ANIMATED PRESENTATION OF SOME GREEDY ALGORITHMS

By

YI YIN

Bachelor of Science
Xiamen University
Xiamen, P. R. China
1985

Master of Science
Xiamen University
Xiamen, China
1990

Doctor of Philosophy
Oklahoma State University
Stillwater, Oklahoma
1998

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1999

OKLAHOMA STATE UNIVERSITY

ANIMATED PRESENTATION OF SOME

GREEDY ALGORITHMS

Thesis Approved:

Jacques E. LaFrance

[illegible]

J Chandler

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGMENTS

This project was really a pioneer work to me. Without the encouragement of my adviser, Dr. Jacques LaFrance, I might have given up a long time ago. Without his guidance and help, I could not imagine that I would get any results with this program. I appreciate has valuable teaching in three classes. I also appreciate the help and trust he gave me when I was his teaching assistant.

I offer gratitude to Dr. George Hedrick for serving on my committee and constantly providing encouragement. His expertise was very helpful. His critical reviews of this project and countless personal conversations were very helpful to my program. I really enjoy the time I sat in his classes and the time I was a Teaching Assistant under his advice.

I also thank Dr. John Chandler for serving on my committee and providing critical reviews of this manuscript. I appreciate his valuable teaching in the classroom.

I would also like to acknowledge the Department of Computer Science, without which this work would have been impossible.

I extend personal thanks to my wife, Jianhong Guo, and our lovely daughter, Jenny, for their patience and understanding of my absence while completing my degree.

Finally, I thank my mother, Sulan Yin, and father, Youde Zhang, for their unconditional support not only during my education, but in my entire life.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

1.1 Background

Algorithm animation is the use of computer animation to show how algorithms for computer programs work. Algorithm animation is a powerful tool for studying an algorithm's behavior. It could be used in teaching computer science courses, in designing and analyzing algorithms, in making technical drawings, and in documenting programs.

There have been continuous efforts to develop algorithm animation systems. The first general-purpose system of algorithm animation was produced in the early 1980's. The most recent work was to develop interactive animation of several types of binary search trees using Java.

Some data structures have already been presented using animation at Oklahoma State University. These include the B-tree, stack, queue, list, and binary search trees. However, there has been little work done on interactive animation of greedy algorithms.

Java is a new object-oriented programming language. Java is one part of the integrated set of systems that support World Wide Web (WWW) communication. Java integrates with Hypertext Markup Language (HTML) through applets. Applets are essential Java programs that the user's browser automatically downloads (as part of Web page observation) and executes.

1.2 Objectives

This project develops an animation system to illustrate a group of greedy algorithms in detail. The objective of the project is to develop an animated web page for teaching the greedy algorithms interactively on the Internet by using the Java language. We want to offer an integrated hypermedia learning environment for greedy algorithms. The system animates the following greedy algorithms: Kruskal's algorithm and Prim's algorithm for the minimum spanning tree problem, and Dijkstra's algorithm for the shortest path problem.

The goals of this thesis are: 1) To provide a graphical user interface (GUI) to enable the user to observe a graphical representation of the greedy algorithms. 2) To use event-driven programming techniques to let the user to interact with the animation system at run time.

This thesis contains five chapters. The second chapter describes the minimum spanning tree problem and the shortest path problem and reviews the history of algorithm animation and the Java language. The design and implementation are analyzed in chapter III. The environment and platform for this project are described in chapter IV. Finally, conclusions and suggestions for future work are included in chapter V.

II. LITERATURE REVIEW

2.1 Algorithm animation

Algorithm animation is a powerful tool for studying an algorithm's behavior. It could be used in teaching computer science courses, in designing and analyzing algorithms, in making technical drawings, and in documenting programs.

2.1.1 Definition

An algorithm is a computational procedure that takes some values as input and produces some values as output. Thus an algorithm is regarded as a sequence of well-defined computational steps that transform the input into the output [1]. An algorithm is guaranteed to halt eventually.

Algorithm animation is the use of computer animation to show how algorithms for computer programs work. Algorithm animation is one particular instance of software visualization, which is the use of images, graphics and animation to illustrate computer algorithms, programs and processes [2]. Algorithm animation is concerned with demonstrating the behavior of a program or algorithm by visualizing the fundamental operations of the program as it runs. Such displays are useful both for education and for research in the design and analysis of algorithms.

2.1.2 History of algorithm animation

There have been continuous efforts to develop algorithm animation systems. The first general-purpose systems of algorithm animation in the early 1980's used monochrome displays. One of the early systems is BALSA, developed by Brown and others in the middle 1980's [6]. BALSA was constrained by a lack of computational power for real-time two-dimensional graphics [2]. As computational power has increased, so has the sophistication of the graphical techniques used for animating algorithms.

BALSA-II is an improvement of BALSA and is a much more popular system [7]. It has been used to supplement the teaching of computer programming. BALSA-II is a standalone system based on a kernel running on Macintoshes. Views for new algorithms are constructed in Pascal using an animation library. The algorithm is also programmed in Pascal. Synchronization between the algorithm and the view is achieved by adding interesting events that trigger an action in the animation.

Brown has also completed a successor to BALSA-II called Zeus [8] in Digital's Systems Research Center in Palo Alto, where he has investigated the use of color and sound for algorithm animation [9]. Color was designed as an integral part of Zeus. The Zeus system uses non-speech sound to convey the workings of algorithms. Zeus can run on multiprocessor machines and has been used to animate a variety of algorithms for teaching and research purposes.

In the late 1980's, Stasko [10] implemented a system called TANGO (short for "Transition-based ANimation GeneratiOn") that can be used to create algorithm animations. In newer versions of his system, the emphasis is on developing animations with three-dimensional views of data. Direct manipulation programming techniques were also integrated into the animation development [12]. TANGO provided an elegant framework for specifying 2D animations [2]. At about the same time, Duisberg pursued decoupling visualization from the program code so that visualizations can be specified and modified more easily [11].

Another animation system is Anim developed by Bentley and Kernighan [13]. Anim is used at Bell Labs, running under UNIX and X windows, to produce simple program animations. Contrary to BALSA-II and TANGO, Anim is not interactive, but more like a "Camcorder and VCR" [13]. Anim's advantages are due to the wide availability of the underlying hardware and software and in its reliance on general purpose UNIX tools like AWK. However, Anim also has disadvantages in that it is not interactive, its user interface is relatively crude, and it demands an experienced UNIX programmer to do more than just run the movies.

In the early 1990's, Brown and Najork [2] at Digital's Systems Research Center reported use of three-dimensional interactive graphics for algorithm animation. 3D interactive graphics provides another level of expressiveness to the animations.

Most recently, Ierardi and Li at the University of Southern California (URL: http://langevin.use.edu/BST/) have worked on a new algorithm animation system. Their project "Animating Algorithm, I" (1997) developed interactive animation of several types of binary search trees using Java. The algorithms implemented so far include: 1) Standard binary search trees; 2) Splay trees use bottom-up splaying or top-down splaying; 3) Red-black trees; 4) Randomized binary search trees.

There has been research on computer animation in the Computer Science Department at Oklahoma State University since 1970's. In recent years, there have been several studies on data structure animation. Arra [20] animated array, stack, queue, and linked list structures using the C++ language. Kummetha [21] developed a level-linked R* tree using X windows. Shen [22] studied AVL tree, B-tree, red-black tree, and splay tree algorithms using the C language. Xu [23] visualized list, stack, and queue, using Lingo scripting language. Harvick [24] presented animation for stack, linked list, queue, binary tree, and AVL tree using DSA programming language. Lin [25] worked on B trees using C++.

2.1.3 Benefits of algorithm animation

Animation is usually taken as a useful aid to learning, especially when the material to be learned is difficult. Algorithm animation should help beginners in programming to understand the algorithms. There are two main criteria to judge whether or not animation assists learning: mastery of knowledge and time of learning [15].

1) Mastery of Knowledge

According to Byrne [15], learners who make and test predictions of what is going to happen at each step of an algorithm learn more efficiently than learners who passively take in the information. Here it seems logical that an animation might be of help as Byrne states [15]: "A learner could certainly make predictions from a static textual/graphic presentation of an algorithm, but perhaps the advantage of an animation is that it will spontaneously encourage a learner to make the predictions without being prompted and will provide the learner with rapid feedback about the accuracy of his or her predictions."

Understanding is a process of building relations among new concepts and among new concepts and old concepts, which is affected by background knowledge. Learners with related background can relate the material to their knowledge, and make analogies or abstractions so as to build up relations to the background knowledge within a shorter time [16].

Animations can help the learning of knowledge for new learners. However, it seems that algorithm animation is not so useful in assisting understanding for advanced learners [15, 16]. One possible reason is that the advanced learners have been well prepared for logical, procedural and abstract reasoning abilities that are needed in understanding complex data structures and algorithms.

2) Learning Speed

On the other hand, animation should speed up the learning process of both beginners and advanced learners.

Algorithms are very suitable for animation compared to other subjects, because the usual underlying data structures can be easily represented in graphical form [15]. For such data structures, operations can be illustrated by manipulations of the graphical representations. When reading the pseudo-codes or text descriptions for an algorithm, people usually construct a mental image of how a data structure changes. Sometimes people prefer to make the mental image more concrete and thus more accessible in the later stages of learning by drawing a picture. Therefore, text with figures is preferred for learning algorithms. However, if exposed to figures only, people still make efforts to imagine the transition between figures, interpolating the motion in-between. By using a computerized algorithm animation, the mental animation is enhanced by an concrete one [15]. Usually, the computerized animation is faster than the mental one, and more importantly, the computerized one is absolutely accurate. So we can expect that animation will speed learning.

The learner will possibly still perform a mental animation when a computerized animation is shown. If this is the case, the learner is predicting the result of the animation [15]. The prediction can be wrong. So the algorithm animation also plays a role of automated tutor, validating or correcting the prediction with immediate feedback.

2.2 Greedy algorithm

2.2.1 Introduction

Many computing algorithms can be regarded as a finite series of guesses. In the travelling salesman problem, for example, we try (guess) each possible tour in turn, then determine its cost. When we have tried them all, we know which one is the optimum (least cost) one. However, we must try them all before we can be certain that we know which is the optimum one, leading to an $O(n!)$ algorithm [3].

Intuitive strategies, such as building up the salesman's tour by adding the city that is closest to the current city, can readily be shown to produce sub-optimal tours. This is an example of a greedy strategy.

2.2.2 Definition

A greedy strategy is a heuristic for optimization. At each step of an algorithm, one of several possible choices must be made. The greedy strategy makes the choice that is the best at the moment [1]. Each step increases the size of the partial solution and is based on local optimization: the choice selected is that which produces the largest immediate gain while maintaining feasibility. Greedy algorithms are very efficient but generally are not guaranteed to find globally optimal solutions to problems. However, a greedy strategy is the best choice for certain problems.

2.2.3 Properties of greedy strategy

How can we tell if a greedy algorithm will solve a particular optimization problem? There is no way in general, but there are two ingredients that are exhibited by most problems that lead themselves to a greedy strategy: the greedy-choice property and optimal substructure [1].

Greedy-choice property: a globally optimal solution can be arrived at by making a locally optimal (greedy) choice. In a greedy algorithm, we make whatever choice seems best at the moment and then solve the subproblems arising after the choice is made. The choice made by a greedy algorithm may depend on choices so far, but it cannot depend on any future choices or on the solutions to subproblems.

Optimal substructure: an optimal solution to the problem contains within it optimal solutions to subproblems. This property is a key ingredient of assessing the applicability of dynamic programming as well as a greedy algorithm.

There is a class of algorithms, the greedy algorithms, for which we can find a solution by using only knowledge available at the time the next choice (or guess) must be made. The greedy method is quite powerful and works well for a range of problems, including the minimum spanning tree problem, the shortest path problem from a single source, and Chvatal's greedy set-covering heuristic [1].

In this project, I will look at two common groups of greedy algorithms: the minimum spanning tree algorithms and shortest path algorithms.

2.2.4 Minimum Spanning Trees

A spanning tree of a graph, *G*, is a set of $|V|-1$ edges that connect all vertices of the graph. A spanning tree of a graph is just a subgraph that contains all the vertices of the graph and is a tree. A graph may have many spanning trees. Suppose the edges of the graph have weights or lengths. The weight of a tree is just the sum of the weights of its edges. The minimum spanning tree is the tree with the minimum total weight. There can be more than one minimum spanning tree for each graph. Obviously, different trees within a graph may have different weights [1].

In general, it is possible to construct multiple spanning trees for a graph, G. If a cost, c_{ij}, is associated with each edge, $e_{ij} = (v_i,v_j)$, then the minimum spanning tree is the set of edges, E_{span}, forming a spanning tree, such that:

$$C = \text{sum}(\ c_{ij} \mid \text{all } e_{ij} \text{ in } E_{span}\ )$$

is a minimum. A minimum spanning tree connects all the vertices and minimizes the sum of the lengths of its edges for a connected, undirected graph with a positive length associated with each edge [5].

There are two common algorithms for solving the minimum spanning tree problem: Kruskal's algorithm and Prim's algorithm.

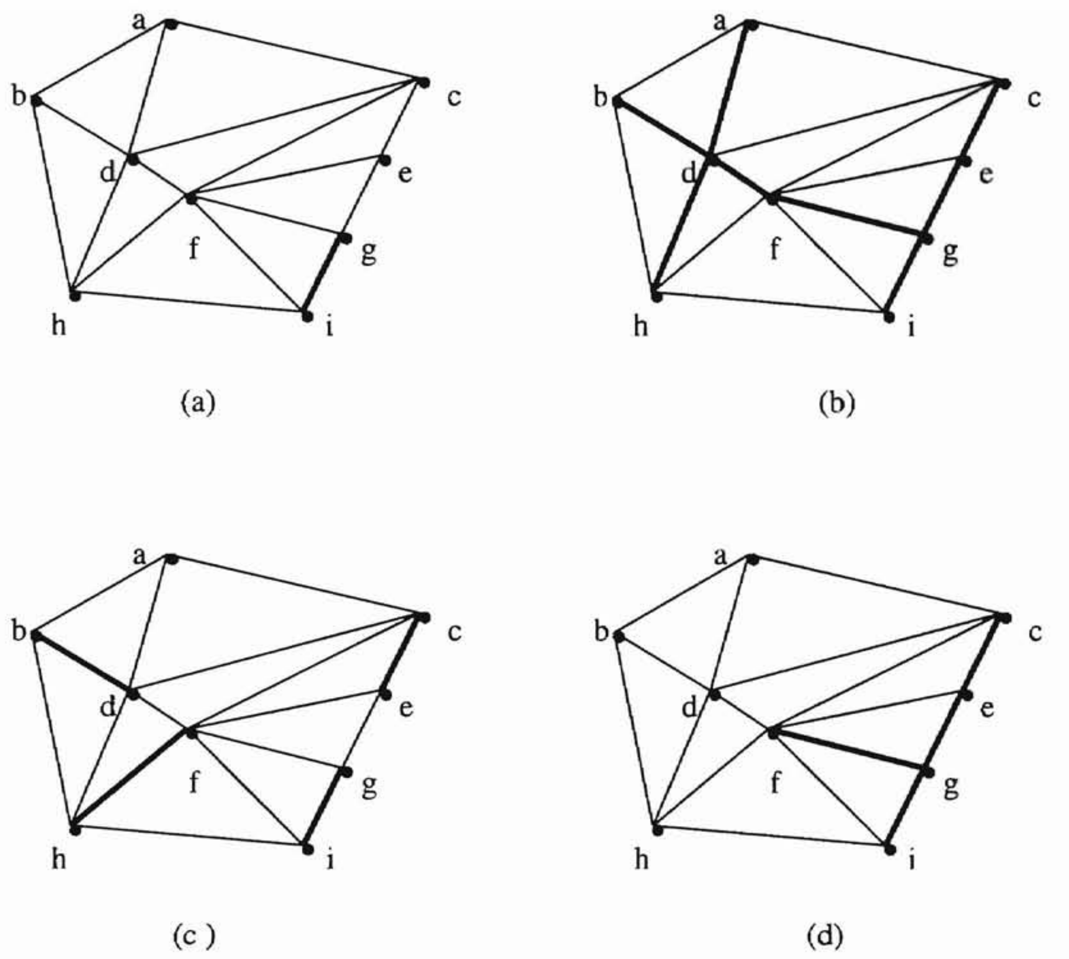


Figure 1. The difference between Kruskal's and Prim's Algorithms

Figure 1 intends to explain the minimum spanning tree (MST) for a graph, without marking the weight of each edge. The thick lines indicate that these edges are part of the MST. Figure 1-(a) shows a graph with the direct connections among vertices. Edge e(i, g), represented by the only thick line, is the start edge. Figure 1-(b) shows the MST for the graph of Figure 1-(a). Figure 1-(c) and figure 1-(d) show an intermediate step for Kruskal's algorithm and Prim's algorithm, respectively. We will explain these two algorithms in the following sections.

2.2.4.1 Kruskal's Algorithm

Kruskal's algorithm was first presented by J.B. Kruskal in 1956 [5]. Kruskal's algorithm is the easiest to understand and probably the best one for solving the problem by hand. Kruskal's algorithm constructs a minimum spanning tree incrementally. For graphs that are not dense (the number of edges $<< n^2$), Kruskal's algorithm is more efficient [1].

This algorithm creates a forest of trees at intermediate steps (Figure 1-(c)). Initially the forest consists of n single node trees (and no edges). At each step, we add one (the cheapest) edge so that it joins two trees together. If it were to form a cycle, it would simply link two nodes that were already part of a single connected tree, so that this edge would not be needed.

2.2.4.2 Prim's Algorithm:

Prim's algorithm was first presented by R.C. Prim in 1957 [5]. Rather than build a subgraph one edge at a time as in Kruskal's algorithm, Prim's algorithm builds a single tree (Figure 1-(d)), not a forest, one vertex at a time. Prim's algorithm adds the cheapest edge that extends the tree to include a new vertex.

Prim's algorithm operates much like Dijkstra's algorithm for finding shortest paths in a graph (see next section). Prim's algorithm has the property that the selected edges in the set *A* always form a single tree. The key to implementing Prim's algorithm efficiently is to make it easy to select a new edge to be added to the tree formed by the edges in set *A* [1].

If a priority queue is implemented as a binary heap, the total time for both Kruskal's algorithm and Prim's algorithm is $O(|E| \lg|V|)$. If we use a Fibonacci heap to implement the priority queue, the running time of Prim's algorithm improves to $O(|E| + |V| \lg|V|)$, which is better if $|V|$ is much smaller than $|E|$ [1].

2.2.5 Shortest Path Algorithm

Single-source shortest-path algorithms are a family of algorithms that, given a directed graph with weighted edges, find the shortest path from a designated vertex, called the source, to all other vertices. The length of a path is defined to be the sum of the weights of the edges along the path [5].

All single-source shortest-path algorithms have certain common features: first, they assign a cost to each vertex, indicating the length of the shortest path found so far from the source to this vertex. Initially, the cost will be infinite. Then they repeatedly choose an edge e from vertex u to vertex v, test if it can lower the cost associated with v, that is, if COST(v) > COST(u) + WEIGHT(e), and if so, indeed lower v's cost. Algorithms differ in their choice of which edge to examine next [2].

Dijkstra's algorithm theoretically is the most efficient known algorithm without constraints for the shortest path problem [1]. Dijkstra's algorithm solves the single-source shortest path problem on a weighted, directed graph $G = (V, E)$ for the case in which all edge weights are nonnegative: $w(u, v) \geq 0$ for each edge $(u, v) \in E$.

Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined [1]. That is, for all vertices $v \in S$, we have d[v] = δ(s, v). The algorithm repeatedly select the vertex u ∈ V-S with the minimum shortest-path estimate, insert u into s, and relaxes all edges leaving V.

Because Dijkstra's algorithm always chooses the "lightest" or "closest" vertex in unvisited vertices to insert into the set of visited vertices, we say that it uses a greedy strategy.

To illustrate how the labeling algorithm works, look at the weighted, undirected graph of Figure 2-(a). The weights in the graph can represent any cost function, e.g. distance. For example, we want to find the shortest path from A to D. We start out by marking

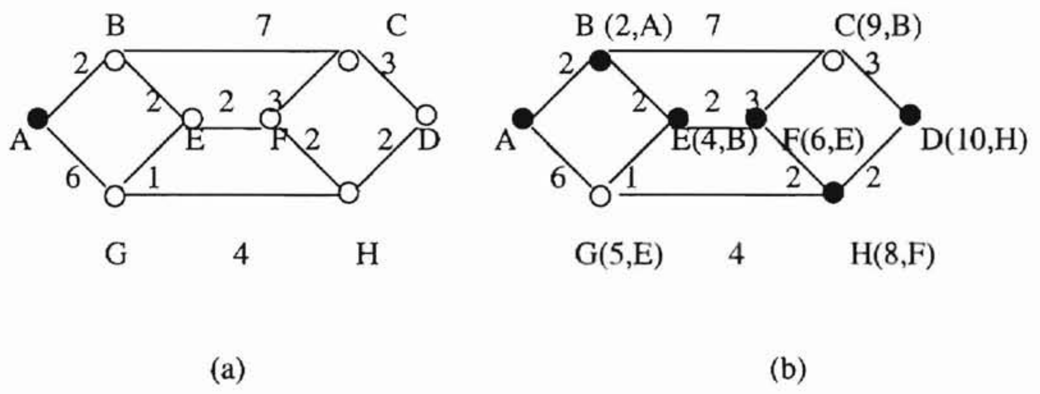


Figure 2. The start and final steps in computing the shortest path from A to D.

each of the nodes adjacent to A (the working node), relabeling each one with the distance from A [27]. Whenever a node is relabeled, we also label it with the node from which the probe was made, so we can reconstruct the final path later. After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively labeled node with the smallest value. Figure 2-(b) shows the result of the algorithm, with the chosen nodes filled.

Dijkstra's algorithm was first published by E.W. Dijkstra in 1959 [4]. It originally used a linear array but contained no mention of a priority queue. Later version of Dijkstra's algorithm implemented a priority queue as either a binary heap or a Fibonacci heap.

If the priority queue is implemented as a linear array, the running time is $O(|V|^2 + |E|) = O(|V|^2)$. If the priority queue is implemented as binary heap, the running time of Dijkstra's algorithm is $O((|V| + |E|) \lg|V|) = O(|E| \lg|V|)$, assuming all vertices are reachable from the source. If the priority queue is implemented as Fibonacci heap, then the running time of Dijkstra's algorithm is $O(|V| \lg|V| + |E|)$ [1].

Shortest-path problems are the most fundamental commonly encountered problems in the study of communication within a network. A shortest-path algorithm is widely used in many forms because it is relatively simple and easy to implement. Since repeated determination of shortest paths in large networks often forms the core (and inner loop) of many transportation planning and utilization packages [18], the search for faster and faster shortest path procedures never ends.

2.3 Java for animation

Java is a new object-oriented programming language developed by Sun Microsystems. Java makes it possible for program developers to create content that can be delivered to and run by users on their own computers. With the delivery platform as the Web page, Java software can support a variety of information tasks with true interactivity. Sun's Java web site (http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html) currently says that "Java is a simple, robust, object-oriented, platform-independent multithreaded, dynamic general-purpose programming environment. It is best for creating applets and applications for the Internet, intranets and any other complex distributed networks." Java has many capabilities that can be used to create animations, multimedia presentations, and further interactivity.

Java is one part of the integrated set of systems that support World Wide Web (WWW) communication. Java integrates with Hypertext Markup Language (HTML) through applets. Applets are Java programs that the user's browser automatically downloads (as part of Web page observation) and executes. Java can run truly complex Internet applications on server computers [19]. Because Java applications run on a server computer, they can be fast and highly interactive. Because the resulting applications are small, they come across the Internet as fast as most graphic files. With graphical input and output possible through the applet displayed on the page, Java extends the Web into higher levels of interactivity. It is entirely possible that in the long run, the model of people buying programs, installing them, and running them locally will be replaced by a model in which people click on Web pages, get applets downloaded to do work for them, possibly in conjunction with a remote server or database [27].

Java is a high-level programming language, similar to C, C++, and Modula-3, but it is much simpler. All the unnecessary features of high-level programming languages are left out. Java has implementations linking tightly with platform-native communication libraries (in C or perhaps C++). These libraries do most of the difficult, speed-and-memory-intensive work of communication [17]. The Java APIs provide a convenient, consistent wrapper for these various platforms so that if a networking application is written in Java, it instantly works on any platform that supports the required Java APIs, or runtime modules. The runtime module translates the bytecodes into machine instructions for a particular computer.

There are some additional solid benefits to writing communicating applications using Java [17]. Because Java is a nice dynamic, interpreted language, rapid prototyping of applications is potentially easier than with C or C++.

In addition, Java provides some specific language features such as exceptions, which can make communications programming easier.

Finally, the broader Java APIs provide a fairly consistent set of libraries for user interface, file system, and so forth that work across platforms, and it's fairly easy to wrap communications code quickly within a GUI-based testbed. Java provides a solid platform for developing communicating applications [17].

Sun's Java web site (http://java.sun.com/docs/books/tutorial/getStarted/intro/changemylife.html) states that a program written in Java is clearer and requires less effort than many other languages. Java can help a programmer do the following:

- Get started quickly: Java is not only a powerful object-oriented language, but also easy to learn, especially for programmers already familiar with C or C++.
- Write less code and develop programs faster: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in Java can be smaller than the same program in C++. So program development time may be faster than writing the same program in C++.
- Write better code: The Java language encourages good coding practices, and its garbage collection helps avoid memory leaks. Java's object orientation, and its wide-

ranging, easily extendible API encourage reusing other people's tested code and thereby introduces fewer bugs.

- Avoid platform dependencies with 100% Pure Java: Because pure Java programs are compiled into machine-independent bytecodes, they run consistently on any Java platform.
- Distribute software more easily: Applets can be easily upgraded from a central server. Applets take advantage of the Java feature of allowing new classes to be loaded "on the fly," without recompiling the entire program.

The disadvantage of the Java language is that it is slow when compared with C or C++, because the Java compiler cannot compile source code to machine code. The users must use the Java interpreter to interpret the bytecode every time when they want to run the programs. In addition, the Java language is not very stable because it is so new. New features are added to it so quickly that a program might be "too old" to be accepted by the current compiler if it were written one year ago.

III. DESIGN AND IMPLEMENTATION ISSUES

This project develops an animation system to illustrate a group of greedy algorithms in detail. The objective of the project is to develop an animated web page for teaching the greedy algorithms interactively on the Internet by using the Java language. The system animates the following greedy algorithms: Kruskal's algorithm and Prim's algorithm for the minimum spanning tree problem, and Dijkstra's algorithm for the shortest path problem.

The animation system is made available on the Internet. The temporary WWW address is: http://www.su.okstate.edu/students/yiyin/greedy.html. There are brief introductions about the system. It has an easy to use interface that illustrates an algorithm through its results. A user can choose one of the three algorithms to run. Target users are students learning greedy algorithms for the first time. A user can run the animation system remotely using common Java-compatible Web browsers such as Netscape, Internet Explorer, or HotJava.

The system is interactive. It will accept parameters from the user at runtime and pass them to the applets. With the interactive visualization, the user can change some graphical parameters interactively at running time, and immediately see the resulting structure produced by the algorithm under consideration.

3.1 Significant programming techniques

The animation system is written mostly in Java. A Java source file is created using a plain text editor, vi under UNIX and NotePad under MS-Windows. Source codes are saved and compiled with the Java development kit (JDK) by running the JAVAC compiler program from the same directory. The compiler for JDK version 1.1.6 is used to compile the Java programs in this project. The results of this compilation are Java class files, which produce three applets.

The next step is to embed the applets into an HTML document for Web pages, which is saved as greedy.html. All the files reside in the same directory. Because the applets are embedded, it takes on all of the typical attributes of Java applets [19]. They exist within a window, between <APPLET> and </APPLET>, as defined by the HTML code and will be loaded/unloaded appropriately as the Web pages are loaded and unloaded from the Web browser.

Event-driven programming is used to implement the project. A Java GUI is composed of many GUI components. The GUI components generate events that indicate that specific actions have occurred. Event-driven programming is a technique for constructing software that operates in response to events as they occur [28]. Synchronization between the algorithm and the GUI is achieved by adding interesting events that trigger an action in the algorithm animation. The following Java events are used in the project: ActionEvent, KeyEvent, ItemEvent, AdjustmentEvent, MouseEvent, and MouseMotionEvent.

Multiple threads are used for the project. A thread is a sequential flow of execution through a program that occurs at the same time another sequential flow of execution is processing the same program [28]. Threads are created by implementing the Runnable interface and then implementing the run method. A background thread is calculating and drawing the working node while the current thread displays the screen. The program itself can split into multiple threads of execution that seem to run concurrently.

3.2 Main Elements of the Project

The entry point of the animation system is a Web page that is titled "Animation of Greedy Algorithms". The animation system is made available on the Internet. The temporary WWW address is: http://www.su.okstate.edu/students/yiyin/greedy.html. It might eventually be integrated into Dr. LaFrance's home page (URL address: http://a.cs.okstate.edu/~jacques/).

The home page has a brief introduction to greedy algorithms. There are three choices of greedy algorithms: Dijkstra's algorithm, Kruskal's algorithm, and Prim's algorithm. Clicking on each hyperlink will lead to one web page corresponding to the respective algorithm. Each algorithm web page has general information about that algorithm and how to use the animation system. The most important element of the algorithm web page is a hyperlink connecting to a Java applet that actually implements the animation of the algorithm.

There are three Java packages: Path, Kruskal, and Prim, each containing Java classes for animation of the corresponding algorithm. Although the animations of the three algorithms have similar control panels of the GUIs, the designs for the three algorithms are different in order to research and compare different approaches to solve similar problems. Also, the input data are provided differently for the three algorithms. We explain the classes in each package in next section. We will explain the other aspects of the packages later in this chapter.

3.2.1 Dijkstra's Algorithm - Path Package

Dijkstra's algorithm constructs a set S of vertices whose final shortest-path weights from the source s have already been determined. That is, for all vertices $v \in S$, we have $d[v] = \delta(s, v)$. The algorithm repeatedly selects the vertex $u \in V-S$ with the minimum shortest-path estimate, insert u into s, and relaxes all edges leaving V.

Following is the pseudo-code of Dijkstra's algorithm [1].

```
DIJKSTRA(G, W, S)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 S ← 0
3 Q ← V[G]
4 while Q ≠ 0
5       do u ← EXTRACT-MIN(Q)
6             S ← S ∪ {u}
7             for each vertex v ∈ Adj[u]
8                   do RELAX(u, v, w)
```

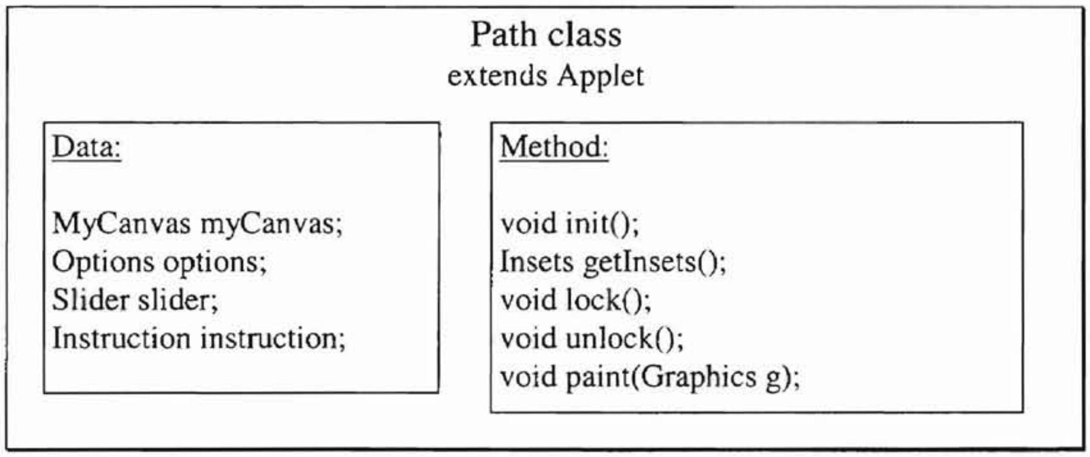
Dijkstra's algorithm chooses edges as shown above. Line 1 performs the initialization of d (a shortest-path estimate) and π (node's predecessor field) values, and line 2 initializes the set S to the empty set. Line 3 then initializes the priority queue Q to contain all the vertices in V-S = V - 0 = V. Each time through the while loop of lines 4 - 8, a vertex u is extracted from Q = V-S and inserted into set S. (the first time through this loop, u = s) Vertex u, therefore, has the smallest shortest-path estimate of any vertex in

V-S. Then, lines 7 - 8 relax (e.g. test whether we can improve the shortest path to v found so far by going through u and, if so, updating d[v] and π[v]) each edge (u, v) leaving u, thus updating the estimate d[v] and the predecessor π[v] if the shortest path to v can be improved by going through u. Vertices are never inserted into Q after line 3 and that each vertex is extracted from Q and inserted into S exactly once, so that the while loop of lines 4 - 8 iterates exactly |V| times.

Following are the descriptions of classes in the Path package.

1. Path Class

This is the driving class (Figure 3) of the package. The animation of shortest path problem is started here. This class initializes all of the variables and creates all of the panels. It builds an applet by extending the Applet class. This applet defines the graphical user interface (GUI) with which the user actually interacts.



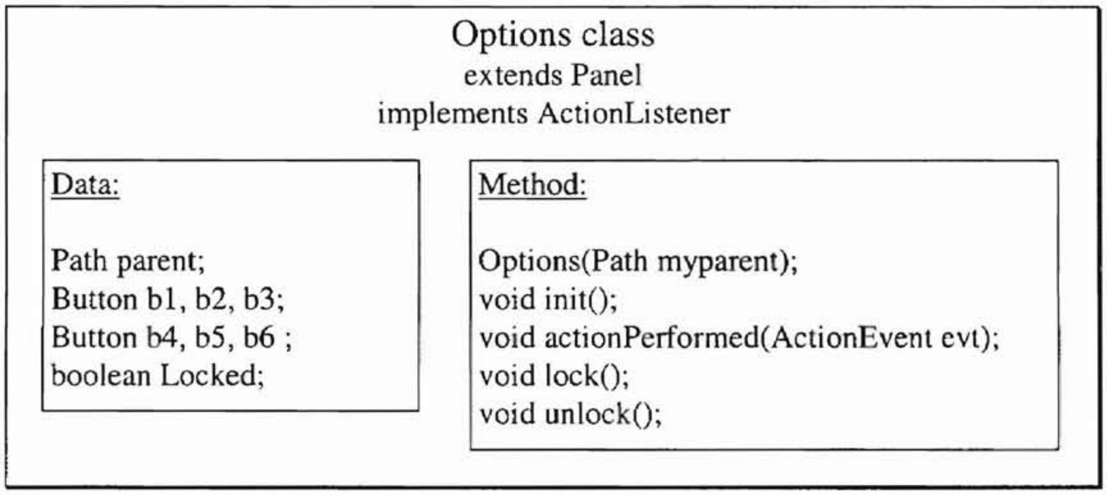
```
MyCanvas myCanvas;
Options options;
Slider slider;
Instruction instruction;
```

```
void init();
Insets getInsets();
void lock();
void unlock();
void paint(Graphics g);
```

Figure 3 Descriptions of the Path class.

2. Options Class

This is the control panel (Figure 4) on the right of the GUI. There are several buttons for showing, running, and clearing the visualization of the algorithm. All the buttons are event-listeners. When the user's mouse clicks on one button, the button listens to the click and then takes corresponding actions.



```
Options class
extends Panel
implements ActionListener

Data:

Path parent;
Button b1, b2, b3;
Button b4, b5, b6 ;
boolean Locked;

Method:

Options(Path myparent);
void init();
void actionPerformed(ActionEvent evt);
void lock();
void unlock();
```

Figure 4 Descriptions of the Options class in the Path package.

3. Slider Class

This is the control panel (Figure 5) on the left of the GUI. There is a scroll bar for adjusting the running speed of the animation. This class implements AdjustmentListener, which will set the running speed of the algorithm according to the adjusted value of the scroll bar.

Slider class
extends Panel
implements ActionListener, AdjustmentListener

Data:	Method:
static final int PAUSE, VISIBLE, MINB, MAXB;	Slider(Path myparent);
Path parent;	void init();
Button b1, b2, b3;	void actionPerformed(ActionEvent evt);
Scrollbar scroll;	adjustmentValueChanged (AdjustmentEvent evt);
boolean Locked;	void lock();
	void unlock();

Figure 5 Descriptions of the Slider class.

4. Instruction Class

This is the display panel (Figure 6) on the top of the GUI. It consists of two elements: a choice box and a text area. When an item in the choice box is clicked, the corresponding explanation is shown in the text area.

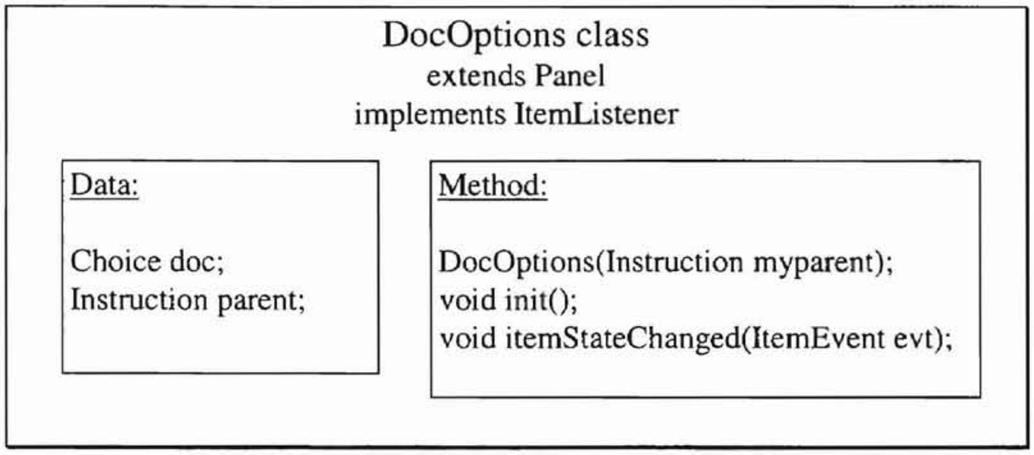
Instruction class
extends Panel

Data:	Method:
DocOptions docopt;	void init();
DocText doctext;	

Figure 6 Descriptions of the Instruction class in the Path package.

5. DocOptions Class

This panel (Figure 7) implements a choice box from which key words can be chosen to show instructions in the text area of its parent, an object of Instruction Class.



DocOptions class
extends Panel
implements ItemListener

Data:

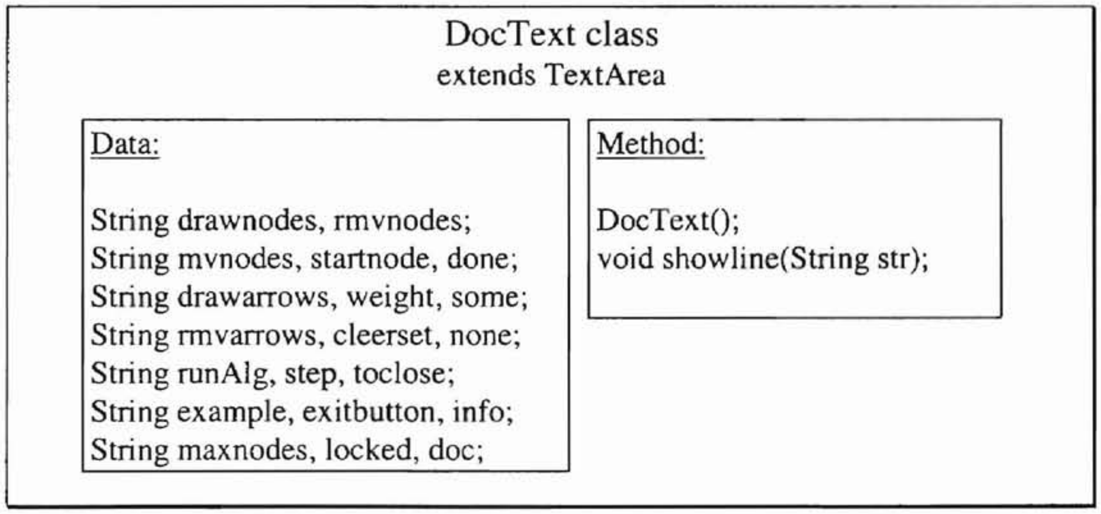
Choice doc;
Instruction parent;

Method:

DocOptions(Instruction myparent);
void init();
void itemStateChanged(ItemEvent evt);

Figure 7 Descriptions of the DocOptions class in the Path package.

6. DocText Class



DocText class
extends TextArea

Data:

String drawnodes, rmvnodes;
String mvnodes, startnode, done;
String drawarrows, weight, some;
String rmvarrows, cleerset, none;
String runAlg, step, toclose;
String example, exitbutton, info;
String maxnodes, locked, doc;

Method:

DocText();
void showline(String str);

Figure 8 Descriptions of the DocText class in the Path package.

This is the text area (Figure 8) in the Instruction Class. The text area is used to display instructions when a key word is chosen or the algorithm is running.

7. MyCanvas Class

MyCanvas class
extends Canvas
implements Runnable, KeyListener, MouseMotionListener, MouseListener

Data:

final int MAXNODES, MAX;
final int NODESIZE, DIJKSTRA ;
final int NODERADIX;
Point node[], arrow[][];
Point startp[][], endp[][];
Point thispoint, oldpoint;
float dir_x[][], dir_y[][];
boolean algedge[][], changed[];
Color colornode[];
int weight[][], dist[], finaldist[];
int numchanged, neighbours, step;
int mindist, minnode, minstart, minend;
int numnodes, emptyspots, hitnode;
int node1, node2, startgraph;
Image offScreenImage;
Graphics offScreenGraphics;
Dimension offScreenSize;
Thread algrthm;
Path parent;

Method:

MyCanvas(Path myparent);
void lock(); void unlock();
void start(); void init();
void clear(); void reset();
void runAlg(); void stepAlg();
void stop(); void initAlg();
void run(); void nextStep();
void showExample();
void keyPressed(KeyEvent evt);
void keyReleased(KeyEvent evt);
void mousePressed(MouseEvent evt);
void mouseDragged(MouseEvent evt);
void mouseReleased(MouseEvent evt);
boolean nodeHit(int x, int y, int dist);
boolean arrowHit(int x, int y, int dist);
void nodeDelete();
void changeWeight(int x, int y);
void arrowUpdate(int p1, int p2, int w);
synchronized void update(Graphics g);
void drawArrow(Graphics g, int i, int j);

Figure 9 Descriptions of the MyCanvas class.

This is the heart of the Path package. It actually implements Dijkstra's algorithm (Figure 9). The Runnable interface is implemented so that a thread will be created to execute the algorithm. This canvas will listen to key events and mouse events in order to add, delete, and move nodes. It can also take an action of change start node and weight of an edge, according to which event occurs.

3.2.2 Kruskal's Algorithm – Kruskal Package

The algorithm manages a set A, a forest, that is always a subset of some minimum spanning tree. At each step, an edge (u, v) is determined that can be added to A without violating this invariant. The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

Following is the pseudo-code of Kruskal's algorithm [1]:

```
KRUSKAL(G, w)
1 A <- 0
2 for each vertex v ∈ V[G]
3     do MAKE-SET(v)
4 sort the edge of E by nondecreasing weight w
5 for each edge (u, v) ∈ E, in order by nondecreasing weight
6     do if FIND-SET(u) ≠ FIND-SET(v)
7            then A ← A ∪ {(u, v)}
8                 union (u, v)
9 return A
```

Lines 1 - 3 initialize the set A to the empty set and create |v| tree, one containing each vertex. The edges in E are sorted into order by nondecreasing weight in line 4. The for loop in lines 5 - 8 checks, for each edge (u, v), whether the endpoint u and v belong to the same tree. If they do, then the edge (u, v) cannot be added to the forest without creating a cycle, and the edge is discarded. Otherwise, the two vertices belong to different trees, and the edge (u, v) is added to A in line 7, and the vertices in the two tree are finally merged in line 8.

Following are the descriptions of classes in the Kruskal package.

1. MST Class

MST class
extends Applet
implements Runnable, ActionListener, ItemListener, AdjustmentListener

Data:

static int MAX_WEIGHT;
static int OP_COUNT;
int curr_size;
int next_size;
Panel disp, top, bot;
Button reset, step, run, clear;
TextField weight_field, size_field,
  edge_field, cost_field, mst_field;
Scrollbar slider;
Graph G;
CheckboxGroup group;
Checkbox show, hide;
Thread thread;
Instruction instruction ;
static final int MinVert, MaxVert ;

Method:

void init(); void runAlg();
void start(); void run();
void clear(); void stop();
Insets getInsets();
void update(Graphics g);
void paint(Graphics g);
void newProblem(int n);
void adjustmentValueChanged(AdjustmentEvent e);
void actionPerformed(ActionEvent e);
void itemStateChanged(ItemEvent e);
static void CountOp();

Figure 10 Descriptions of the MST class.

This is the main class (Figure 10) for the animation of Kruskal's algorithm. This class initializes all of the variables and creates all of the panels for controlling parameters and progressing Kruskal's algorithm.

2. Instruction Class

This is the display panel (Figure 11) on the top of the GUI. It consists of two elements: a choice box and a text area. When an item in the choice box is clicked, the corresponding explanation is shown in the text area.

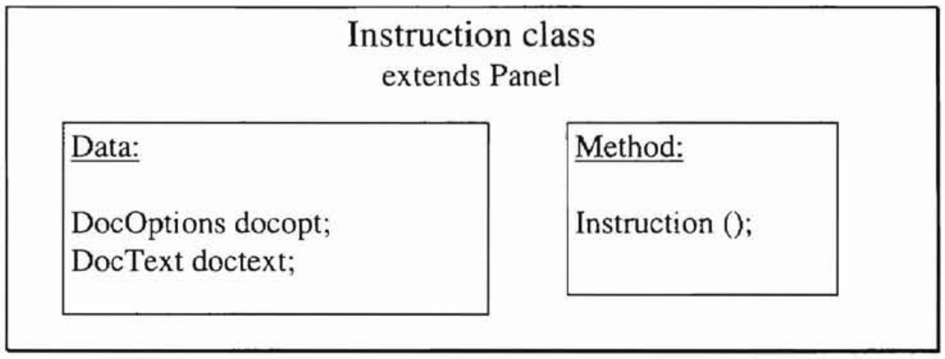
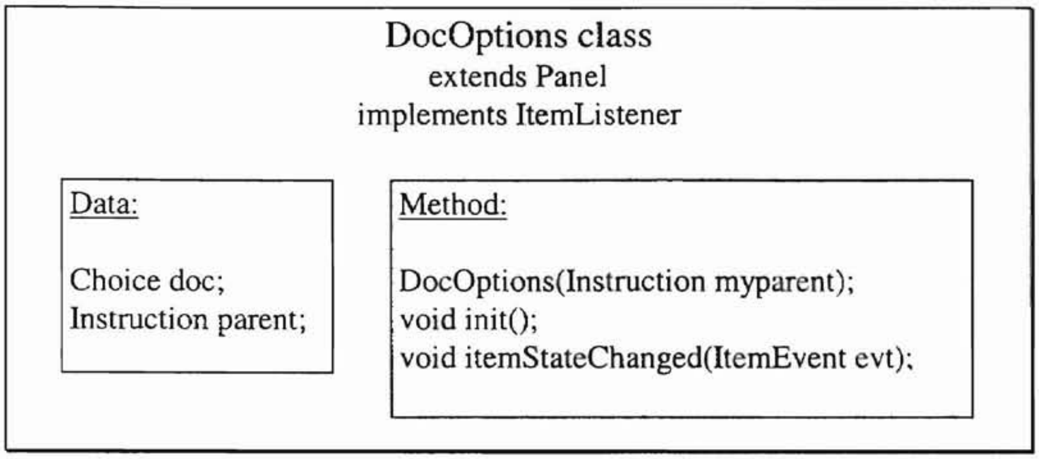


Figure 11 Descriptions of the Instruction class in the Kruskal package.

3. DocOptions Class

This panel implements a choice box (Figure 12) from which key words can be chosen to show instructions in the text area of its parent, an object of Instruction Class.



DocOptions class
extends Panel
implements ItemListener

Data:

```
Choice doc;
Instruction parent;
```

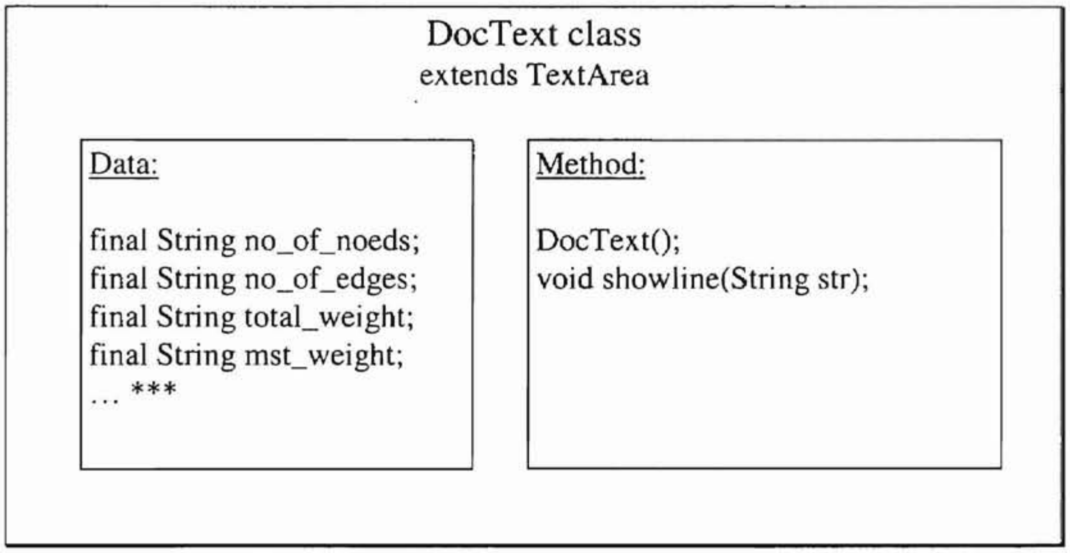
Method:

```
DocOptions(Instruction myparent);
void init();
void itemStateChanged(ItemEvent evt);
```

Figure 12 Descriptions of the DocOptions class in the Kruskal package.

4. DocText Class

This is the text area (Figure 13) in the Instruction Class. The text area is used to display instructions when a key word is chosen or the algorithm is running.



DocText class
extends TextArea

Data:

```
final String no_of_noeds;
final String no_of_edges;
final String total_weight;
final String mst_weight;
... ***
```

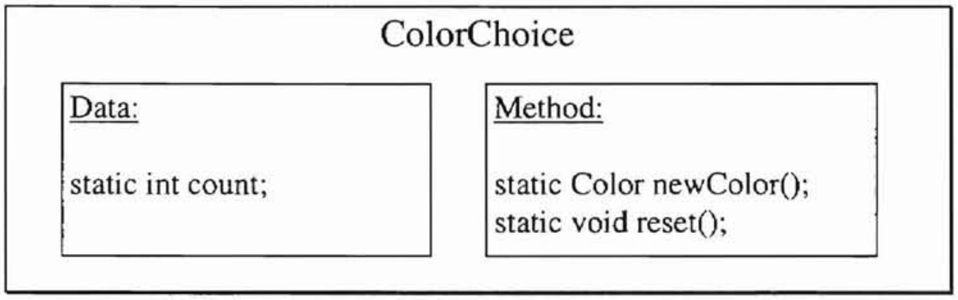
Method:

```
DocText();
void showline(String str);
```

Figure 13 Descriptions of the DocText class in the Kruskal package.

5. ColorChoice Class

This class creates a set of 64 distinct colors (Figure 14). Each call to newColor steps to the "next" color, and wraps around to the beginning again after generating all 64 colors.



ColorChoice

Data:

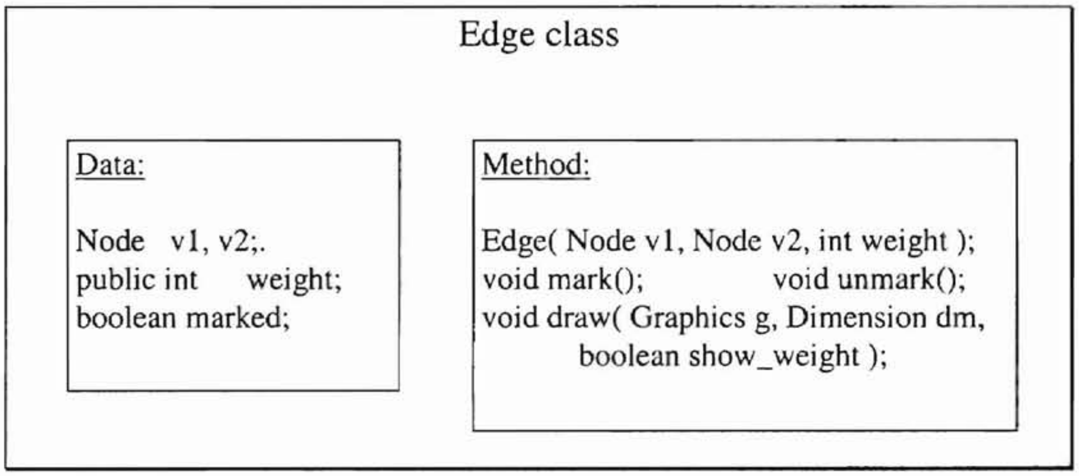
```
static int count;
```

Method:

```
static Color newColor();
static void reset();
```

Figure 14 Descriptions of the ColorChoice class.

6. Edge Class



Edge class

Data:

```
Node   v1, v2;.
public int     weight;
boolean marked;
```

Method:

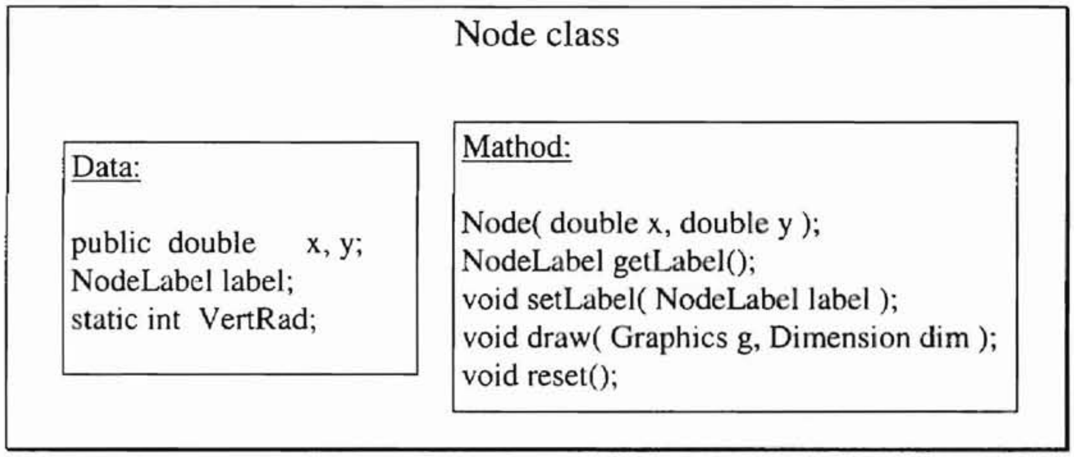
```
Edge( Node v1, Node v2, int weight );
void mark();              void unmark();
void draw( Graphics g, Dimension dm,
         boolean show_weight );
```

Figure 15 Descriptions of the Edge class in the Kruskal package.

The Edge class (Figure 15) encapsulates the notion of a weighted graph edge. It provides a method for drawing the edge (with or without label) using two different "styles", depending on whether the edge is marked or not. Marked edges indicate the minimum spanning tree.

7. Node Class

The Node class (Figure 16) encapsulates the notion of a labeled node in a graph. A node has both a spatial position (i.e. 2D coordinates) and a reference referred to as a NodeLabel. The node class also has method to draw itself.


Node class

Data:

```
public double    x, y;
NodeLabel label;
static int  VertRad;
```

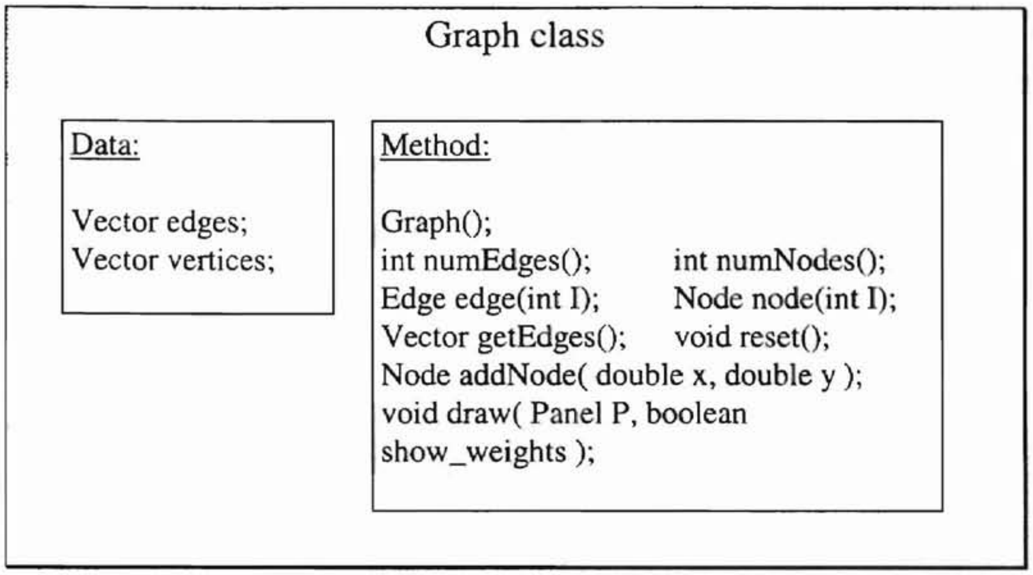
Mathod:

```
Node( double x, double y );
NodeLabel getLabel();
void setLabel( NodeLabel label );
void draw( Graphics g, Dimension dim );
void reset();
```

Figure 16 Descriptions of the Node class in the Kruskal package.

8. Graph Class

The Graph class (Figure 17) encapsulates a graph with weighted edges, where any subgraphs can be marked and displayed differently. Its reset method unmarks all of the edges and nodes.


Graph class

Data:

Vector edges;
Vector vertices;

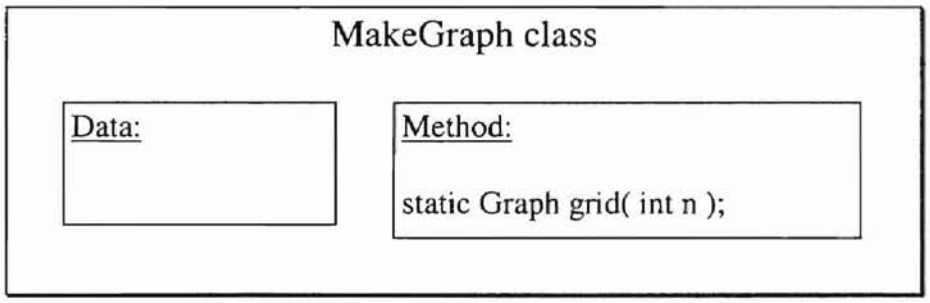
Method:

```
Graph();
int numEdges();          int numNodes();
Edge edge(int I);        Node node(int I);
Vector getEdges();       void reset();
Node addNode( double x, double y );
void draw( Panel P, boolean
show_weights );
```

Figure 17 Descriptions of the Graph class.

9. MakeGraph Class

This class (Figure 18) generates an example for Kruskal's algorithm. The static grid method generates a sparse graph that consists of an n × n grid of nodes, including the edges.


MakeGraph class

Data:

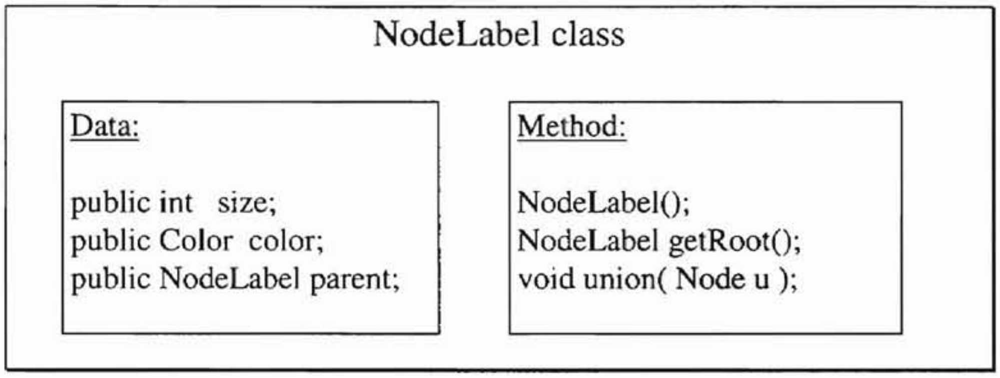
Method:

```
static Graph grid( int n );
```

Figure 18 Descriptions of the MakeGraph class.

10. NodeLabel Class

This class (Figure 19) is used to create unique labels for nodes. Each instance of NodeLabel contains a color and a size, which indicates how many nodes share this label. The getRoot method follows parent pointers up the tree, while the union method makes the tree containing node u point to this label.


NodeLabel class

Data:

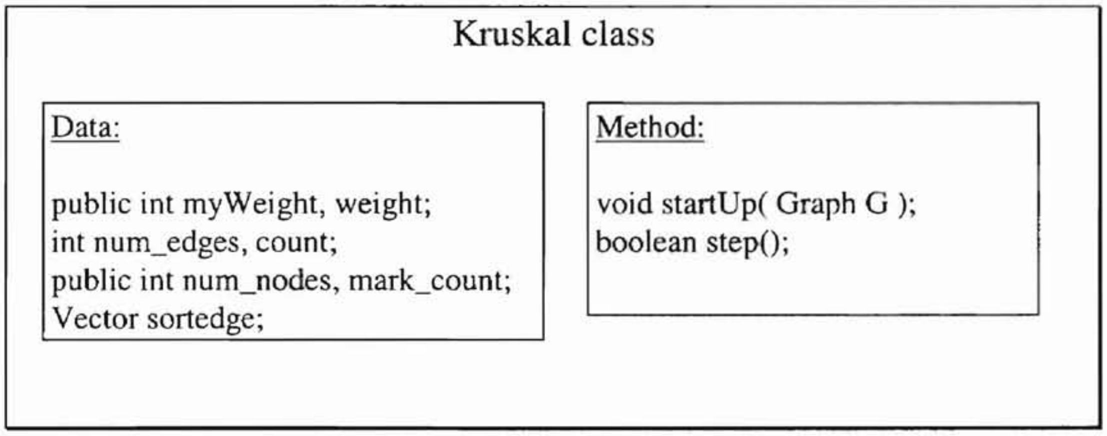
```
public int   size;
public Color  color;
public NodeLabel parent;
```

Method:

```
NodeLabel();
NodeLabel getRoot();
void union( Node u );
```

Figure 19   Descriptions of the NodeLabel class.

11. Kruskal Class


Kruskal class

Data:

```
public int myWeight, weight;
int num_edges, count;
public int num_nodes, mark_count;
Vector sortedge;
```

Method:

```
void startUp( Graph G );
boolean step();
```

Figure 20   Descriptions of the Kruskal class.

This is the class that implements Kruskal's algorithm (Figure 20). It is responsible for sorting the edges (in linear time) and incrementally building the minimum spanning tree.

3.2.3 Prim's Algorithm – Prim Package

Unlike Kruskal's algorithm, Prim's algorithm has the property that the edges in the selected set always form a single tree. Prim's algorithm choose the cheapest edge that extends the tree to include a new node.

Following is the pseudo-code for Prim's algorithm [1].

```
MST-PRIM
1  Q ← V[G]
2  for each u ∈ Q
3       do key[u] ← 0
4  key[r] ← 0
5  π[r] ← NIL
6  while Q ≠ 0
7       do u ← EXTRACT-MIN(Q)
8            for each v ∈ Adj[u]
9                 do fi v ∈ Q and w(u, v) < key[v]
10                     then π[v] ← u
11                          key[v] ← w(u, v)
```

Lines 1 - 4 initialize Q to contain all the vertices and set the key of each vertex to ∝, except for the root r, whose key is set to 0. Line 5 initializes π[r] to NIL, since the root r has no parent. Throughout the algorithm, the set V-Q contains the vertices in the tree being grown. Line 7 identifies a vertex u ∈ Q incident on a light edge crossing the cut (V-Q, Q). Removing u from the set Q adds it to the set V-Q of vertices in the tree. Lines 8 - 11 update the key and π fields of every vertex v adjacent to u but not in the tree.

1. Prim Class

Prim class
extends Applet
implements Runnable, ItemListener

Data:	Method:

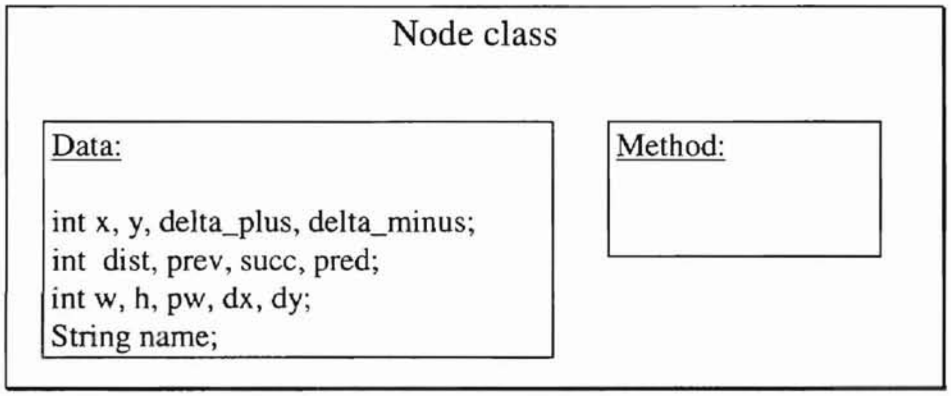
int n, m, u, snode, iteration, step;
int pre_s_first, pre_s_last;
boolean isdigraph;
Node v[];
Edge e[];
TextField weight_field, mst_field;
Thread thread;
Graphics g;
Instruction instruction;
Options options;
Panel disp, left;
CheckboxGroup group;
Checkbox sample1, sample2, sample3;
public int pause;
	void init(); void update(Graphics G);
	void clear(); void stop();
	void stepAlg(); void runAlg();
	void start(); void run();
	void step1(); void step2();
	void step3(); void step4();
	void rdb(); void init_sub();
	void itemStateChanged(ItemEvent e);
	void getInput(String filename);
	Insets getInsets();
	input_graph(InputStream is);
	append_pre_s(int i);
	void remove_pre_s(int i);
	int totalWeight();
	int calWeight();
	double weight(Node n, double x, double y);
	void paintNode(Graphics g, Node n, FontMetrics fm);
	drawArrow(Graphics g, int x1, int y1, int x2, int y2);
	void paintEdge(Graphics g, Edge e, FontMetrics fm);
	void paint(Graphics graph);

Figure 21 Descriptions of the Prim class.

This is the driving class (Figure 21) for Prim's algorithm. This class initializes all of the variables and creates all of the panels for controlling parameters and progressing Prim's algorithm.

2. Node Class

This class (Figure 22) defines all of the variables necessary to identify a node. It is actually similar to struct in C or C++. There is no method in the class.

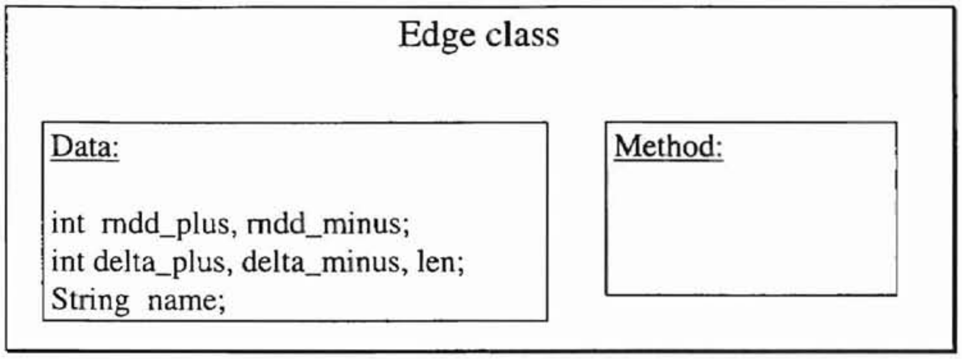


```
int x, y, delta_plus, delta_minus;
int  dist, prev, succ, pred;
int w, h, pw, dx, dy;
String name;
```

Figure 22 Descriptions of the Node class in the Prim package.

3. Edge Class

The Edge class (Figure 23) defines all of the variables necessary to identify an edge. As the Node class mentioned above, there is no method implemented in this class.


Edge class

Data:

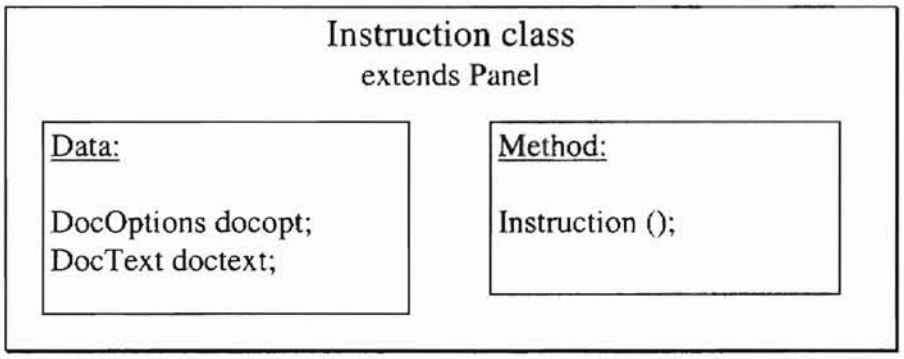
```
int rndd_plus, rndd_minus;
int delta_plus, delta_minus, len;
String  name;
```

Method:

Figure 23 Descriptions of the Edge class in the Prim package.

4. Instruction Class

This is the display panel (Figure 24) on the top of the GUI. It consists of two elements: a choice box and a text area. When an item in the choice box is clicked, the corresponding explanation is shown in the text area.


Instruction class
extends Panel

Data:

```
DocOptions docopt;
DocText doctext;
```

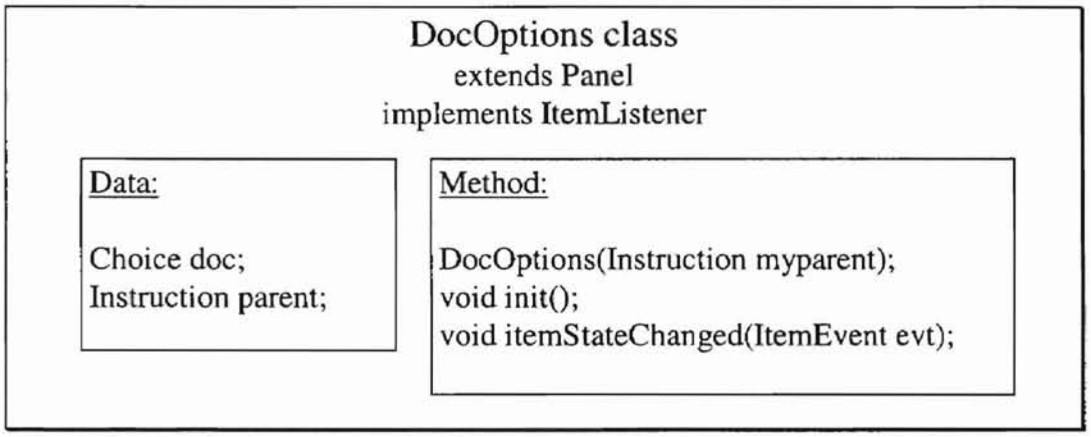
Method:

```
Instruction ();
```

Figure 24 Descriptions of the Instruction class in the Prim package.

5. DocOptions Class

This panel (Figure 25) implements a choice box from which key words can be chosen to show instructions in the text area of its parent, an object of Instruction Class.

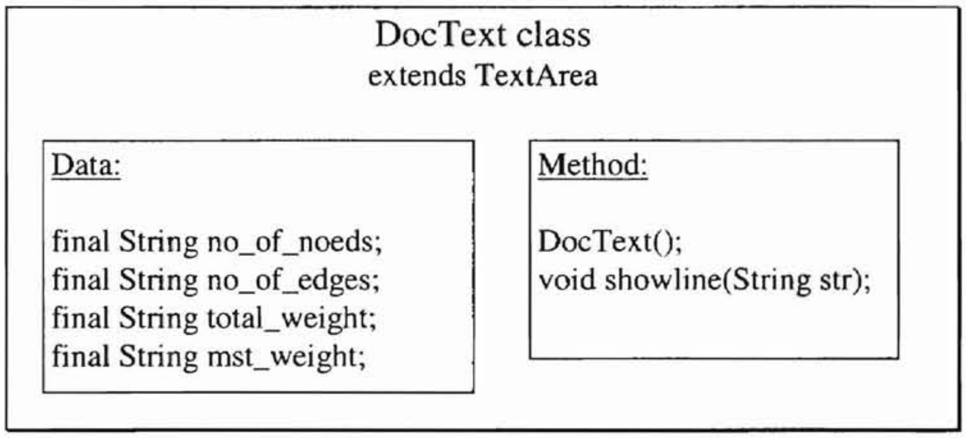


```
Choice doc;
Instruction parent;
```

```
DocOptions(Instruction myparent);
void init();
void itemStateChanged(ItemEvent evt);
```

Figure 25 Descriptions of the DocOptions class in the Prim package.

6. DocText Class



```
final String no_of_noeds;
final String no_of_edges;
final String total_weight;
final String mst_weight;
```

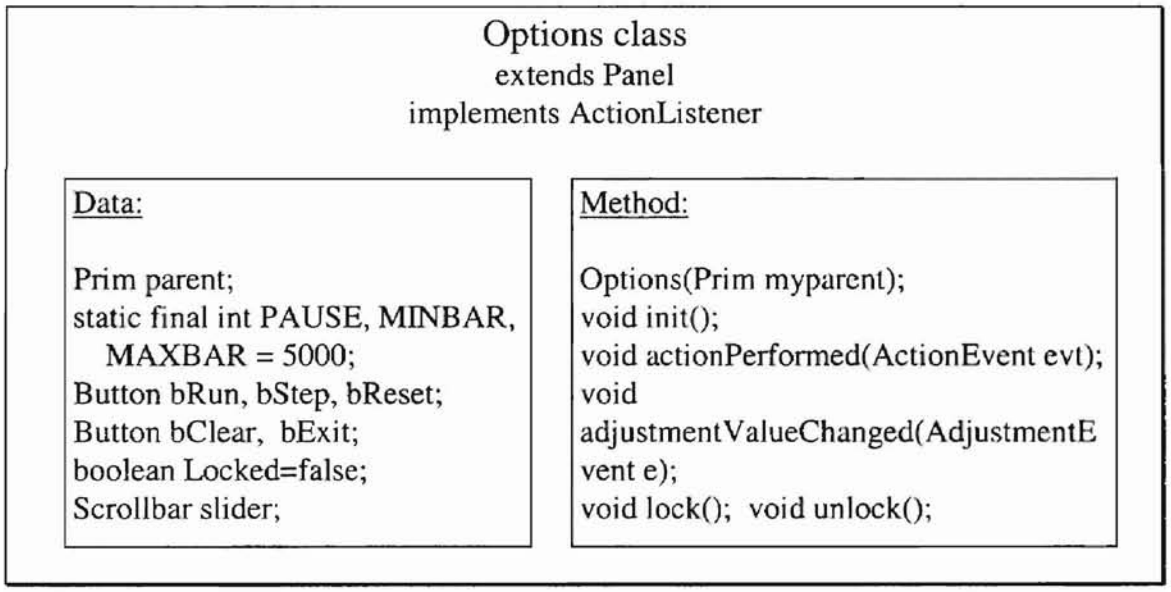
```
DocText();
void showline(String str);
```

Figure 26 Descriptions of the DocText class in the Prim package.

This is the text area (Figure 26) in the Instruction Class. The text area is used to display instructions when a key word is chosen or the algorithm is running.

7. Options Class

This is the control panel (Figure 27) on the right of the GUI. There are several buttons for showing, running, and clearing the visualization of the algorithm. All the buttons are event-listeners. When the user's mouse clicks on one button, the button listens to the click and then takes corresponding actions.



Options class
extends Panel
implements ActionListener

Data:	Method:

Prim parent;
static final int PAUSE, MINBAR,
MAXBAR = 5000;
Button bRun, bStep, bReset;
Button bClear, bExit;
boolean Locked=false;
Scrollbar slider;
	Options(Prim myparent);
	void init();
	void actionPerformed(ActionEvent evt);
	void
	adjustmentValueChanged(AdjustmentE
	vent e);
	void lock(); void unlock();

Figure 27 Descriptions of the Options class in the Prim package.

3.3 Relationships of classes

The processing of classes in one of the three packages is based on communication among their objects as outlined below.

3.3.1 Path package

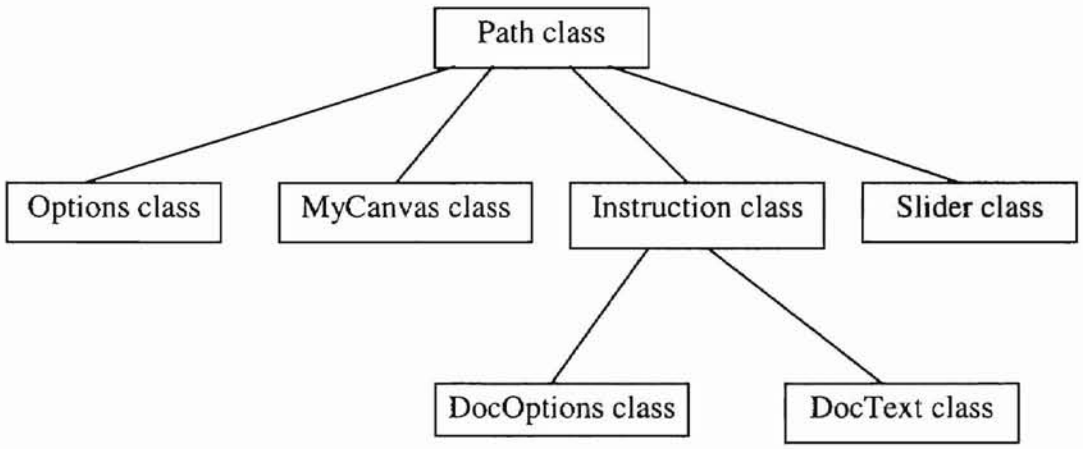


Figure 28 The relationships of classes in the Path package.

The Path class in the Path package (Figure 28) is the main method. The program is started from this class. This class initializes all of the variables and creates all of the panels for the GUI.

The Options class and Slider class provide the user with the control of the program. The MyCanvas class show the visualization of Dijkstra's algorithm. The Instruction class explains how to use the system and what happens step by step when the program is running.

3.3.2 Kruskal's Package

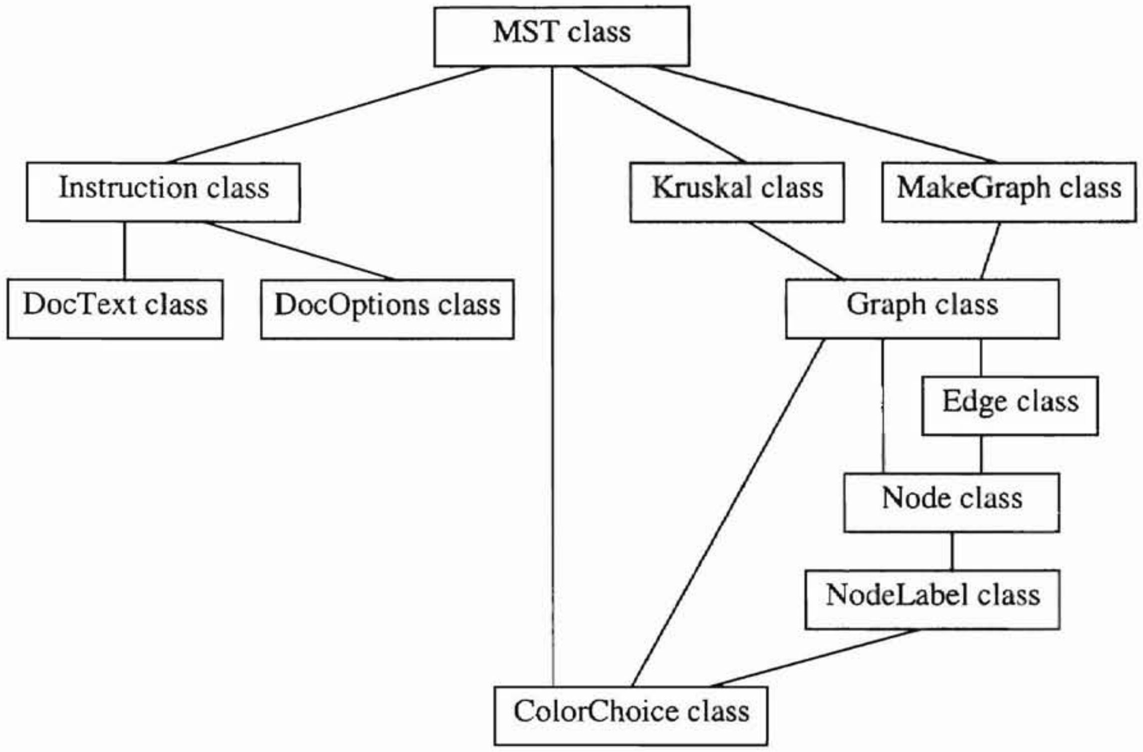


Figure 29 The relationships of classes in the Kruskal package

The MST class is the driving class in the Kruskal package (Figure 29). The subprogram for Kruskal's algorithm is started from this class. This class initializes all of the variables and creates all of the panels.

The MakeGraph class initializes the GUI when it is first called by the MST class, then the Kruskal class visualizes the progress of each step of Kruskal's algorithm by calling the Graph class. The Graph class depends on the Node class and Edge class to draw graphics on the screen when the program is running.

3.3.3 Prim package

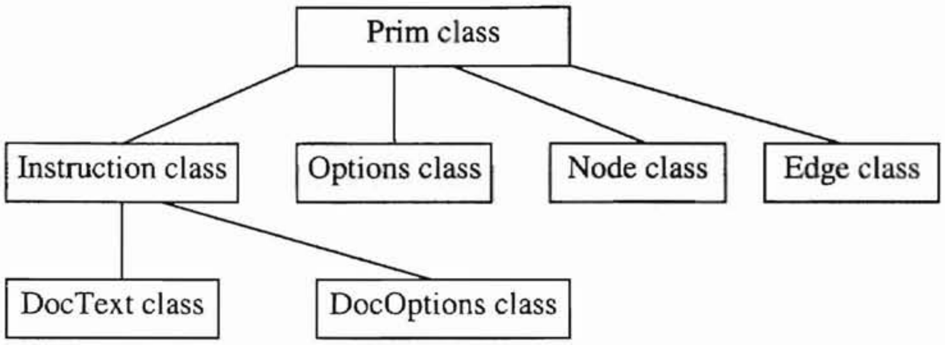


Figure 30 The relationships of classes in the Prim package.

The Prim class is the driving class in the Prim package (Figure 30). This class initializes all of the variables and creates all of the panels for the GUI. This class also visualizes the progress of each step of Prim's algorithm.

As in the other two subprograms, the Instruction class explains how to use the subprogram and what happens step by step when the program is running. The Options class provides the user with the control over the running of the program.

3.4 Description and analysis of input data

We have one default input data set for the Path package for Dijkstra's algorithm. The data set has 10 nodes and 18 edges. The user chooses the default input data by clicking the Example 1 button. The user can add or delete as many nodes as he wants. The user can also build his own input data by clicking the Example 2 button and then adding nodes and drawing edges. Clicking on the center display area will add one node onto the screen.

The first node added to the graphics will be the start node. Pressing down the Ctrl key and clicking on a node will delete that node. A node can be moved to a new position by pressing down the Shift key and clicking a node and then dragging it to the new position. An edge can be added by clicking on an existing node and dragging it to another existing node. The weight of an edge can be adjusted by moving the triangle on that edge. The user has control over how to arrange the input data.

For the Kruskal package for Kruskal's algorithm, the input data are generated by a method in the MST class. The input data ranges between 9 (3×3) to 100 (10×10) nodes. Each node has a randomly chosen color. The user controls the size of input data by adjusting the scrollbar on the right controlling panel. Then the user can display the new input data set by clicking the reset button.

We have three input data files for the Prim package for Prim's algorithm. The input data files are stored on the server computer. When the user chooses one of the three files by clicking one of the choice boxes, the applet, which is running on the client computer, will establish a remote connection to the server computer by initiating a URL object. Then the corresponding data are transferred to the client computer. The new input data will be displayed immediately after the user clicks one of the choice boxes.

3.5 Animation Approach

In a graphical user interface (GUI), the display shows various graphical objects as icons and input devices such as buttons and scroll bars. Using the keyboard or mouse, the user can manipulate these objects directly on the screen at runtime. In this project, graphical objects can be dragged, buttons can be pushed, and scroll bars can be scrolled.

With GUIs, the user can directly interact with the objects on the display instead of watching the animation passively.

In this project, the image objects are nodes that are represented by many identical circles. These nodes are connected together by edges, with the edge weight printed beside each one. Whenever an animating operation occurs, the program changes one node or edge each time.

We basically achieve the animation by marking the next chosen node with a different color. There are two major operations, marking the working nodes (and edges) and marking the chosen node (and edges), to show the user how the algorithm progresses. The roles for marking color are different for the three algorithm animations in order to help the user distinguish the different algorithms.

In the animation for Dijkstra's algorithm, the start node is marked green and other nodes are originally marked dark gray. While the algorithm progresses, the working nodes and edges are marked red. Then the chosen node and edge are permanently marked yellow. No distance is marked for a node when the applet is initiated. The distances from the start node to the nodes are also shown for the working nodes.

In the animation for Kruskal's algorithm, the nodes are originally marked with 64 different colors, even though the difference between some close colors may not be distinguishable by human eyes. An edge is marked red after it is included into one of the forests. All of the nodes in a forest are changed to the same color. The color is decided by the color of the first chosen node in this forest. Eventually all of the temporary forests will merge into one forest, so the nodes will be marked the same color when the algorithm finishes running.

For the animation of Prim's algorithm, the nodes are originally represented by a pink circle with a gray filling color and the edges are marked gray. The working nodes and edges are marked blue. The chosen node is first filled with pink and then marked red. The chosen edge is marked red.

3.6 Major features of the Project

The animation system includes the following features:

- Interactive. A high degree of interactivity is necessary to keep users interested and improve their understanding of the fundamental concepts of an algorithm. Input data is saved and loaded using files on the web server or interactively provided by the user at remote computer. Users are able to select their own input. For novice users, on the other hand, a predefined input data set should be offered. Users also are able to interact with the system when an algorithm is running. In the system, controller elements emphasizing the main steps invite the user to interact with the animation by starting the next logical step of the algorithm.
- Adjustable time (speed). The user can control the speed of transitions in the presented algorithm: they may be sudden or gradual. Hence the user can see the changes to the data more clearly according to their learning speeds. The animation system runs at a range of comfortable speed. It should be independent from hardware and degrade gracefully if the system ran on slower hardware.
- Help is available for the system as a whole, for components on the user interface, and for the common terminology of greedy algorithms.

- Color used to convey information clearly. Color plays an important role on the system. The progress of the running algorithm is mainly indicated by changing color. Color is used for five distinct purposes: encoding the state of data structures, tying views together, highlighting activity, emphasizing patterns, and making history visible.
- Includes algorithm analysis. Statistics of the algorithm's behavior is included wherever appropriate. Once an algorithm is run several times, it would be useful for the student to be able to see how different options affect the operation of the algorithms. These options are either parameters to the algorithms themselves, or variations in the input to the algorithm. Statistics information is shown during the execution of the animations. Visual comparisons between different similar algorithms further deepens the understanding of the student.

IV. ENVIRONMENT AND PLATFORM

This chapter contains a description of the environment and operating system that are used for implementation the animation system of this project.

4.1 Java development kit

The Java development kit (JDK) contains the software and tools needed to compile, debug, and run applets and applications written using the Java programming language. The JDK provides a compiler that is necessary to compile all kinds of Java programs. It also provides an interpreter to run Java applications. The JDK Applet Viewer or any Java-compatible Web browsers can be used to run Java applets.

The Java programs in this project are compiled by the JDK version 1.1.6 compiler. Applets that depend on any new 1.1.6 APIs may not work on any browsers that support only JDK 1.1.2 or earlier versions, such as Internet Explorer 3.0 and Netscape Navigator 4.02, or earlier versions.

JDK version 1.1.6 was released on April 24, 1998. In JDKs prior to version 1.1.6, field fileNameMap of java.net.URLConnection was public. In JDK 1.1.6 and later, fileNameMap is private. Accessor and mutator methods getFileNameMap and setFileNameMap have been added to java.net.URLConnection beginning in JDK 1.1.6. Also, java.io.File follows proper Windows conventions. Windows recognizes a distinct working directory for each drive. A bare drive name refers to the working directory for that drive.

4.2 World Wide Web (WWW) and Web browsers

The Internet is the collection of interconnected networks all over the world. Users on the Internet can exchange electronic mail with each other, transfer files from computer to computer that may be thousands of miles away, and access databases of information remotely [27].

In its short history, the Internet has experienced exponential growth. Much of the growth comes from connecting existing networks to the Internet. The glue that holds the Internet together is the TCP/IP reference model and TCP/IP protocol stack.

In the early 1990s, the WWW (World Wide Web) made another revolution and brought millions of new, nonacademic users to the Internet [27]. This application did not change any of the underlying facilities but made them easier to use for nonacademic people. Together with Web browsers such as the Mosaic viewer and Netscape Navigator, the WWW made it possible for a site to set up a number of pages in information containing text, pictures, sound, and even video, with embedded links to other pages. By clicking on a link, the user is suddenly transported to the page pointed to by the link.

The WWW is mostly used on the Internet, but they do not mean the same thing. The Web refers to a body of information, an abstract space of knowledge, while the Internet refers to the physical side of the global network [28]. The WWW uses the Internet to transmit hypermedia documents between computer users internationally. It is possible to use the WWW software without having to use the Internet. But Internet is necessary in order to make full use of and participate in the WWW.

Web software is designed based on a distributed client-server architecture. The language that Web clients and servers use to communicate with each other is called the

HyperText Markup Language (HTML). By embedding the markup commands within each HTML file and standardizing them, it becomes possible for any Web browser to read and reformat any Web page [27]. HTML is widely praised for its ease of use. It allows users to produce Web pages that include text, graphics, and pointers to other Web pages. We have used HTML 3.0 to write the Web pages for this project.

Web pages must be viewed via Web browsers. The user should be able to view the Web pages of this project using common Web browsers such as Netscape Navigator, Internet Explorer, and HotJava. However, the user must use Navigator version 4.06, Internet Explorer version 4, or higher version. Because only these new versions support JDK 1.1.6, the new version of the Java language used to develop the programs in this project.

4.3 Platform

The computer used for developing Java programs in this project is Sun Microsystems Enterprise 3000. This machine is configured with two UltraSparc processors each running at 256 MHz, and 256MB of main memory. It is running Sun's Solaris 2.5.1 (SunOS 2.5.1) operating system. The Solaris 2.x is a full 32-bit operating system based on UNIX System V Release 4 (SVR4). It has extensive functionality in areas such as symmetric multiprocessing (MP) with multithreads (MT), real-time functionality, increased security, and improved system administration. The Solaris operating system is one of the operating systems that has a Java compiler for JDK 1.1.6.

The programs of this project have also been tested in Windows 98 operating system with a Java compiler for JDK 1.1.6. If the programs are compiled and stored on a

Windows 98 operating system, the user can view the project locally using a common Web browser, without having to connect to the Internet.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced an interactive animation of some greedy algorithms. The project is delivered as a Web page that can be viewed over the Internet by a common Web browser such as Netscape Navigator (4.06 or higher version), Internet Explorer, and HotJava.

5.1 Conclusions

This project develops an animation system to illustrate a group of greedy algorithms in detail. The animated web page can be used for teaching the greedy algorithms interactively on the Internet. The system offers an integrated hypermedia learning environment for greedy algorithms. The system animates the following greedy algorithms: Kruskal's algorithm and Prim's algorithm for the minimum spanning tree problem, and Dijkstra's algorithm for the shortest path problem.

The animation system is written mostly in Java. All the files reside in the same directory. The applets are embedded in the Web pages and will be loaded and unloaded appropriately as the Web pages are loaded and unloaded from the Web browser.

The animation system is made available on the Internet. The temporary WWW address is: http://www.su.okstate.edu/students/yiyin/greedy.html. It has an easy to use interface that illustrates an algorithm through its results. Target users are students learning greedy algorithms for the first time. A user can run the animation system remotely using common Java-compatible Web browsers such as Netscape, Internet Explorer, or HotJava.

The system is interactive. It will accept parameters from the user at runtime and pass them to the applets. With the interactive visualization, the user can also change some graphical parameters interactively at running time, and immediately see the resulting structure produced by the algorithm under consideration.

5.2 Future Work

The following are some of the areas where future work on this application is suggested:

1. Adding a third dimension for the graph. The third dimension can be used in many ways, such as to represent an extra dimension in the data, to illustrate the passage of time or the algorithm's progress for an aesthetic effect. The third dimension may also be used to provide state information about an algorithm.
2. Using multiple views. It is generally more effective to illustrate an algorithm with several different views than with a single monolithic view. The animation system will use multiple views, each displaying a small number of aspects of the algorithm. Each view is easy to comprehend in isolation, and the composition of several views is more informative than the sum of their individual contributions.
3. Using accompanying sound to convey information more enjoyable. Then it will be a truly multimedia system.
4. Providing a rewindable event history, which allows users to run algorithm traces backwards, or step slowly through an algorithm's history. When animation is used to explain an unfamiliar algorithm, it is helpful to present a static view of the history of the algorithm and its data structure. It allows the user to become familiar with the

dynamic behavior of the algorithm at his own speed, and to focus on the crucial events where significant changes happen, without paying too much attention to repetitive events.

LITERATURE CITED

1. Gloor, P., Dynes, S., Lee, I. *Animated Algorithms: A Hypermedia Learning Environment for Introduction to Algorithms.* The MIT Press, Cambridge, Mass. 1993.
2. Brown, M.H., Najork, M.A. "Algorithm Animation Using 3D Interactive Graphics." *Technical Report 110a*, DEC Systems Research Center. 1993.
3. Sedgewick, R. *Algorithms.* Addison-Wesley, Reading, Mass. 1988.
4. Dijkstra, E.W. "A note on two problems in connection with graphs." *Numer. Math.* 1: 269-271. 1959.
5. Moret, B.M.E., Shapiro, H.D. *Algorithms from P to NP.* Benjamin/Cummings, Redwood City, CA. 1991.
6. Brown, M.H., R. Sedgewick, R. "A System for Algorithm Animation." *ACM Computer Graphics.* 18(3): 177-186. 1984.
7. Brown, M.H. "Exploring Algorithms Using Balsa-II." *IEEE Computer* (May 1988) 14-36. 1988.
8. Brown, M.H. "Zeus: A System for Algorithm Animation and Multi-View Editing." *Proceedings, 1991 IEEE Workshop on Visual Languages* 4-9. 1991.
9. Brown, M.H., Hershberger, J. "Color and Sound in Algorithm Animation." *Technical Report 76a*, DEC Systems Research Center. 1991.
10. Stasko, J.T. "Tango: A Framework and System for Algorithm Animation." *IEEE Computer* (September 1990) 27-39. 1990.

11. Duisberg, R.A. "Animated Graphical Interfaces." *ACM SIG CHI 86 Conference on Human Factors in Computing Systems*. 131-136. 1986.

12. Stasko, J.T. "Three-Dimensional Computation Visualization." *Technical Report GIT-GVU-92-20,* Georgia Institute of Technology, Atlanta, GA. 1992.

13. Bentley, J.L., Kernighan, B.W. "A System for Algorithm Animation." *Computing Systems* (Winter 1991) 4(1): 5-31. 1991.

14. Tufte, E.R. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Conn. 1983.

15. Byrne, M.D., Richard Catrambone, R., Stasko, J.T. "Do Algorithm Animations Aid Learning?" *Technical report GIT-GVU-96-18 (August, 1996)*, Georgia Institute of Technology, GA. 1996.

16. Stasko, J.T., Lawrence, A. "Empirically Assessing Algorithm Animations as Learning Aids." Chapter 28 in: *Software Visualization : Programming As a Multimedia Experience*, John T. Stasko, John B. Domingue, Marc H. Brown, and Blaine A. Price (eds.). The MIT Press, Cambridge, Mass. 1997.

17. Courtois, T. *Java Networking and Communications*. Prentice-Hall, Inc, Upper Saddle River, NJ. 1998.

18. Deo, N. and Pang, C. "Shortest-path algorithm: taxonomy and annotation." *Networks* 14: 275-323. 1984.

19. Stanek, W. *Web Publishing Unleashed.* Sams.net Publishing, Indianapolis, IN. 1996.

20. Arra, S.K. *Object-oriented Data Structure Animation.* Unpublished Master's Thesis. Oklahoma State University, 1992.

21. Kummetha, V.C.S.R. *A Level Linked R* tree Structure with an Application Using x-window Graphical Interface.* Unpublished Master's Thesis. Oklahoma State University, 1993.

22. Shen, H.C. *A Visual Aid for the Learning of Tree-based Data Structures.* Unpublished Master's Thesis. Oklahoma State University, 1994.

23. Xu, C. *Multimedia Visualization of Abstract Data Type.* Unpublished Master's Thesis. Oklahoma State University, 1997.

24. Harvick, L.H. *Rule Based Data Structure Animation.* Unpublished Master's Thesis. Oklahoma State University, 1997.

25. Lin, B.H. *An Object-oriented Graphic User Interface for Visualization of B-tree's Animator.* Unpublished Master's Thesis. Oklahoma State University, 1997.

26. Lewis, J., Loftus, W. *Java Software solutions: Foundations of Program Design.* Addison-Wesley, Reading, Massachusetts, 1998.

27. Tanenbaum, A.S. *Computer Networks.* Prentice Hall PTR, Upper Saddle River, New Jersey, 1996.

28. Pfaffenberger, B. *Netscape Navigator: Surfing the Web and Exploring the Internet.* AP Professional, Chestnut Hill, Massachusetts, 1995.

VITA 2

YI YIN

Candidate for the Degree of
Master of Science

Dissertation: ANIMATED PRESENTATION OF SOME GREEDY ALGORITHMS

Major field: Computer Science

Biographical:

Personal Data: Born in Guilin, Guangxi, P. R. China, July, 20, 1964, the son of Youde Zhang and Sulan Yin.

Education: Graduated 10th High school of Guilin, Guilin, Guangxi, China, July 1981; received Bachelor of Science degree in Botany from Xiamen University, Xiamen, Fujian, China, July 1985; received Master of Science degree in Plant Ecology from Xiamen University, Xiamen, Fujian, China, December 1990; received Doctor of Philosophy degree in Plant Sciences from Oklahoma State University, Stillwater, Oklahoma, May, 1998; completed the requirements for the Master of Science degree at Oklahoma State University in December, 1998.

Professional Experience: Teaching Assistant, Department of Biology, Guangxi Normal University, China, August, 1985 to July, 1987; Research Assistant, Department of Biology, Xiamen University, China, August, 1987 to December, 1990; Instructor, Department of Biology, Guangxi Normal University, China, January, 1991 to July, 1993; Research Assistant, Department of Botany, Oklahoma State University, August, 1993 to December, 1996; Teaching Assistant, Department of Botany, Oklahoma State University, January, 1996 to December, 1997; Teaching Assistant, Department of Computer Science, Oklahoma State University, January, 1998 to December, 1998.

Professional Membership: Phycological Society of America, January, 1994 to December, 1997; American Society of Limnology and Oceanography, January, 1994 to December, 1997.