS(Theta + 3.141593 / 6)
   y3 = y + 10 * SIN(Theta + 3.141593 / 6)
```

```
    'compute xp,yp, coordinates for painting arrowhead
    xp = x + (NODERADIUS \ 2) * COS(Theta)
    yp = y + (NODERADIUS \ 2) * SIN(Theta)

    'convert all coordinates to screen coordinates
    y = 480 - y
    y1 = 480 - y1
    y2 = 480 - y2
    y3 = 480 - y3
    y4 = 480 - y4
    yp = 480 - yp

    'draw arrowhead & fill
    LINE (x, y)-(x3, y3), LTRED
    LINE (x3, y3)-(x4, y4), LTRED
    LINE (x4, y4)-(x, y), LTRED
    PAINT (xp, yp), LTRED, LTRED

END SUB

' ------------------------------------------------------------------
' EdgeChangeColor: set edge color to a new value.
' ------------------------------------------------------------------
SUB EdgeChangeColor (e AS GraphEdge, Kolor)
  e.EdgeColor = Kolor
END SUB

' ------------------------------------------------------------------
' EdgeCreate: adds an edge to the Edge Table. The current color
' and the two nodes connected by the edge are recorded. E is an
' available edge found using the FindFreeEdge routine.
' ------------------------------------------------------------------
SUB EdgeCreate (e AS GraphEdge, EColor, Node1, Node2)
SHARED Graph() AS INTEGER
SHARED Session$

  e.EdgeColor = EColor
  e.Node1 = Node1
  e.Node2 = Node2

END SUB

' ------------------------------------------------------------------
' EdgeDelete: Remove an edge from a graph by setting the corresponding
' elements in the adjacency matrix to 0.
' ------------------------------------------------------------------
SUB EdgeDelete (G() AS INTEGER, Node1, Node2)
  G(Node1, Node2) = 0
  G(Node2, Node1) = 0
END SUB

' ------------------------------------------------------------------
' EdgeFind: Given the indexes of two nodes, return the index in the
' edge table of the edge which connects the two nodes. If the two
' specified nodes are not connected by an edge return 0 (FALSE).
' ------------------------------------------------------------------
SUB EdgeFind (Edges() AS GraphEdge, Node1, Node2, EIndex)
  Found = FALSE
  i = 1
  WHILE (i <= MAXEDGES) AND NOT (Found)
```

```
    IF (Edges(i).Node1 = Node1) AND (Edges(i).Node2 = Node2) THEN
      Found = TRUE
    ELSEIF (Edges(i).Node2 = Node1) AND (Edges(i).Node1 = Node2) THEN
      Found = TRUE
    ELSE
      i = i + 1
    END IF
  WEND
  IF Found = TRUE THEN EIndex = i ELSE EIndex = 0
END SUB

' ------------------------------------------------------------------
' EdgeFindFree: Unused entries in the EdgeTable are marked with -1
' in the EdgeColor field. This routine searches for the first unused
' entry and returns its index.
' ------------------------------------------------------------------
SUB EdgeFindFree (Edges() AS GraphEdge, EdgeIndex)
  Found = FALSE
  i = 1
  WHILE NOT (Found)
    IF Edges(i).EdgeColor = -1 THEN
      Found = TRUE
      EdgeIndex = i
    ELSE
      i = i + 1
    END IF
  WEND
END SUB

' ------------------------------------------------------------------
' EdgeHide: Remove an edge from the edge table and clears it from
' the user display.
' ------------------------------------------------------------------
SUB EdgeHide (e AS GraphEdge)
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge

  e.EdgeColor = -1
  GraphShow Nodes(), Edges()
END SUB

' ------------------------------------------------------------------
' EdgeHighlight: Make edge visible on the user display with the outer
' portion of the edge in the specified color to highlight the edge.
' ------------------------------------------------------------------
SUB EdgeHighlight (e AS GraphEdge, Nodes() AS GraphNode, Kolor)
  MouseHide
  x1 = Nodes(e.Node1).NodeXCoord
  y1 = Nodes(e.Node1).NodeYCoord
  x2 = Nodes(e.Node2).NodeXCoord
  y2 = Nodes(e.Node2).NodeYCoord

  IF y1 > y2 THEN       'swap nodes for drawing
     temp = y1
     y1 = y2
     y2 = temp
     temp = x1
     x1 = x2
     x2 = temp
  END IF
  IF (x2 >= x1) THEN
    LINE (x1 + 2, y1 - 2)-(x2 + 2, y2 - 2), Kolor
    LINE (x1 - 2, y1 + 2)-(x2 - 2, y2 + 2), Kolor
  ELSE
    LINE (x1 + 2, y1 + 2)-(x2 + 2, y2 + 2), Kolor
    LINE (x1 - 2, y1 - 2)-(x2 - 2, y2 - 2), Kolor
  END IF
  NodeShow e.Node1
  NodeShow e.Node2
  MouseShow
END SUB

' ------------------------------------------------------------------
' EdgeShow: Make edge visible on the user display. The indexes of
' the two nodes connected by the edge are kept in the fields Node1
' and Node2. These are used to access the Node Pool to determine
' the screen locations of the nodes. If Digraph is TRUE then Node1
' is the source node and Node2 is the destination node.
' ------------------------------------------------------------------
SUB EdgeShow (e AS GraphEdge, Nodes() AS GraphNode)
SHARED Digraph AS INTEGER

  MouseHide
  x1 = Nodes(e.Node1).NodeXCoord
  y1 = Nodes(e.Node1).NodeYCoord
  x2 = Nodes(e.Node2).NodeXCoord
  y2 = Nodes(e.Node2).NodeYCoord

  IF y1 > y2 THEN       'swap nodes for drawing
     temp = y1
     y1 = y2
     y2 = temp
     temp = x1
     x1 = x2
     x2 = temp
  END IF

  IF x2 >= x1 THEN
    IF NOT Digraph THEN           'thinner edge line for digraph
      LINE (x1 + 2, y1 - 2)-(x2 + 2, y2 - 2), e.EdgeColor
      LINE (x1 - 2, y1 + 2)-(x2 - 2, y2 + 2), e.EdgeColor
    END IF
    LINE (x1 + 1, y1 - 1)-(x2 + 1, y2 - 1), e.EdgeColor
    LINE (x1 - 1, y1 + 1)-(x2 - 1, y2 + 1), e.EdgeColor
  ELSE
    IF NOT Digraph THEN
      LINE (x1 + 2, y1 + 2)-(x2 + 2, y2 + 2), e.EdgeColor
      LINE (x1 - 2, y1 - 2)-(x2 - 2, y2 - 2), e.EdgeColor
    END IF
    LINE (x1 + 1, y1 + 1)-(x2 + 1, y2 + 1), e.EdgeColor
    LINE (x1 - 1, y1 - 1)-(x2 - 1, y2 - 1), e.EdgeColor
  END IF
  LINE (x1, y1)-(x2, y2), e.EdgeColor
  NodeShow e.Node1
  NodeShow e.Node2
  IF Digraph THEN
     EdgeArrowHead e
  END IF
  MouseShow
END SUB
```

```
' ----------------------------------------------------------------
' EdgeTableInit: Sets all entries in the edge table to unused (-1).
' ----------------------------------------------------------------
SUB EdgeTableInit (Edges() AS GraphEdge)
  FOR i = 1 TO MAXEDGES
     Edges(i).EdgeColor = -1
  NEXT i
END SUB

' ----------------------------------------------------------------
' GraphBreadthFirstTraversal: perform a breadth first traversal of
' graph.
' ----------------------------------------------------------------
SUB GraphBreadthFirstTraversal (G() AS INTEGER, START)
SHARED Visited() AS INTEGER
SHARED InQueue() AS INTEGER
SHARED Queue() AS INTEGER
SHARED Head AS INTEGER
SHARED Tail AS INTEGER
SHARED Nodes() AS GraphNode
SHARED ActualNodes AS INTEGER

  'Initialize Data Structures
  FOR k = 1 TO ActualNodes
    InQueue(k) = FALSE
    Queue(k) = 0
    Visited = FALSE
  NEXT k
  QueueInit Head, Tail
  QueueAdd Queue(), Head, Tail, START
  InQueue(START) = TRUE
  WHILE NOT (QueueEmpty(Head, Tail))
    QueueRemove Queue(), Head, Tail, CurNode
    Visited(CurNode) = TRUE
    NodeChangeColor Nodes(CurNode), LTRED
    NodeShow CurNode
    SOUND 440, 1
    XDelay (1)
    FOR k = 1 TO ActualNodes
      IF NOT (InQueue(k)) AND (G(CurNode, k) <> 0) THEN
      QueueAdd Queue(), Head, Tail, k
        InQueue(k) = TRUE
      END IF
    NEXT k
  WEND
END SUB

' ----------------------------------------------------------------
' GraphDepthFirstTraversal: perform a depth first traversal of the
' graph beginning at the indicated start index.
' ----------------------------------------------------------------
SUB GraphDepthFirstTraversal (G() AS INTEGER, START)
SHARED Visited() AS INTEGER
SHARED ActualNodes AS INTEGER

  FOR k = 1 TO ActualNodes
     Visited(k) = FALSE
  NEXT k
  GraphSearchFrom START
END SUB
```

```
' ---------------------------------------------------------------
' GraphEmpty: Returns TRUE if a graph is empty.
' ---------------------------------------------------------------
FUNCTION GraphEmpty (G() AS INTEGER)
SHARED ActualNodes AS INTEGER

  GraphEmpty = TRUE
  FOR i = 1 TO ActualNodes
    FOR j = 1 TO ActualNodes
      IF G(i, j) <> 0 THEN GraphEmpty = FALSE
    NEXT j
  NEXT i
END FUNCTION

' ========================================================
' GraphHighlightMatrix: Highlights a column in the adjacency
' matrix.
' ========================================================
SUB GraphHighlightMatrix (G() AS INTEGER, row AS INTEGER, col AS
INTEGER)
SHARED ActualNodes AS INTEGER
SHARED MatrixRegion AS Region

   IF G(row, col) = 0 THEN
      Kolor = MEDLTGRAY
   ELSE
      Kolor = ROBINSEGG
   END IF
   RegionPrint MatrixRegion, row + 2, 2 * col + 5, STR$(G(row, col)),
Kolor

END SUB

' ---------------------------------------------------------------
' GraphInit: Takes a graph in an unknown state and initializes it
' to an empty graph in which all edge connections are set to 0.
' ---------------------------------------------------------------
SUB GraphInit (G() AS INTEGER)

  FOR i = 1 TO MAXNODES
    FOR j = 1 TO MAXNODES
      G(i, j) = 0
    NEXT j
  NEXT i
END SUB

' ---------------------------------------------------------------
' GraphSearchFrom: Main routine of the depth-first traversal. This
' subroutine is recursive.
' ---------------------------------------------------------------
SUB GraphSearchFrom (NodeIndex)
SHARED Visited() AS INTEGER
SHARED Graph() AS INTEGER
SHARED Nodes() AS GraphNode
SHARED ActualNodes AS INTEGER

  Visited(NodeIndex) = TRUE
  NodeChangeColor Nodes(NodeIndex), LTRED
  NodeShow NodeIndex
  SOUND 440, 1
  XDelay (1)
  FOR j = 1 TO ActualNodes
    IF NOT (Visited(j)) AND (Graph(NodeIndex, j) <> 0) THEN
      GraphSearchFrom j
    END IF
  NEXT j
END SUB

' --------------------------------------------------------------------
' GraphShow: Show the graph on the user display. Nodes to be displayed
' are in the Node Pool with the NodeVisible attribute set to TRUE.
' Edges in the Edge Table are included if the EdgeColor is not -1.
' --------------------------------------------------------------------
SUB GraphShow (Nodes() AS GraphNode, Edges() AS GraphEdge)
DIM i AS INTEGER
SHARED WorkRegion AS Region
SHARED ActualNodes AS INTEGER

  LabResetWorkRegion
  FOR i = 1 TO ActualNodes
    IF Nodes(i).NodeVisible = TRUE THEN
      NodeShow i
    END IF
  NEXT i
  FOR i = 1 TO MAXEDGES
    IF Edges(i).EdgeColor <> -1 THEN
      EdgeShow Edges(i), Nodes()
    END IF
  NEXT i
END SUB

' --------------------------------------------------------------------
' GraphShowMatrix: Displays the adjacency matrix for a graph or
' network in the MatrixRegion special window.
' --------------------------------------------------------------------
SUB GraphShowMatrix (G() AS INTEGER)
SHARED MatrixRegion AS Region
SHARED ActualNodes AS INTEGER

  RegionPrint MatrixRegion, 1, 10, "ADJACENCY MATRIX", ROBINSEGG
  RegionPrint MatrixRegion, 2, 8, "A B C D E F G H I J K L", WHITE
  FOR row = 1 TO ActualNodes
    RegionPrint MatrixRegion, row + 2, 5, CHR$(row + 64), WHITE
    FOR col = 1 TO ActualNodes
      IF G(row, col) = 0 THEN
      PrintColor = MEDLTGRAY
      ELSE
      PrintColor = LTRED
      END IF
      RegionPrint MatrixRegion, row + 2, 2 * col + 5, STR$(G(row, col)),
PrintColor
    NEXT col
  NEXT row
END SUB

' --------------------------------------------------------------------
' LabButtonService: Services a button press by calling the appropriate
' subroutine depending on the button's ActionCode.
' --------------------------------------------------------------------
SUB LabButtonService (B AS ButtonType)
SHARED MesgRegion AS Region
SHARED MatrixRegion AS Region
SHARED DONE AS INTEGER
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED Graph() AS INTEGER
SHARED ConfirmRegion AS Region
SHARED Frame AS INTEGER
SHARED NumFrames AS INTEGER
SHARED ActualNodes AS INTEGER
SHARED Net AS INTEGER
SHARED ANS$()
SHARED FrameText$()
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER

  SELECT CASE B.ActionCode
  CASE ADDNODE
    LabDoAddNode Nodes()
  CASE ADDEDGE
    LabDoAddEdge Nodes()
  CASE ANS.A
    IF Frame >= 1 THEN
       MID$(ANS$(Frame, 5), 2, 1) = "A"
       XDelay (1)
       RegionClear MesgRegion
       RegionPrint MesgRegion, 1, 1, "A", TEXTKOLOR
    END IF
  CASE ANS.B
    IF Frame >= 1 THEN
       MID$(ANS$(Frame, 5), 2, 1) = "B"
       XDelay (1)
       RegionClear MesgRegion
       RegionPrint MesgRegion, 1, 1, "B", TEXTKOLOR
    END IF
  CASE ANS.C
    IF Frame >= 1 THEN
       MID$(ANS$(Frame, 5), 2, 1) = "C"
       XDelay (1)
       RegionClear MesgRegion
       RegionPrint MesgRegion, 1, 1, "C", TEXTKOLOR
    END IF
  CASE ANS.D
    IF Frame >= 1 THEN
       MID$(ANS$(Frame, 5), 2, 1) = "D"
       XDelay (1)
       RegionClear MesgRegion
       RegionPrint MesgRegion, 1, 1, "D", TEXTKOLOR
    END IF
  CASE RUNDEPTH
    LabDoTraversal Graph(), Nodes()
  CASE RUNBREADTH
    LabDoTraversal Graph(), Nodes()
  CASE RUNSPAN
    LabDoMinSpan Graph(), Nodes(), Edges()
  CASE RUNPATH
    LabDoShortestPath Graph(), Nodes(), Edges()
  CASE RUNSORT
    LabDoTopSort Graph(), Nodes(), Edges()
  CASE RUNDEMO
    Frame = 1
    DemoRun
  CASE STARTTEST
    Digraph = FALSE
    LabResetWorkRegion
    LabSetupGraph Graph(), 1, NumNodes(), NodeList(), NumEdges(),
EdgeList()
    GraphShow Nodes(), Edges()
    Frame = 1
    TestRun
  CASE FORWARD
    IF Frame < NumFrames THEN
      Frame = Frame + 1
      DemoRun
    ELSE
      RegionConfirm ConfirmRegion, "End of Demo, QUIT", Response
      IF Response = TRUE THEN DONE = TRUE
    END IF
  CASE NEXTQUES
    IF Frame < NumFrames THEN
      Frame = Frame + 1
      TestRun
    ELSE
      RegionConfirm ConfirmRegion, "End of Test, QUIT", Response
      IF Response = TRUE THEN DONE = TRUE
    END IF
  CASE BACK
    IF Frame > 1 THEN
       Frame = Frame - 1
       DemoRun
    ELSE
       RegionClear MesgRegion
       RegionPrint MesgRegion, 1, 1, "At start of demo!", LTRED
       XDelay (2)
       RegionClear MesgRegion
    END IF
  CASE PREVQUES
    IF Frame > 1 THEN
       Frame = Frame - 1
       TestRun
    ELSE
       RegionClear MesgRegion
       RegionPrint MesgRegion, 1, 1, "At start of test!", LTRED
       XDelay (2)
       RegionClear MesgRegion
    END IF
  CASE CLRGRAPH
    LabDoClearGraph Nodes(), Edges(), Graph()
    IF Net THEN LabShowWeights
  CASE CLRBALSA
    LabDoClearGraph Nodes(), Edges(), Graph()
    LabSetupBalsa
  CASE CLRMATRIX
    LabDoClearGraph Nodes(), Edges(), Graph()
    RegionClear MatrixRegion
    RegionBorder MatrixRegion
    GraphShowMatrix Graph()
    IF Net THEN LabShowWeights
  CASE CLRNET
    LabDoClearGraph Nodes(), Edges(), Graph()
    LabSetupNet
    LabShowWeights
```

```
   CASE CLRTEST
     TestClear ANS$(), NumFrames
     Frame = 1
     TestRun
   CASE CHECKBALSA
     LabCheckBalsa Graph()
   CASE CHECKNET
     LabCheckNet Graph()
   CASE CHECKTEST
     TestCheck FrameText$(), ANS$(), NumFrames
   CASE HELP
     LabDoHelp
   CASE QUIT
     RegionConfirm ConfirmRegion, "QUIT", Response
     IF Response = TRUE THEN DONE = TRUE
   END SELECT
 END SUB

' -------------------------------------------------------------------
' LabCheckBalsa: CHECK button action for the Balsa Airlines problem.
' Checks the graph built by the user against the correct solution.
' -------------------------------------------------------------------
SUB LabCheckBalsa (G() AS INTEGER)
SHARED MesgRegion AS Region
SHARED ActualNodes AS INTEGER
DIM ANS(1 TO MAXNODES, 1 TO MAXNODES)

  'Build solution graph
  GraphInit ANS()
  EdgeAdd ANS(), 2, 3, 1
  EdgeAdd ANS(), 2, 6, 1
  EdgeAdd ANS(), 3, 7, 1
  EdgeAdd ANS(), 2, 7, 1
  EdgeAdd ANS(), 7, 11, 1
  EdgeAdd ANS(), 6, 11, 1
  Correct = TRUE
  FOR row = 1 TO ActualNodes
    FOR col = 1 TO ActualNodes
      IF G(row, col) <> ANS(row, col) THEN Correct = FALSE
    NEXT col
  NEXT row
  RegionClear MesgRegion
  IF Correct THEN
    Mesg$ = "Solution Correct!"
  ELSE
    Mesg$ = "Sorry...Not correct."
  END IF
  RegionPrint MesgRegion, 1, 1, Mesg$, TEXTKOLOR
  XDelay (2)
  RegionClear MesgRegion
END SUB

' -------------------------------------------------------------------
' LabCheckNet: CHECK button action for the network errand problem.
' Checks the graph built by the user against the correct solution.
' -------------------------------------------------------------------
SUB LabCheckNet (G() AS INTEGER)
SHARED MesgRegion AS Region
SHARED ActualNodes AS INTEGER
DIM ANS(1 TO MAXNODES, 1 TO MAXNODES)
```

```
   'Build solution graph
   GraphInit ANS()
   EdgeAdd ANS(), 6, 7, 5
   EdgeAdd ANS(), 6, 11, 1
   EdgeAdd ANS(), 6, 10, 4
   EdgeAdd ANS(), 7, 10, 2
   EdgeAdd ANS(), 7, 11, 3
   EdgeAdd ANS(), 10, 11, 3
   Correct = TRUE
   FOR row = 1 TO ActualNodes
     FOR col = 1 TO ActualNodes
       IF G(row, col) <> ANS(row, col) THEN Correct = FALSE
     NEXT col
   NEXT row
   RegionClear MesgRegion
   IF Correct THEN
     Mesg$ = "Solution Correct!"
   ELSE
     Mesg$ = "Sorry...Not correct."
   END IF
   RegionPrint MesgRegion, 1, 1, Mesg$, TEXTKOLOR
   XDelay (2)
   RegionClear MesgRegion
END SUB

' ------------------------------------------------------------------
' LabDoAddEdge: adds an edge to the graph and displays it.
' ------------------------------------------------------------------
SUB LabDoAddEdge (Nodes() AS GraphNode)
SHARED InfoRegion AS Region
SHARED MesgRegion AS Region
SHARED Edges() AS GraphEdge
SHARED Session$
SHARED Graph() AS INTEGER

   RegionClear MesgRegion
   NumNodes = 0
   FOR i = 1 TO MAXNODES
      IF Nodes(i).NodeVisible = TRUE THEN
       NumNodes = NumNodes + 1
      END IF
   NEXT i
   IF (NumNodes < 2) THEN
      RegionPrint MesgRegion, 1, 1, "Need at least 2 nodes!", LTRED
      XDelay (3)
      RegionClear MesgRegion
   ELSE
      RegionPrint MesgRegion, 1, 1, "Click on first node at left.",
TEXTKOLOR
      Visible = FALSE
      WHILE NOT Visible
        LabGetNodeSelection Nodes(), NodeIndex1
        Visible = Nodes(NodeIndex1).NodeVisible
      WEND
      RegionClear MesgRegion
'      XDelay (1)
      RegionPrint MesgRegion, 1, 1, "Click on second node at left.",
TEXTKOLOR
      Visible = FALSE
      WHILE NOT Visible
        LabGetNodeSelection Nodes(), NodeIndex2
        IF NodeIndex2 <> NodeIndex1 THEN Visible =
Nodes(NodeIndex2).NodeVisible
      WEND
      'Get edge weight for network and set edge color.
      IF Session$ = "NETLAB" OR Session$ = "PATHLAB" OR Session$ =
"SPANLAB" OR Session$ = "NIMP01LAB" THEN
        LabGetEdgeWeight Weight
        SELECT CASE Weight        'set edge color according to weight
       CASE 1: Kolor = YELLOW
       CASE 2: Kolor = ROBINSEGG
       CASE 3: Kolor = LTGRAY
       CASE 4: Kolor = SPRINGGREEN
       CASE 5: Kolor = WHITE
        END SELECT
      ELSE
        Weight = 1                'graph case
      END IF
      EdgeFindFree Edges(), EdgeIndex
      EdgeCreate Edges(EdgeIndex), Kolor, NodeIndex1, NodeIndex2
      EdgeAdd Graph(), NodeIndex1, NodeIndex2, Weight   'update adj
matrix
      EdgeShow Edges(EdgeIndex), Nodes()
      RegionClear MesgRegion
  END IF
END SUB

' ------------------------------------------------------------------
' LabDoAddNode: adds a node to the graph and displays it on the user
' display.
' ------------------------------------------------------------------
SUB LabDoAddNode (Nodes() AS GraphNode)
SHARED InfoRegion AS Region
SHARED MesgRegion AS Region

  RegionClear MesgRegion
  RegionPrint MesgRegion, 1, 1, "Click on a node at left to add.",
TEXTKOLOR
  LabGetNodeSelection Nodes(), NodeIndex
  NodeShow NodeIndex
  RegionClear MesgRegion
END SUB

' ------------------------------------------------------------------
' LabDoClearGraph: resets the graph and clears it from the user display.
' ------------------------------------------------------------------
SUB LabDoClearGraph (Nodes() AS GraphNode, Edges() AS GraphEdge, G() AS
INTEGER)
SHARED WorkRegion AS Region
SHARED ConfirmRegion AS Region
SHARED ActualNodes AS INTEGER
SHARED NodeLabel$

  RegionConfirm ConfirmRegion, "CLEAR GRAPH", Response
  IF Response = TRUE THEN
    RegionClear WorkRegion
    FOR i = 1 TO ActualNodes
      Nodes(i).Label = MID$(NodeLabel$, i, 1)
      Nodes(i).NodeVisible = FALSE
      NodeShowDimmed Nodes(i)
    NEXT i
    GraphInit G()
    EdgeTableInit Edges()
  END IF
  XDelay (1)
END SUB

' ----------------------------------------------------------------
' LabDoHelp: Displays help for particular lab or demo.
' display.
' ----------------------------------------------------------------
SUB LabDoHelp
SHARED TextRegion AS Region
SHARED InfoRegion AS Region
SHARED MatrixRegion AS Region
SHARED Graph() AS INTEGER
SHARED Session$
SHARED SessionHelp$()
SHARED SessionDescr$()
SHARED Frame AS INTEGER
SHARED FrameText$()

  RegionClear TextRegion
  FOR i = 1 TO MAXHELP
    RegionPrint TextRegion, i, 4, SessionHelp$(i), YELLOW
  NEXT i
  RegionPrint TextRegion, MAXHELP + 1, 24, "Click HERE to continue...",
TEXTKOLOR
  x = 0: y = 0
  WHILE NOT ((x >= 216) AND (x <= 416) AND (y >= 176) AND (y <= 191))
    WaitForMousePress x, y
  WEND
  RegionClear TextRegion
  FOR i = 1 TO MAXDESCR
    IF RIGHT$(Session$, 3) = "LAB" OR (Frame = 0) THEN
       RegionPrint TextRegion, i, 4, SessionDescr$(i), TEXTKOLOR
    ELSE
       RegionPrint TextRegion, i, 4, FrameText$(Frame, i), TEXTKOLOR
    END IF
  NEXT i
  IF (Session$ = "GIMP01LAB") OR (Session$ = "NIMP01LAB") THEN
    RegionClear MatrixRegion
    RegionBorder MatrixRegion
    GraphShowMatrix Graph()
  END IF
END SUB

' ================================================================
' LabDoMinSpan: carry out the minimum spanning tree lab session.
' ================================================================
SUB LabDoMinSpan (Graph() AS INTEGER, Nodes() AS GraphNode, Edges() AS
GraphEdge)
SHARED MesgRegion AS Region
SHARED ActualNodes AS INTEGER

  IF GraphEmpty(Graph()) THEN
    RegionClear MesgRegion
    RegionPrint MesgRegion, 1, 1, "No network defined!", LTRED
    XDelay (2)
    RegionClear MesgRegion
  ELSE
    FOR i = 1 TO ActualNodes 'redisplay nodes in original color
      IF Nodes(i).NodeVisible THEN
```

```
          NodeShow i
        END IF
      NEXT i
      NetMinSpanTree Graph()
      'Restore original color
      FOR i = 1 TO ActualNodes
         IF Nodes(i).NodeVisible THEN
          Nodes(i).NodeColor = NODEKOLOR
         END IF
      NEXT i
   END IF
END SUB

' ================================================================
' LabDoShortestPath: carry out shortest path lab session.
' ================================================================
SUB LabDoShortestPath (G() AS INTEGER, Nodes() AS GraphNode, Edges() AS
GraphEdge)
SHARED MesgRegion AS Region
SHARED ActualNodes AS INTEGER

  IF GraphEmpty(G()) THEN
    RegionClear MesgRegion
    RegionPrint MesgRegion, 1, 1, "No graph defined!", LTRED
    XDelay (2)
    RegionClear MesgRegion
  ELSE
    FOR i = 1 TO ActualNodes 'redisplay nodes & edges in original color
      IF Nodes(i).NodeVisible THEN
       NodeShow i
      END IF
    NEXT i
    FOR i = 1 TO MAXEDGES
       IF Edges(i).EdgeColor <> -1 THEN
        EdgeShow Edges(i), Nodes()
       END IF
    NEXT i
    RegionClear MesgRegion
    RegionPrint MesgRegion, 1, 1, "Click on Start node.", TEXTKOLOR
    LabGetNodeSelection Nodes(), START
    NodeChangeColor Nodes(START), LTRED
    NodeShow START
    RegionClear MesgRegion
'    XDelay (1)
    RegionPrint MesgRegion, 1, 1, "Click on Destination node.",
TEXTKOLOR
    LabGetNodeSelection Nodes(), Dest
    RegionClear MesgRegion
    NetShortestPath G(), START, Dest

    'Restore original color
    FOR i = 1 TO ActualNodes
       IF Nodes(i).NodeVisible THEN
        Nodes(i).NodeColor = NODEKOLOR
       END IF
    NEXT i
  END IF
END SUB

' ================================================================
' LabDoTopSort: carry out topological sort lab session.
' ====================================================================
SUB LabDoTopSort (Graph() AS INTEGER, Nodes() AS GraphNode, Edges() AS
GraphEdge)
SHARED MesgRegion AS Region
SHARED ActualNodes AS INTEGER

  IF GraphEmpty(Graph()) THEN
    RegionClear MesgRegion
    RegionPrint MesgRegion, 1, 1, "No graph defined!", LTRED
    XDelay (2)
    RegionClear MesgRegion
  ELSE
    FOR i = 1 TO ActualNodes 'redisplay nodes in original color
      IF Nodes(i).NodeVisible THEN
       NodeShow i
      END IF
    NEXT i
    NetTopologicalSort Graph()
    'Restore original color
    FOR i = 1 TO ActualNodes
       IF Nodes(i).NodeVisible THEN
        Nodes(i).NodeColor = NODEKOLOR
       END IF
    NEXT i
  END IF
END SUB

' ====================================================================
' LabDoTraversal: carry out traversal lab session. Session code is
' checked to see which traversal to do.
' ====================================================================
SUB LabDoTraversal (Graph() AS INTEGER, Nodes() AS GraphNode)
SHARED MesgRegion AS Region
SHARED ActualNodes AS INTEGER
SHARED Session$

  IF GraphEmpty(Graph()) THEN
    RegionClear MesgRegion
    RegionPrint MesgRegion, 1, 1, "No graph defined!", LTRED
    XDelay (2)
    RegionClear MesgRegion
  ELSE
    FOR i = 1 TO ActualNodes 'redisplay nodes in original color
      IF Nodes(i).NodeVisible THEN
       NodeShow i
      END IF
    NEXT i
    RegionClear MesgRegion
    RegionPrint MesgRegion, 1, 1, "Click on node to start.", TEXTKOLOR
    LabGetNodeSelection Nodes(), NodeIndex
    RegionClear MesgRegion
    IF (LEFT$(Session$, 5) = "BREAD") THEN
      GraphBreadthFirstTraversal Graph(), NodeIndex
    ELSE
      GraphDepthFirstTraversal Graph(), NodeIndex
    END IF
    'Restore original color
    FOR i = 1 TO ActualNodes
       IF Nodes(i).NodeVisible THEN
        Nodes(i).NodeColor = NODEKOLOR
       END IF
    NEXT i
 END IF
END SUB

' ----------------------------------------------------------------
' LabDrawLabScreen: Draws the user display including the regions and
' buttons.
' ----------------------------------------------------------------
SUB LabDrawLabScreen
SHARED DemoRegion AS Region
SHARED TextRegion AS Region
SHARED InfoRegion AS Region
SHARED MesgRegion AS Region
SHARED WorkRegion AS Region
SHARED ControlRegion AS Region
SHARED MatrixRegion AS Region
SHARED Button() AS ButtonType
SHARED Nodes() AS GraphNode
SHARED SessionDescr$()
SHARED ActualNodes AS INTEGER
SHARED Net AS INTEGER

  'Display Regions
  RegionClear DemoRegion
  LINE (DEMOX1, DEMOY1)-(DEMOX2, DEMOY2), 7, BF   'Demo border
  RegionBorder WorkRegion
  RegionBorder ControlRegion
  RegionClear InfoRegion
  RegionClear MesgRegion

  'Display Buttons
  FOR i = 1 TO MAXBUTTONS
    ButtonDraw Button(i)
  NEXT i

  'Display Dimmed Nodes
  FOR i = 1 TO ActualNodes
     NodeShowDimmed Nodes(i)
  NEXT i

  'Display Description
  RegionClear TextRegion
  FOR i = 1 TO MAXDESCR
    RegionPrint TextRegion, i, 4, SessionDescr$(i), TEXTKOLOR
  NEXT i
  MouseReset
  MouseShow                    'turn on for the first time
END SUB

' ================================================================
' LabGetEdgeWeight:Show edge colors and prompt for user selection.
' ================================================================
SUB LabGetEdgeWeight (Weight)
SHARED MesgRegion AS Region

  RegionClear MesgRegion
'  XDelay (1)
  RegionPrint MesgRegion, 1, 1, "Select Edge Weight 1 thru 5", TEXTKOLOR
  WeightSelected = FALSE
  WHILE NOT WeightSelected
    WaitForMousePress x, y
    IF (448 <= y) AND (y <= 462) THEN
       FOR i = 1 TO 5
        IF (120 + i * 32 <= x) AND (x <= 142 + i * 32) THEN
           WeightSelected = TRUE
           Weight = i
        END IF
       NEXT i
    END IF
  WEND
END SUB

'------------------------------------------------------------------
' LabGetNodeSelection: Waits for the user to select a node in the work
' area and returns the index in the node pool of the selected node.
' ------------------------------------------------------------------
SUB LabGetNodeSelection (Nodes() AS GraphNode, NodeIndex)
SHARED ActualNodes AS INTEGER

  NodeSelected = FALSE
  WHILE NOT NodeSelected
    WaitForMouseClick x, y
    FOR i = 1 TO ActualNodes
      x1 = Nodes(i).NodeXCoord - NODERADIUS
      x2 = Nodes(i).NodeXCoord + NODERADIUS
      y1 = Nodes(i).NodeYCoord - NODERADIUS
      y2 = Nodes(i).NodeYCoord + NODERADIUS
      IF ((x >= x1) AND (x <= x2) AND (y >= y1) AND (y <= y2)) THEN
       NodeSelected = TRUE
       NodeIndex = i
      END IF
    NEXT i
  WEND
END SUB

' ------------------------------------------------------------------
' LabGetSessionData: Retrieves the appropriate data from the lab
' data file LABDATA.DAT and stores the data for use by the program.
' The session code is used to access the correct DATA.
' ------------------------------------------------------------------
SUB LabGetSessionData (Code$, Descr$(), SHelp$(), Button() AS
ButtonType, NLabel$, ActNodes)

  OPEN "LABDATA.DAT" FOR INPUT AS #1
  DO                                'get data until correct session
    INPUT #1, SessionCode$
    INPUT #1, NumDescr              'how many descr lines?
    FOR i = 1 TO NumDescr           'get description
       INPUT #1, Descr$(i)
    NEXT i
    FOR i = NumDescr + 1 TO MAXDESCR 'fill in blank lines
       Descr$(i) = ""
    NEXT i
    INPUT #1, NumHelp               'how many help lines?
    FOR i = 1 TO NumHelp            'get help
       INPUT #1, SHelp$(i)
    NEXT i
    FOR i = NumHelp + 1 TO MAXHELP  'fill in blank lines
       SHelp$(i) = ""
    NEXT i
    FOR i = 1 TO MAXBUTTONS         'skip button data
       INPUT #1, ButtonText$, TextColor, x1, y1, x2, y2, Action, Active
      ButtonCreate Button(i), x1, y1, x2, y2, BUTTONLIGHT, BUTTONDARK,
BUTTONFACE, TextColor, ButtonText$, Action, Active
    NEXT i
    INPUT #1, NLabel$
    INPUT #1, ActNodes
  LOOP UNTIL SessionCode$ = Code$
  CLOSE #1
END SUB

' -----------------------------------------------------------------
' LabGetUserAction: Waits for a mouse click, then searches the Button
' Table to see which button, if any, was selected.
' -----------------------------------------------------------------
SUB LabGetUserAction (choice)
SHARED Button() AS ButtonType

  WaitForMousePress x, y
  choice = 0
  DONE = FALSE
  WHILE (choice < MAXBUTTONS) AND (DONE = FALSE)
    choice = choice + 1
    x1 = Button(choice).x1
    y1 = Button(choice).y1
    x2 = Button(choice).x2
    y2 = Button(choice).y2
    IF (x >= x1) AND (x <= x2) AND (y >= y1) AND (y <= y2) THEN
      DONE = TRUE
    END IF
  WEND
  IF (DONE = FALSE) THEN choice = 0
END SUB

' -----------------------------------------------------------------
' LabResetWorkRegion: Clears WorkRegion and shows dimmed nodes.
' -----------------------------------------------------------------
SUB LabResetWorkRegion
SHARED WorkRegion AS Region
SHARED Nodes() AS GraphNode
SHARED ActualNodes AS INTEGER

  RegionClear WorkRegion
  FOR i = 1 TO ActualNodes
    NodeShowDimmed Nodes(i)
  NEXT i
END SUB

' -----------------------------------------------------------------
' LabSetPalette: Sets the palette to that used in the NeoBook portion
' of the courseware.
' -----------------------------------------------------------------
SUB LabSetPalette (Colors() AS SINGLE)
   Colors(0) = XColorValue(0, 0, 0)      'Black
   Colors(1) = XColorValue(0, 0, 42)     'Blue
   Colors(2) = XColorValue(63, 63, 21)   'Yellow
   Colors(3) = XColorValue(45, 63, 63)   'RobinsEgg
   Colors(4) = XColorValue(42, 0, 0)     'Red
   Colors(5) = XColorValue(50, 50, 50)   'LtGray
   Colors(6) = XColorValue(42, 42, 42)   'MedLtGray
   Colors(7) = XColorValue(32, 32, 32)   'MedGray
   Colors(8) = XColorValue(24, 24, 24)   'MedDkGray
   Colors(9) = XColorValue(17, 17, 17)   'DkGray
   Colors(10) = XColorValue(39, 63, 31)  'SpringGreen
   Colors(11) = XColorValue(63, 21, 21)  'LtRed
   Colors(12) = XColorValue(63, 16, 0)   'Persimmon
   Colors(13) = XColorValue(42, 21, 0)   'Cinnamon
   Colors(14) = XColorValue(63, 63, 45)  'Vanilla
   Colors(15) = XColorValue(63, 63, 63)  'White
   FOR k = 0 TO 15
     PALETTE k, Colors!(k)
   NEXT k
END SUB

' ------------------------------------------------------------
' LabSetupBalsa: Displays nodes for Balsa Airlines problem.
' ------------------------------------------------------------
SUB LabSetupBalsa
   NodeShow 2
   NodeShow 3
   NodeShow 6
   NodeShow 7
   NodeShow 11
END SUB

' ------------------------------------------------------------
' LabSetUpGraph: Sets up a graph based on specified nodes and edges.
' GNum indicates which of two lists of nodes as edges to use.
' ------------------------------------------------------------
SUB LabSetupGraph (G() AS INTEGER, GNum, NumNodes(), NodeList(),
NumEdges(), EdgeList())
SHARED NodeLabel$
SHARED Edges() AS GraphEdge
SHARED Nodes() AS GraphNode

   GraphInit G()
   NodePoolInit Nodes()
   EdgeTableInit Edges()
   FOR i = 1 TO NumNodes(GNum)          'create nodes
      N = NodeList(GNum, i)
      NodeCenter N, x, y
      NodeCreate Nodes(N), MID$(NodeLabel$, N, 1), NODEKOLOR,
LABELKOLOR, x, y
   NEXT i
   FOR i = 1 TO NumEdges(GNum)             'create edges & add to graph
      N1 = EdgeList(GNum, i, 1)            'first node for edge
      N2 = EdgeList(GNum, i, 2)            'second node for edge
      Weight = EdgeList(GNum, i, 3)        'edge weight
      EdgeFindFree Edges(), EIndex
      SELECT CASE Weight       'set edge color according to weight
      CASE 1: Kolor = YELLOW
      CASE 2: Kolor = ROBINSEGG
      CASE 3: Kolor = LTGRAY
      CASE 4: Kolor = SPRINGGREEN
      CASE 5: Kolor = WHITE
      END SELECT
      EdgeCreate Edges(EIndex), Kolor, N1, N2
      EdgeAdd G(), N1, N2, Weight   'update adj matrix
   NEXT i

END SUB

' ------------------------------------------------------------
' LabSetupMatrix: Display MatrixRegion and adjacency matrix. Also,
' deactivate and clear nodes on the bottom row of the work area.
' -------------------------------------------------------------------
SUB LabSetupmatrix (G() AS INTEGER)
SHARED MatrixRegion AS Region

   RegionClear MatrixRegion
   RegionBorder MatrixRegion
   GraphShowMatrix G()
END SUB

' -------------------------------------------------------------------
' LabSetupNet: Displays nodes for Network Errand problem.
' -------------------------------------------------------------------
SUB LabSetupNet
   NodeShow 6
   NodeShow 7
   NodeShow 10
   NodeShow 11
END SUB

' -------------------------------------------------------------------
' LabSetupSession: Does special actions at the start of a session.
' -------------------------------------------------------------------
SUB LabSetupSession (Session$)
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED Graph() AS INTEGER
SHARED Net AS INTEGER
SHARED Digraph AS INTEGER
SHARED ShowMatrix AS INTEGER

  SELECT CASE Session$
    CASE "GRAPHLAB"
      LabSetupBalsa
      Net = FALSE
      Digraph = FALSE
      ShowMatrix = FALSE
    CASE "NETLAB"
      LabSetupNet
      Net = TRUE
      Digraph = FALSE
      ShowMatrix = FALSE
      LabShowWeights
    CASE "PATHLAB"
      Net = TRUE
      Digraph = FALSE
      ShowMatrix = FALSE
      LabShowWeights
    CASE "SPANLAB"
      Net = TRUE
      Digraph = FALSE
      ShowMatrix = FALSE
      LabShowWeights
    CASE "SORTLAB"
      Net = FALSE
      Digraph = TRUE
      ShowMatrix = FALSE
    CASE "GIMP01LAB"
      LabSetupmatrix Graph()
      Net = FALSE
      Digraph = FALSE
      ShowMatrix = TRUE
    CASE "NIMP01LAB"
      LabSetupmatrix Graph()
      Net = TRUE
      Digraph = FALSE
      ShowMatrix = TRUE
      LabShowWeights
    END SELECT
END SUB

' ====================================================================
' LabShowWeights: Displays the color key for edge weights for
' networks in the lower part of the Work Area. The constants
' WGHTX1 and WGHTY1 are coordinates selected to simplify displaying
' the color boxes.
' ====================================================================
SUB LabShowWeights
SHARED MesgRegion AS Region

  MouseHide
  col = WGHTX1 \ 8 - 8              'position for text
  row = WGHTY1 \ 16 + 1
  XPrintText row, col, "                             ", TEXTKOLOR
  XPrintText row, col, "Edge Weight:1   2   3   4   5", TEXTKOLOR
  FOR i = 1 TO 5
    x1 = WGHTX1 + i * 32
    x2 = x1 + 20
    LINE (x1, WGHTY1 + 14)-(x2, WGHTY1), LTGRAY, B
  NEXT i
  PAINT (WGHTX1 + 34, WGHTY1 + 7), YELLOW, LTGRAY
  PAINT (WGHTX1 + 66, WGHTY1 + 7), ROBINSEGG, LTGRAY
  PAINT (WGHTX1 + 98, WGHTY1 + 7), LTGRAY, LTGRAY
  PAINT (WGHTX1 + 130, WGHTY1 + 7), SPRINGGREEN, LTGRAY
  PAINT (WGHTX1 + 162, WGHTY1 + 7), WHITE, LTGRAY
  MouseShow
END SUB

' --------------------------------------------------------------------
' LabTheEnd: prints a closing message for the session.
' --------------------------------------------------------------------
SUB LabTheEnd
SHARED DemoRegion AS Region

  MouseHide                      'hide it for good
  RegionClear DemoRegion
  XPrintText 15, 33, "End of Session", TEXTKOLOR
  XDelay (1)
END SUB

' --------------------------------------------------------------------
' LabTitleScreen: prints an opening title screen
' --------------------------------------------------------------------
SUB LabTitleScreen
SHARED DemoRegion AS Region

  RegionClear DemoRegion
  XPrintText 12, 23, "INTRODUCTION TO GRAPHS & NETWORKS", TEXTKOLOR
  XPrintText 15, 24, "Laboratory/Demonstration Session", TEXTKOLOR
  XPrintText 18, 30, "by Thomas E. Beutel", TEXTKOLOR
  XPrintText 19, 25, "Mount Vernon Nazarene College", TEXTKOLOR
  XPrintText 20, 35, "Mar 1997", TEXTKOLOR
  XDelay (2)
END SUB

' ===================================================================
' NetAllIncluded: Returns TRUE if all elements of Visited are TRUE.
' ===================================================================
FUNCTION NetAllIncluded (Visited() AS INTEGER)
SHARED ActualNodes AS INTEGER
SHARED Nodes() AS GraphNode

  AllIncluded = TRUE
  FOR i = 1 TO ActualNodes
     IF Nodes(i).NodeVisible THEN
      IF NOT (Visited(i)) THEN
         AllIncluded = FALSE
      END IF
     END IF
  NEXT i
  NetAllIncluded = AllIncluded
END FUNCTION

' ===================================================================
' NetFindMin: Given a network, N, return the indexes of two nodes,
' i and j, such that i is included in the minimum spanning tree for
' the network, j is not yet included, and the edge weight between
' the two nodes is a minimum of the available nodes.
' ===================================================================
SUB NetFindMin (N() AS INTEGER, Node1, Node2)
SHARED Visited() AS INTEGER
SHARED ActualNodes AS INTEGER
SHARED Nodes() AS GraphNode
DIM MinWeight AS INTEGER

  MinWeight = MAXWT                    'set weight to max. value
  FOR i = 1 TO ActualNodes
    IF Nodes(i).NodeVisible THEN      'is node included in graph?
       IF Visited(i) THEN             'included in min span tree?
        FOR j = 1 TO ActualNodes
           IF Nodes(j).NodeVisible THEN
            IF NOT (Visited(j)) THEN
               IF (N(i, j) <> 0) AND (N(i, j) < MinWeight) THEN
                  Node1 = i
                  Node2 = j
                  MinWeight = N(i, j)
               END IF
            END IF
           END IF
        NEXT j
       END IF
    END IF
  NEXT i
END SUB

' ===================================================================
' NetMinSpanTree: Finds the minimum spanning tree for the network, N,
' using Prim's algo. The spanning tree is highlighted on the display.
' ===================================================================
SUB NetMinSpanTree (N() AS INTEGER)
SHARED Visited() AS INTEGER
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
SHARED ActualNodes AS INTEGER

  'initialize visited array
  FOR k = 1 TO ActualNodes
     Visited(k) = FALSE
  NEXT k

  'find first node in Node Pool which is actually in the network
  k = 1
  WHILE NOT (Nodes(k).NodeVisible)
     k = k + 1
  WEND

  Visited(k) = TRUE
  NodeChangeColor Nodes(k), LTRED
  NodeShow k
  SOUND 440, 1
  XDelay (1)

  DO
     NetFindMin N(), i, j
     Visited(j) = TRUE
     NodeChangeColor Nodes(j), LTRED
     NodeShow j
     SOUND 440, 1
     XDelay (1)
     EdgeFind Edges(), i, j, EdgeIndex
     EdgeHighlight Edges(EdgeIndex), Nodes(), HIGHLTCOLOR
  LOOP UNTIL (NetAllIncluded(Visited()))
END SUB

' ------------------------------------------------------------------
' NetShortestPath: Finds the shortest path between two specified
' nodes using Dijkstra's algorithm.
' ------------------------------------------------------------------
SUB NetShortestPath (N() AS INTEGER, START, Dest)
DIM Included(1 TO MAXNODES) AS INTEGER  'mark nodes as included
DIM Distance(1 TO MAXNODES) AS INTEGER  'distance to node from start
DIM Path(1 TO MAXNODES) AS INTEGER      'next node on path
SHARED ActualNodes AS INTEGER
SHARED Nodes() AS GraphNode

   'intialize arrays
   FOR i = 1 TO ActualNodes
      Included(i) = FALSE
      IF (N(START, i) <> 0) THEN
       Distance(i) = N(START, i)
      ELSE
       Distance(i) = MAXWT
      END IF
      IF (N(START, i) <> 0) THEN
       Path(i) = START
      ELSE
       Path(i) = 0
      END IF
   NEXT i
   Included(START) = TRUE

   'Find all shortest paths
   DO
      NextNode = START
      NextNodeDist = Distance(NextNode)
      FOR i = 1 TO ActualNodes
       IF (Distance(i) < NextNodeDist) AND NOT (Included(i)) AND
Nodes(i).NodeVisible THEN
          NextNode = i
          NextNodeDist = Distance(i)
       END IF
      NEXT i
      Included(NextNode) = TRUE

      FOR i = 1 TO ActualNodes
       IF (N(NextNode, i) <> 0) AND NOT (Included(i)) THEN 'node
connected
          IF NextNodeDist + N(NextNode, i) < Distance(i) THEN
            Distance(i) = NextNodeDist + N(NextNode, i)
             Path(i) = NextNode
          END IF
       END IF
      NEXT i
   LOOP UNTIL (NetAllIncluded(Included()))
   NetShowShortestPath Path(), START, Dest
END SUB

' -------------------------------------------------------------
' NetShowShortestPath: Recursively highlights the nodes and edges
' in the shortest path from Start to Dest as detemined by the
' shortest path algorithm.
' -------------------------------------------------------------
SUB NetShowShortestPath (Path() AS INTEGER, START, Dest)
SHARED Nodes() AS GraphNode
SHARED Edges() AS GraphEdge
DIM Last AS INTEGER             'local copy for when return

   Last = Dest
   IF (Last <> START) THEN
      NetShowShortestPath Path(), START, Path(Last)
      Node1 = Last
      Node2 = Path(Last)
      NodeChangeColor Nodes(Last), LTRED
      NodeShow Last
      SOUND 440, 1
      XDelay (1)
      EdgeFind Edges(), Node1, Node2, EIndex
      EdgeHighlight Edges(EIndex), Nodes(), HIGHLTCOLOR
   END IF
END SUB

' -------------------------------------------------------------
' NetTopologicalSort: Performs a topological sort of the nodes in a
' network.
' -------------------------------------------------------------
SUB NetTopologicalSort (G() AS INTEGER)
DIM Queue(1 TO MAXNODES) AS INTEGER
DIM Head AS INTEGER
DIM Tail AS INTEGER
DIM InDegree(1 TO MAXNODES) AS INTEGER
DIM OrderedList(1 TO MAXNODES) AS INTEGER
DIM NODE AS INTEGER
DIM TNode AS INTEGER
SHARED ActualNodes AS INTEGER
SHARED Nodes() AS GraphNode
```

```
    'set initial indegree for each node. A node has an incoming edge
    'if its column entry in the adjacency matrix is not zero
    FOR col = 1 TO ActualNodes
       InDegree(col) = 0
       FOR row = 1 TO ActualNodes
       IF G(row, col) <> 0 THEN
          InDegree(col) = InDegree(col) + 1
       END IF
       NEXT row
    NEXT col
    QueueInit Head, Tail
    FOR NODE = 1 TO ActualNodes
      IF InDegree(NODE) = 0 AND Nodes(NODE).NodeVisible THEN
        QueueAdd Queue(), Head, Tail, NODE
       END IF
    NEXT NODE
    Count = 0
    WHILE NOT QueueEmpty(Head, Tail)
       QueueRemove Queue(), Head, Tail, TNode
       Count = Count + 1
       OrderedList(Count) = TNode
       FOR NODE = 1 TO ActualNodes
        IF (G(TNode, NODE) <> 0) THEN
           InDegree(NODE) = InDegree(NODE) - 1
           IF InDegree(NODE) = 0 THEN
              QueueAdd Queue(), Head, Tail, NODE
           END IF
        END IF
       NEXT NODE
    WEND

    'show nodes in topological order
    FOR i = 1 TO Count
       Nodes(OrderedList(i)).Label = CHR$(48 + i)'set label to count
       NodeShow OrderedList(i)
       SOUND 440, 1
       XDelay (1)
    NEXT i
END SUB

' ------------------------------------------------------------------
' NodeCenter: given the index of a node in the node pool return the
' x,y coordinate of the center of a circle used to represent the node.
' ------------------------------------------------------------------
SUB NodeCenter (NODE, x, y)
  x = ((NODE - 1) MOD 4) * 64 + STARTX
  y = ((NODE - 1) \ 4) * 64 + STARTY
END SUB

' ------------------------------------------------------------------
' NodeChangeColor: set the color for a node to a new value.
' ------------------------------------------------------------------
SUB NodeChangeColor (N AS GraphNode, Kolor)
  N.NodeColor = Kolor
END SUB

' ------------------------------------------------------------------
' NodeCreate: Creates an entry in the node pool with the specified
' attributes. Nodes in the node pool are not part of a graph until
' an GraphAddNode is performed. A created node is initially invisible.
' ------------------------------------------------------------------
SUB NodeCreate (N AS GraphNode, Label$, NColor, LColor, x, y)
  N.Label = Label$
  N.NodeColor = NColor
  N.LabelColor = LColor
  N.NodeXCoord = x
  N.NodeYCoord = y
  NodeVisible = FALSE
END SUB

' ------------------------------------------------------------------
' NodeHide: Hides a GraphNode on the user display. The node is
' is displayed dimmed to indicate its position but is not part
' of a graph.
' ------------------------------------------------------------------
SUB NodeHide (N AS GraphNode)
  MouseHide
  CIRCLE (N.NodeXCoord, N.NodeYCoord), NODERADIUS, MEDDKGRAY
  PAINT (N.NodeXCoord, N.NodeYCoord), MEDDKGRAY, MEDDKGRAY
  col = (N.NodeXCoord + 4) \ 8        'column for label
  N.NodeVisible = FALSE
  MouseShow
END SUB

' ------------------------------------------------------------------
' NodePoolInit: Initialize all entries in the node pool.
' ------------------------------------------------------------------
SUB NodePoolInit (Nodes() AS GraphNode)
SHARED NodeLabel$

  FOR i = 1 TO MAXNODES
    Nodes(i).Label = MID$(NodeLabel$, i, 1)
    Nodes(i).NodeColor = NODEKOLOR
    Nodes(i).LabelColor = LABELKOLOR
    NodeCenter i, Nodes(i).NodeXCoord, Nodes(i).NodeYCoord
    Nodes(i).NodeVisible = FALSE
  NEXT i
END SUB

' ------------------------------------------------------------------
' NodeShow: Displays a GraphNode on the user display. The node is
' is displayed in the color stored for the node at the coordinates
' defined for the node when it was created. These coordinates are
' used to compute the row and column for displaying the node label.
' The NodeVisible attribute is also changed to TRUE.
' ------------------------------------------------------------------
SUB NodeShow (NIndex)
SHARED Nodes() AS GraphNode

  MouseHide
  CIRCLE (Nodes(NIndex).NodeXCoord, Nodes(NIndex).NodeYCoord),
NODERADIUS, DRAWKOLOR%
  PAINT (Nodes(NIndex).NodeXCoord, Nodes(NIndex).NodeYCoord),
Nodes(NIndex).NodeColor, DRAWKOLOR

' The next two lines are commented out. See below for explanation.

   'col = INT((Nodes(NIndex).NodeXCoord + 4) / 8)       'column for
label
   'row = INT((Nodes(NIndex).NodeYCoord + 8) / 16)      'row for label
```

```
    COLOR Nodes(NIndex).LabelColor

 ' The following code is included because of an unexplained problem
 ' with the LOCATE statement. Although the program works fine within
 ' the QuickBASIC environment, when it is compiled, it crashes on
 ' the LOCATE statement when it uses row and col computed from the
 ' node's x and y coordinates!

   SELECT CASE NIndex
   CASE 1
     LOCATE 15, 9
   CASE 2
     LOCATE 15, 17
   CASE 3
     LOCATE 15, 25
   CASE 4
     LOCATE 15, 33
   CASE 5
     LOCATE 19, 9
   CASE 6
     LOCATE 19, 17
   CASE 7
     LOCATE 19, 25
   CASE 8
     LOCATE 19, 33
   CASE 9
     LOCATE 23, 9
   CASE 10
     LOCATE 23, 17
   CASE 11
     LOCATE 23, 25
   CASE 12
     LOCATE 23, 33
   CASE 13
     LOCATE 27, 9
   CASE 14
     LOCATE 27, 17
   CASE 15
     LOCATE 27, 25
   CASE 16
     LOCATE 27, 33
   END SELECT

  'LOCATE row, col
  PRINT Nodes(NIndex).Label;
  COLOR TEXTKOLOR
  Nodes(NIndex).NodeVisible = TRUE
  MouseShow
END SUB

' ------------------------------------------------------------------
' NodeShowDimmed: Displays a dimmed version of a node in the work
' area of the user display. User clicks on dimmed node to add it
' to a graph.
' ------------------------------------------------------------------
SUB NodeShowDimmed (N AS GraphNode)
  MouseHide
  CIRCLE (N.NodeXCoord, N.NodeYCoord), NODERADIUS, MEDDKGRAY
  PAINT (N.NodeXCoord, N.NodeYCoord), MEDDKGRAY, MEDDKGRAY
  MouseShow
END SUB
```

```
' ------------------------------------------------------------------
' QueueAdd: Adds an element to a queue.
' ------------------------------------------------------------------
SUB QueueAdd (Q() AS INTEGER, Head, Tail, Item)
  IF QueueEmpty(Head, Tail) THEN
     Head = Head + 1
  END IF
  Tail = Tail + 1
  Q(Tail) = Item
END SUB

' ------------------------------------------------------------------
' QueueEmpty: return TRUE if queue is empty.
' ------------------------------------------------------------------
FUNCTION QueueEmpty (Head, Tail)
  QueueEmpty = (Head = 0) AND (Tail = 0)
END FUNCTION

' ------------------------------------------------------------------
' QueueInit: Initialize a Queue to empty.
' ------------------------------------------------------------------
SUB QueueInit (Head, Tail)
  Head = 0
  Tail = 0
END SUB

' ------------------------------------------------------------------
' QueueRemove: Removes an item from a queue.
' ------------------------------------------------------------------
SUB QueueRemove (Q() AS INTEGER, Head, Tail, Item)
  IF NOT (QueueEmpty(Head, Tail)) THEN
    Item = Q(Head)
    IF Head = Tail THEN        'last item
       QueueInit Head, Tail    'reset to empty state
    ELSE
       Head = Head + 1
    END IF
  END IF
END SUB

' ------------------------------------------------------------------
' RegionBorder: Draws a 3D border around a region. The colors used
' for the border are the default BORDERDARK and BORDERLIGHT.
' ------------------------------------------------------------------
SUB RegionBorder (R AS Region)
  MouseHide
  LINE (R.x1, R.y1)-(R.x2, R.y1), BORDERDARK
  LINE (R.x1 + 1, R.y1 + 1)-(R.x2 - 1, R.y1 + 1), BORDERDARK
  LINE (R.x1, R.y1)-(R.x1, R.y2), BORDERDARK
  LINE (R.x1 + 1, R.y1 + 1)-(R.x1 + 1, R.y2 - 1), BORDERDARK
  LINE (R.x2, R.y1)-(R.x2, R.y2), BORDERLIGHT
  LINE (R.x2 - 1, R.y1 + 1)-(R.x2 - 1, R.y2 - 1), BORDERLIGHT
  LINE (R.x2, R.y2)-(R.x1, R.y2), BORDERLIGHT
  LINE (R.x2 - 1, R.y2 - 1)-(R.x1 + 1, R.y2 - 1), BORDERLIGHT
  MouseShow
END SUB

' ------------------------------------------------------------------
' RegionClear: Clears the indicated Region to its BGColor.
' ------------------------------------------------------------------
```

```
SUB RegionClear (R AS Region)
  MouseHide
  LINE (R.x1 + 2, R.y1 + 2)-(R.x2 - 2, R.y2 - 2), R.BGColor, BF
  MouseShow
END SUB

' ------------------------------------------------------------------
' RegionConfirm: Pops up a region with a message and a place to click
' in response to the message. The message is one line and is printed
' in a fixed place in the region. It identifies the operation being
' confirmed. The response is returned.
' ------------------------------------------------------------------
SUB RegionConfirm (R AS Region, Mesg$, Response)
DIM Image(1 TO 6800) AS INTEGER

  MouseHide
  GET (R.x1, R.y1)-(R.x2, R.y2), Image
  LINE (R.x1, R.y1)-(R.x2, R.y2), YELLOW, BF
  RegionClear R
  NumCols = (R.x2 - R.x1) \ 8 + 1
  col = (NumCols \ 2) - (LEN(Mesg$) \ 2)      'center Mesg$
  RegionPrint R, 1, col, Mesg$, YELLOW
  RegionPrint R, 2, 10, "Are You Sure?", YELLOW
  RegionPrint R, 5, 9, "[YES]", YELLOW
  RegionPrint R, 5, 20, "[NO]", YELLOW
  Response = -2
  MouseShow
  WHILE Response = -2
     WaitForMousePress x, y
     IF ((x >= 256) AND (x <= 295) AND (y >= 208) AND (y <= 223)) THEN
      Response = TRUE
     ELSEIF ((x >= 344) AND (x <= 375) AND (y >= 208) AND (y <= 223))
THEN
      Response = FALSE
     END IF
  WEND
  MouseHide
  PUT (R.x1, R.y1), Image, PSET
  MouseShow
END SUB

' ------------------------------------------------------------------
' RegionCreate: Creates a region at the specified coordinates with
' the specified background color.
' ------------------------------------------------------------------
SUB RegionCreate (R AS Region, x1, y1, x2, y2, BGColor)
  R.x1 = x1
  R.y1 = y1
  R.x2 = x2
  R.y2 = y2
  R.BGColor = BGColor
END SUB

' ------------------------------------------------------------------
' RegionPrint: Prints text to the specified region, at the specified
' RegionRow and RegionCol, in the given color. Within a Region rows
' and columns are labelled beginning at 1.
' ------------------------------------------------------------------
SUB RegionPrint (R AS Region, RegionRow, RegionCol, Mesg$, Kolor)
   MouseHide
   Row0 = R.y1 \ 16 + 1                'Base Row of Region
   Col0 = R.x1 \ 8 + 1                'Base Column of Region
   NumRows = (R.y2 - R.y1) \ 16 + 1
   NumCols = (R.x2 - R.x1) \ 8 + 1
   IF (RegionRow < NumRows) AND (RegionCol < NumCols) THEN
     IF (RegionCol + LEN(Mesg$)) < NumCols THEN
       LOCATE Row0 + RegionRow, Col0 + RegionCol
       COLOR Kolor
       PRINT Mesg$;
       COLOR TextColor
     END IF
   END IF
   MouseShow
END SUB

' ========================================================
' TestCheck: Check user's answers to test; indicate if
' incorrect and show correct answer.
' ========================================================
SUB TestCheck (FrameText$(), ANS$(), NumFrames AS INTEGER)
SHARED MesgRegion AS Region
SHARED InfoRegion AS Region
SHARED Frame AS INTEGER

   CorrectAns$ = LEFT$(ANS$(Frame, 5), 1)
   UserAnswer$ = RIGHT$(ANS$(Frame, 5), 1)
   IF UserAnswer$ = CorrectAns$ THEN
      RegionPrint InfoRegion, 5, 1, "CORRECT!!!" + SPACE$(20), LTRED
   ELSEIF UserAnswer$ <> " " THEN
      TestShowQuestion FrameText$(), Frame
      TestShowAns ANS$(), Frame
      RegionPrint InfoRegion, 5, 1, "INCORRECT! Correct answer is " +
CorrectAns$, LTRED
      XDelay (3)
   END IF
END SUB

' ========================================================
' TestClear: Clears the user-entered answers for a self-test
' ========================================================
SUB TestClear (ANS$(), NumFrames AS INTEGER)
SHARED ConfirmRegion AS Region

   RegionConfirm ConfirmRegion, "Clear Answers", Response
   IF Response = TRUE THEN
      FOR i = 1 TO NumFrames
       MID$(ANS$(i, 5), 2, 1) = " "
      NEXT i
   END IF
END SUB

' -----------------------------------------------------------------
' TestGetSessionData:Retrieve session data from file.
' -----------------------------------------------------------------
SUB TestGetSessionData (Code$, Descr$(), NumFrames, FText$(), ANS$(),
ActNodes)
SHARED NumNodes() AS INTEGER
SHARED NumEdges() AS INTEGER
SHARED NodeList() AS INTEGER
SHARED EdgeList() AS INTEGER

  FileName$ = Code$ + ".DAT"
```

```
   CLOSE #1
   OPEN FileName$ FOR INPUT AS #1
   INPUT #1, NumDescr               'how many descr lines?
   FOR i = 1 TO NumDescr            'get description
      INPUT #1, Descr$(i)
   NEXT i
   FOR i = NumDescr + 1 TO MAXDESCR 'fill in blank lines
      Descr$(i) = ""
   NEXT i
   INPUT #1, NumFrames             'number of questions
   FOR i = 1 TO NumFrames          'get text for each question
      INPUT #1, NumLines
      FOR j = 1 TO NumLines
       INPUT #1, FText$(i, j)
      NEXT j
      FOR j = 1 TO 5
       INPUT #1, ANS$(i, j)      'get answers
      NEXT j
   NEXT i
   INPUT #1, NumGraphs
   FOR i = 1 TO NumGraphs
      INPUT #1, NumNodes(i)
      FOR j = 1 TO NumNodes(i)
       INPUT #1, NodeList(i, j)
      NEXT j
      INPUT #1, NumEdges(i)
      FOR j = 1 TO NumEdges(i)
       INPUT #1, EdgeList(i, j, 1), EdgeList(i, j, 2), EdgeList(i, j, 3)
      NEXT j
   NEXT i
   INPUT #1, ActNodes
   CLOSE #1

END SUB

' ========================================================
' TestRun: run a self-test.
' ========================================================
SUB TestRun
SHARED FrameText$()
SHARED Frame AS INTEGER
SHARED ANS$()
SHARED MesgRegion AS Region
SHARED InfoRegion AS Region
SHARED Session$

   TestShowQuestion FrameText$(), Frame
   RegionClear InfoRegion
   RegionClear MesgRegion
   XDelay (1)
   TestShowAns ANS$(), Frame
   IF LEFT$(Session$, 1) = "N" THEN 'network test
      LabShowWeights
   END IF
END SUB

' ========================================================
' TestShowAns: Display multiple choice answers and user's
' current answer (initialized to space) in InfoRegion.
' ========================================================
SUB TestShowAns (ANS$(), Frame AS INTEGER)
```

```
SHARED MesgRegion AS Region
SHARED InfoRegion AS Region

   RegionClear MesgRegion
   RegionClear InfoRegion
   FOR i = 1 TO 4
      RegionPrint InfoRegion, i, 1, CHR$(64 + i) + ":" + ANS$(Frame, i),
TEXTKOLOR
   NEXT i
   RegionPrint MesgRegion, 1, 1, RIGHT$(ANS$(Frame, 5), 1), TEXTKOLOR

END SUB

' ========================================================
' TestShowQuestion: Display the question for the associated
' Frame from FrameText$() in the TextRegion.
' ========================================================
SUB TestShowQuestion (FrameText$(), Frame AS INTEGER)
SHARED TextRegion AS Region
SHARED NumFrames AS INTEGER

   RegionClear TextRegion
   FOR i = 1 TO MAXDESCR
      RegionPrint TextRegion, i, 4, FrameText$(Frame, i), TEXTKOLOR
   NEXT i
   RegionPrint TextRegion, 1, 63, STR$(Frame) + " OF " +
STR$(NumFrames), TEXTKOLOR

END SUB

DEFSNG A-Z
' -------------------------------------------------------------
' XColorValue: Returns an integer value corresponding to the amount
' of red, green and blue in a designated color.
' -------------------------------------------------------------
FUNCTION XColorValue! (RED!, GREEN!, BLUE!)
  XColorValue! = 65536 * BLUE! + 256 * GREEN! + RED!
END FUNCTION

DEFINT A-Z
' -------------------------------------------------------------
' XDelay: delays the specified amount of time in seconds. The time
' is read from the system clock, so the delay should not be affected
' by CPU speed.
' -------------------------------------------------------------
SUB XDelay (Seconds!)
  START! = TIMER
  Finish! = START! + Seconds!
  WHILE TIMER < Finish!
  WEND
END SUB

' -------------------------------------------------------------
' XPrintText: prints text at a specified row and column, and in a
' specified color.
' -------------------------------------------------------------
SUB XPrintText (row, col, Mesg$, Kolor)
  LOCATE row, col
  COLOR Kolor
  FOR i = 1 TO LEN(Mesg$)
    PRINT MID$(Mesg$, i, 1);
  NEXT i
END SUB

FUNCTION XRadians! (Degrees AS SINGLE)
   XRadians! = Degrees * 3.141593 / 180
END FUNCTION
```

Appendix G

Data Files for Animated Demonstrations, Lab Sessions and Self-Tests

```
"DUMMYLAB"
4
"No demo or lab session has been specified.  The user display and"
"some controls are active. You can, for example, use the ADD NODE"
"and ADD EDGE buttons to create a graph. The CLEAR, HELP and QUIT"
"buttons are also active."
1
"             Sorry...there is no help for this session."
"ADD NODE",5,322,346,412,370,1,-1
"DEL NODE",5,322,378,412,402,0,0
"ADD EDGE",5,322,410,412,434,3,-1
"DEL EDGE",5,322,442,412,466,0,0
"RUN",5,418,346,508,370,0,0
"PAUSE",5,418,378,508,402,0,0
"CONTINUE",5,418,410,508,434,0,0
"RESTART",5,418,442,508,466,0,0
"CLEAR",5,514,346,604,370,50,-1
"CHECK",5,514,378,604,402,0,0
"HELP!",10,514,410,604,434,70,-1
"QUIT",11,514,442,604,466,80,-1
"ABCDEFGHIJKLMNOP"
16
"GRAPHLAB"
7
"Balsa airlines offers roundtrip flights among the following five"
"cities: Boston, Columbus, Denver, New York and Philadelphia. Not"
"all of the flights are direct,  only the ones between Boston and"
"Columbus, Boston and Denver, Philadelphia and New York, New York"
"and Columbus, Denver and Philadelphia,  and Boston and New York."
"Add edges below to complete the graph. Nodes are identified with"
"the first letter of the city's name. Use CHECK to check answer."
4
"Use the  ADD EDGE button to build the graph in the  area below."
"Use the  CLEAR  button to  clear the  work area if you  need to"
"start over.  Use CHECK to check your answer.  Select QUIT  when"
"you are done."
"ADD NODE",5,322,346,412,370,0,0
"DEL NODE",5,322,378,412,402,0,0
"ADD EDGE",5,322,410,412,434,3,-1
"DEL EDGE",5,322,442,412,466,0,0
"RUN",5,418,346,508,370,0,0
"PAUSE",5,418,378,508,402,0,0
"CONTINUE",5,418,410,508,434,0,0
"RESTART",5,418,442,508,466,0,0
"CLEAR",5,514,346,604,370,51,-1
"CHECK",5,514,378,604,402,61,-1
"HELP!",10,514,410,604,434,70,-1
"QUIT",11,514,442,604,466,80,-1
" BC  DN   P      "
16
"DEPTHLAB"
7
"                    DEPTH-FIRST TRAVERSAL                      "
"In this lab session you will build a  graph  and then initiate a"
"depth-first traversal from various starting points. Use ADD NODE"
"and ADD EDGE to build a graph.  Use  RUN to start the traversal."
"You will be prompted to select a start node.  When the traversal"
"is complete, you may use RUN to do  another traversal specifying"
"a different start node."
4
"Use the ADD NODE and ADD EDGE buttons to build the graph in the"
```

```
"area below.  Use the  CLEAR  button to  clear the  work area if"
"you  need to start over. Use RUN to start the algorithm. Select"
"QUIT when you are done."
"ADD NODE",5,322,346,412,370,1,-1
"DEL NODE",5,322,378,412,402,0,0
"ADD EDGE",5,322,410,412,434,3,-1
"DEL EDGE",5,322,442,412,466,0,0
"RUN",5,418,346,508,370,10,-1
"PAUSE",5,418,378,508,402,0,0
"CONTINUE",5,418,410,508,434,0,0
"RESTART",5,418,442,508,466,0,0
"CLEAR",5,514,346,604,370,50,-1
"CHECK",5,514,378,604,402,0,0
"HELP!",10,514,410,604,434,70,-1
"QUIT",11,514,442,604,466,80,-1
"ABCDEFGHIJKLMNOP"
16
"BREADTHLAB"
7
"                    BREADTH-FIRST TRAVERSAL                    "
"In this lab session you will build a  graph  and then initiate a"
"breadth-first  traversal from various starting points.  Use  the"
"ADD NODE and ADD EDGE buttons to build a graph. Use RUN to start"
"the traversal. You will be prompted to select a start node. When"
"the traversal is done,  you may use  RUN to do another traversal"
"specifying a different start node."
4
"Use the ADD NODE and ADD EDGE buttons to build the graph in the"
"area below.  Use the  CLEAR  button to  clear the  work area if"
"you  need to start over. Use RUN to start the algorithm. Select"
"QUIT when you are done."
"ADD NODE",5,322,346,412,370,1,-1
"DEL NODE",5,322,378,412,402,0,0
"ADD EDGE",5,322,410,412,434,3,-1
"DEL EDGE",5,322,442,412,466,0,0
"RUN",5,418,346,508,370,11,-1
"PAUSE",5,418,378,508,402,0,0
"CONTINUE",5,418,410,508,434,0,0
"RESTART",5,418,442,508,466,0,0
"CLEAR",5,514,346,604,370,50,-1
"CHECK",5,514,378,604,402,0,0
"HELP!",10,514,410,604,434,70,-1
"QUIT",11,514,442,604,466,80,-1
"ABCDEFGHIJKLMNOP"
16
"GIMP01LAB"
7
"  GRAPH IMPLEMENTATION LAB"
"Build a graph in the work area"
"below and watch the adjacency"
"matrix.  Entries  are made in"
"the adjacency matrix as edges"
"are added.  Only twelve nodes"
"are used in this lab session."
3
"Use the ADD NODE and ADD EDGE buttons to  build a  graph in the"
"area below. Use the CLEAR button to  clear the work area if you"
"need to start over. Select QUIT when you are done."
"ADD NODE",5,322,346,412,370,1,-1
"DEL NODE",5,322,378,412,402,0,0
"ADD EDGE",5,322,410,412,434,3,-1
```

```
"DEL EDGE",5,322,442,412,466,0,0
"RUN",5,418,346,508,370,0,0
"PAUSE",5,418,378,508,402,0,0
"CONTINUE",5,418,410,508,434,0,0
"RESTART",5,418,442,508,466,0,0
"CLEAR",5,514,346,604,370,52,-1
"CHECK",5,514,378,604,402,0,0
"HELP!",10,514,410,604,434,70,-1
"QUIT",11,514,442,604,466,80,-1
"ABCDEFGHIJKLMNOP"
12
"NETLAB"
7
"You have  four  errands at four drive-thru  businesses which are"
"located relative to each other as follows (distances in blocks):"
"   BANK: 5 from FAST FOOD, 2 from VIDEO STORE, 3 from LIBRARY   "
"   FAST FOOD: 1 from LIBRARY, 4 from VIDEO STORE                "
"   VIDEO STORE: 3 from LIBRARY                                  "
"Add edges below to complete a NETWORK representing the problem. "
"Nodes are identified with the first letter of a business name.  "
4
"Use the  ADD EDGE button to build the network in the work area. "
"Use the  CLEAR  button to  clear the  work area if you  need to "
"start over.  Use CHECK to check your answer.  Select QUIT  when "
"you are done."
"ADD NODE",5,322,346,412,370,0,0
"DEL NODE",5,322,378,412,402,0,0
"ADD EDGE",5,322,410,412,434,3,-1
"DEL EDGE",5,322,442,412,466,0,0
"RUN",5,418,346,508,370,0,0
"PAUSE",5,418,378,508,402,0,0
"CONTINUE",5,418,410,508,434,0,0
"RESTART",5,418,442,508,466,0,0
"CLEAR",5,514,346,604,370,53,-1
"CHECK",5,514,378,604,402,62,-1
"HELP!",10,514,410,604,434,70,-1
"QUIT",11,514,442,604,466,80,-1
"     FB  VL     "
16
"SPANLAB"
6
"                    MINIMUM SPANNING TREE                       "
"In this lab session you will build a network  and  then  run the"
"minimum spanning tree algorithm.  Use  ADD NODE and  ADD EDGE to"
"build a network. Use RUN to start the algorithm. Nodes and edges"
"will be highighted as they are included in the  minimum spanning"
"tree."
4
"Use the ADD NODE and ADD EDGE buttons to build a network in the"
"area below.  Use the  CLEAR  button to  clear the  work area if"
"you  need to start over. Use RUN to start the algorithm. Select"
"QUIT when you are done."
"ADD NODE",5,322,346,412,370,1,-1
"DEL NODE",5,322,378,412,402,0,0
"ADD EDGE",5,322,410,412,434,3,-1
"DEL EDGE",5,322,442,412,466,0,0
"RUN",5,418,346,508,370,13,-1
"PAUSE",5,418,378,508,402,0,0
"CONTINUE",5,418,410,508,434,0,0
"RESTART",5,418,442,508,466,0,0
"CLEAR",5,514,346,604,370,50,-1
```

```
"CHECK",5,514,378,604,402,0,0
"HELP!",10,514,410,604,434,70,-1
"QUIT",11,514,442,604,466,80,-1
"ABCDEFGHIJKL"
12
"PATHLAB"
6
"                       SHORTEST PATH LAB                        "
"In this lab session you will build a network  and  then  run the"
"shortest path  algorithm to find the  shortest path  between two"
"nodes.  Use  ADD NODE and  ADD EDGE to build a network.  Use RUN"
"to start the algorithm.  Nodes and  edges will be  highighted to"
"show the shortest path."
4
"Use the ADD NODE and ADD EDGE buttons to build a network in the"
"area below.  Use the  CLEAR  button to  clear the  work area if"
"you  need to start over. Use RUN to start the algorithm. Select"
"QUIT when you are done."
"ADD NODE",5,322,346,412,370,1,-1
"DEL NODE",5,322,378,412,402,0,0
"ADD EDGE",5,322,410,412,434,3,-1
"DEL EDGE",5,322,442,412,466,0,0
"RUN",5,418,346,508,370,12,-1
"PAUSE",5,418,378,508,402,0,0
"CONTINUE",5,418,410,508,434,0,0
"RESTART",5,418,442,508,466,0,0
"CLEAR",5,514,346,604,370,50,-1
"CHECK",5,514,378,604,402,0,0
"HELP!",10,514,410,604,434,70,-1
"QUIT",11,514,442,604,466,80,-1
"ABCDEFGHIJKL"
12
"SORTLAB"
5
"                       TOPOLOGICAL SORT                         "
"In this lab  session you will build a  graph  and  then  run the"
"topological sort algorithm.  Use  ADD NODE and ADD EDGE to build"
"a graph. Use RUN to start the algorithm. Nodes will be  labelled"
"to show the topological order."
4
"Use the  ADD NODE and  ADD EDGE buttons to build a graph in the"
"area below.  Use the  CLEAR  button to  clear the  work area if"
"you  need to start over. Use RUN to start the algorithm. Select"
"QUIT when you are done."
"ADD NODE",5,322,346,412,370,1,-1
"DEL NODE",5,322,378,412,402,0,0
"ADD EDGE",5,322,410,412,434,3,-1
"DEL EDGE",5,322,442,412,466,0,0
"RUN",5,418,346,508,370,14,-1
"PAUSE",5,418,378,508,402,0,0
"CONTINUE",5,418,410,508,434,0,0
"RESTART",5,418,442,508,466,0,0
"CLEAR",5,514,346,604,370,50,-1
"CHECK",5,514,378,604,402,0,0
"HELP!",10,514,410,604,434,70,-1
"QUIT",11,514,442,604,466,80,-1
"            "
12
"NIMP01LAB"
7
" NETWORK IMPLEMENTATION LAB"
```

```
"Build a  network in the  area"
"below and watch the adjacency"
"matrix.  Entries  are made in"
"the adjacency matrix as edges"
"are added.  Only twelve nodes"
"are used in this lab session."
3
"Use the ADD NODE and ADD EDGE buttons to build a network in the"
"area below. Use the CLEAR button to  clear the work area if you"
"need to start over. Select QUIT when you are done."
"ADD NODE",5,322,346,412,370,1,-1
"DEL NODE",5,322,378,412,402,0,0
"ADD EDGE",5,322,410,412,434,3,-1
"DEL EDGE",5,322,442,412,466,0,0
"RUN",5,418,346,508,370,0,0
"PAUSE",5,418,378,508,402,0,0
"CONTINUE",5,418,410,508,434,0,0
"RESTART",5,418,442,508,466,0,0
"CLEAR",5,514,346,604,370,52,-1
"CHECK",5,514,378,604,402,0,0
"HELP!",10,514,410,604,434,70,-1
"QUIT",11,514,442,604,466,80,-1
"ABCDEFGHIJKLMNOP"
12
```

```
7
"                    INTRODUCTION TO GRAPHS DEMO"
""
"This animated demonstration presents terms and concepts related"
"to graphs.  Simple graphs, as well as  depth-first and breadth-"
"first traversals, are animated to help the  student   visualize"
"these abstract structures and operations."
"                       Click on RUN to start..."
5
4
"                              TERMINOLOGY"
""
"A graph is a nonlinear, non-hierarchical structure comprised of"
"NODES..."
1
"Some nodes are directly connected such as F and G."
3
""
"Nodes which are not directly connected are connected by a  PATH"
"such as F to K to J."
6
"                             TRAVERSALS"
"Visiting  all of the  nodes in a  graph is  called a TRAVERSAL."
"A  DEPTH-FIRST  traversal  moves as far as it can along a  path"
"before trying another path.  For example, starting at node B in"
"the graph below,  a depth-first traversal would visit the nodes"
"in the following order: B-C-G-J-N-L-E-I."
5
""
"A BREADTH-FIRST  traversal visits each node adjacent to a given"
"node before moving further along a path. For example,  starting"
"at node B in the graph below, a  breadth-first  traversal would"
"visit the nodes in the order: B-C-E-G-I-J-L-N."
2
4
6
7
10
11
4
6,7,1
7,11,1
10,11,1
6,11,1
8
2
3
5
7
9
10
12
14
7
2,3,1
3,7,1
7,10,1
7,12,1
10,14,1
2,5,1
5,9,1
```

```
7
"                    GRAPH IMPLEMENTATION DEMO"
"This animated demonstration focuses on the data structures used"
"to implement graphs. The relationship between the graph and the"
"data structure is shown, as well as the meaning of key terms as"
"they relate to the data structures."
""
"                       Click on RUN to start..."
3
7
"                         ADJACENCY MATRIX"
""
"A primary data structure used in implementation of graphs is the"
"ADJACENCY MATRIX, a two-dimensional array. Each dimension of the"
"array is labeled with the nodes in the graph.  If two  nodes are"
"directly connected, the corresponding array element is set  to a"
"nonzero value. In digraphs, two symmetrical elements are set."
7
"                         DEGREE OF A NODE"
""
"The DEGREE of a node is the number of edges  associated with the"
"node.  The  number  of  nonzero  entries  in  the  column of the"
"adjacency matrix corresponding to a node gives the degree of the"
"node.  Node A in the  graph  below  has two  edges and there are"
"two 1's in the column for A in the adjacency matrix."
7
"                      PATH BETWEEN TWO NODES"
""
"A PATH exists between two nodes, I and J, if there is a  nonzero"
"entry in the adjacency matrix corresponding to  node I and  some"
"other node, between that node and another node, etc. terminating"
"in a nonzero entry between  some node and  node J.  In the graph"
"below there is a path from A to F to E."
1
4
1
2
5
6
3
1,2,1
1,6,1
5,6,1
6
```

```
7
"                  INTRODUCTION TO NETWORKS DEMO"
""
"This animated demonstration presents terms and concepts related"
"to networks. The realtionship of networks to graphs, as well as"
"the minimum spanning tree, shortest path, and  topological sort"
"algorithms are presented."
"                   Click on RUN to start..."
4
7
"                       TERMINOLOGY"
""
"A network, like a graph, consists of nodes and edges.  Unlike a"
"graph, each edge has an associated EDGE WEIGHT. In the  network"
"below, the edge weight of the  edge from  F to G is 1 (Yellow)."
"Other edges have  different  weights as  indicated by the color"
"code."
7
"                  MINIMUM SPANNING TREE"
""
"The  MINIMUM SPANNING TREE  for a  network is a  network  which"
"includes all nodes of the  original network,  connected by only"
"those edges which have the  lowest  total edge weight.  Not all"
"nodes will necessarily be directly connected. Notice during the"
"animated demo, edges which are included are highlighted."
5
"                        SHORTEST PATH"
""
"The SHORTEST PATH between two  specified  nodes in a network is"
"the path with the lowest edge weight. In the network below, the"
"shortest path from node A to node H is A-F-J-G-H."
7
"                       TOPOLOGICAL SORT"
"A directed graph or network can be  used to  represent problems"
"such as scheduling problems.  A directed edge between two nodes"
"implies that the 'source' node precedes the 'destination node.'"
"The TOPOLOGICAL SORT of a  digraph gives the order in which the"
"nodes  should be  visited to  take  into  account  the  implied"
"precedence between the nodes."
2
4
6
7
10
11
3
6,7,1
7,11,2
10,11,5
8
1
2
3
4
6
7
8
10
9
1,2,1
2,3,3
```

```
3,4,2
1,6,2
2,7,5
4,8,4
6,10,2
10,7,1
7,8,1
12
```

```
7
"               NETWORK IMPLEMENTATION DEMO"
"This animated demonstration focuses on the data structures used"
"to implement networks.  The relationship between a network  and"
"the  data  structure is  shown.  The use of the  data structure"
"in the implementation of the minimum spanning tree algorithm is"
"also shown."
"                   Click on RUN to start..."
5
7
"                      ADJACENCY MATRIX"
""
"A primary data structure used in implementation of a  network is"
"the ADJACENCY MATRIX, a two-dimensional array. Each dimension of"
"the array is labeled with the nodes in the network. If two nodes"
"are directly connected, the  corresponding  array element is set"
"to the edge weight."
7
"               MINIMUM SPANNING TREE EXAMPLE"
"As an example of how  the adjacency matrix is used to  implement"
"network algorithms, consider the  minimum spanning tree.  First,"
"one node is included, e.g. A, and the  edge weight for each node"
"connected to A is recorded. Nodes not directly connected are set"
"to some maximum weight. In the graph below the weights are:     "
"                  B: 4,  E: 2,  F: MAX"
7
""
"The node with the lowest weight from a node  already included is"
"now added.  This  is  node E (weight of 2), and the  weights are"
"updated to reflect the weight from any  already  included  node."
"The new weights would be:"
"                  B: 3, (from A to E to B)"
"                  F: 7, (from A to E to F)"
4
""
"The process is repeated.  Node B has the least weight (3),  so it"
"is included and the weight for F is recomputed as:"
"                  F: 4, (from A to E to B to F)"
3
""
"Finally, F is included with the edge B-F.  The  highlighted  edges"
"show the minimum spanning tree."
1
4
1
2
5
6
5
1,2,4
1,5,2
2,6,2
5,6,5
2,5,1
6
```

```
6
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"This self-test is intended to provide a check on your  learning"
"in the area of terms and concepts related to graphs.           "
""
"                   Click on START to begin..."
10
3
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"Information is represented in a graph by the..."
"Edges"
"Paths"
"Nodes"
"Data"
"C "
4
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"Relationships between entities are represented in a graph"
"by the..."
"Edges"
"Paths"
"Nodes"
"Data"
"A "
3
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"The degree of a node in a graph is..."
"Value of the associated data."
"Number of edges for the node."
"Number of paths for the node."
"Position of node in the graph."
"B "
3
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"Graph theory was invented by..."
"Newton"
"Leibnitz"
"Euler"
"Einstein"
"C "
3
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"A sequence of edges connecting two nodes is called a(n)..."
"Chain"
"Series"
"Degree"
"Path"
"D "
5
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"A graph containing  unidirectional edges; that is,  edges"
"which can be  travelled in only one  direction is  called"
"a(n)..."
"Digraph"
```

```
"One-way Graph"
"Acyclic Graph"
"Unidirectional Graph"
"A "
5
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"A traversal which follows a path from a start node as far"
"as possible  before  trying  other edges connected to the"
"start node is called a(n)..."
"Breadth First Traversal"
"Depth First Traversal"
"Binary Traversal"
"Linear Traversal"
"B "
5
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"A  traversal  which  follows an  edge from a start node to"
"each adjacent node before  following a  given path further"
"from the start node is called a(n)..."
"Breadth First Traversal"
"Depth First Traversal"
"Binary Traversal"
"Linear Traversal"
"A "
4
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"Using the graph below, a depth-first traversal starting at"
"node B would visit nodes in the order..."
"B C F H I L K N"
"B C H L F I K N"
"B C H K N L F I"
"B F I C H L K N"
"C "
4
"                INTRODUCTION TO GRAPHS SELF-TEST"
""
"Using the graph below, a breadth-first traversal  starting"
"at node B would visit nodes in the order..."
"B C F H I L K N"
"B C H L F I K N"
"B C H K N L F I"
"B F I C H L K N"
"A "
1
8
2
3
6
8
9
11
12
14
8
2,3,1
2,6,1
3,8,1
6,9,1
```

```
6,11,1
8,11,1
8,12,1
11,14,1
16
```

```
6
"                GRAPH IMPLEMENTATION SELF-TEST"
""
"This self-test is intended to provide a check on your  learning"
"in the area of data structures used to implement graphs."
""
"                  Click on START to begin..."
7
4
"                GRAPH IMPLEMENTATION SELF-TEST"
""
"A common data structure used to implement the edges in a graph"
"is called a(n)..."
"One Dimensional Array"
"Linked List"
"Adjacency Matrix"
"Edge List"
"C "
4
"                GRAPH IMPLEMENTATION SELF-TEST"
""
"Data values  associated with the  nodes in a  graph  are  often"
"stored in a(n)..."
"One Dimensional Array"
"Linked List"
"Adjacency Matrix"
"Two Dimensional Array"
"A "
4
"                GRAPH IMPLEMENTATION SELF-TEST"
""
"In an adjacency matrix, M, representing an undirected graph, if"
"two nodes I and J are directly connected..."
"M[I,J] = M[J,I] always"
"M[I,J] = 0"
"M[I,J] = 0 and M[J,I] <> 0"
"M[I,J] <> 0"
"A "
4
"                GRAPH IMPLEMENTATION SELF-TEST"
""
"In an adjacency matrix,  M,  representing a directed graph,  if"
"two nodes I and J are directly connected..."
"M[I,J] = M[J,I] always"
"M[I,J] = 0"
"M[I,J] <> 0 and M[J,I] <> 0"
"M[I,J] <> 0 and M[J,I] = 0"
"D "
4
"                GRAPH IMPLEMENTATION SELF-TEST"
""
"The number of nonzero  entries in the  column of the  adjacency"
"matrix which corresponds to a given node represents the..."
"Edge Weight"
"Degree"
"Path"
"Data"
"B "
4
"                GRAPH IMPLEMENTATION SELF-TEST"
""
```

```
"In an  adjacency  matrix representing an  undirected graph, the"
"number of nonzero entries will equal..."
"Number of nodes"
"Number of edges"
"Twice the number of edges"
"Half the number of edges"
"C "
6
"                GRAPH IMPLEMENTATION SELF-TEST"
""
"Given the  graph  below,  assume that the  adjacency  matrix is"
"such that row 1 represents node A, row 2 represents node B, etc"
"and  similarly for the  columns.  Then, row 1 of the  adjacency"
"matrix would be..."
"1 0 1 0 1 0"
"1 1 0 0 0 0"
"0 1 0 0 0 1"
"0 1 0 0 1 1"
"D "
1
5
1
2
3
5
6
6
1,2,1
2,3,1
1,5,1
1,6,1
2,5,1
3,6,1
16
```

```
6
"                    NETWORKS SELF-TEST"
""
"This self-test is intended to help you  check your  learning of"
"key terms and concepts related to networks."
""
"                  Click on START to begin..."
7
4
"                    NETWORKS SELF-TEST"
""
"The term which applies to the 'cost' of  traversing an  edge in"
"a network is the..."
"Degree"
"Threshhold"
"Edge Value"
"Weight"
"D "
4
"                    NETWORKS SELF-TEST"
""
"The subnetwork which  includes all  nodes with the  least total"
"edge weight needed to connect all nodes is called the..."
"Shortest Path"
"Topological Sort"
"Minimum Spanning Tree"
"Minimum Weight Network"
"C "
4
"                    NETWORKS SELF-TEST"
""
"The sequence of edges connecting two nodes with the  least edge"
"weight between the two nodes is called the..."
"Shortest Path"
"Topological Sort"
"Minimum Spanning Tree"
"Minimum Weight Network"
"A "
4
"                    NETWORKS SELF-TEST"
""
"The  algorithm  which  you  might  use to  determine the  least"
"expensive way of connecting several cities by road would be..."
"Shortest Path"
"Topological Sort"
"Minimum Spanning Tree"
"Minimum Weight Network"
"C "
4
"                    NETWORKS SELF-TEST"
""
"The  algorithm  which  you  might use to  determine the optimum"
"schedule for a set of tasks in a project would be..."
"Shortest Path"
"Topological Sort"
"Minimum Spanning Tree"
"Minimum Weight Network"
"B "
4
"                    NETWORKS SELF-TEST"
""
```

```
"Given  the  network  below, the  minimum  spanning  tree  would"
"include the edges..."
"AB, AE, AF, FJ, FG, BC"
"AE, EJ, JF, FG, CG, BC"
"AB, AC, AF, FG, FJ, EJ"
"AB, BC, AF, FJ, EJ, CG"
"D "
4
"                    NETWORKS SELF-TEST"
""
"Given the network below, the shortest path from node A to node"
"J includes the edges..."
"AB, BC, CG, FG, FJ"
"AF, FJ"
"AF, EJ"
"AE, AF, AJ"
"B "
1
7
1
2
3
5
6
7
10
8
1,2,2
2,3,3
1,5,5
1,6,2
3,7,1
6,7,4
5,10,1
6,10,1
12
```

```
6
"                 NETWORK IMPLEMENTATION SELF-TEST"
""
"This self-test is intended to provide a check on your  learning"
"in the area of data structures used to implement networks."
""
"                   Click on START to begin..."
3
4
"                 NETWORK IMPLEMENTATION SELF-TEST"
""
"The primary difference between a  network and a  graph is that"
"the edges in a network..."
"are always directed."
"never form cycles."
"include an edge weight."
"are always undirected."
"C "
3
"                 NETWORK IMPLEMENTATION SELF-TEST"
""
"Entries in the adjacency matrix for a network represent the.."
"Edge Weight"
"Data Value"
"Direction of edge"
"Number of associated nodes."
"A "
6
"                 NETWORK IMPLEMENTATION SELF-TEST"
""
"Given the network below,  assume that the  adjacency  matrix is"
"such that row 1 represents node A, row 2 represents node B, etc"
"and  similarly for the  columns.  Then, row 5 of the  adjacency"
"matrix would be..."
"0 2 0 0 1 5"
"2 0 0 0 1 2"
"1 1 0 0 0 4"
"5 2 0 0 4 0"
"C "
1
4
1
2
5
6
6
1,2,2
1,6,5
1,5,1
2,5,1
2,6,2
5,6,4
12
```