MULTIMEDIA DATA STRUCTURES

LEARNING SYSTEM

By

DONGBI (CARL) LUO

Bachelor of Science
Jiangsu University of Science and Technology
Jiangsu, China
1982

Master of Science
Jiangsu University of Science and Technology
Jiangsu, China
1989

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

MULTIMEDIA DATA STRUCTURES

LEARNING SYSTEM

Thesis Approved:

Jacques E. LaFrance

Thesis Adviser

J Chandler

H. E. Hedrick

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGEMENTS

Praise be to God. Because of His Mercy I was finally able to finish my thesis.

I would like to express my sincere appreciation to all the people who assisted me in this study. I am especially grateful to my major adviser, Dr. Jacques E. LaFrance, for his encouragement and guidance throughout my academic study at Oklahoma State University.

My sincere appreciation is also extended to Dr. J. P. Chandler, Dr. G. E. Hedrick, and Dr. H. K. Dai for serving on my graduate committee and providing valuable suggestions, ideas, and support.

My respectful and very special thanks to my father Shuming Luo, my mother Linzhen Gao, my wife Ningning Wang, and my daughter Xinmiao Luo for their love, encouragement, support, and confidence in me. And I would like to thank all other members of my family for their love and support.

Finally, I would like to thank Computer Science Department and Mechanical and Aerospace Engineering Department for their support during these two and a half years of study.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

II. LITERATURE REVIEW 4

2.1 Data Structures 4
2.2 Visualization 6
2.3 Multimedia 9

III. DESIGN AND IMPLEMENTATION 12

3.1 Hardware and Software 12
3.2 System Design 14
3.2.1 Design Strategies 14
3.2.2 Implementation 17
3.2.3 System Movie Making Procedure 21
3.3 Design MultimediaAvlTree Movie 27
3.4 Implement AVL Tree 41
3.5 MultimediaBSTree Movie 45
3.6 Source Code Design 48

IV. TESTING AND RUNNING 53

V. SUMMARY AND FUTURE WORK 72

5.1 Summary 72
5.2 Future Work 73

BIBLIOGRAPHY 74

Chapter Page

APPENDIXES....77

APPENDIX A: AVL Tree Lingo Code....78

A.1 Main Script....78

A.2 Insert Function....82

A.3 Delete Function....86

A.4 Search Function....100

A.5 Other Handlers....104

APPENDIX B: Binary Search Tree Lingo Code....121

B.1 Main Script....121

B.2 Insert Function....124

B.3 Delete Function....127

B.4 Search Function....137

B.5 Other Handlers....141

APPENDIX C: Programmer's Guide....150

LIST OF TABLES

Table Page

2.1 Personal Computers Have Evolved Gradually to Include Multimedia Capabilities..... 10

3.1 Lingo Script versus Object-oriented Programming Language..... 13

3.2 Sound Files..... 16

3.3 Director 6 Main Control Items and Their Functions..... 18

3.4 All Contents of AVL Tree Definitions and Conceptions..... 40

3.5 The Contents of Binary Search Tree Definitions and Conceptions..... 46

LIST OF FIGURES

Figure Page

2.1 The Interface of Director 6.0 11

3.1 System Welcome Window 19

3.2 System Main Menu Window 20

3.3 Main Movie Internal Cast Window 21

3.4 System Movie Script Window 22

3.5 System Text 7: AvlTree Window 23

3.6 System Script of Cast Member 7: AvlTree Window 23

3.7 System Score Behavior Window 24

3.8 System Score Script Window 24

3.9 System Score Window 25

3.10 Change Rollover Pointer Image Dialog Box 27

3.11 AVL Tree Interface 29

3.12 The Options of Operations Menu 30

3.13 The Options of File Menu 30

3.14 The Options of Shows Menu 30

3.15 The Options of Help Menu 30

3.16 The Options of Speed Menu 30

3.17 The Example before Double Rotation 32

Figure Page

3.18 The Example after Double Rotation..... 33

3.19 SingleRotateWithLeft Rotation Pattern..... 34

3.20 SingleRotateWithRight Rotation Pattern.....34

3.21 DoubleRotateWithRight Rotation Pattern.....34

3.22 AVL Tree Instruction Window..... 35

3.23 About AVL Tree Window..... 35

3.24 Definitions of AVL Tree Window..... 36

3.25 Select Movie Properties from "WinAB" Window..... 37

3.26 Text Cast Member Properties Window..... 37

3.27 Make Sure to Close Movie Window..... 40

3.28 AVL Tree Internal Cast Window..... 41

3.29 All the Objects on AVL Tree Movie Stage..... 43

3.30 The Sprite Number of the Objects on the Stage..... 44

3.31 MultimediaBSTree Movie Interface..... 47

3.32 Flow Chart for MultimediaAvlTree Movie.....49

4.1 Full AVL Tree.....55

4.2 Tree Full Message.....56

4.3 Out of Stage Message..... 57

4.4 No Input Data Message..... 58

4.5 No Such Node in the Tree Message..... 59

4.6 This Node Is already in the Tree Message..... 60

4.7 Example after Insertion of Node 3, 2, 1..... 61

Figure Page

4.8 Example after Insertion of Node 4, 5, 6, 7..... 62

4.9 Example after Insertion of Node16..... 63

4.10 Example after Insertion of Node15..... 64

4.11 Example after Insertion of Node14..... 65

4.12 Example after Insertion of Node13..... 66

4.13 Example after Insertion of Node12..... 67

4.14 Example after Insertion of Node11..... 68

4.15 Example after Insertion of Node10..... 69

4.16 Example after Insertion of Node 8..... 70

4.17 Example after Insertion of Node 9..... 71

CHAPTER I

INTRODUCTION

With the development of the modern computer, the need for programs becomes more acute, and this requires students and programmers to give more careful attention to data structures and algorithms. A data structure is a mathematical abstract model which consists of data and the operations applied to the data. A model is a tentative description of a theory or system that accounts for all of its known properties. Therefore, the model must be proved, tested, and solved. A systematic tool for solving this well-specified computational problem has been known as an algorithm [11]. To meet the needs for modern techniques of programming, algorithms must be represented in a certain method as clearly and effectively as possible.

Multimedia will play an important role in developing educational techniques. With a large-screen projector and multimedia playback system, a teacher can use multimedia titles to enhance classroom presentation and stimulate questions. The students can further explore topics at home using a multimedia platform.

A multimedia system [16] [29] includes multiple information channels through which communications can be made. The information presented to users can be visualization (text, image, animation, video), audio, or other signals for human sensory systems.

Visualization is the process of transforming information into a visual form [14], such as representing systems, concepts, or objects with computer multimedia, graphics, and users' interfaces [23]. It enables users to observe their simulations and computations. Visualization is a very active and swiftly changing technique. This field has tremendously affected such diverse areas as research, education, science, industry, military, and entertainment. Applying visualization techniques can clearly and effectively represent algorithms and help users improve the understanding of its complex ideas.

Knowing these benefits of multimedia visualization technology rouses us to develop this multimedia system as an assistant tool for learning data structures. As we know, data structures are so important for computer science that almost every program has one or more data structures. Since data structures can make data easy for storage, transfer, retrieval, and maintenance in the program. The study of data structures would be much easier if the learner could be able to visualize the representations of its various complicated operations associated with its algorithm. Therefore, this system could be definitely helpful for learning such abstract data structures.

The primary purpose of this thesis is to design and develop a flexible and user-friendly software for simulating the animated operations of data structures as a teaching and learning tool with multimedia technology. We name this software as Multimedia Data Structures Learning System, or MDSL system. It can help the user to execute and visualize the immediate effects of each step of an operation on each data structure, as well as explain the changes in visualization with sound. The emerging representation formats, such as sound and animation, are much less developed than those for static

graphics [5] [10]. I have explored both sound and animation representation formats in this project. My major task for this thesis program is general design of the MDSL system. Because of limited time for thesis study, I only developed MultimediaAvlTree (AVL Tree; 2,450 lines lingo code, 46 handlers which is similar to the functions or routines in C language) and MultimediaBSTree (Binary Search Tree; 1,650 lines lingo code, 31 handers) software in the MDSL system. I have also modified some previous visualization or multimedia data structures software and imported them to the MDSL system.

The MDSL system has been developed by using Macromedia Director version 6.0. The Lingo scripting language of Director can precisely handle the implementation of this software. This multimedia system has been built on the Microsoft Windows 98 operating system. But it also can run on the MS Windows 95/NT and Mac operating system.

In this thesis, Chapter II will present a discussion on previous work related to data structures, visualization, and multimedia. I will describe all the designing and implementing details and explain various problems which I met in designing the software in Chapter III. Chapter VI will be the testing and running of the system. The summary of the thesis and suggestions for future work are presented in Chapter V.

CHAPTER II

LITERATURE REVIEW

This study involves the design and implementation of an MDSL system in the environment of Macromedia Director 6. The fields associated with its considerations of this system could be grouped into the following three topics: Data Structures, Visualization, and Multimedia.

2.1 Data Structures

Considerable research has been done to develop the study of data structures and algorithms during the past years. Computer programming developed from a craft to an academic discipline in the early 1970s. The pioneering work on the concept and study of data structure was done by many authors such as Dahl [12] and Wirth [32]. Because of their initial outstanding contributions toward this development, programs become the concrete formulations of abstract algorithms based on particular representations and structures of data. During 1980s and 1990s, with the development of the modern computer, people realized the increasing role data structures and algorithms play in the computer programming and its applications. Many scientists pay more attentions on the development of data structures and algorithms.

For example, Reingold and Hansen [26] redifined the concept of data structures. Based on his definition, the data structure could be represented in a set of function associated operations, a storage structure implemented by the functions, and a set of algorithms achieving the result for the corresponding functions.

Baron and Shapiro [6] described two ways for implementing data structures as sequentially storing elements in contiguous memory locations and linked elements based on some logical interconnections.

The tree-based data structures have the organization of linked structures. Aho and Sethi [2] defined a tree as a collection of elements called nodes, one of which is distinguished as a root, along with a relation ("parenthood") that places a hierarchical structure on the nodes.

Adelson-Velskii and Landis [1] introduced the AVL tree as a binary search tree with a balance condition that for every node in the tree, the heights of the left and right subtrees can differ by at most 1.

Weiss [31] published his textbook in which the principle and implementation of data structures were systematically analyzed, described, and coded in C.

Appleton [3] designed a tutorial and C++ implementation of AVL trees. The full C++ source code distribution of AVL trees and AVL tree library was established in web site: AvlTrees.tar.gz (21KB, gzipped tar file). But there is no deletion function in this web site.

Although considerable work has been done to develop the comprehension of data structures, with some researchers even attempting to use a large number of examples and

graphic methods to show how the algorithm works, it is still not easy to understand the implementation of those dynamic data structures. The keen learner would benefit even more if the algorithms could be visualized and tested easily.

2.2 Visualization

Visualization technique is a method of presenting data and data structures, and is an active research area. It enriches the process of scientific discovery and fosters profound and unexpected insights. Since computers have become essential tools to scientists, scientists have to formulate models of natural phenomena using mathematics. In order to simulate complex events, they must automate their models as computer algorithms. That is, scientists analyze their observations of nature in terms of mathematical models, but the volumes of observed data dictate that these analyses be automated as computer algorithms. Unlike hand computations, automated computations are invisible, and their sheer volume makes them difficult to comprehend. Thus scientists need tools to make their computations visible, and this has motivated active development of scientific visualization systems.

In the early days, visualization systems were done more in films and video tapes. The visualization was static and the users could not take part in the situation. Examples: Knowlton [17] directed the first computer-generated movie which showed how an assembly level language was implemented in 1966, Booth's movie [8] showed several algorithms on PQ-trees in terms of different inputs using a hash table and a graph, and Baecker's algorithm animation film [4] showed visualization of nine sorting algorithms.

Thus, since scientists have focused on representations of data structures and created a variety of commercial systems, several visualization systems have been done to display automatically some static graphs of a program's data structures, but they cannot show how the data structure changes and which operation is being executed. These systems can be called static displays which only show the image of data, relationship of data, and overall data structures. For examples, Myers [24] built a system for displaying a data structure which can allow users to specify the variable name to get its graphical display and select one of the formats associated with each data type. Wilson Lee [18], in his master thesis, designed a data structures display system which implements linked-lists, binary search trees, and B-trees on VT100 type terminals. Zernik [33] introduced a system named "Using Visualization Tools to Understand Concurrency" which uses graphs to provide a logical view of execution according to computational threads, messages, and synchronization events, and overcomes the concurrency bugs, as well as giving the users a clear picture of concurrency. But these systems are rigid because they display graphics as characters rather than as a combination of pixels. On the other hand, these systems don't have the operations of deletion for any data structure. As is common with many data structures, the hardest operation is deletion. Once we have found the node to be deleted, we need to consider several possibilities with many complicated cases. In my opinion, as a teaching and learning tool, a system has to include deletion operations for data structures. Otherwise, the system is not a complete one. So, in the MDSL system, I have developed two data structures movies (MultimediaBSTree and MultimediaAvlTree) which have involved all of three operations (insertion, deletion, and search).

Nowadays, the visualization system has become one of the most exciting and rapidly growing fields in computer science. In order to overcome the static display's shortcomings, scientists have developed a variety of visualization systems of dynamic displays which show the behaviors of the algorithms and indicate the sequences inside the codes of the algorithms.

The VIS-AD (VISualization for Algorithm Development) system was designed by Hibbard, Dyer, and Paul [15] to help scientists visualize their computations. This system can be understood in terms of its data model, computational model, and display model. Unfortunately, this visualization system focuses on the visual programming paradigm in an algorithm oriented way. Data itself cannot be accessed by the user directly, but exists only internally.

Shen's TBDSV (Tree-Based Data Structures Visualization) system [28] is built on the X windows system. The TBDSV system simulated AVL tree, Red-Black tree, B-tree, and Splay tree, but it cannot accept the user's input for visualization. In his system, there is no deletion operation available.

Furthermore, researchers have realized that human perception also depends on the other four senses besides the vision sense exploited by graphs, texts, and animations. The exploitation of the other human senses will be a great benefit for presenting and comprehending complex information. For example, Bly [7] found that graphics plus sound annotation were more effective than just graphics. With the development of sound hardware and software, it is now possible to use more sound and clearer sound in the visualization system which can be called "Multimedia Visualization System" since microphone input of voice annotation is a common feature of personal computers. The

Multimedia Visualization System is a useful tool both for interpreting image data fed into a computer and for generating images for complex multi-dimensional data sets. It will help users comprehend the task of an algorithm and explore different scenes in the construction of the data structures. Therefore, this system is appropriate for lab exercises and distance learning as a teaching and research tool.

Explicitly or implicitly, visualization systems are also based on a display model in which data and information are communicated to users [27]. Clearly, there is a need for implementations of interface to complex data structures. Director 6 Multimedia Studio is a great platform as a display model to implement the visualization system for my project.

2.3 Multimedia

Multimedia is a collection of various media, such as animation, sound, graphics, text, photography, and video, which come together to constitute a singular form of communication [19]. It effectively creates a sequence of events that will communicate an idea usually with both sound and visual support. Typically, multimedia productions are developed and controlled by computers [9].

In the early days of multimedia, the presentation depended heavily on skillfully designed support devices, such as drawings, charts, graphics, and actual products. Nowadays, the art of multimedia is still the same. However, the method of multimedia and the new support products have ushered in a new era of sight and sound that adds a variety of stimulus to contemporary presentations. Table 2.1 shows that personal computers have evolved gradually to include multimedia capabilities.

Multimedia, like the computer itself, is a tool. This tool will be helpful in the promotion of ideas, concepts, and services for developers. There are no limits for its use. Multimedia processes are especially beneficial in the education area.

Years	multimedia capabilities
1984-1987	Simple; Slide Shows; Basic interactivity
1988-1992	Faster Processors; Sophisticated graphical interface; Interactive environments grow
1993-1999	Compact disk quality sound; 3-D animation; Presentation software enhanced; Sophisticated authoring environments; Connection to externals; Cross platform development

Table 2.1 Personal Computers Have Evolved Gradually to Include Multimedia Capabilities

Multimedia visualization systems allow their users to learn faster as well as achieve better recall results. With computer-based learning techniques, the individual can move at his own pace. In addition, a computer-based learning program has the ability to change lessons and data for certain levels of staff learning. Another advantage of these systems is that the content can include a variety of multimedia elements. The integration of sound and visualization allows the learning technology to be a highly effective medium. Such processes are particularly successful in the area of flight and driving simulators [13].

There are many areas to consider when a multimedia project is planned. A few are the hardware platform, the audience or user, and the design process. Perhaps the most important event to consider is the choice of software. Software is like the engine of a car; it takes you where you want to go. Having a software program that provides a workable multimedia authoring environment can determine the success of your end project. There are numerous software programs on today's market, each with their own unique features.

My project requires developing a serious interactive content that demands considerable extensibility. Therefore, I chose Macromedia Director 6 as the software program.

Director allows the development of a wide range of presentation types from the very basic to the extremely sophisticated. Director has great strength in animation and interactivity. The score windows in Director 6 allow virtually any action to happen based on information entered in the score script window (see Figure 2.1). Director also contains a complete set of painting tools to allow both the creation of a graphic object as well as the modification of imported graphics.

The study of multimedia user interfaces has not matured into an independent discipline. We need to examine a range of research contributions in disparate areas which contribute to our understanding of these new interfaces.

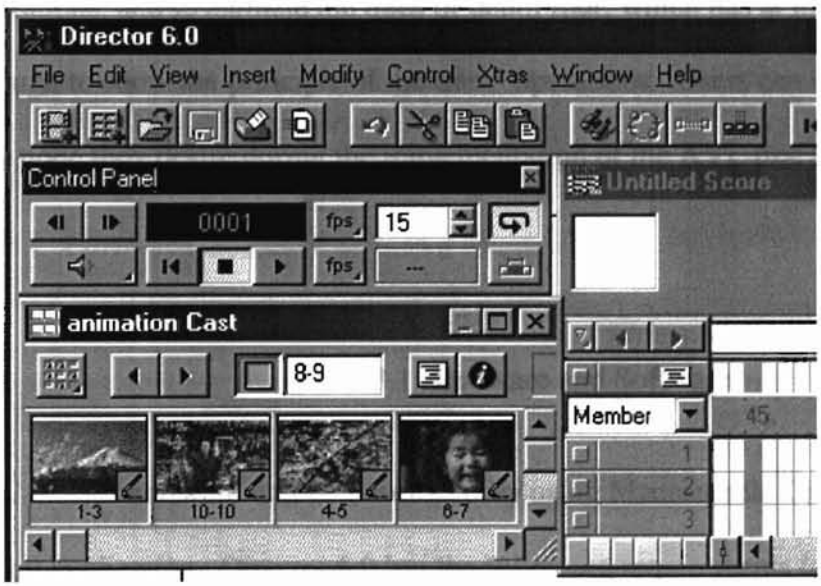

Figure 2.1 The Interface of Director 6.0

CHAPTER III

DESIGN AND IMPLEMENTATION

As stated above, the primary goal of this project is to design and develop a flexible and user-friendly software (MDSL system) for simulating the operations of data structures as a teaching and learning tool with multimedia technology. In this chapter, besides system design and implementation, we also describe the design and implementation of two main data structures movies (MultimediaAvlTree and MultimediaBSTree) in the MDSL system. In these two movies, both the AVL tree and binary search tree can be implemented to simulate insertion, deletion, and search operations, and to animate the trees of depth four, which has at most 31 nodes. This is enough to show the principle of the trees' operations. Users can insert, delete, and search any node they desire during the implementation of the AVL tree and Binary Search tree.

3.1 Hardware and Software

This software will be developed by using Macromedia Director 6 and will be a 32-bit application. Director 6.0 has the following requirements for the system [19]:

Windows System Requirements:

- 486 DX, SX or greater

- Windows 95/98, Windows NT 4.0, or later Direct 3D recommended
- 16 MB of available RAM
- 80 MB free hard disk space for installation
- 640×480(13-inch) resolution and 8-bit color (256 colors) monitor or higher
- 8-bit or 16-bit sound card

Macintosh System Requirements:

- Power Macintosh
- System 7.5 or later with Quick Time extension. QuickDraw 3D recommended
- 16 MB of available RAM
- 80 MB free hard disk space for installation
- 640×480(13-inch) resolution and 8-bit color (256 colors) monitor or higher
- 8-bit or 16-bit sound card

Macromedia Director 6 is the world's foremost powerful authoring tool for multimedia production and for the Internet. It was introduced in March 1997. The Lingo scripting language of Director 6 can precisely control text, sound, graphics, and digital video. Lingo also has some concepts which are very similar to object-oriented

Common Term					
Lingo Script	Parent script	Child object	Property variable	Handler	Ancestor
Object-Oriented	Class	Class instance	Instance variable	Method	Super class

Table 3.1 Lingo Script versus Object-oriented Programming Language

programming language. Table 3.1 shows common terms between Lingo script and object-oriented programming language. Therefore, we can develop my project by using Director 6 on Windows or Macintosh and play back executables on the platform (such as OS/2, SGI, OS/9), over the Internet via Shockwave plug-in or Java™, or many interactive TV formats.

This multimedia system will be built and run on the Microsoft Windows 95/98 or NT operating system, which have good software reusability, maintainability and accessibility. These provides a smooth and flexible open system user interface at a time that users are accustomed to window-style interfaces. This system can also run under Windows 95/98/NT and MacOS.

3.2 System Design

The operation of the multimedia system can be of a supportive nature, allowing the designer to concentrate on content, and not the technical aspects of presenting. A good feature of software is its capability to be used easily. Toward the goal of building a user-friendly and reliable MDSL system, this section will discuss some design strategies and implementation decisions which I have considered and made during the process of designing this system.

3.2.1 Design Strategies

People remember 20% of what they see, 40% of what they see and hear, and 70% of what they see, hear and do. This is also the basis for the learn-by-doing philosophy embedded in my thesis design. This system can accept the user's input algorithms for

multimedia visualization. That means the users can not only see and hear something but also do something when they are learning. Actually, the implementations of the MDSL system are controlled by users. The users can input the test data during the running time and see the result.

As a teaching and learning tools, user interface has become very important. Therefore this system is developed in windows environment and on the Macromedia Director 6.0 platform, which have more benefits for designing the system user interface. It can follow all user interface principles formulated by Sommerville [30]. That is, the user interface should use terms and concepts which anticipated users are familiar with, be appropriately consistent, include some mechanism which allows users to recover from their errors, incorporate some form of user guidance, and the user should not be surprised by the system.

Color can enrich the implementations of visualization and communicate information presented to users more efficiently [15]. In my program, the current node always is red and other nodes are all black. When the current node moves and animates in the stage, the red current node can vividly show the operations of data structures in order to hold audiences' attention.

Audio can improve a multimedia presentation in many ways, but the most important way is that it is used to enhance or augment the presentation of information and instruction. In my program, sound effects have been merged into the program from a different channel controlled by Lingo script. These sound effects (see Table 3.1) are particularly related to programs that have an instructional message. For example, when the user inserts, deletes, or searches a node from a data structure, there is a voice message

which says " You inserted, deleted, or searched a node". If the user performs some illegal operations, such as trying to search a node when there is no such node in the tree, there will be an alarm dialog box which pops up with a text message, a system beep and a voice message which says "This node is not in the tree, please try again".

Sound file name	Sound contents
StartWelcome	Welcome to multimedia data structures learning system. Please click one of the movies in main menu, and enjoy it.
Inputdata	Sorry, please click input data field and enter a data.
Instruction	Please read this instruction.
outStage	Oops, out of the stage, please try again.
treeful	Oops, this tree is full.
Insert	You insert a new node.
Delete	You delete a node.
Search	You search a node.
Nodel	This node is not in the tree, please try again.
Noinsert	This node is already in the tree.
Slrotate	Need single rotation with left.
Srrotate	Need single rotation with right.
Dlrotate	Need double rotation with left.
Drrotate	Need double rotation with right.

Table 3.2 Sound Files

Animation offers the temporal juxtapositions that graphics lack. Unlike print or graphics, animation is a dynamic medium. We get a sense of relative timing, position, direction, and speed of action. We need no captions, because the message is conveyed by motion and image[25]. Therefore, we can get the most impressive smooth motion with the animation technique.

People are always looking for new ways to educate their children. If they are having fun, they learn better. Computer animation can be used to make exciting and fun movies into which education can be easily incorporated. It is much more interesting to learn math, for example, when the letters are nice, colorful, and flying on your TV screen.

You don't need to solve problems on plain black and white paper. Actually, there is much more to animation than fun. Animation has grown from being purely an entertainment medium to being one of the most powerful ways to get your point across. Whether you aim to deliver complex visual information or simply to keep the viewer's attention, animation is truly a powerful medium. Another reason for using computer animation to simulate events as opposed to models is that variables can be programmed into a computer and then very easily changed with a stroke of a button or select a menu item in the interface.

In my project, the operations of data structures are implemented by using animation techniques. For example, in MultimediaAvlTree movie, animation techniques are used in the movement of the nodes in the tree. That is, animation has been used to show the user the tree's operations and rotations. The speed actions of animation are controlled by the "wait" handler in Lingo Script (see Appendix A), which has 18 speed grades. Since different computers might have different speeds, the users can simply click the SpeedUp and SpeedDown buttons on the stage to adjust the animation speed in the running time, which makes the system more flexible and user-friendly.

3.2.2 Implementation

Since the MDSL system is developed by using Macromedia Director 6.0, we first have to know Director 6.0 Control items [20] [21] [22] shown in Table 3.3.

The MDSL system consists of two main windows: System Welcome Window and System Main Menu Window.

Figure 3.1 shows " System Welcome Window". When users run this system, the

system first shows this window. With beautiful music, this window will gradually

Control item	Functions
Paint window	provides the same tools in a paint application such as Microsoft paint, supports Photoshop filters and new tweenable filters for graphic effects, and be used to create and edit the user interface.
Cast window	a multimedia database of graphics, text, sound effects, music and Lingo scripts, and contains all the information in a movie.
Score window	keeps track of each cast member on the stage in each frame of a movie and controls tempo and the timing of sounds, transitions, and palette changes.
Control panel	provides a set of controls similar to those on a VCR. The user can use them to play, stop, or rewind a movie.
Stage window	Stage in which movie appears. It is always open.
Sound control	The user can import the sounds and music into a movie and can control it with Lingo script language or a temp setting.
Lingo script	director's scripting language that adds interactivity to the multimedia project. It can combine animation and sound in ways that score alone cannot.

Table 3.3 Director 6 Main Control Items and Their Functions

move from the center to the top of stage. After this window disappears on the top of stage, the system starts to run sound file "startWelcome" (see Table 3.2). The sound explanation that is "Welcome to Multimedia Data Structures Learning System. Please click one of the movies in the main menu, and enjoy it" will speak out from the computer microphone. Accompanying this sound explanation, next window "System Main Menu Window" (shown in Figure 3.2) appears in the center of stage. And the data structures movies appear on the stage one by one. This time, users can select one of these movies to play just according to the instruction of this sound explanation.

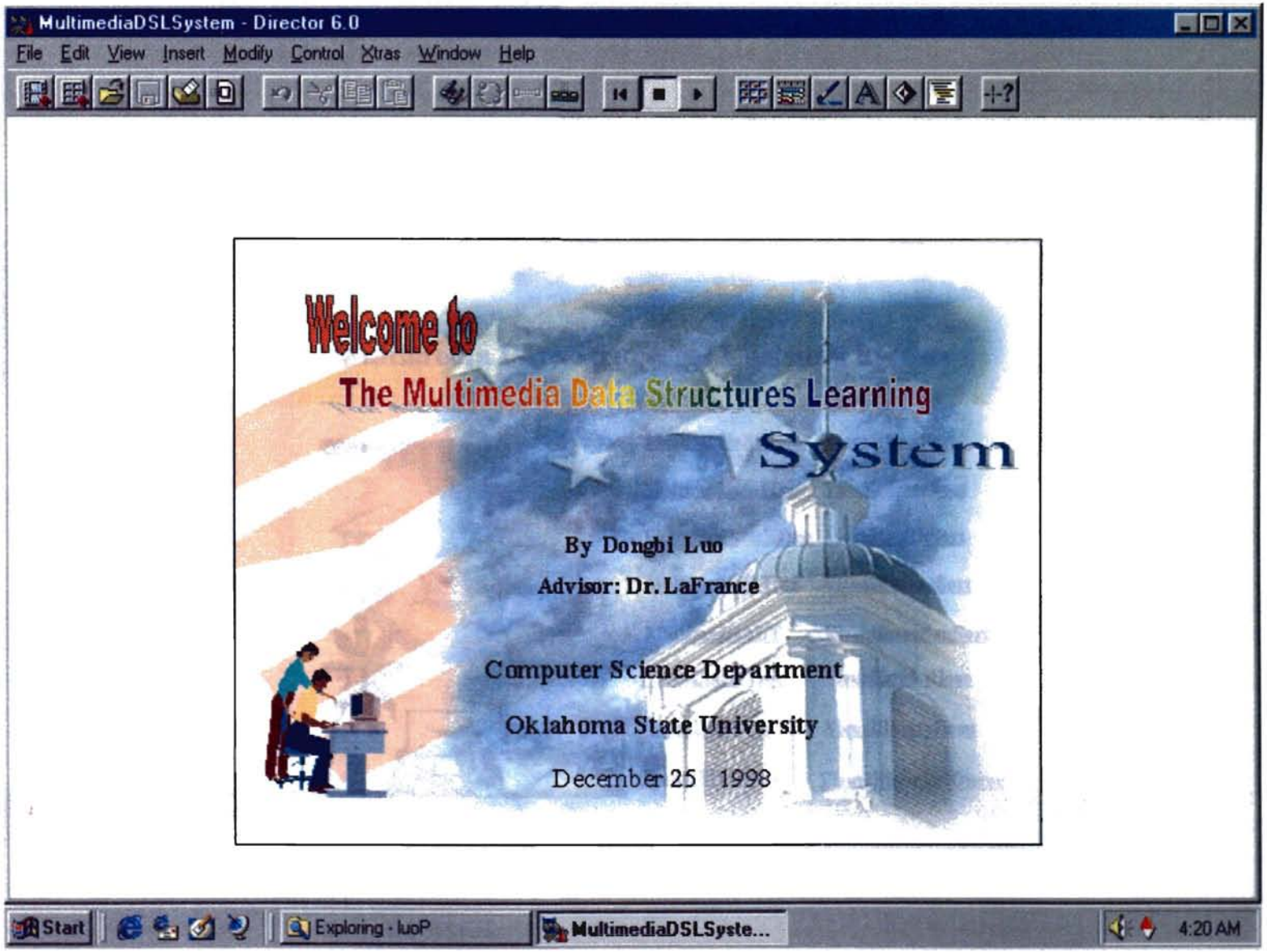

Figure 3.1 System Welcome Window

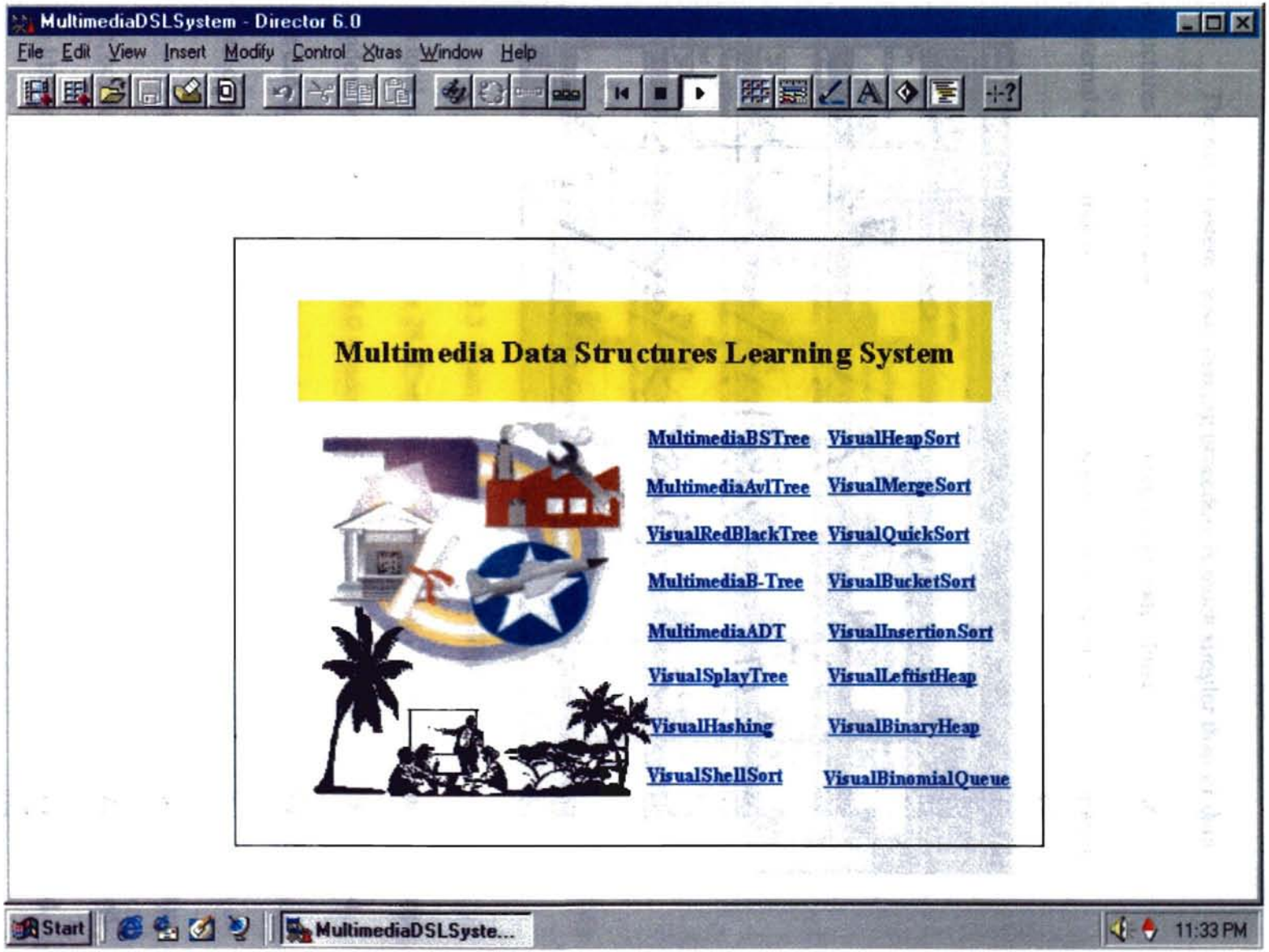

Figure 3.2 System Main Menu Window

3.2.3 System Movie Making Procedure

The main system movie making procedure is much simpler than its data structures' movies because of its simple Lingo script code. First, we should do the "Internal Cast" window of this movie, which shown in Figure 3.3. We can see that

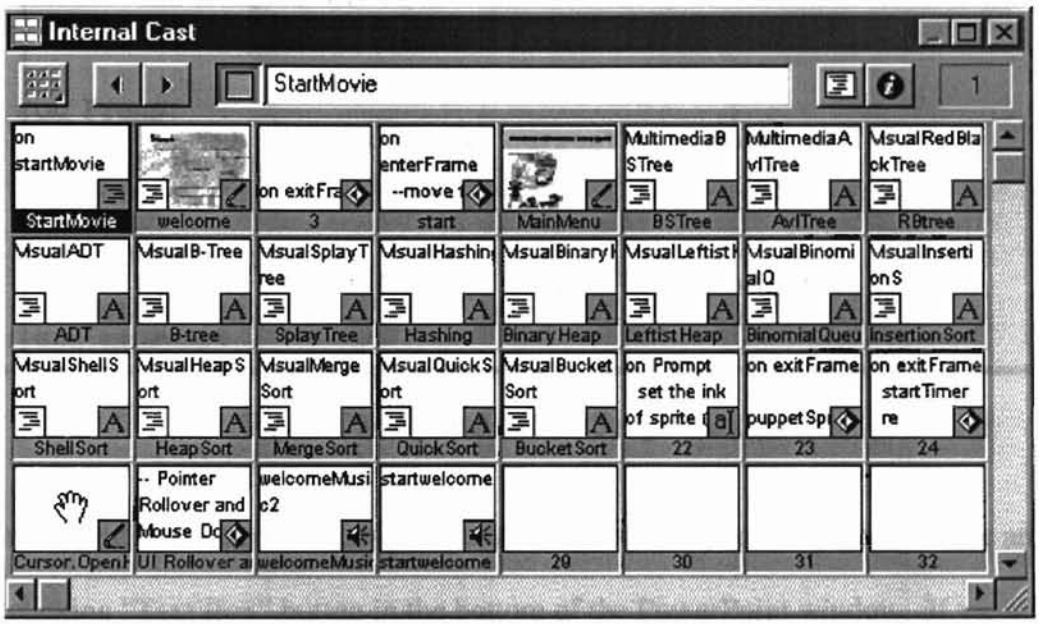

Figure 3.3 Main Movie Internal Cast Window

the total cast number used is 28. Cast 1 is the movie script. Click it and click the script window button in Director 6.0 window. The script window will pop up as Figure 3.4. We can directly code the movie script in this window, which is shown in Figure 3.4. Then close the script window and save it. Cast 1 is done.

Cast 2 is a picture file. It is made by using MS PowerPoint. Save it as a picture file, copy, and paste to the Paint window. Then import it into cast 2. The picture used the library tower of Oklahoma State University and the American flag as background. Type "Welcome to Multimedia Data Structures Learning System" by using "wordArt..."

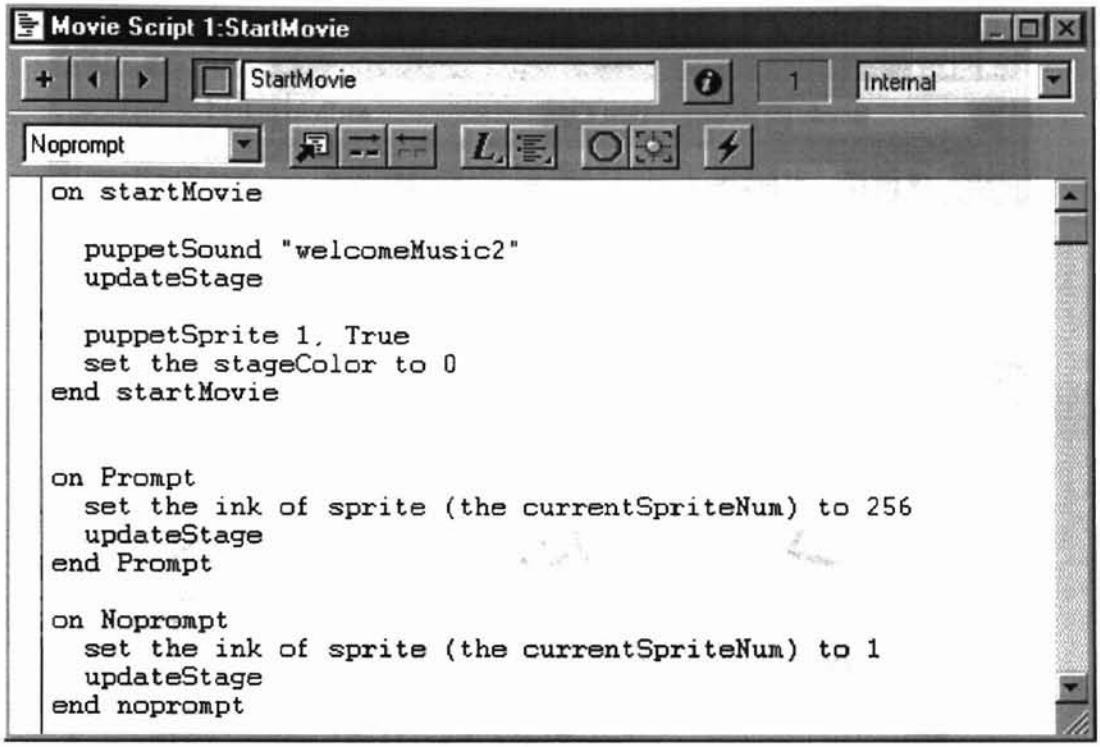

Figure 3.4 System Movie Script Window

from PowerPoint insert menu > picture. And type author, adviser, school name and date by clicking "Text Box" button in the bottom of the PowerPoint window. If we use Director Paint window or MS word window to make this picture, the text box cannot be the same color of picture as the background. This is why we use MS PowerPoint to make cast 2. In the same way, we can make cast 5 which is another picture file.

Cast 6~21 are "Text" messages on the stage, which can connect to data structures movies. For example, when you make cast 7 (named AvlTree), first click cast 7, and click text window button in the Director 6.0 window. The text window will pop up as Figure 3.5. We can type text content (MultimediaAvlTree) in the text window. Then click the Cast Number Script button in the text window. There is a script of cast member 7: AvlTree window will pop up as shown in Figure 3.6. We can code cast script in this window as shown in Figure 3.6. Then close these windows and save them.

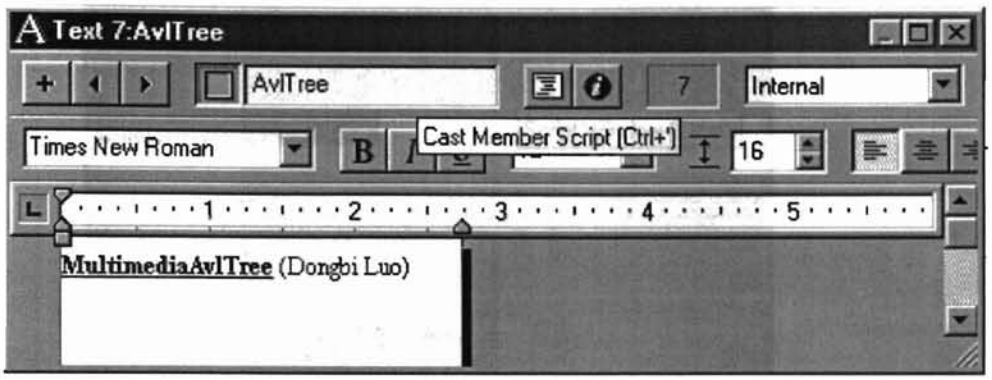

Figure 3.5 System Text 7: AvlTree Window

Figure 3.6 System Script of Cast Member 7: AvlTree Window

Cast 22~24 are all score behavior. If you click one of them, the Behavior Inspector window will show up as Figure 3.7. When we click the script window in this window, there will be a score script window pop up as Figure 3.8. We can see this time consumer score behavior from Lingo script. They are put into the script channel in the score window shown as Figure 3.9. The script channel stores behaviors or instructions written in Lingo that are executed when the movie reaches a particular frame.

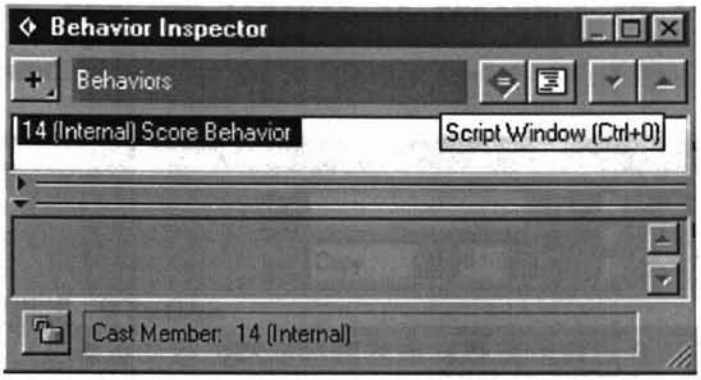

Figure 3.7 System Score Behavior Window

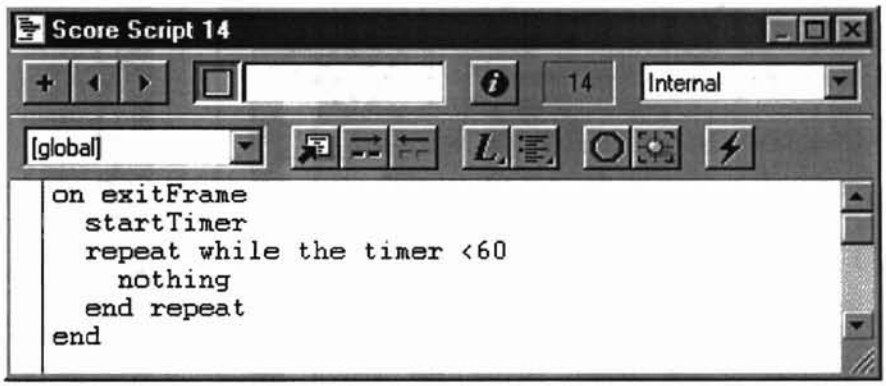

Figure 3.8 System score Script Window

Therefore, these score behaviors are put into different frames in the script channel in order to control every data structures movie entering the stage at a different time. When the playback head reaches frame 25, the movie will loop here because Lingo in the score script window is:

```
on exitFrame
   go to the frame
   puppetSprite 1, False
   set the visible of sprite 1 to False
   puppetSprite 2, True

   set the soundEnabled to False

   --set the main menu to the window
   set the locV of sprite 2 to 196
   set the locH of sprite 2 to 260
```

```
    set the visible of sprite 2 to True
    updateStage
end
```

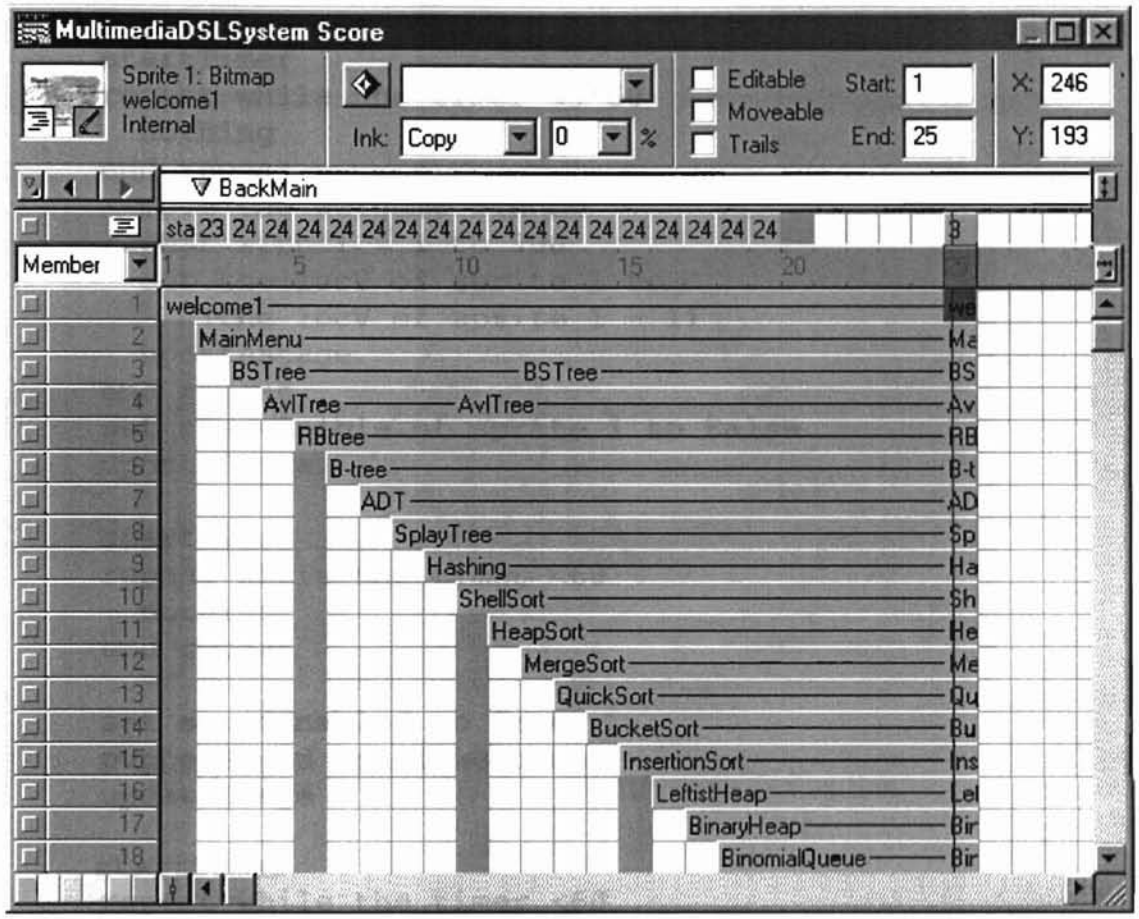

Figure 3.9 System Score Window

This handler names exitFrame because it is executed when the playback head will exit frame 25. Obviously, the first line Lingo "go to the frame" causes the movie loop in the frame 25.

When the movie starts, the playback head is in frame 1. The Lingo script stored in the script channel of frame 1 is:

```
on enterFrame
  --move the welcome window and instruction window
  --into or off the stage
  puppetSprite 1, True
  set the visible of sprite 2 to False
```

```
  set the locV of sprite 1 to 190
  set the locH of sprite 1 to 245
  set the visible of sprite 1 to True
  updateStage

  startTimer
  repeat while the timer <3*60
    nothing
  end repeat
  --move the welcome window
  repeat with i = 1 to 300
    set the locV of sprite 1 to
       (the locV of sprite 1 - 1)
    updateStage
  end repeat
  set the visible of sprite 1 to False
  updateStage

  startTimer
  repeat while the timer <60
    nothing
  end repeat

  set soundEnabled to True
  puppetSound "startwelcome"
  updateStage

  startTimer
  repeat while the timer <60
    nothing
  end repeat
end

on exitFrame
  puppetSprite 1, FALSE
  set the visible of sprite 1 to False
end
```

This phrase Lingo controls the "System welcome window" to display, move, and sound explanation.

In addition, we have used the behavior of Cursor Rollover and Mouse Down in the System Main Menu Window that cursor changes from Arrow to Hand when the user's mouse cursor points to the text box of one of data structures movies. This cursor

change benefits for reminding user that the pointed data structure movie will be selected to implement if he clicks mouse. In order to do so, we need to select the Behavior Library from Director menu item Xtras. The Behavior Library Cast window will pop up. Drag cast 47 (behavior of pointer rollover and mouse down) into system cast 26 (see

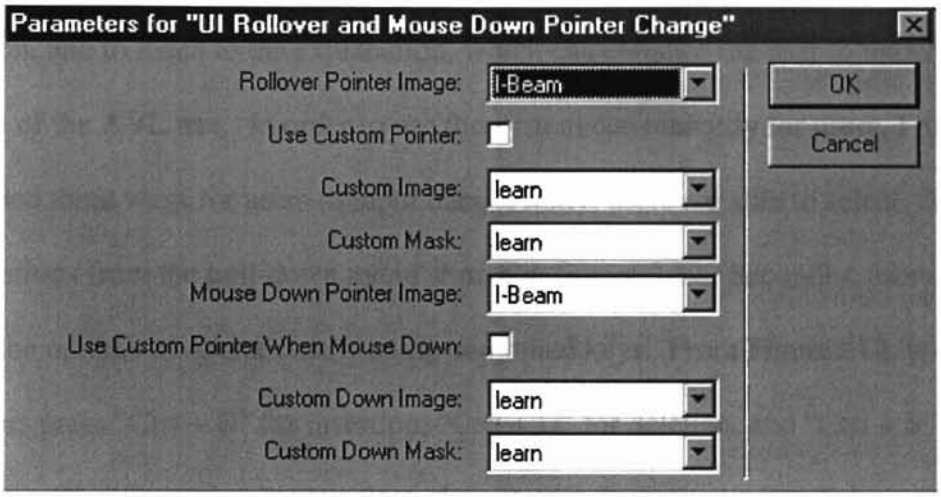

Figure 3.10 Change Rollover Pointer Image Dialog Box

Figure 3.3). Then drag system cast 26 to every text box of data structures movies in the stage. When we drag system cast 26 to one of these text boxes, there is a dialog box that will pop up as shown in Figure 3.10. Click first pull-down arrow and select "Hand" item instead of "I-Beam" item. Click OK. This behavior has been attached to this text box. When the user points to this text box, the cursor will change from arrow to hand image.

3.3 Design MultimediaAvlTree Movie

MultimediaAvlTree is one of the main data structures movies in the MDSL system. The user can implement the AVL tree by selecting MultimediaAvlTree from the main menu (see Figure 3.2). In the meantime, an empty data structure of the AVL tree

is created automatically by calling the newTree handler in startMovie (see Appendix A: Lingo source code) script. The user can work on the AVL tree interface shown in Figure 3.11. Then the user is able to select any one of the operations of the AVL tree. But the user has to select the insertion operation first for the empty tree. The user is also able to input the data at running time, to execute an operation, to watch the animation of the operation, and to listen to the explanation, which can enhance the user to understand the concept of the AVL tree. In order to use the system conveniently for users, I have developed three ways for users to input data. Firstly, users are able to select the options of operations from the pull-down menu shown in Figure 3.12. Secondly, users are able to select the options of operations by using combined keys. From Figure3.12, you can see that users press "Ctrl + D" for insertion, "Ctrl + D" for deletion, and "Ctrl + S" for searching. Thirdly, users able to select the options of operations by clicking the buttons in the AVL tree interface (see Figure 3.11).

Besides the "Operations" menu item, there are three other menu items which are "File", "Shows", "Help", and "Speed" shown in Figure 3.13, 3.14, 3.15, and 3.16 respectively.

From Figure 3.14, we can see that there are five options in the "Shows" of the pull-down menu item. If users select the option " Operations History" in the running time, the "Operations history" text box and "Close" button will display on the stage (see Figure 3.11). They will disappear if users click the "Close" button. The other four options in the menu "Shows" are all rotation patterns which will display on the stage to show users clearly how the rotation could be executed when the AVL tree needs to do these rotations. These rotations patterns will cover the "Operation history" text box,

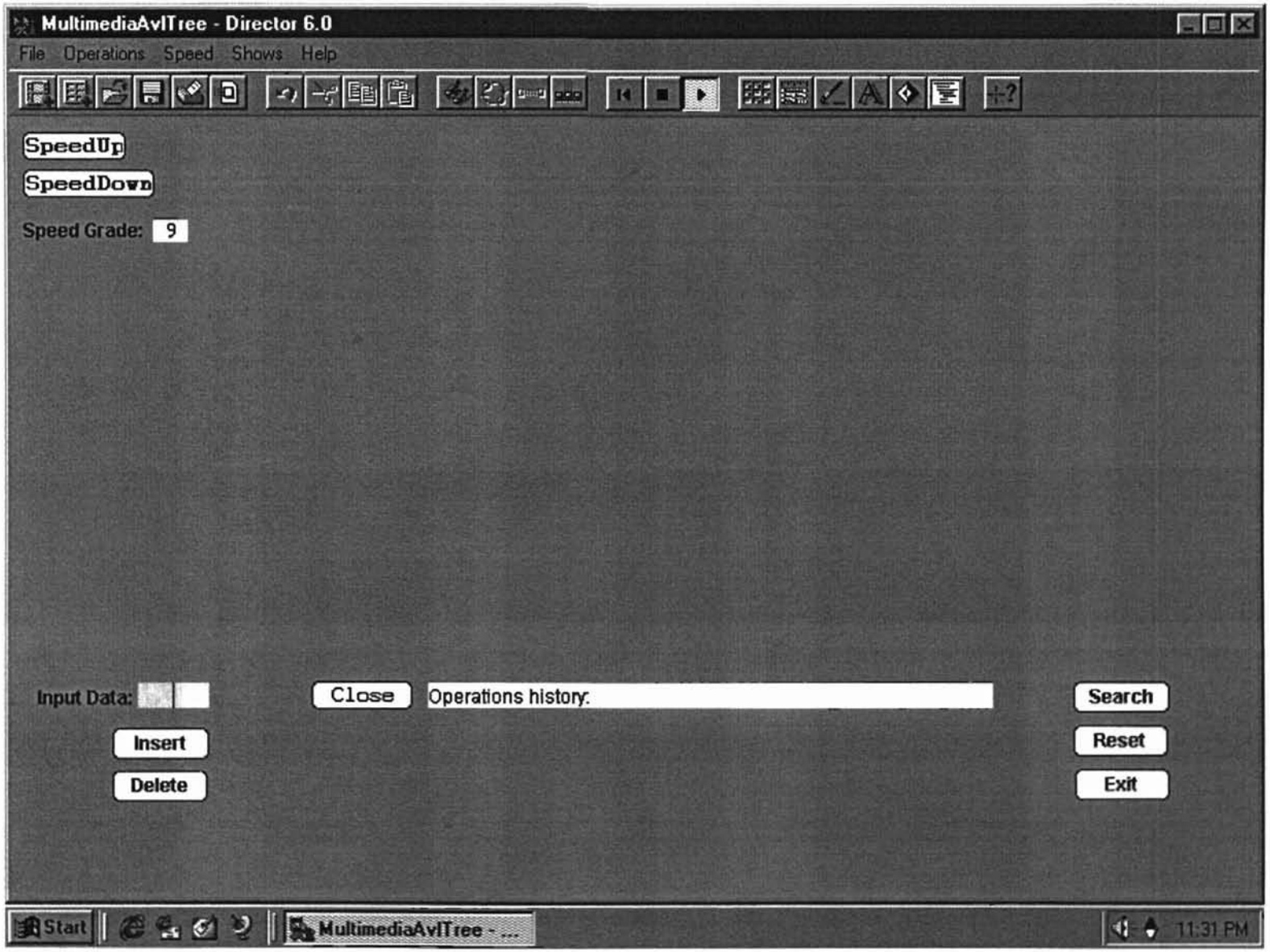

Figure 3.11 AVL Tree Interface

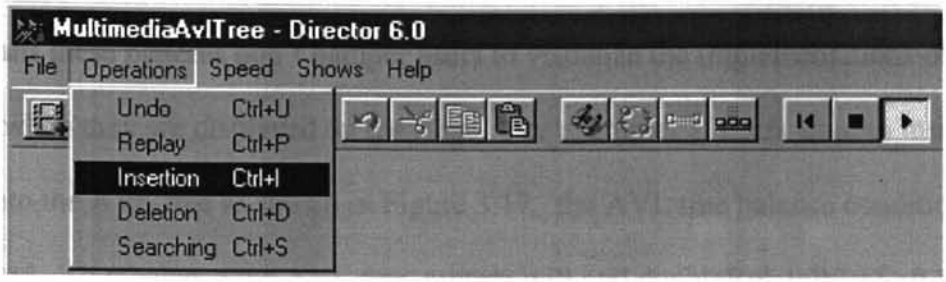

Figure 3.12 The Options of Operations Menu

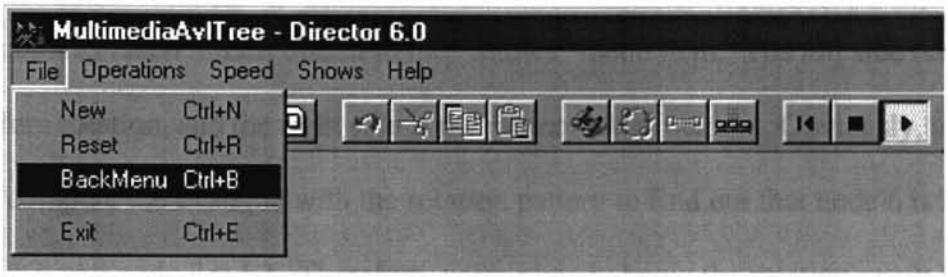

Figure 3.13 The Options of File Menu

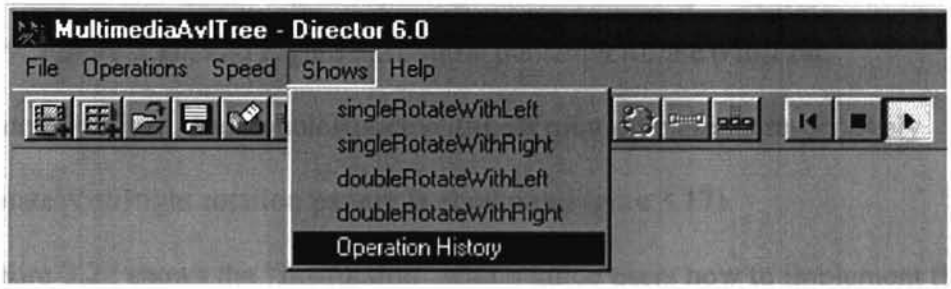

Figure 3.14 The Options of Shows Menu

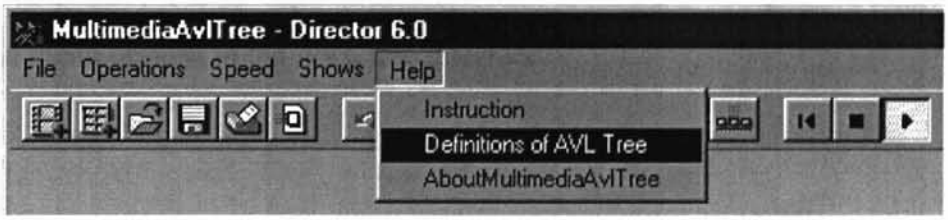

Figure 3.15 The Options of Help Menu

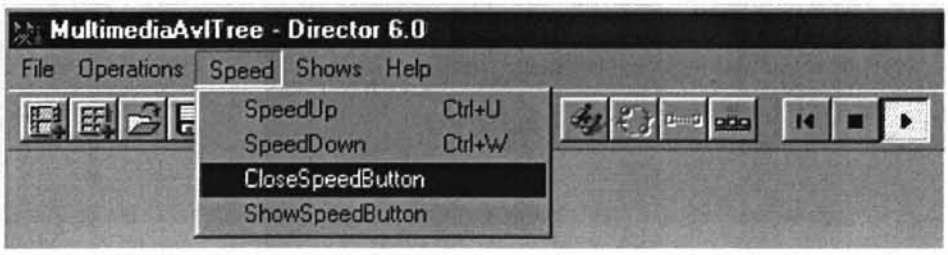

Figure 3.16 The Options of Speed Menu

which means these patterns don't hamper users to visualize the implementations of the AVL tree when they are displayed on the stage. For example, if users want to insert node 14 into the AVL tree as shown in Figure 3.17, the AVL tree balance condition will be destroyed. To rebalance the AVL tree, system will call doubleRotateWithLeft handler to do double rotation. Before the insertion, users could display this rotation pattern by clicking doubleRotateWithLeft option in the "Shows" menu item. The left side is the pattern before rotation, and the right side is the pattern after rotation. Obviously, users can easily compare the example with the rotation pattern to find out that node 6 is k1, node 15 is k3, and node 7 is k2. Therefore, users already know the rotation result from the rotation pattern. Figure 3.18 shows the result of the example after double rotation. Figure 3.19, Figure 3.20, and Figure 3.21 show the singleRotateWithLeft, singleRotateWithRight, and doubleRotateWithLeft rotation patterns respectively (doubleRotateWithRight rotation pattern is shown in Figure 3.17).

Figure 3.22 shows the "Instruction" which guide users how to implement the AVL tree movie because the "Instruction" shows all functions and properties of the buttons and menu options in the AVL tree interface.

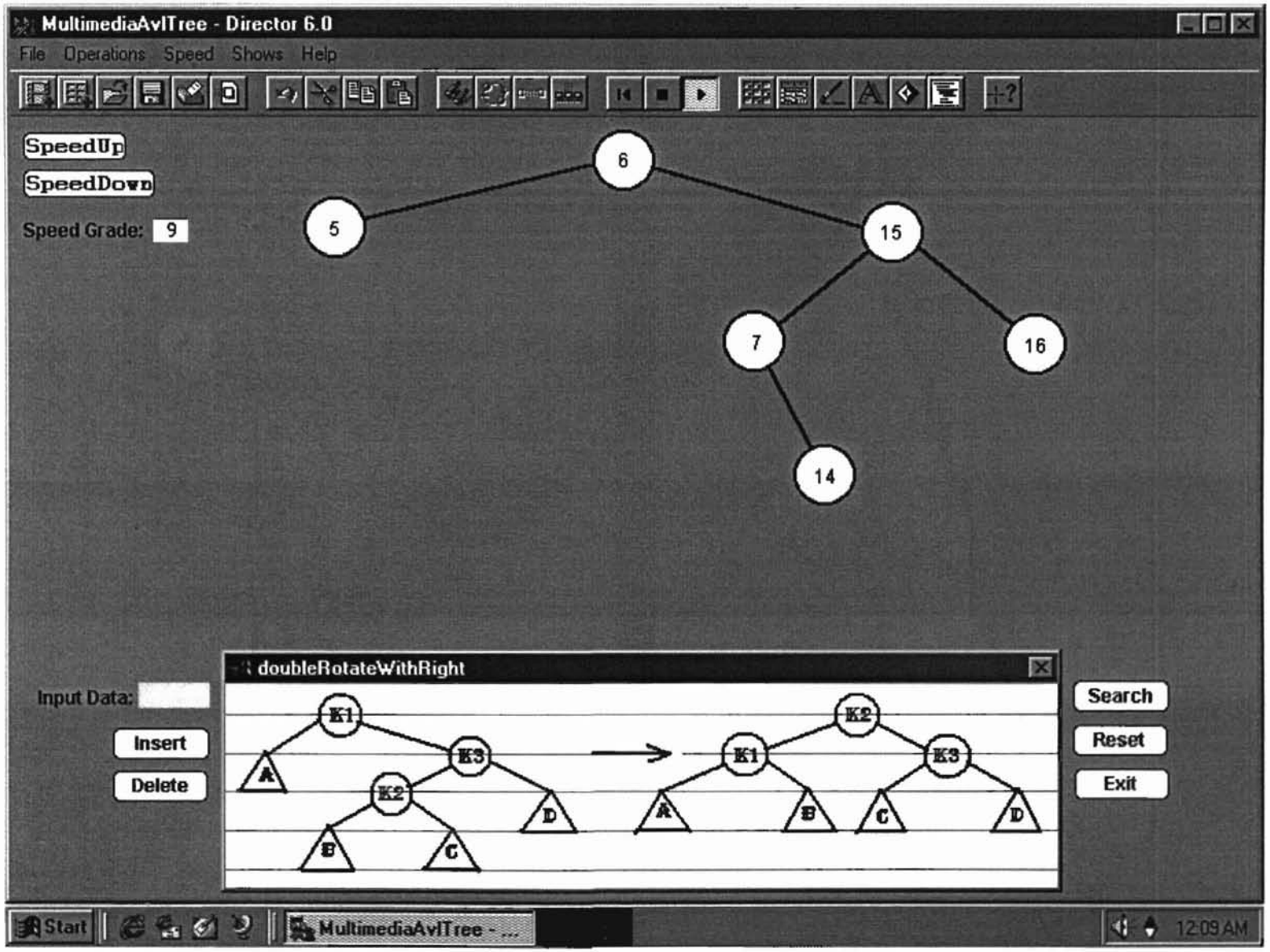

Figure 3.17 The Example before Double Rotation

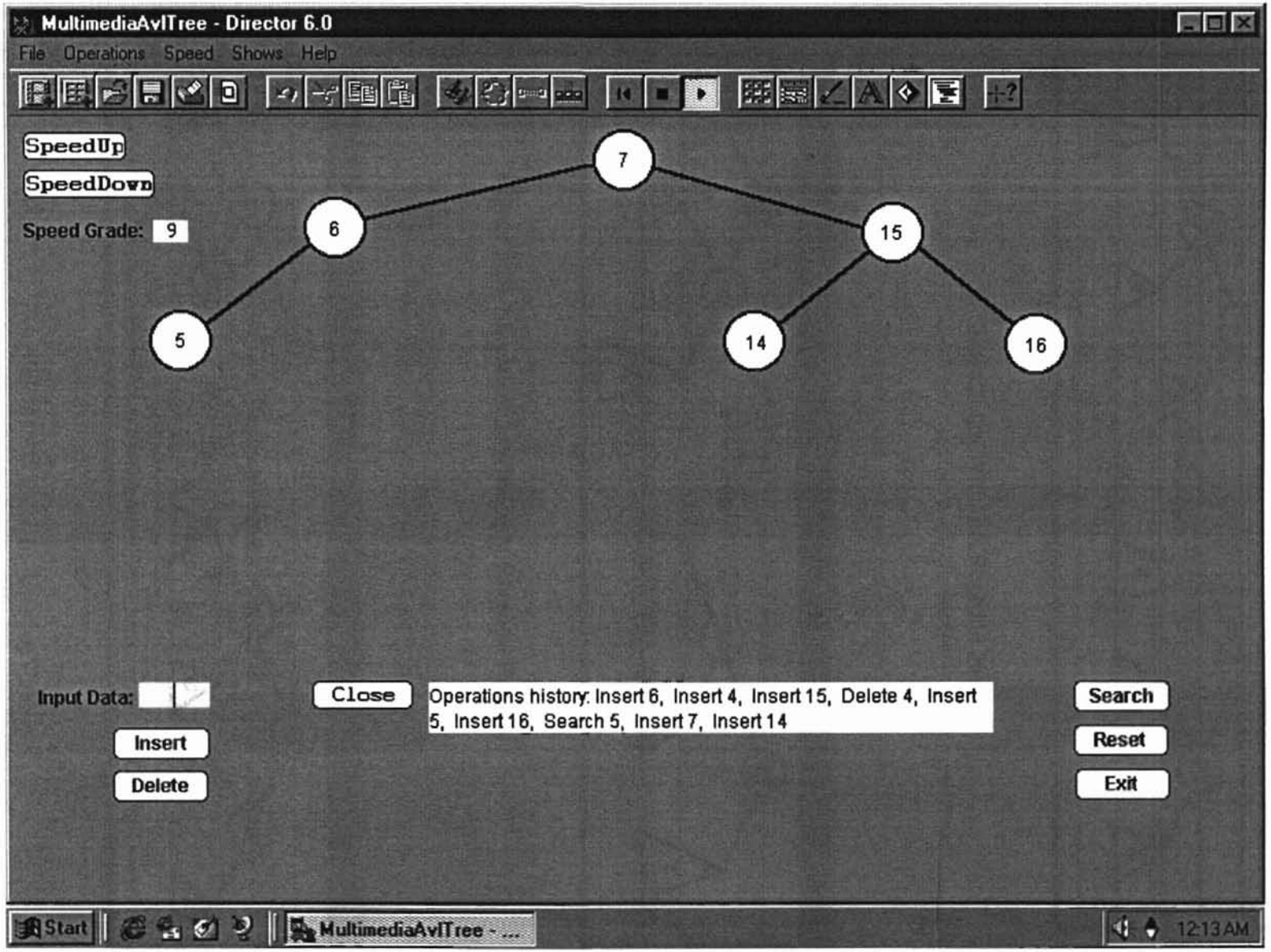

Figure 3.18 The Example after Double Rotation

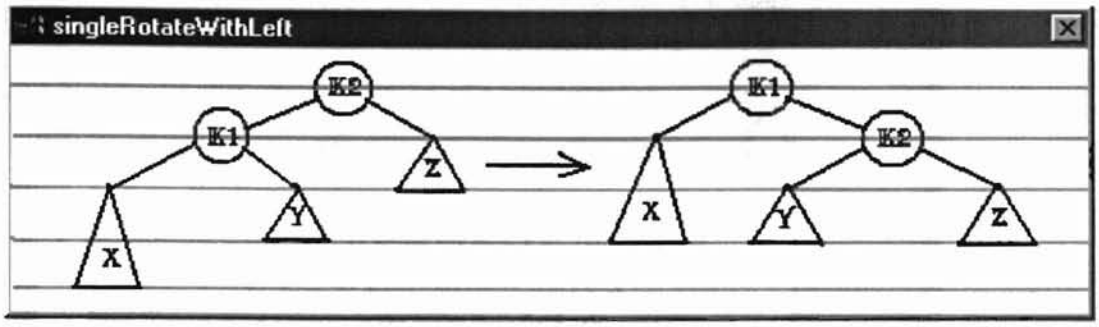

Figure 3.19 SingleRotateWithLeft Rotation Pattern

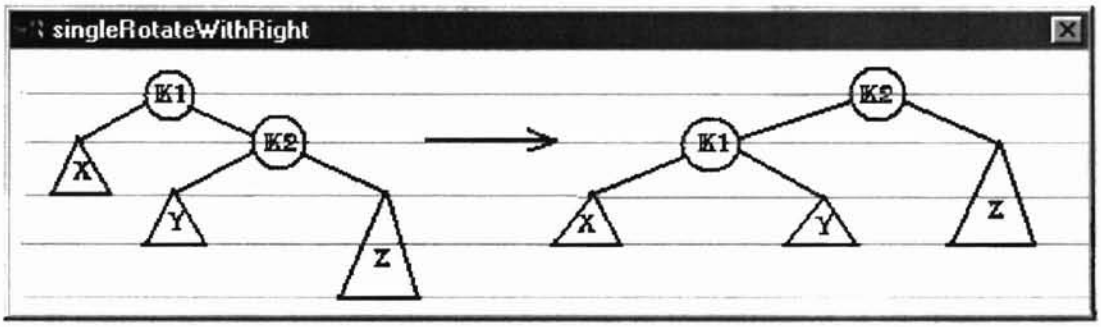

Figure 3.20 SingleRotateWithRight Rotation Pattern

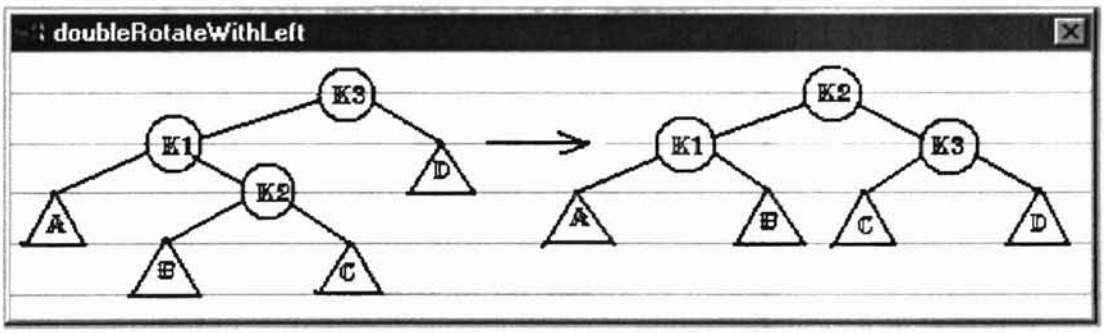

Figure 3.21 DoubleRotateWithRight Rotation Pattern

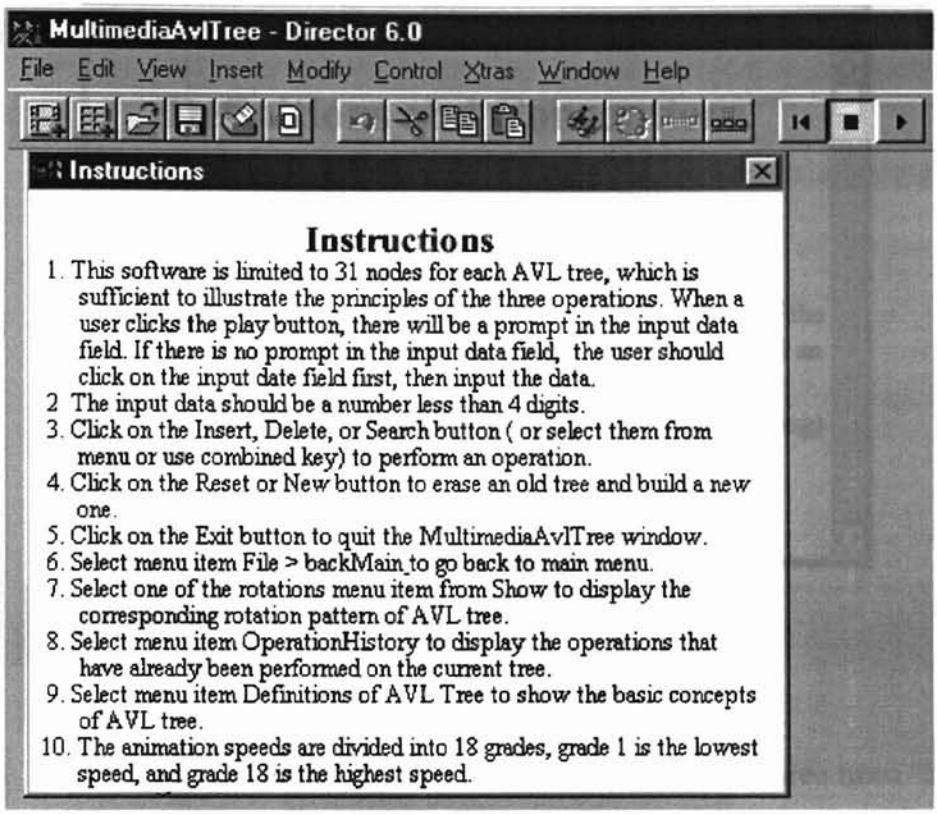

Figure 3.22 AVL Tree Instruction Window

If users select the AboutMultimediaAvlTree or Definition of AVL Tree in the "Help"menu item, Figure 3.23 or Figure 3.24 will pop up at the upper left corner of the stage.

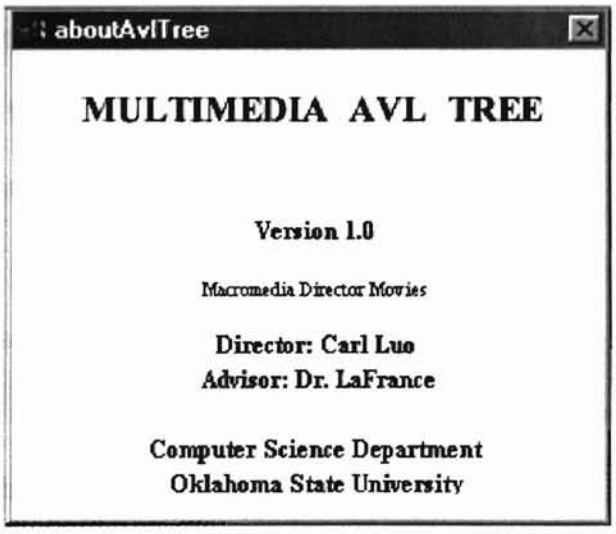

Figure 3.23 About AVL Tree Window

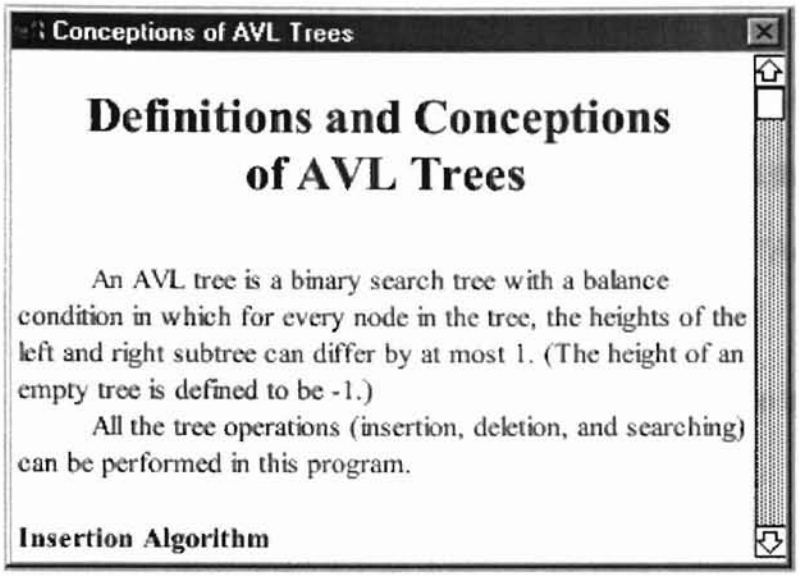

Figure 3.24 Definitions of AvlTree Window

Users can read definitions and conceptions of the AVL tree from "Definitions of AVL Tree Window" if they don't know about the AVL tree. Actually, the contents of the AVL tree definitions and conceptions are shown in Table 3.4, which are more than users can see in Figure 3.24. In order to save the "Definitions of AVL Tree Window" space and make the users read the contents of the AVL tree definitions and conceptions conveniently, I have made the scrolling bar in the right side of "Definitions of AVL Tree Window". This Window is supported by the "WinAB" movie. I can put the scrolling bar in this window as follows:

1. Type all contents of the AVL tree definitions and conceptions in the text message window of the "WinAB" movie.
2. Store this text message into the "winAB" internal cast window as a cast member.
3. Drag this cast member and put it at the corner of the upper left stage.

4. Select the internal properties from the movie menu, which is shown in Figure 3.25. There will be a "Text Cast Member Properties" window that will pop up (as shown in Figure 3.26). Click the pull-down arrow at the right side of the framing text box. Select the "Scrolling" item instead of "Adjust to Fit" item. Then click OK.
5. Adjust the side of "Definitions of AVL Tree Window" to look like Figure 3.24.

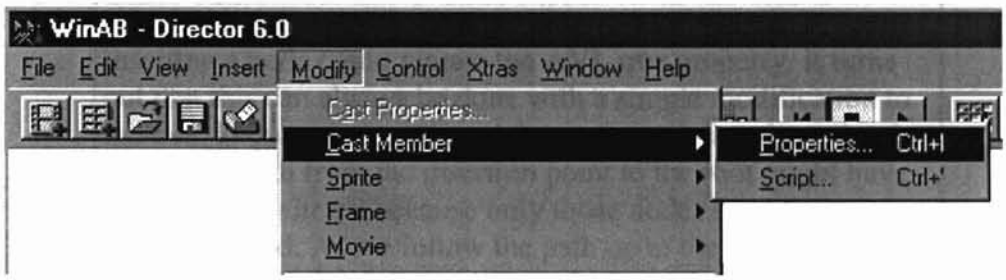

Figure 3.25 Select Movie Properties from "WinAB" Window

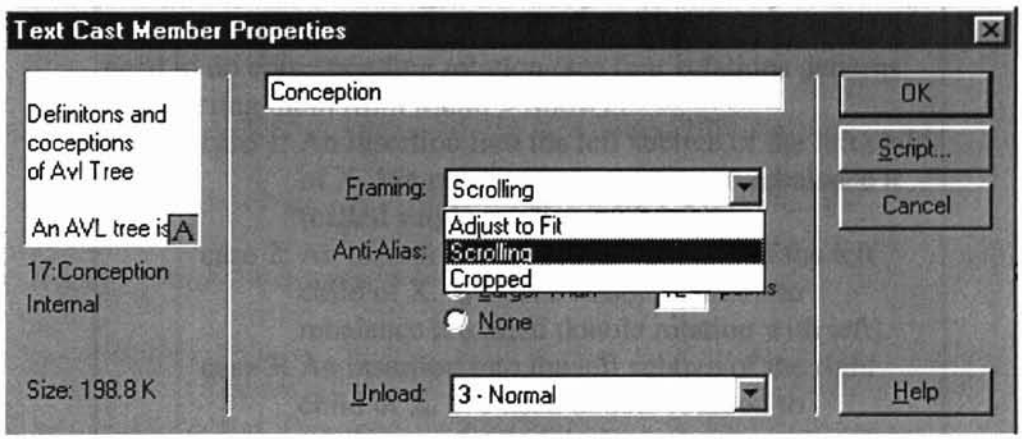

Figure 3.26 Text Cast Member Properties Window

Definitions and conceptions of AVL Tree

An AVL tree is a binary search tree with a balance condition in which for every node in the tree; the height of the left and right subtree can differ by at most 1. (The height of an empty tree is defined to be -1.)

All the tree operations (insertion, deletion, and searching) can be performed in this program.

Insertion Algorithm

When we do an insertion, we need to update all the balancing information for the nodes on the path back to the root, but the reason that insertion is potentially difficult is that inserting a node could violate the AVL tree property. It turns out that this can always be done with a simple modification to the tree, known as a rotation. After an insertion, only nodes that are on the path from the insertion point to the root might have their balance altered because only those nodes have their subtree altered. As we follow the path up to the root and update the balancing information, we may find a node whose new balance violates the AVL condition and do the rotation to rebalance the tree.

Rotation

Let us call the node X that must be rebalanced. Since the X's two subtrees' height differs by two, it is easy to see that a violation might occur in the following four cases in which we need to do corresponding rotation (see four rotations patterns by selecting them from Menu > Show) :

case 1: An insertion into the left subtree of the left child of X. We need a single rotation to rebalance it (called single rotation with left).

case 2: An insertion into the right subtree of the left child of X. We need double rotation to rebalance it (called double rotation with left).

case 3: An insertion into the left subtree of the right child of X. We need double rotation to rebalance it (called double rotation with right).

case 4: An insertion into the right subtree of the right child of X. We need single rotation to rebalance it (called single rotation with right).

Single rotation:

The pattern singleRotateWithLeft shows single rotation that fixes case 1. The before picture is on the left, and the after picture is on the right. Node k2 violates the AVL tree balance property because its left subtree is two levels deeper than its right subtree. The situation depicted is the only possible case 1 scenario that allows k2 to satisfy the AVL property before an insertion but violate it afterwards. Subtree X has grown to an extra level, causing it to be exactly two levels deeper than Z. Y cannot be at the same level as the new X because then k2 would have been out of balance before the insertion, and Y cannot be at the same level as Z because then k1 would be the first node on the path toward the root that was in violation of AVL balancing condition.

In order to ideally rebalance the tree, we would like to move X up a level and Z down a level. Note that this is actually more than the AVL property would require. To do this , we rearrange nodes into an equivalent tree as shown in the second part of this pattern.

Here is an abstract scenario: visualize the tree as being flexible. Grab the child node k1, close your eyes, and shake it, letting gravity take hold. The result is that k1 will be the new root. k2>k1, so k2 becomes the right child of k1 in the new tree. X and Y remain as the left child of k1 and right child of k2, respectively. Subtree Y, which holds items that are between k1 and k2 in the original tree, can be placed as k2's left child in the new tree and satisfy all the ordering requirements.

The pattern singleRotateWithright shows single rotation that fixes case 4 which represents a symmetric case.

Double rotation:

The pattern doubleRotateWithLeft shows double rotation that fixes case 2. In this case, it includes following two single rotations:

1. Single rotate between k1 and k2,
   k3->Left=singlerotateWithRight(k3-Left);
2. Single rotate between k3 and k2,
   return singleRotateWithLeft(k3);

The pattern doubleRotateWithRight shows double rotation that fixes case 3 which represents a symmetric case.

Deletion Algorithm

Binary search tree deletion algorithm:

case 1: If the deleted node is a leaf, it can be deleted immediately

case 2: If the deleted node only has one child, the node can be deleted after its parent adjusts a pointer to bypass the node. case 3: If the deleted node has both the left child and right child, exchange this node with the smallest node of the right subtree. Then delete the node. Deletion in AVL trees is the same as in a binary search tree, as above described. The rebalancing is as follows: First, case 3 needs one more thing. After the exchange, the deletion continues down the right subtree and eventually deletes the exchanged node. Second, after a deletion, only nodes that are on the path from the deleted node or exchange node to the root might have their balance altered because only those nodes have their subtree altered. As we follow the path up to the root and update the balancing information, we may find a node whose new balance violates the AVL condition and do the rotation to rebalance the tree.

Table 3.4 All Contents of AVL Tree Definitions and Conceptions

If users click "Exit" button on the stage, select "Exit" option from menu "File" item, or press combined keys "Ctrl + E", there will be a "Close This Movie ?" window (as show in Figure 3.27) that will pop up at the upper left corner of the stage. This

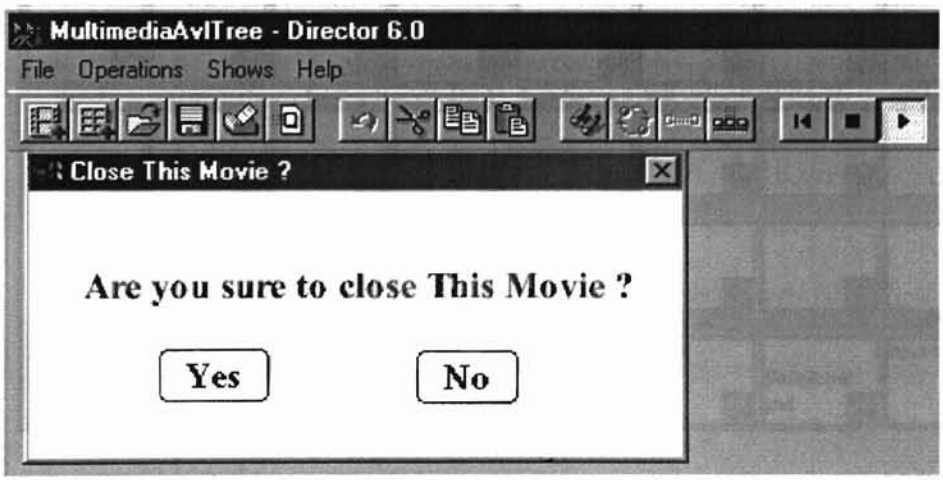

Figure 3.27 Make Sure to Close Movie Window

window wants users to make sure whether they really need to close the movie they are working on or not.

3.4 Implement AVL Tree

First, I have made the AVL Tree "Internal Cast" Window shown in Figure 3.28. Cast 3 is the "Menu" Lingo script for MultimediaAvlTree movie. Other main Lingo scripts are stored in Cast 1, Cast 4, and Cast 5 (see Appendix A). Cast 11, Cast 12, and Cast 13 represent respectively the AVL tree's node, left edge, and right edge. I have

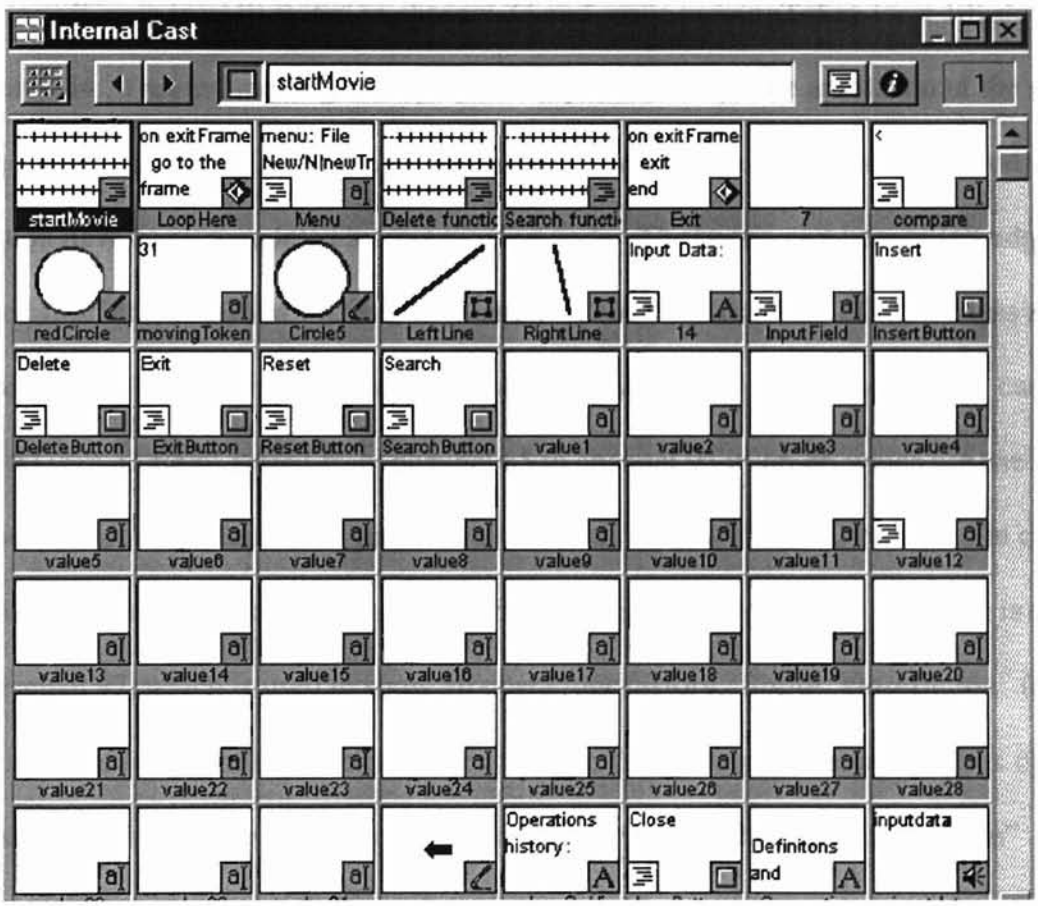

Figure 3.28 AVL Tree Internal Cast Window

dragged 31 of them into the movie stage shown in Figure 3.29. In order to code the simulation of the AVL tree operations easily, I have put all objects (called sprite in Director movie) on the stage into the first frame in the score window in order and controlled them visibly or invisibly by Lingo script when users implement the AVL tree. For example, 31 nodes are stored from channel 1 to channel 31. The compare signal (cast 8) is stored in channel 32, which is used to compare the insert node or search node with the current node. If the insert node or search node is larger than the current node, the insert node or search node will go to the right child of the current node. Otherwise, it will go to the left child of the current node. The moving node is made up of Sprite redCircle (cast 9) stored in channel 33 and sprite movingToken (cast 10) stored in channel 34. This red moving node plays an important role in the animation of the AVL tree operations. Actually, all movements of the insertion, deletion, searching, and rotation have been completed by this moving node. Thirty one numbers of left edges are stored from channel 35 to channel 49 in order. Thirty one numbers of right edges are stored from channel 50 to channel 64 in order. Sprites "value 1" ~ "value 31" (cast 21~ cast 51) are all fields stored from channel 65 to channel 95 in order, which are used to show the values for each node respectively. Sprite "Arrow" (cast 32) is stored in channel 96, which is used to show searched node flashily. Sprite "showOpHi" (cast 53) is stored in channel 97, which is used to show all operations history. Sprite "closeButton" (cast 54) is stored in channel 98, which is used to close the operations history text box on the stage. Cast 14 is a text type cast stored in channel 99, which has "Input Data" content. Sprite "inputField" (cast 15) is stored in channel 100, which is used to show input data on the stage. Sprites "insertButton" (cast 16), "deleteButton" (cast 17),

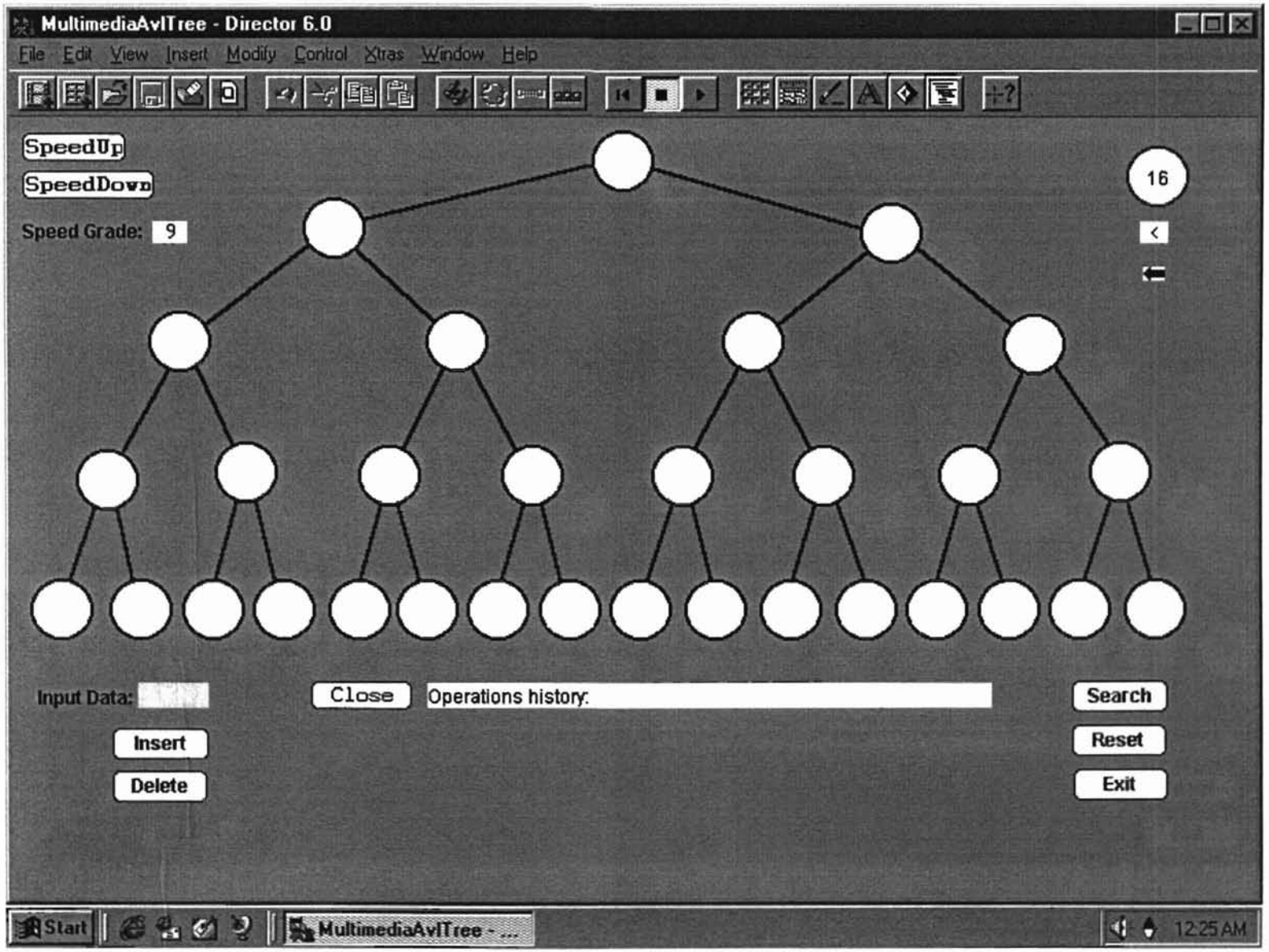

Figure 3.29 All Objects on the AVL Tree Movie Stage

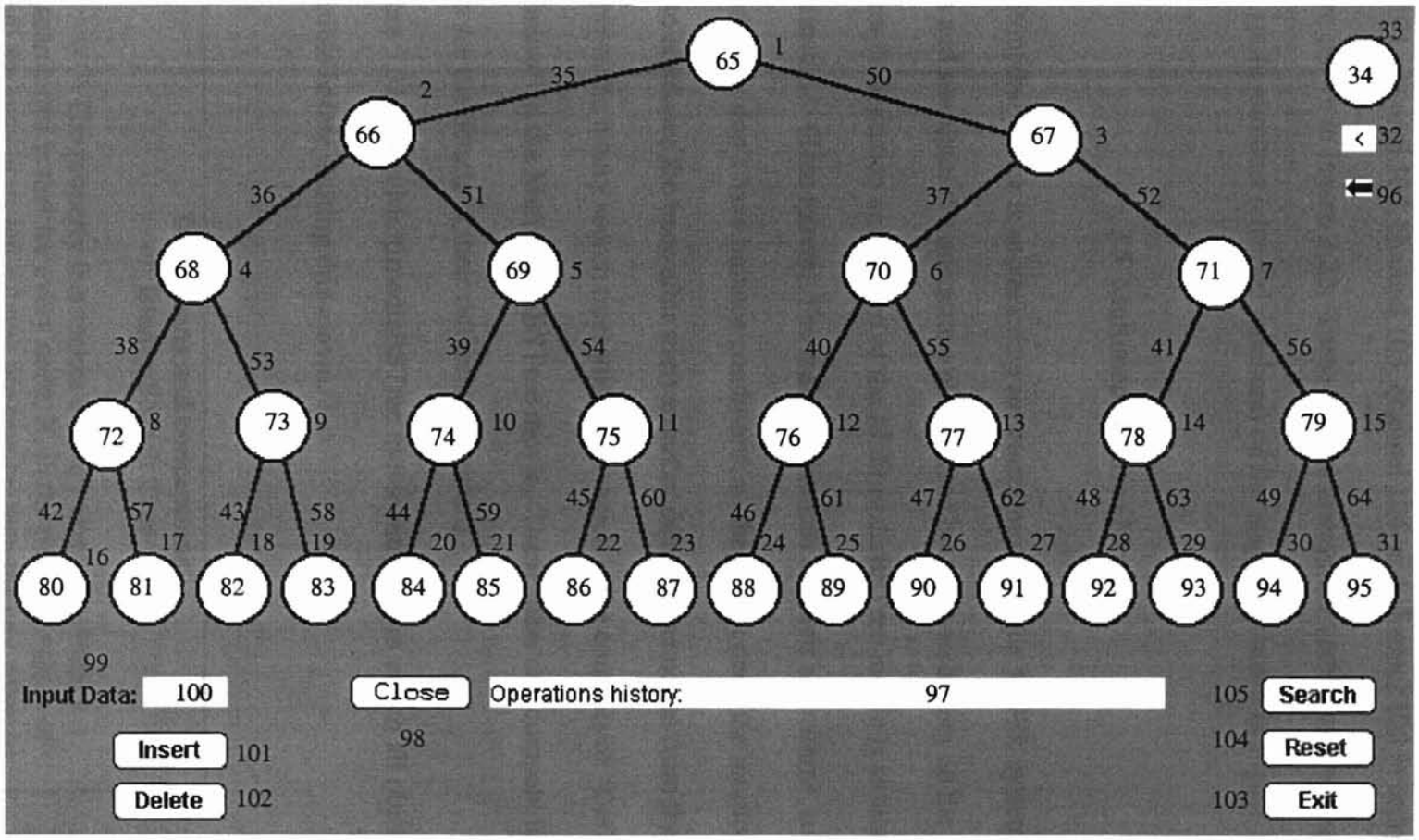

Figure 3.30 The Sprite number of the objects on the Stage

"ExitButton" (cast 18), "ResetButton" (cast 19), and "searchButton" (cast 20) are stored in channel 101, channel 102, channel 103, channel 104, and channel 105 in order, whose functions are shown in Figure 3.22. These buttons always are visible on the stage. In general, all sprite numbers (channel numbers) on the stage are shown in Figure 3.30.

3.5 MultimediaBSTree Movie

MultimediaBSTree is another data structures movie in the MDSL system. It implements and simulates the operations of animated binary search trees on the Director stage. The implementation and design of the MultimediaBSTree movie is similar to that of the MultimediaAvlTree movie. The main differences are their algorithms, such as binary search trees don't have balance conditions, so they don't need the rotation operations to rebalance the trees after users insert or delete some nodes from the trees. Table 3.5 shows the binary search trees algorithm which is the contents of "Definitions of BSTree Window" in the MultimediaBSTree movie. The readers can compare it with Table 3.4 to see the details of their different algorithms.

Figure 3.31 shows MultimediaBSTree movie interface in which all objects on the stage are visible before starting this movie.

Definitions and conceptions of Binary Search Tree

The property that makes a binary tree into a binary search tree is that for every node, X, in the tree, the values of all the keys in its left subtree are smaller than the key value in X, and the values of all the keys in its right subtree are larger than the key value in X.

All the tree operations (insertion, deletion, and searching) can be performed in this program.

Insertion Algorithm

The insertion is conceptually simple. To insert X into tree T, proceed down the tree as you would with a Find. If X is found, do nothing. Otherwise, insert X at the last spot on the path traversed. The Insertion into a binary search tree C code is as following:

```
Insert(ElementType  X, SearchTree T)
{
        if(T= =NULL)
        {
          /*Create and return a one-node tree */
          T = malloc(sizeof(struct TreeNode));
          if(T= =NULL)
                FatalError("Out of space!!!");
          else
          {
            T- >Element=X;
            T- >Left=T->Right=NULL;
          }
        }
        else
        if(X< T->Element)
          T->Left=Insert(X, t->Left);
        else
        if(X>T->Element)
          T->Right=Insert(X, T->Right);
        /* Else X is in the tree already; we'll do nothing */

        return T;
}
```

Deletion Algorithm

case 1: If the deleted node is a leaf, it can be deleted immediately

case 2: If the deleted node only has one child, the node can be deleted after its parent adjusts a pointer to bypass the node.

case 3: If the deleted node both has the left child and right child, exchange this node with the smallest node of the right subtree. Then delete the node.

Table 3.5 The Contents of Binary Search Tree Definitions and Conceptions

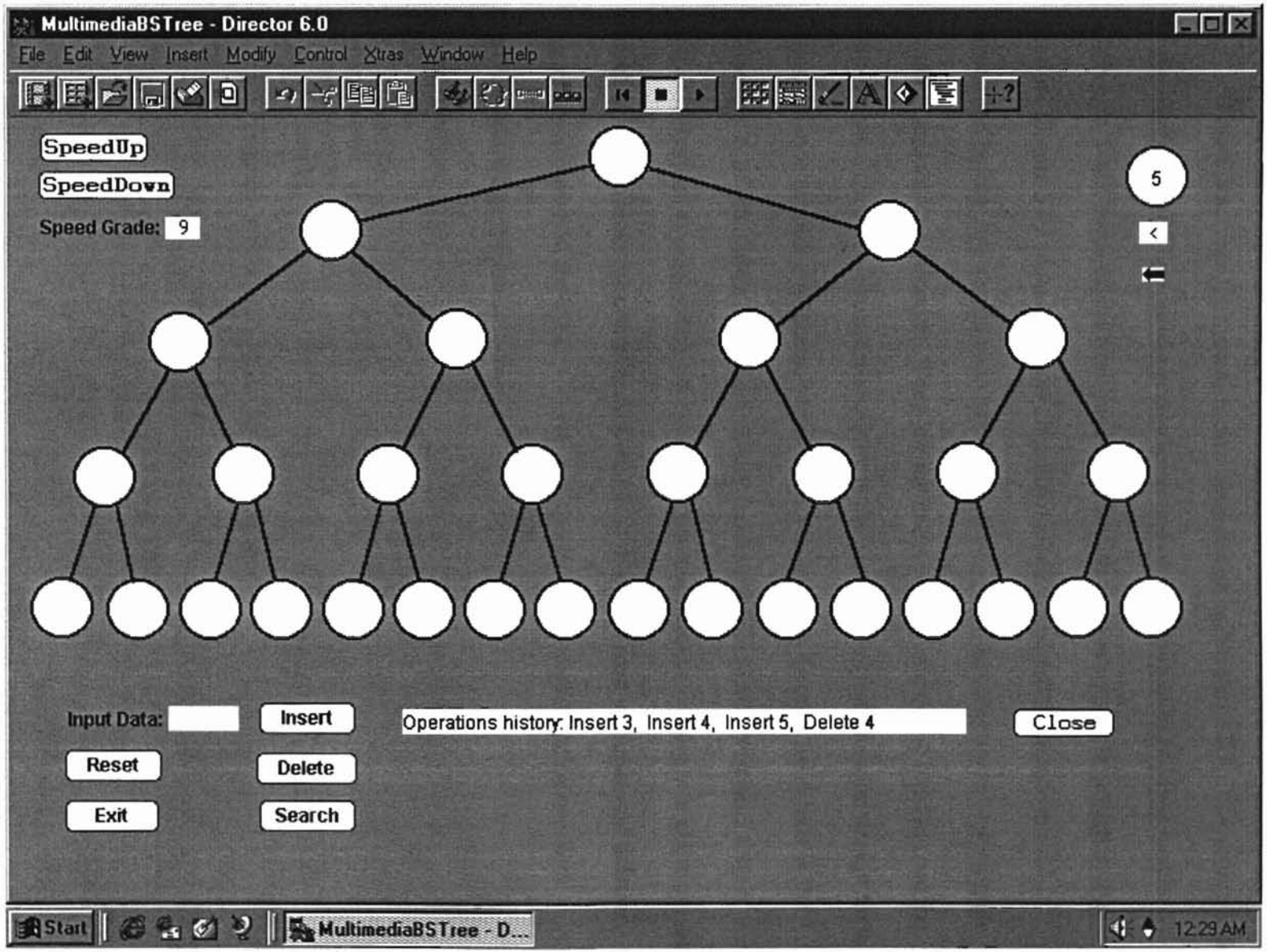

Figure 3.31 MultimediaBSTree Movie Interface

3.6 Source Code Design

I have described the system source code design in Section 3.2.3. The source code of the MultimediaBSTree movie is similar to that of the MultimediaAvlTree movie. Therefore, I will focus on the MultimediaAvlTree movie's source code design in this section. Figure 3.32 shows the flow chart for the MultimediaAvlTree movie. As mentioned before, this source code has 46 handlers. I will describe the following main handlers and their pseudocode. They are the insert handler, the delete handler, and the adjustbalance handler. The search handler is the same as the insert handler except for no adjustBalance handler call. Other handlers source code can be seen for details in the Appendix A of this thesis. There are two global linked lists in this program. One is gNodeList which is used to link every node in the tree in order. Another is gHistoryList which is used to store operation history in order. The AVL tree insertion and deletion algorithm is shown in the Table 3.4. Its insert, delete, and adjustBalance handlers pseudocode are as follows:

Insert handler:

```
On Insert
   global gHistoryList, gNodeList
   --check if the tree is full
   --check if input data is blank or space

   if input data is not in the tree

      set input data=Token
    set root=currentValue
    while loop (current node has value)
       if currentValue>Token
          insert node move to left child
          set current node= left child
       else
          insert node move to right child
          set current node=right child
       end if
```

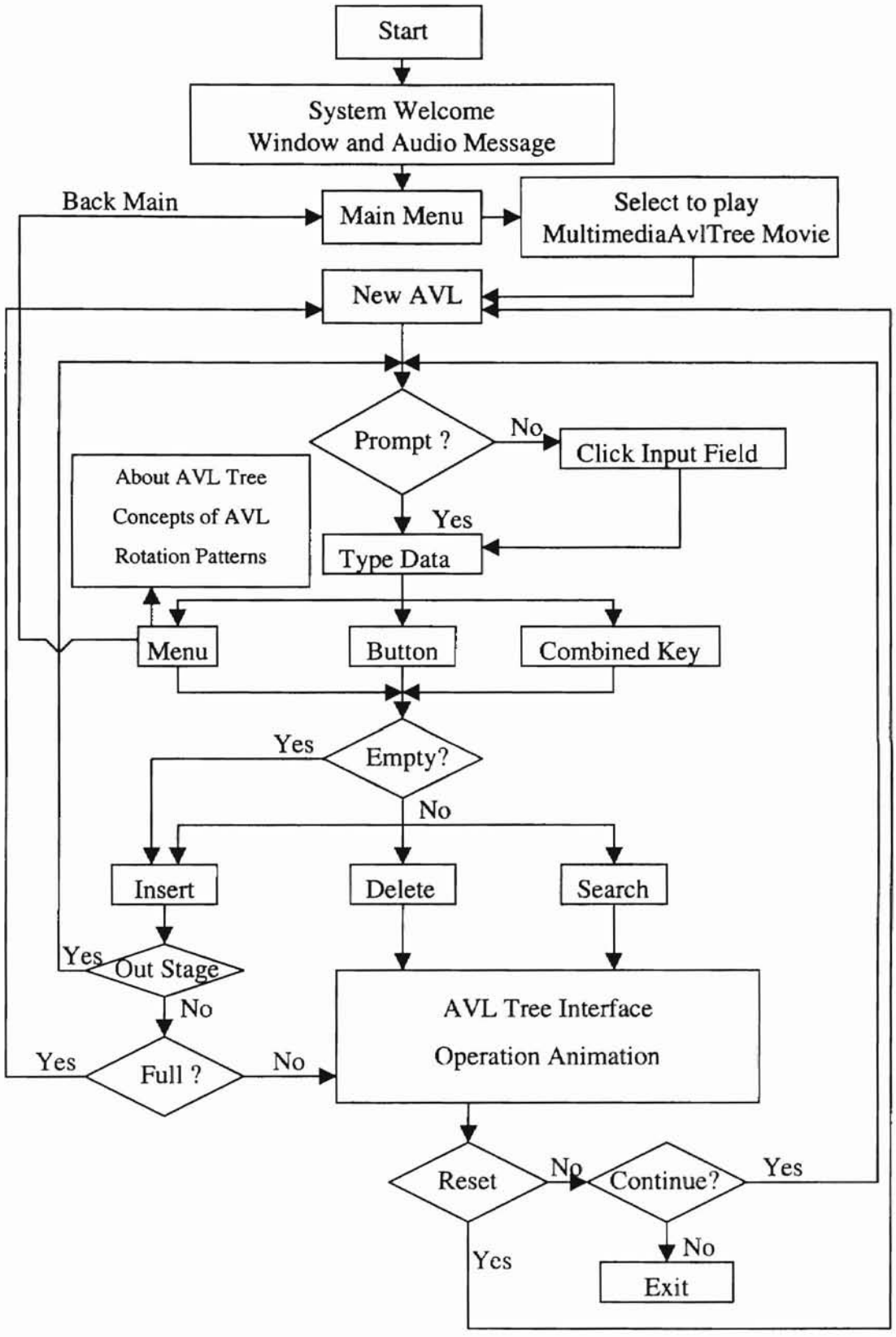

Figure 3.32 Flow Chart for MultimediaAvlTree Movie

```
      end while loop

      adjustblance(current node)

      add Token into gNodeList
      add Insert and string(Token) into gHistoryList

   else
      alert This node already in the tree

   end if
end insert


adjustbalance handler:

   on adjustBalance(currentNode)
      set parent=(currentNode)/2
      set GP=parent/2

      if no grandparent
         exit
      end if

      while loop (grandparent exist)
                  if (GP's height difference between left
            and right =2)
                  --need rotation
            if parent is left child of GP and
               currentNode is left child of parent
               --case 1
               singleRotateWithLeft(GP)
            else if parent is right child of GP and
               currentNode is left child of parent
               --case 2
               doubleRotateWithRight(GP)
            else if parent is left child of GP and
               currentNode is right child of parent
               --case 3
               doubleRotateWithLeft(GP)
            else if parent is right child of GP and
               currentNode is right child of parent
               --case 4
               singleRotateWithRight(GP)
            end if

         else
            --up one level
            set currentNode=parent
            set parent=currentNode/2
            set GP=parent/2
         end if
```

```
      end while loop
   end adjustBalance


delete handler:

   on delete
      global gHistoryList, gNodeList
      --check if input data is a blank or space
      --check if deleted node is not  in the tree
      while loop
       find deleted node position and treeHeight
    end while loop

    --delete this node in many cases
    if deleted node is a leaf
     deletenode
    else if deleted node only has a left subtree
     deleteNode
     move left subtree up one level
    else if deleted node only has a right subtree
     deleteNode
     move right subtree up one level
    else if deleted node have both left and
            right subtree
     find minimum node in the right subtree
     exchange the positions between deleted
       node and minimum node
     deleteNode
     if current deleted node only has a
          left subtree
        move right subtree up one level
     else
         move left subtree up one level
     end if
    end if

deleteBalance

    add deleted node into gNodeList
    add Insert and string(deleted node) into
        gHistoryList
end delete
```

In addition, Undo and Replay functions are very important for this system. If users make a mistake, do a wrong operation, or don't see clearly last step action of the movie playing, they can select the Undo option from the menu operations item to go back one step, then play again, or select the Replay from menu to replay the movie. Therefore,

we add the Undo and Replay functions in the system. The Replay function is similar to the Undo function. The Undo will set on flag with which all operations have no animations and delete the last element from gKeyList and gOpNameList. The Replay will do all animation operations (see Appendix A for details). The following code is the Undo function pseudocode.

```
on undo
  declare global variables

  set gUndoFlag=1
  reset

  set the highest gSpeedGrade

  delete the last element of gKeyList and gOpNameList

  --recover AVL tree except last element
  x=numbers of the elements in gKeyList
  loop from i=1 to i=x
    Token= ith elements in gKeyList

    if Operation = "Insert" then
      insert operation
    else if Operation = "Delete" then
      delete operation
    else
      search operation
    end if
  end loop

  set the normal gSpeedGrade
  set gUndoFlag=0
end undo
```

CHAPTER IV

TESTING AND RUNNING

We dedicate the first part of this chapter to testing the MultimediaAvlTree movie's error message. Then we are running the MultimediaAvlTree movie with the example in book [31] (pp114-pp119). Following is error message testing:

1. Figure 4.1 shows the full AVL tree. If the user wants to insert another node 32 into this full AVL tree, system will beep, pop up the error message "Sorry, this tree is full !" as shown in Figure 4.2, and audio message "Oops, this tree is full".
2. If the user wants to insert a node 9 in to the AVL tree as shown in Figure 4.3, this node will go to level 5. It is over system level limited: level 4. So, the system will beep, pop up the error message "Sorry, Out of Stage, Try again", and audio message "Oops, out of stage, please try again".
3. If the user doesn't type data in the input data field, then do any one of operations, there will be an error message that will pop up as shown in Figure 4.4.
4. If the user wants to delete or search a node which is not in the tree, system will beep, pop up the error message "This node is not in the tree, please try again !" as shown in Figure 4.5, and audio message " This node is not in the tree, please try again".

5. If the user wants to insert a node which is already in the tree, system will beep, pop up the error message "This node is already in the tree" as shown in Figure 4.6, and audio message "This node is already in the tree".

Running insertion operation with an example:

Insert the nodes 3,2,1, (as shown in Figure 4.7) and then 4 though 7 in sequential order (the result is shown in Figure 4.8). Then we continue our previous example by inserting the nodes 10 though 16 in reverse order (as shown in Figure 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15 respectively), followed by inserting node 8 (as shown in Figure 4.16) and then inserting node 9 (as shown in Figure 4.17). Since node 8 is in level 4, node 9 should be node 8 right child in level 5. So insertion of node 9 caused "Sorry, Out of stage! Try again!" message to pop up.

Besides running the insertion operation, we were also testing and running deletion and search operations continuously during the development of this program.

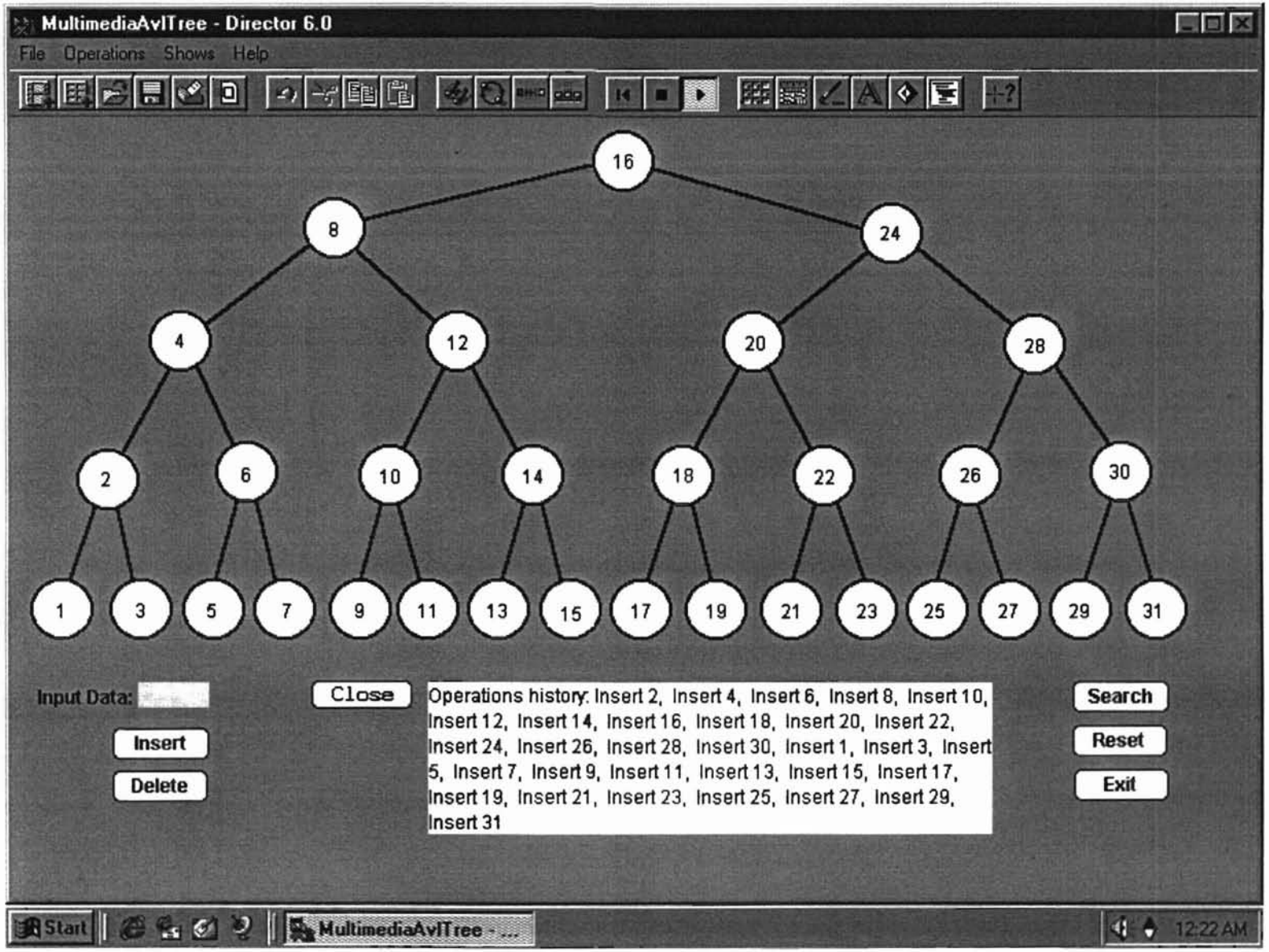

Figure 4.1 Full AVL Tree

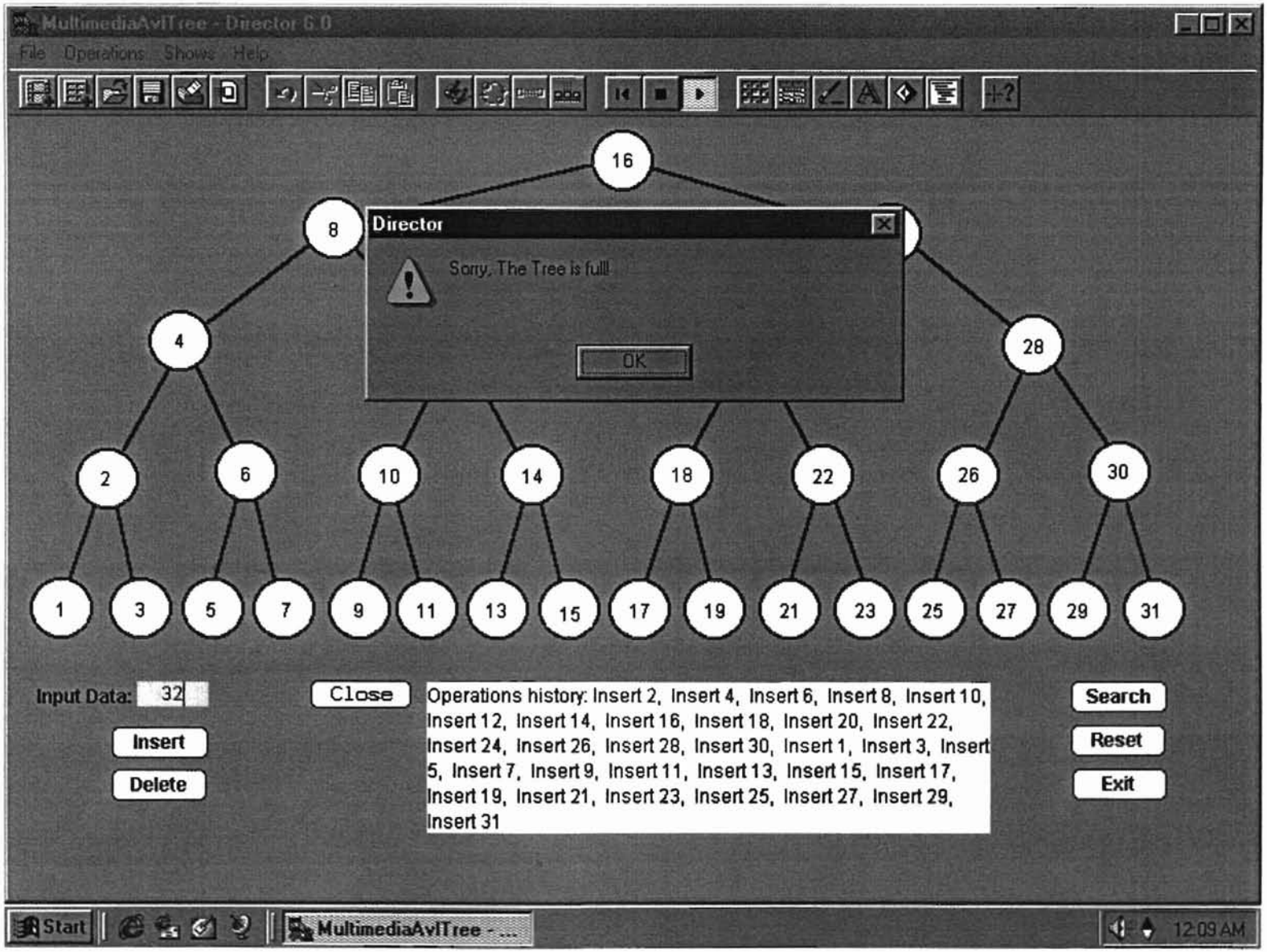

Figure 4.2 Tree full Message

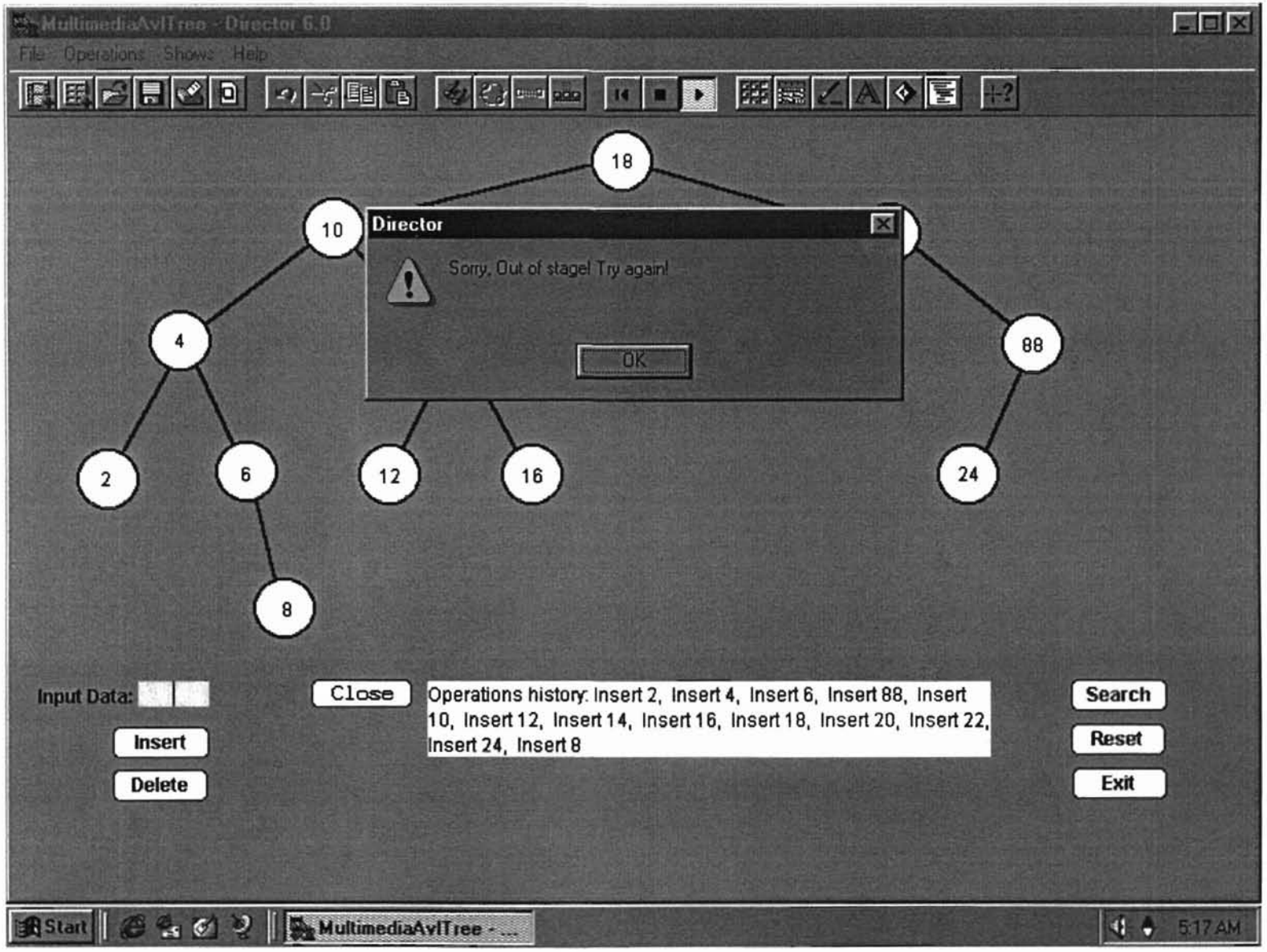

Figure 4.3 Out of Stage Message

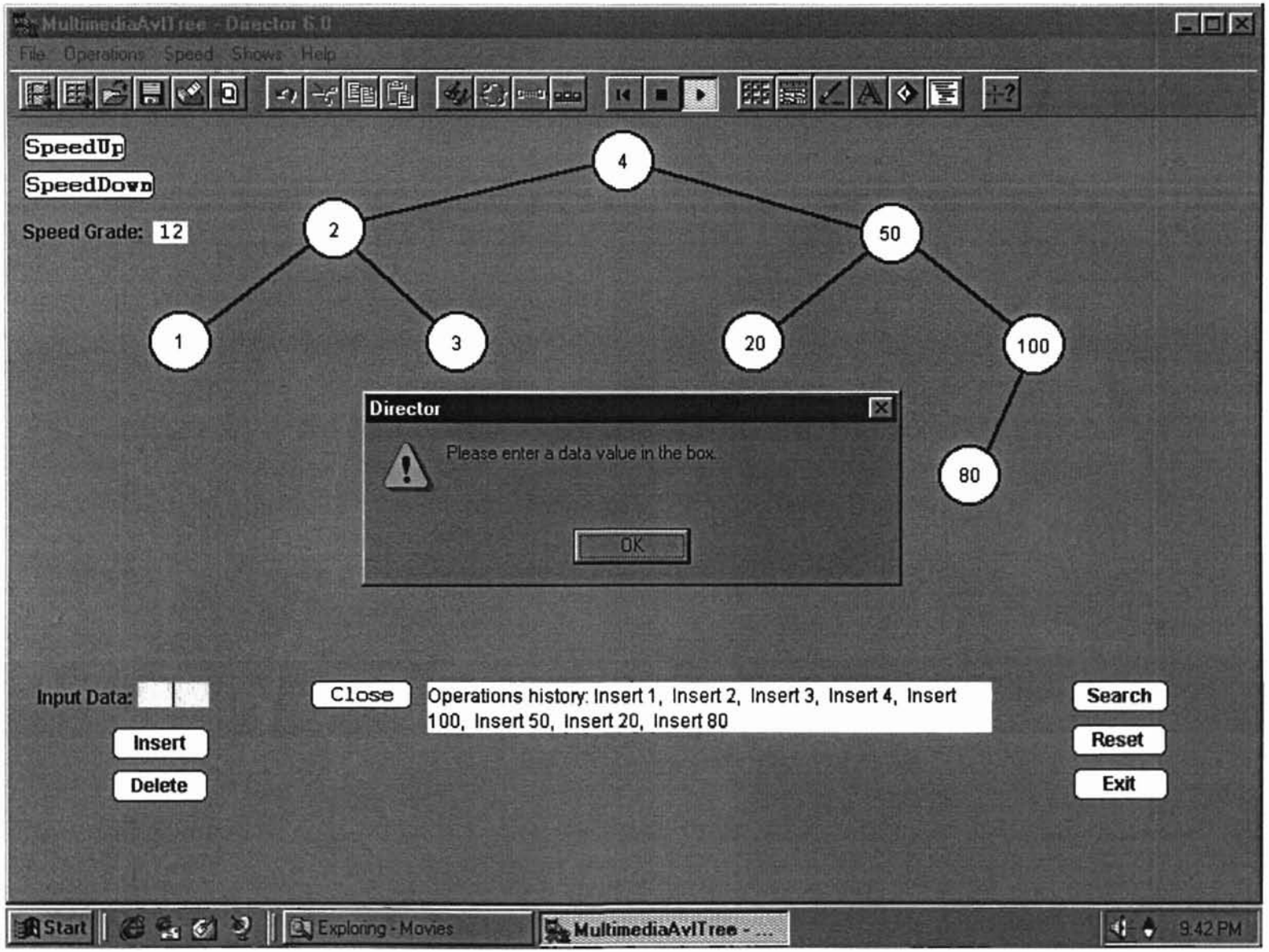

Figure 4.4 No Input Data Message

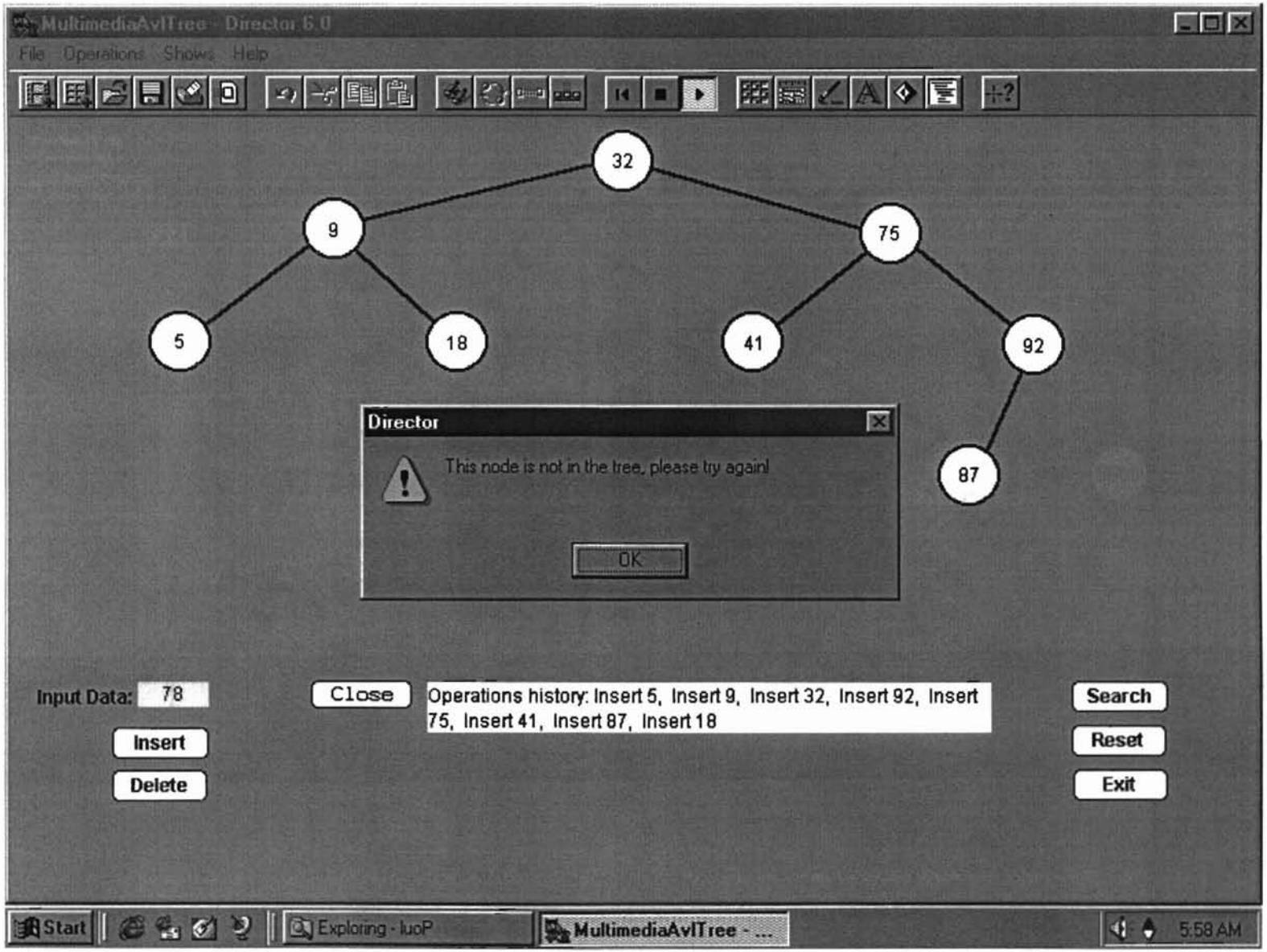

Figure 4.5 No Such Node in the Tree Message

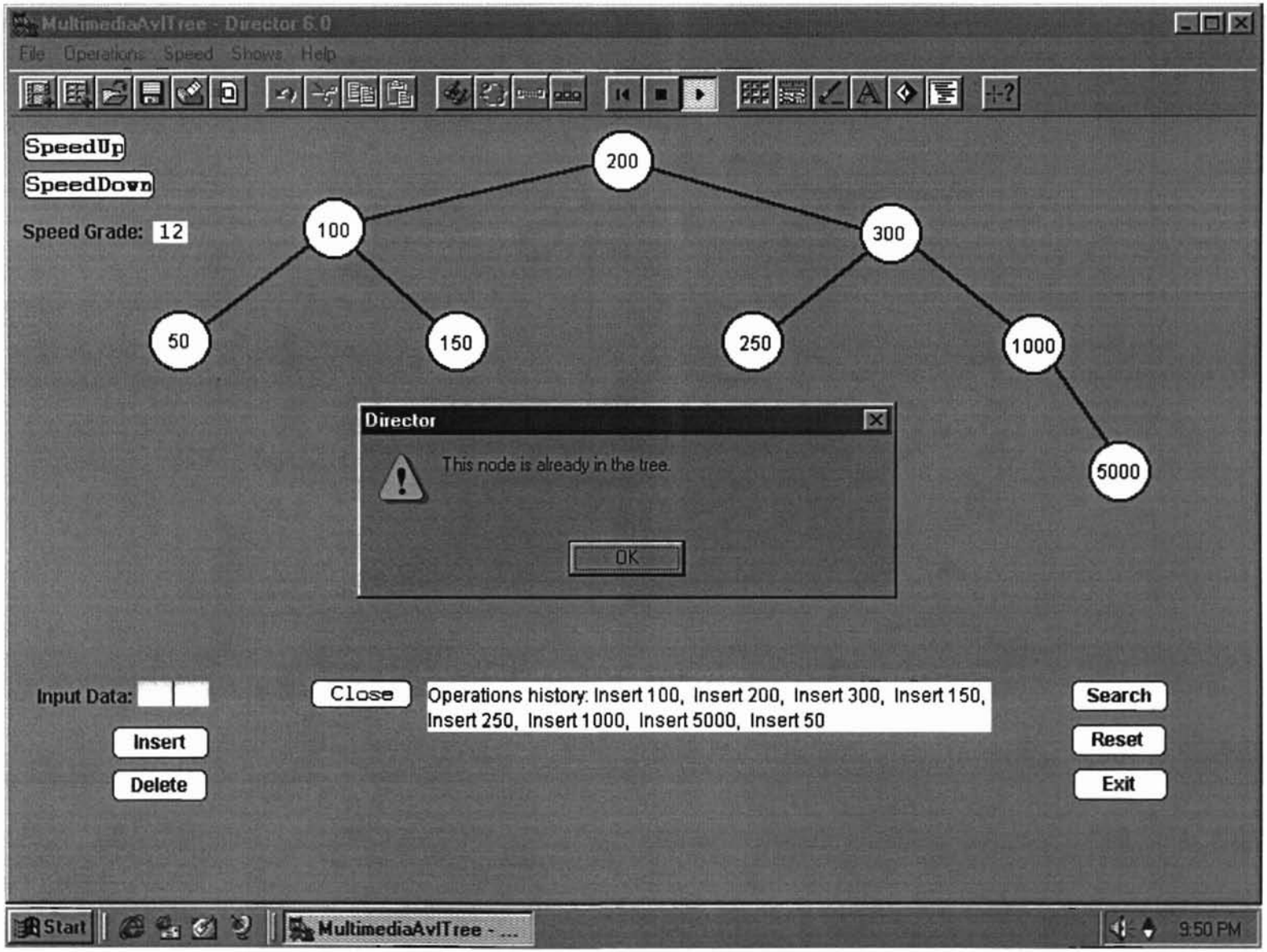

Figure 4.6 This Node Is already in the Tree Message

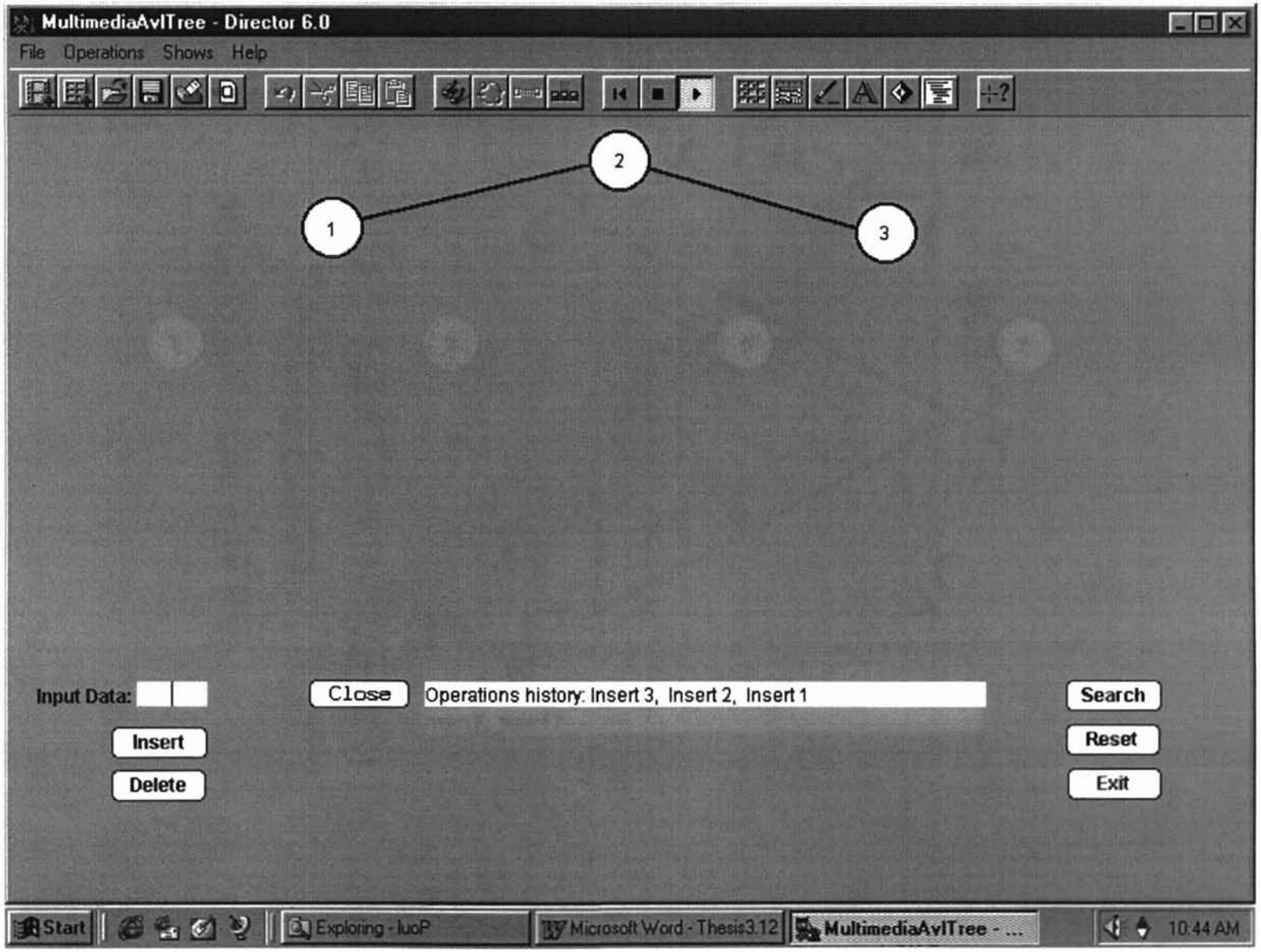

Figure 4.7 Example after Insertion of Node 3, 2, 1

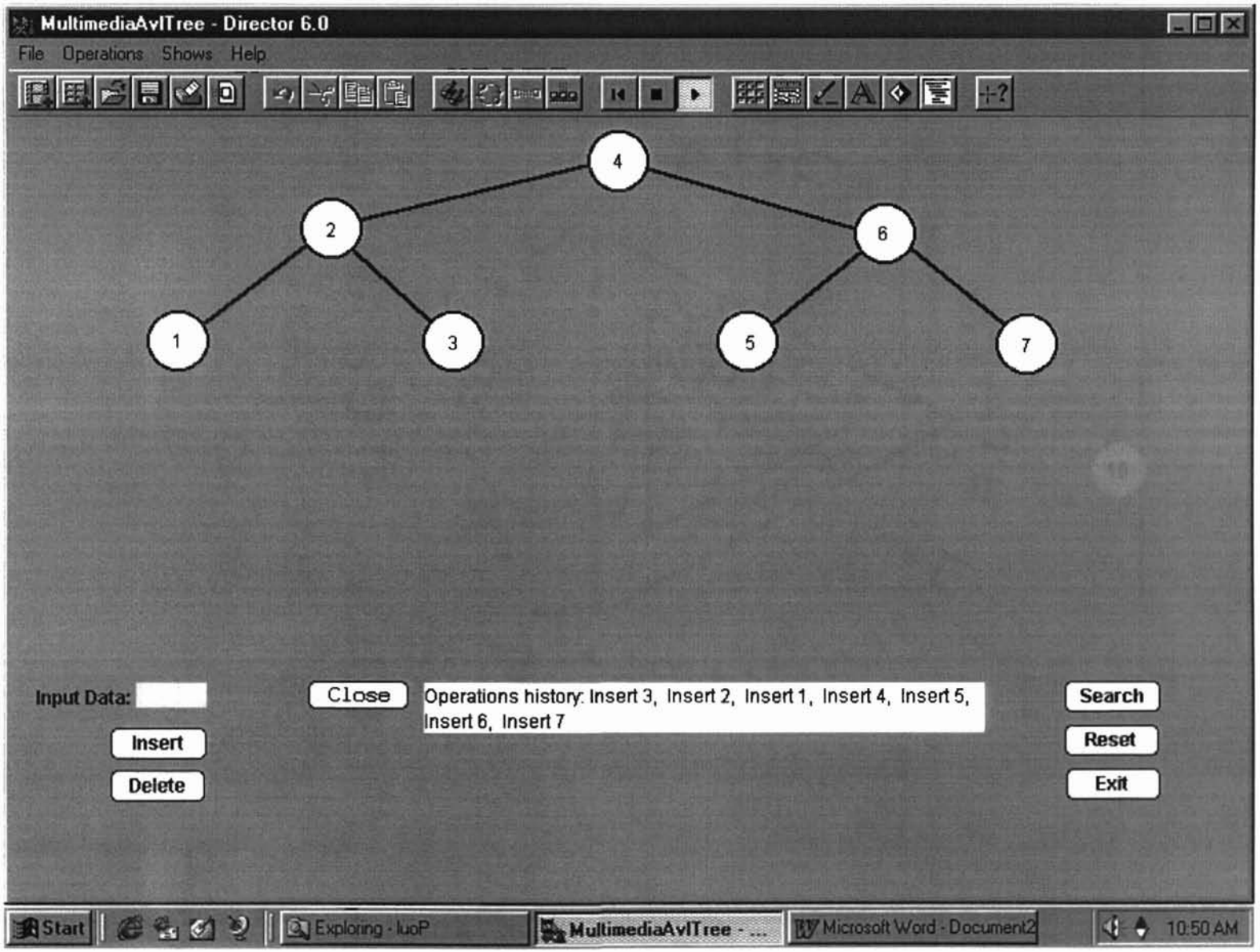

Figure 4.8 Example after Insertion of Node 4, 5, 6, 7

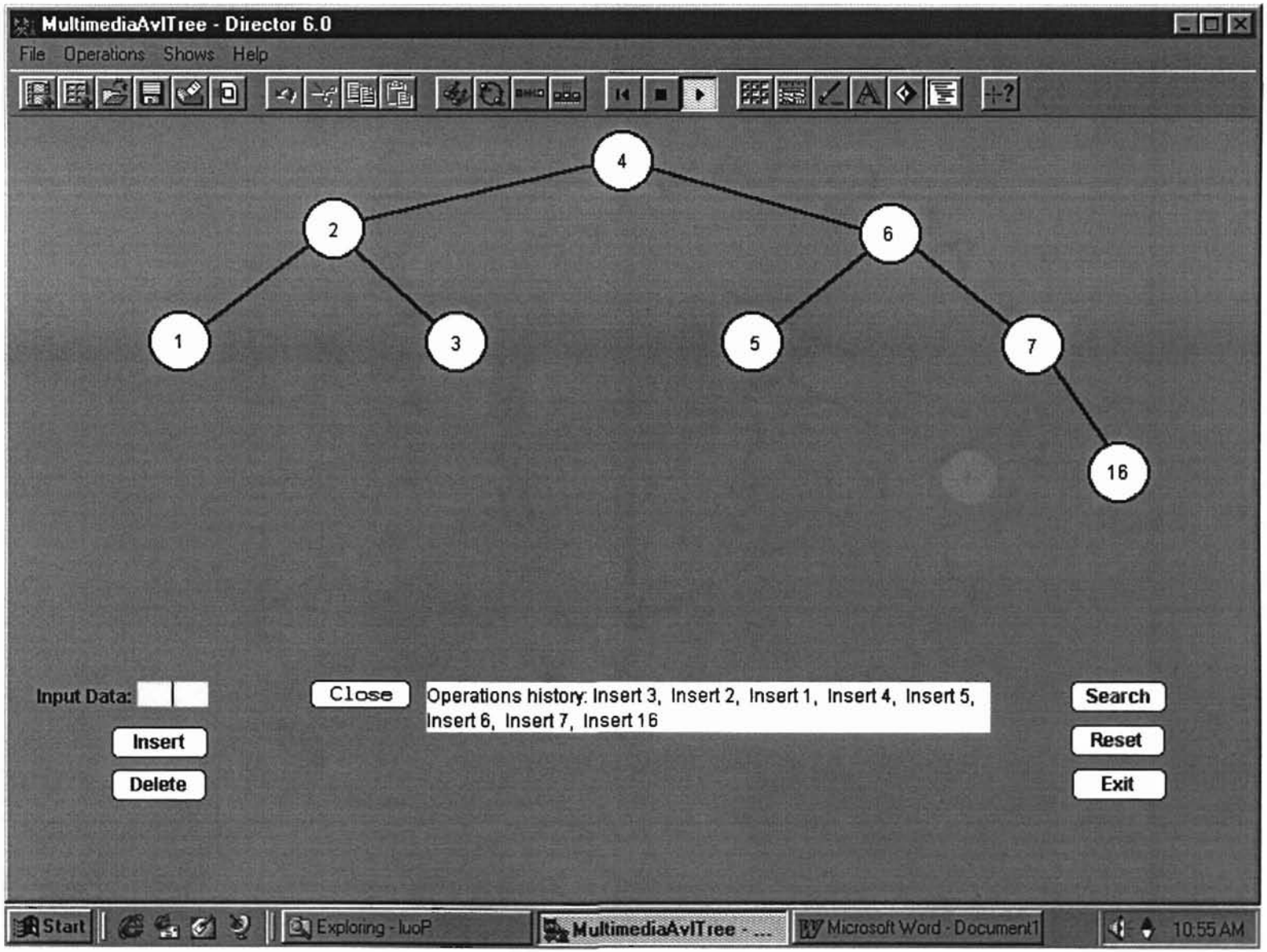

Figure 4.9 Example after Insertion of Node 16

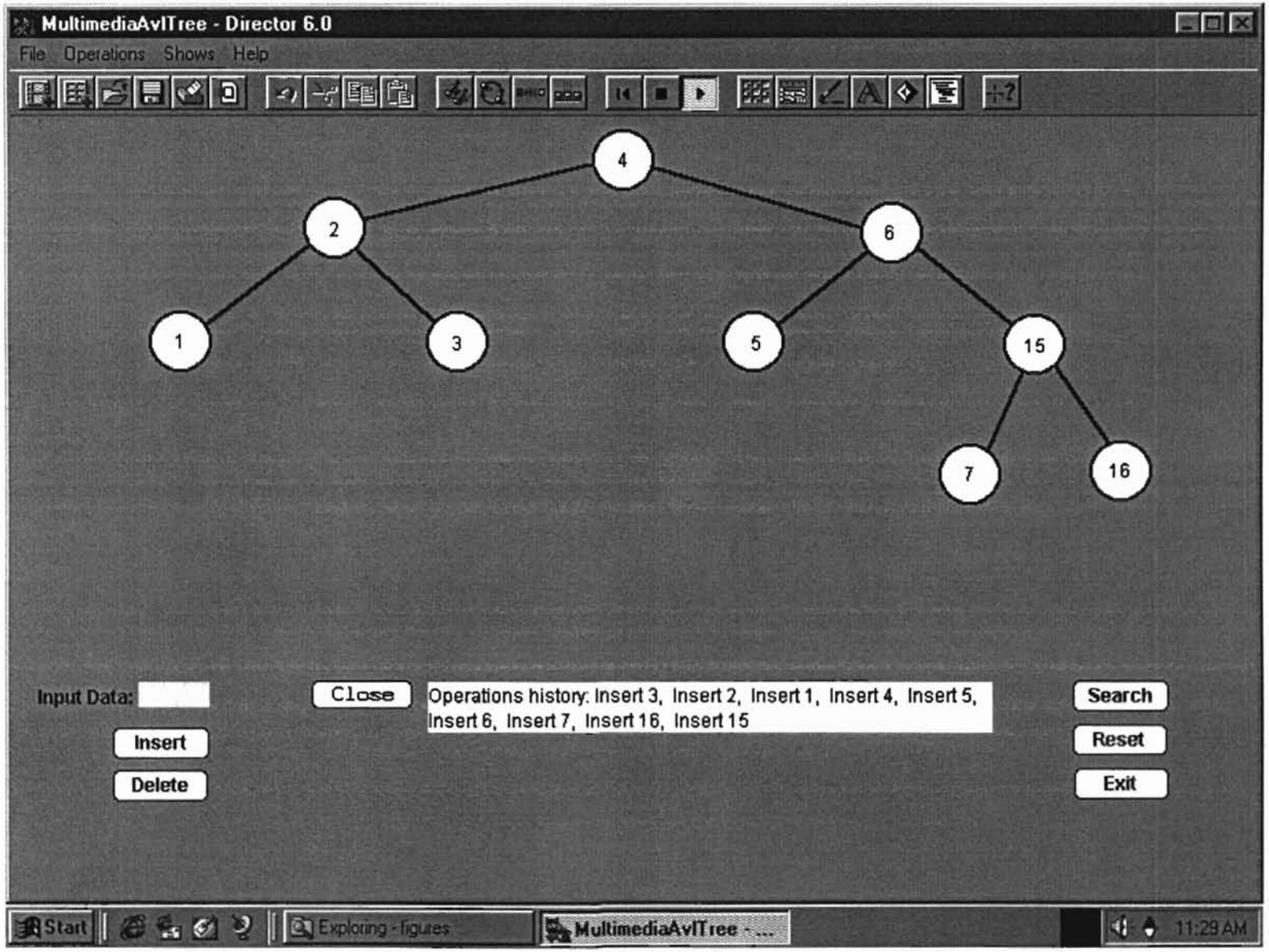

Figure 4.10 Example after Insertion of Node 15

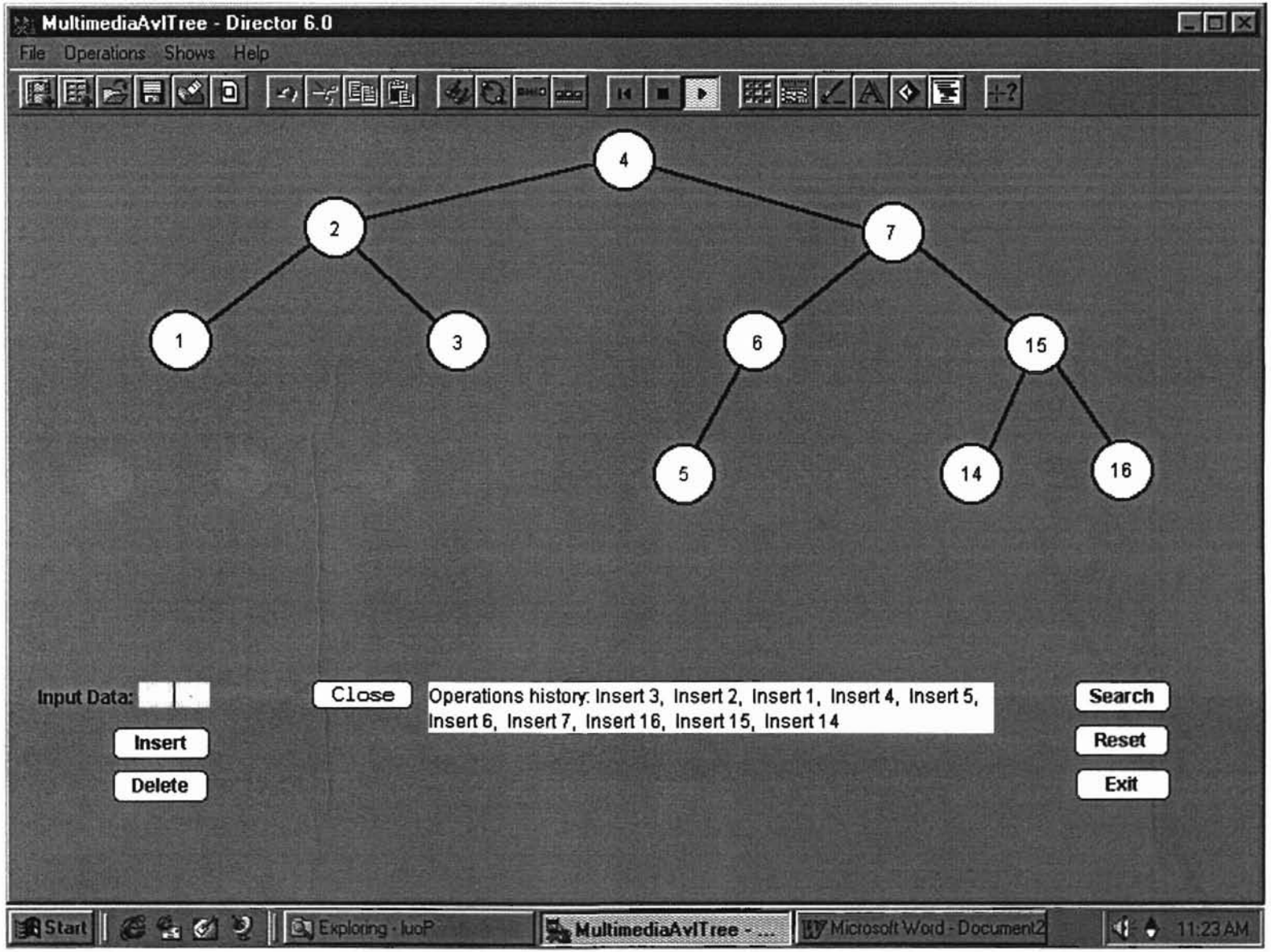

Figure 4.11 Example after Insertion of Node 14

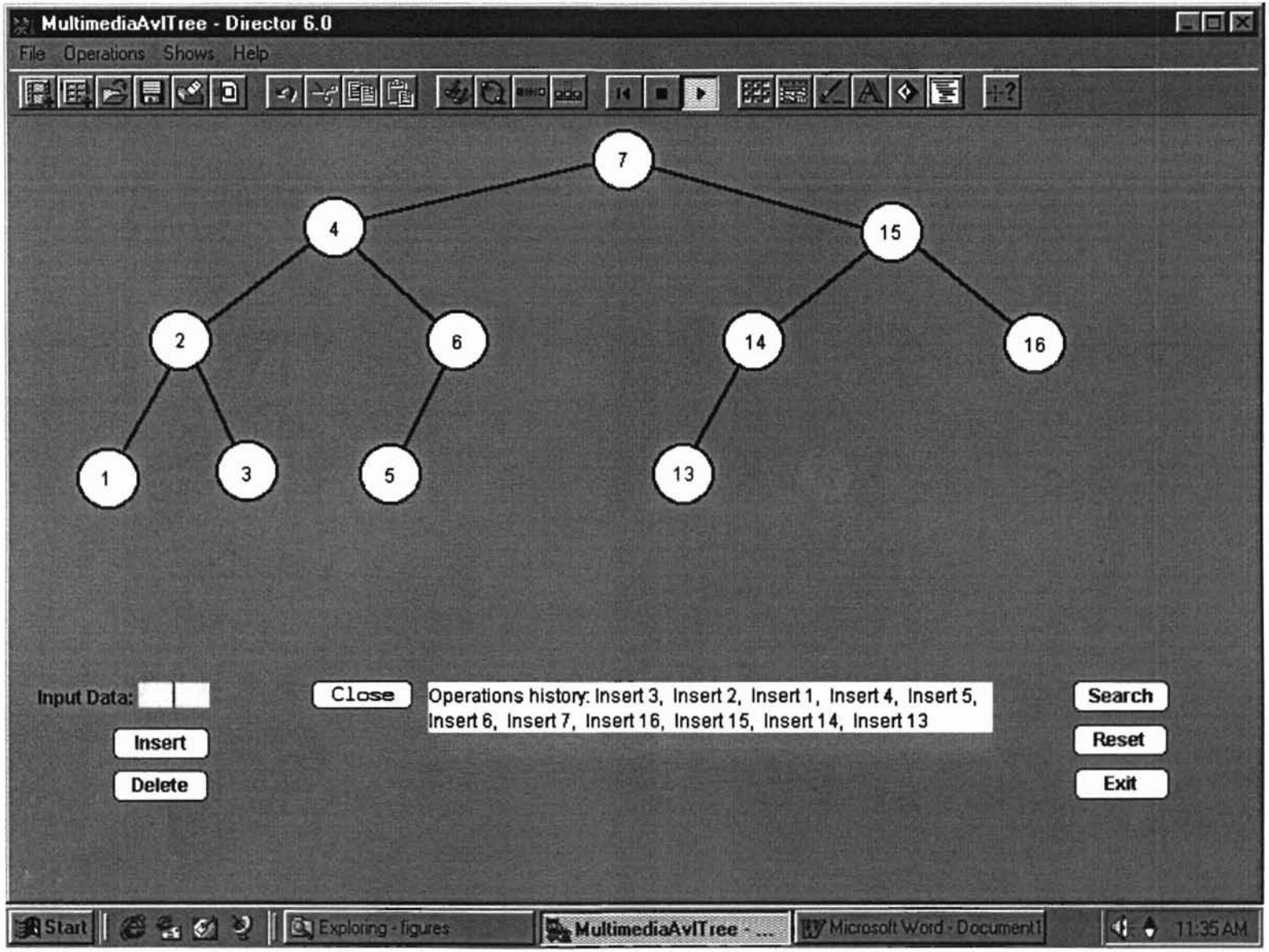

Figure 4.12 Example after Insertion of Node 13

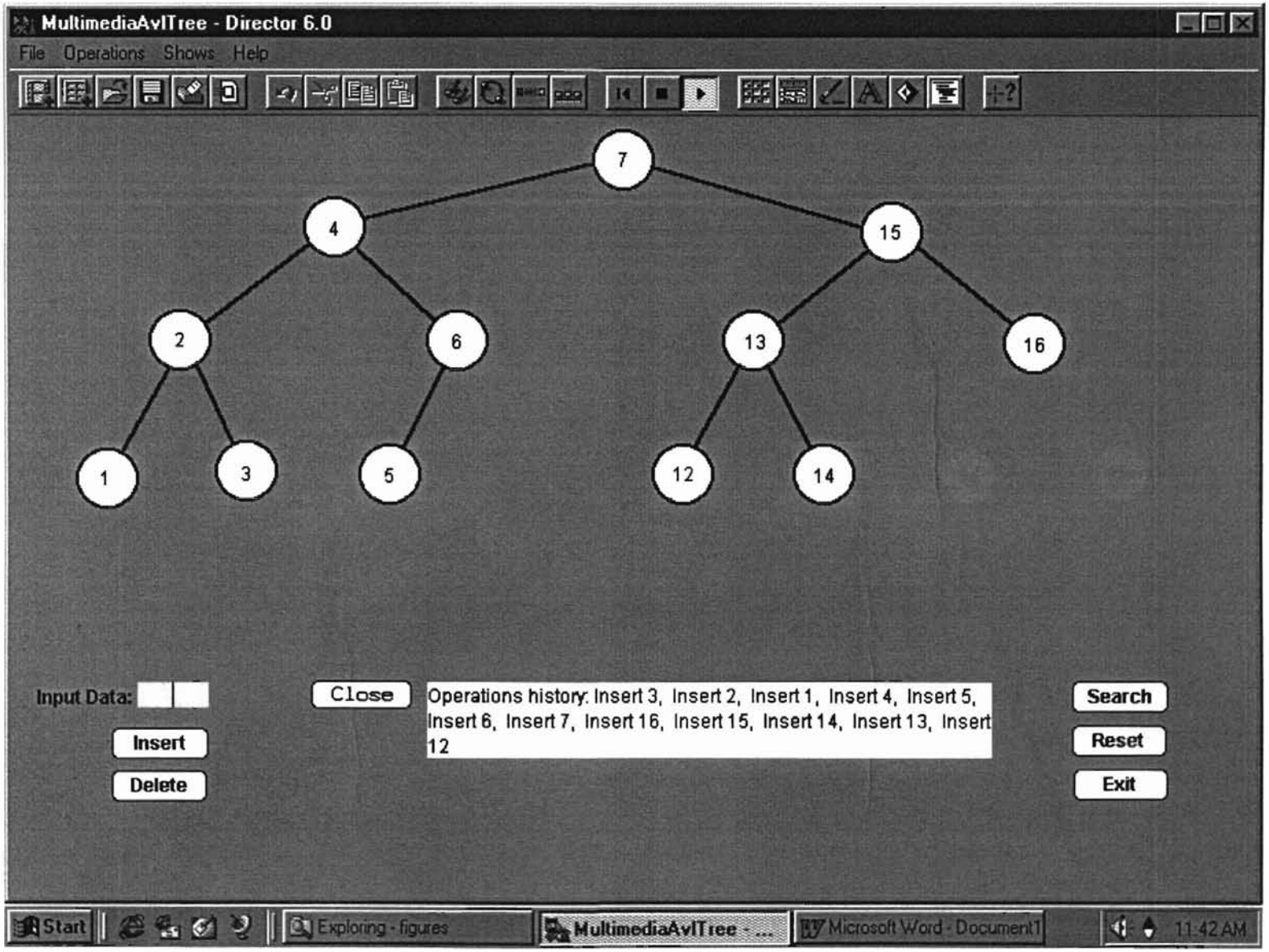

Figure 4.13 Example after Insertion of Node 12

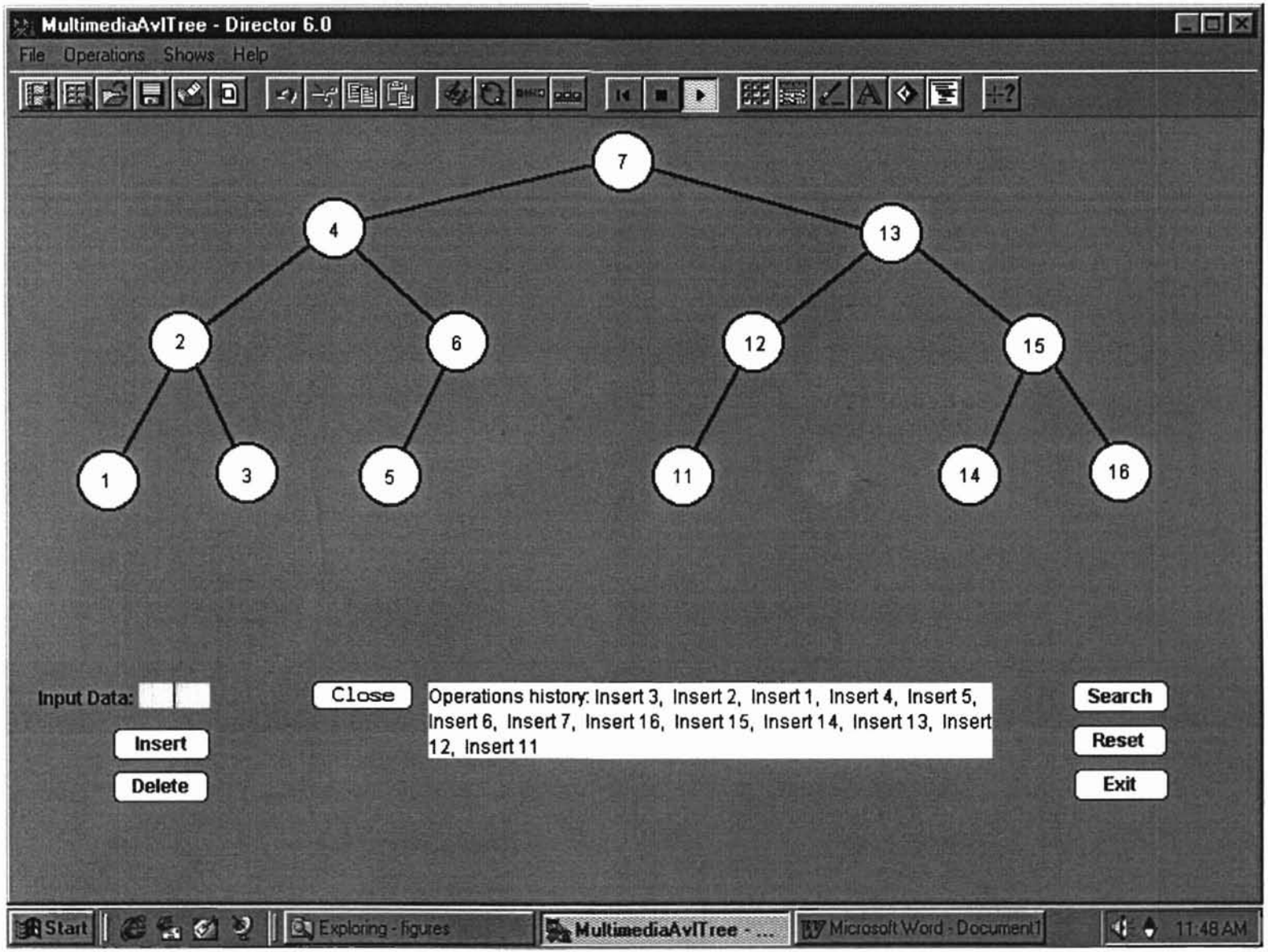

Figure 4.14 Example after Insertion of Node 11

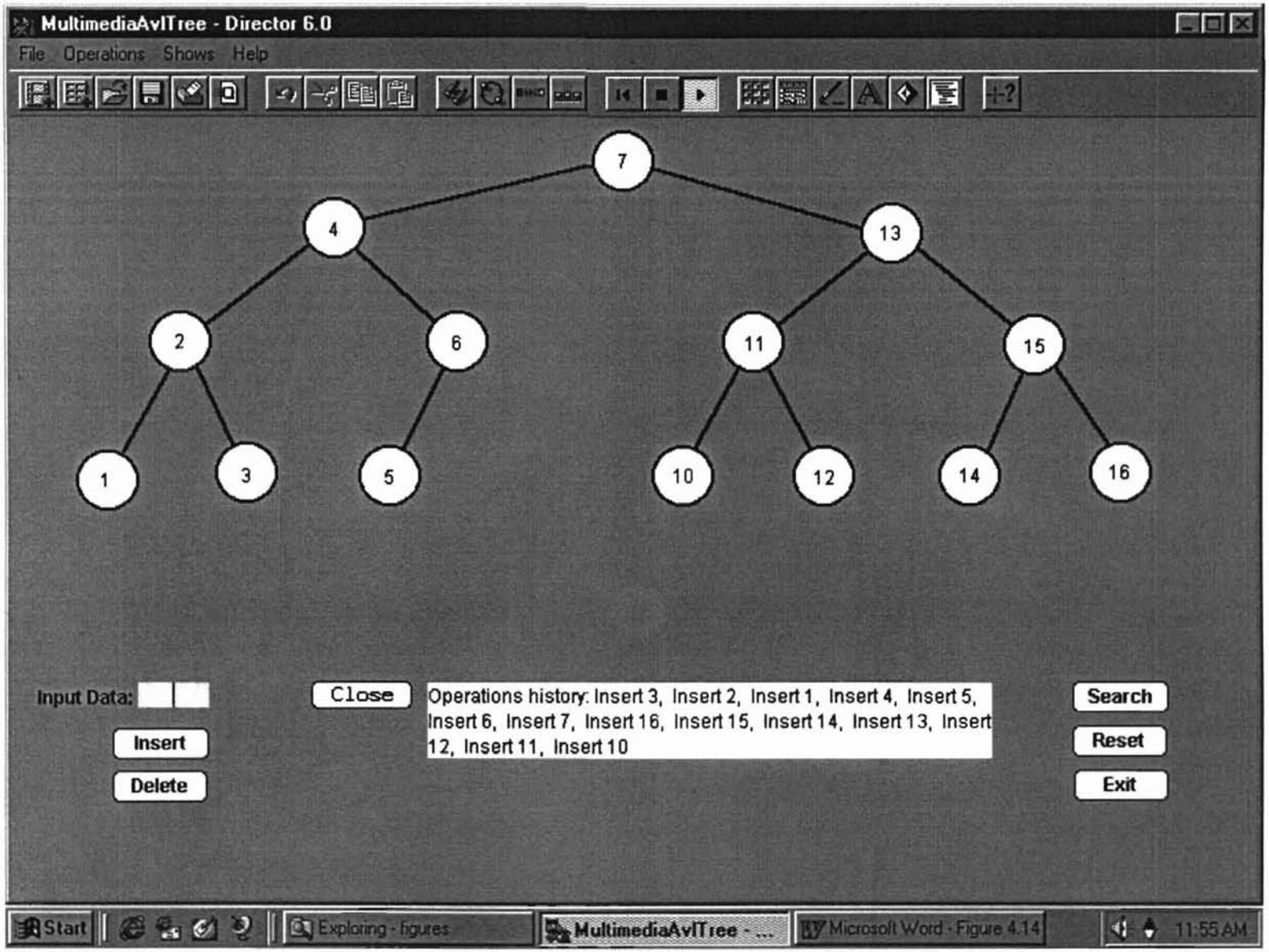

Figure 4.15 Example after Insertion of Node 10

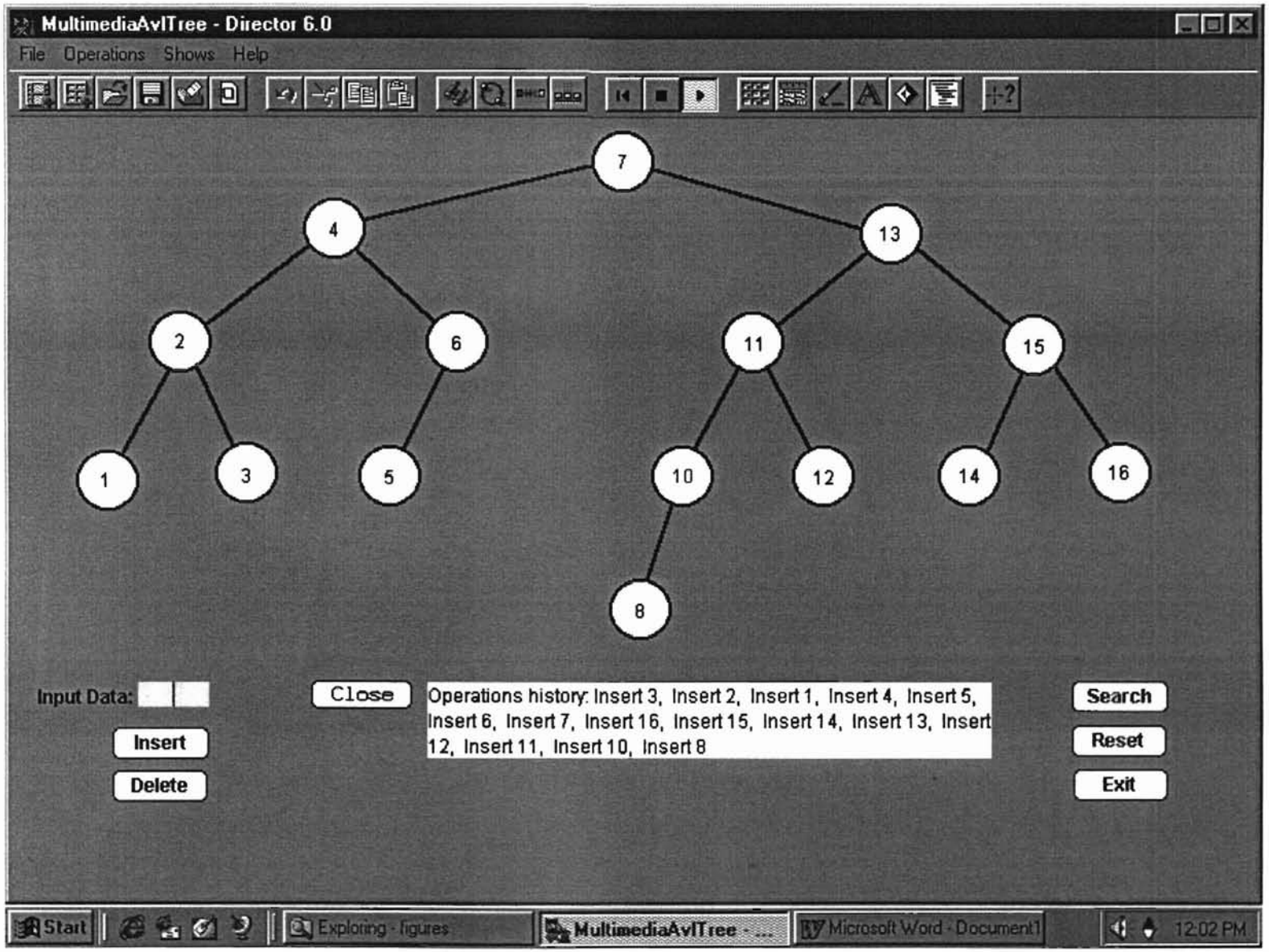

Figure 4.16 Example after Insertion of Node 8

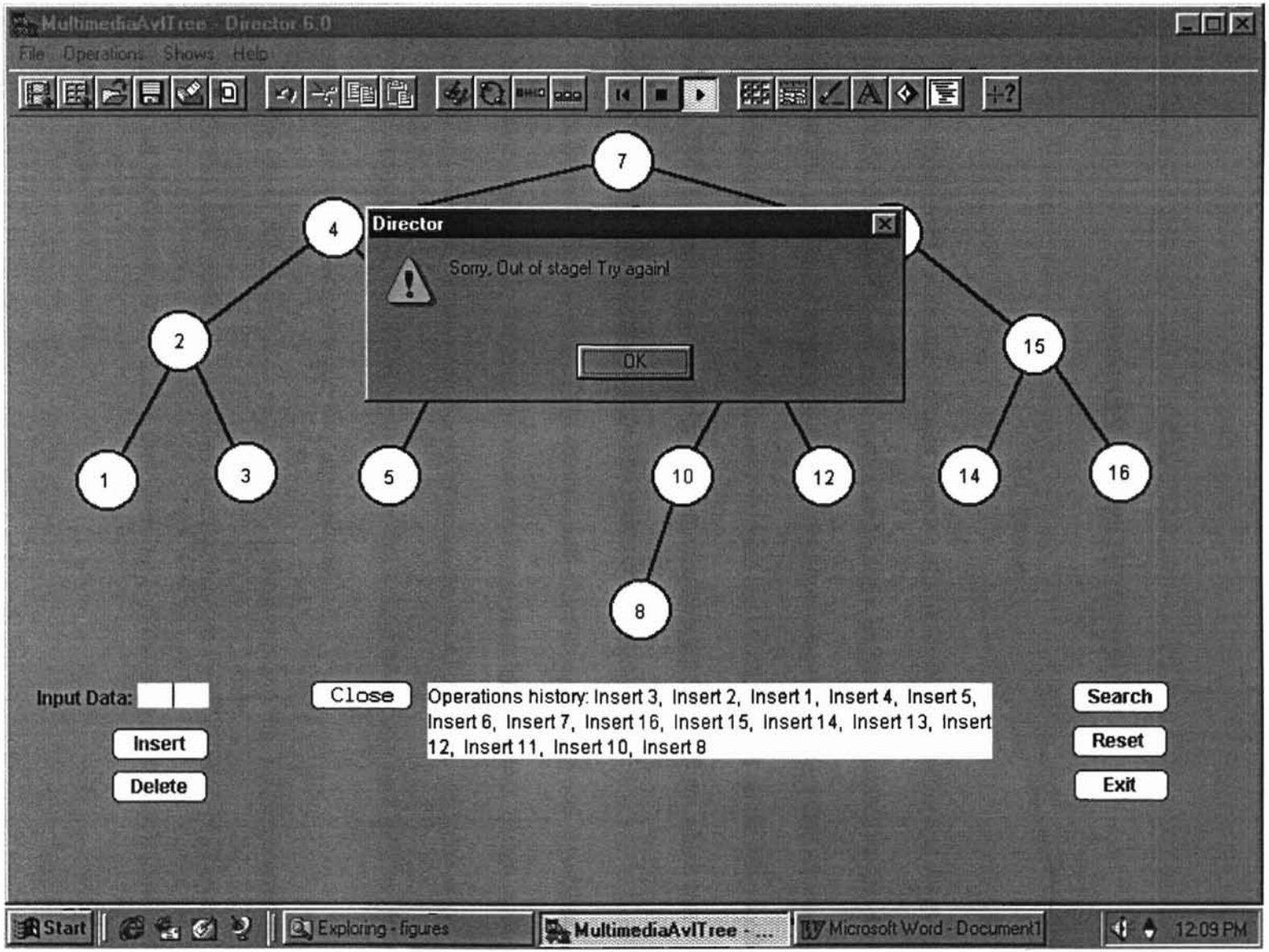

Figure 4.17 Example after Insertion of Node 9

CHAPTER V

SUMMARY AND FUTURE WORK

5.1 Summary

As mentioned earlier, the primary purpose of this thesis is to design and develop, with scientific visualization and multimedia technology, an flexible, interactive, and user-friendly MDSL system for simulating the animated operations of abstract data structures as a teaching and learning tool. We have developed the MDSL system on the Microsoft Windows 98/NT operating system with Macromedia Director 6.0 platform which has good software accessibility, reusability, and maintainability.

For better understanding of the abstract data structures concepts and algorithms, we have explored not only the human vision sensory system by using text, image, and animation but also the human audio sense by using the sound message. In order to get the best learning result, the MDSL system provides a good interactive user interface to let users take part in simulations while they are learning.

The use of color animation in the MDSL system enriches the implementation of the visualization and communicates information to users more efficiently. In general, users can get more information during their learning because of the good environment provided by the MDSL system.

5.2 Future Work

This implementation performs the good features and fulfills requirements for the MDSL system. However, there are some other features and schemes that could be developed for the future works which are listed below:

1. Currently, only the MultimediaAvlTree, MultimediaBSTree, VisualRedBlackTree, MultimediaB-Tree, and MultimediaADT are available. The others (shown in Figure 3.2) are not available yet. Therefore, we have a lot of work to do. In the appendix, there is a programmer's guide. If anyone develops a data structure movie, he/she can follow this guide to add the movie into the system.
2. Use the video technology in this multimedia system.
3. Add some examinations and quizzes into every data structure movie to test the users' learning results. The examinations and quizzes should be divided into four or five levels for the different level learners.
4. Publish the MDSL system to OSU Computer Science Department web site, and use the Shockwave plug-in as Internet multimedia movie for distance learning.

BIBLIOGRAPHY

[1] Adelson-Velskii, G. M. and E. M. Landis. An algorithm for the organization of information. *Soviet. Mat. Doklady,* 3(3): pp. 1259-1263, 1962.

[2] Aho, A. V., R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Reading. Massachusetts. Addison-Wesley, 1985.

[3] Appleton, B. URL: http://www.enteract.com/~bradapp/ftp/src/libs/C++/AvlTrees.htr.

[4] Baecker, Ronald. *Sorting Out Sorting.* 16mm color film with sound (25 minutes), The Dynamic Graphics Project Computer Systems Research Group, University of Toronto, Toronto, Canada, 1981.

[5] Baecker, R. M. and W. A. S. Buxton. *Reading in Human-Computer Interaction.* Los Altos, CA: Morgan Kaufmann,1987.

[6] Baron, R. J. and L.G. Shapiro. *Data Structures and Their Implementations.* New York. Van Nostrand Reinhold, 1983.

[7] Bly, B. A. Presenting information in sound. *Proceeding of CHI'85 Conference on Human Factors in Computing Systems.* New York: ACM Press, pp. 371-375, 1985.

[8] Booth, K. *PQ-Trees.* 16mm color silent films (12 minutes), 1975.

[9] Buford, J.F.K. *Multimedia Systems.* University of Massachusetts, Lowell, New York, ACM Press,1994.

[10] Buxton, W., S. A. Bly, S. P. Frysinger, and D. Lunney. Communications with sound. *Proceedings of the ACM SIGCHI Human Factors in Computing Systems Conference.* New York: ACM Press, pp. 115-119, 1984.

[11] Cormen, T. H., C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press; New York, McGraw-Hill, 1990.

[12] Dahl, O.J., E. W. Dijkstra, and C. A. R. Hoare. *Structured Programming*. New York, Academic Press, 1972.

[13] Earnshaw, R. A. and J. A. Vince. *Multimedia System & Applications*. San Diego, CA, Academic Press Inc, pp. 133-140, 1995.

[14] Gershon, N. D. From perception to visualization. *Computer Graphics*. 27(5): pp. 414-417, 1992.

[15] Hibbard, W., C. Dyer and B. Paul. Display of scientific data structures for algorithm visualization. *Visualization*, Boston, IEEE, pp. 139-146,1992.

[16] Keyes, Jessica. *The McGraw-Hill Multimedia Handbook*. New York, R. R. Donnelley & Sons Company. 1994.

[17] Knowlton, K. C. *L6: Bell Telephone Laboratories Low-level Linked List Language, Two Black and White Sound Films*. 1966.

[18] Lee, Wilson. *An Implementation of a Data Structures Display System*. M.S. Thesis. Computer Science Department, Oklahoma State University, Stillwater, OK, 1988.

[19] Lopuck, Lisa. *Designing Multimedia: A Visual Guide To Multimedia and Online Graphic Design*. Berkeley, CA, Peachpit Press, 1996.

[20] *Macromedia Director 6 Lingo Dictionary*. Macromedia, Inc., 1997.

[21] *Macromedia Director 6 Using Director*. Macromedia, Inc., 1997.

[22] *Macromedia Director 6 Using Lingo*. Macromedia, Inc., 1997.

[23] McCormick, B. H., T. A. DeFanti, and M. D. Brown. Visualization in scientific computing. *Computer Graphics*. 21 (6): pp. 1-14, 1987.

[24] Myers, B. A. INCENSE: A system for displaying data structures. *Computer Graphics*. 17 (3): pp. 115-125, 1983.

[25] Petrik, Paula and Dubrovsky, Ben. *Creating and Designing Multimedia with Director Version 5.0*. New Jersey, Prentice-Hall, Inc., 1997

[26] Reingold, E. M. and W. J. Hansen. *Data Structures*. Boston. Little, Brown and Company, 1983.

[27] Robertson, R. K., R.A. Earnshaw, D. Thalman, M. Grave, J. Gallup, and E. M. De Jong. Research issues in the foundations of visualization. *Computer Graphics and Applications*. 14 (2): pp. 73-76, 1994.

[28] Shen, Hung-che. *A Visual Aid for the Learning of Tree-based Data Structure*. M.S. Thesis. Computer Science Department, Oklahoma State University, Stillwater, OK, 1994.

[29] Sheu, Bing., and Ismail, Mohammed. *Microsystems Technology for Multimedia Applications*. New York, The Institute of Electrical and Electronics Engineers, Inc., 1995.

[30] Sommerville, I., *Software Engineering (third edition)*, Reading, MA: Addison-Wesley Publishing Co., 1989.

[31] Weiss, M. A. *Data Structures and Algorithm Analysis in C*. Menlo Park. CA: Addison-Wesley Publising Co., 1996.

[32] Wirth, N. *Systematic Programming:* An introduction. Englewood Cliffs, NJ, Prentice-Hall Inc., 1973.

[33] Zernik, D., M. Snir, and D. Malki. Using visualization tools to understand concurrency. *IEEE Software*. 9 (3): pp. 87-92, 1992.

APPENDIXES

APPENDIX A

AVL TREE LINGO CODE

```
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                                                              +
--+                        Main  Script                          +
--+                                                              +
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on startMovie

  -- global variable declarations
  global gNodeList,gHistoryList,gSpeedGrade
  global gKeyList, gOpNameList, gUndoFlag, gReplayFlag

  set gKeyList = []
  set gOpNameList = []
  set gUndoFlag=0
  set gReplayflag=0

  newTree

  set the stageColor to 43
  set the backColor of sprite 96 to 43
  set gSpeedGrade to 9
  set the text of member "speedField" to string(gSpeedGrade)

  installMenu 3

end startMovie

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
on reset
  global gHistoryList, gNodeList
  global gKeyList, gOpNameList, gUndoFlag, gReplayFlag

  set gHistoryList = []
  set gNodeList = []
  history

  if gUndoFlag=0 and gReplayFlag=0 then
    set gKeyList=[]
    set gOpNameList=[]
  end if

  repeat with i=1 to 98
    puppetSprite i,True
    set the visible of sprite i to False
  end repeat
```

```
  set the text of member "InputField" to ""

  repeat with i=21 to 51
    set the text of member i to ""
  end repeat
end reset

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on newTree
  reset
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on backMain
  play done
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on stopmovie
  repeat with i=1 to 98
    set the visible of sprite i to False
  end repeat

  set the text of member "InputField" to ""

  repeat with i=21 to 51
    set the text of member i to ""
  end repeat
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on closeWin
  set the modal of window "WinAB" to False
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(10,50,330,180)

  tell window "WinAB" to go to frame "Exit"
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on instruction
  puppetsound "instruction"
  updateStage
  set the modal of window "WinAB" to False
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(10,50,380,345)

  tell window "WinAB" to go to frame "Instruction"
  open window "WinAB"
end
```

```
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on SLrotation
  set the modal of window "WinAB" to False
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(247,490,777,620)

  tell window "WinAB" to go to frame "SLrotation"
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on SRrotation
  set the modal of window "WinAB" to False
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(247,490,777,620)

  tell window "WinAB" to go to frame "SRrotation"
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on DLrotation
  set the modal of window "WinAB" to False
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(247,490,777,620)

  tell window "WinAB" to go to frame "DLrotation"
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on DRrotation
  set the modal of window "WinAB" to False
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(247,490,777,620)

  tell window "WinAB" to go to frame "DRrotation"
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on aboutAvlTree
  set the modal of window "WinAB" to True
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(10,50,300,280)

  tell window "WinAB" to go to frame "AboutAvlTree"
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
on definition
  set the modal of window "WinAB" to True
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(10,50,390,300)

  tell window "WinAB" to go to frame "Definition"
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


on history
  global gHistoryList

  set operationNum to count(gHistoryList)
  if operationNum=0 then
    set operations="Operations history: "
  else
    set operations="Operations history: "¬
    & getAt(gHistoryList, 1)
    repeat with i=2 to operationNum
      put ",  " & getAt(gHistoryList,i) after operations
    end repeat
  end if

  puppetSprite 97, True
  puppetSprite 98, True
  set the visible of sprite 97 to True
  set the visible of sprite 98 to True
  set the text of member "showOpHi" to operations
  updateStage
end history

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on closeButton
  puppetSprite 97, False
  puppetSprite 98, False
  set the visible of sprite 97 to False
  set the visible of sprite 98 to False
end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                          Menu Script                          +
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

menu: File
New/N|newTree
Reset/R|reset
BackMenu/B|backMain
    (-
Exit/E| closeWin
menu: Operations
Undo/U|undo
```

```
Replay/P|replay
Insertion/I| insert
Deletion/D| delete
Searching/S| search
menu: Speed
SpeedUp/V|speedUp
SpeedDown/W|speedDown
CloseSpeedButton|closeSpeed
ShowSpeedButton|showSpeed
menu: Shows
singleRotateWithLeft| SLrotation
singleRotateWithRight| SRrotation
doubleRotateWithLeft| DLrotation
doubleRotateWithRight| DRrotation
Operation History| history
menu: Help
Instruction| instruction
Definitions of AVL Tree| Definition
AboutMultimediaAvlTree| aboutAvlTree


--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                      Insert Function                       +
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on insert
  global gNodeList, gHistoryList, gSpeedGrade
  global gKeyList, gOpNameList, gUndoFlag, gReplayFlag

  --check if the NodeList is full
  if count(gNodeList)=31 then
    puppetsound "treeful"
    updateStage
    alert "Sorry, The Tree is full!"
    exit
  end if

  --let Token = input data
  set Token to value(the text of member"InputField")

  --check if Token is blank or a space
  if Token="" or Token=" " then
    puppetsound "inputdata"
    updateStage
    alert "Please enter a data value in the box."
    exit
  end if

  --remove input data from InputField
  put "" into field "InputField"
  updateStage

  if gUndoFlag=0 then
    puppetsound "insert"
    updateStage
  end if
```

```
--active compare,movingNode,and movingToken
puppetSprite 33, True
puppetSprite 34, True
puppetSprite 32, True

--close "winAB" window
close window "winAB"

if count(gNodeList)=0 then
  set the visible of sprite 1 to True
  set the visible of sprite 65 to True
  set the text of member "value1" = string(Token)
  set the enabled of menuItem "undo" of menu "Operations" to True
  set the enabled of menuItem "Replay" of menu "Operations" to¬
      True
  updateStage

  if gUndoFlag=0 then
    add gKeyList, Token
    add gOpNameList, "Insert"
  end if

  add gNodeList, Token
  add gHistoryList,"Insert " & string(Token)
  history
else
  if getOne(gNodeList,Token)=0 then
    set currentNum=1
    set currentValue=value(the text of member "value1")

    set the text of member "movingToken" to string(Token)

    set TreeLevelcount=-1

    repeat while voidP(currentValue)=False
      set currentSprite=currentNum
      set TreeLevelcount=TreeLevelcount+1

      --Check if tree levels are more than 4
      if TreeLevelcount>=4 and the visible of sprite (currentNum)¬
         =True then
        set the visible of sprite 32 to False
        set the visible of sprite 33 to False
        set the visible of sprite 34 to False
        puppetsound "outStage"
        updateStage
        alert "Sorry, Out of stage! Try again!"
        exit
      end if

      if gUndoFlag=0 then
        --move compare,movingToken, and movingNode to the right of
        --currentNode
        set the locH of sprite 32 to the right of sprite¬
            currentSprite
        set the locV of sprite 32 to the locV of sprite¬
            currentSprite-6
```

```
  set the locH of sprite 33 to the right of sprite¬
      currentSprite+40
  set the locV of sprite 33 to the locV of sprite¬
      currentSprite
  set the locH of sprite 34 to the locH of sprite 33-13
  set the locV of sprite 34 to the locV of sprite 33-8
end if

if currentValue > Token then
  set the text of member "Compare" to ">"
  set edge=currentNum+34
  set currentNum=currentNum*2
else
  set the text of member "Compare" to "<"
  set edge=currentNum+49
  set currentNum=currentNum*2+1
end if

if gUndoFlag=0 then
  set the visible of sprite 32 to True
  set the visible of sprite 33 to True
  set the visible of sprite 34 to True
  updateStage
end if

set temp=gSpeedGrade
set gSpeedGrade = 6
wait
set gSpeedGrade=temp

--make compare field invisible
set the visible of sprite 32 to False

--move insert node to the current node position
repeat with i=the locH of sprite 33 down to ¬
  (the locH of sprite currentSprite)
  set the locH of sprite 33 to (the locH of sprite 33-1)
  set the locH of sprite 34 to (the locH of sprite 34-1)
  wait
  updateStage
end repeat

if gUndoFlag=0 then
  --move insert node to the child position of current node
  set cnH=the locH of sprite currentNum
  set cnV=the locV of sprite currentNum
  set csH=the locH of sprite currentSprite
  set csV=the locV of sprite currentSprite

  if currentValue > Token then
    --move to left child
    moveOneNodeLeftDown(csH,csV,cnH,cnV)

  else
    --move to right child
    moveOneNodeRightDown(csH,csV,cnH,cnV)
```

```
        end if
      end if

      wait

      if the visible of sprite (currentNum)=False then
        --make the edge and current node visible
        set the visible of sprite edge to True
        set the visible of sprite currentNum to True
        set the visible of sprite currentNum+64 to True
        set the text of member 20+currentNum=string(Token)
        wait
        updateStage
        exit repeat
      else
        set currentValue=the text of member (20+currentNum)
      end if
    end repeat

    updateStage
    wait

    adjustbalance(currentNum)

    set the visible of sprite 33 to False
    set the visible of sprite 34 to False

    if gUndoFlag=0 and gReplayFlag=0 then
      add gKeyList, Token
      add gOpNameList, "Insert"
    end if

    add gNodeList, Token
    add gHistoryList, "Insert " & string(Token)
  else
    puppetsound "noinsert"
    updateStage
    alert "This node is already in the tree."
  end if
end if

history

end insert

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on adjustBalance thisNum

  set thisParent=thisNum/2
  set GP=thisParent/2

  if GP<1 then
    exit
  end if

  repeat while GP<>0
```

```
    if(abs(getLeftHeigh(GP)-getRightHeigh(GP))=2) then

      --need rotation
      if((thisParent mod 2)=0 and (thisNum mod 2)=0) then
        --case 1
        puppetsound "slrotate"
        SLrotation
        updateStage
        wait
        singleRotateWithLeft(GP)
      else if((thisParent mod 2)<>0 and (thisNum mod 2)=0) then
        --case 2
        puppetsound "drrotate"
        DRrotation
        updateStage
        wait
        DoubleRotateWithRight(GP)
      else if((thisParent mod 2)=0 and (thisNum mod 2)<>0) then
        --case 3
        puppetsound "dlrotate"
        DLrotation
        updateStage
        wait
        DoubleRotateWithLeft(GP)
      else if((thisParent mod 2)<>0 and (thisNum mod 2)<>0) then
        --case 4
        puppetsound "srrotate"
        SRrotation
        updateStage
        wait
        singleRotateWithRight(GP)
      end if

    else
      --up one level
      set thisNum=thisParent
      set thisParent=thisNum/2
      set GP=thisParent/2

    end if

  end repeat

end adjustBalance


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                        Delete Function                     +
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on delete
  global gNodeList, gHistoryList, gSpeedGrade
  global gKeyList, gOpNameList, gReplayFlag, gUndoFlag

  --let Tokken = input data
  set Token to value(the text of member"InputField")
```

```
  --check if Token is blank or a space
  if Token="" or Token=" " then
    puppetsound "inputdata"
    updateStage
    alert "Please enter an data in the box."
    exit
  end if

  --check if delete node is not in the Tree
  if getOne(gNodeList, Token)=0 then
    puppetsound "nodel"
    updateStage
    alert "This node is not in the tree, please try again!"
    exit
  end if

  if gReplayFlag=0 and gUndoFlag=0 then
    puppetsound "delete"
    updateStage
  end if

  --remove input data from InputField
  put "" into field "InputField"
  updateStage

  --close "winAB" window
  close window "winAB"

  --active compare,movingNode,and movingToken
  puppetSprite 33, True
  puppetSprite 34, True
  puppetSprite 32, True

  --set the backColor of movingNode to the backColor of stage
  --set the backColor of sprite 8 to 43

  --initialize current node and movingToken
  set currentNum=1
  set currentValue=value(the text of member "value1")
  set the text of member "movingToken" to string(Token)

  --the root is the delete node
  if currentValue=string(Token) then
    set the text of member "Compare" to "="

    if gUndoFlag=0 then
      --move compare,movingToken, and movingNode to the right of
root
      set the locH of sprite 32 to the right of sprite 1
      set the locV of sprite 32 to the locV of sprite 1-6
      set the locH of sprite 33 to the right of sprite 1+40
      set the locV of sprite 33 to the locV of sprite 1
      set the locH of sprite 34 to the locH of sprite 33-13
      set the locV of sprite 34 to the locV of sprite 33-8

      set the visible of sprite 32 to True
```

```
    set the visible of sprite 33 to True
    set the visible of sprite 34 to True
    updateStage
  end if

  set temp=gSpeedGrade
  set gSpeedGrade = 6
  wait
  set gSpeedGrade=temp

  --make compare field invisible
  set the visible of sprite 32 to False

  if gUndoFlag=0 then
    --move delete node to the root
    repeat with i=the locH of sprite 33 down to ¬
        (the locH of sprite 1)
      set the locH of sprite 33 to (the locH of sprite 33-1)
      set the locH of sprite 34 to (the locH of sprite 34-1)
      wait
      updateStage
    end repeat
  end if

end if

set TreeHeight=0

--this big loop is for finding delete node
repeat while currentValue<>string(Token)
  set currentSprite=currentNum
  set TreeHeight=TreeHeight+1

  if gUndoFlag=0 then
    --move compare,movingToken, and movingNode to the right of
    --currentNode
    set the locH of sprite 32 to the right of sprite currentSprite
    set the locV of sprite 32 to the locV of sprite¬
        currentSprite-6
    set the locH of sprite 33 to the right of sprite¬
        currentSprite+40
    set the locV of sprite 33 to the locV of sprite currentSprite
    set the locH of sprite 34 to the locH of sprite 33-13
    set the locV of sprite 34 to the locV of sprite 33-8
  end if


  if currentValue > Token then
    set the text of member "Compare" to ">"
    set edge=currentNum+34
    set currentNum=currentNum*2
  else
    set the text of member "Compare" to "<"
    set edge=currentNum+49
    set currentNum=currentNum*2+1
  end if
```

```
  if gUndoFlag=0 then
    set the visible of sprite 32 to True
    set the visible of sprite 33 to True
    set the visible of sprite 34 to True
    updateStage
  end if

  set temp=gSpeedGrade
  set gSpeedGrade = 6
  wait
  set gSpeedGrade=temp

  --make compare field invisible
  set the visible of sprite 32 to False

  if gUndoFlag=0 then
    --move delete node to the current node position
    repeat with i=the locH of sprite 33 down to ¬
        (the locH of sprite currentSprite)
      set the locH of sprite 33 to (the locH of sprite 33-1)
      set the locH of sprite 34 to (the locH of sprite 34-1)
      wait
      updateStage
    end repeat


    --move delete node to the child position of current node
    set cnH=the locH of sprite currentNum
    set cnV=the locV of sprite currentNum
    set csH=the locH of sprite currentSprite
    set csV=the locV of sprite currentSprite

    if currentValue > Token then
      --move to left child
      moveOneNodeLeftDown(csH,csV,cnH,cnV)
    else
      --move to right child
      moveOneNodeRightDown(csH,csV,cnH,cnV)
    end if

    wait
  end if

  if the text of member (20+currentNum)=String(Token) then
    set the locH of sprite 33 to the locH of sprite currentNum
    set the locV of sprite 33 to the locV of sprite currentNum
    set the visible of sprite 34 to False
    wait
    updateStage
    exit repeat
  else
    set currentValue=the text of member(20+currentNum)
  end if

end repeat

set deletFlag=0
```

```
set node=currentNum

--delete this node in many cases
case TreeHeight of

  4:--deleted node is a leaf in deep 4
    deleteNode(currentNum,edge)
  3:--deleted node is a leaf
    if (the visible of sprite (2*currentNum)=False and¬
      the visible of sprite (2*currentNum+1)=False) then
      deleteNode(currentNum,edge)
      set deletFlag=1

      --deleted node has a left child
    else if (the visible of sprite (2*currentNum)=True and¬
      the visible of sprite (2*currentNum+1)=False) then
      set the visible of sprite (currentNum+34)=False
      nodeRightUp((2*currentNum),(currentNum))
      set deletFlag=0

      --deleted node has a right child
    else if (the visible of sprite (2*currentNum)=False and¬
      the visible of sprite (2*currentNum+1)=True) then
      set the visible of sprite (currentNum+49)=False
      nodeLeftUp((2*currentNum+1),(currentNum))
      set deletFlag=0

      --deleted node has both left and right child as leaf
    else if (the visible of sprite (2*currentNum)=True and¬
      the visible of sprite (2*currentNum+1)=True) then
      set the visible of sprite (currentNum+49)=False
      nodeLeftUp((2*currentNum+1),(currentNum))
      set deletFlag=0

    end if

  2:--deleted node is a leaf
    if (the visible of sprite (2*currentNum)=False and¬
      the visible of sprite (2*currentNum+1)=False) then
      deleteNode(currentNum,edge)
      set deletFlag=1

      --deleted node only has a left subtree
    else if (the visible of sprite (2*currentNum)=True and¬
      the visible of sprite (2*currentNum+1)=False) then
      deleteNode(currentNum)
      set the visible of sprite (currentNum+34) to False
      rightUp3node(2*currentNum,currentNum)
      set deletFlag=1

      --deleted node only has a right subtree
    else if (the visible of sprite (2*currentNum)=False and¬
      the visible of sprite (2*currentNum+1)=True) then
      deleteNode(currentNum)
      leftUp3node((2*currentNum+1),currentNum)
      set deletFlag=1
```

```
    --deleted node has both left and right subtree
  else if (the visible of sprite (2*currentNum)=True and¬
    the visible of sprite (2*currentNum+1)=True) then

    if the visible of sprite (4*currentNum+2)=True then
      --right child has a left leaf
      set the visible of sprite (2*currentNum+35) to False
      nodeLeftUp((4*currentNum+2),(currentNum))
      set deletFlag=0
    else
      --right child has not a left leaf
      set the visible of sprite (currentNum+49) to False
      nodeLeftUp((2*currentNum+1),currentNum)
      set node=(2*currentNum+1)
      set deletFlag=1
      if the visible of sprite (4*currentNum+3)=True then
        --right child has a right leaf
        set the visible of sprite (2*currentNum+50) to False
        set the visible of sprite (currentNum+49) to True
        nodeLeftUp((4*currentNum+3),(2*currentNum+1))
        set deletFlag=0
      end if
    end if
  end if


1:--deleted node is a leaf
  if (the visible of sprite (2*currentNum)=False and¬
    the visible of sprite (2*currentNum+1)=False) then
    deleteNode(currentNum,edge)
    set deletFlag=1

    --deleted node only has a left subtree

  else if (the visible of sprite (2*currentNum)=True and¬
    the visible of sprite (2*currentNum+1)=False) then


    set the visible of sprite (currentNum+34) to False
    nodeRightUp(2*currentNum,currentNum)
    set deletFlag=1


    if the visible of sprite (4*currentNum+1)=True then
      --left child has a right subtree
      set the visible of sprite (2*currentNum+49) to False
      set the visible of sprite (currentNum+49) to True
      nodeRightUp((4*currentNum+1),(2*currentNum+1))

      if the visible of sprite (8*currentNum+3)=True then
        --left child has a right left child
        set the visible of sprite (4*currentNum+50) to False
        set the visible of sprite (2*currentNum+50) to True
        nodeRightUp((8*currentNum+3),(4*currentNum+3))
        set deletFlag=0
      end if
```

```
    if the visible of sprite (8*currentNum+2)=True then
      --left child has a right right child
      set the visible of sprite (4*currentNum+35) to False
      set the visible of sprite (2*currentNum+35) to True
      nodeRightUp((8*currentNum+2),(4*currentNum+2))
      set deletFlag=0
    end if

  end if

  --left child has a left subtree
  if the visible of sprite (4*currentNum)=True then
    set the visible of sprite (currentNum+34) to True
    rightUp3node(4*currentNum,2*currentNum)
  end if



  --deleted node only has a right subtree
else if (the visible of sprite (2*currentNum)=False and¬
  the visible of sprite (2*currentNum+1)=True) then

  set the visible of sprite (currentNum+49) to False
  nodeLeftUp((2*currentNum+1),currentNum)
  set deletFlag=1


  if the visible of sprite (4*currentNum+2)=True then
    --right child has a left subtree
    set the visible of sprite (2*currentNum+35) to False
    set the visible of sprite (currentNum+34) to True
    nodeLeftUp((4*currentNum+2),(2*currentNum))

    if the visible of sprite (8*currentNum+4)=True then
      --right child has a left left child
      set the visible of sprite (4*currentNum+36) to False
      set the visible of sprite (2*currentNum+34) to True
      nodeLeftUp((8*currentNum+4),(4*currentNum))
      set deletFlag=0
    end if

    if the visible of sprite (8*currentNum+5)=True then
      --right child has a left right child
      set the visible of sprite (4*currentNum+51) to False
      set the visible of sprite (2*currentNum+49) to True
      nodeLeftUp((8*currentNum+5),(4*currentNum+1))
      set deletFlag=0
    end if

  end if

  --right child has a right subtree
  if the visible of sprite (4*currentNum+3)=True then
    set the visible of sprite (currentNum+49) to True
    leftUp3node((4*currentNum+3),(2*currentNum+1))
  end if
```

```
    --deleted node has both left and right subtree
  else if(the visible of sprite (2*currentNum)=True and¬
    the visible of sprite (2*currentNum+1)=True) then


    set minNode=findMinFromRightSubtree(currentNum)
    set node=minNode
    set deletFlag=1

    if minNode=(8*currentNum+4) then
      --right child has a left left child
      set the visible of sprite (4*currentNum+36) to False
      nodeLeftUp((8*currentNum+4),(currentNum))
      set deletFlag=0

    else if minNode=(4*currentNum+2) then
      --right child has a left child
      set the visible of sprite (2*currentNum+35) to False
      nodeLeftUp((4*currentNum+2),(currentNum))

      if the visible of sprite (8*currentNum+5)=True then
        --right child has a left right child
        set the visible of sprite (4*currentNum+51) to False
        set the visible of sprite (2*currentNum+35) to True
        nodeLeftUp((8*currentNum+5),(4*currentNum+2))
        set deletFlag=0
      end if

    else if minNode=(2*currentNum+1) then
      --right child has not a left subtree
      set the visible of sprite (currentNum+49) to False
      nodeLeftUp((2*currentNum+1),currentNum)

      if the visible of sprite (4*currentNum+3)=True then
        --right child has a right subtree
        set the visible of sprite (currentNum+49) to True
        set the visible of sprite (2*currentNum+50) to False
        leftUp3node((4*currentNum+3),(2*currentNum+1))
      end if
    end if
  end if


0:--deleted node is a leaf
  if (the visible of sprite (2*currentNum)=False and¬
    the visible of sprite (2*currentNum+1)=False) then
    deleteNode(currentNum,edge)

    --deleted node only has a left subtree

  else if (the visible of sprite (2*currentNum)=True and¬
    the visible of sprite (2*currentNum+1)=False) then


    set the visible of sprite (currentNum+34) to False
    rightUp3node(2*currentNum,currentNum)
```

```
if the visible of sprite (8*currentNum+3)=True then
  --left child has a right right subtree
  set the visible of sprite (4*currentNum+50) to False
  set the visible of sprite (2*currentNum+50) to True
  nodeRightUp((8*currentNum+3),(4*currentNum+3))

  if the visible of sprite (16*currentNum+7)=True then
    --left child has a right right right leaf
    set the visible of sprite (8*currentNum+52) to False
    set the visible of sprite (4*currentNum+52) to True
    nodeRightUp((16*currentNum+7),(8*currentNum+7))
  end if

  if the visible of sprite (16*currentNum+6)=True then
    --left child has a right right left leaf
    set the visible of sprite (8*currentNum+37) to False
    set the visible of sprite (4*currentNum+37) to True
    nodeRightUp((16*currentNum+6),(8*currentNum+6))
  end if

end if

if the visible of sprite (8*currentNum+2)=True then
  --left child has a right left subtree
  set the visible of sprite (4*currentNum+35) to False
  set the visible of sprite (2*currentNum+35) to True
  nodeRightUp((8*currentNum+2),(4*currentNum+2))

  if the visible of sprite (16*currentNum+5)=True then
    --left child has a right left right leaf
    set the visible of sprite (8*currentNum+51) to False
    set the visible of sprite (4*currentNum+51) to True
    nodeRightUp((16*currentNum+5),(8*currentNum+5))
  end if

  if the visible of sprite (16*currentNum+4)=True then
    --left child has a right left left leaf
    set the visible of sprite (8*currentNum+36) to False
    set the visible of sprite (4*currentNum+36) to True
    nodeRightUp((16*currentNum+4),(8*currentNum+4))
  end if

end if

if the visible of sprite (8*currentNum+1)=True then
  --left child has a left right subtree
  set the visible of sprite (4*currentNum+49) to False
  set the visible of sprite (2*currentNum+49) to True
  nodeRightUp((8*currentNum+1),(4*currentNum+1))

  if the visible of sprite (16*currentNum+3)=True then
    --left child has a left right right leaf
    set the visible of sprite (8*currentNum+50) to False
    set the visible of sprite (4*currentNum+50) to True
    nodeRightUp((16*currentNum+3),(8*currentNum+3))
```

```
    end if

    if the visible of sprite (16*currentNum+2)=True then
      --left child has a left right left leaf
      set the visible of sprite (8*currentNum+35) to False
      set the visible of sprite (4*currentNum+35) to True
      nodeRightUp((16*currentNum+2),(8*currentNum+2))
    end if

  end if

  --left child has a left left subtree
  if the visible of sprite (8*currentNum)=True then
    set the visible of sprite (2*currentNum+34) to True
    rightUp3node(8*currentNum,4*currentNum)
  end if



  --deleted node only has a right subtree
else if (the visible of sprite (2*currentNum)=False and¬
  the visible of sprite (2*currentNum+1)=True) then

  set the visible of sprite (currentNum+49) to False
  leftUp3node((2*currentNum+1),currentNum)


  if the visible of sprite (8*currentNum+4)=True then
    --right child has a left left subtree
    set the visible of sprite (4*currentNum+36) to False
    set the visible of sprite (2*currentNum+34) to True
    nodeLeftUp((8*currentNum+4),(4*currentNum))

    if the visible of sprite (16*currentNum+8)=True then
      --right child has a left left left leaf
      set the visible of sprite (8*currentNum+38) to False
      set the visible of sprite (4*currentNum+34) to True
      nodeLeftUp((16*currentNum+8),(8*currentNum))
    end if

    if the visible of sprite (16*currentNum+9)=True then
      --right child has a left left right leaf
      set the visible of sprite (8*currentNum+53) to False
      set the visible of sprite (4*currentNum+49) to True
      nodeLeftUp((16*currentNum+9),(8*currentNum+1))
    end if

  end if

  if the visible of sprite (8*currentNum+5)=True then
    --right child has a left right subtree
    set the visible of sprite (4*currentNum+51) to False
    set the visible of sprite (2*currentNum+49) to True
    nodeLeftUp((8*currentNum+5),(4*currentNum+1))

    if the visible of sprite (16*currentNum+10)=True then
      --right child has a left right left leaf
```

```
      set the visible of sprite (8*currentNum+39) to False
      set the visible of sprite (4*currentNum+35) to True
      nodeLeftUp((16*currentNum+10),(8*currentNum+2))
    end if

    if the visible of sprite (16*currentNum+11)=True then
      --right child has a left right right leaf
      set the visible of sprite (8*currentNum+54) to False
      set the visible of sprite (4*currentNum+50) to True
      nodeLeftUp((16*currentNum+11),(8*currentNum+3))
    end if

  end if

  if the visible of sprite (8*currentNum+6)=True then
    --right child has a right left subtree
    set the visible of sprite (4*currentNum+37) to False
    set the visible of sprite (2*currentNum+35) to True
    nodeLeftUp((8*currentNum+6),(4*currentNum+2))

    if the visible of sprite (16*currentNum+12)=True then
      --right child has a right left left leaf
      set the visible of sprite (8*currentNum+40) to False
      set the visible of sprite (4*currentNum+36) to True
      nodeLeftUp((16*currentNum+12),(8*currentNum+4))
    end if

    if the visible of sprite (16*currentNum+13)=True then
      --right child has a right left right leaf
      set the visible of sprite (8*currentNum+55) to False
      set the visible of sprite (4*currentNum+51) to True
      nodeLeftUp((16*currentNum+13),(8*currentNum+5))
    end if

  end if

  --right child has a right right subtree
  if the visible of sprite (8*currentNum+7)=True then
    set the visible of sprite (2*currentNum+50) to True
    leftUp3node((8*currentNum+7),(4*currentNum+3))
  end if



  --deleted node has both left and right subtree
else if(the visible of sprite (2*currentNum)=True and¬
  the visible of sprite (2*currentNum+1)=True) then


  set minNode=findMinFromRightSubtree(currentNum)
  set node=minNode

  if minNode=(16*currentNum+8) then
    --right child has a left left left leaf
    set the visible of sprite (8*currentNum+38) to False
    nodeLeftUp((16*currentNum+8),currentNum)
    set deletFlag=0
```

```
else if minNode=(8*currentNum+4) then
  --right child has a left left subtree
  set the visible of sprite (4*currentNum+36) to False
  nodeLeftUp((8*currentNum+4),currentNum)
  set deletFlag=1

  if the visible of sprite (16*currentNum+9)=True then
    --right child has a left left right leaf
    set the visible of sprite (8*currentNum+53) to False
    set the visible of sprite (4*currentNum+36) to True
    nodeLeftUp((16*currentNum+9),(8*currentNum+4))
    set deletFlag=0
  end if

else if minNode=(4*currentNum+2) then
  --right child has not a left left subtree
  set the visible of sprite (2*currentNum+35) to False
  nodeLeftUp((4*currentNum+2),currentNum)
  set deletFlag=1

  if the visible of sprite (8*currentNum+5)=True then
    --right child has a left right subtree
    set the visible of sprite (2*currentNum+35) to True
    set the visible of sprite (4*currentNum+51) to False
    leftUp3node((8*currentNum+5),(4*currentNum+2))
  end if

else if minNode=(2*currentNum+1) then
  --right child has not a left subtree
  set the visible of sprite (currentNum+49) to False
  nodeLeftUp((2*currentNum+1),currentNum)
  set deletFlag=1


  if the visible of sprite (4*currentNum+3)=True then
    --right child has a right subtree
    set the visible of sprite (2*currentNum+50) to False
    set the visible of sprite (currentNum+49) to True
    nodeLeftUp((4*currentNum+3),(2*currentNum+1))

    if the visible of sprite (8*currentNum+6)=True then
      --right child has a right left subtree
      set the visible of sprite (4*currentNum+37) to False
      set the visible of sprite (2*currentNum+35) to True
      nodeLeftUp((8*currentNum+6),(4*currentNum+2))

      if the visible of sprite (16*currentNum+12)=True then
        --right child has a right left left leaf
        set the visible of sprite (8*currentNum+40) to False
        set the visible of sprite (4*currentNum+36) to True
        nodeLeftUp((16*currentNum+12),(8*currentNum+4))
        set deletFlag=0
      end if

      if the visible of sprite (16*currentNum+13)=True then
        --right child has a right left right leaf
```

```
                set the visible of sprite (8*currentNum+55) to False
                set the visible of sprite (4*currentNum+51) to True
                nodeLeftUp((16*currentNum+13),(8*currentNum+5))
                set deletFlag=0
              end if

            end if


            if the visible of sprite (8*currentNum+7)=True then
              --right child has a right right subtree
              set the visible of sprite (2*currentNum+50) to True
              leftUp3node((8*currentNum+7),(4*currentNum+3))
            end if

          end if

        end if
      end if

  end case

  if deletFlag=1 then
    deletReBalance(node)
  end if

  if gReplayFlag=0 and gUndoFlag=0 then
    add gKeyList, Token
    add gOpNameList, "delete"
  end if

  add gHistoryList,"Delete " &string(Token)
  deleteOne gNodeList,Token

  history

end delete


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on deletReBalance thisNum

  if (thisNum mod 2)=0 then
    set GP=thisNum/2
    set thisP=2*GP+1
    if the visible of sprite (2*thisP+1) = True then
      set thisNum =2*thisP+1
    else
      set thisNum=2*thisP
    end if
  else
    set GP=thisNum/2
    set thisP=2*GP
    if the visible of sprite (2*thisP) = True then
      set thisNum =2*thisP
    else
```

```
    set thisNum=2*thisP+1
  end if
end if

if GP<1 then
  exit
end if

repeat while GP<>0
  if GP=1 then
    if (getLeftHeigh(1)-getRightHeigh(1))=2 then
      set thisP=2
      set GP=1
      if (getLeftHeigh(2)>= getRightHeigh(2)) then
        set thisNum=4
      else
        set thisNum=5
      end if
    else if (getLeftHeigh(1)-getRightHeigh(1))=-2 then
      set thisP=3
      set GP=1
      if (getLeftHeigh(3)<= getRightHeigh(3)) then
        set thisNum=7
      else
        set thisNum=6
      end if
    end if
  end if


  if(abs(getLeftHeigh(GP)-getRightHeigh(GP))=2) then

    --need rotation
    if((thisP mod 2)=0 and (thisNum mod 2)=0) then
      --case 1
      puppetsound "slrotate"
      SLrotation
      wait
      updateStage
      singleRotateWithLeft(GP)
    else if((thisP mod 2)<>0 and (thisNum mod 2)=0) then
      --case 2
      puppetsound "drrotate"
      DRrotation
      wait
      updateStage
      DoubleRotateWithRight(GP)
    else if((thisP mod 2)=0 and (thisNum mod 2)<>0) then
      --case 3
      puppetsound "dlrotate"
      DLrotation
      wait
      updateStage
      DoubleRotateWithLeft(GP)
    else if((thisP mod 2)<>0 and (thisNum mod 2)<>0) then
      --case 4
      puppetsound "srrotate"
```

```
        SRrotation
        wait
        updateStage
        singleRotateWithRight(GP)
      end if

    else
      --up one level
      set thisNum=thisP
      set thisP=thisNum/2
      set GP=thisP/2

    end if

  end repeat

end deletReBalance


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                    Search Function                         +
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
on search
  global gNodeList, gHistoryList, gSpeedGrade
  global gKeyList, gOpNameList, gReplayflag

  --let Tokken = input data
  set Token to value(the text of member"InputField")

  --check if Token is blank or a space
  if Token="" or Token=" " then
    puppetsound "inputdata"
    updateStage
    alert "Please enter an data in the box."
    exit
  end if

  --check if search node is not in the Tree
  if getOne(gNodeList, Token)=0 then
    puppetsound "nodel"
    updateStage
    alert "This node is not in the tree, please try again!"
    exit
  end if

  if gReplayFlag=0 then
    puppetsound "search"
    updateStage
  end if

  --remove input data from InputField
  put "" into field "InputField"
  updateStage

  --close "winAB" window
  close window "winAB"
```

```
--active compare,movingNode,and movingToken
puppetSprite 33, True
puppetSprite 34, True
puppetSprite 32, True

--set the backColor of movingNode to the backColor of stage
set the backColor of sprite 8 to 43

--initialize current node and movingToken
set currentNum=1
set currentValue=value(the text of member "value1")
set the text of member "movingToken" to string(Token)

--the root is the search node
if currentValue=string(Token) then
  set the text of member "Compare" to "="

  --move compare,movingToken, and movingNode to the right of root
  set the locH of sprite 32 to the right of sprite 1
  set the locV of sprite 32 to the locV of sprite 1-6
  set the locH of sprite 33 to the right of sprite 1+40
  set the locV of sprite 33 to the locV of sprite 1
  set the locH of sprite 34 to the locH of sprite 33-13
  set the locV of sprite 34 to the locV of sprite 33-8

  set the visible of sprite 32 to True
  set the visible of sprite 33 to True
  set the visible of sprite 34 to True
  updateStage

  set temp=gSpeedGrade
  set gSpeedGrade = 6
  wait
  set gSpeedGrade=temp

  --make compare field invisible
  set the visible of sprite 32 to False

  --move delete node to the root
  repeat with i=the locH of sprite 33 down to ¬
        (the locH of sprite 1)
    set the locH of sprite 33 to (the locH of sprite 33-1)
    set the locH of sprite 34 to (the locH of sprite 34-1)
    wait
    updateStage
  end repeat

end if



repeat while currentValue<>string(Token)
  set currentSprite=currentNum

  --move compare,movingToken, and movingNode to the right of
  --currentNode
```

```
set the locH of sprite 32 to the right of sprite currentSprite
set the locV of sprite 32 to the locV of sprite currentSprite-6
set the locH of sprite 33 to the right of sprite¬
    currentSprite+40
set the locV of sprite 33 to the locV of sprite currentSprite
set the locH of sprite 34 to the locH of sprite 33-13
set the locV of sprite 34 to the locV of sprite 33-8


if currentValue > Token then
  set the text of member "Compare" to ">"
  set currentNum=currentNum*2
else
  set the text of member "Compare" to "<"
  set currentNum=currentNum*2+1
end if

set the visible of sprite 32 to True
set the visible of sprite 33 to True
set the visible of sprite 34 to True
updateStage

set temp=gSpeedGrade
set gSpeedGrade = 6
wait
set gSpeedGrade=temp

--make compare field invisible
set the visible of sprite 32 to False

--move delete node to the current node position
repeat with i=the locH of sprite 33 down to ¬
      (the locH of sprite currentSprite)
  set the locH of sprite 33 to (the locH of sprite 33-1)
  set the locH of sprite 34 to (the locH of sprite 34-1)
  wait
  updateStage
end repeat


--move delete node to the child position of current node
set cnH=the locH of sprite currentNum
set cnV=the locV of sprite currentNum
set csH=the locH of sprite currentSprite
set csV=the locV of sprite currentSprite
set Hmove=abs(csH-cnH)
set Vmove=abs(csV-cnV)

if currentValue > Token then
  --move to left child
  set yy=0.00
  repeat with i=1 to Hmove
    set the locH of sprite 33 to (the locH of sprite 33-1)
    set the locH of sprite 34 to the locH of sprite 33-13
    if (i mod 4)=0 then
      set xx=integer(4*Vmove/Hmove)
```

```
        --following five lines fix the errors
        set yy=yy+(4.00*Vmove/Hmove)-xx
        if yy>=1 then
          set yy=yy-1
          set xx=xx+1
        end if

        set the locV of sprite 33 to (the locV of sprite 33+xx)
        set the locV of sprite 34 to the locV of sprite 33-8
        wait
        updateStage
      end if
    end repeat
  else
    --move to right child
    set yy=0.00
    repeat with i=1 to Hmove
      set the locH of sprite 33 to (the locH of sprite 33+1)
      set the locH of sprite 34 to the locH of sprite 33-13
      if (i mod 4)=0 then
        set xx=integer(4*Vmove/Hmove)

        --following five lines fix the errors
        set yy=yy+(4.00*Vmove/Hmove)-xx
        if yy>=1 then
          set yy=yy-1
          set xx=xx+1
        end if

        set the locV of sprite 33 to (the locV of sprite 33+xx)
        set the locV of sprite 34 to the locV of sprite 33-8
        wait
        updateStage
      end if
    end repeat
  end if

  wait

  if the text of member (20+currentNum)=String(Token) then
    set the locH of sprite 33 to the locH of sprite currentNum
    set the locV of sprite 33 to the locV of sprite currentNum
    set the visible of sprite 34 to False
    wait
    updateStage
    exit repeat
  else
    set currentValue=the text of member(20+currentNum)
  end if

end repeat

if gReplayFlag=0 then
  add gKeyList, Token
  add gOpNameList, "Insert"
end if
```

```
  add gHistoryList,"Search " &string(Token)
  history

  --flash the arrow to show the node is found
  repeat with i=1 to 8
    set the locH of sprite 96 to the right of sprite currentNum+12
    set the locV of sprite 96 to the locV of sprite currentNum
    set the visible of sprite 96 to True

    set temp=gSpeedGrade
    set gSpeedGrade = 4
    wait
    set gSpeedGrade=temp

    updateStage
    set the visible of sprite 96 to False

    set temp=gSpeedGrade
    set gSpeedGrade = 4
    wait
    set gSpeedGrade=temp

    updateStage
  end repeat

end search



--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                        Other Handlers                      +
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on getLeftHeigh thisNum
  --get thisNum left subtree heigh
  set leftHeigh=0
  set T=2*thisNum

  if the visible of sprite T =True and T<=31 then
    --it has left subtree
    set leftHeigh=1

    if T=2 or T=3 then
      set T_Heigh=1
    else if T>3 and T<=7 then
      set T_Heigh=2
    else if T>7 and T<=15 then
      set T_Heigh=3
    else if T>15 and T<=31 then
      set T_Heigh=4
    end if

    case T_Heigh of
      4:
        exit
      3:
        if (the visible of sprite (2*T)=True¬
```

```
            or the visible of sprite (2*T+1)=True) then
          set leftHeigh=2
        end if
      2:
        if (the visible of sprite (4*T)=True¬
          or the visible of sprite (4*T+1)=True¬
          or the visible of sprite (4*T+2)=True¬
          or the visible of sprite (4*T+3)=True) then
          set leftHeigh=3
        else if (the visible of sprite (2*T)=True¬
          or the visible of sprite (2*T+1)=True) then
          set leftHeigh=2
        end if
      1:
        if (the visible of sprite (8*T)=True¬
          or the visible of sprite (8*T+1)=True¬
          or the visible of sprite (8*T+2)=True¬
          or the visible of sprite (8*T+3)=True¬
          or the visible of sprite (8*T+4)=True¬
          or the visible of sprite (8*T+5)=True¬
          or the visible of sprite (8*T+6)=True¬
          or the visible of sprite (8*T+7)=True) then
          set leftHeigh=4

        else if (the visible of sprite (4*T)=True¬
          or the visible of sprite (4*T+1)=True¬
          or the visible of sprite (4*T+2)=True¬
          or the visible of sprite (4*T+3)=True) then
          set leftHeigh=3

        else if (the visible of sprite (2*T)=True¬
          or the visible of sprite (2*T+1)=True) then
          set leftHeigh=2
        end if

    end case

  end if

  return leftHeigh

end getLeftHeigh

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on getRightHeigh thisNum
  --get thisNum right subtree heigh
  set rightHeigh=0
  set T=2*thisNum+1

  if the visible of sprite T =True and T<=31 then
    --it has right subtree
    set rightHeigh=1

    if T=2 or T=3 then
      set T_Heigh=1
    else if T>3 and T<=7 then
```

```
      set T_Heigh=2
    else if T>7 and T<=15 then
      set T_Heigh=3
    else if T>15 and T<=31 then
      set T_Heigh=4
    end if

    case T_Heigh of
      4:
        exit
      3:
        if (the visible of sprite (2*T)=True¬
           or the visible of sprite (2*T+1)=True) then
          set rightHeigh=2
        end if
      2:
        if (the visible of sprite (4*T)=True¬
          or the visible of sprite (4*T+1)=True¬
          or the visible of sprite (4*T+2)=True¬
          or the visible of sprite (4*T+3)=True) then
          set rightHeigh=3
        else if (the visible of sprite (2*T)=True¬
          or the visible of sprite (2*T+1)=True) then
          set rightHeigh=2
        end if
      1:
        if (the visible of sprite (8*T)=True¬
          or the visible of sprite (8*T+1)=True¬
          or the visible of sprite (8*T+2)=True¬
          or the visible of sprite (8*T+3)=True¬
          or the visible of sprite (8*T+4)=True¬
          or the visible of sprite (8*T+5)=True¬
          or the visible of sprite (8*T+6)=True¬
          or the visible of sprite (8*T+7)=True) then
          set rightHeigh=4

        else if (the visible of sprite (4*T)=True¬
          or the visible of sprite (4*T+1)=True¬
          or the visible of sprite (4*T+2)=True¬
          or the visible of sprite (4*T+3)=True) then
          set rightHeigh=3

        else if (the visible of sprite (2*T)=True¬
          or the visible of sprite (2*T+1)=True) then
          set rightHeigh=2
        end if

    end case

  end if

  return rightHeigh

end getRightHeigh

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
on singleRotateWithLeft K2

  if the visible of sprite (2*K2+1) =True then
    if the visible of sprite (4*K2+3) =True then
      nodeRightDown((4*K2+3),(8*K2+7))
      set the visible of sprite (4*K2+52) to True
      set the visible of sprite (2*K2+50) to False
    end if

    if the visible of sprite (4*K2+2) =True then
      nodeRightDown((4*K2+2),(8*K2+6))
      set the visible of sprite (4*K2+37) to True
      set the visible of sprite (2*K2+35) to False
    end if

    nodeRightDown((2*K2+1),(4*K2+3))
    set the visible of sprite (2*K2+50) to True
    set the visible of sprite (K2+49) to False
    wait
    updateStage
  end if

  nodeRightDown(K2,(2*K2+1))
  set the visible of sprite (K2+49) to True
  set the visible of sprite (K2+34) to False

  nodeRightUp(2*K2,K2)
  set the visible of sprite (2*K2+34) to False

  if ((4*K2+1)<=31 and the visible of sprite (4*K2+1) =True) then
    set the visible of sprite (2*K2+49) to False
    set the visible of sprite (2*K2+35) to True
    rightUp3node((4*K2+1),(4*K2+2))
  end if

  if ((4*K2)<=31 and the visible of sprite (4*K2) =True) then
    set the visible of sprite (K2+34) to True
    set the visible of sprite (2*K2+34) to False
    nodeRightUp((4*K2),(2*K2))

    if ((8*K2+1)<=31 and the visible of sprite (8*K2+1) =True) then
      set the visible of sprite (4*K2+49) to False
      set the visible of sprite (2*K2+49) to True
      rightUp3node((8*K2+1),(4*K2+1))
    end if

    if (8*K2)<31 and the visible of sprite (8*K2) =True then
      set the visible of sprite (4*K2+34) to False
      set the visible of sprite (2*K2+34) to True
      rightUp3node((8*K2),(4*K2))
    end if

  end if

  wait
  updateStage
```

```
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on singleRotateWithRight K2

  if the visible of sprite (2*K2) =True then
    if the visible of sprite (4*K2) =True then
      nodeLeftDown((4*K2),(8*K2))
      set the visible of sprite (4*K2+34) to True
      set the visible of sprite (2*K2+34) to False
    end if

    if the visible of sprite (4*K2+1) =True then
      nodeLeftDown((4*K2+1),(8*K2+1))
      set the visible of sprite (4*K2+49) to True
      set the visible of sprite (2*K2+49) to False
    end if

    nodeLeftDown((2*K2),(4*K2))
    set the visible of sprite (2*K2+34)=True
    set the visible of sprite (K2+34) to False
    wait
    updateStage
  end if

  nodeLeftDown(K2,2*K2)
  set the visible of sprite (K2+34)=True
  set the visible of sprite (K2+49) to False

  nodeLeftUp((2*K2+1),K2)
  set the visible of sprite (2*K2+50) to False

  if (4*K2+2)<=31 and the visible of sprite (4*K2+2) =True then
    set the visible of sprite (2*K2+35) to False
    set the visible of sprite (2*K2+49) to True
    leftUp3node((4*K2+2),(4*K2+1))
  end if

  if (4*K2+3)<=31 and the visible of sprite (4*K2+3) =True then
    set the visible of sprite (2*K2+50) to False
    set the visible of sprite (K2+49) to True
    nodeLeftUp((4*K2+3),(2*K2+1))

    if ((8*K2+6)<=31 and the visible of sprite (8*K2+6) =True) then
      set the visible of sprite (4*K2+37) to False
      set the visible of sprite (2*K2+35) to True
      leftUp3node((8*K2+6),(4*K2+2))
    end if

    if ((8*K2+7)<=31 and the visible of sprite (8*K2+7) =True) then
      set the visible of sprite (4*K2+52) to False
      set the visible of sprite (2*K2+50) to True
      leftUp3node((8*K2+7),(4*K2+3))
    end if
```

```
  end if

  wait
  updateStage

end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on doubleRotateWithRight K1

  --rotate between K3 and K2
  singleRotateWithLeft(2*K1+1)

  --rotate between K1 and K2
  singleRotateWithRight(K1)

  wait
  updateStage

end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on doubleRotateWithLeft K3

  --rotate between K1 and K2
  singleRotateWithRight(2*K3)

  --rotate between K3 and K2
  singleRotateWithLeft(K3)

  wait
  updateStage

end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
on wait
  global gSpeedGrade

  case gSpeedGrade of
    1: --lowest animation speed
      startTimer
      repeat with x=1 to 1000000
        nothing
      end repeat
    2:
      startTimer
      repeat with x=1 to 500000
        nothing
      end repeat
    3:
      startTimer
      repeat with x=1 to 250000
```

```
    nothing
  end repeat
4:
  startTimer
  repeat with x=1 to 125000
    nothing
  end repeat
5:
  startTimer
  repeat with x=1 to 62500
    nothing
  end repeat
6:
  startTimer
  repeat with x=1 to 31250
    nothing
  end repeat
7:
  startTimer
  repeat with x=1 to 15600
    nothing
  end repeat
8:
  startTimer
  repeat with x=1 to 7800
    nothing
  end repeat
9:
  startTimer
  repeat with x=1 to 3900
    nothing
  end repeat
10:
  startTimer
  repeat with x=1 to 2000
    nothing
  end repeat
11:
  startTimer
  repeat with x=1 to 1000
    nothing
  end repeat
12:
  startTimer
  repeat with x=1 to 500
    nothing
  end repeat
13:
  startTimer
  repeat with x=1 to 300
    nothing
  end repeat
14:
  startTimer
  repeat with x=1 to 200
    nothing
  end repeat
```

```
    15:
      startTimer
      repeat with x=1 to 100
        nothing
      end repeat
    16:
      startTimer
      repeat with x=1 to 50
        nothing
      end repeat
    17:
      startTimer
      repeat with x=1 to 25
        nothing
      end repeat
    18:
      startTimer
      repeat with x=1 to 2
        nothing
      end repeat

  end case
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on speedUp
  global gSpeedGrade

  if gSpeedGrade>=18 then
    alert "This is the highest animation speed"
    exit
  else
    set gSpeedGrade=gSpeedGrade+1
  end if

  set the text of member "speedField" to string(gSpeedGrade)

end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on speedDown
  global gSpeedGrade

  if gSpeedGrade<=1 then
    alert "This is the lowest animation speed"
    exit
  else
    set gSpeedGrade=gSpeedGrade-1
  end if

  set the text of member "speedField" to string(gSpeedGrade)

end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
on closeSpeed
  set the visible of sprite 106 to False
  set the visible of sprite 107 to False
  set the visible of sprite 108 to False
  set the visible of sprite 109 to False
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on showSpeed
  set the visible of sprite 106 to True
  set the visible of sprite 107 to True
  set the visible of sprite 108 to True
  set the visible of sprite 109 to True
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on undo
  global gNodeList,gHistoryList,gSpeedGrade
  global gKeyList, gOpNameList, gUndoFlag

  set gUndoFlag=1
  reset

  set temp=gSpeedGrade
  set gSpeedGrade = 18
  closeButton

  --delete the last element of gKeyList and gOpNameList
  set x=count(gKeyList)
  deleteAt gKeyList,x
  deleteAt gOpNameList,x

  --recover AVL tree except last element
  repeat with i=1 to (x-1)

    set y=getAt(gKeyList,i)
    set the text of member "InputField" to string(y)

    if getAt(gOpNameList,i) = "Insert" then
      insert
    else if getAt(gOpNameList,i) = "Delete" then
      delete
    else
      add gHistoryList,"Search " &string(y)
      history
    end if
  end repeat

  if count(gKeyList)=0 then
    set the enabled of menuItem "undo" of menu "Operations" to False
  end if

  set gSpeedGrade=temp
  set gUndoFlag=0
```

```
end undo

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on replay
  global gNodeList,gHistoryList,gSpeedGrade
  global gKeyList, gOpNameList, gReplayFlag

  set gReplayFlag=1
  reset

  --delete the last element of gKeyList and gOpNameList
  set x=count(gKeyList)

  --recover AVL tree except last element
  repeat with i=1 to x

    set y=getAt(gKeyList,i)
    set the text of member "InputField" to string(y)

    if getAt(gOpNameList,i) = "Insert" then
      insert
    else if getAt(gOpNameList,i) = "Delete" then
      delete
    else
      search
    end if
  end repeat

  if count(gKeyList)=0 then
    set the enabled of menuItem "Replay" of menu "Operations"¬
        to False
  end if

  set gReplayFlag=0
end replay

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on deleteNode thisNode,thisEdge
  set the text of member (20+thisNode) to ""
  wait
  set the visible of sprite thisNode to False
  set the visible of sprite (thisNode+64) to False
  set the visible of sprite thisEdge to False
  set the visible of sprite 33 to False
  set the visible of sprite 34 to False
  wait
  updateStage
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on moveOneNodeLeftUp fromH,fromV,toH,toV
  global gUndoFlag

  if gUndoflag=0 then
```

```
    set Hmove=abs(fromH-toH)
    set Vmove=abs(fromV-toV)

    --move movingNode to start node
    set the locH of sprite 33 to fromH
    set the locV of sprite 33 to fromV
    set the locH of sprite 34 to the locH of sprite 33-13
    set the locV of sprite 34 to the locV of sprite 33-8
    set the visible of sprite 33 to True
    set the visible of sprite 34 to True

    wait
    updateStage

    set yy=0.00
    repeat with i=1 to Hmove
      set the locH of sprite 33 to (the locH of sprite 33-1)
      set the locH of sprite 34 to the locH of sprite 33-13
      if (i mod 4)=0 then
        set xx=integer(4*Vmove/Hmove)

        --following five lines fix the errors
        set yy=yy+(4.00*Vmove/Hmove)-xx
        if yy>=1 then
          set yy=yy-1
          set xx=xx+1
        end if

        set the locV of sprite 33 to (the locV of sprite 33-xx)
        set the locV of sprite 34 to the locV of sprite 33-8
        wait
        updateStage
      end if
    end repeat

    set the locH of sprite 33 to toH
    set the locH of sprite 34 to the locH of sprite 33-13
    set the locV of sprite 33 to toV
    set the locV of sprite 34 to the locV of sprite 33-8
    wait
    updateStage

    set the visible of sprite 33 to False
    set the visible of sprite 34 to False
  end if

end moveOneNodeLeftUp

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on moveOneNodeRightDown fromH,fromV,toH,toV
  global gUndoFlag

  if gUndoflag=0 then
    set Hmove=abs(fromH-toH)
    set Vmove=abs(fromV-toV)
```

```
    --move movingNode to start node
    set the locH of sprite 33 to fromH
    set the locV of sprite 33 to fromV
    set the locH of sprite 34 to the locH of sprite 33-13
    set the locV of sprite 34 to the locV of sprite 33-8
    set the visible of sprite 33 to True
    set the visible of sprite 34 to True
    wait
    updateStage

    set yy=0.00
    repeat with i=1 to Hmove
      set the locH of sprite 33 to (the locH of sprite 33+1)
      set the locH of sprite 34 to the locH of sprite 33-13
      if (i mod 4)=0 then
        set xx=integer(4*Vmove/Hmove)

        --following five lines fix the errors
        set yy=yy+(4.00*Vmove/Hmove)-xx
        if yy>=1 then
          set yy=yy-1
          set xx=xx+1
        end if

        set the locV of sprite 33 to (the locV of sprite 33+xx)
        set the locV of sprite 34 to the locV of sprite 33-8
        wait
        updateStage
      end if
    end repeat

    set the locH of sprite 33 to toH
    set the locH of sprite 34 to the locH of sprite 33-13
    set the locV of sprite 33 to toV
    set the locV of sprite 34 to the locV of sprite 33-8
    wait
    updateStage
  end if

end moveOneNodeRightDown

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on moveOneNodeRightUp fromH,fromV,toH,toV
  global gUndoFlag

  if gUndoflag=0 then
    set Hmove=abs(fromH-toH)
    set Vmove=abs(fromV-toV)

    --move movingNode to start node
    set the locH of sprite 33 to fromH
    set the locV of sprite 33 to fromV
    set the locH of sprite 34 to the locH of sprite 33-13
    set the locV of sprite 34 to the locV of sprite 33-8
    set the visible of sprite 33 to True
    set the visible of sprite 34 to True
```

```
    wait
    updateStage

    set yy=0.00
    repeat with i=1 to Hmove
      set the locH of sprite 33 to (the locH of sprite 33+1)
      set the locH of sprite 34 to the locH of sprite 33-13
      if (i mod 4)=0 then
        set xx=integer(4*Vmove/Hmove)

        --following five lines fix the errors
        set yy=yy+(4.00*Vmove/Hmove)-xx
        if yy>=1 then
          set yy=yy-1
          set xx=xx+1
        end if

        set the locV of sprite 33 to (the locV of sprite 33-xx)
        set the locV of sprite 34 to the locV of sprite 33-8
        wait
        updateStage
      end if
    end repeat

    set the locH of sprite 33 to toH
    set the locH of sprite 34 to the locH of sprite 33-13
    set the locV of sprite 33 to toV
    set the locV of sprite 34 to the locV of sprite 33-8
    wait
    updateStage

    set the visible of sprite 33 to False
    set the visible of sprite 34 to False
  end if

end moveOneNodeRightUp

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on moveOneNodeLeftDown fromH,fromV,toH,toV
  global gUndoFlag

  if gUndoflag=0 then
    set Hmove=abs(fromH-toH)
    set Vmove=abs(fromV-toV)

    --move movingNode to start node
    set the locH of sprite 33 to fromH
    set the locV of sprite 33 to fromV
    set the locH of sprite 34 to the locH of sprite 33-13
    set the locV of sprite 34 to the locV of sprite 33-8
    set the visible of sprite 33 to True
    set the visible of sprite 34 to True
    wait
    updateStage

    set yy=0.00
```

```
    repeat with i=1 to Hmove
      set the locH of sprite 33 to (the locH of sprite 33-1)
      set the locH of sprite 34 to the locH of sprite 33-13
      if (i mod 4)=0 then
        set xx=integer(4*Vmove/Hmove)

        --following five lines fix the errors
        set yy=yy+(4.00*Vmove/Hmove)-xx
        if yy>=1 then
          set yy=yy-1
          set xx=xx+1
        end if

        set the locV of sprite 33 to (the locV of sprite 33+xx)
        set the locV of sprite 34 to the locV of sprite 33-8
        wait
        updateStage
      end if
    end repeat

    set the locH of sprite 33 to toH
    set the locH of sprite 34 to the locH of sprite 33-13
    set the locV of sprite 33 to toV
    set the locV of sprite 34 to the locV of sprite 33-8
    wait
    updateStage
  end if

end moveOneNodeLeftDown


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on nodeRightUp thisNum,thatNum

  --move this node right up to that node
  set the visible of sprite thisNum to False
  set the visible of sprite (thisNum+64) to False
  set the text of member 10 to the text of member¬
    (thisNum+20)
  set the text of member (20+thisNum) to ""
  wait
  updateStage

  moveOneNodeRightUp(the locH of sprite thisNum,¬
    the locV of sprite thisNum,the locH of sprite¬
    thatNum,the locV of sprite thatNum)

  --set that node
  set the text of member (thatNum+20) to the text¬
    of member 10
  set the visible of sprite thatNum to True
  set the visible of sprite (thatNum+64) to True
  wait
  updateStage
end
```

```
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on nodeLeftUp thisNum,thatNum

  --remove the right child and move the right child up
  set the visible of sprite thisNum to False
  set the visible of sprite thisNum+64 to False
  set the text of member 10 to the text of member¬
    (thisNum+20)
  set the text of member (thisNum+20) to ""
  wait
  updateStage

  moveOneNodeLeftUp(the locH of sprite thisNum,¬
    the locV of sprite thisNum,the locH of sprite¬
    thatNum,the locV of sprite thatNum)

  --set this node
  set the text of member (thatNum+20) to the text¬
    of member 10
  set the visible of sprite thatNum to True
  set the visible of sprite (thatNum+64) to True
  wait
  updateStage
end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on nodeRightDown thisNum,thatNum

  --move this node right up to that node
  set the visible of sprite thisNum to False
  set the visible of sprite (thisNum+64) to False
  set the text of member 10 to the text of member¬
    (thisNum+20)
  set the text of member (20+thisNum) to ""
  wait
  updateStage

  moveOneNodeRightDown(the locH of sprite thisNum,¬
    the locV of sprite thisNum,the locH of sprite¬
    thatNum,the locV of sprite thatNum)

  --set that node
  set the text of member (thatNum+20) to the text¬
    of member 10
  set the visible of sprite thatNum to True
  set the visible of sprite (thatNum+64) to True
  wait
  updateStage
end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
on nodeLeftDown thisNum,thatNum

  --remove the right child and move the right child up
  set the visible of sprite thisNum to False
  set the visible of sprite thisNum+64 to False
  set the text of member 10 to the text of member¬
    (thisNum+20)
  set the text of member (thisNum+20) to ""
  wait
  updateStage

  moveOneNodeLeftDown(the locH of sprite thisNum,¬
    the locV of sprite thisNum,the locH of sprite¬
    thatNum,the locV of sprite thatNum)

  --set this node
  set the text of member (thatNum+20) to the text¬
    of member 10
  set the visible of sprite thatNum to True
  set the visible of sprite (thatNum+64) to True
  wait
  updateStage
end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


on findMinFromRightSubtree thisNum
  set minNode=2*thisNum+1
  set leftNode=2*minNode

  repeat while (the visible of sprite leftNode =True and¬
    leftNode<=31)
    set minNode=leftNode
    set leftNode=leftNode*2
  end repeat

  return minNode
end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


on rightUp3node fromN,toN

  nodeRightUp(fromN,toN)

  if the visible of sprite (2*fromN+1)=True then
    set the visible of sprite (fromN+49) to False
    set the visible of sprite (toN+49) to True
    nodeRightUp((2*fromN+1),(2*toN+1))
  end if

  if the visible of sprite (2*fromN)=True then
    set the visible of sprite (fromN+34) to False
    set the visible of sprite (toN+34) to true
```

```
    nodeRightUp((2*fromN),(2*toN))
  end if

end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on leftUp3node fromN,toN

  nodeLeftUp(fromN,toN)

  if the visible of sprite (2*fromN)=True then
    set the visible of sprite (fromN+34) to False
    set the visible of sprite (toN+34) to True
    nodeLeftUp((2*fromN),(2*toN))
  end if

  if the visible of sprite (2*fromN+1)=True then
    set the visible of sprite (fromN+49) to False
    set the visible of sprite (toN+49) to true
    nodeLeftUp((2*fromN+1),(2*toN+1))
  end if

end
```

APPENDIX B

BINARY SEARCH TREE LINGO CODE

```
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                                                            +
--+                    Main  Script                            +
--+                                                            +
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on startMovie

  -- global variable declarations
  global gNodeList,gHistoryList,gSpeedGrade
  global gKeyList, gOpNameList, gUndoFlag, gReplayFlag

  set gKeyList = []
  set gOpNameList = []
  set gUndoFlag=0
  set gReplayflag=0

  newTree

  set gSpeedGrade =9
  set the text of member "speedField" to string(gSpeedGrade)
  set the stageColor to 43
  set the backColor of sprite 103 to 43

  installMenu 3

end startMovie

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on reset
  global gHistoryList, gNodeList
  global gKeyList, gOpNameList, gUndoFlag, gReplayFlag

  set gHistoryList = []
  set gNodeList = []
  history

  if gUndoFlag=0 and gReplayFlag=0 then
    set gKeyList=[]
    set gOpNameList=[]
  end if

  repeat with i=1 to 95
    puppetSprite i,True
    set the visible of sprite i to False
  end repeat

  set the visible of sprite 103 to False

  repeat with i=21 to 51
```

```
    set the text of member i to ""
  end repeat

  set the text of member "InputField" to ""
end reset

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on newTree
  reset
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on backMain
  play done
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on stopmovie
  repeat with i=1 to 95
    set the visible of sprite i to False
  end repeat

  set the text of member "InputField" to ""

  repeat with i=21 to 51
    set the text of member i to ""
  end repeat
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on instruction
  puppetsound "instruction"
  updateStage
  set the modal of window "WinAB" to False
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(10,50,380,320)

  tell window "WinAB" to go to frame "instruction1"
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on aboutBSTree
  set the modal of window "WinAB" to True
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(10,50,340,290)

  tell window "WinAB" to go to frame "aboutBSTree"
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on closeWin
  set the modal of window "WinAB" to False
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(10,50,330,180)

  tell window "WinAB" to go to frame "Exit"
```

```
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on definition
  set the modal of window "WinAB" to True
  set the windowtype of window "WinAB" to 4
  set the rect of window "WinAB" to rect(10,50,390,300)

  tell window "WinAB" to go to frame "Definition1"
  open window "WinAB"
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on history
  global gHistoryList

  set operationNum to count(gHistoryList)
  if operationNum=0 then
    set operations="Operations history: "
  else
    set operations="Operations history: "¬
    & getAt(gHistoryList, 1)
    repeat with i=2 to operationNum
      put ",  " & getAt(gHistoryList,i) after operations
    end repeat
  end if

  puppetSprite 104, True
  puppetSprite 105, True
  set the visible of sprite 104 to True
  set the visible of sprite 105 to True
  set the text of member "showOpHi" to operations
  updateStage
end history

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on closeButton
  puppetSprite 104, False
  puppetSprite 105, False
  set the visible of sprite 104 to False
  set the visible of sprite 105 to False
end



--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                        Menu Script                           +
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

menu: File
New/N|newTree
Reset/R|reset
BackMenu/B|backMain
    (-
Exit/E| closeWin
menu: Speed
SpeedUp/V|speedUp
SpeedDown/W|speedDown
CloseSpeedButton|closeSpeed
ShowSpeedButton|showSpeed
```

```
menu: Operations
Undo/U|undo
Replay/P|replay
Insertion/I| insert
Deletion/D| delete
Searching/S| search
menu: Shows
Operation History| history
menu: Help
Instruction| instruction
Definitions of BSTree| Definition
AboutVisualBSTree| aboutBSTree


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                        Insert Function                      +
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on insert
  global gNodeList, gHistoryList, gSpeedGrade
  global gKeyList, gOpNameList, gUndoFlag, gReplayFlag

  --check if the NodeList is full
  if count(gNodeList)=31 then
    puppetsound "treeful"
    updateStage
    alert "Sorry, The Tree is full!"
    exit
  end if

  --let Token = input data
  set Token to value(the text of member"InputField")

  --check if Token is blank or a space
  if Token="" or Token=" " then
    puppetsound "inputdata"
    updateStage
    alert "Please enter a data value in the box."
    exit
  end if

  if gUndoFlag=0 and gReplayFlag=0 then
    puppetsound "insert"
    updateStage
  end if

  --remove input data from InputField
  put "" into field "InputField"
  updateStage

  --close "winAB" window
  close window "winAB"

  --active compare,movingNode,and movingToken
  puppetSprite 33, True
  puppetSprite 34, True
  puppetSprite 32, True

  --set the backColor of movingNode to the backColor of stage
  set the backColor of sprite 8 to 43

  if count(gNodeList)=0 then
    set the visible of sprite 1 to True
```

```
  set the visible of sprite 65 to True
  set the text of member "value1" = string(Token)
  set the enabled of menuItem "undo" of menu "Operations" to True
  set the enabled of menuItem "Replay" of menu "Operations" to True
  updateStage

  if gUndoFlag=0 and gReplayFlag=0 then
    add gKeyList, Token
    add gOpNameList, "Insert"
  end if

  add gNodeList, Token
  add gHistoryList,"Insert " & string(Token)

else
  if getOne(gNodeList,Token)=0 then
    set currentNum=1
    set currentValue=value(the text of member "value1")


    set the text of member "movingToken" to string(Token)

    set TreeLevelcount=-1

    repeat while voidP(currentValue)=False
      set currentSprite=currentNum
      set TreeLevelcount=TreeLevelcount+1

      --Check if tree levels are more than 4
      if TreeLevelcount>=4 and the visible of sprite (currentNum)¬
         =True then
        set the visible of sprite 32 to False
        set the visible of sprite 33 to False
        set the visible of sprite 34 to False
        puppetsound "outStage"
        updateStage
        alert "Sorry, Out of stage! Try again!"
        exit
      end if

      if gUndoFlag=0 then
        --move compare,movingToken, and movingNode to the right of
        --currentNode
        set the locH of sprite 32 to the right of sprite¬
            currentSprite
        set the locV of sprite 32 to the locV of sprite¬
            currentSprite-6
        set the locH of sprite 33 to the right of sprite¬
            currentSprite+40
        set the locV of sprite 33 to the locV of sprite¬
            currentSprite
        set the locH of sprite 34 to the locH of sprite 33-15
        set the locV of sprite 34 to the locV of sprite 33-7
      end if


      if currentValue > Token then
        set the text of member "Compare" to ">"
        set edge=currentNum+34
        set currentNum=currentNum*2
      else
        set the text of member "Compare" to "<"
        set edge=currentNum+49
        set currentNum=currentNum*2+1
```

```
  end if

  if gUndoFlag=0 then
    set the visible of sprite 32 to True
    set the visible of sprite 33 to True
    set the visible of sprite 34 to True
    updateStage
  end if

  set temp=gSpeedGrade
  set gSpeedGrade = 4
  wait
  set gSpeedGrade=temp

  --make compare field invisible
  set the visible of sprite 32 to False

  if gUndoFlag=0 then
    --move insert node to the current node position
    repeat with i=the locH of sprite 33 down to ¬
    (the locH of sprite currentSprite)
      set the locH of sprite 33 to (the locH of sprite 33-1)
      set the locH of sprite 34 to (the locH of sprite 34-1)
      wait
      updateStage
    end repeat
  end if

  if gUndoFlag=0 then
    --move insert node to the child position of current node
    set cnH=the locH of sprite currentNum
    set cnV=the locV of sprite currentNum
    set csH=the locH of sprite currentSprite
    set csV=the locV of sprite currentSprite

    if currentValue > Token then
      --move to left child
      moveOneNodeLeftDown(csH,csV,cnH,cnV)

    else
      --move to right child
      moveOneNodeRightDown(csH,csV,cnH,cnV)

    end if

    wait
  end if

  if the visible of sprite (currentNum)=False then
    --make the edge and current node visible
    set the visible of sprite edge to True
    set the visible of sprite currentNum to True
    set the visible of sprite currentNum+64 to True
    set the text of member 20+currentNum=String(Token)
    wait
    updateStage
    exit repeat
  else
    set currentValue=the text of member (20+currentNum)
  end if
end repeat


set the visible of sprite 33 to False
```

```
      set the visible of sprite 34 to False

      if gUndoFlag=0 and gReplayflag=0 then
        add gKeyList, Token
        add gOpNameList, "Insert"
      end if

      add gNodeList, Token
      add gHistoryList, "Insert " & string(Token)
    else
      puppetsound "noinsert"
      updateStage
      alert "This node is already in the tree."
    end if
  end if

  history

end insert


--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                      Delete Function                      +
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on delete
  global gNodeList, gHistoryList, gSpeedGrade
  global gKeyList, gOpNameList, gUndoFlag, gReplayFlag

  --let Tokken = input data
  set Token to value(the text of member"InputField")

  --check if Token is blank or a space
  if Token="" or Token=" " then
    puppetsound "inputdata"
    updateStage
    alert "Please enter an data in the box."
    exit
  end if

  --check if delete node is not in the Tree
  if getOne(gNodeList, Token)=0 then
    puppetsound "nodel"
    updateStage
    alert "This node is not in the tree, please try again!"
    exit
  end if

  if gReplayFlag=0 and gUndoFlag=0 then
    puppetsound "delete"
    updateStage
  end if

  --remove input data from InputField
  put "" into field "InputField"
  updateStage

  --close "winAB" window
  close window "winAB"

  --active compare,movingNode,and movingToken
  puppetSprite 33, True
  puppetSprite 34, True
  puppetSprite 32, True
```

```
--set the backColor of movingNode to the backColor of stage
set the backColor of sprite 8 to 43

--initialize current node and movingToken
set currentNum=1
set currentValue=value(the text of member "value1")
set the text of member "movingToken" to string(Token)

--the root is the delete node
if currentValue=string(Token) then
  set the text of member "Compare" to "="

  if gUndoFlag=0 then
    --move compare,movingToken, and movingNode to the right of root
    set the locH of sprite 32 to the right of sprite 1
    set the locV of sprite 32 to the locV of sprite 1-6
    set the locH of sprite 33 to the right of sprite 1+40
    set the locV of sprite 33 to the locV of sprite 1
    set the locH of sprite 34 to the locH of sprite 33-15
    set the locV of sprite 34 to the locV of sprite 33-7

    set the visible of sprite 32 to True
    set the visible of sprite 33 to True
    set the visible of sprite 34 to True
    updateStage

    repeat with i=1 to 30
      wait
    end repeat
  end if

  --make compare field invisible
  set the visible of sprite 32 to False

  if gUndoFlag=0 then
    --move delete node to the root
    repeat with i=the locH of sprite 33 down to ¬
        (the locH of sprite 1)
      set the locH of sprite 33 to (the locH of sprite 33-1)
      set the locH of sprite 34 to (the locH of sprite 34-1)
      wait
      updateStage
    end repeat
  end if

end if

set TreeHeight=0

--this big loop is for finding delete node
repeat while currentValue<>string(Token)
  set currentSprite=currentNum
  set TreeHeight=TreeHeight+1

  if gUndoFlag=0 then
    --move compare,movingToken, and movingNode to the right of
    --currentNode
    set the locH of sprite 32 to the right of sprite currentSprite
    set the locV of sprite 32 to the locV of sprite currentSprite-6
    set the locH of sprite 33 to the right of sprite¬
        currentSprite+40
    set the locV of sprite 33 to the locV of sprite currentSprite
    set the locH of sprite 34 to the locH of sprite 33-15
```

```
  set the locV of sprite 34 to the locV of sprite 33-7
end if


if currentValue > Token then
  set the text of member "Compare" to ">"
  set edge=currentNum+34
  set currentNum=currentNum*2
else
  set the text of member "Compare" to "<"
  set edge=currentNum+49
  set currentNum=currentNum*2+1
end if

if gUndoFlag=0 then
  set the visible of sprite 32 to True
  set the visible of sprite 33 to True
  set the visible of sprite 34 to True
  updateStage
end if

set temp=gSpeedGrade
set gSpeedGrade = 6
wait
set gSpeedGrade=temp

--make compare field invisible
set the visible of sprite 32 to False

if gUndoFlag=0 then
  --move delete node to the current node position
  repeat with i=the locH of sprite 33 down to ¬
      (the locH of sprite currentSprite)
    set the locH of sprite 33 to (the locH of sprite 33-1)
    set the locH of sprite 34 to (the locH of sprite 34-1)
    wait
    updateStage
  end repeat


  --move delete node to the child position of current node
  set cnH=the locH of sprite currentNum
  set cnV=the locV of sprite currentNum
  set csH=the locH of sprite currentSprite
  set csV=the locV of sprite currentSprite

  if currentValue > Token then
    --move to left child
    moveOneNodeLeftDown(csH,csV,cnH,cnV)
  else
    --move to right child
    moveOneNodeRightDown(csH,csV,cnH,cnV)
  end if

  wait
end if

if the text of member (20+currentNum)=String(Token) then
  set the locH of sprite 33 to the locH of sprite currentNum
  set the locV of sprite 33 to the locV of sprite currentNum
  set the visible of sprite 34 to False
  wait
  updateStage
  exit repeat
```

```
  else
    set currentValue=the text of member(20+currentNum)
  end if

end repeat

--delete this node in many cases
case TreeHeight of

  4:--deleted node is a leaf in deep 4
    deleteNode(currentNum,edge)
  3:--deleted node is a leaf
    if (the visible of sprite (2*currentNum)=False and¬
      the visible of sprite (2*currentNum+1)=False) then
      deleteNode(currentNum,edge)

      --deleted node has a left child
    else if (the visible of sprite (2*currentNum)=True and¬
      the visible of sprite (2*currentNum+1)=False) then
      set the visible of sprite (currentNum+34)=False
      nodeRightUp((2*currentNum),(currentNum))

      --deleted node has a right child
    else if (the visible of sprite (2*currentNum)=False and¬
      the visible of sprite (2*currentNum+1)=True) then
      set the visible of sprite (currentNum+49)=False
      nodeLeftUp((2*currentNum+1),(currentNum))

      --deleted node has both left and right child as leaf
    else if (the visible of sprite (2*currentNum)=True and¬
      the visible of sprite (2*currentNum+1)=True) then
      set the visible of sprite (currentNum+49)=False
      nodeLeftUp((2*currentNum+1),(currentNum))

    end if

  2:--deleted node is a leaf
    if (the visible of sprite (2*currentNum)=False and¬
      the visible of sprite (2*currentNum+1)=False) then
      deleteNode(currentNum,edge)

      --deleted node only has a left subtree
    else if (the visible of sprite (2*currentNum)=True and¬
      the visible of sprite (2*currentNum+1)=False) then
      deleteNode(currentNum)
      rightUp3node(currentNum)

      --deleted node only has a right subtree
    else if (the visible of sprite (2*currentNum)=False and¬
      the visible of sprite (2*currentNum+1)=True) then
      deleteNode(currentNum)
      leftUp3node(currentNum)

      --deleted node has both left and right subtree
    else if (the visible of sprite (2*currentNum)=True and¬
      the visible of sprite (2*currentNum+1)=True) then

      if the visible of sprite (4*currentNum+2)=True then
        --right child has a left leaf
        set the visible of sprite (2*currentNum+35) to False
        nodeLeftUp((4*currentNum+2),(currentNum))

      else
        --right child has not a left leaf
```

```
      set the visible of sprite (currentNum+49) to False
      nodeLeftUp((2*currentNum+1),currentNum)
      if the visible of sprite (4*currentNum+3)=True then
        --right child has a right leaf
        set the visible of sprite (2*currentNum+50) to False
        set the visible of sprite (currentNum+49) to True
        nodeLeftUp((4*currentNum+3),(2*currentNum+1))
      end if
    end if
  end if


1:--deleted node is a leaf
  if (the visible of sprite (2*currentNum)=False and¬
    the visible of sprite (2*currentNum+1)=False) then
    deleteNode(currentNum,edge)

    --deleted node only has a left subtree

  else if (the visible of sprite (2*currentNum)=True and¬
    the visible of sprite (2*currentNum+1)=False) then


    set the visible of sprite (currentNum+34) to False
    nodeRightUp(2*currentNum,currentNum)


    if the visible of sprite (4*currentNum+1)=True then
      --left child has a right subtree
      set the visible of sprite (2*currentNum+49) to False
      set the visible of sprite (currentNum+49) to True
      nodeRightUp((4*currentNum+1),(2*currentNum+1))

      if the visible of sprite (8*currentNum+3)=True then
        --left child has a right left child
        set the visible of sprite (4*currentNum+50) to False
        set the visible of sprite (2*currentNum+50) to True
        nodeRightUp((8*currentNum+3),(4*currentNum+3))
      end if

      if the visible of sprite (8*currentNum+2)=True then
        --left child has a right right child
        set the visible of sprite (4*currentNum+35) to False
        set the visible of sprite (2*currentNum+35) to True
        nodeRightUp((8*currentNum+2),(4*currentNum+2))
      end if

    end if

    --left child has a left subtree
    if the visible of sprite (4*currentNum)=True then
      set the visible of sprite (currentNum+34) to True
      rightUp3node(2*currentNum)
    end if



    --deleted node only has a right subtree
  else if (the visible of sprite (2*currentNum)=False and¬
    the visible of sprite (2*currentNum+1)=True) then

    set the visible of sprite (currentNum+49) to False
    nodeLeftUp((2*currentNum+1),currentNum)
```

```
  if the visible of sprite (4*currentNum+2)=True then
    --right child has a left subtree
    set the visible of sprite (2*currentNum+35) to False
    set the visible of sprite (currentNum+34) to True
    nodeLeftUp((4*currentNum+2),(2*currentNum))

    if the visible of sprite (8*currentNum+4)=True then
      --right child has a left left child
      set the visible of sprite (4*currentNum+36) to False
      set the visible of sprite (2*currentNum+34) to True
      nodeLeftUp((8*currentNum+4),(4*currentNum))
    end if

    if the visible of sprite (8*currentNum+5)=True then
      --right child has a left right child
      set the visible of sprite (4*currentNum+51) to False
      set the visible of sprite (2*currentNum+49) to True
      nodeLeftUp((8*currentNum+5),(4*currentNum+1))
    end if

  end if

  --right child has a right subtree
  if the visible of sprite (4*currentNum+3)=True then
    set the visible of sprite (currentNum+49) to True
    leftUp3node(2*currentNum+1)
  end if



  --deleted node has both left and right subtree
else if(the visible of sprite (2*currentNum)=True and¬
  the visible of sprite (2*currentNum+1)=True) then


  set minNode=findMinFromRightSubtree(currentNum)

  if minNode=(8*currentNum+4) then
    --right child has a left left child
    set the visible of sprite (4*currentNum+36) to False
    nodeLeftUp((8*currentNum+4),(currentNum))

  else if minNode=(4*currentNum+2) then
    --right child has a left child
    set the visible of sprite (2*currentNum+35) to False
    nodeLeftUp((4*currentNum+2),(currentNum))

    if the visible of sprite (8*currentNum+5)=True then
      --right child has a left right child
      set the visible of sprite (4*currentNum+51) to False
      set the visible of sprite (2*currentNum+35) to True
      nodeLeftUp((8*currentNum+5),(4*currentNum+2))
    end if

  else if minNode=(2*currentNum+1) then
    --right child has not a left subtree
    set the visible of sprite (currentNum+49) to False
    nodeLeftUp((2*currentNum+1),currentNum)

    if the visible of sprite (4*currentNum+3)=True then
      --right child has a right subtree
      set the visible of sprite (currentNum+49) to True
```

```
      leftUp3node(2*currentNum+1)
    end if
  end if
end if


0:--deleted node is a leaf
  if (the visible of sprite (2*currentNum)=False and¬
    the visible of sprite (2*currentNum+1)=False) then
    deleteNode(currentNum,edge)

    --deleted node only has a left subtree

  else if (the visible of sprite (2*currentNum)=True and¬
    the visible of sprite (2*currentNum+1)=False) then


    set the visible of sprite (currentNum+34) to False
    rightUp3node(currentNum)


    if the visible of sprite (8*currentNum+3)=True then
      --left child has a right right subtree
      set the visible of sprite (4*currentNum+50) to False
      set the visible of sprite (2*currentNum+50) to True
      nodeRightUp((8*currentNum+3),(4*currentNum+3))

      if the visible of sprite (16*currentNum+7)=True then
        --left child has a right right right leaf
        set the visible of sprite (8*currentNum+52) to False
        set the visible of sprite (4*currentNum+52) to True
        nodeRightUp((16*currentNum+7),(8*currentNum+7))
      end if

      if the visible of sprite (16*currentNum+6)=True then
        --left child has a right right left leaf
        set the visible of sprite (8*currentNum+37) to False
        set the visible of sprite (4*currentNum+37) to True
        nodeRightUp((16*currentNum+6),(8*currentNum+6))
      end if

    end if

    if the visible of sprite (8*currentNum+2)=True then
      --left child has a right left subtree
      set the visible of sprite (4*currentNum+35) to False
      set the visible of sprite (2*currentNum+35) to True
      nodeRightUp((8*currentNum+2),(4*currentNum+2))

      if the visible of sprite (16*currentNum+5)=True then
        --left child has a right left right leaf
        set the visible of sprite (8*currentNum+51) to False
        set the visible of sprite (4*currentNum+51) to True
        nodeRightUp((16*currentNum+5),(8*currentNum+5))
      end if

      if the visible of sprite (16*currentNum+4)=True then
        --left child has a right left left leaf
        set the visible of sprite (8*currentNum+36) to False
        set the visible of sprite (4*currentNum+36) to True
        nodeRightUp((16*currentNum+4),(8*currentNum+4))
      end if

    end if
```

```
  if the visible of sprite (8*currentNum+1)=True then
    --left child has a left right subtree
    set the visible of sprite (4*currentNum+49) to False
    set the visible of sprite (2*currentNum+49) to True
    nodeRightUp((8*currentNum+1),(4*currentNum+1))

    if the visible of sprite (16*currentNum+3)=True then
      --left child has a left right right leaf
      set the visible of sprite (8*currentNum+50) to False
      set the visible of sprite (4*currentNum+50) to True
      nodeRightUp((16*currentNum+3),(8*currentNum+3))
    end if

    if the visible of sprite (16*currentNum+2)=True then
      --left child has a left right left leaf
      set the visible of sprite (8*currentNum+35) to False
      set the visible of sprite (4*currentNum+35) to True
      nodeRightUp((16*currentNum+2),(8*currentNum+2))
    end if

  end if

  --left child has a left left subtree
  if the visible of sprite (8*currentNum)=True then
    set the visible of sprite (2*currentNum+34) to True
    rightUp3node(4*currentNum)
  end if



  --deleted node only has a right subtree
else if (the visible of sprite (2*currentNum)=False and¬
  the visible of sprite (2*currentNum+1)=True) then

  set the visible of sprite (currentNum+49) to False
  leftUp3node(currentNum)


  if the visible of sprite (8*currentNum+4)=True then
    --right child has a left left subtree
    set the visible of sprite (4*currentNum+36) to False
    set the visible of sprite (2*currentNum+34) to True
    nodeLeftUp((8*currentNum+4),(4*currentNum))

    if the visible of sprite (16*currentNum+8)=True then
      --right child has a left left left leaf
      set the visible of sprite (8*currentNum+38) to False
      set the visible of sprite (4*currentNum+34) to True
      nodeLeftUp((16*currentNum+8),(8*currentNum))
    end if

    if the visible of sprite (16*currentNum+9)=True then
      --right child has a left left right leaf
      set the visible of sprite (8*currentNum+53) to False
      set the visible of sprite (4*currentNum+49) to True
      nodeLeftUp((16*currentNum+9),(8*currentNum+1))
    end if

  end if

  if the visible of sprite (8*currentNum+5)=True then
    --right child has a left right subtree
    set the visible of sprite (4*currentNum+51) to False
```

```
    set the visible of sprite (2*currentNum+49) to True
    nodeLeftUp((8*currentNum+5),(4*currentNum+1))

    if the visible of sprite (16*currentNum+10)=True then
      --right child has a left right left leaf
      set the visible of sprite (8*currentNum+39) to False
      set the visible of sprite (4*currentNum+35) to True
      nodeLeftUp((16*currentNum+10),(8*currentNum+2))
    end if

    if the visible of sprite (16*currentNum+11)=True then
      --right child has a left right right leaf
      set the visible of sprite (8*currentNum+54) to False
      set the visible of sprite (4*currentNum+50) to True
      nodeLeftUp((16*currentNum+11),(8*currentNum+3))
    end if

  end if

  if the visible of sprite (8*currentNum+6)=True then
    --right child has a right left subtree
    set the visible of sprite (4*currentNum+37) to False
    set the visible of sprite (2*currentNum+35) to True
    nodeLeftUp((8*currentNum+6),(4*currentNum+2))

    if the visible of sprite (16*currentNum+12)=True then
      --right child has a right left left leaf
      set the visible of sprite (8*currentNum+40) to False
      set the visible of sprite (4*currentNum+36) to True
      nodeLeftUp((16*currentNum+12),(8*currentNum+4))
    end if

    if the visible of sprite (16*currentNum+13)=True then
      --right child has a right left right leaf
      set the visible of sprite (8*currentNum+55) to False
      set the visible of sprite (4*currentNum+51) to True
      nodeLeftUp((16*currentNum+13),(8*currentNum+5))
    end if

  end if

  --right child has a right right subtree
  if the visible of sprite (8*currentNum+7)=True then
    set the visible of sprite (2*currentNum+50) to True
    leftUp3node(4*currentNum+3)
  end if



  --deleted node has both left and right subtree
else if(the visible of sprite (2*currentNum)=True and¬
  the visible of sprite (2*currentNum+1)=True) then


  set minNode=findMinFromRightSubtree(currentNum)

  if minNode=(16*currentNum+8) then
    --right child has a left left left leaf
    set the visible of sprite (8*currentNum+38) to False
    nodeLeftUp((16*currentNum+8),currentNum)

  else if minNode=(8*currentNum+4) then
    --right child has a left left subtree
```

```
    set the visible of sprite (4*currentNum+36) to False
    nodeLeftUp((8*currentNum+4),currentNum)

    if the visible of sprite (16*currentNum+9)=True then
      --right child has a left left right leaf
      set the visible of sprite (8*currentNum+53) to False
      set the visible of sprite (4*currentNum+36) to True
      nodeLeftUp((16*currentNum+9),(8*currentNum+4))
    end if

  else if minNode=(4*currentNum+2) then
    --right child has not a left left subtree
    set the visible of sprite (2*currentNum+35) to False
    nodeLeftUp((4*currentNum+2),currentNum)

    if the visible of sprite (8*currentNum+5)=True then
      --right child has a left right subtree
      set the visible of sprite (2*currentNum+35) to True
      leftUp3node(4*currentNum+2)
    end if

  else if minNode=(2*currentNum+1) then
    --right child has not a left subtree
    set the visible of sprite (currentNum+49) to False
    nodeLeftUp((2*currentNum+1),currentNum)


    if the visible of sprite (4*currentNum+3)=True then
      --right child has a right subtree
      set the visible of sprite (2*currentNum+50) to False
      set the visible of sprite (currentNum+49) to True
      nodeLeftUp((4*currentNum+3),(2*currentNum+1))

      if the visible of sprite (8*currentNum+6)=True then
        --right child has a right left subtree
        set the visible of sprite (4*currentNum+37) to False
        set the visible of sprite (2*currentNum+35) to True
        nodeLeftUp((8*currentNum+6),(4*currentNum+2))

        if the visible of sprite (16*currentNum+12)=True then
          --right child has a right left left leaf
          set the visible of sprite (8*currentNum+40) to False
          set the visible of sprite (4*currentNum+36) to True
          nodeLeftUp((16*currentNum+12),(8*currentNum+4))
        end if

        if the visible of sprite (16*currentNum+13)=True then
          --right child has a right left right leaf
          set the visible of sprite (8*currentNum+55) to False
          set the visible of sprite (4*currentNum+51) to True
          nodeLeftUp((16*currentNum+13),(8*currentNum+5))
        end if

      end if


      if the visible of sprite (8*currentNum+7)=True then
        --right child has a right right subtree
        set the visible of sprite (2*currentNum+50) to True
        leftUp3node(4*currentNum+3)
      end if
    end if
  end if
end if
```

```
  end case

  if gUndoFlag=0 and gReplayFlag=0 then
    add gKeyList, Token
    add gOpNameList, "Delete"
  end if

  add gHistoryList,"Delete " &string(Token)
  deleteOne gNodeList,Token

  history

end delete


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                      Search Function                       +
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on search
  global gNodeList, gHistoryList, gSpeedGrade
  global gKeyList, gOpNameList, gReplayFlag

  --let Tokken = input data
  set Token to value(the text of member"InputField")

  --check if Token is blank or a space
  if Token="" or Token=" " then
    puppetsound "inputdata"
    updateStage
    alert "Please enter an data in the box."
    exit
  end if

  --check if search node is not in the Tree
  if getOne(gNodeList, Token)=0 then
    puppetsound "nodel"
    updateStage
    alert "This node is not in the tree, please try again!"
    exit
  end if

  if gReplayFlag=0 then
    puppetsound "search"
    updateStage
  end if

  --remove input data from InputField
  put "" into field "InputField"
  updateStage

  --close "winAB" window
  close window "winAB"

  --active compare,movingNode,and movingToken
  puppetSprite 33, True
  puppetSprite 34, True
  puppetSprite 32, True

  --set the backColor of movingNode to the backColor of stage
  set the backColor of sprite 8 to 43

  --initialize current node and movingToken
  set currentNum=1
```

```
set currentValue=value(the text of member "value1")
set the text of member "movingToken" to string(Token)

--the root is the search node
if currentValue=string(Token) then
  set the text of member "Compare" to "="

  --move compare,movingToken, and movingNode to the right of root
  set the locH of sprite 32 to the right of sprite 1
  set the locV of sprite 32 to the locV of sprite 1-6
  set the locH of sprite 33 to the right of sprite 1+40
  set the locV of sprite 33 to the locV of sprite 1
  set the locH of sprite 34 to the locH of sprite 33-15
  set the locV of sprite 34 to the locV of sprite 33-7

  set the visible of sprite 32 to True
  set the visible of sprite 33 to True
  set the visible of sprite 34 to True
  updateStage

  set temp=gSpeedGrade
  set gSpeedGrade = 6
  wait
  set gSpeedGrade=temp

  --make compare field invisible
  set the visible of sprite 32 to False

  --move delete node to the root
  repeat with i=the locH of sprite 33 down to ¬
        (the locH of sprite 1)
    set the locH of sprite 33 to (the locH of sprite 33-1)
    set the locH of sprite 34 to (the locH of sprite 34-1)
    wait
    updateStage
  end repeat

end if


repeat while currentValue<>string(Token)
  set currentSprite=currentNum

  --move compare,movingToken, and movingNode to the right of
  --currentNode
  set the locH of sprite 32 to the right of sprite currentSprite
  set the locV of sprite 32 to the locV of sprite currentSprite-6
  set the locH of sprite 33 to the right of sprite currentSprite+40
  set the locV of sprite 33 to the locV of sprite currentSprite
  set the locH of sprite 34 to the locH of sprite 33-15
  set the locV of sprite 34 to the locV of sprite 33-7


  if currentValue > Token then
    set the text of member "Compare" to ">"
    set currentNum=currentNum*2
  else
    set the text of member "Compare" to "<"
    set currentNum=currentNum*2+1
  end if

  set the visible of sprite 32 to True
  set the visible of sprite 33 to True
  set the visible of sprite 34 to True
```

```
updateStage

set temp=gSpeedGrade
set gSpeedGrade = 6
wait
set gSpeedGrade=temp

--make compare field invisible
set the visible of sprite 32 to False

--move delete node to the current node position
repeat with i=the locH of sprite 33 down to ¬
      (the locH of sprite currentSprite)
  set the locH of sprite 33 to (the locH of sprite 33-1)
  set the locH of sprite 34 to (the locH of sprite 34-1)
  wait
  updateStage
end repeat


--move delete node to the child position of current node
set cnH=the locH of sprite currentNum
set cnV=the locV of sprite currentNum
set csH=the locH of sprite currentSprite
set csV=the locV of sprite currentSprite
set Hmove=abs(csH-cnH)
set Vmove=abs(csV-cnV)

if currentValue > Token then
  --move to left child
  set yy=0.00
  repeat with i=1 to Hmove
    set the locH of sprite 33 to (the locH of sprite 33-1)
    set the locH of sprite 34 to the locH of sprite 33-15
    if (i mod 4)=0 then
      set xx=integer(4*Vmove/Hmove)

      --following five lines fix the errors
      set yy=yy+(4.00*Vmove/Hmove)-xx
      if yy>=1 then
        set yy=yy-1
        set xx=xx+1
      end if

      set the locV of sprite 33 to (the locV of sprite 33+xx)
      set the locV of sprite 34 to the locV of sprite 33-7
      wait
      updateStage
    end if
  end repeat
else
  --move to right child
  set yy=0.00
  repeat with i=1 to Hmove
    set the locH of sprite 33 to (the locH of sprite 33+1)
    set the locH of sprite 34 to the locH of sprite 33-15
    if (i mod 4)=0 then
      set xx=integer(4*Vmove/Hmove)

      --following five lines fix the errors
      set yy=yy+(4.00*Vmove/Hmove)-xx
      if yy>=1 then
        set yy=yy-1
        set xx=xx+1
```

```
            end if

            set the locV of sprite 33 to (the locV of sprite 33+xx)
            set the locV of sprite 34 to the locV of sprite 33-7
            wait
            updateStage
          end if
        end repeat
      end if

      wait

      if the text of member (20+currentNum)=String(Token) then
        set the locH of sprite 33 to the locH of sprite currentNum
        set the locV of sprite 33 to the locV of sprite currentNum
        set the visible of sprite 34 to False
        wait
        updateStage
        exit repeat
      else
        set currentValue=the text of member(20+currentNum)
      end if

    end repeat

    if gReplayFlag=0 then
      add gKeyList, Token
      add gOpNameList, "Search"
    end if

    add gHistoryList,"Search " &string(Token)

    history

    --flash the arrow to show the node is found
    repeat with i=1 to 8
      set the locH of sprite 103 to the right of sprite currentNum+12
      set the locV of sprite 103 to the locV of sprite currentNum
      set the visible of sprite 103 to True

      set temp=gSpeedGrade
      set gSpeedGrade = 6
      wait
      set gSpeedGrade=temp

      updateStage
      set the visible of sprite 103 to False

      set temp=gSpeedGrade
      set gSpeedGrade = 6
      wait
      set gSpeedGrade=temp

      updateStage
    end repeat

  end search
```

```
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--+                          Other Handlers                        +
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on wait
  global gSpeedGrade

  case gSpeedGrade of
    1: --lowest animation speed
      startTimer
      repeat with x=1 to 1000000
        nothing
      end repeat
    2:
      startTimer
      repeat with x=1 to 500000
        nothing
      end repeat
    3:
      startTimer
      repeat with x=1 to 250000
        nothing
      end repeat
    4:
      startTimer
      repeat with x=1 to 125000
        nothing
      end repeat
    5:
      startTimer
      repeat with x=1 to 62500
        nothing
      end repeat
    6:
      startTimer
      repeat with x=1 to 31250
        nothing
      end repeat
    7:
      startTimer
      repeat with x=1 to 15600
        nothing
      end repeat
    8:
      startTimer
      repeat with x=1 to 7800
        nothing
      end repeat
    9:
      startTimer
      repeat with x=1 to 3900
        nothing
      end repeat
    10:
      startTimer
      repeat with x=1 to 2000
        nothing
      end repeat
    11:
      startTimer
      repeat with x=1 to 1000
        nothing
      end repeat
    12:
```

```
      startTimer
      repeat with x=1 to 500
        nothing
      end repeat
    13:
      startTimer
      repeat with x=1 to 300
        nothing
      end repeat
    14:
      startTimer
      repeat with x=1 to 200
        nothing
      end repeat
    15:
      startTimer
      repeat with x=1 to 100
        nothing
      end repeat
    16:
      startTimer
      repeat with x=1 to 50
        nothing
      end repeat
    17:
      startTimer
      repeat with x=1 to 25
        nothing
      end repeat
    18:
      startTimer
      repeat with x=1 to 2
        nothing
      end repeat

  end case
end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on speedUp
  global gSpeedGrade

  if gSpeedGrade>=18 then
    alert "This Highest animation speed"
    exit
  else
    set gSpeedGrade=gSpeedGrade+1
  end if

  set the text of member "speedField" to string(gSpeedGrade)

end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on speedDown
  global gSpeedGrade

  if gSpeedGrade<=1 then
    alert "This Lowest animation speed"
    exit
  else
```

```
    set gSpeedGrade=gSpeedGrade-1
  end if

  set the text of member "speedField" to string(gSpeedGrade)

end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on closeSpeed
  set the visible of sprite 106 to False
  set the visible of sprite 107 to False
  set the visible of sprite 108 to False
  set the visible of sprite 109 to False
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on showSpeed
  set the visible of sprite 106 to True
  set the visible of sprite 107 to True
  set the visible of sprite 108 to True
  set the visible of sprite 109 to True
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on undo
  global gNodeList,gHistoryList,gSpeedGrade
  global gKeyList, gOpNameList, gUndoFlag

  set gUndoFlag=1
  reset

  set temp=gSpeedGrade
  set gSpeedGrade = 18
  closeButton

  --delete the last element of gKeyList and gOpNameList
  set x=count(gKeyList)
  deleteAt gKeyList,x
  deleteAt gOpNameList,x

  --recover AVL tree except last element
  repeat with i=1 to (x-1)

    set y=getAt(gKeyList,i)
    set the text of member "InputField" to string(y)

    if getAt(gOpNameList,i) = "Insert" then
      insert
    else if getAt(gOpNameList,i) = "Delete" then
      delete
    else
      add gHistoryList,"Search " &string(y)
      history
    end if
  end repeat

  if count(gKeyList)=0 then
    set the enabled of menuItem "undo" of menu "Operations" to False
  end if

  set gSpeedGrade=temp
```

```
  set gUndoFlag=0
end undo

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on replay
  global gNodeList,gHistoryList,gSpeedGrade
  global gKeyList, gOpNameList, gReplayFlag

  set gReplayFlag=1
  reset

  --delete the last element of gKeyList and gOpNameList
  set x=count(gKeyList)

  --recover AVL tree except last element
  repeat with i=1 to x

    set y=getAt(gKeyList,i)
    set the text of member "InputField" to string(y)

    if getAt(gOpNameList,i) = "Insert" then
      insert
    else if getAt(gOpNameList,i) = "Delete" then
      delete
    else
      search
    end if
  end repeat

  if count(gKeyList)=0 then
    set the enabled of menuItem "Replay" of menu "Operations" to
False
  end if

  set gReplayFlag=0
end replay

on deleteNode thisNode,thisEdge
  set the text of member (20+thisNode) to ""
  wait
  set the visible of sprite thisNode to False
  set the visible of sprite (thisNode+64) to False
  set the visible of sprite thisEdge to False
  set the visible of sprite 33 to False
  set the visible of sprite 34 to False
  wait
  updateStage
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on moveOneNodeLeftUp fromH,fromV,toH,toV
  global gUndoFlag

  if gUndoFlag=0 then
    set Hmove=abs(fromH-toH)
    set Vmove=abs(fromV-toV)

    --move movingNode to start node
    set the locH of sprite 33 to fromH
    set the locV of sprite 33 to fromV
    set the locH of sprite 34 to the locH of sprite 33-15
    set the locV of sprite 34 to the locV of sprite 33-7
```

```
    set the visible of sprite 33 to TRUE
    set the visible of sprite 34 to TRUE

    wait
    updateStage

    set yy=0.00
    repeat with i=1 to Hmove
      set the locH of sprite 33 to (the locH of sprite 33-1)
      set the locH of sprite 34 to the locH of sprite 33-15
      if (i mod 4)=0 then
        set xx=integer(4*Vmove/Hmove)

        --following five lines fix the errors
        set yy=yy+(4.00*Vmove/Hmove)-xx
        if yy>=1 then
          set yy=yy-1
          set xx=xx+1
        end if

        set the locV of sprite 33 to (the locV of sprite 33-xx)
        set the locV of sprite 34 to the locV of sprite 33-7
        wait
        updateStage
      end if
    end repeat

    set the locH of sprite 33 to toH
    set the locH of sprite 34 to the locH of sprite 33-15
    set the locV of sprite 33 to toV
    set the locV of sprite 34 to the locV of sprite 33-7
    wait
    updateStage

    set the visible of sprite 33 to False
    set the visible of sprite 34 to False
  end if

end moveOneNodeLeftUp

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on moveOneNodeRightDown fromH,fromV,toH,toV
  global gUndoFlag

  if gUndoFlag=0 then
    set Hmove=abs(fromH-toH)
    set Vmove=abs(fromV-toV)

    --move movingNode to start node
    set the locH of sprite 33 to fromH
    set the locV of sprite 33 to fromV
    set the locH of sprite 34 to the locH of sprite 33-15
    set the locV of sprite 34 to the locV of sprite 33-7

    wait
    updateStage

    set yy=0.00
    repeat with i=1 to Hmove
      set the locH of sprite 33 to (the locH of sprite 33+1)
      set the locH of sprite 34 to the locH of sprite 33-15
      if (i mod 4)=0 then
        set xx=integer(4*Vmove/Hmove)
```

```
        --following five lines fix the errors
        set yy=yy+(4.00*Vmove/Hmove)-xx
        if yy>=1 then
          set yy=yy-1
          set xx=xx+1
        end if

        set the locV of sprite 33 to (the locV of sprite 33+xx)
        set the locV of sprite 34 to the locV of sprite 33-7
        wait
        updateStage
      end if
    end repeat

    set the locH of sprite 33 to toH
    set the locH of sprite 34 to the locH of sprite 33-15
    set the locV of sprite 33 to toV
    set the locV of sprite 34 to the locV of sprite 33-7
    wait
    updateStage
  end if

end moveOneNodeRightDown

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on moveOneNodeRightUp fromH,fromV,toH,toV
  global gUndoFlag

  if gUndoFlag=0 then
    set Hmove=abs(fromH-toH)
    set Vmove=abs(fromV-toV)

    --move movingNode to start node
    set the locH of sprite 33 to fromH
    set the locV of sprite 33 to fromV
    set the locH of sprite 34 to the locH of sprite 33-15
    set the locV of sprite 34 to the locV of sprite 33-7
    set the visible of sprite 33 to TRUE
    set the visible of sprite 34 to TRUE
    wait
    updateStage

    set yy=0.00
    repeat with i=1 to Hmove
      set the locH of sprite 33 to (the locH of sprite 33+1)
      set the locH of sprite 34 to the locH of sprite 33-15
      if (i mod 4)=0 then
        set xx=integer(4*Vmove/Hmove)

        --following five lines fix the errors
        set yy=yy+(4.00*Vmove/Hmove)-xx
        if yy>=1 then
          set yy=yy-1
          set xx=xx+1
        end if

        set the locV of sprite 33 to (the locV of sprite 33-xx)
        set the locV of sprite 34 to the locV of sprite 33-7
        wait
        updateStage
      end if
    end repeat
```

```
    set the locH of sprite 33 to toH
    set the locH of sprite 34 to the locH of sprite 33-15
    set the locV of sprite 33 to toV
    set the locV of sprite 34 to the locV of sprite 33-7
    wait
    updateStage

    set the visible of sprite 33 to False
    set the visible of sprite 34 to False
  end if

end moveOneNodeRightUp

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on moveOneNodeLeftDown fromH,fromV,toH,toV
  global gUndoFlag

  if gUndoFlag=0 then
    set Hmove=abs(fromH-toH)
    set Vmove=abs(fromV-toV)

    --move movingNode to start node
    set the locH of sprite 33 to fromH
    set the locV of sprite 33 to fromV
    set the locH of sprite 34 to the locH of sprite 33-15
    set the locV of sprite 34 to the locV of sprite 33-7

    wait
    updateStage

    set yy=0.00
    repeat with i=1 to Hmove
      set the locH of sprite 33 to (the locH of sprite 33-1)
      set the locH of sprite 34 to the locH of sprite 33-15
      if (i mod 4)=0 then
        set xx=integer(4*Vmove/Hmove)

        --following five lines fix the errors
        set yy=yy+(4.00*Vmove/Hmove)-xx
        if yy>=1 then
          set yy=yy-1
          set xx=xx+1
        end if

        set the locV of sprite 33 to (the locV of sprite 33+xx)
        set the locV of sprite 34 to the locV of sprite 33-7
        wait
        updateStage
      end if
    end repeat

    set the locH of sprite 33 to toH
    set the locH of sprite 34 to the locH of sprite 33-15
    set the locV of sprite 33 to toV
    set the locV of sprite 34 to the locV of sprite 33-7
    wait
    updateStage
  end if

end moveOneNodeLeftDown
```

```
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on nodeRightUp thisNum,thatNum

  --move this node left up to that node
  set the visible of sprite thisNum to False
  set the visible of sprite (thisNum+64) to False
  set the text of member 10 to the text of member¬
    (thisNum+20)
  set the text of member (20+thisNum) to ""
  wait
  updateStage

  moveOneNodeRightUp(the locH of sprite thisNum,¬
    the locV of sprite thisNum,the locH of sprite¬
    thatNum,the locV of sprite thatNum)

  --set that node
  set the text of member (thatNum+20) to the text¬
    of member 10
  set the visible of sprite thatNum to True
  set the visible of sprite (thatNum+64) to True
  wait
  updateStage
end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on nodeLeftUp thisNum,thatNum

  --remove the right child and move the right child up
  set the visible of sprite thisNum to False
  set the visible of sprite thisNum+64 to False
  set the text of member 10 to the text of member¬
    (thisNum+20)
  set the text of member (thisNum+20) to ""
  wait
  updateStage

  moveOneNodeLeftUp(the locH of sprite thisNum,¬
    the locV of sprite thisNum,the locH of sprite¬
    thatNum,the locV of sprite thatNum)

  --set this node
  set the text of member (thatNum+20) to the text¬
    of member 10
  set the visible of sprite thatNum to True
  set the visible of sprite (thatNum+64) to True
  wait
  updateStage
end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on findMinFromRightSubtree thisNum
  set minNode=2*thisNum+1
  set leftNode=2*minNode

  repeat while (the visible of sprite leftNode =True and¬
    leftNode<=31)
    set minNode=leftNode
    set leftNode=leftNode*2
```

```
  end repeat

  return minNode
end


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on rightUp3node toNode
  set the visible of sprite (toNode+34) to False
  nodeRightUp((2*toNode),toNode)

  if the visible of sprite (4*toNode)=True then
    set the visible of sprite (2*toNode+34) to False
    set the visible of sprite (toNode+34) to true
    nodeRightUp((4*toNode),(2*toNode))
  end if

  if the visible of sprite (4*toNode+1)=True then
    set the visible of sprite (2*toNode+49) to False
    set the visible of sprite (toNode+49) to True
    nodeRightUp((4*toNode+1),(2*toNode+1))
  end if
end

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

on leftUp3node toNode
  set the visible of sprite (toNode+49) to False
  nodeLeftUp((2*toNode+1),toNode)

  if the visible of sprite (4*toNode+3)=True then
    set the visible of sprite (2*toNode+50) to False
    set the visible of sprite (toNode+49) to true
    nodeLeftUp((4*toNode+3),(2*toNode+1))
  end if

  if the visible of sprite (4*toNode+2)=True then
    set the visible of sprite (2*toNode+35) to False
    set the visible of sprite (toNode+34) to True
    nodeLeftUp((4*toNode+2),(2*toNode))
  end if
end
```

APPENDIX C

PROGRAMMER'S GUIDE

1. Description

The MDSL system is designed and developed for simulation the animated operations of abstract data structures as a teaching and learning tool with scientific visualization and multimedia technology. We have developed the MDSL system on the Microsoft Windows 98/NT operating system with Macromedia Director 6.0 platform. I have developed two data structures movies which are the MultimediaAvlTree (46 handlers) and the MultimediaBSTree (31 handlers) movie in this system. These movie programs were coded in Lingo script language. There are other three movies available in the system, which are VisualB-Tree, VisualRedBlackTree, and VisualADT. The MDSL system provides a good interactive user interface to let users select data structures algorithm for the multimedia simulations while they are learning.

2. Implementation

Any new data structures developed on Macromedia Director platform can be added into the MDSL system as following way:

i. Choose a data structures movie name from the Main Menu as shown in Figure 3.2 (programmer also can change the name in the Main Menu).

ii. Add the backMain menu option in the menu File item or put the backMain button on the stage.

iii. Write the backMain handler in the source code as:

```
on backMain
        Playdone
end backMain
```

vi. Open MultimediaDSLSystem movie "Internal Cast" window, double click the cast whose name is the same as that of your movie. Then click "Cast member Script" button in the "Text Cast Member Properties Window", and type your movie name instead of "NotAvailableMovie" in the mouseUp hander.

v. Put the new data structures movie in the same directory as the system in.

VITA

Dongbi (Carl) Luo

Candidate for the Degree of

Master of Science

Thesis: MULTIMEDIA DATA STRUCTURE LEARNING SYSTEM

Major Field: Computer Science

Biographical:

Personal Data: Born in Jiangsu, China, January 21, 1957, the son of Shuming Luo and Linzhen Gao.

Education: Graduated from Zhanjiang 5th High School, Jiangsu, China, in June 1975; Received Bachelor of Science degree and Master of Science degree in Mechanical Engineering from Jiangsu University of Science and Technology in June 1982 and June 1989, respectively. Completed the requirements for the Master of Science degree with a major in computer Sceince at Oklahoma State University in May 1999.

Professional Experience: Mechanical Engineer, Zhanjiang Mechanical Manufacturing Plant, Jiangsu, China, July, 1982, to August, 1986; Assistant Professor/ Programmer/ Researcher, East China Shipbuilding University, Jiangsu, China, July, 1989, to January, 1995; Employed by Computer Information Services as a Computer Lab Consultant, Oklahoma State University, August, 1997, to present.

Professional Memberships: Member of the Association for Computing Machinery (ACM)