University of Tennessee, Knoxville

TRACE: Tennessee Research and Creative Exchange

Doctoral Dissertations

Graduate School

5-2009

Sketchmate: A digital drawing tool for the splay tree data structure

Michael Cajetan Orsega
University of Tennessee

Follow this and additional works at: https://trace.tennessee.edu/utk_graddiss

Recommended Citation

Orsega, Michael Cajetan, "Sketchmate: A digital drawing tool for the splay tree data structure. " PhD diss., University of Tennessee, 2009.
https://trace.tennessee.edu/utk_graddiss/5989

This Dissertation is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a dissertation written by Michael Cajetan Orsega entitled "Sketchmate: A digital drawing tool for the splay tree data structure." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Bradley T. Vander Zanden, Major Professor

We have read this dissertation and recommend its acceptance:

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:
I am submitting herewith a dissertation written by Michael Cajetan Orsega entitled "Sketchmate: A Digital Drawing Tool for the Splay Tree Data Structure." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Bradley T. Vander Zanden, Major Professor

We have read this dissertation
and recommend its acceptance:

Lynne E. Parker

James S. Plank

Christopher H. Skinner

Accepted for the Council:

Carolyn R. Hodges
Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Sketchmate: A Digital Drawing Tool for the Splay Tree Data Structure

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Michael Cajetan Orsega
May 2009

Copyright © 2009 by Michael Cajetan Orsega.
All rights reserved.

Dedication

For Kaitlin and Jackson, proof that with support you can do anything you put your mind to.

"I can do everything through him who gives me strength." - Philippians 4:13

Acknowledgments

I would like to thank all of my committee members, Dr. Brad Vander Zanden, Dr. Lynne Parker, Dr. Jim Plank, and Dr. Chris Skinner for serving on my committee and providing the various assistance and direction as I worked on my research.

I would also like to offer a special thank you to Dr. Vander Zanden whom I met with on a weekly basis over the past few years to discuss a variety of different topics including, but not limited to, the structure and design of Java programs, the thought pattern of students and users, general pedagogy, job searching advice, hiking, and a variety of other things unrelated to my research.

Additionally, I owe a special thank you to Dr. Skinner who spent countless hours helping me plan, prepare, analyze, and write up my research. Your office door always seemed to be open and you always stopped what you were working on to help answer my questions. The only way I could possibly repay you would be to attempt to emulate what you have done for me when I'm in a similar situation working with my students.

I would also like to thank Dr. David Banks, Dr. Vander Zanden, and their respective students for agreeing to allow me use of the time within their classes to run my experiments with Sketchmate. Additionally, I appreciate the assistance from Dr. Skinner's students, Emily Kirk and Emily Fuller, for their help conducting my final experiment.

Finally, I would like to thank my wife Kristina and my children Kaitlin and Jackson for their love and support during my time at The University of Tennessee. It is hard to believe Jackson does not know of a time when his Daddy wasn't in school. We have all had an interesting time over the past few years and I hope that I can return to normal life soon and become the husband and father you deserve.

"We did it, we did it, we did it. Yea!" – Dora the Explorer

Abstract

When discussing complex concepts, people often draw pictures to provide concrete representations of these ideas. Instructors use images to reinforce information when teaching abstract concepts. Computer programs can be used to help present these ideas and provide added benefits.

This dissertation explores the creation of such a computer program and the lessons learned during the development process. We chose to focus on the splay tree data structure and report the many details we learned during development. Our findings are useful for others creating teaching tools in other domains.

Currently there are a variety of existing programs for splay tree instruction, however these tools lack functionality. We designed Sketchmate to add more functionality and present it as two different programs: an instructor version and a student version.

The instructor's version gives users the ability to create specific tree configurations and provides a history mechanism to revert the tree to a previous state. The student version presents two different types of splay-tree exercises. One type allows students to manipulate individual nodes through drag and drop operations. The second exercise type allows students to click buttons to demonstrate higher level operations.

Both versions employ a novel algorithm to lay out tree nodes to maximize screen space. Additionally, both make use of smooth animation between the algorithm steps, providing context for the user.

We conducted a series of experiments with each version to measure the usefulness and ease of use for Sketchmate. Results from tests with the instructor's tool suggest users can prepare and demonstrate more examples using Sketchmate and those examples are more accurate than with pencil and paper.

Experiments with the student's tool suggest students working example problems enjoy using Sketchmate more than pencil and paper. Also, students using Sketchmate with feedback complete their exercises quicker and still perform just as well on follow up quizzes, suggesting that they learned just as much, with less time spent with the material.

These results form the basis for concluding that Sketchmate should prove useful as a possible learning tool for teaching splay trees, both for instructors during algorithm presentation and for students working exercises.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

It is often stated that a picture is worth a thousand words, probably because there are many different points and perspectives that can come about by viewing relationships within images. This fact is often leveraged in the classroom when instructors introduce complex, abstract concepts to their students. Often mathematics instructors will draw images to help explain geometric relationships. Chemistry instructors draw a variety of symbols to explain chemical reactions and molecular structure. These images are used as a way to reinforce the abstract concepts instructors explain during their lectures.

The use of images to supplement instruction is typical in the realm of computer science when presenting abstract concepts and ideas. Instructors make use of images when attempting to show students what is happening in the computer's memory for example. Whether discussing the memory allocation of a simple variable, an array variable, or the overall layout of the stack and heap space, images are used along with the oral or written descriptions of these ideas.

Later, when students begin learning about different ways of storing data through the use of specific data structures like trees and graphs, instructors will continue to draw images on the board to help students better visualize how the data is logically arranged. Again, the whiteboard is typically used because it is readily available and extremely flexible, allowing them to draw any image they desire.

Typically instructors create their images on a whiteboard (or blackboard) or an overhead projector. These methods offer a variety of advantages. First, these tools are readily available in most classrooms and require very little, if any training for use. These tools also allow instructors to freely create any shape or text they like very quickly. However, this free-form creation also serves as a disadvantage in that it is easy for the instructor to make a mistake due to a tool's inability to have any knowledge of the material the instructor is attempting to teach. Additionally, the whiteboard does not provide any mechanism for logging the instructor's images. And while overhead slides would provide the opportunity to make copies of the images, this process would require additional time.

By contrast, instructors could also just as easily make use of computer programs to create the images. And while computers may not be available in every classroom, the availability of laptops has increased the ability to use the computer for instruction. Computer programs can offer some advantages over the whiteboard in that they can be written

to provide domain-specific information about the material being presented and limit certain operations to reduce the possibility of instructors presenting incorrect information. These programs can also be written in a way such that each image can be stored in an external file for later use. This means that instructors can then make those files available to students who missed class, to help answer questions during office hours, or just to provide them to students for review.

As an example of how a computer can take advantage of domain-specific knowledge, a computer program could produce smooth animations that show how rotations can be used to rebalance a tree or revert to a previous state of a structure to answer student questions without re-drawing. Trying to simulate these operations on a whiteboard would require drawing and re-drawing structures which takes time and instructors may make errors when drawing them. Thus, when using a whiteboard the instructor must focus on performing the operations correctly and drawing the structures accurately, as well as describing clearly the manipulations they are performing. In contrast, with an appropriate computer-assisted drawing tool that ensures that operations are performed correctly and structures are drawn accurately, the instructor can focus principally on providing clear explanations of the manipulations they are performing on the trees.

In addition to lecturing with images, instructors typically reinforce the material by having students solve homework problems. While these problems come in a variety of different types and formats, one useful method is to have students draw pictures depicting the intermediate states of an algorithm's progress. Exercises such as these are typically done using paper and pencil because they are so readily available. However, once again this method has drawbacks. First, when working with large data structures, students are forced to recopy all of the structure's data, even if they have nothing to do with the current action related to the algorithm. Second, it is not possible to provide the student with immediate feedback about whether or not the answer is correct, and if it is incorrect, then to provide feedback about why it is incorrect. Finally, while these hard copy records are easy to collect and keep track of, grading requires the instructor to take a detailed look at them and often demands a large amount of the instructor's time.

In an attempt to address some of these drawbacks for both students and instructors, we created Sketchmate. Because we developed Sketchmate from scratch, our first decision needed to be whether we wanted to develop a tool that could address a few simple operations on a variety of different structures, or if we should just pick one structure and fully explore all of the possible operations. We chose to select one structure and develop the program in a way that would make it easy to expand to a variety of different structures at a later time. By doing this, we would be able to better develop a tool that would be easy to use and useful for students. We could then modify Sketchmate so that users can manipulate other structures.

We wanted to select a structure that was not trivial, yet was complex enough that instructors were more likely to use images to support the concepts. Additionally we wanted a structure that could easily be adapted to other data structures later. Based on these requirements, we selected the splay tree structure. A splay tree is a binary search tree that has an extra set of restrictions placed on its operations. Splay trees are often covered in an introductory data structures course which increases the need for such a tool. Also there are a variety of other data structures based on binary trees (like heaps) or on binary search

trees (like red-black trees); therefore future work on these other structures could build on our work on the splay tree data structure.

Sketchmate comes in two different versions, one intended for instructors and one for students. The instructor's version allows users to quickly create any splay tree that they like and then perform a series of operations on that tree. Additionally, Sketchmate gives the user the ability to view the history of operations and revert back to a previous version of the tree. At that time, the instructor can either demonstrate the operations again, or choose a new set of operations to demonstrate. During this entire process, the program will be doing the calculations so the instructor is ensured of correct results, thus freeing the instructor to concentrate more on describing the operations clearly and less on drawing the diagram accurately.

The student version of Sketchmate is designed to be used as a homework practice tool. Instructors can create external data files containing splay-tree problems. Students can then start Sketchmate and use the mouse to perform drag and drop operations, as well as button clicks to manipulate the splay trees provided so as to demonstrate the required operations. If students should attempt an invalid or incorrect action in the process, Sketchmate is capable of delivering detailed feedback specific to that situation. This student practice and specific feedback can then be used to give students a better grasp of how the algorithms work and hopefully increase their learning. As an additional function for the instructors, the student's version of Sketchmate is capable of instantly grading student work, thus giving instructors more time to focus on other class related needs.

Both versions of Sketchmate were developed through a series of incremental development steps. At each step additional functionality was created, tested, and sometimes changed in order to develop an intuitive, useful tool. We began the research with the instructor's version. One important goal was to create a program that instructors could use quickly without investing a lot of time upfront learning how to use the program. We also wanted to make sure that Sketchmate's functionality was such that it met the necessary requirements for preparing and delivering a lecture.

To this end, after developing a stable version of Sketchmate, we ran a study to evaluate it. The first experiment in the study was designed to measure the amount of time to create a specific set of splay tree demonstrations and the accuracy of the results. Our purpose was to find out if users could learn Sketchmate and produce results as quickly as with pencil and paper, a tool they were already familiar with. Additionally since people make mistakes, we were able to use these results to get an idea of how often this happened. Our results showed that participants using Sketchmate were able to prepare significantly more demonstrations and that their results were significantly more accurate than those using pencil and paper.

The second experiment with the instructor's tool used the same version of the program and the same subjects. However, this time participants were asked to prepare one specific splay-tree demonstration and then present that demonstration in the form of a lecture. Responses were evaluated based on time to deliver content and accuracy. Once again, our purpose was to measure the speed and efficiency of Sketchmate and compare them with a whiteboard, the traditional form of delivery. Results suggested that participants using Sketchmate were able to deliver their material significantly faster and their images were significantly more accurate than those using the whiteboard.

After completing the evaluation of the instructor's tool, we took the animation building blocks and added new functionality around them to develop the student's tool. Research suggests that user interaction with a computer program will increase student learning, and therefore we designed the student version to be used as a homework delivery mechanism. We also wanted to harness the power of the computer and use Sketchmate to deliver immediate, specific, constructive feedback when students made mistakes on their homework. However when designing the experiments, we wanted to isolate the effects of using Sketchmate to simply replicate paper-and-pencil drawing from the effects of providing detailed, constructive feedback. For that reason, we split our second study into two different experiments, run with students from two different semesters of an introductory data structures course. Both experiments used students who were currently learning about splay trees. The first experiment used a version of the student tool that provided only minimal feedback (a message box telling the user they attempted an incorrect action). The second experiment used Sketchmate with richer feedback specific to the action the user attempted and which provided more details when the action was incorrect. We measured the amount of time to complete the exercises and then gave students a follow up pop quiz. We hoped to use the completion time as a gauge of the ease of use for Sketchmate. We used the pop quiz results to give us a measure of student learning.

The remainder of this dissertation provides further details about Sketchmate and the experiments we used to evaluate it. Chapter 2 gives a summary of related work on both the use of animations and images for instruction, as well as a summary of similar existing computer programs. Chapter 3 gives a brief overview of the splay tree data structure for reference. Chapter 4 specifies the details of both the instructor and student versions of Sketchmate, including unique features and design justifications. The next two chapters, 5 and 6, give the detailed description of each experiment, including their results and a discussion of those results. Finally Chapter 7 gives conclusions and possible future work.

Chapter 2

Related Work

This chapter reviews some work that has been conducted in the past on a variety of tools used for instruction. We begin in Section 2.1 by presenting research comparing instructions using text with images and text only. This research is not within the domain of computer science, however the basic tenants of using text with images is relevant. Next Section 2.2 covers a variety of tools available to instructors. We present both generic presentation packages such as Microsoft PowerPoint and programs designed specifically for computer science education. Finally, Section 2.3 describes five different existing programs specifically designed for instructors covering splay trees.

2.1 Animations for Instruction

Although Sketchmate's purpose is not as an animation tool, the only existing splay-tree programs are animators and the research has focused on the use of animations to help instruction. Instructors have long made use of diagrams and animations to help clarify complex operations in many fields. But the question remains as to how useful these images are for the students.

Research by Mayer, et al. has studied the value of the images used along with textual information regarding complex, mechanical systems and found the illustrations do help to improve the recall of explanative information [Mayer, 1989]. As an extension of this knowledge, researchers also experimented with animations and their usefulness [Mayer and Anderson, 1991, Mayer and Anderson, 1992, Mayer and Sims, 1994]. These studies showed that animations were helpful as a means of creating a connection between the concepts and the application of the concepts.

Although some results are encouraging for the use of animation, other studies suggest mixed results. Reiber, Boyce and Assad examined the use of animation for studying Newton's Laws [Rieber et al., 1990]. They found that there was no significant difference between the presentation of static graphics, animated graphics, or no graphics at all. However, it should be noted that because the material was Newton's Laws, all subjects had some real world experience with the material before beginning instruction.

Other researchers have found mixed results in learning outcomes when simply adding visual aids to lectures. Chris Hundhausen and his associates performed a meta-analysis of 24 algorithm visualization studies. One of the results of this analysis was that students

who are more actively engaged with the animation process tend to learn better [Hundhausen et al., 2002]. Research by Narayanan et al. showed similar results and also found that hypermedia presentations are better than purely static presentations [Narayanan and Hegarty, 2002].

Work by Palmiter and Elkerton showed mixed results when using animations. They provided users with a set of instructions for performing a task. Some of the users were additionally shown an animated demonstration [Palmiter and Elkerton, 1991]. When tested immediately following the instruction, the subjects who saw the animation performed better and stated that they enjoyed the experience better than those who received only text. However, when participants were tested again one week later, those who saw the animation did not perform as well as those who did not. On the whole the studies suggest that presenting concepts using animated illustrations does no harm and may do some good, especially if a way is found to actively involve students in the animation process. It is also clear that instructors are likely to continue to convey concepts by drawing and transforming diagrams, and that providing helpful tools to assist them is a useful goal.

Pane, Corbett and John compared the presentation of biological processes with high quality static images and animations [Pane et al., 1996]. Their study showed that there was no significant difference in learning between the images and animations. Due to the high monetary and time cost to create the animations, they suggest that it may not be worth it to spend the effort at creating the animations.

Due to the mixed success at using animations for education, our primary goal here is to offer a tool that allows the user to quickly generate animations. It is natural to believe that computer science instructors are going to continue to diagram and illustrate complex conceptual algorithms, whether the literature shows it is significantly suitable or not. For this reason, we want to give them a tool that minimizes the amount of time to learn the tool, time to create an animation and time distributing animations to their students. The next section discusses possible options for computer science instructors.

2.2 Existing Programs (General)

The use of animation to describe computer science algorithms dates back to the early 1980's with the creation of movies like *Sorting out Sorting* [Baecker, 1981]. It is natural to understand that as technology has progressed, it has become easier and cheaper to make movies, animations and tools to help explain complex ideas.

Within the realm of Computer Science education, previous work has focused on two main areas: programs where you modify existing code to produce animations and programs where you write code specifically to draw animations.

We begin this section with a description of graphical editing programs that can be used to create freeform images. Next we present the computer science specific tools that allow for modifying code to generate images and then discuss the programs which allow the user to create the specific code to create images. In the next section, we discuss a set of computer programs designed specifically for presentation and instruction using tools such as tablet personal computers and personal digital assistants (PDA). We conclude the section with a discussion of the existing tools specific for splay trees.

2.2.1 Non-Domain Specific Graphical Editing Programs

Currently there are a number of general graphical editing tools that can be used to create presentations on splay trees. Software tools such as Microsoft PowerPoint give the user the ability to design splay-tree images and even to simulate animation by creating multiple images on different slides. Also, Adobe Flash provides the ability to create animations along a timeline with primitive shapes like boxes and lines. However, the creation of these images is time consuming. The reason for the awkwardness of creating animated images using these tools is that they do not contain specific knowledge about the instructional domain of splay trees and therefore cannot provide time-saving sketching, manipulation or animation assistance for the instructor.

2.2.2 Modifying Existing Code

The first group of animation schemes includes programs such as JVALL [Dershem et al., 2002], VIPS [Shimomura and Isoda, 1991], XTANGO [Stasko, 1992], POLKA [Stasko and Kraemer, 1993] and others [Baker et al., 1999, Hansen et al., 1998, Rasala, 1999, Sangwan et al., 1998] that allow the user to write their algorithms in a popular language, such as Java or C++. Once the programmer writes their code for the algorithm, they replace existing function calls or data types with calls to animation libraries. These libraries serve as a wrapper around the functionality of the algorithm. That is, in addition to performing the desired action, such as adding a link or rotating a node, the code also generates images that show the resulting structure. In most cases, this is essentially a tool that allows the programmer to visually debug their program code.

While a visual debugger can be very helpful for addressing issues in implementations of these algorithms, the programmer still needs to have some idea of the steps necessary in the algorithm in order to create any type of image. Additionally the tool's programming language limits who will use which tool because Java programmers need to use Java libraries and C++ programmers, C++ libraries.

By contrast, our tool is a precursor to implementation for any student who is learning an algorithm. It allows the student to focus on the concepts, ideas and steps of the process without worrying about things like syntax and library calls. Once the student interacts with our illustrator, they have a better chance to turn to whatever programming language they choose and implement the algorithm in code.

2.2.3 Writing Animation Code

Another class of animators includes programs such as Balsa II [Brown, 1988], Ska [Hansen et al., 2002] and Step94 [Lieberman and Fry, 1995]. These programs require the programmer to write a particular set of code whose sole purpose is to create an animation. Some of these programs use common programming languages, while others have developed their own scripting language for the animation.

While these tools are useful for the creation of generic animations, they require that the programmer have a thorough understanding of the algorithm. These tools also require an in depth knowledge of either their programming language or their application interface. While most were developed so that the programmer would need little knowledge of their system, some admit that there is a steep learning curve.

Two other animation tools, JAWAA [Pierson and Rodger, 1998, Akingbade et al., 2003] and ANIMAL [Rößling and Freisleben, 2002, Rößling et al., 2000] do allow for easy drag and drop creation of animations with little knowledge of the algorithm involved. This gives the user the power to create nice animations, but does not provide for standard built-in operations depending on the specific data structure. This tool also allows the user to perform actions on the data structure that are not allowable, such as removing a node from the middle of a stack or queue.

We appreciate the power and usefulness of these programs for animation generation. However, the goal of our project is to make a tool that allows for easy generation of animations without an in-depth knowledge of the algorithm. These animations should be creatable on the fly while the instructor is lecturing in the classroom. In addition, our tool seeks to allow the user to only perform structure specific operations.

2.2.4 Notation and Presentation Tools

The tools we have presented so far create animations of algorithms specific to the Computer Science field, which is one of our goals. Additionally, our illustrator is used to present concepts and ideas to students. When presenting, the lecturer typically also needs to make notations on the presentation. The following tools are designed for notation and presentation of general material.

The first tool, ScreenCrayons [Dan R. Olsen et al., 2004] is a general notation tool. This allows the user to create a screen capture, then easily highlight and create notes on the material. It also employs a file management system for easy retrieval of the notes. Although our tool does not incorporate the ability to make notes along with their images, it is something that we will consider adding in the future.

The remainder of the tools allow the presenter to project their notes onto previously created slides. This is done with tablet personal computers [Buckalew and Porter, 1994, Anderson et al., 2004, Wilkerson et al., 2005, Golub, 2004, Berque et al., 2001, Berque, 2006] and personal digital assistants [Myers, 2001, Myers et al., 1998]. In each case, the lecture can use their own handwriting to notate the material and then save these notes for later review. Our system models this behavior by allowing the user to create the animations and then make notations a separate window. The illustrator links the notes with that particular image. Therefore during playback of the animation, the viewer can watch the animation along with a description of what is taking place. Once again, this is functionality that we will consider adding to Sketchmate later.

2.3 Existing Programs (Splay Trees)

There are also a variety of tools available that have some of the same characteristics as Sketchmate. The majority of these tools appear as either a standalone Java application or a Java applet embedded into a web page. Three good examples of these applications are Princeton University's "Growing Tree" [Sanders et al., 2002], University of La Laguna's "Java Models" [Gogeshvili, 2002], and University of Southern California's "Splay Tree Applet" [Ierardi and Li, 1996].

All of these tools provide users with a set of controls for selecting splay-tree operations and for navigating through the animations. All three applications allow the user to delete

or find a value in a splay tree. Java Models and Growing Tree also give users the ability to insert any value they wish and to control the speed of the animation. However there are some differences in how the programs begin. Java Models and Splay Tree Applet both begin with a random, pre-populated tree, which is helpful for quickly getting started with some examples. On the other hand Growing Tree begins with an empty tree; however it also provides a mechanism to generate a random splay tree with which to work.

A fourth tool, New York University's "Binary Search Tree Visualization" [Biermann, 1998], also begins with a pre-populated tree and allows for any value to be inserted, deleted or found. The unique part of this tool is the way it presents the algorithm's steps in a comic strip-like form. That is, the initial tree is displayed and the user selects an operation. Next a set of forward and backward controls is enabled which allow the user to scroll through the steps of the operation. Although this tool does not employ animation, it does provide a snapshot of the tree at each step, along with a textual description. Users are also able to step back in the process to see previous versions of the tree.

One serious limitation of each of these tools is the ability to begin with any tree configuration. For example, if an instructor wishes to discuss a specific example tree in the class textbook, none of these tools gives them the ability to start out with such a tree. Users would be able to create a tree by inserting values, however because of the splay tree's rebalancing after each insertion, the instructor would need to know the exact insertion order, which could be challenging. Additionally, the time required to insert nodes one by one would take away time from instruction.

Another issue with these programs is the lack of a history mechanism. For example, if during lecture a student asked the instructor to repeat the steps and explanation again, there is no quick way to do so. Instead, the instructor would need to start the program over from the beginning. And if they had created their specific tree by adding values one at a time, this could be quite time consuming.

As far as the use of these programs for students to practice homework problems, none allow users to load a pre-defined set of homework exercises to solve. The existing programs do allow users to click buttons to perform splay operations. However because there is no problem to solve, there is no way to end the session and have the work graded. By designing Sketchmate to load a particular problem set, now instructors can assign problems for any tree they desire. For example with Sketchmate, instructors can now create problems that come directly from either the chapter material in the textbook, or in the questions at the end of the chapter. Instructors can also design their problem set to run in the class time allotted and can select the types and difficulty of problems to make the most out of the students' time.

Additionally, none of the programs allow users to perform much practice with a splay tree. They may provide a splay operation that allows a user to select a node and splay it. However, they do not allow the user to perform the sub-steps associated with a splay operation, such as disconnecting a subtree from the main tree, re-arranging nodes, and establishing new connections between nodes. Neither do they allow the user to perform a complete insertion or deletion operation, since the user cannot choose the insertion point for a node, or manually delete a node. Also when animation does occur, these programs have multiple parts of the tree moving at the same time. This movement is sometimes unnecessary as parts of the tree unaffected by the algorithm are sometimes moving, which can lead to distraction for the user.

Only one of these tools, "Binary Search Tree Visualization" provides a history mechanism to allow users to step back in the algorithm. And while this tool is unique in that feature, it uses a set of cartoon images, rather than animation, to illustrate the changes to a tree. The program does highlight the affected parts of the tree using color, however it does not provide the movement within the relevant parts of the tree as Sketchmate does. By keeping one single image in Sketchmate, we hope to provide more continuity between each step of the algorithm so that students will be able to better see how the splay-tree structure is changed as the algorithm progresses.

Finally, none of these programs have a mechanism built in so that user actions are logged in a file for future reference. This is probably due to the fact that these tools are available on the internet as Java applets. Nonetheless, with Sketchmate we wanted to have a tool that gave users a permanent record of their work. This record is useful for the instructor's tool because it provides a history for things like class notes. It is also useful in the student tool because now students can submit this log file to their instructors as proof that they completed the exercises. Also the fact that the log file contains grading information provides extra help for instructors as they can now use that time when they would have been grading for other things like creating better exercises or answering student questions.

The final program, TRAKLA2 [Korhonen and Malmi, 2000,Myller et al., 2007] is similar to the student version of Sketchmate in that it presents a predefined problem to the user and allows the user to use mouse actions to manipulate the tree. Additionally, TRAKLA2 provides a mechanism for automatic grading of the user's work. However, TRAKLA2 differs in that there are a wide variety of algorithm problems that are not limited to splay trees. For example, there are problems that ask students to work on a variety of binary search tree operations like insertion and deletion. It also contains problems related to sorting algorithms like Quicksort, Radix sort, and Heapsort. Overall, TRAKLA2's allows users to work a wide variety of different problem types. However, TRAKLA2 does not allow for problems with a set configuration of values. Therefore while instructors may be able to assign problems in TRAKLA2, they cannot be assured that the students will get practice on particular node arrangements.

Additionally, while TRAKLA2 does provide an automated grading feature, it is simply an overall score at the end of the problem. That is, there is no feedback on what users did correctly or incorrectly. Instead, it just grades the result and provides a numeric value. Sketchmate improves on this by including immediate feedback when the user performs an incorrect action. When this happens, Sketchmate provides a description of what the user attempted and when that type of action would have been appropriate.

Overall the TRAKLA2 tool is very useful in that it provides what we consider to be a breadth-first investigation for a tool used for teaching data structures. By contrast, Sketchmate is being developed as a depth-first investigation. We wanted to select one particular structure and be sure that the program could be constructed in such a way as to be as useful and intuitive as possible before proceeding to the next structure.

Chapter 3

Splay Tree Overview

Sketchmate is designed to perform and display basic splay-tree operations. A thorough description of splay trees can be found in a standard introductory textbook on data structures [Weiss, 2006,Lewis and Denenberg, 1991,Goodrich and Tamassia, 2004,Shaffer, 2000]. This chapter presents a quick overview of splay trees.

A splay tree is a semi-balanced binary search tree. As a binary search tree it has the following properties:

1. Each node contains at most 2 children
2. Each node stores a value
3. The left subtree of a node contains only values that are less than the node's value
4. The right subtree of a node contains only values that are greater than the node's value (in this dissertation we only consider splay trees with unique values)

Splay trees are self-balancing. That is, any operation performed on the tree will be done in such a way as to keep roughly half the nodes in the left subtree and half the nodes in the right subtree. Doing so helps to keep the running times of each subsequent operation as efficient as possible. In fact, it has been shown that the amortized running time of each of the splay-tree operations, find, insert, and delete is $O(\log n)$. An amortized running time is calculated by measuring the time required to perform a sequence of operations averaged over all the operations performed. As stated in Cormen, "Amortized analysis can be used to show that the average cost of an operation is small, if one averages over a sequence of operations, even though a single operation within the sequence might be expensive. Amortized analysis differs from average-case analysis in that probability is not involved; an amortized analysis guarantees that average performance of each operation in the worst case." [Cormen et al., 2001].

3.1 Basic Splay Operations (Rotations)

Each of the three splay tree operations use a series of single and double rotations known as splay operations. These rotations, described below, are designed to bring the selected node to a place in the tree that is closer to the root.

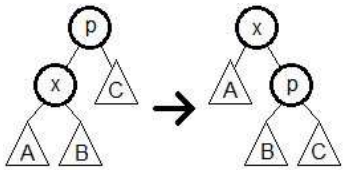

Figure 3.1: Initial tree on left that is transformed after performing a zig operation on node x. Note how the target node x is moved up by rotating its parent node down to the right.

3.1.1 Single Rotations: Zig and Zag

There are two basic splay operations, each of which consists of only a single rotation. A zig operation moves the target node up by rotating its parent node down to the right, whereas its mirror operation, a zag, moves the target node up by rotating the parent node down to the left. Figure 3.1 shows the initial and final states for a zig operation on node x, with p as the parent, and A, B, and C as subtrees. Zig and zag operations are performed only when the immediate ancestor of the node is the root of the tree.

3.1.2 Double Rotations: Zig-Zig, Zag-Zag, Zig-Zag, and Zag-Zig

If the node is two or more levels from the root of the tree, then one of four splay operations, zig-zig, zag-zag, zig-zag, and zag-zig, is chosen to advance the node two levels toward the root. Each of these splay operations is a double rotation made up of different combinations of zigs and zags. Figure 3.2 shows the initial and final states of a zag-zig operation on node x, with p as the parent, g as the grandparent, and A, B, C, and D as subtrees.

3.2 The Splay Tree Find Operation

The first step in a find operation is to locate the desired value in the tree. This is done by applying the rules of binary search trees that state that at any given node in the tree, every value in the left subtree is less than the current node's value and that every value in the right subtree is greater than the current node's value. Therefore by starting at the root node, a recursive series of comparisons can be made between the value we are interested in and the current node's value. If our value is less than the current node's value, we move to the left child otherwise we move to the right child and compare the value there. Eventually the comparisons will lead us to either the desired node or to a null node. If the latter occurs (i.e., the value is not found), the last non-null node is used in the remainder of the steps.

The next step in the find operation is to perform a series of splay operations to bring the selected node to the root of the tree. Double rotations (zig-zig, zag-zag, zig-zag, or zag-zig) are performed until the selected node either reaches the root, in which case we are done, or the node becomes the child of the root. In the second case, we then perform a single rotation (zig or zag) to bring the selected node to the root.

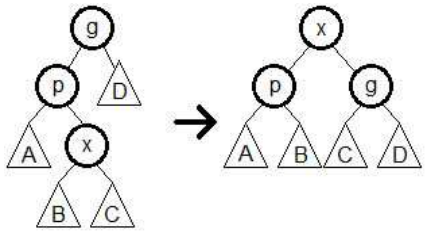

Figure 3.2: Initial tree on left that is transformed after performing a zag-zig operation on node x. x has been moved up two levels by rotating p down to the left and g down to the right.

Once the selected node is at the root, we compare that node's value with the value being searched. If they match, then we successfully located the value in the tree, otherwise we know that the tree did not contain this value.

3.3 The Splay Tree Insert Operation

The first step in the insert operation is to determine the unique insertion location in the current tree. This process is similar to the first step of the find operation in that we begin at the root and compare the value to be inserted with the current node. The process continues with comparisons until a null node is reached. This null node is the insertion location and the null value is replaced with a node containing the value to be inserted. After insertion, the new node is brought to the root of the tree through a series of splay operations.

3.4 The Splay Tree Delete Operation

The first step in the delete operation is to locate the node to be deleted in the tree. Once again this is done by starting at the root and comparing the value to be deleted with the current node's value and moving either left or right based on the rules of the binary search tree just as we did in the find operation.

Once this is done, a series of splay operations is performed to bring this selected node to the root of the tree. Once the selected node is located at the root, we remove it from the tree, thus potentially leaving us with two distinct trees.

The next step in the delete process will have us splay the largest value in the left subtree to the root of that tree. By splaying this particular node, we are assured of ending up with a root having no right child in this left subtree. Therefore we can now connect the two subtrees by making the root of the right subtree become the right child of the root of the left subtree.

Chapter 4

Description of Sketchmate

Sketchmate was initially designed only to assist instructors when teaching the splay tree data structure. However, we ultimately saw this tool being useful not only as a tool to be used for demonstrating algorithms, but also as a tool to be used by students when practicing splay tree algorithms. For this reason, we developed two distinct versions of Sketchmate: one for the instructor and one for the student. Both programs are Java standalone applications based on the same set of Java classes. However, each program has a unique interface with different functionality.

This chapter describes the details about the interface and functionality of each program. The end of the chapter provides an explanation of our design choices.

4.1 The Instructor's Tool

Sketchmate is designed to assist instructors when lecturing to a class or helping a student during one-on-one interaction. Sketchmate is comprised of three different window panes: a design pane that allows an instructor to create custom or random trees, an animate pane that allows the instructor to manipulate the tree, and a revert pane that allows the instructor to revert back to previous versions of the tree.

4.1.1 The Design Pane

The type of trees that can be created in the design pane was derived from informal discussions with a variety of instructors and students. From these discussions we identified several types of trees that instructors would be likely to create:

1. Splay trees created from scratch by adding nodes, one at a time.
2. Random splay trees created either with or without a specified number of nodes and/or a range of node values.
3. Splay trees created by providing a specified list of node values in a particular sequence. This tree differs from the type in item 1 because the entire tree should be shown at once, without showing the steps required to create each node.
4. Specific, complete splay trees that the instructor wants to copy from a textbook.

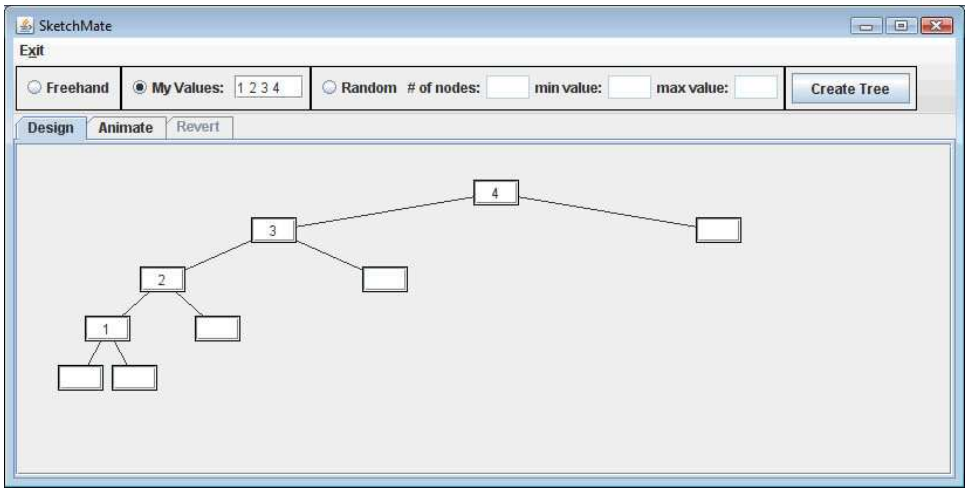

Figure 4.1: Sketchmate's design pane

We initially envisioned allowing users to sketch a tree, much as they would on a white-board. However, after investigations with several prototypes, we found that trees can be constructed more quickly using the keyboard and simply tabbing between nodes. We also found that users were most comfortable filling in each level of a tree before proceeding to the next level (i.e., a breadth-first traversal). The design pane that ultimately evolved is shown in Figure 4.1.

An interesting interaction with the design pane occurs when the user wants to duplicate a splay tree, such as one illustrated in a textbook. Initially a single text box representing the root of the tree is presented. As soon as the user types a value into this text box, two more text boxes are displayed: one for the root's left child and one for its right child. The user can then use the tab key to navigate to the next node. Once again, as a value is entered in this new node, two new text boxes are created for this node's children. Subsequent presses of the tab key will visit the nodes in an order-level (breadth-first) traversal. Additionally, the up arrow key will shift the selection to the current node's parent and the down arrow will shift the selection to the current node's left child. This type and tab navigation technique has proven efficient and well liked by test subjects.

In addition to constructing the tree this way, Sketchmate also allows users to provide a list of node values. A tree is then created by inserting these nodes into an empty splay tree in the order they are entered. Users can key their values into a text box at the top of the application and then click a button to have the tree created.

The final creation method that Sketchmate provides in the design pane is the ability to create a random tree. Users use a text box to enter the number of values they want in their tree and then click a button to generate the tree. Sketchmate proceeds to generate a set of random numbers and then creates the tree in a similar fashion as the previous method. That is, these random values are inserted into an empty splay tree in the order they were generated, thus creating a unique, random tree.

The animate pane described in the next section provides the way for a user to create a tree from scratch by inserting one node at a time.

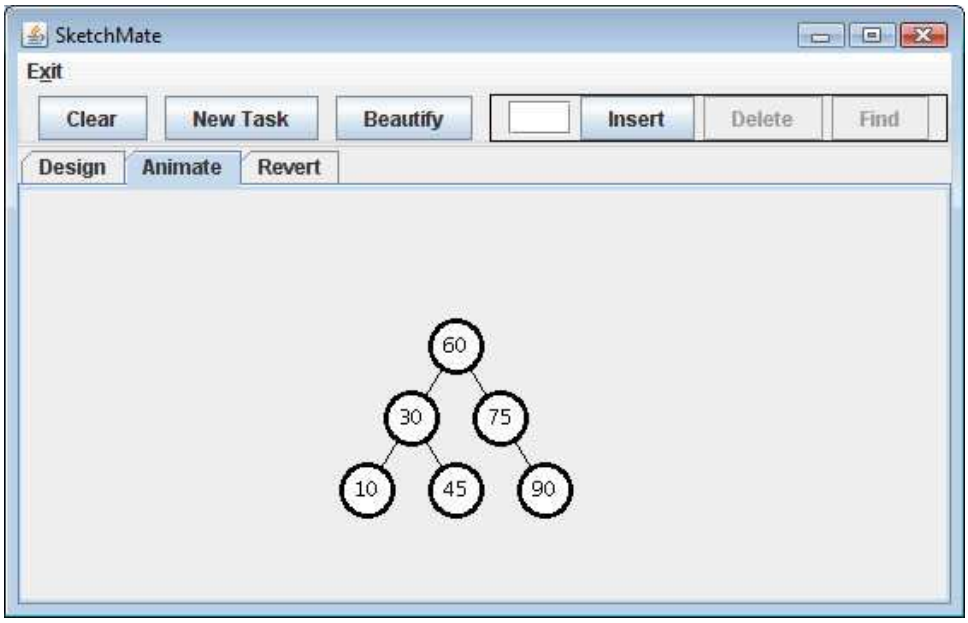

Figure 4.2: Sketchmate's animate pane.

4.1.2 The Animate Pane

After creating a tree using the design pane, users can use the animate pane (see Figure 4.2) to perform three different types of splay-tree operations: insert, delete, find. After the user chooses a particular operation, the actions needed to perform this operation, such as locating a node or splaying a node to the root, are shown as a sequence of animations. Sketchmate currently pauses at the end of each animated action, which gives the instructor a chance to insert commentary that explains what is happening.

For example, when performing a rotation that will detach a subtree from a node, a pause is inserted just after the subtree becomes detached (see Figure 4.3), allowing the instructor to explain why the subtree was detached and to describe the next rotational step.

Sketchmate also gives an instructor the ability to prompt the class for what the next step will be, thus increasing student interaction. While some other existing tools give the user the ability to pause the animation, our user testing suggests that the animations themselves are too course-grained. In particular, existing animation tools move multiple parts of the tree simultaneously and we have found that doing so tends to be overwhelming to the viewer, and that it is better to move single pieces of the tree sequentially.

4.1.3 The Revert Pane

Another unique feature of Sketchmate is the revert pane, (see Figure 4.4). Each time an operation is performed in the animate pane, the operation and the resulting tree are saved and can later be displayed in the revert pane. The operations that have been performed on the tree are listed along the left side of the window as radio buttons. When the user

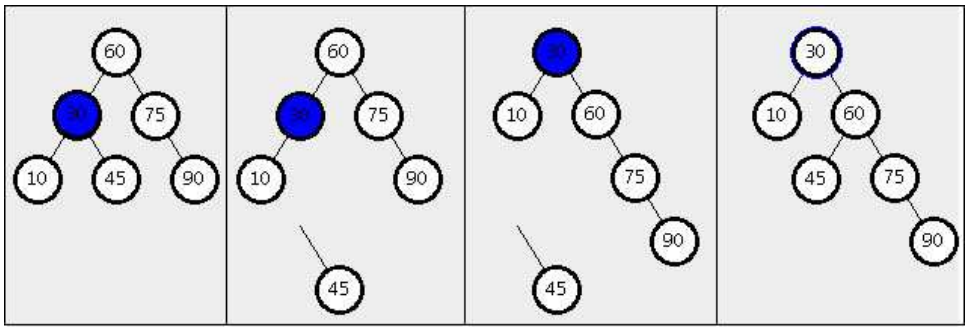

Figure 4.3: The steps necessary to perform the zig operation on the node with value 30. Frame 1: Initial tree. Frame 2: Right subtree is detached. Frame 3: Node 30 is rotated right. Frame 4: Subtree is reattached as a left child of 60.

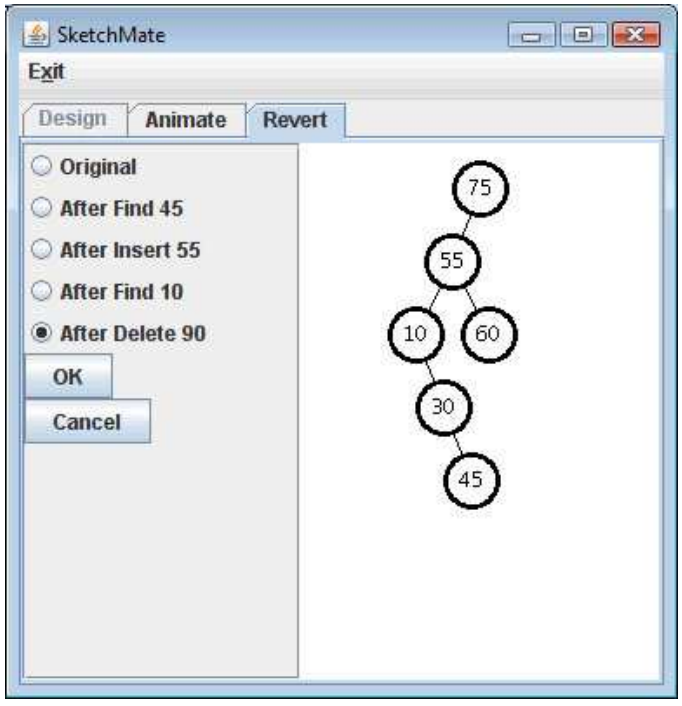

Figure 4.4: Sketchmate's revert pane

selects one of the operations, the tree that resulted from that operation is shown on the pane to the right. The instructor can then request that that version of the tree be restored to the animate pane.

This display of historical operations leads to some interesting uses. The most obvious use, and the reason why this pane was created, was so that instructors could revert back to a previous state of the tree for further discussion. It is likely that an instructor may be demonstrating an example and a student in the class either wants to see the demonstration again or perhaps would like to see a different operation performed on that tree in its previous state. In either of these cases, the instructor can simply click this pane, select a previous version of the tree, and return to the animate pane to demonstrate the operation.

Another use of this pane could be at the end of the demonstration, as a review of the steps performed. This is especially useful when beginning with an empty tree and then showing the state of the tree after inserting values one at a time. The instructor can go through the demonstration and at the end switch over to the revert pane. The instructor can then sequentially click the radio buttons to show the students how the tree changed with each insertion.

4.1.4 Complete Operation Examples in Sketchmate

Because the nature of Sketchmate's demonstrations is through animation, it is difficult to get the full experience of using Sketchmate by reading about it. However, the following figures attempt to provide a flavor of the animations provided by Sketchmate by showing the Sketchmate screen displays at each step in the process for a find (Figure 4.5), insert (Figure 4.6), and delete (Figure 4.7) operation. Note that the transitions between frames is done via Sketchmate animations, which last roughly 1 second per transition.

4.2 The Student's Tool

The student version of Sketchmate is designed to be used by a student to complete homework problems about splay trees and their operations. It presents students with two different types of problems: demonstrating one of the six splay operations, or demonstrating a find, insert, or delete operation. The first type of problem is designed to give the students fine-grained practice with the fundamental operation used to self-balance splay trees, while the second type of problem is designed to give the students courser-grained practice with how splay operations are used by the various splay tree methods to re-balance the tree.

The initial student version of Sketchmate provided only primitive feedback in that it prevented users from performing invalid operations but did not explain why the operation was invalid. For example, if a user attempted to perform a zag operation on a node that was a left child, Sketchmate displayed an error message saying that this was an invalid operation. Note that this operation is invalid because the zag operation requires that the current node be the right child of its parent. This version of Sketchmate also alerted the student if the student tried to submit an answer before performing all of the steps in the operation. Sketchmate would display a dialog box stating "The answer is incomplete. Please try to finish this problem." The dialog box also contained two buttons with the text, "Submit Anyway" and "I will keep trying".

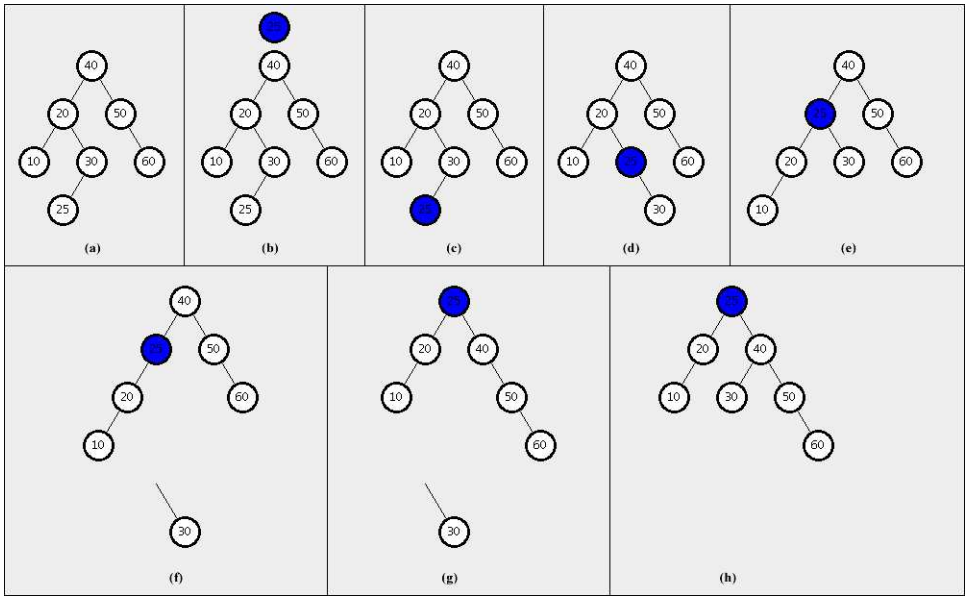

Figure 4.5: Screen capture of the intermediate steps of a find operation using Sketchmate. Frame (a) shows the initial configuration of the tree before we perform a find operation for node 25. The first step in this operation is to locate the node using binary search tree rules. Sketchmate displays an extra, highlighted node shown in frame (b) to demonstrate this process. Frame (c) shows the location of the node to be splayed to the root. The first step in this process is a zig-zag operation, which is shown in frames (d) and (e). The next step is to perform a zig operation. This will rotate 25 to the root of the tree and make node 40 become its right child. Because 25 already has a right child, node 30 is first detached from the tree as shown in frame (f), then the rotation occurs in frame (g) and finally node 30 is reattached to its new parent, 40 in frame (h).

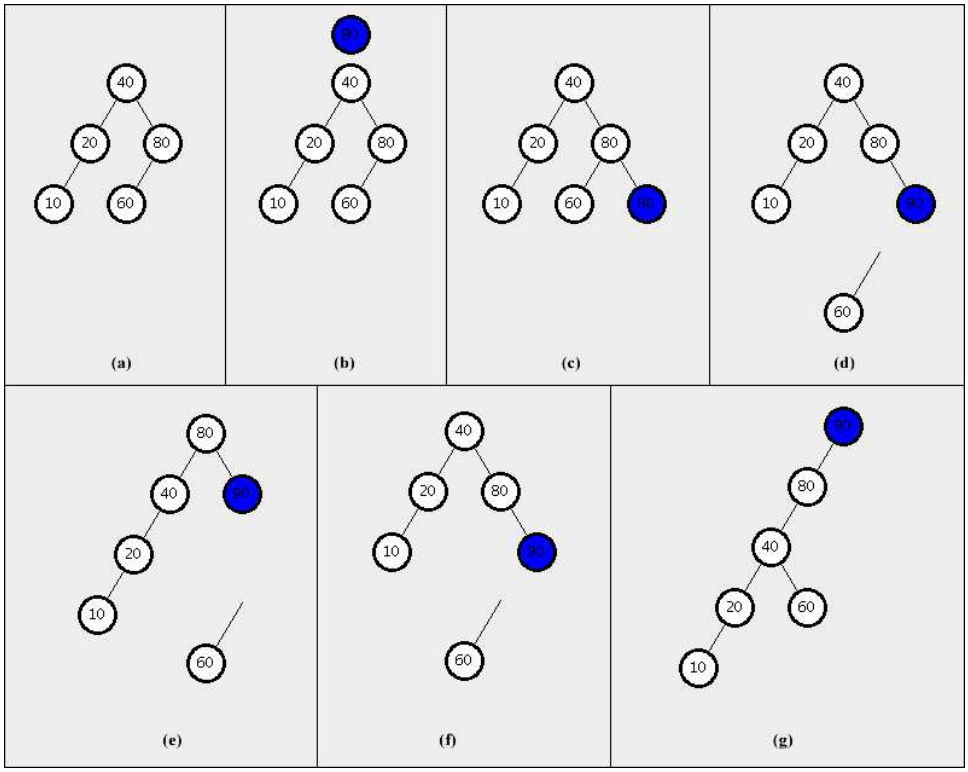

Figure 4.6: Screen capture of the intermediate steps of an insert operation using Sketchmate. Frame (a) shows the initial state of the tree. The first step is to locate the insertion point for this value. Sketchmate displays an extra, highlighted node shown in frame (b) to demonstrate the process. Frame (c) shows the newly inserted node. At this point the new node will be splayed to the root using zag-zag operation. As in previous examples, any subtrees that will be replaced are detached and moved away from the tree, as shown when performing the first zag in frame (d). After the rotation occurs in frame (e), node 60 is reattached to its new parent, node 40 as shown in frame (f). Finally, frame (g) shows the final state of the tree after the second zag operation is performed and node 90 has been splayed to the root of the tree.

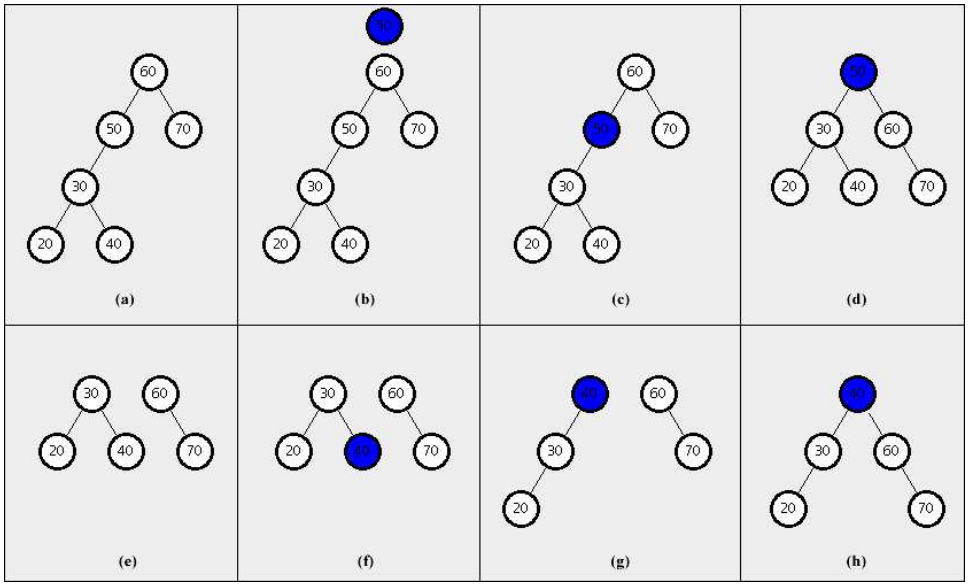

Figure 4.7: Screen capture of the intermediate steps of a delete operation using Sketchmate. Frame (a) shows the initial configuration of the tree before we delete node 50. The first step is to locate the insertion point for this value. Sketchmate displays an extra, highlighted node shown in frame (b) to demonstrate the process. Once 50 is located as shown in frame (c), then next step is to splay this node to the root. In this example, we perform a zig operation on 50 and the result is the tree in frame (d). Next we remove node 50 from the tree, resulting in two trees as shown in frame (e). The next step is to locate the largest value in the left tree, frame (f). Next we splay this node to the root with a zag operation and the result is shown in frame (g). The final step is to connect the two trees by making the root of the right subtree become the right child of the root of the left subtree, resulting in the tree in frame (f).

The reason for omitting detailed feedback from the initial student version was so that we could obtain a baseline for comparing Sketchmate to paper and pencil. Once we performed the experiment to obtain the baseline information, we then added additional functionality that provided detailed feedback to the student when the student performed an invalid operation.

The program begins by loading a pre-made exercise file. Each exercise entry contains the tree structure and the operation to be performed. Sketchmate loads the entire file and then displays the questions and their associated trees, one at a time, to the user. The user's start time is logged in an external data file. All user actions, such as button clicks and mouse drags, are then logged. This information is used not only for Sketchmate development purposes, but also to provide instructors with information about how students are thinking about the algorithm's processes and the amount of time students are spending on each problem.

When the student submits their problem for grading, the external data file stores the user's final tree configuration, the correct tree configuration, and the user's score for that particular problem. When the user finishes all problems, a cumulative score for the exercise and the exit time is recorded at the end of the file.

A description of the two different problem types follows. We initially describe the initial student version with minimal feedback and then describe the richer forms of feedback that were later added.

4.2.1 Demonstrating Splay Operations

The six splay operations (zig, zig-zag, etc) are the fundamental actions performed by the main operations (find, insert, delete) to rebalance the splay tree. These splay operations consist of making and breaking existing connections within the tree itself. Therefore we attempted to give students full control over making and breaking these connections in the program.

Sketchmate presents users with a toolbar that is very similar for both types of problems. Specifically, the toolbar displays the current question and a "Submit Answer" button. Additionally, the Part I toolbar contains "Undo" and "Start Over" buttons in case the user makes a mistake. The Undo button is used to undo the previous action, whereas the Start Over button will reset the tree to its original configuration at the start of the problem.

We split the main program window in half (see Figure 4.8), with the left half containing the original (Before) tree and a set of directions for how to attach and detach nodes. Tree nodes are drawn as circles, with the values displayed inside the circles. Nodes are arranged as typically shown in textbooks with the root at the top and lines showing the relationship between parent and child nodes. We chose to highlight the node of interest for the particular question by shading its interior. In our initial Sketchmate prototype, this pane contained only the directions, and an initial tree was not shown. However, during initial tests of the program, we discovered users would get confused after a few changes to their trees, because they had lost their original context, and would forget exactly what they wanted their trees to look like, even when they initially had a pretty clear vision of what it should look like. The addition of the "Before" tree alleviated this problem.

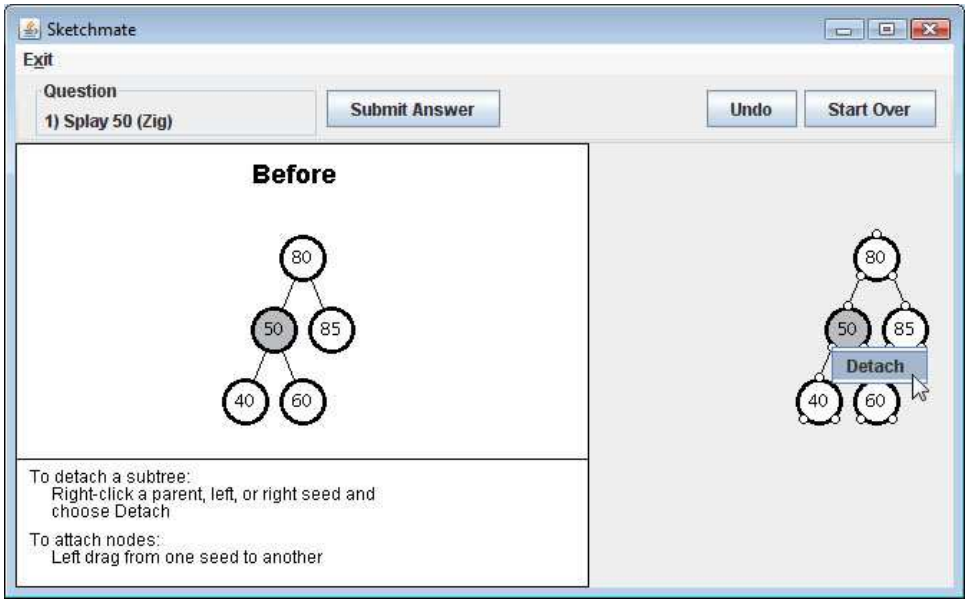

Figure 4.8: Sketchmate screenshot for a splay operation exercise where students are asked to demonstrate one of six basic splay operations. The figure shows the Detach shortcut menu when the user right clicks on a seed.

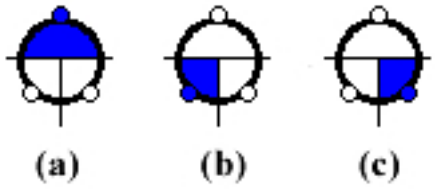

Figure 4.9: The image shows how the node is split into regions for determining which seed is selected. Pane (a) shows the area that selects the parent seed, (b) shows the area for the left child, and (c) shows the area for the right child.

The right half of the main window contains an image of the tree to be manipulated, with values displayed inside circles and lines depicting the relationships. However, we also included smaller circles we call "seeds" at the connection points for the parent, left child, and right child points. These seeds are the locations where users will indicate exactly what action they want to perform to manipulate the tree.

User testing showed that it was very natural to connect nodes by using the mouse to left drag from one seed to another. Users begin their drag motion inside one of the seeds and as they drag, a line is drawn to give them feedback about what nodes they are about to connect. Nodes are programmatically split into three regions so that when the pointer comes over a region, the appropriate seed is highlighted (see Figure 4.9). We found that highlighting the seed gave users a better perspective on the connection they were about to make when they release the mouse button, and that splitting the nodes into regions gave the user a bigger target area for the mouse.

Should the user release in an area that is not part of a node (i.e., outside any of the node circles), the line disappears and no connection is made. However, when the user does release the mouse button over a valid area, the line immediately makes a connection between the initial and final seeds. Also the original line between the new child and its old parent is broken and that line disappears. Once the lines are appropriately displayed, an animation begins which brings the new child and its subtree to the logical place in the image. For example in Figure 4.8, suppose we attempt to make 40 become the right child of 85. The original connection between 40 and 50 is broken and a new line will connect 40 and the right seed of 85. Sketchmate will then move node 40 to the right so that it is displayed as a right child of 85.

If there was already a child in place at that position before the connection was made, then the original connection is broken and the new orphaned subtree is moved down toward the bottom of the screen, so as to be out of the way of the rest of the tree. For example in Figure 4.8, suppose we attempt to make node 40 become the right child of 80, which already has a right child of 85. Then the original connection between 80 and 85 is broken and the newly orphaned subtree, rooted at 85, is moved down toward the bottom of the screen.

Additionally, users are able to disconnect a subtree on their own through the use of the seeds. That is, users can right click any seed and a shortcut menu is displayed for "Detach". As with the automatic detaching, choosing to detach a subtree will remove the

line connecting the subtree to its parent and then move the resulting orphaned subtree toward the bottom of the screen.

The Undo button is used to revert the tree to its state before the just-completed action. The Start Over button reverts the tree to its original state. Neither of these actions take place with animation; instead the tree is simply redrawn. Users signify that they are finished with this problem and ready for the next by clicking the "Submit Answer" button.

One useful feature of Sketchmate is its ability to automatically grade student submissions. Because we wanted Sketchmate to come as close as possible to mimicking how students would normally work with pencil and paper, we did not require any specific sequence of steps when they are manipulating the trees. For example, users are free to detach every single node from the tree and reassemble it node by node if they choose to do so. Unfortunately, this also means that there is no way for Sketchmate to award partial credit for a splay operation. Therefore each problem is graded by matching the user's resulting tree with the tree that would result if the operation were performed correctly. By having Sketchmate do this matching automatically, it saves instructors or teaching assistants the work of checking each tree node by node. This process is not only tedious for humans, but it is error-prone as well.

4.2.2 Demonstrating Find, Insert, and Delete Operations

The second type of problem that Sketchmate can ask students to perform is the find, insert, and delete splay-tree operations, which build upon the fundamental splay operations. The instructional idea is that the instructor initially gives students an opportunity to practice the fundamental splay operations with the first type of problem (i.e., splay problems), and the students are now in a position where they can simply call the appropriate splay operation and have the program do it for them. For this reason, six buttons (Zig, Zag, Zig-Zig, Zag-Zag, Zig-Zag, and Zag-Zig) are added to the toolbar for the second problem set (see Figure 4.10).

Once again, the main window is split in half, with the left half containing the directions appropriate for the particular operation. We found that it was not necessary to include the original tree structure for these problems, possibly because users are not asked to manually break the tree into multiple subtrees, and because each subsequent operation depends solely on the result of the previous operation.

The right half of the main window contains the tree to be manipulated. This half of the window allows users to select a node that they are interested in splaying. When a node is selected, its boundary color is changed from black to blue to show that it has been selected. Once selected, the user is able to click the button associated with the appropriate splay operation. When the button is pressed, the animation is performed and the tree is transformed appropriately. If the user attempts any splay operation other than the correct one, a dialog box is displayed telling the user that the action is incorrect. This is done simply because each splay operation is only possible on a specific arrangement of nodes. For example, when the left child of a node is selected, it is not possible to perform a zag splay operation because the zag requires the selected node to be a right child.

As expected, the insert and delete operations require more steps and therefore require more directions and some differences in the interface. Specifically, the insert operation begins with the user inserting the new node into the tree using the insertion rules for an

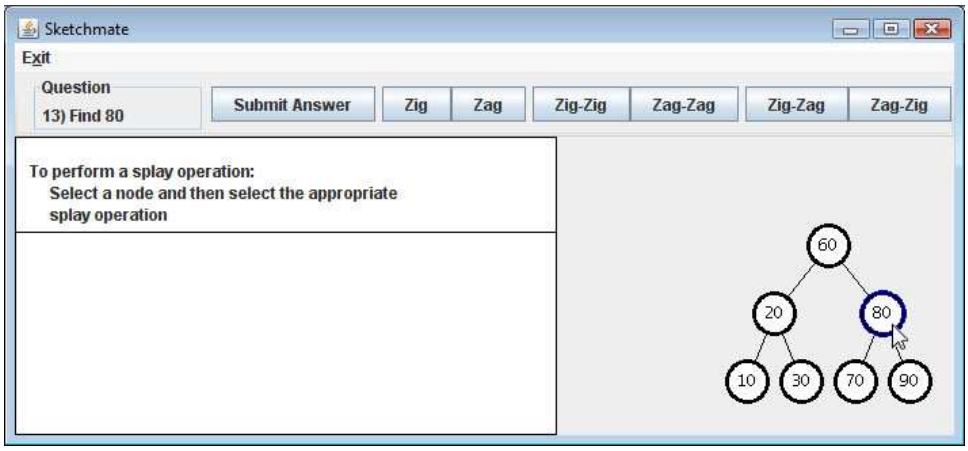

Figure 4.10: Sketchmate screenshot for a Part II exercise where students are asked to demonstrate either a find, insert, or delete operation. The figure shows node 80 being selected for splaying.

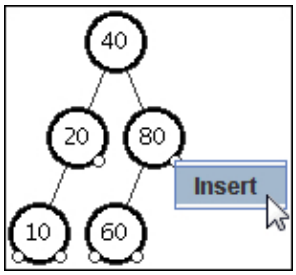

Figure 4.11: Screen capture image of the manipulation pane showing the available seeds when inserting a node. Users will right click on a seed and choose Insert from the shortcut menu.

ordinary, unbalanced binary search tree. We once again used the idea of using seeds to make connections, and displayed seeds at every empty child link (see Figure 4.11). The user right clicks the seed and chooses "Insert" from the resulting shortcut menu. Sketchmate responds by creating a new node with the appropriate value, placing it in the appropriate location in the figure, and adding a line connecting the new node to its parent. Once again if the user chooses to insert the node at the wrong location, a dialog box appears telling the user that this insertion is not correct. Once the node has been created, the user splays the new node to the root using the appropriate splay buttons in the toolbar.

When performing the delete operation, the user first splays the node to be deleted to the root and then deletes the node. We allow users to delete a node by right clicking and choosing delete from the shortcut menu (see Figure 4.12). Deleting a node will remove the circle and the two lines to its children, thus possibly creating two subtrees. Users are only allowed to delete the node specified by the problem and that node must be at the root before it can be removed. Doing anything else results in an incorrect action dialog.

The next step in the delete operation is to splay the largest value in the left subtree to the root of that tree. After that happens, users must connect the two subtrees by making the root of the right subtree become the right child of the root of the left subtree. In order to allow the student to make this connection, Sketchmate displays the seeds as soon as the largest value in the left subtree has been splayed to the root. Now users must left drag to make the appropriate connection, just as they did when demonstrating the different splay operations in the Part I exercises. Once the correct connection is made, the nodes are animatedly moved into their appropriate locations based on their new relationships.

Once again, we leverage the power of the computer so that Sketchmate can grade the students' work on these problems. This time we can award partial credit because there are a specific sequence of operations that take place for each of the find, insert, and delete operations. Obviously the user who performs every step of the operation correctly receives full credit, which we arbitrarily set at 10 points. However, if the user makes a mistake along the way (e.g., choosing a zig operation, when they should have chosen a zag operation), Sketchmate deducts 1 point for each incorrect action. Therefore a student who makes 3 mistakes, but completes all steps in the operation would receive 7/10.

When we tested this grading scheme, we quickly realized that users would be able to game the system by loading a problem and clicking the submit button. Because the user

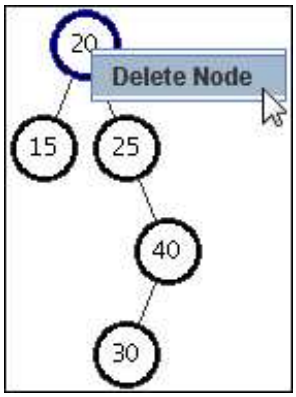

Figure 4.12: Screen capture image of the manipulation pane showing the shortcut menu for deleting a node.

performed no incorrect actions, they would receive full credit. To account for this, we changed the grading system to detect whether or not the user completed the problem. If they did not complete every step in the operation, we awarded 1 point based on each correct action. Under this new scheme, users who clicked the submit button right away received no credit. However if they began the problem and quit in the middle, they would receive some credit. For example, a user who performed the first step in the insert operation by placing the node in the correct position in the tree, but then clicked the submit button, would only receive 1/10.

4.2.3 Feedback

Another unique component of Sketchmate is its ability to provide immediate, detailed feedback when users perform an incorrect action. By providing detailed, corrective feedback as soon as the user attempts an action, we hoped to increase student learning.

While designing Sketchmate, we realized that user errors could be grouped into distinct classifications. The methods and classifications are discussed next.

Feedback methods

When designing the method for providing user feedback when an error occurred, we again wanted to keep things simple and not unduly interfere with the user's work. The first version of the student Sketchmate simply displayed a dialog box explaining that the user attempted an incorrect action. Although we could have continued to pop up a dialog box in order to give the detailed feedback, we found that it became intrusive. That is, users were able to view the feedback details, but then were required to close that dialog in order to proceed and attempt another action. This required users to carefully read and attempt to absorb the information before attempting to apply this knowledge to the current problem.

To overcome this issue, Sketchmate instead displays the feedback information in the left pane, below the area displaying the instructions for using Sketchmate (Figure 4.13). This left pane is split and a highlighted heading with the words "Incorrect Action" are shown,

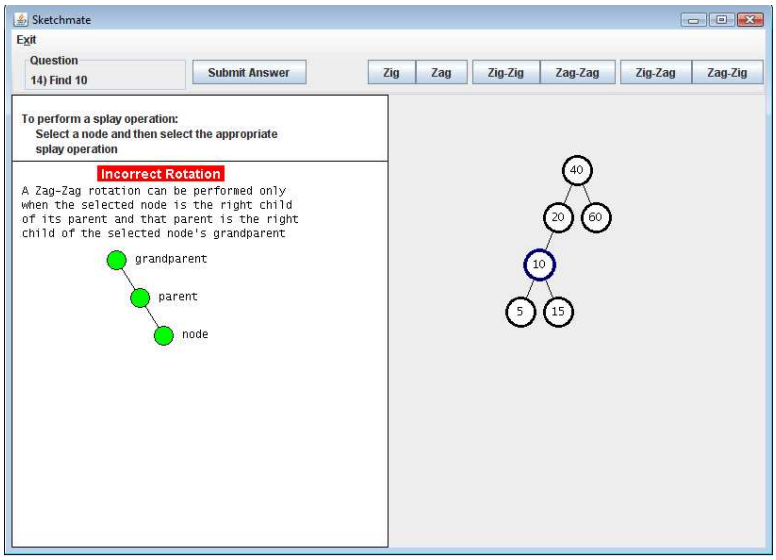

Figure 4.13: Screen capture image of Sketchmate showing detailed feedback in the lower left pane when the user attempted to perform a zag-zag operation. The correct operation was a zig-zig operation.

followed by additional details. We found that this area was large enough to accommodate the feedback information and we could continue to display this information while the user was working.

By contrast, Sketchmate's feedback when the user attempts to click the submit button is displayed in a pop-up dialog box because this is a permanent operation. When users choose to submit their answer anyway, Sketchmate closes the dialog, but then presents the user with the set of steps necessary for the correct solution to the problem.

Basic splay operation feedback

Sketchmate is designed so that users demonstrating a basic splay operation (zig, zag-zig, etc) can do so by manipulating the tree in any way they like. For example, users can choose to detach every node from the tree and then reassemble the tree node by node. And while this allows the user to more naturally interact with the tree as they would with paper and pencil, it does not allow for an appropriate method of feedback for individual steps because users can perform the steps in any order they choose.

For this reason, the splay operation feedback is provided when the user submits their tree for grading. At that point, the user's tree is compared with the correct tree. If differences occur, then animated feedback is provided. Once again Sketchmate takes advantage of the left pane. The "before" image is replaced with a set of text instructions explaining a set of steps that will correctly perform the requested operation (Figure 4.14). The text description begins with a heading describing the splay operation that is supposed to be performed. Underneath this heading Sketchmate provides a numbered set of steps. Text

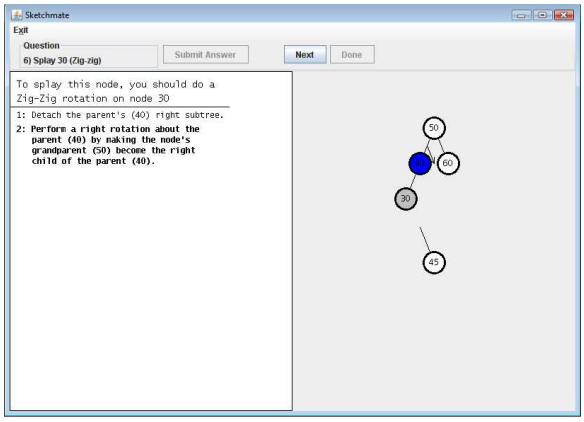

Figure 4.14: Screen capture image of Sketchmate showing detailed feedback for a basic splay operation. The text description of the steps necessary is shown in the left pane, while the image is shown in the right pane. The user controls the animation steps using the "Next" and "Done" buttons in the toolbar.

for the current step is displayed in a bold font in order to distinguish it from earlier steps that have completed.

As each step is highlighted, the right pane displays an appropriate image just before the action in that step is executed. In this way we give the users an explanation of what is about to take place. Arrows are added to the image to help direct the users' attention to a particular area of the tree.

When the users are ready to view the step in action, they click the "Next" button in the toolbar. This will set Sketchmate into motion and an animation, similar to those provided in the instructor's tool, is displayed. Upon completion of the splay operation, Sketchmate disables the "Next" button and enables the "Done" button, which emphasizes that the operation is complete. When users click the "Done" button, both the left and right panes are cleared, the buttons are removed from the toolbar and the next problem is displayed.

Early submission

When working on a find, insert, or delete operation problem, students may either intentionally or inadvertently click the "Submit Answer" button. When this occurs, a dialog box appears telling the user that the current operation is not yet complete. In addition, the dialog presents a textual description (Figure 4.15) of the next step in the operation and allows the user to choose to either continue working on this problem, or submit it.

We chose the specific verbiage in this message, "Please try to finish this problem" and inside the buttons, "Submit Anyway" and "I will keep trying", as a way to encourage our users to keep working.

Incorrect node selection

Sketchmate allows users to attempt a splay operation on any tree node, whether it is the appropriate node to splay or not. When users select an incorrect node, no feedback is given

Figure 4.15: Dialog box resulting from the user choosing to prematurely submit their answer when attempting to demonstrate an insert operation. The dialog explains that the answer is not complete and provides more details about the next step. Users can then choose to click a button to either submit the answer anyway or continue working on the problem.

until they click one of the splay operation buttons (zig, zag-zig, etc). At that time, the feedback area in the lower left pane tells the user exactly which node should have been selected (Figure 4.16).

Incorrect splay operation

If the user does select the correct node for splaying, yet chooses an incorrect splay operation the feedback area does not tell the user the correct splay operation, but does provide the details about the operation they selected. Figure 4.17 shows the feedback when the user attempts to do a zag operation where a zig operation is required. The feedback not only gives a textual description of the operation the user selected, but also a figure using a generic tree for the basic arrangement of nodes that would be required to make the chosen operation the correct operation.

Attemping to rotate before inserting

When inserting a new value into a splay tree, the first step is to place the new value in the appropriate place in the existing tree. However Sketchmate will allow users to attempt a splay operation prior to doing the insertion. When that occurs, the feedback area displays a message explaining that the first step in the operation must be an insertion(Figure 4.18).

Incorrect insert location

As previously explained, the first step in an insert operation is to place the new value in the tree. This insertion must be done in a way as to preserve the ordered property of a binary search tree. The ordered property states that at any node in the tree, the value stored at the current node is greater than all values in the left subtree and less than all values in the right subtree. Because of this property, there is exactly one location where the new value may be placed (assuming there are no duplicates in the tree). If the user chooses to insert the value at a different location, the feedback area displays a message explaining that the properties of a binary search tree must be maintained and then explains exactly where the node should be inserted (Figure 4.19).

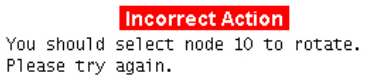

Figure 4.16: Sketchmate feedback when the user selects the incorrect node to splay. Text explains that the incorrect node was selected and then gives the correct node.

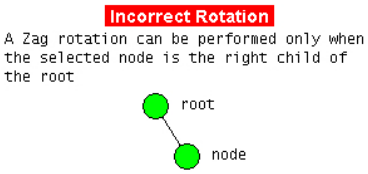

Figure 4.17: Sketchmate feedback when the user selects the incorrect splay operation. Feedback shown is the result of a user selecting a zag operation, where a zig operation was appropriate.

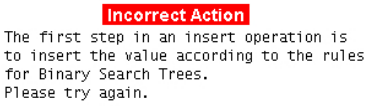

Figure 4.18: Sketchmate feedback when the user attempts a splay operation before inserting the value into the tree.

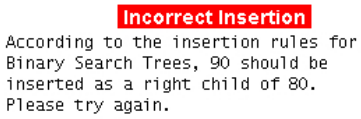

Figure 4.19: Sketchmate feedback when the user attempts to insert a new value into the tree in an incorrect location. Feedback explains that the insert location is dependent upon binary search tree properties and then gives a text description of this location in the current tree.

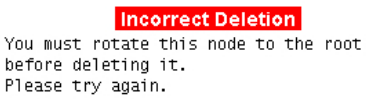

Figure 4.20: Sketchmate feedback when the user attempts to delete the correct node from the tree before splaying it to the root. Feedback text explains that it must be first splayed to the root.

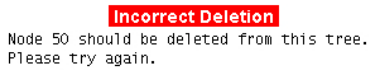

Figure 4.21: Sketchmate feedback when the user attempts to delete the wrong node from the tree. In this example, the user was supposed to delete the node with value 50, but attempted to delete a different node.

Deleting node before splaying

The first step in a delete operation is to splay the node to be deleted to the tree's root. However Sketchmate will allow users to attempt to delete the node at any time. Therefore if the user attempts to delete the node before splaying it to the root, the feedback area displays a message explaining that the node must be first splayed to the root (Figure 4.20).

Deleting the incorrect node

Sketchmate allows users to attempt to delete any node they choose. For that reason, if the user chooses to delete the incorrect node, the feedback area displays text explaining which node should be deleted (Figure 4.21).

Incorrect node to be splayed after removing a node in the delete operation

During a delete operation, after the node to be deleted is splayed to the root of the tree and then removed, the result is two separate splay trees. The next step in the operation is to splay the largest value in the left tree to the root. This action requires the user to not only select the correct node to be splayed, but also to choose the correct splay operation. If the user selects the correct node, but chooses an incorrect splay operation, the feedback delivered is the same as any other time an incorrect splay operation is chosen (see section 4.2.3).

However, if the incorrect node is selected at this time, a different message is given. Feedback for this error explains to the user the rule that says the next step is to splay the largest value in the left tree to the root. Additionally, the message includes the value of that node (Figure 4.22).

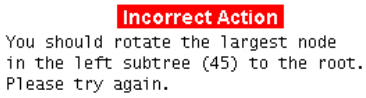

Figure 4.22: Sketchmate feedback when the user attempts to splay any node other than the one containing the largest value in the left subtree. Feedback explains the rule for the next step and gives the value of that node.

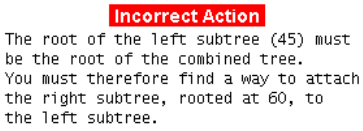

Figure 4.23: Sketchmate feedback when the user attempts to connect the two trees using incorrect seeds during the final step of a delete operation. Feedback explains that the root of the right tree needs to be made the right child of the root of the left tree.

Incorrectly Re-uniting the Two Subtrees in a Delete Operation

After the largest value in the left tree is splayed to the root, the two trees must be connected. This is done in Sketchmate by having the user drag to make a connection between the right child seed of the left tree's root and the parent seed of the right tree's root. Note that the user can start at either seed, meaning they can drag from the right child seed to the parent seed or vice versa. However in order to present the problem fairly, Sketchmate displays seeds at all unconnected seed locations for every node. The result can be that users choose to make a connection between incorrect nodes. When this occurs, the feedback area displays a message explaining which nodes are to be used to connect the two trees (Figure 4.23).

4.3 Design Justifications

We developed Sketchmate from scratch in an incremental fashion. We began with ideas from other programs and attempted to build upon them. However, with each step we evaluated the design decisions both formally and informally. When certain design decisions were not appropriate for one reason or another, Sketchmate was changed. Our main focus was to create a program that was as intuitive as possible so that users would not need to invest a large amount of time in order to use it. This section explains these design choices and provides an explanation about them.

4.3.1 The Instructor's Tool

One of the drawbacks to many existing pedagogical applications is the amount of time necessary to learn how to use them. Since instructors are busy people, a steep learning

curve can be a serious impediment to the adoption of a tool. Consequently, many of our design choices were aimed at keeping the interface as intuitive as possible so that users would be able to quickly and easily use Sketchmate.

Tree creation with 'type and tab'

Sketchmate's ability to quickly draw trees that are illustrated in textbooks is unique from other existing tree animation programs. However, the ultimate way of providing this functionality differs from the seemingly natural way of drawing a tree. Initially we considered allowing a user to sketch a tree by dropping nodes onto the drawing canvas and then connecting the nodes with lines. We also experimented with allowing users to simultaneously create a node and its connection by dragging from a pre-existing seed. These techniques mimic the drawing techniques that Beeler discovered that users employ when drawing trees using paper-and-pencil ([Beeler, 2007]). However this technique is not as efficient on a computer because it requires users to employ the mouse to create the shape of the tree and then employ the keyboard to type out the desired values. In order to enter the appropriate values, the user must either use the mouse to select each node and then type an appropriate string, or else tab from node to node and enter a string as each node is selected. The first method is inefficient because the user must keep iterating between using the mouse and the keyboard, which involves a continual repositioning of the hands. The tabbing method is much more efficient, and gave us the idea that if nodes could be created by tabbing as well, then the user would never have to reposition their hands while creating the tree.

We experimented with different node layouts. We initially presented the nodes side by side in a line. This design is similar to how the values would be stored in an array with the root value stored in element 0, the root's left child in element 1, the root's right child in element 2, etc. We quickly realized that this layout was helpful if the instructor was discussing how to implement a tree with an array. However, we also found that most instructors preferred to sketch out the nodes as they would appear in a traditional binary tree with the root centered at the top of the tree and its left child below and to the left and the right child below and to the right, etc.

Based on these experiences, we settled on the level-order scheme described in Section 4.1.1. While we never performed a user study to find out how much faster it was to create a tree using the tabbing technique as opposed to the sketching and typing technique, it was obvious from informal testing that users could create trees significantly faster using the tabbing technique.

Node layout algorithm

Initially the layout of the tree's nodes seemed trivial. We would simply draw child nodes below the parent. Left children would be drawn to the left and right children to the right by a fixed amount of space. This algorithm works fine when creating complete trees (trees that have all nodes present when traversing in level order, Figure 4.24) or full trees (trees where every node has either two children or no children, Figure 4.25). However, this algorithm ends up wasting a lot of excess space for trees with only a few nodes that are spread across the tree as seen in Figure 4.26 because each node must be spaced as if it were complete, even though there are no nodes present. It is also possible to get a more

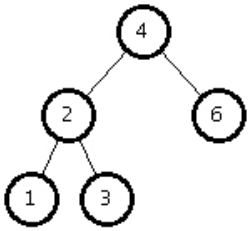

Figure 4.24: A complete binary tree where every node is present when the tree is traversed in level-order.

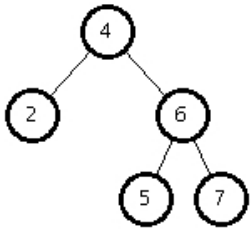

Figure 4.25: A full binary tree where every node has either two children or no children.

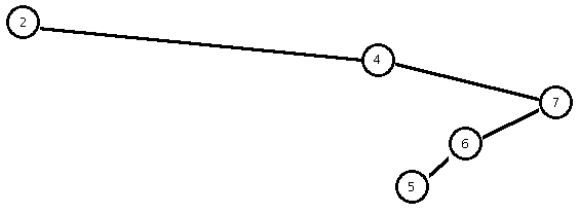

Figure 4.26: Resulting tree when employing the simple layout algorithm for a sparsely filled tree. Note that the extra space is the result of the algorithm accounting for nodes which are not present.

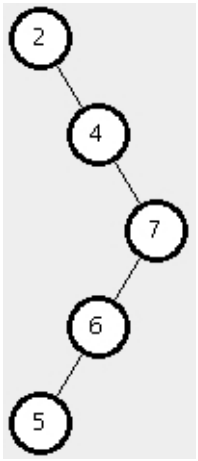

Figure 4.27: The same tree as Figure 4.26 when using the layout algorithm employed by Sketchmate.

compact representation if we throw away the assumption of completeness, but still not allow nodes in separate subtrees to overlap in the x-direction (i.e., all the nodes in a right subtree will have an x-coordinate that is strictly greater than the x-coordinate of any node in the left subtree). It turns out that this representation is still rather wasteful of space and can present problems in displaying trees more than a few levels deep without resorting to scrolling.

We searched for a tree layout algorithm that would allow nodes to overlap horizontally as long as they did not also overlap vertically, but could not find any published algorithms that would allow such overlapping. We therefore settled on the layout algorithm described below. For full and complete trees it creates images similar to the ones produced by the algorithm described above, but for sparse trees it produces a more compact and pleasing result (Figure 4.27).

The layout algorithm is split into two different dimensions: x and y. The nodes are processed in level order, one level at a time. The y layout is easy as it is purely based on the node's level within the tree. Therefore, the root (at level 0) is placed with the smallest value for y. The root's children (level 1) are then placed below the root (a higher y value) with the formula:

$$Y = V\_SPACING + (node\ height + V\_SPACING) * level\ number$$

Note that the value of V_SPACING is just an arbitrary constant buffer value so that nodes have a little space between them.

The x layout is more complex. In this case, we traverse the tree level by level starting at the lowest level. Note that all of these nodes are leaf nodes. The algorithm goes as follows:

1. Start with the lowest level and build a list of nodes on this level.
   (a) Remove the first node from the list and assign it an x position of 0.
   (b) Remove the next node from the list. This node will be placed at the x value:

X = previous node's x + node width + H_SPACING

(c) Continue with step b until all the nodes in that level have been placed

2. Continue with the next highest level until you reach the root:

(a) Build a list of nodes for that level.

(b) Remove the first node in the list. The x location of this node will be determined based on its children:

Nodes with 2 children:

X = (left child's x + right child's x) / 2

Nodes with only a left child:

X = left child's x + (node width/2 + H_SPACING / 2)

Nodes with only a right child:

X = right child's x – (node width/2 + H_SPACING / 2)

Nodes with no children:

X = previous node's x + H_SPACING

(c) Check to see if this node's x value causes it to overlap with the previous node in this level. Do this by calculating shift as:

Shift = current node's x – H_SPACING – node width – previous node's x

(d) If Shift < 0 then there is an overlap:

To fix the overlap:

i. Move the current node by (-Shift) to the right (i.e., node x = node x – Shift).

ii. Additionally, because we moved this node to the right, we also must move this node's descendants and the neighbors to the right of the descendants to the right by the same amount to maintain their positions relative to the newly moved node. For example, if the node I am moving has a left child and a right child, then moving the current node to the right will mean that now the right child is directly beneath the newly moved node, instead of slightly to the right as it was before.

Thus to keep this relative layout, we will:

A. Get a list of nodes on the next lower level

B. Shift nodes that are either a descendent or a node to the right of a descendant by (–Shift) in the x direction (to the right)

Note: This means that we are always shifting nodes to the right and the only possible new overlap that may occur will happen on the current level and that will be fixed as we visit the nodes in this level from left to right. Figure 4.28 and Figure 4.29 provide examples and explanations of how the overlap comes about and how each is fixed.

The toolbar

When we designed the toolbar at the top of the Sketchmate window, we wanted to display only the essential controls. Therefore, in the animate pane we added three buttons, one

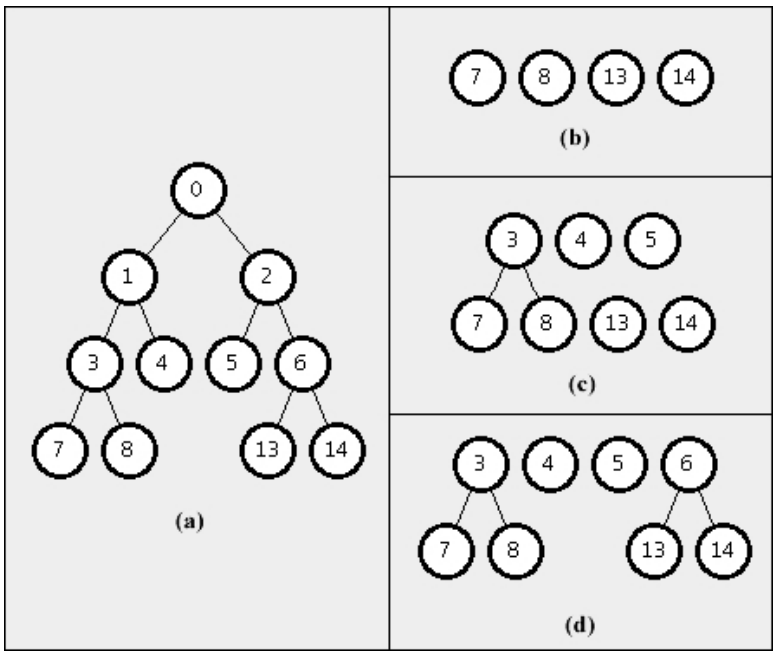

Figure 4.28: Frame (a) shows the tree that is being laid out. According to the algorithm, the last level in the tree (values 7, 8, 13, and 14) are laid out first based on step 1 of the algorithm. The result is shown in frame (b). Having completed that level, we next lay out the third level (values 3, 4, 5, and 6) using step 2 of the algorithm. Frame (c) shows the tree after the first three values (3, 4, and 5) have been laid out without conflict. Node 3's position is based upon the positions of its children. Nodes 4 and 5 have no children and therefore are positioned based on the node before them (3 and 4, respectively). There are no overlaps at this point. However, when attempting to lay out value 6, there will be an overlap because 6's position must be based on its children, 13 and 14. You can see from frame (c) that node 6 would need to be placed exactly where node 5 is currently located. Therefore we fix the overlap with step 2d and shift node 6 and all of its descendents to the right. After applying this step, we have successfully laid out the bottom two levels. The remaining nodes in this tree (0, 1, and 2) can be laid out based on the positions of their respective children and no overlaps occur.

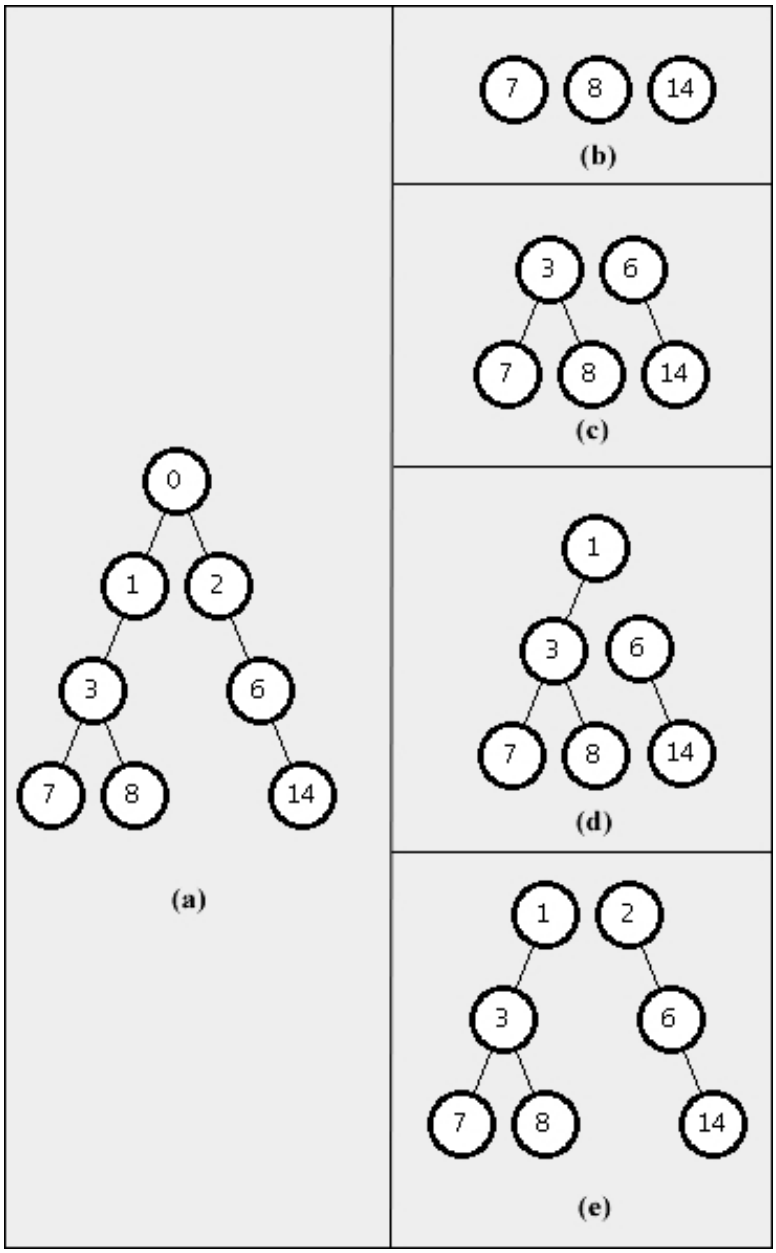

Figure 4.29: Frame (a) shows the tree that is being laid out. According to the algorithm, the last level in the tree (values 7, 8, and 14) are laid out first based on step 1 of the algorithm. The result is shown in frame (b). Having completed that level, we next lay out the third level (values 3 and 6) using step 2 of the algorithm where each node's position is based on the position of its children. Next we lay out the level immediately above this one (values 1 and 2). Node 1's position is based on its child, node 3 as shown in frame (d). However now we must lay out node 2 based on its child, node 6. At this point the overlap occurs because node 2 would need to be placed directly on top of node 1. Therefore we work through step 2d of the algorithm and shift node 2 and all of its descendents to the right, resulting in frame (e). The final step is to position node 0 based on its children, nodes 1 and 2.

for each operation (find, insert, and delete). Additionally, we placed a single text field to be used for inserting a value into the tree. After some initial tests, we found that users also wanted a way to clear the tree and start over. For that, we placed a clear button that was separated from the other buttons so that accidental clicks would be minimized.

While these buttons kept the interface simple and easy to use, we needed a new set of controls for the animations: next step and cancel. We initially considered placing these animation buttons in the toolbar with the operation buttons and simply enabling the appropriate buttons when necessary (the user cannot start a new operation until the current animation is complete so the operation buttons need to be disabled). However, we found that displaying both sets of buttons cluttered the toolbar and led to confusion for the user as to how the buttons were to be used. Instead, when the animation begins, we chose to replace the operation buttons with a set of animation buttons. Now only relevant controls were present based on what the user is currently doing.

In addition to the next animation button, we found it useful to place a "done" button in the toolbar. This button is initially disabled, but becomes enabled when the animation is complete. Before adding this button, we found that users were somewhat disoriented in that they did not know exactly when the animation was complete. We did have the operation buttons replace the animation buttons; however this replacement did not seem to be enough of a transition to notify users. We also found that users would get in the habit of clicking the next button over and over, and mistakenly click an operation button by accident when the animation was complete. By placing the done button in the toolbar, we now forced users to stop mindlessly clicking next and move their mouse to click done, which in turn gave them a better reference that the animation was complete.

Beautifying the tree

Sometimes when performing a sequence of operations on a tree, we found that the layout became visually displeasing because the tree continued to use spacing that was initially appropriate, but after the animation was not. For example, the layout algorithm depends on the number of nodes in that particular level. Therefore, the spacing for a level with 8 nodes is different than the spacing for a level with 3 nodes. When performing a splay operation that moves nodes and their children from one level to another, the spacing will sometimes change. This results in uneven spacing between nodes of the same level and makes for a displeasing node configuration. This change in spacing can also lead to nodes overlapping one another, thus making parts of the tree invisible.

In order to get around this problem, when each animation is complete, the layout algorithm is run and the correct placement of the nodes is calculated. If any of the nodes should be outside a small tolerance (5 pixels), then a subroutine runs which moves the nodes to their calculated positions. Additionally, because we did not want to surprise our users at the end of the animation by having the nodes instantaneously appear in new locations, this beautification is animated over the course of about 1 second.

Reverting to a previous tree configuration

One feature lacking in existing programs is the ability to undo previous operations and return to an earlier tree configuration. As noted, this is especially helpful to instructors

who are asked to repeat their explanation during lecture. Our problem however was how to distinguish the undo of an animation step, with the undo of an operation. We solved this by the creation of the revert pane. Now users who wanted to view a previous tree state could switch to a different tabbed pane and see a list of the operations performed. Many applications simply present a textual list of commands that can be undone. However, we found that because this list did not take up very much room in our window, we were able to split it and show the resulting tree after the operation was performed. Displaying the tree gives the user both a text-based cue and an image to determine how far back they want to step.

The list of steps is presented as a series of mutually exclusive radio buttons. The user can select the step they want to revert to and then click an OK button. Doing so will select that tree and immediately move them to the animate pane for further manipulation. While the revert pane was initially meant solely for undo operations, we found that a beneficial side-effect of adding the tree images was that an instructor could use the radio buttons to flip through the states of a tree after a series of operations had been completed, thus illustrating to students how the tree evolved over a number of operations.

Animation steps

When designing the animation, we spent a considerable amount of time attempting to create movements that were both pleasing to watch and easy to understand. Initially Sketchmate was designed to show the entire splay operation as a single animation, which is the way all existing tree animation applications perform. However we found that doing so resulted in multiple nodes being moved at the same time, which caused confusion as to where the user should direct their attention (multiple moving objects cause the user's attention to flip back and forth between objects, thus creating continual context shifts, which tends to be disorienting).

To alleviate this situation, we broke each movement into its own separate step. We also inserted a pause in the animation after each step so that the user could choose how much time they wanted to spend explaining the step. This becomes most important when the splay operation calls for a child node to be detached from the current node prior to the rotation. This can be seen in the zig rotation shown in Figure 4.30, where the left child (40) of the current node (3) is detached. By stopping after the detachment of the child, instructors can pause to describe what just happened, or simply continue at their own pace.

4.3.2 The Student's Tool

When designing the student's tool, we wanted to continue the theme of creating an intuitive, simple, and easy to use interface so that users would be able to use the program as quickly as possible. While we did use the instructor tool's animation functionality when appropriate, the bulk of the program required a new approach. This new approach was required because students would be using the tool as a way of responding to splay tree questions as opposed to using it as a supplement to lecture.

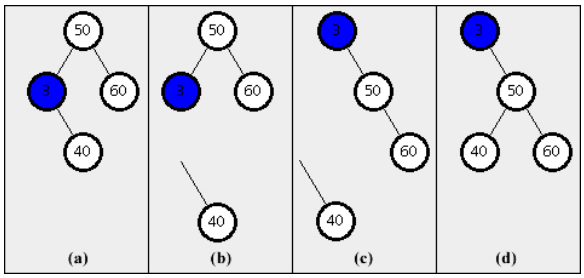

Figure 4.30: Screen shots at the end of each step of a zig operation. Frame (a) shows the initial tree. Frame (b) shows the end of the first animation step where the left child (40) is detached from the tree. Frame (c) shows the result after the rotation. Frame (d) shows the final configuration of the tree once the detached child has been reattached in its new position.

The toolbar

As in the instructor's tool, we attempted to keep the interface simple by placing as little on the screen as possible at any given time. This is evident in Sketchmate's toolbar, which consists of (1) the question, and (2) a submit button. Additionally we found that users working on the basic splay operations would sometimes desire to undo a previous action, and therefore we added buttons for undo and start over.

When working on a find, insert, or delete operation, users required some way to select which splay operation they wanted to be performed. We experimented with placing the six splay buttons inside the large frame along with the tree, however with the extra space in the toolbar, it seemed to look cleaner when we put all buttons inside the toolbar. By keeping our toolbar simple, we feel that users have a clearer idea of exactly what is required of them.

The "before" pane

As previously mentioned, we found that when performing the basic splay operations by dragging to connect nodes, it was easy to become disoriented as to what the tree originally looked like and how we planned to proceed with the operation. For that reason, we split the main pane into two areas: one for the original problem, and one to be used to manipulate the tree. By adding this one original image, we helped to better define the problem's perspective for the user when working the problem.

Sketchmate instructions

Another design choice was how to present the Sketchmate instructions. We considered writing out a short user manual, which ended up being a single sheet containing an explanation of all functionality. Unfortunately this meant that users needed to know when certain functions were available. For example, users cannot perform a right-click on a node to delete it when performing a find operation (because no nodes are deleted during this

operation). This separate sheet also meant that the users would require the sheet until they became more comfortable with Sketchmate.

We found that when we added the before image in the splay operation problems, that there was always a little extra room underneath the image. We exploited this area by placing the directions for manipulating the tree there.

The left "before" pane did not originally exist for the find, insert, and delete operations. However, after adding the Sketchmate instructions to the left pane for the splay operation problems, and seeing how successful it was, we decided to do the same thing for the find, insert, and delete operations. For reasons cited earlier, displaying the before tree on the left was not as useful for these types of problems, so we simply used the left pane for instructions relevant for that particular type of problem (i.e., find, insert, delete). This allowed us to totally eliminate the need for a user manual and, based on how quickly students mastered Sketchmate during its experimental evaluation, we suspect it made Sketchmate easier to use.

Manipulating tree node connections

During the design of Sketchmate's interface, we always attempted to link natural user actions to Sketchmate functionality. When it came to deciding how to connect one node to another, a drag and drop operation seemed natural and appropriate. We also ended up adding to each node the three seeds for parent, left child, and right child. This addition had a couple of benefits. First it gave the users a place on the node to focus on when dragging. Second, it helps Sketchmate divine the user's intentions. Initially the nodes were created without seeds. This allowed users to drag between nodes; however we needed some way of knowing which child (left or right) to make the connection. Although we could have prompted the user, or used a combination of key presses with the drag, it made more sense to provide a single contact point for each connection.

Finally, a fast and easy way to disconnect a node was desired. We considered allowing the user to select a connection and delete it using the delete key, but this would require a mouse interaction followed by a keyboard interaction. Since a mouse interaction was required in any event, we settled on a right click on the appropriate seed, followed by a popup menu choice for deletion. This seemed to work well since users were not observed having difficulty deleting connections.

Seed selection

Another interesting finding about the seeds was that we wanted to keep them small, yet they needed to be large enough for the user to click inside them. We found that the seeds presented an uncomfortably small target area and so we ended up allowing a seed to be selected based on a region within the node, as opposed to requiring the pointer to be directly over the seed (see Figure 4.9). Because there was only one seed on top of the node, the entire top half of the node selects the parent seed, whereas the bottom half of the node is split with the left half selecting the left child and the right half selecting the right child. By splitting the node this way, we could now display a smaller seed, yet still provide the user with a large target with which to select the seed.

One final addition to the seeds was the highlighting of the currently selected seed. That is, when the user moves their pointer within one of the seed's regions, that particular seed is highlighted by filling in its circle. Now the user not only has the power of using a larger region for seed selection, but they can also see which seed will be selected as they move the mouse.

Chapter 5

Experimental Evaluation of Sketchmate for the Instructor

To validate Sketchmate's utility to instructors, we conducted two experiments. The first experiment compared Sketchmate with paper and pencil during the preparation of a lecture. Based on the assumption that an instructor would want to prepare some splay tree notes for reference before the actual lecture period, we wanted to get a relative measure of how long it took to draw out a set of splay trees using Sketchmate and paper and pencil. Therefore we made a list of specific splay trees and operations to be performed on them and had the participants draw them out using the different tools. This experiment measured the number of examples completed in a fixed amount of time and the accuracy of those sketches.

The second experiment attempted to give us an idea of how each tool would be used in a classroom environment by comparing Sketchmate with the whiteboard when participants described an example problem. Because often times in class, students raise their hands to ask related questions, we also followed up with incidental questions about the splay tree. We timed all participant responses to give us a measurement of the relative amount of time needed to answer such questions with each tool. In addition to the presentation time, we also measured the accuracy of the participants' sketches.

5.1 Experiment I: Lecture Preparation

In the first experiment, we evaluated Sketchmate relative to paper and pencil in the preparation of splay-tree examples to be used for lecture. The experiment used a within-subjects, post-test only design. We gave participants a list of trees and operations to perform on those trees and asked them to work through the examples using Sketchmate and paper and pencil. The hypotheses were that participants would be able to prepare more examples using Sketchmate and that those examples would be more accurate than those created with paper and pencil. Since all participants worked with both Sketchmate and paper and pencil, we had them rate which one they thought was easier and which they liked using better. The hypothesis here was that participants would prefer Sketchmate.

5.1.1 Subjects, Setting and Materials

After receiving Institutional Review Board (IRB) approval from The University of Tennessee, we recruited 25 students (21 male, 4 female; mean age 23.8) from an upper level undergraduate / graduate course, CS594 - Scripting Languages, in the Electrical Engineering and Computer Science department at The University of Tennessee. Although splay-tree content was not addressed in this course, all participants were familiar with this data structure. Participants received extra credit in the course for participating.

The experiment took place using the computers in a Computer Science lab on campus. All machines had a Pentium 4 processor with 1 GB of RAM and were running the Debian operating system. Participants were provided a copy of the Sketchmate program.

Additionally, experimenters developed one set of 18 specific splay-tree tasks. Each of the 18 splay-tree tasks consisted of first copying the picture of the tree from the exercise and then performing 3 or 4 individual operations chosen from the operations insert, delete, or find. Although this study could have been run with a series of operations on the same splay tree (e.g., 1 splay tree with 60 sequential operations), operations within each task were limited to 3 - 4 so that participant errors would not be propagated for more than a few operations. These procedures allowed us to preserve the integrity of our experiment as participants were more likely to be working with the trees we intended them to work with, as opposed to trees they may have developed following numerous sequential errors. Grouping our operations in this fashion also assisted in scoring the accuracy because scoring on the operations within a task is similar to grading a multi-step math problem where individual operations can be marked as correct if they are performed correctly in light of previous errors.

5.1.2 Design and Analysis

A within-subjects, post-test only design was used to test for differences on accuracy and efficiency of splay-tree operations prepared using Sketchmate (experimental condition) and paper and pencil (control condition). To control for sequence effects, condition order was counterbalanced across participants. The dependent variables included the number of operations completed in 15 minutes and the percentage of completed operations that were correct. For each participant the final operation was not scored for accuracy unless it was completed. Paired samples t-tests were used to test for significant differences across dependent variables. All differences were considered significant at the $\rho < 0.05$ level.

5.1.3 Procedures

Participants met in a lab classroom and were given a brief, 10-minute overview of splay trees. For those participants working first with Sketchmate, the experimenter used Sketchmate to present the overview. For those working with paper and pencil first, the experimenter used the whiteboard to present the overview. This initial review covered basic single and double splay operations in a splay tree and how these splay operations are used in insert, delete and find operations.

Following the review, participants were allotted 15 minutes to work on the 18 splay-tree tasks using either Sketchmate or paper and pencil. Participants were asked to complete these tasks as quickly and accurately as possible. After completing the tasks using the first

Table 5.1: Instructor Experiment I Summary Data. SD is the standard deviation.

	Sketchmate (n=25) Mean (SD)	**Paper/Pencil (n=25) Mean (SD)**
Number completed	39.44 (6.131)	25.84 (5.893)
Percent completed correctly	99.12 (1.7)	71.23 (20.1)

Table 5.2: Instructor Experiment I User Satisfaction Data

	Sketchmate (n=25) Mean (SD)	**Paper/Pencil (n=25) Mean (SD)**
Create tree	6.83 (0.373)	4.00 (1.500)
Demonstrate find	6.40 (0.748)	2.84 (1.255)
Demonstrate rotations	6.36 (0.933)	2.40 (1.058)

tool, the experimenter provided a second, shorter review (about 2 minutes), of a single rotation using the other tool (either whiteboard or Sketchmate). Participants were then given 15 minutes to complete the identical splay-tree tasks using the second tool.

At the end of the session, participants filled out a Likert scale questionnaire. For each question, participants responded by rating each tool on a scale of 1 (unfavorable) to 7 (favorable). Paper and pencil products were collected and the computer saved Sketchmate responses. The primary experimenter scored all responses for operation accuracy and number of operations completed. A second experimenter independently scored 80% of the paper and pencil and Sketchmate assignments. Interscorer agreement was calculated for the sums on each assignment. Results showed interscorer agreement was 100% for operations completed and 95% (38 of the 40 assignments) for operations completed accurately. The disagreements on total operations completed accurately never exceeded more than one on any assignment. These data suggest that student responses were scored consistently.

5.1.4 Results and Discussion

The means and standard deviations across groups and dependent variables are shown in Table 5.1. As predicted, both measures showed a significant difference between the group using Sketchmate and those using paper and pencil. Specifically, participants using Sketchmate completed more operations ($t(24) = 10.213$, $\rho < .001$) and had a higher percent correct ($t(24) = 7.027$, $\rho < .001$).

Data from the exit questionnaire is shown in Table 5.2. Note that the scale ran from 1 (unfavorable) to 7 (favorable). Participants indicated that it was easier to create splay trees ($t(25) = 2.207$, $\rho < .001$), show the find operation ($t(24) = 12.815$, $\rho < .001$), and demonstrate tree rotations ($t(25) = 14.733$, $\rho < .001$) using Sketchmate, as opposed to paper and pencil.

Results showed that participants completed about 2.6 operations per minute (39.44/15) with Sketchmate, but only about 1.7 (25.84/15) operations per minute with paper and pencil. These results suggest that instructors will be able to prepare lecture material much more efficiently using Sketchmate as opposed to drawing them using paper and pencil. Perhaps more importantly, diagrams prepared with Sketchmate were almost always accurate (less than 1% errors) and these errors were always because the participant miscopied the tree. In contrast, almost 30% of the diagrams prepared using paper and pencil were wrong. These results suggest that instructors using Sketchmate may prepare better lecture examples that do not confuse students with inaccurate diagrams.

In addition to being able to prepare more examples, more accurately, participants also found Sketchmate easier to use and liked the results better. It should also be noted that participants were given only 2 minutes of instruction specific to Sketchmate. Only two of the participants asked questions during the experiment and none of the questions pertained to the tool itself. The lack of questions despite the limited instruction is important because one of our main goals is to provide a tool that instructors can simply pick up and use, without the need for spending a large amount of time learning how to use the tool.

5.2 Experiment II: Lecture Presentation

Experiment II was designed to simulate a lecture environment where each participant from Experiment I was asked to deliver a presentation on a provided splay-tree operation. A between-subjects, post-test only design was used with participants randomly assigned to either a Sketchmate or whiteboard condition. After delivering the initial presentation, participants were asked follow-up questions. To simulate the incidental questions that students may ask during a lecture, participants were not given these follow-up questions ahead of time. Dependent variables included the accuracy of their responses to the questions and the number of seconds required to respond to each incidental question. The hypotheses were that participants using Sketchmate would answer the questions more quickly and more accurately.

5.2.1 Subjects, Setting, and Materials

The participants in Experiment II were the same as those who participated in Experiment I. The presentations were given in one of three different, empty classrooms, depending on availability. Participants were provided with either a computer running the Sketchmate program or a whiteboard, marker, and eraser.

5.2.2 Design and Analysis

A between-subjects, post-test only design was used to test for differences across groups when responding to incidental questions in an analogue lecture environment. Participants responded using either Sketchmate or the whiteboard. The dependent variables were the number of seconds to complete both the second and third (incidental) operations and the accuracy of the resulting trees. Directional t-tests were used to test for significant differences across dependent variables with Bonferroni corrections used to control experiment-wise error rates. All differences were considered significant at the $\rho < 0.05$ level.

Table 5.3: Instructor Experiment II Summary Data

	Sketchmate (n=13) Mean (SD)	**Paper/Pencil (n=12) Mean (SD)**
Splay operations correct	5.00 (0.00)	1.75 (0.754)
Demonstration time (seconds)	166.46 (17.376)	611.92 (111.226)

5.2.3 Procedures

After the participants completed the first experiment where they used both Sketchmate and paper and pencil, they were randomly assigned to deliver their presentations using either Sketchmate (experimental group) or the whiteboard (control group). Every participant from the first experiment also participated in this experiment. All were given the same splay tree and asked to prepare the same presentation showing the splay operations necessary to insert a value into the tree. They were also told to be prepared to answer additional questions about the tree, but were not told the specific questions ahead of time. Participants chose appointment times within one week of the first experiment to give them enough time to prepare and be comfortable with their materials.

After completing their presentation, which lasted roughly 5 to 10 minutes, we attempted to simulate what might happen in a classroom and asked the participant to go back to the original tree and show the steps necessary to insert a different value into the tree. When this insertion was complete, we asked them to show the steps needed to insert a third value into the original tree.

The primary experimenter used a stopwatch and recorded the number of seconds required for the participants to draw and describe each operation. Operational accuracy was based solely on the final diagram, not the verbal descriptions. As in Experiment I, partial credit was awarded when splay steps were performed correctly in light of previous errors. The computer saved Sketchmate diagrams and a video-recorder was used to collect permanent products, which were used to score whiteboard responses. A second experimenter independently scored 50% of the responses using the videotape records. Interscorer agreement was calculated for the number of seconds to respond and the resulting trees. Results showed interscorer agreement was 100% for operations completed accurately and 100% for number of seconds to complete an operation (within 2 seconds). These data suggest that student responses were scored consistently.

5.2.4 Results and Discussion

The means and standard deviations for operation completion times and accuracy rates are included in Table 5.3. As expected, participants using Sketchmate completed more of their splay operations correctly ($t(23) = 15.574$, $\rho < .001$) and completed them in less time ($t(23) = 4.117$, $\rho < .001$).

These results show that on average participants presenting with Sketchmate responded to each question in less than 1.5 minutes whereas those presenting with the whiteboard required over 5 minutes to respond to each question. Furthermore, participant diagrams

were always correct when using Sketchmate, but only 35% were accurate when using the whiteboard.

Extrapolating these data to a 10-minute question-and-answer session following a lecture shows that using Sketchmate would allow our participants to respond to 7 student questions and each response would be accurate. However, using the whiteboard our participants would have been able to respond to only two questions and at least one of the diagrams would be wrong, which would likely hinder student learning. Thus the instructor using Sketchmate can leverage its efficiency by addressing more student questions, and address them more clearly with accurate diagrams.

5.3 Limitations

There are external validity limitations associated with our experiments. Our participants were students, as opposed to instructors. By selecting upper-level undergraduate and graduate students, we hoped our participants were familiar with splay trees and had some degree of presentation experience. However, those who regularly teach splay trees may have been able to complete the preparation and presentation tasks more quickly and more accurately.

Although using students in the role of instructors certainly is not preferred, it should be noted that these participants would be very similar to recent graduates who are teaching for the first time in that they have learned the material at one time, but that material may not be fresh in their minds. This is also similar to seasoned instructors, such as my committee chair who teaches data structures on a rotating basis and therefore must refresh himself each time he teaches the course. Thus these upper level students, while not our target population, do match a segment of the population to which we are aiming Sketchmate.

Additionally, in Experiment II we used analogue conditions to mimic students asking unexpected questions in class. Participants responded to experimenter-developed, pre-planned questions in a room with only the primary experimenter present. Future researchers could address these external validity limitations by running similar studies with professors or instructors serving as participants in their typical classrooms when responding to unplanned student-generated questions (i.e., generalization across participants, settings, and tasks). Also future researchers could conduct similar studies with tools designed to target other complex structures that are frequently drawn (e.g., graph data structures, chemical structures).

5.4 General Discussion

The results of these initial experiments are encouraging. We feel we have succeeded in creating a tool that helps an instructor prepare for lectures more quickly and more accurately. Also, the instructor using this tool to help deliver the lecture will be able to both present animations of the operations as they take place and create the animations quickly, easily, and most importantly, accurately. Sketchmate's accuracy is important because when instructors accidentally draw an incorrect tree, students are likely to become frustrated and confused. Additionally, some students may have to unlearn concepts that they accidentally

acquired while viewing the incorrect tree. Since Sketchmate always produces the correct result, students will always be taught the correct way to perform splay-tree operations.

Sketchmate's accuracy also comes into play when the instructors are asked follow-up questions because while it is easy to prepare a lecture with predefined examples, students often pose questions that the instructor may not have considered. In these cases, instructors using Sketchmate can rest assured that they will always perform the steps for the algorithm correctly, and therefore correctly answer the questions.

Chapter 6

Experimental Evaluation of Sketchmate for the Student

When we designed the student version of Sketchmate, we had four goals in mind:

1. Improve student learning outcomes,
2. Decrease the amount of time students need to spend on practice materials,
3. Provide a pleasurable learning environment, and
4. Provide an intuitive interface that could be learned quickly.

We executed two experiments which were designed to determine whether or not we achieved these goals. Both experiments involved having students in an introductory data structures class perform a set of lab exercises. Half the students used Sketchmate and half used paper and pencil.

To determine whether Sketchmate improved learning outcomes, we followed the lab activity with an unannounced pop-quiz. To determine whether Sketchmate decreased the amount of time students need to spend on practice materials, we measured participants' completion time on a set of splay-tree tasks. To determine whether Sketchmate provided a pleasurable learning environment, we asked students to complete a Likert-scale questionnaire, with 1 being the most unfavorable opinion and 7 being the most favorable opinion. Finally, to determine whether Sketchmate provides an intuitive interface, we provided no advance instruction in how to use Sketchmate. We examined the students' completion times and the number of questions they asked about the tool to subjectively evaluate their experience. We thought that either excessively long completion times or numerous questions about how to use the tool would be evidence of an interface that was more challenging to learn than we wanted.

In order to get a baseline measure of learning outcomes and student efficiency, our first experiment used an initial version of Sketchmate which provided a minimal amount of feedback. This version would only give the users a warning if they chose an incorrect action. It did not explain why the action was wrong or provide any other details. For reasons explained in the following section, we could not be sure whether or not students would be able to perform exercises more quickly with this initial version of Sketchmate, and because of the minimal feedback, we did not expect learning outcomes to be improved.

The second experiment conducted was identical to the first in all procedures and materials, with the exception of Sketchmate itself. This version of Sketchmate was designed to provide a rich set of feedback, including an explanation of why the user's selection was incorrect and prompts on what the step in that particular problem would be. Once again we measured the amount of time to complete the exercise and the amount of material learned. We hoped that both practice times would be decreased, and learning outcomes would be improved in this version of Sketchmate.

6.1 Experiment I: No Feedback

The first experiment was a between-subjects, post-test only design. We gave participants a list of trees and operations to perform on the trees and asked them to work through these problems using either Sketchmate or paper and pencil. This experiment was designed to test three different hypotheses.

Our first hypothesis was that there was no significant difference in student learning, as measured by a pop-quiz, when completing a lab exercise with either method. The reason for structuring our main hypothesis as we did is that the feedback-poor version of Sketchmate in effect provides an automated paper-and-pencil environment, but does not provide any augmented learning facilities, such as immediate feedback. Hence we did not anticipate that there would be any difference in student learning. There were also reasons to be concerned that an automated environment might actually lessen learning. For example, paper-and-pencil requires students to repetitively copy nodes, which might bore and distract them, or might slow them down and provide a more active learning experience.

Second, we tested for a significant difference in the amount of time participants needed to complete their lab exercise between the two groups. Our thinking was that students using Sketchmate would not be required to recopy each tree node and hence may be able to complete the exercise more quickly than those students redrawing the trees at each step with pencil and paper. However, the students were also not given any instruction in how to use Sketchmate, and hence the time required learning to use the tool might cause students to spend more time solving exercises with it than with paper and pencil.

One thing we specifically did not measure was the score on the lab exercise itself. This was because we could not maintain consistent scoring between paper and pencil and Sketchmate. Because it is not possible to perform certain actions (for example, you cannot perform a zig splay operation on a node that is a right child), we had to have some way of penalizing participants to keep them from guessing. However, there was no way to penalize a participant making a similar attempt with paper and pencil.

Finally, we tested for a significant difference in participant perception of difficulty between the two methods. We hoped that the ease of use data would show that participants enjoyed using Sketchmate at least as well, if not better, than using paper and pencil.

6.1.1 Subjects, Setting, and Materials

After receiving Institutional Review Board (IRB) approval from The University of Tennessee, we recruited 36 students (34 male, 2 female; mean age 22.1) from the Fall 2008

undergraduate course in data structures (CS140) in the Electrical Engineering and Computer Science department at The University of Tennessee. The experiment was run during the semester as part of a class exercise.

The experiment took place using the computers in a Computer Science lab on campus. All machines had a Pentium 4 processor with 1 GB of RAM and were running the Ubuntu operating system. Participants were provided a copy of the Sketchmate program on a 1 GB USB drive.

Additionally, experimenters developed one set of 21 splay-tree tasks. Each problem presented a different splay tree and participants showed the resulting tree after a specific operation. The first 12 problems involved demonstrating basic splay operations such as zig, zag, zig-zag, etc (i.e., type 1 problems). The remaining 9 problems were equally divided between find, insert, and delete operations (i.e., type 2 problems).

6.1.2 Design and Analysis

A between-subjects, post-test only design was used to test for differences on efficiency and learning for participants using Sketchmate (experimental condition) and paper and pencil (control condition). The dependent variables included the amount of time to complete the lab exercise and the overall score on the pop quiz. Independent samples t-tests were used to test for significant differences across dependent variables. All differences were considered significant at the $\rho < 0.05$ level.

6.1.3 Procedures

The data structures course was structured as a single, combined lecture class with two different lab sections. The experiment took place during the lab sessions in the computer lab. Participants had been given a lecture in class and were given a further overview (roughly 1 hour) of splay trees and their operations in the lab. Examples were presented at the whiteboard for the zig, zag, zig-zag, and zig-zig splay operations, as well as one example for each of the find, insert, and delete operations. Participants were encouraged to ask questions as they normally would in a classroom setting. Experimenters gave no demonstration or explanation of Sketchmate or how to use it.

Following the review, researchers distributed the lab exercise. Half of each lab section used Sketchmate and the other half used pencil and paper to complete the exercises. Paper and pencil participants were asked to show their work on the paper provided. Also, each was asked to raise their hand at the completion of the exercise so that the experimenter could accurately mark their completion time and collect their work. All paper and pencil products were collected and the computer saved Sketchmate responses and the time to complete the exercise on the USB drive, which was then collected. Additionally, experimenters set up a video recorder to record paper and pencil students for a permanent record of completion times.

At the end of the session, participants filled out a Likert scale questionnaire. For each question, participants responded by rating the method they used (either Sketchmate or paper and pencil) on a scale of 1 (unfavorable) to 7 (favorable).

At the beginning of the next lecture period, all students present were given a paper and pencil pop quiz on splay-tree operations similar to those used in the lab exercise. We chose

Table 6.1: Student Experiment I Summary Data. SD is the standard deviation.

	Sketchmate			Paper/Pencil		
	n	**Mean (SD)**	**Median**	**n**	**Mean (SD)**	**Median**
Time in seconds	n=20	1931 (625)	1813	n=20	2188 (695)	2166
Quiz percentage	n=19	70.95 (23.6)	80.00	n=17	71.18 (27.8)	80.00

to use a pop-quiz over an announced quiz in an effort to better control the experiment. That is, we feared that if the students knew a quiz was coming, they might choose to study for it and we would not be able to control 1) the amount of time they spent studying for it, and 2) the method they used for studying for it (Sketchmate or paper/pencil).

Only results from students who completed both the lab exercise and the quiz were used in this study. Although students were told that the exercise and the pop quiz were part of their class grade, any student who submitted work for either the exercise or the quiz received full credit. Also, after the pop quiz, students who did not use Sketchmate were given the opportunity to do so, so as not to put anyone at a disadvantage in their learning.

The primary experimenter scored all responses for the time to complete the lab and for pop-quiz accuracy. Each pop-quiz question was given partial credit based on the amount of work shown and the correctness of the result. For example, students who demonstrated the insert operation by inserting the node, but then not splaying the new node to the root only received 25% credit for this answer.

A second, independent experimenter also viewed one of the lab sessions to check for interscorer agreement. Results showed an agreement of completion times within 5 seconds for 90% (10 of 11) of the participants. This experimenter also checked half of the pop quizzes resulting in 100% agreement. These data suggest that the times and quiz results were scored consistently.

6.1.4 Results and Discussion

Table 6.1 shows the mean and standard deviations across dependent variables for the control and experimental groups. There was no significant difference between those using Sketchmate and those using paper and pencil with regard to completion time ($t(34) = -1.165$, $\rho = .25$) or quiz score ($t(34) = -0.27$, $\rho = .979$).

Data from the exit questionnaire is shown in Table 6.2. Note that the scale ran from 1 (very difficult) to 7 (very easy). Results showed participants found it significantly easier to demonstrate the insert ($t(38) = 3.386$, $\rho = .002$, Effect Size = 0.48) and delete ($t(38) = 3.043$, $\rho = .004$, Effect Size = 0.44) operations with Sketchmate compared to paper/pencil. Additionally, results approached significance for participants who found it easier to use paper/pencil over Sketchmate to demonstrate the splay operation ($t(38) = -1.766$, $\rho = .085$, Effect Size = 0.27).

Results were not statistically significant, but did show that students using Sketchmate completed the exercises more quickly than with paper and pencil. Although participants were not given instruction on Sketchmate, it is possible that they used some time during the exercise to become familiar with how to use the program. If true, this would increase the completion time and may account for the fact that our time difference was not significant.

Table 6.2: Student Experiment I Ease of Use Data. SD is the standard deviation.

Operation Performed	**Sketchmate (n=20) Mean (SD)**	**Paper/Pencil (n=20) Mean (SD)**
Splay	4.32 (1.42)	5.05 (1.20)
Find	5.37 (1.34)	4.76 (1.30)
Insert	5.58 (1.02)	4.33 (1.23)
Delete	5.16 (1.17)	3.95 (1.32)

Additionally, while students using Sketchmate didn't perform better on a follow-up pop quiz, they also didn't perform any worse. This is important because there was concern that participants would not interact with the material as much when using Sketchmate. Therefore we feared that the Sketchmate students would not perform as well on the pop-quiz. The lack of significant differences between homework methods supports the idea that using Sketchmate did not deter students from learning.

Participants seemed to find Sketchmate easier to use when attempting to demonstrate the more complex operations: insert and delete. And although the results did not show a significant difference between groups, the effect size (insert 0.44 and delete 0.48) suggest that there may be a medium effect for the preference of Sketchmate over paper and pencil. One possible reason for this may be because both of these operations require multiple steps. For example, a delete requires a find operation to bring the desired node to the tree's root, then after removal, the largest value in the left subtree must be splayed to the root before a final connection between the two subtrees is performed. Sketchmate is designed such that users cannot progress unless they have already completed the current step. While this may seem like a way of giving the students the answer, it can also be seen as another method of instruction.

Finally, while participants seemed to slightly prefer paper and pencil to Sketchmate when demonstrating the basic splay operations, it was difficult to know exactly why. Again our results were not statistically significant. Additionally the effect size (0.27) tended toward a small effect in the preference of paper and pencil over Sketchmate. And while our questionnaire did contain a free response area for participants to list possible improvements to the program, none listed anything specific to these operations. Also, because participants used either Sketchmate or paper and pencil, it was not possible for them to directly compare one tool to the other. This may have led to the lack of comments regarding how to improve the splay demonstration functionality.

6.2 Experiment II: Detailed Feedback

Once again, the experiment was a between-subjects, post-test only design. We gave participants a list of trees and operations to perform on the trees and asked them to work through these problems using either Sketchmate or paper and pencil. This experiment was also designed to test the same three hypotheses.

Our first hypothesis tested was that there was no significant difference in student learning, as measured by a pop-quiz, when completing a lab exercise with either method. The reason for structuring our hypothesis as we did is that the feedback-rich version of Sketchmate essentially provides an extra set of immediate, corrective instruction. Therefore we wanted to see if there was any effect on learning for those using Sketchmate when completing the practice exercises.

Second, we tested for a significant different in the amount of time participants needed to complete their lab exercise between the two groups. We were unsure if the students using Sketchmate would require more time, because they would be viewing detailed, corrective instruction when they made mistakes. However, if the students were able to better learn the material, then it may also be possible that they could complete the remainder of the exercises more quickly based on their new knowledge.

Finally, we tested for a significant difference in participant perception of difficulty between the two methods. We hoped that the ease of use data would show that participants enjoyed using Sketchmate at least as well, if not better, than using paper and pencil.

6.2.1 Subjects, Setting, and Materials

Using the same Institutional Review Board (IRB) approval as the previous experiment, we recruited 34 students (30 male, 4 female; mean age 21.8) from the Spring 2009 undergraduate course in data structures (CS140) in the Electrical Engineering and Computer Science department at The University of Tennessee. The experiment was run during the semester as part of a class exercise.

The experiment took place using the computers in a Computer Science lab on campus. All machines had a Pentium 4 processor with 1 GB of RAM and were running the Ubuntu operating system. Participants were provided a copy of the Sketchmate program on a 1 GB USB drive.

Additionally, experimenters used the same set of 21 splay-tree tasks from the Fall 2008 experiment. Each problem presented a different splay tree and participants showed the resulting tree after a specific operation. The first twelve problems involved demonstrating basic splay operations such as zig, zag, zig-zag, etc (i.e., type 1 problems). The remaining nine problems were equally divided between find, insert, and delete operations (i.e., type 2 problems).

6.2.2 Design and Analysis

A between-subjects, post-test only design was used to test for differences on efficiency and learning for participants using Sketchmate (experimental condition) and paper and pencil (control condition). The dependent variables included the amount of time to complete the lab exercise and the overall score on the pop quiz. Independent samples t-tests were used to test for significant differences across dependent variables. All differences were considered significant at the $\rho < 0.05$ level.

6.2.3 Procedures

The procedures for this experiment were identical to those used in the Fall 2008 experiment (Section 6.1.3) with one exception. As a check for interscorer agreement on completion

Table 6.3: Student Experiment II Summary Data. SD is the standard deviation.

	Sketchmate			Paper/Pencil		
	n	**Mean (SD)**	**Median**	**n**	**Mean (SD)**	**Median**
Time in seconds	n=17	1453 (362)	1424	n=17	2332 (488)	2264
Quiz percentage	n=12	55.00 (26.2)	57.50	n=11	51.18 (19.3)	50.00

Table 6.4: Student Experiment II Ease of Use Data. SD is the standard deviation.

Operation Performed	**Sketchmate (n=12) Mean (SD)**	**Paper/Pencil (n=11) Mean (SD)**
Splay	4.05 (1.48)	5.18 (1.19)
Find	5.00 (1.54)	5.00 (1.65)
Insert	5.12 (1.11)	5.00 (1.37)
Delete	4.76 (1.82)	4.00 (1.50)

times a second, independent researcher sat in the back of the classroom and wrote down the completion time for each of the pencil and paper participants. The primary experimenter later used the videotape to check these times. Results showed an agreement of completion times within 2 seconds for 100% of the participants. Also a second experimenter graded 48% (10 of 21) of the pop quizzes resulting in 90% (9 of 10) agreement. These data suggest that the times were scored consistently.

6.2.4 Results and Discussion

Table 6.3 shows the mean and standard deviations across dependent variables for the control and experimental groups. Results showed that participants using Sketchmate completed their exercises in significantly less time ($t(32) = -5.958$, $\rho < .001$) than those using paper and pencil. However there was no significant difference between the groups based on quiz score ($t(21) = 0.329$, $\rho = .745$).

Data from the exit questionnaire is shown in Table 6.4. Note that the scale ran from 1 (very difficult) to 7 (very easy). Results showed participants found it significantly easier to demonstrate the splay ($t(32) = -2.433$, $\rho = .021$, Effect Size = 0.40) operations with paper/pencil compared to Sketchmate. None of the other comparisons were significant.

Results showed that students were able to complete the exact same exercises nearly 40% faster when using Sketchmate versus paper and pencil. This means that an instructor can leverage this faster pace by possibly providing students with 30 problems to be solved during a 50 minute lab period, instead of 20. This increased number of problems would not only provide more chances for students to practice the algorithms, but would also expose them to a wider variety of tree configurations. Hopefully these extra problems could be used to increase student learning. Alternatively, the instructor can assign the same number

of problems and anticipate getting roughly the same learning outcomes, while demanding less of the students time.

The results from the follow up quiz showed no significant difference between the two methods. However there was still a 14% improvement (57% versus 50%) and it is possible that increasing the number of participants could produce a significant result. And while there is no statistical significance, results suggest that even though using Sketchmate for the homework exercises did not improve learning, it also did not hurt. One possible explanation for the lack of a difference could be due to the fact that the Sketchmate participants simply did not spend as much time with the material as the other group.

Survey results showed no significant differences between Sketchmate and paper and pencil for any of the operations except splay. For the splay operation, participants preferred using paper and pencil. Once again, participants were given a free response area where they could write out suggestions for possible improvement, yet none listed anything specific to the splay operation.

Additionally even though the numeric survey data did not support a preference for Sketchmate, participants made numerous comments in the free response area regarding how much they enjoyed using Sketchmate. Specific comments include, "Animation of splay operations made it much easier to understand how they work" and "It helped to visualize each step."

When comparing survey data between the experiments using the different versions of the student tool (Sketchmate with and without feedback), participants reported that the version without feedback was slightly easier to use for all operations. This could mean that the feedback mechanism itself is making the program harder to use. However because participants did not use both tools, a direct, side-by-side comparison was not made.

Overall, participants seemed to have an easy time using Sketchmate. Again, no class time was used to teach participants how to use Sketchmate and there was only one question about it during the experiment. These experiences suggest that Sketchmate is an intuitive tool that requires little time to learn how to use.

6.3 Limitations

One limitation for these experiments was the lack of participants. Although some results were statistically significant, it would have been nice to have a larger sample size for each experiment. Unfortunately we were limited by the number of students who typically enroll in this course each term, typically 40 - 50 students.

Another limitation in these experiments was the inability to have participants use both Sketchmate and paper and pencil, then compare their preferences between the two tools. By having them experience both methods, we may have gotten a more accurate measurement of preference.

6.4 General Discussion

The results of these experiments were somewhat expected. That is, while it would have been nice if the Sketchmate students had performed better on the pop quiz, it may be

likely that they did not perform better simply because they did not spend as much time with the material as the pencil/paper group.

That being said, because the Sketchmate group spent less time, that would mean that instructors could rectify the time-on-task imbalance by assigning more problems to these students. This would increase their time spent with the material and give them more practice with the operations. Best of all, with an increased number of problems, it would be more likely that they would make a mistake along the way, thus providing the computer a chance to provide additional, immediately feedback on the algorithm's steps. This extra practice and instruction may help increase student learning.

In both experiments, survey results showed that pencil and paper was easier to use than Sketchmate when demonstrating the splay operation. While this certainly could be because of Sketchmate itself, it should also be noted that these problems were the first ones presented (i.e., they were presented before the find/insert/delete questions). And because no instruction was given for using Sketchmate, it is possible that they reported the increased difficulty simply because they were having issues figuring out how to use the tool.

One encouraging result from these experiments was that overall participants using Sketchmate seemed to note that they enjoyed using the computer to do their work in the free response portion of the survey. One participant wrote, "Visually seeing the tree ... gives a better understanding of trees." Another person told the experimenter that they enjoyed using the computer, simply because it was a different form of delivery. By presenting students with a different method to answer questions and work with the material, instructors can help provide variety that helps make the learning experience more enjoyable.

Finally, if you consider that no extra time was given to teaching how to use Sketchmate and overall participants seemed to enjoy working with Sketchmate, the fact that they did not perform any better on the quiz does not detract from this tool's use. That is, simply giving students a different way of viewing and interacting with these concepts may be a benefit in itself as it can help to better engage students and keep them more interested in the material. Also, Sketchmate's ability to grade the exercises provide an extra benefit to those responsible for grading student work as this becomes an automated process.

Chapter 7

Conclusions / Future Work

We developed and evaluated an instructional program called Sketchmate. Sketchmate was designed in two forms: one for instructors during lecture preparation and presentation and a second for students' use as they practiced splay-tree algorithms. We designed and performed a set of experiments to measure the effectiveness of Sketchmate relative to traditional tools. The following section discusses some of the lessons we learned during the process, which may be applied by others developing similar tools in the future. Next we present our conclusions from the different experiments we conducted and possible future research.

7.1 General Conclusions

The overreaching goal of this research was to discover techniques that might be useful when developing instructional and learning tools. To that end, we aimed to create a tool that was easy to use, required less effort than traditional methods (such as paper and pencil), provided pleasing animations, and improved learning outcomes. We selected one particular concept, the splay-tree data structure, and fully investigated the creation of a tool to help teach this concept so as to gain as much information about the development process as possible. The following lessons learned about the development of Sketchmate can be extrapolated to other structures and other educational disciplines.

7.1.1 Node Layout

Initially it seemed to be a trivial exercise to place nodes on the screen in an efficient manner. We learned that while this is an easy exercise when placing nodes in complete trees, it is more difficult for sparse trees because they become too spread out on the screen. To address this, we developed a novel layout algorithm so that nodes could be compactly placed in a fashion that was not only attractive, but that made sense to the user, and maximized screen space.

Additionally, we found that these tree manipulations would often cause layout issues. That is, the layout process for one tree configuration often times is not appropriate after a tree rotation takes place. To address this, we needed to essentially write collision detection code and when such a collision or overlap of nodes occurred, Sketchmate needed to apply

the layout algorithm to the current tree configuration so that it would continue to be attractive and easy to understand.

7.1.2 Animation Movements

Moving objects during the animations presented another interesting lesson. We found that while it is easy to create animations that move any number of tree nodes simultaneously, doing so was distracting as users were not able to focus on each part individually. This is especially noticeable when there is a need to disconnect a subtree on a splay rotation. Initially Sketchmate was designed so that the tree nodes would begin their rotation at the same time as the appropriate subtree was disconnected, moved out of the way, and then reconnected at the end. To provide for better comprehension, we broke these movements out into separate steps and required the user to click a "Next" button between each step. Thus we were able to have the subtree disconnect and move out of the way first, then show the tree rotation, followed by a reconnection of the subtree in three distinct steps. This gave users the ability to focus on that particular stage of the algorithm and watch how relevant parts of the tree were affected at each step.

7.1.3 Instructors Use of History Mechanism

When developing the instructor tool, one of our goals was to provide the ability to step back to a previous state of the tree. However, one thing we did not anticipate was the different ways this mechanism could be used. Certainly we expected users to make use of this feature to go back and provide another demonstration of the algorithm while explaining the steps. We envisioned instructors could do this and ask students to focus on a particular area of the structure. However, we also found that instructors could step back to a previous state and demonstrate an entirely different operation. For example, perhaps the instructor originally showed the steps needed to delete a particular value from the tree. By using the history mechanism, instructors could now go back to the original state of the tree and show the steps necessary to delete a different value. Doing so gives the instructor the ability to not only demonstrate different examples, but can also allow them to address questions when curious students ask questions about the algorithm.

Another unanticipated way in which instructors used the history mechanism was to quickly review the operations they had performed. This review provided a nice summary of a lecture on splay trees.

7.1.4 Usability of Student Tool

When redesigning Sketchmate for the student tool, we found that we could design Sketchmate's interface such that user instructions could be placed on the screen along with the current problem. By doing so, we were able to eliminate the need for a user's manual, thus possibly decreasing the amount of time necessary to learn how to use the tool.

Also in the realm of ease of use, we found that there are times when it is helpful to maintain a representation of the original state of the tree before any manipulations have taken place. This original image helps to provide a reference point when working through the steps of an algorithm.

7.1.5 Student Feedback

When developing the rich feedback for the student tool, we found that we could set up a classification scheme for user errors. We took advantage of these patterns when deciding what feedback we would give. Finally, we found that we could maximize our space and the usefulness of our feedback by providing generic images, along with text descriptions. This helped to give students a better context of why their actions were incorrect so that they could better learn from their mistakes.

7.1.6 Experimental Design

Finally, during our research we learned that there are a number of interesting questions we can ask about Sketchmate. We also found that designing valid experiments to answer these questions is challenging because it is difficult to control for individual design elements. For instance, we originally wanted to maximize the power of the computer to provide immediate, specific feedback to the students. However, we learned that feedback has a powerful effect on learning. Therefore we needed to run two different experiments so as to differentiate between the effects of Sketchmate itself with the effects of the feedback.

Experimental design for these experiments was also challenging simply based on the sample sizes we had to work with. We wanted to run our experiments with the instructor's tool with actual instructors. However the logistics for such an experiment were not feasible. For that reason, we chose to design the experiments to work with upper level students in an analogue environment. While these conditions may not have been as preferable as an in-situ environment, our results were still important and useful.

7.2 Sketchmate for the Instructor

Results from the experiments with the instructor's tool showed that using Sketchmate helped improve the efficiency of both preparing for lecture and for presenting splay-tree material. The experimental results also showed that the trees produced by Sketchmate were more accurate than those completed with either paper and pencil or the whiteboard. The results suggest that Sketchmate lessens the cognitive burden on instructors who are presenting material about splay trees, thus allowing them to focus more of their attention on the students and their presentation. The results also suggest that instructors can use Sketchmate to present more splay-tree examples and to do so having confidence that their examples are correct.

Future research based on the instructor's tool could be done with instructors as they used Sketchmate in the classroom. Such experiments could simply reproduce what we have done here with our analogue environment and actually test our hypothesis in-situ.

Another useful experiment could also make use of actual instructors with Sketchmate and measure student learning relative to the two tools. That is, instructors could be recruited to use their traditional methods with one section of their class and then use Sketchmate with another. Student learning could be measured using specifically designed quiz questions, similar to what we used in experiments with the student tool.

It may also be useful to have instructors use the other splay-tree tools currently available in the classroom. This would provide another benchmark on the usefulness of Sketchmate relative to other tools.

Finally, because we found Sketchmate to be useful when working with splay trees, it would be interesting to continue to develop Sketchmate so that it could work with other data structures. We designed the Java code files for Sketchmate in such a way as to make use of class inheritance, therefore it could easily be extended to work with binary search trees, AVL trees, red-black trees, or heaps. Similar experiments to those already conducted could then be performed to measure Sketchmate's usefulness for these other structures, thus providing a wider breadth of coverage within the subject of data structures and possibly even be extended to other domains.

7.3 Sketchmate for the Student

Results from the experiments with the student's tool showed that students using the version of Sketchmate which provided immediate feedback could complete exercises more quickly than those students using paper and pencil. This result is encouraging because it means that students learn just as much using Sketchmate as compared with pencil and paper, but with less time invested with the material.

Unfortunately students did not ever perform significantly better on a follow up, unannounced quiz over the material using Sketchmate. One possible future experiment could be run which holds time constant and presents an endless number of homework exercises. By doing so, we could assure that both sets of students spend the same amount of time with the material. By holding time constant, we could then measure student learning on a follow up quiz to test for a significant difference in learning.

Another possible experiment for the student tool could be similar to what we suggested with the instructor's tool. That is, another set of experiments could be developed to compare Sketchmate with other existing tools. The biggest challenge to such an experiment is that there are no other tools existing that have all of the capabilities of Sketchmate. However, experiments could be designed to compare tools based on the features that the tools do share.

Our results that compared Sketchmate to paper and pencil in terms of ease of use were mixed. We had hoped that participants would have found Sketchmate at least as easy to use as pencil and paper, if not easier. However one problem with our small sample size was that we were not able to allow participants to use both tools. Future experiments could be designed to allow students to use both tools and then provide their thoughts comparing the relative ease of use for each tool.

Also related to the student survey results, we noted that both groups of participants found it easier to use paper and pencil when working the splay exercises. However, because the splay operations were always presented first, students were learning how to use Sketchmate during these problems, thus possibly making these problems seem harder. Future experiments could be conducted by mixing up the different question types so that the splay operation is presented at a different time in the problem set. Other experiments could be done where a researcher could give a quick summary of how to use Sketchmate, thus reducing the learning curve for students when working these exercises.

Bibliography

[Akingbade et al., 2003] Akingbade, A., Finley, T., Jackson, D., Patel, P., and Rodger, S. H. (2003). Jawaa: easy web-based animation from cs 0 to advanced cs courses. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 162–166, New York, NY, USA. ACM Press.

[Anderson et al., 2004] Anderson, R., Anderson, R., Simon, B., Wolfman, S. A., VanDeGrift, T., and Yasuhara, K. (2004). Experiences with a tablet pc based lecture presentation system in computer science courses. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 56–60, New York, NY, USA. ACM Press.

[Baecker, 1981] Baecker, R. M. (1981). Sorting out sorting. *SIGGRAPH Video Review*, 7.

[Baker et al., 1999] Baker, R. S., Boilen, M., Goodrich, M. T., Tamassia, R., and Stibel, B. A. (1999). Testers and visualizers for teaching data structures. In *SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 261–265, New York, NY, USA. ACM Press.

[Beeler, 2007] Beeler, M. (2007). A binary heap animator for sketchmate. MS Project in Lieu of Thesis, The University of Tennessee.

[Berque, 2006] Berque, D. (2006). An evaluation of a broad deployment of dyknow software to support note taking and interaction using pen-based computers. *J. Comput. Small Coll.*, 21(6):204–216.

[Berque et al., 2001] Berque, D., Johnson, D. K., and Jovanovic, L. (2001). Teaching theory of computation using pen-based computers and an electronic whiteboard. In *ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 169–172, New York, NY, USA. ACM Press.

[Biermann, 1998] Biermann, H. (1998). Binary search tree visualization. Website. `http://www.cs.nyu.edu/algvis/java/`. (accessed June 1, 2008).

[Brown, 1988] Brown, M. H. (1988). Exploring algorithms using balsa-ii. *Computer*, 21(5):14–36.

[Buckalew and Porter, 1994] Buckalew, C. and Porter, A. (1994). The lecturer's assistant. In *SIGCSE '94: Proceedings of the twenty-fifth SIGCSE symposium on Computer science education*, pages 193–197, New York, NY, USA. ACM Press.

[Byrne et al., 1999] Byrne, M., Catrambone, R., and Stasko, J. (1999). Evaluating animations as student aids in learning computer algorithms.

[Cormen et al., 2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*, page 404405. The MIT Press, second edition.

[Dan R. Olsen et al., 2004] Dan R. Olsen, J., Taufer, T., and Fails, J. A. (2004). Screencrayons: annotating anything. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 165–174, New York, NY, USA. ACM Press.

[Dershem et al., 2002] Dershem, H. L., McFall, R. L., and Uti, N. (2002). Animation of java linked lists. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 53–57, New York, NY, USA. ACM Press.

[Gogeshvili, 2002] Gogeshvili, A. (2002). Java models. Website. `http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm`. (accessed June 1, 2008).

[Golub, 2004] Golub, E. (2004). Handwritten slides on a tabletpc in a discrete mathematics course. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 51–55, New York, NY, USA. ACM Press.

[Goodrich and Tamassia, 2004] Goodrich, M. T. and Tamassia, R. (2004). *Data Structures and Algorithms in Java*. John Wiley and Sons, Inc, third edition.

[Hamilton-Taylor and Kraemer, 2002] Hamilton-Taylor, A. G. and Kraemer, E. (2002). Ska: supporting algorithm and data structure discussion. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 58–62, New York, NY, USA. ACM Press.

[Hansen et al., 1998] Hansen, K. M., Ravn, A. P., and Stavridou, V. (1998). From safety analysis to software requirements. *Software Engineering*, 24(7):573–584.

[Hansen et al., 2002] Hansen, S., Narayanan, N. H., and Hegarty, M. (2002). Designing educationally effective algorithm visualizations. *J. Vis. Lang. Comput.*, 13(3):291–317.

[Hundhausen et al., 2002] Hundhausen, C. D., Douglas, S. A., and Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *J. Vis. Lang. Comput.*, 13(3):259–290.

[Ierardi and Li, 1996] Ierardi, D. and Li, T.-W. (1996). Splay tree applet. Website. `http://www.ibr.cs.tu-bs.de/courses/ss98/audii/applets/BST/SplayTree-Example.html`. (accessed June 1, 2008).

[Korhonen and Malmi, 2000] Korhonen, A. and Malmi, L. (2000). Algorithm simulation with automatic assessment. *SIGCSE Bull.*, 32(3):160–163.

[Lewis and Denenberg, 1991] Lewis, H. R. and Denenberg, L. (1991). *Data Structures and Their Algorithms*. Harper Collins.

[Lieberman and Fry, 1995] Lieberman, H. and Fry, C. (1995). Bridging the gulf between code and behavior in programming. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 480–486, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[Mayer, 1989] Mayer, R. E. (1989). Systematic thinking fostered by illustrations in scientic text. *Journal of Educational Psychology*, 81(2):240–246.

[Mayer and Anderson, 1991] Mayer, R. E. and Anderson, R. B. (1991). Animations need narrations: An experimental test of a dual-coding hypothesis. *Journal of Educational Psychology*, 83(4):484–490.

[Mayer and Anderson, 1992] Mayer, R. E. and Anderson, R. B. (1992). The instructive animation: Helping students build connections between words and pictures in multimedia learning. *Journal of Educational Psychology*, 84(4):444–452.

[Mayer and Sims, 1994] Mayer, R. E. and Sims, V. K. (1994). For whom is a picture worth a thousand words? extensions of a dual-coding theory of multimedia learning. *Journal of Educational Psychology*, 86(3):389–401.

[Myers, 2001] Myers, B. A. (2001). Using handhelds and PCs together. *Communications of the ACM*, 44(11):34–41.

[Myers et al., 1998] Myers, B. A., Stiel, H., and Gargiulo, R. (1998). Collaboration using multiple pdas connected to a pc. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 285–294, New York, NY, USA. ACM Press.

[Myller et al., 2007] Myller, N., Laakso, M., and Korhonen, A. (2007). Analyzing engagement taxonomy in collaborative algorithm visualization. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 251–255, New York, NY, USA. ACM.

[Narayanan and Hegarty, 2002] Narayanan, N. H. and Hegarty, M. (2002). Multimedia design for communication of dynamic information. *Int. J. Hum.-Comput. Stud.*, 57(4):279–315.

[Palmiter and Elkerton, 1991] Palmiter, S. and Elkerton, J. (1991). An evaluation of animated demonstrations of learning computer-based tasks. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 257–263, New York, NY, USA. ACM Press.

[Pane et al., 1996] Pane, J. F., Corbett, A. T., and John, B. E. (1996). Assessing dynamics in computer-based instruction. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 197–204, New York, NY, USA. ACM Press.

[Pierson and Rodger, 1998] Pierson, W. C. and Rodger, S. H. (1998). Web-based animation of data structures using jawaa. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 267–271, New York, NY, USA. ACM Press.

[Rasala, 1999] Rasala, R. (1999). Automatic array algorithm animation in c++. In *SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 257–260, New York, NY, USA. ACM Press.

[Rieber et al., 1990] Rieber, L. P., Boyce, M. J., and Assad, C. (1990). The effects of computer animation on adult learning and retrieval tasks. *J. Comput. Based Instruct.*, 17(2):46–52.

[Rößling and Freisleben, 2002] Rößling, G. and Freisleben, B. (2002). Animal: A system for supporting multiple roles in algorithm animation.

[Rößling et al., 2000] Rößling, G., Schüler, M., and Freisleben, B. (2000). The animal algorithm animation tool. In *ITiCSE '00: Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education*, pages 37–40, New York, NY, USA. ACM Press.

[Sanders et al., 2002] Sanders, C., Segewick, B., and Wayne, K. (2002). Growing tree. Website. `http://www.cs.princeton.edu/introcs/GrowingTree/GT.jnlp`. (accessed June 1, 2008).

[Sangwan et al., 1998] Sangwan, R. S., Korsh, J. F., and Paul S. LaFollette, J. (1998). A system for program visualization in the classroom. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 272–276, New York, NY, USA. ACM Press.

[Shaffer, 2000] Shaffer, C. A. (2000). *A Practical Introduction to Data Structures and Algorithm Analysis, C++ Edition*. Prentice Hall, second edition.

[Shimomura and Isoda, 1991] Shimomura, T. and Isoda, S. (1991). Linked-list visualization for debugging. *IEEE Softw.*, 8(3):44–51.

[Stasko, 1992] Stasko, J. (1992). Animating algorithms with xtango. *SIGACT News*, 23(2):67–71.

[Stasko et al., 1993] Stasko, J., Badre, A., and Lewis, C. (1993). Do algorithm animations assist learning?: an empirical study and analysis. In *CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 61–66, New York, NY, USA. ACM Press.

[Stasko, 1990a] Stasko, J. T. (1990a). Tango: A framework and system for algorithm animation. *Computer*, 23(9):27–39.

[Stasko, 1990b] Stasko, J. T. (1990b). Tango: A framework and system for algorithm animation. *Computer*, 23(9):27–39.

[Stasko, 1991] Stasko, J. T. (1991). Using direct manipulation to build algorithm animations by demonstration. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 307–314, New York, NY, USA. ACM Press.

[Stasko, 1997] Stasko, J. T. (1997). Using student-built algorithm animations as learning aids. In *SIGCSE '97: Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, pages 25–29, New York, NY, USA. ACM Press.

[Stasko and Kraemer, 1993] Stasko, J. T. and Kraemer, E. (1993). A methodology for building application-specific visualizations of parallel programs. *J. Parallel Distrib. Comput.*, 18(2):258–264.

[Weiss, 2006] Weiss, M. A. (2006). *Data Structures and Algorithm Analysis in C++*. Pearson Education, Inc, third edition.

[Wilkerson et al., 2005] Wilkerson, M., Griswold, W. G., and Simon, B. (2005). Ubiquitous presenter: increasing student access and control in a digital lecturing environment. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 116–120, New York, NY, USA. ACM Press.

Vita

Michael Cajetan Orsega was born on August 21, 1972 in Washington, DC. He graduated from The Pennsylvania State University in May 1994 with a degree in physics. He began work toward a masters degree at The University of Georgia in the geology department, with special focus on geophysics. In 1996, Michael left school to work as a geological consultant near Atlanta, Georgia. In 1999 he left the industry to return to The University of Georgia to pursue his masters degree in applied mathematics. After completing this degree in 2001, he accepted a faculty position at Central Carolina Community College in Sanford, North Carolina. He also served on the faculty of Kaplan University as both a professor and course developer. In 2005, Michael began work toward his PhD in computer science at The University of Tennessee. He completed this degree in 2009 and has accepted a faculty position at The University of West Georgia in its computer science department.

In addition to these educational pursuits, Michael also holds a Professional Geologist License (Georgia PG001562). He also wrote and currently facilitates four online courses through Education To Go, Inc. Titles include "Introduction to C# Programming", "Intermediate C# Programming", "Introduction to Alice Programming", and "Introduction to Python Programming." As of May 2009, over five thousand students have signed up for these courses.