RED-BLACK TREE ANIMATION

By

YONGZHONG ZHANG

Bachelor of Science

Zhongshan University

Guangzhou, Guangdong

People's Republic of China

1990

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
December, 1998

RED-BLACK TREE ANIMATION

Thesis Approved:

Jacques E. LaFrance

Thesis Adviser

[illegible]

J Chandler

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGMENTS

My sincere appreciation goes to all the people who assisted me in this study and my stay here at Oklahoma State University.

I am especially grateful to my major advisor, Dr. Jacques E. LaFrance, for his constructive guidance, inspiration and friendship during the whole process. My sincere appreciation extends to Dr. John P. Chandler and George E. Hedrick for serving as my graduate committee members and providing invaluable guidance and assistance, and also for their friendship.

I am indebted to my family for their patience, encouragement and support. I am particularly grateful to my wife, Tong Xie, for her love, understanding, suggestions. Many thanks go to my parents and grandparents for their never-ending emotional support.

Finally, I would like to thank the Department of Computer Science for providing me the study opportunity.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

II. RELATED WORK 5

Data Structure Animation 5
Work at Oklahoma State University 8

III. RED-BLACK TREE ISSUES 9

Introduction 9
Basic Operations 11
Advanced Topics 13

IV. COMPUTER ANIMATION ISSUES.......... 17

Brief History 17
Computer-Assisted Animation 18

V. DESIGN AND TESTING 20

Visualization Organization 21
Testing 23

VI. SUMMARY AND FUTURE WORK 25

Summary 25
Future Work 25

BIBLIOGRAPHY 27
APPENDIXES 30

APPENDIX A—PROOF OF LEMMA 30
APPENDIX B—COMPUTER ANIMATION TERMS 31
APPENDIX C—MOVIE SCRIPT 32

LIST OF FIGURES

Figure Page

1. Rotations 12
2. Insertion Templates 14
3. Deletion Templates 15
4. Main Menu Interface 20
5. Animator Interface 22
6. An Indexed Complete Binary Search Tree 23

CHAPTER I

INTRODUCTION

Since the using of the film "*Sorting Out Sorting*" [Baecker 81] and BALSA [Brown 88], the study of computer animation as a teaching tool has been catching more and more attention within the past two decades [Ross 94][Stasko 96][Domingue 97], with the advance of computer techniques for creating and manipulating graphics displays.

Graphics have been used for centuries to communicate information effectively among people and aid in the comprehension of complex information. Because patterns and shapes are inherently less abstract than numbers and languages, pictures convey information more readily and permit easier retention than textual or verbal representation of the same information [Dudley 82]. With the advanced technology, the computer's information-handling capacities are now being implemented by the ability of hardware and software systems to condense, code, and display information in graphic forms that make large amounts of data comprehensible. Computer graphics can give people fast perception, vivid display of data relationships, and simplification of complex data.

Educators constantly are seeking new ways to improve instruction, to facilitate learning, and to hold the attention of students. The power of computers motivates the study and practice of using dynamic visualization as educational aids.

Though most of the text books do have pictures, the drawbacks of these figures are:

1. They give only static illustration
2. They usually do not have enough examples or even no examples
3. Possibly they don't show each step of a process
4. They don't have interactive capability.

Ross pointed out the problem [Ross 94]:

The problem is that no matter how clever or how animated our lecture is, when the student is faced with trying to review and assimilate the information later, the spirit of animation has long since left the body of knowledge. It would be very helpful to the student to be able to rerun the animation over and over, perhaps on other problem instances, while studying.

Since in computer science education, data structures and algorithms are a significant part, and a lot of research and animations focuses on this field. Algorithm animation has been proven to be a powerful tool in teaching data structures and algorithm analysis [Lawrence 94].

Algorithm animation uses computer-assisted visualization to make the comprehension of the functionality of algorithm easier. It actually serves two fundamental purposes as an instructional aid: it provides a concrete depiction of the abstractions and the operations which are inherent in an algorithm or program, and it portrays the dynamics of a time-evolving process [Stasko 96].

Algorithm animations don't have the drawback of static diagrams in textbooks. Stasko gave the additional advantages of algorithm animation:

Animations might encourage and make it relatively easy for the learner to make and test predictions of what is going to happen at each step of the algorithm. This prediction element could help the learner understand the algorithm better than a learner who passively takes in the information. A learner could certainly make predictions from a static textual/graphic presentation of an algorithm, but perhaps the advantage of an animation is that it will spontaneously encourage a learner to make the predictions without being prompted and will provide the learner with rapid feedback about the accuracy of his or her predictions. In addition, an animation might be more salient than a set of static images and thus, might help the student remember the algorithm better. An animation also might help learning because it displays the features that one should presumably attend to. That is, the items in the animation might, by their presence, provide information to the learner that might not be easily discernible otherwise. Finally, an animation might also encourage the learner to self-explain the behaviour of the algorithm. Self-explanation can increase the likelihood of a learner to integrate the new information with existing knowledge structures, thus making the learner mote likely to transfer the information to novel situations.

This thesis aims at the animated presentation of red-black trees. Such dynamic displays are helpful for students to comprehend the task of an algorithm to maintain the red-black tree structure.

Data structure algorithm animation can be either static or dynamic. Static animation means that the animation can only be seen for a fixed set of inputs. Dynamic animation means that the users can interact with the animation process of the computer, using their own inputs at run time. Dynamic animation enables users to observe and explore the dynamic behaviors of the data structure through interactive graphical displays. In this thesis, all animations are dynamic animations in order to achieve the greatest benefits for the learners.

The software used to develop this animation is Macromedia Director 6.0. It is one of the most powerful tools which can create a variety of multimedia productions. Some of the key features of Director 6.0 are [Director]:

1. wide-ranging cross-platform authoring and delivery
2. flexible authoring metaphors and tools
3. broad web authoring and delivery
4. approachable, professional user interface
5. powerful media, animation and integration functions

The Lingo scripting language of Director has object-oriented behavior. It can synchronize and integrate media elements or objects, such as interface, sound, and animation effectively.

This application is built on a PC, in the Windows 95 environment and can be delivered on the Internet, Windows NT, as well as both 68K and PowerPC of Mac OS system. It has the features of maintainability, accessibility, flexibility and interactivity.

CHAPTER II

RELATED WORK

This chapter gives an overview of previous and current work related to data structure algorithm animation. We focus on the educational animations. We will also mention some important systems, although their goals maybe not primarily for education, but they can be used for education.

Data Structures Algorithm Animation

Knowlton's film, "*L6: Bell Telephone Laboratories Low-Level Linked List Language*" [Knowlton 66], which showed the manipulation of lists at an assembly level. It is known the first to use dynamic display techniques as opposed to static techniques using a movie film medium, and the first to address the visualization of a data structure. Other well-known related movies were Hopgood's movie on hashing algorithms [Hopgood 74] and Booth's *PQ-Trees* [Booth 75].

Baecker's system developed in the mid-1970s was the first known system to aim at algorithm animation. It was not interactive and eventually resulted in the film "*Sorting Out Sorting*". This 25-minute, color, narrated educational film used animated graphics to explain how nine different sorting algorithms manipulate their data. It also illustrated the speed differences by showing each algorithm working on a data set of twenty-five hundred random elements [Price 93].

All of the work above had the limitation that they were films. Though their display style was dynamic, they did not allow viewers to interact with the model being displayed. Viewers must watch the films in the exact form in which it was produced, with the exact parameters, the exact data and even at the exact speed.

Computer static graphical displays of data structures emerged in mid-1970s. These systems had the advantage that data structures in any arbitrary program can be viewed without altering the program in any way. However, the disadvantage is that the generic representation does not necessarily convey how the data really was used. Moreover, they did not reveal how the algorithm was processing the data. They did not update the display corresponding to the change of data. Some of these systems are Incense, PV, GDBX, PROVIDE [Brown 88].

The 1980's were the beginning of modern computer animation with the introduction of the bit-mapped display and window interface technology. The first interactive and most important system of this era was BALSA, followed by BALSA-II, produced at Brown University. Many production systems using modern human-computer interface technology have been developed since then. The following gives brief descriptions of BALSA, Zeus, and TANGO. More systems information can be obtained from other literature [Price 93] and web sites [Zeus] [KMi].

BALSA was written in C and developed for both education and research. It was installed in a lab equipped with 55 Apollo workstations. All nodes were connected via a LAN. This system was used in undergraduate teaching [Brown, 85]. It allowed the instructor to give a running commentary on the prepared graphical animation running on each student's machine. Students could control these scripted animations (stop, start, speed control, replay, etc.). Several of the animations could also be played backwards. The entire set of scripts was integrated with the textbook *Algorithms* [Sedgewick 83]. BALSA is a pioneering work and it has become the benchmark against which all subsequent systems have been measured.

Zeus, developed by the Systems Research Center (SRC) of Digital Equipment Corporation(DEC), was mainly designed for research. This system was written for Modula-3 on DEC workstations [Brown 92]. Zeus is also noteworthy for its object-oriented design. In Zeus, an algorithm is annotated with procedure calls that identify the fundamental operations to be displayed. An annotation, called an interesting event, has parameters that identify program data. SRC has another educational system called CAT (Collaborative Active Textbooks), which is based on Zeus. Its Java version is JCAT [Brown 96] [JCAT].

TANGO was built up by Stasko. The current version is XTANGO [Stasko 92], which runs on most popular Unix workstations that can compile C and use the X-windows systems. It is a general purpose algorithm animation system that supports programmers developing color, real-time, continuous, 2 & 1/2-dimensional animation of their own algorithms and programs. The focus of the system is its ease of use. The basic process of developing an animation consists of implementing the algorithm in C (another language can be used, it must just produce a trace file which is read by a C program driver) and then deciding on the important events to be portrayed during the execution of the algorithm. These events then activate animation routines implemented in a separate file using the XTANGO animation package to create and manipulate objects (circles, squares, lines, and so on). Transitions on objects include movement, color change, resizing, and filling, as well as others.

Work At Oklahoma State University

The research and practice on the algorithm animation has been started for many years at computer science department of Oklahoma State University [Lee 88] [Arra 92] [Kummetha 93] [Shen 94] [Muktavaram 96] [Harvick 97] [Lin 97].

The work of this thesis is part of a larger project ANIMATED PRESENTATION OF DATA STRUCTURES. This system is developed as a teaching tool, which helps students comprehend the task of an algorithm to maintain a data structure. Interactive animation of these data structure algorithms will allow the students to explore different scenarios in the construction of data structures and better understand what their programs need to do.

Several animations of data structure algorithms have been finished. These include linked list, queue, stack [Xu 97], binary search tree [Shen 98], B-tree…

Animations are being done for other data structures, such as AVL trees.

All of these animations are developed to run on both Windows and the Mac OS operating systems, and they can be distributed over the web. Wide-ranging cross-platform authoring and delivery is one of the Macromedia Director's key features.

CHAPTER III

RED-BLACK TREE ISSUES

Introduction

The real world requires the manipulations in dynamic (finite) sets, which can change over time. Operations on a dynamic set can be grouped into two categories: querying operations, which simply return information about the set, and modifying operations, which change the set. Typical operations are *search (access), insert* and *delete*. The time taken to execute a set operation is usually measured in terms of the size of the set given as one of its arguments [Cormen 90].

In this paper, we call internal nodes of a tree, nodes, which are key-bearing nodes, and external nodes of a tree, leaves (NIL), which means that the value in that field is a pointer referring to no object at all.

A binary tree is a finite set of elements which is either empty or is partitioned into three disjoint subsets. The first subset contains a single element called the root of the tree. The other two subsets are themselves binary tree, called the left and the right subtrees of the original tree. A complete binary tree with n nodes has n+1 leaves. Information can be stored in the leaves and/or nodes of a tree. In some applications, we use only one possibility [Mehlhorn 84]. An important application of binary trees is their use in searching, because a special type of binary tree, the binary search tree, in which all keys are arranged according to a total order manner, is well-suited for storing ordered list of keys. Binary search tree supports dynamic set operations in _ (h) worst-case time, where h is the height of the tree [Weiss 93].

Balanced trees are a well-known solution to problem of maintaining a dynamic set so as to be able to perform the operations *access, insert and delete* quickly. These operations can be executed in _ (log n) time on a set of n elements if it is represented by a balanced tree [Knuth 73]. The idea of balancing a search tree is due to the invention of AVL trees, the oldest and most classical data structure for balanced trees. Balanced trees are not limited to binary trees, such as some famous trees such as splay trees, B-trees and their variants are also balanced trees [Sedgewick 83] [Cormen 90]. All known classes of balanced trees can be divided into two groups: height-balanced and weight-balanced trees. In height-balanced trees, one balances the height of the subtrees. In weight-balanced trees, one balances the number of nodes in the subtrees if a node is used as weight.

A red-black tree is a height-balanced binary tree, with one extra bit of storage per node: its color, which is either red or black. Red-black trees introduced by Bayer in 1972. He called them "Symmetric binary B-trees" [Bayer 72]. Guibas and Sedgewick studied their properties and related kinds of trees, and introduced the red/black color convention [Guibas 78]. The following is the definition of red-black tree according to Cormen [Cormen 90]. A binary search tree is a red-black tree if it satisfies the following red-black tree properties:

a. Every node is either red or black
b. Every leaf (NIL) is black

c. If a node is red, then both its children are black

d. Every simple path from a node to a descendant leaf contains the same number of black nodes

From the definition, we know immediately that a red-black tree is balanced by the constraint that a path from the root to a leaf is not more than twice as long as any other path. In other words, the longest path from an internal node to a leaf can not exceed twice that of a shortest path from that node to a leaf since both paths have the same number of black nodes.

We define black-height of a node x, bh(x), is the number of black nodes on any path from x to a leaf, not counting x. Leaves have black-height 0. The black-height of a tree is the black-height of its root.

As mentioned above, there are two ways to represent a list using red-black trees. The keys can be stored either in the internal nodes, so that the order in the list corresponds to symmetric order in the tree, or in the leaves, so that the order in the list corresponds to left-to-right order among leaves. Here we will use the former representation.

Basic Operations

One nice property of a red-black tree is that the *access* operation for standard binary search tree works without any modification. This section just focuses on the *insert* and *delete* operations.

The way to make red-black trees operations efficient in the worst case is to impose a balance condition that forces the height of an n-node tree to be _ (log n). This requires rebalancing the tree after each update operation.

Lemma A red-black tree with n nodes has height at most 2log(n+1).

The proof of this lemma is in appendix A. An immediate consequence of this lemma is that the dynamic set operations can be executed in _ (log n) time on red-black trees, because a binary search tree of height h executes its operations in _ (h), as mentioned before. Since operations *insert* and *delete* modify the tree, the result may violate the red-black tree properties, the rebalancing operations are needed if so. Updating is accomplished through node recoloring and tree rotation, which is a local operation in a search tree that preserves in-order traversal key ordering. Figure 1. gives the illustration of the rotations.

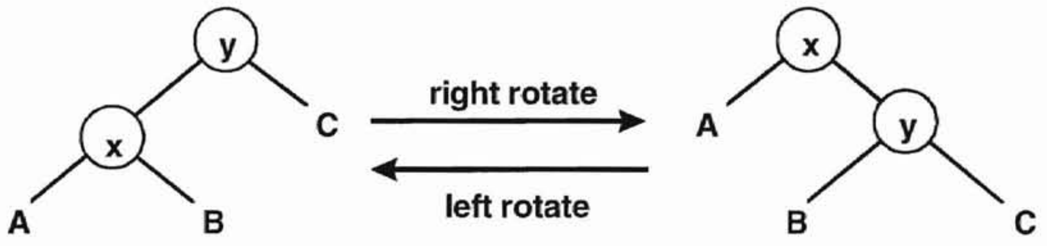


Figure 1. *Rotations. Note that in both trees, an in-order traversal yields: A x B y C*

Inserting In Red-Black Tree

i. Find a place for insertion as binary search tree, end at a dummy-leaf. Replace the leaf by new node with two new dummy-leaves.

ii. Color the new node red.

iii. Use the following templates (Figure 2.) for the insertion. In case 3, propagation may be needed, that is, to apply the recoloring transformation in case 3 until it no longer applies, and then if necessary, apply one of the transformation in case 2 and 4.

Deleting in Red-black Tree

Deleting an item is similar to the above but slightly more complicated.

i. Find the node to be deleted, delete it as in binary search tree.

ii. If node is red, the tree is still a red-black tree, otherwise the replacing node is short; i.e., paths down from it contain one fewer black node than paths from sibling. We use a minus symbol, _, to denote the shortness.

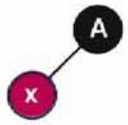
iii. Use the following templates (Figure 3.) for deletion. Let x be node to be deleted, y be the child of x.

Advanced Topics

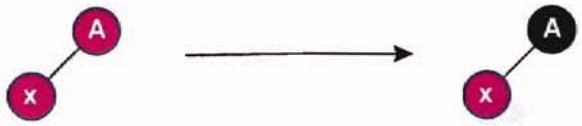
Red-black trees can also perform the join and split operations in logarithmic time. To join red-black trees, we need to store with each root its black-height, then while descending through the tree, we can compute the black-height of each node visited in constant time until we reach the appropriate node to join. Splitting a red-black tree at item x is accomplished by cutting off all subtrees along the path from root to x, and then we perform a sequence of joins to concatenate all the left ones to form T_L and all right ones to form T_R [Tarjan 83] [Booth 90].

In addition to having the O(log n) time bound for the operations above, the alternative top-down implementations have extremely similar performance [Guibas 78].

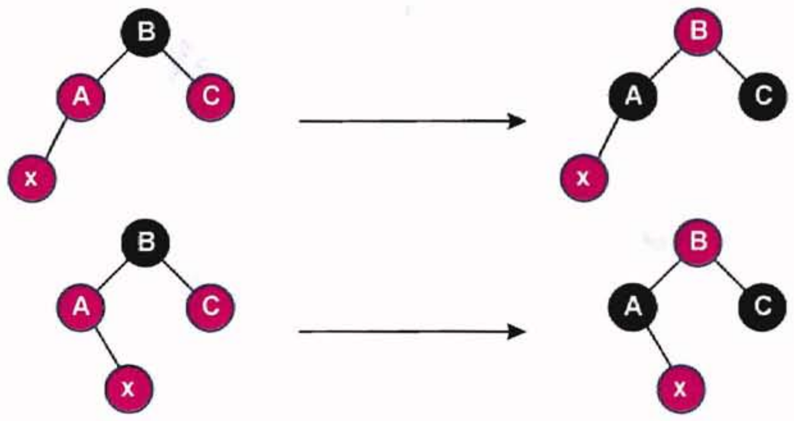
Case 1: p(x) is black, no update needed



Case 2: p(x) is red, g(x) doesn't exist



Case 3: p(x) is red, u(x) is red



Case 4: Terminal case. p(x) is red, u(x) is black

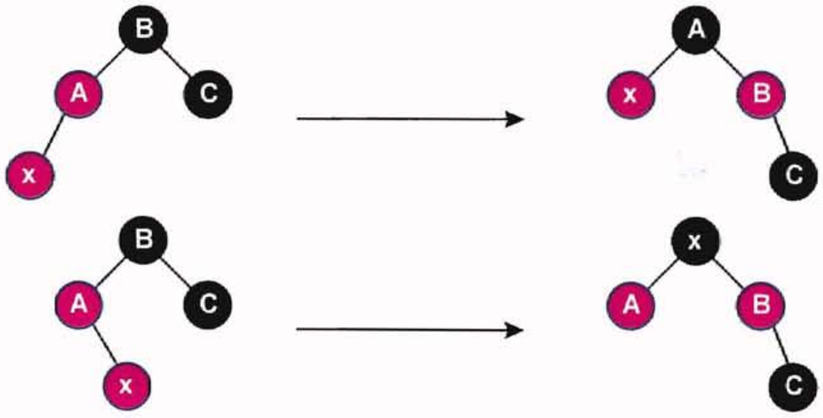
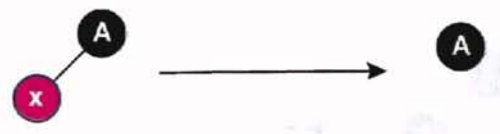


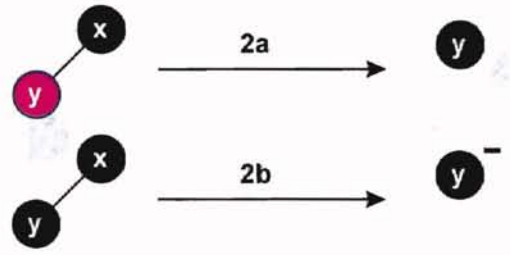
Figure 2. *Insertion templates. Case 3 may cause propagation,*

other cases are terminal cases.

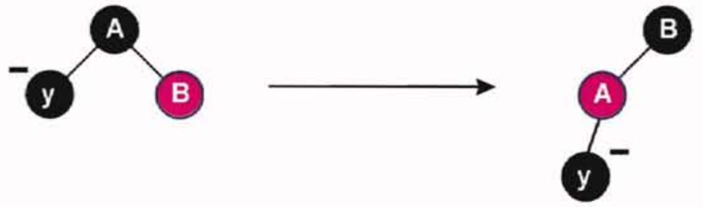
Case 1: red leaf, no update is needed



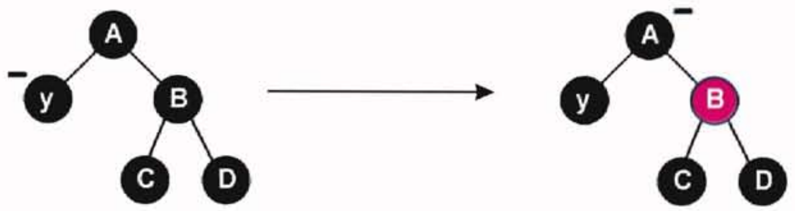
Case 2: black semi-leaf, if child is red, recolor it (2a), otherwise push the shortness up the tree (2b).



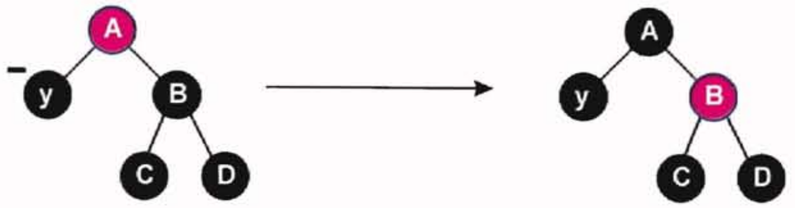
Case 3: s(y) is red



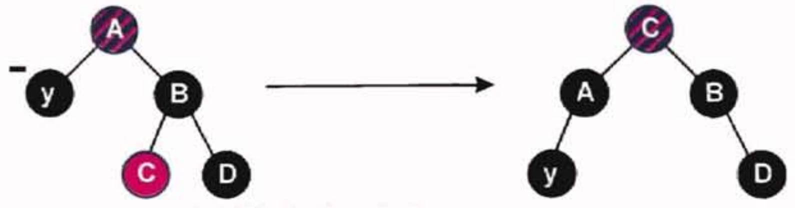
Case 4: s(y) is black, p(y) is black



Case 5: Terminal case. s(y) is black, p(y) is red



Case 6: Terminal case. s(y) is black, p(y) is either, s(y)'s left child is red, s(y)'s right child is black



Case 7: Terminal case. s(y) is black, p(y) is either, s(y)'s left child is either, s(y)'s right child is red

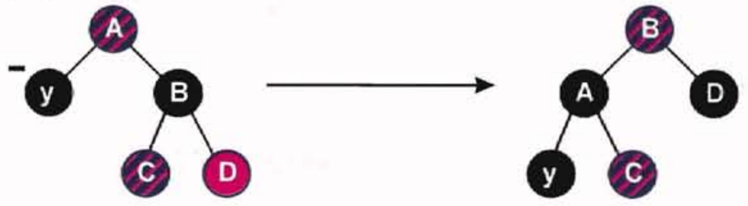


Figure 3. *Deletion templates. Case 2b, 3 and 4 may cause propagation, other cases are terminal cases.*

The conventions used in the figures above are the following:

p(x) ------ the parent of node x

u(x) ------ the uncle of node x

g(x) ------ the grandparent of node x

s(x) ------ the sibling of node x

🔴	———	Red node
●	———	Black node
▨	———	Either, i.e., node is either red or black

CHAPTER IV

COMPUTER ANIMATION ISSUES

Animation is based on the physiological fact that the image of an object perceived by the human eye persists in the brain for a brief period of time after the object no longer exists in the real world. This phenomena, called visual retention, is related to the chemistry of the retina and to the structure of cells and neurons in the eye. Smooth animation is achieved in cinematography and television by consecutively displaying images at a faster rate than the period of visual retention. The critical image update rate for smooth animation has been determined to be between 22 and 30 images per second [Sanchez 95].

Computer animation can be defined as the simulation of movement or of lifelike actions by the manipulation of digital objects. The notion of digital simulation of movement is the core of this definition [Sanchez 95].

Brief History

In 1824, Mark Roget published a paper, *"The Persistence of Vision with regard to Moving Objects*" [Morrison 94], which revealed the principle that the human eye retains an image for a fraction of a second longer than the image is actually present.

In 1831, Joseph A. Plateau invented the first animation machine called Phenakistoscope. This device consisted of a disk with a series of drawings and windows.

When the disk is rotated, viewers would see the drawings in rapid sequence [Thalmann 90].

The birth of computer graphics was a military system introduced in 1951. Computer animation was first developed in mid-1960s. The first computer-animated-film was *Hunger*, produced by Peter Folders in 1974 [Thalmann 90].

Computer-Assisted Animation

Computer animations can be divided into two classifications: computer-assisted animation and computer-generated animation.

Computer-assisted animation, also referred to as key-framed animation, consists of six possible functions, summarized by Thalmann: the input of drawings, the production of in-betweens, the specification of the motion of an object along a path, the coloring of drawings and backgrounds, the synchronization of the motion with sound, and the initiation of recording a sequence of film [Thalmann 90].

Computer-generated animation, also referred to as modeled animation, is used in the creation of three-dimensional images.

Sanchez described some techniques on computer-assisted animation.

Frame-By-Frame Animation

In frame-by-frame animation, the computer generates the required images, which are recorded or stored for playback at a later time. This playback can take place in the same machine that generated the image set or in another media. The generated images can either recorded in video tape or stored in computer disk. When the image set is complete, the animation can be viewed by playing back.

Interactive Animation

Interactive animation refers to computer objects that are moved at the user's desire. At present, the most common interactive devices are the keyboard and mouse. In general, the notion of interactive animation includes any technology in which the user exercises some levels of control over computer-animated action.

CHAPTER V

DESIGN AND TESTING

The goal of this thesis is red-black tree animation. Only the *insert* and *delete* operations are implemented. They are completely different from those of binary search trees (BST). Generally, the design and implementation of this animation will follow the style of previous work at Oklahoma State University. Eventually all work can be integrated into one whole data structure algorithm animation system.

The application consists of four Director movies which are presented as windows on a computer. Each one contains menu and/or button to enhance the interactivity of the graphical display. The figure below gives a display of the main menu of this application.

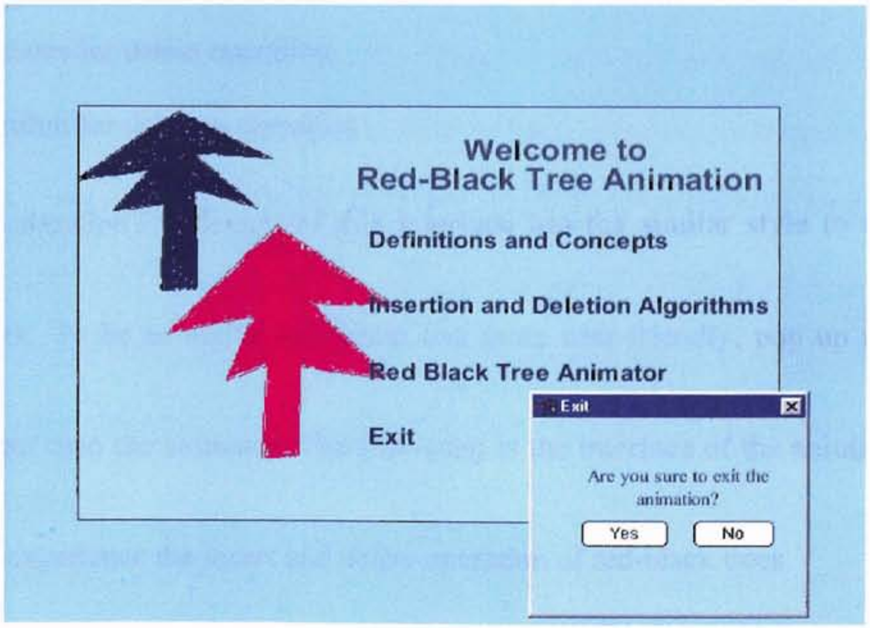


Figure 4. *Main menu interface.*

Visualization Organization

Specifically, the red-black tree animation consists of three parts: the presentation of the definition of a red-black tree, explanation of its operations, and the interactive animation of the algorithm.

Presentation of Definition

a. Different definitions will be given.

b. Some important concepts and features of red-black tree, such as black-height, shortness.

c. An example of a red-black tree.

Explanation of Operations

a. Templates for insert operation

b. Algorithm for insertion operation

c. Templates for delete operation

d. Algorithm for deletion operation

Algorithm AnimationThe design of this interface has the similar style to that of the previous work. To be an active animation and more user-friendly, pop up menus and buttons are put onto the animator. The following is the interface of the animator, where learners will experience the *insert* and *delete* operation of red-black trees

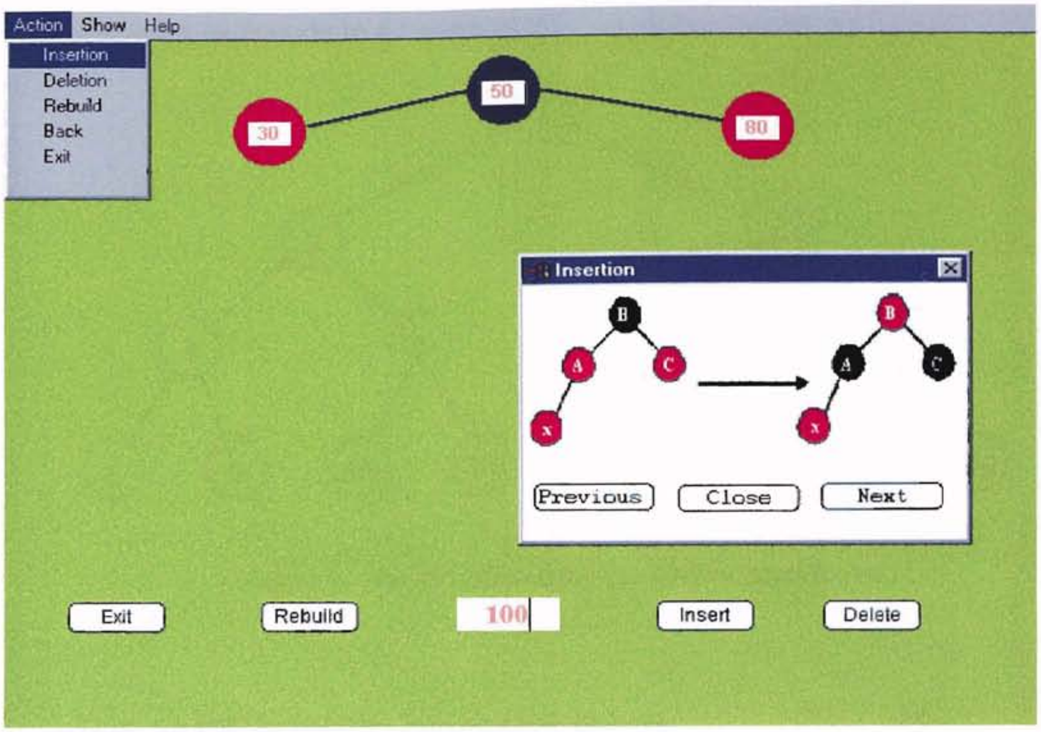


Figure 5. *The animator interface.*

Because rotations and propagation could happen, the implementation of the operations is more complicated than that of in BST. An example can explain more clearly how to implement the animation. When a new node is inserted, it is first compared with the root, and go along the path as in BST. If a rotation happens, a node X is moved to the proper place, then the line connecting X and its parent disappears, the new parent of X then is moved to the right position if X has that new parent. The movement of the key nodes (x, y in figure) in a rotation are displayed. Finally, a new line connecting X with its parent is drawn. The animations of other nodes in the rotation run under the same method.

The visualization is basing on the inherent fact in the binary search trees: the one-to-one correspondence between the node in a BST and an entry in an array.

We index each node in a complete BST, as shown below:

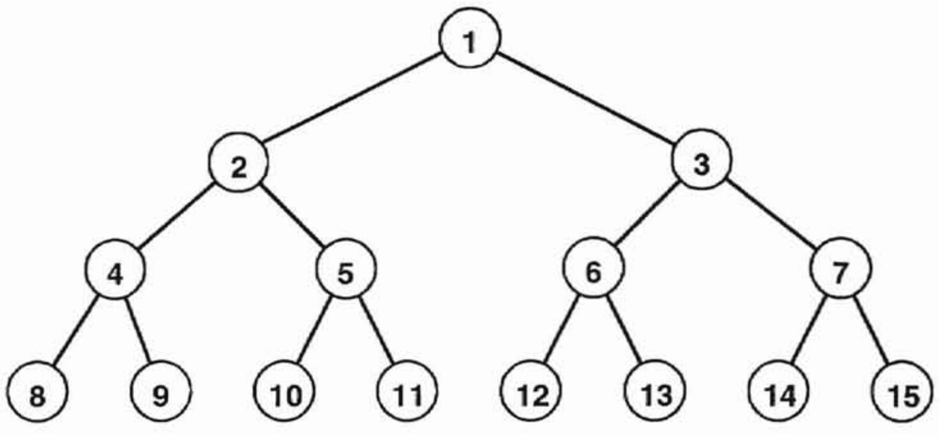


Figure 6. An indexed complete binary search tree

Then we put these nodes into an array orderly:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ...

We can find that the relationship between the tree and the array is:

1. If node in position n has children, its left child and right child are at the entry 2n, 2n+1 of the array, respectively.
2. The parent of any node in position n (n>1) is in the position _n/2_.

Testing

The purpose of testing is to ensure that the software works as intended. The test process includes the user interface testing, data handling testing and object movement testing (animation testing).

1. The objective of the testing of all the user interfaces, including the pop up windows, is to ensure the ease of use.

2. The objective of the testing of data handling, including the error handling, is to ensure the functionality of the actions.
3. The objective of the testing of image movement is to ensure that during the operation, the learners can view clearly the path along with a node is moving.

CHAPTER VI

SUMMARY AND FUTURE WORK

Summary

Advanced technology leads to inscreasing capability of computers. It gives us a new way to develop some new teaching tools that can help students have better understanding of the information they get from the instructor's lectures and from their textbooks.

This thesis project is to build an interactive red-black tree animation. This application provides a user-friendly interface. Users can interact with it to get the presentation of red-black tree's definition, key concepts and its operations and their algorithms. Multiple windows, menus, buttons, and event-drive design allow the learners to rerun the red-black tree animation over and over. The learners are able to try out the software on their own data, at their own pace, so that they can get better comprehension of what a red-black tree is and how it performs insertion and deletion operations.

This application is developed on Windows 95 environment using Macromedia Director 6.0. It can be delivered on the Internet, Windows NT, as well as both the Motorola 68K and the PowerPC of Mac OS system. It has the features of reusability, maintainability, accessibility and interactivity.

Future Work

The work here is a part of a larger project, which is planned to be a teaching tool to help computer science students to learn more effectively while studying data structures and algorithms. Sound can be added to the application to achieve a better multimedia effect and to enhance the presentation quality, and help users understand what is happening easier while using it, thus help them have a better understanding of the abstract concepts and algorithms.

More of the animations of other data structure algorithms can be constructed, and all work could be woven together into an integrated system suitable for use in laboratory exercises or distance learning in the future.

BIBLIOGRAPHY

Arra, Shravan K., "Object-Oriented Data Structure Animation", *M.S. Thesis*, Computer Science Department, Oklahoma State University, 1992.

Baecker, Ronald, *Sorting Out Sorting*, 16mm color sound film, 25 minutes, Computer Science Department, University of Toronto, 1981.

Bayer, R., "Symmetric binary B-trees: Data Structure and Maintenance algorithms", *Acta Informatica* 1, pp. 290-306, 1972.

Booth, Heather D., "Some Fast Algorithms on Graphs and Trees", *Ph.D. dissertation*, Princeton University, 1990.

Booth, Kellog S., *PQ Trees,* 16mm color silent film, 12 minutes, 1975.

Brown, Marc H., "Zeus: A System for Algorithm Animation and Multi-View Editing", *Systems Research Center Research Report 75*, Digital Equipment Corporation, 1992.

Brown, Marc H., *Algorithm Animation*, The MIT Press, 1988.

Brown, Marc H. and Najork, Marc A., "Collaborative Active Textbooks: A Web-Based Algorithm Animation System for an Electronic Classroom", *Systems Research Center Research Report 142*, Digital Equipment Corporation, 1996.

Brown, M. H., and Sedgewick R., "Techniques For Algorithm Animation", *IEEE Software*, pp. 28-39, 1985.

Cormen, R. H., Rivest, R.L., and Leiserson, C. E., *Introduction to Algorithms*, The MIT Press under a joint production-distribution agreement with the McGraw-Hill Books, 1990.

Director, Macromedia Inc., WWW page, 1998.
URL: http://www.macromedia.com/software/director/productinfo/.

Domingue, John and Mulholland, Paul, "Teaching Programming at a Distance: The Internet Software Visualization Laboratory", *Journal of Interactive Media in Education*, 1997(1), pp. 1-24.

Dudley, Timothy K., "Computer and Graphics: A Technology Comes of Age", in *Interactive Computer Systems*, edited by William C. House, Petrocelli Books, Inc., 1984.

Guibas, Leo J. and Sedgewick R., "A Dichromatic Framework for Balanced Trees", *Proceedings of the 19th Annual Symposium on Foundations of Computer Science, IEEE Computer Society*, pp. 8-12, 1978.

Harvick, Lee Hou., "Rule Based Data Structure Animation", *M.S. Thesis*, Computer Science Department, Oklahoma State University, 1997.

Hoopgood, F. Robert A., "Computer Animation Used as a Tool in Teaching Computer Science", *Proc. 1974 IFIP Congress,* 1974.

JCAT, "*Java-Based Collaborative Active Textbooks*", Systems Research Center, Digital Equipment Corporation WWW page, 1998.
URL: http://www.research.digital.com/SRC/JCAT/.

KMi, "*Software Visualization Research in KMi*", Knowledge Media Institution WWW page.
URL: http://kmi.open.ac.uk/sv/.

Knowlton, K. C., *L6: Bell Telephone Laboratories Low-Level Linked List Language*, 16mm black and with sound film, 16 minutes, Murray Hill, NJ: Technical Information Libraries, Bell Laboratories, Inc., 1966.

Knuth, Donald E., *The Art of Computer Programming, Volume 3/Sorting And Searching*, Addison-Wesley Publishing Co., 1973.

Kummetha, V. C. S. Reddy, "A Level Linked R* Tree Structure With An Application Using X-Window Graphical Interface", *M.S. Thesis*, Computer Science Department, Oklahoma State University, 1993.

Lawrence, A., Badre, A., and Stasko, J., "Empirically Evaluating The Use Of Animation To Teach Algorithms", *IEEE Symp. On Visual Languages*, pp. 48-54, 1994.

Lee, Wilson, "An Implementation Of A Data Structures Display System", *M.S. Thesis*, Computer Science Department, Oklahoma State University, 1988.

Lin, Betty, "An Object-Oriented Graphic User Interface For Visualization Of B-Trees' Animation", *M.S. Thesis*, Computer Science Department, Oklahoma State University, 1997.

Mehlhorn, Kurt, *Sorting and Searching, Volume 1 of Data Structures and Algorithms*, Springer-Verlag, 1984.

Morrison, Mike, *Become A Computer Animator*, Sams Pub., 1994

Muktavaram, Vikas, "Visualization Of Sorting Algorithms", *M.S. Thesis*, Computer Science Department, Oklahoma State University, 1996.

Price, B. A., Baecker, R. M., and Small, I. S., "A Principled Taxonomy of Software Visualization", *Journal of Visual Languages and Computing*, **4**(3) pp.211-266, 1993.

Ross, Rockford J., "Animation in Computer Science Education", *SIGACT News*, Volume 25, No.2, pp.40-49, 1994.

Sanchez, Julio and Canton, Maria P., *Computer Animation Programming Methods and Techniques*, McGraw-Hill Inc., 1995.

Sedgewick, Robert, *Algorithms*, Addison-Wesley Publishing Company, Inc. 1983.

Shen, Bin, "An Instructional Module For Teaching About Binary Search Trees", *M.S. Thesis*, Computer Science Department, Oklahoma State University, 1998.

Shen, Hung-Che, "A Visual Aid For The Learning Of Tree-Based Data Structures", *M.S. Thesis*, Computer Science Department, Oklahoma State University, 1994.

Stasko, John T., Byrne, Michael D. and Catrambone Richard, "Do Algorithm Animations Aid Learning?", *Technical Report GIT-GVU-96-18*, Georgia Institute of Technology, 1996.

Stasko, John, "Animating Algorithms with XTANGO", *SIGACT News*, 23(2), 67-71, 1992.

Tarjan, R. E., "Data Structure And Network Algorithm", *SIAM*, Philadelphia, 1983

Thalmann, Nadia M. and Thalmann, Daniel, *Computer Animation*, Second revised edition, Springer-Verlag, 1990.

Weiss, Mark A., *Data Structures and Algorithm Analysis in C*, Benjamin/Cummings Publishing Company, Inc., 1993.

Xu, Cong, "Multimedia Visualization Of Abstract Data Type", *M.S. Thesis*, Computer Science Department, Oklahoma State University, 1997.

Zeus, "*Algorithm Animation at SRC* ", Systems Research Center, Digital Equipment Corporation WWW page, 1998
URL: http://www.research.digital.com/SRC/zeus/home.html.

APPENDIXES

APPENDIX A

Lemma

A red-black tree with n internal nodes has height at most 2lg(n+1)

proof

Show that subtree starting at x contains at least 2bh(x)-1 internal nodes. By induction on height of x:

if x is a leaf then bh(x) = 0, 2bh(x)-1

Assume x has height h, x's children have height h -1

x's children black-height is either bh(x) or bh(x) -1

By induction x's children subtree has 2bh(x)-1-1 internal nodes

So subtree starting at x contains

2bh(x)-1-1 + 2bh(x)-1-1 + 1 = 2bh(x)-1 internal nodes

let h = height of the tree rooted at x

bh(x) >= h/2

So n >= 2h/2-1 <=> n + 1 >= 2h/2 <=> lg(n+1) >= h/2

h <= 2lg(n+1)

APPENDIX B

Computer-Assisted Animation ---Animation produced with the use of a software program and a computer to draw or model each frame, produce the in-betweens and color them, as opposed to animation produced with cells, clay, or puppets etc.

Computer-Generated Animation --- Animation produced using a computer and various software programs to create images within the computer and manipulate them. These images exist as digital information until they are output to the CRT monitor, film/video recorder or printer.

Keyframe --- The main positions of a model along its movement path.

In-between(s) --- The frame(s) that lies in sequence between two keyframes.

APPENDIX C

The following is the movie script, the main part of scripts in this application:

```
-- ***********************************************************************
-- *                                                                      *
-- *                          Movie Script                                *
-- *                                                                      *
-- ***********************************************************************
on startMovie
  global gOperationList, gNodeList

  set gOperationList = []
  set gNodeList = []

  invisible
  installmenu 5
end

-- ***********************************************************************
on Intemp
  set the modal of window "templates" to TRUE
  set the windowtype of window "templates" to 4
  tell window "templates" to go to frame "Intemp"
  tell window "templates" to set the title of window "templates" to "Insertion"
  set the rect of window "templates" to rect(5,50,280,220)
  open window "templates"
  --forget window "templates"
end Intemp

-- ***********************************************************************
on Deltemp
  set the modal of window "templates" to TRUE
  set the windowtype of window "templates" to 4
  tell window "templates" to go to frame "Deltemp"
  tell window "templates" to set the title of window "templates" to "Deletion"
  set the rect of window "templates" to rect(5,50,280,220)
  open window "templates"
  --forget window "templates"
end Deltemp

-- ***********************************************************************
on ShowHistory
  -- alert "Doesn't work."
  global gOperationList

  set totalNum to count(gOperationList)
  if totalNum=0 then
    alert "No operation yet."
    exit
  end if
  set operations=getAt(gOperationList, 1)
  repeat with x=2 to totalNum
    put ",  " & getAt(gOperationList,x) after operations
  end repeat

  go to frame "History"
  puppetSprite 46, TRUE
  puppetSprite 47, TRUE
  set the visible of sprite 46 to TRUE
  set the visible of sprite 47 to TRUE
  set the text of member "showOps" to operations
  --  set the text of member "Another" to operations
  updateStage
end ShowHistory

-- *******************************************************************
on rebuild
  global gOperationList, gNodeList
  set gOperationList = []
  set gNodeList = []
  invisible
  cleardata
end rebuild

-- *******************************************************************
on backMain
  play done
end

-- *******************************************************************
on finish
  set the modal of window "Wins" to TRUE
  set the windowtype of window "Wins" to 4
  set WinTop to ((the stageTop + the stageBottom)/2-75)
  set  WinBottom to ((the stageTop + the stageBottom)/2+75)
  set WinLeft to ((the stageLeft + the stageRight)/2-100)
  set WinRight to ((the stageLeft + the stageRight)/2+100)
  set the rect of window "Wins" = rect(WinLeft, WinTop, WinRight, WinBottom)
  tell window "Wins" to go to frame "exit"
  open window "Wins"
  --forget window "Wins"
end finish

-- *******************************************************************
on stopmovie
  invisible
  cleardata
  set the text of member "Key1" to ""
  repeat with x=17 to 47
    set the text of member x to ""
  end repeat
end
```

```
-- ************************************************************************
on Instruct
  set the modal of window "Wins" to FALSE
  set the windowtype of window "Wins" to 4
  set the rect of window "Wins" to rect(5,50,413,298)

  tell window "Wins" to go to frame "Help"
  open window "Wins"
end

on AboutThis
  set the modal of window "Wins" to TRUE
  set the windowtype of window "Wins" to 4
  set the rect of window "Wins" to rect(5,50,286,228)

  tell window "Wins" to go to frame "About"
  open window "Wins"
end

-- ************************************************************************
-- *                          Insertion Routine                           *
-- ************************************************************************
on insert
  global gNodeList, gOperationList

  if the text of member "FieldInput" = "" then
    alert "Please enter an integer in the box."
    exit
  end if

  set NewKey to value(the text of member "FieldInput")
  puppetSprite 91, TRUE
  puppetSprite 92, TRUE

  if count(gNodeList)=0 then
    set NewNode=new(script "MakeNewNode", NewKey, 1)

    display NewNode
    add gNodeList, NewKey
    add gOperationList, "Insert  " & string(NewKey)

    ColorRootBlack
  else
    if getOne(gNodeList, NewKey)=0 then
      set CurKey=value(the text of member "Key1")
      set CurNum=1

      set the text of member "MovingKey" to string(NewKey)
      repeat while voidP(CurKey)=FALSE
        set CurSprite=3*CurNum-1

        set the locH of sprite 91 to the right of sprite CurSprite-1
        set the locV of sprite 91 to the locV of sprite CurSprite
        set the locH of sprite 92 to the locH of sprite 91+17
      set the locV of sprite 92 to the locV of sprite 91

      if CurKey > NewKey then
        set the text of member "Compare" to ">"
       set CurNum=CurNum*2
      else
        set the text of member "Compare" to "<"
       set CurNum=CurNum*2+1
      end if

      set the visible of sprite 91 to TRUE
      set the visible of sprite 92 to TRUE
      updateStage

      longWait
      if CurNum=16 or CurNum>=31 then
       decline
        exit
      end if

      set string=the text of member (the memberNum of sprite (3*CurNum-1))
      if the visible of sprite (3*CurNum-1)=TRUE then
       set CurKey=value(string)
      else
        exit repeat
      end if
     end repeat

     set the visible of sprite 91 to FALSE
     set the visible of sprite 92 to FALSE

     set NewNode=new(script "MakeNewNode", NewKey, CurNum)

    display NewNode
     if CurNum>=4 and (getColor(getParentNum(CurNum))=34 or
getColor(getParentNum(CurNum))=35) then
      adjust(CurNum)
       if the foreColor of sprite 1 <> 255 then
        ColorRootBlack
      end if
    end if

    add gNodeList, NewKey
     add gOperationList, "Insert " & string(NewKey)

   else
     alert "This node already in the tree."
   end if
  end if

  cleardata
end insert

-- **********************************************************************
-- *                    This handler will be called when red rule is not satisfied         *
-- ************************************************************************
on adjust thisNum
  repeat while (thisNum<>1)
    if (getColor(getParentNum(thisNum))=35) then
      set parentNum=getParentNum(thisNum)
      set grandparentNum=getGrandparentNum(thisNum)

      if (parentNum mod 2)=0 then
        set uncle=parentNum+1
        if getColor(uncle)=35 or getColor(uncle)=34 then
          colorBlack(parentNum)
          colorBlack(uncle)
          colorRed(grandparentNum)
          set thisNum=grandparentNum
        else
          if (thisNum mod 2)<>0 then
            leftRotate(parentNum)
          end if

          colorBlack(parentNum)
          colorRed(grandparentNum)

          rightRotate(grandparentNum)
          exit repeat
        end if

    else
      set uncle=parentNum-1
      if getColor(uncle)=35 or getColor(uncle)=34 then
        colorBlack(parentNum)
        colorBlack(uncle)
        colorRed(grandparentNum)
        set thisNum=grandparentNum
      else
        if (thisNum mod 2)= 0 then
          rightRotate(parentNum)
        end if
        colorBlack(parentNum)
        colorRed(grandparentNum)

        leftRotate(grandparentNum)
        exit repeat
      end if

    end if
  else
    exit repeat
  end if
  end repeat
end

-- ************************************************************************
-- *                              Right Rotation Routine                           *
-- *********************************************************************
on rightRotate nodeNum
  set KeySprite=3*NodeNum-1
  set rightNum=2*nodeNum+1
  set rightKeySprite=3*rightNum-1

  set height=the locV of sprite rightKeySprite-the locV of sprite KeySprite
  set width=the locH of sprite rightKeySprite-the locH of sprite KeySprite
  set times=integer(height/3)-1

  puppetSprite KeySprite-1, TRUE
  puppetSprite KeySprite, TRUE
  puppetSprite 93, TRUE
  set the text of member "movingNode"=the text of member (the memberNum of sprite
KeySprite)
  set the foreColor of member "movingNode"=the foreColor of sprite (KeySprite-1)
  set the loc of sprite 93 to the loc of sprite KeySprite

  set the visible of sprite KeySprite-1 = FALSE
  set the visible of sprite KeySprite = FALSE
  set the visible of sprite 93 = TRUE
  updateStage

  repeat with x=1 to times
    set the locV of sprite 93 = the locV of sprite 93 + 3
    set the locH of sprite 93=the locH of sprite 93+(integer(3*width/height))
    wait
    updateStage
  end repeat
  set the loc of sprite 93 to the loc of sprite rightKeySprite
  --set the visible of sprite rightKeySprite = FALSE
  set the visible of sprite rightKeySprite-1 = FALSE
  updateStage

  if (the text of member (the memberNum of sprite rightKeySprite))<>"" and the visible of
sprite rightKeySprite=TRUE then
    if rightNum=3 or rightNum=5 or rightNum=7 or rightNum=11 or rightNum=15 then
      RRmoveSubTreeDown1 rightNum
    else
      RRmoveSubTreeDown2 rightNum
    end if
  end if

  puppetSprite rightKeySprite-1, TRUE
  puppetSprite rightKeySprite, TRUE
  set the foreColor of sprite (rightKeySprite-1)=the foreColor of member "movingNode"
  set the text of member (the memberNum of sprite rightKeySprite)=the text of member
"movingNode"

  set the visible of sprite rightKeySprite = TRUE
  set the visible of sprite rightKeySprite-1 = TRUE
  set the visible of sprite rightKeySprite-2 = TRUE
  set the visible of sprite 93 = FALSE
  updateStage
  RRmoveSubTreeUp(2*nodeNum)
  longwait
end

-- ************************************************************************
-- *                 Handlers to move sub-tree down whiling rotating        *
-- ************************************************************************
on RRmoveSubTreeDown1 subRootNum
  set leftChildNum=2*subRootNum
  set rightChildNum=2*subRootNum+1
  if subRootNum<16 then
    if the visible of sprite (3*leftChildNum-1)=TRUE and (the text of member (the
memberNum of sprite (3*leftChildNum-1)))<>"" then
     RRmoveSubTreeDown1(leftChildNum)
    end if

    if the visible of sprite (3*rightChildNum-1)=TRUE and (the text of member (the
memberNum of sprite (3*rightChildNum-1)))<>"" then
     RRmoveSubTreeDown1(rightChildNum)
    end if
  end if

  if subRootNum=3 then
    set newNum=7
  else if subRootNum=5 then
    set newNum=11
  else if subRootNum=6 or subRootNum=7 then
    set newNum=subRootNum+8
  else if subRootNum=10 or subRootNum=11 then
    set newNum=subRootNum+12
  else if subRootNum>=12 and subRootNum<15 then
    set newNum=subRootNum+16
  else
    decline
    exit
  end if
  nodeMoving(newNum, subRootNum)
end

-- ************************************************************************
on RRmoveSubTreeDown2 subRootNum
  set leftChildNum=2*subRootNum
  set rightChildNum=2*subRootNum+1
  if subRootNum<16 then
    if the visible of sprite (3*leftChildNum-1)=TRUE and (the text of member (the
memberNum of sprite (3*leftChildNum-1)))<>"" then
     RRmoveSubTreeDown2(leftChildNum)
    end if

    if the visible of sprite (3*rightChildNum-1)=TRUE and (the text of member (the
memberNum of sprite (3*rightChildNum-1)))<>"" then
     RRmoveSubTreeDown2(rightChildNum)
    end if
  end if

  if subRootNum=9 then
   set newNum=subRootNum+10
  else if subRootNum=13 then
   set newNum=subRootNum+14
  else
   decline
   exit
  end if

  nodeMoving(newNum, subRootNum)
end

-- ********************************************************************
-- *              Handlers for move the left sub-tree up in Right Rotation Routin        *
-- ********************************************************************
on RRmoveSubTreeUp nodeNum
  set KeySprite=3*nodeNum-1
  set parentKeySprite=(nodeNum/2)*3-1

  set height=the locV of sprite KeySprite-the locV of sprite parentKeySprite
  set width=the locH of sprite parentKeySprite-the locH of sprite KeySprite
  set times=integer(height/3)-1

  puppetSprite KeySprite-1, TRUE
  puppetSprite KeySprite, TRUE
  puppetSprite 93, TRUE
  set the text of member "movingNode"=the text of member (the memberNum of sprite
KeySprite)
  set the foreColor of member "movingNode"=the foreColor of sprite (KeySprite-1)
  set the loc of sprite 93 to the loc of sprite KeySprite
  updateStage

  set the visible of sprite KeySprite-2 = FALSE
  set the visible of sprite KeySprite-1 = FALSE
  set the visible of sprite KeySprite = FALSE
  set the visible of sprite 93 = TRUE

  repeat with x=1 to times
   set the locV of sprite 93 = the locV of sprite 93 - 3
   set the locH of sprite 93=the locH of sprite 93+(integer(3*width/height))
   wait
   updateStage
  end repeat
  set the loc of sprite 93 to the loc of sprite parentKeySprite
  updateStage

  puppetSprite parentKeySprite-1, TRUE
  puppetSprite parentKeySprite, TRUE
  set the foreColor of sprite (parentKeySprite-1)=the foreColor of member "movingNode"
  set the text of member (the memberNum of sprite parentKeySprite)=the text of member
"movingNode"
```

```
  set the visible of sprite parentKeySprite = TRUE
  set the visible of sprite parentKeySprite-1 = TRUE
  set the visible of sprite 93 = FALSE
  updateStage

  set leftChild=2*nodeNum
  set rightChild=2*nodeNum+1

  if (3*rightChild-1)<90 then
    if the visible of sprite (3*rightChild-1)=TRUE then
      if rightChild=5 then
        RRtransferSubTree1 rightChild
      else if rightChild=9 or rightChild=13 then
        RRtransferSubTree2 rightChild
      else
        set newNode=rightChild+1
        nodeMoving(newNode, rightChild)
      end if
    end if
  end if

  if (3*leftChild-1)<90 then
    if the visible of sprite (3*leftChild-1)=TRUE then
      if leftChild=4 then
        RRupingSubTree1 leftChild
      else if leftChild=8 or leftChild=12 then
        RRupingSubTree2 leftChild
      else
        nodeMoving(leftChild/2, leftChild)
      end if
    end if
  end if
end

-- ********************************************************************
on RRupingSubTree1 subRootNum
  if subRootNum=4 then
    set newNode=2
  else if subRootNum=8 or subRootNum=9 then
    set newNode=subRootNum-4
  else
    set newNode=subRootNum-8
  end if
  nodeMoving(newNode, subRootNum)

  set leftChild=2*subRootNum
  set rightChild=leftChild+1
  if leftChild<31 then
    if the visible of sprite (3*leftChild-1)=TRUE then
      RRupingSubTree1 leftChild
    end if
  end if
  if rightChild<31 then
    if the visible of sprite (3*rightChild-1)=TRUE then
    RRupingSubTree1 rightChild
   end if
  end if
end

-- ********************************************************************
on RRupingSubTree2 subRootNum
  if subRootNum=8 then
   set newNode=4
  else if subRootNum=12 then
   set newNode=6
  else if subRootNum=24 or subRootNum=25 then
   set newNode=subRootNum-12
  else
   set newNode=subRootNum-8
  end if
  nodeMoving(newNode, subRootNum)

  set leftChild=2*subRootNum
  set rightChild=leftChild+1
  if leftChild<31 then
    if the visible of sprite (3*leftChild-1)=TRUE then
     RRupingSubTree2 leftChild
   end if
  end if
  if rightChild<31 then
    if the visible of sprite (3*rightChild-1)=TRUE then
     RRupingSubTree2 rightChild
   end if
  end if
end

-- ********************************************************************
on RRtransferSubTree1 subRootNum
  set leftChild=2*subRootNum
  set rightChild=leftChild+1

  if leftChild<31 then
    if the visible of sprite (3*leftChild-1)=TRUE then
     RRtransferSubTree1 leftChild
   end if
  end if
  if rightChild<31 then
    if the visible of sprite (3*rightChild-1)=TRUE then
     RRtransferSubTree1 rightChild
   end if
  end if

  if subRootNum=5 then
   set newNode=6
  else if subRootNum=10 or subRootNum=11 then
   set newNode=subRootNum+2
  else
   set newNode=subRootNum+4
  end if

  nodeMoving(newNode, subRootNum)
end

on RRtransferSubTree2 subRootNum
  set leftChild=2*subRootNum
  set rightChild=leftChild+1

  if leftChild<31 then
    if the visible of sprite (3*leftChild-1)=TRUE then
      RRtransferSubTree2 leftChild
    end if
  end if
  if rightChild<31 then
    if the visible of sprite (3*rightChild-1)=TRUE then
      RRtransferSubTree2 rightChild
    end if
  end if

  if subRootNum=9 or subRootNum=13 then
    set newNode=subRootNum+1
  else
    set newNode=subRootNum+2
  end if

  nodeMoving(newNode, subRootNum)
end

-- *****************************************************************
-- *                     Left Rotation Routine                     *
-- *****************************************************************
on leftRotate nodeNum
  set nodeSprite=3*nodeNum-1
  set leftNum=2*nodeNum
  set leftSprite=3*leftNum-1

  set height=the locV of sprite leftSprite-the locV of sprite nodeSprite
  set width=the locH of sprite nodeSprite-the locH of sprite leftSprite
  set times=integer(height/3)-1

  puppetSprite nodeSprite, TRUE
  puppetSprite leftSprite, TRUE
  puppetSprite 93, TRUE
  set the text of member "movingNode"=the text of member (the memberNum of sprite
nodeSprite)
  set the foreColor of member "movingNode"=the foreColor of sprite (nodeSprite-1)
  set the loc of sprite 93 to the loc of sprite nodeSprite
  updateStage

  set the visible of sprite nodeSprite-1 = FALSE
  set the visible of sprite nodeSprite = FALSE
  set the visible of sprite 93 = TRUE
  repeat with x=1 to times
    set the locV of sprite 93 = the locV of sprite 93 + 3
    set the locH of sprite 93=the locH of sprite 93-(integer(3*width/height))
    wait
    updateStage
  end repeat

  set the loc of sprite 93 to the loc of sprite leftSprite
  --  set the visible of sprite leftSprite = FALSE
  set the visible of sprite leftSprite-1 = FALSE
  updateStage

  if the visible of sprite leftSprite=TRUE and (the text of member (the memberNum of sprite
leftSprite))<>"" then
    if leftNum=2 or leftNum=4 or leftNum=6 or leftNum=8 or leftNum=12 then
      LRmoveSubTreeDown1 leftNum
    else
      LRmoveSubTreeDown2 leftNum
    end if
  end if

  puppetSprite leftSprite-1, TRUE
  puppetSprite leftSprite, TRUE
  set the foreColor of sprite (leftSprite-1)=the foreColor of member "movingNode"
  set the text of member (the memberNum of sprite leftSprite)=the text of member
"movingNode"

  set the visible of sprite leftSprite = TRUE
  set the visible of sprite leftSprite-1 = TRUE
  set the visible of sprite leftSprite-2 = TRUE
  set the visible of sprite 93 = FALSE
  updateStage

  LRmoveSubTreeUp(2*nodeNum+1)
  longwait
end

--  *******************************************************************
-- *                 Handlers for move the left sub-tree up in Left Rotation Routin          *
--  *******************************************************************
on LRmoveSubTreeUp nodeNum
  set KeySprite=3*nodeNum-1
  set parentKeySprite=(nodeNum/2)*3-1

  set height=the locV of sprite KeySprite-the locV of sprite parentKeySprite
  set width=the locH of sprite KeySprite-the locH of sprite parentKeySprite
  set times=integer(height/3)-1

  puppetSprite KeySprite, TRUE
  puppetSprite parentKeySprite, TRUE
  puppetSprite 93, TRUE
  set the text of member "movingNode"=the text of member (the memberNum of sprite
KeySprite)
  set the foreColor of member "movingNode"=the foreColor of sprite (KeySprite-1)
```

```
 set the loc of sprite 93 to the loc of sprite KeySprite
 updateStage

 set the visible of sprite KeySprite-2 = FALSE
 set the visible of sprite KeySprite-1 = FALSE
 set the visible of sprite KeySprite = FALSE
 set the visible of sprite 93 = TRUE

 repeat with x=1 to times
   set the locV of sprite 93 = the locV of sprite 93 - 3
   set the locH of sprite 93=the locH of sprite 93-(integer(3*width/height))
  wait
  updateStage
 end repeat
 set the loc of sprite 93 to the loc of sprite parentKeySprite
 updateStage

 puppetSprite parentKeySprite-1, TRUE
 puppetSprite parentKeySprite, TRUE
 set the foreColor of sprite (parentKeySprite-1)=the foreColor of member "movingNode"
 set the text of member (the memberNum of sprite parentKeySprite)=the text of member
"movingNode"

 set the visible of sprite parentKeySprite = TRUE
 set the visible of sprite parentKeySprite-1 = TRUE
 set the visible of sprite 93 = FALSE
 updateStage

 set leftChild=2*nodeNum
 set rightChild=2*nodeNum+1

 if (3*leftChild-1)<90 then
   if the visible of sprite (3*leftChild-1)=TRUE then
    if leftChild=6 then
      LRtransferSubTree1 leftChild
    else if leftChild=10 or leftChild=14 then
      LRtransferSubTree2 leftChild
    else
      set newNode=leftChild-1
      nodeMoving(newNode, leftChild)
    end if
  end if
 end if

 if (3*rightChild-1)<90 then
   if the visible of sprite (3*rightChild-1)=TRUE then
    if rightChild=7 then
      LRupingSubTree1 rightChild
    else if rightChild=11 or rightChild=15 then
      LRupingSubTree2 rightChild
    else
      nodeMoving(rightChild/2, rightChild)
    end if
  end if
  end if
end

-- ********************************************************************
on LRupingSubTree1 subRootNum
  if subRootNum=7 then
    set newNode=3
  else if subRootNum=14 or subRootNum=15 then
    set newNode=subRootNum-8
  else
    set newNode=subRootNum-16
  end if
  nodeMoving(newNode, subRootNum)

  set leftChild=2*subRootNum
  set rightChild=leftChild+1
  if leftChild<31 then
    if the visible of sprite (3*leftChild-1)=TRUE then
     LRupingSubTree1 leftChild
    end if
  end if
  if rightChild<31 then
    if the visible of sprite (3*rightChild-1)=TRUE then
     LRupingSubTree1 rightChild
    end if
  end if
end

-- ********************************************************************
on LRupingSubTree2 subRootNum
  if subRootNum=11 then
    set newNode=5
  else if subRootNum=15 then
    set newNode=7
  else if subRootNum=22 or subRootNum=23 then
    set newNode=subRootNum-12
  else
    set newNode=subRootNum-16
  end if
  nodeMoving(newNode, subRootNum)

  set leftChild=2*subRootNum
  set rightChild=leftChild+1
  if leftChild<31 then
    if the visible of sprite (3*leftChild-1)=TRUE then
     LRupingSubTree2 leftChild
    end if
  end if
  if rightChild<31 then
    if the visible of sprite (3*rightChild-1)=TRUE then
     LRupingSubTree2 rightChild
    end if
  end if
end
```

```
-- ***********************************************************************
on LRtransferSubTree1 subRootNum
  set leftChild=2*subRootNum
  set rightChild=leftChild+1

  if leftChild<31 then
    if the visible of sprite (3*leftChild-1)=TRUE then
      LRtransferSubTree1 leftChild
    end if
  end if
  if rightChild<31 then
    if the visible of sprite (3*rightChild-1)=TRUE then
      LRtransferSubTree1 rightChild
    end if
  end if

  if subRootNum=6 then
    set newNode=5
  else if subRootNum=12 or subRootNum=13 then
    set newNode=subRootNum-2
  else
    set newNode=subRootNum-4
  end if

  nodeMoving(newNode, subRootNum)
end

-- ***********************************************************************
on LRtransferSubTree2 subRootNum
  set leftChild=2*subRootNum
  set rightChild=leftChild+1

  if leftChild<31 then
    if the visible of sprite (3*leftChild-1)=TRUE then
      LRtransferSubTree2 leftChild
    end if
  end if
  if rightChild<31 then
    if the visible of sprite (3*rightChild-1)=TRUE then
      LRtransferSubTree2 rightChild
    end if
  end if

  if subRootNum=10 or subRootNum=14 then
    set newNode=subRootNum-1
  else
    set newNode=subRootNum-2
  end if

  nodeMoving(newNode, subRootNum)
end

on nodeMoving newNode, subRootNum
```

```
  set newKeySprite=3*newNode-1
  set newNodeSprite=newKeySprite-1
  set subRootKeySprite=3*subRootNum-1
  set subRootNodeSprite=subRootKeySprite-1
  puppetSprite newKeySprite, TRUE
  puppetSprite newNodeSprite, TRUE
  puppetSprite newNodeSprite-1, TRUE
  puppetSprite subRootKeySprite, TRUE
  puppetSprite subRootNodeSprite, TRUE
  puppetSprite subRootNodeSprite-1, TRUE

  set the text of member (the memberNum of sprite newKeySprite) = the text of member (
the memberNum of sprite subRootKeySprite)
  set the foreColor of sprite newNodeSprite = the foreColor of sprite subRootNodeSprite

  set the visible of sprite (newNodeSprite-1)=TRUE
  set the visible of sprite newKeySprite=TRUE
  set the visible of sprite newNodeSprite=TRUE
  set the visible of sprite subRootKeySprite=FALSE
  set the visible of sprite subRootNodeSprite=FALSE
  set the visible of sprite subRootNodeSprite-1=FALSE
  updateStage
end

-- **********************************************************************
on LRmoveSubTreeDown1 subRootNum
  set leftChildNum=2*subRootNum
  set rightChildNum=2*subRootNum+1
  if subRootNum<16 then
    if the visible of sprite (3*leftChildNum-1)=TRUE and (the text of member (the
memberNum of sprite (3*leftChildNum-1)))<>"" then
      LRmoveSubTreeDown1(leftChildNum)
    end if

    if the visible of sprite (3*rightChildNum-1)=TRUE and (the text of member (the
memberNum of sprite (3*rightChildNum-1)))<>"" then
      LRmoveSubTreeDown1(rightChildNum)
    end if
  end if

  if subRootNum>=2 and subRootNum<4 then
    set newNum=subRootNum+2
  else if subRootNum>=4 and subRootNum<6 then
    set newNum=subRootNum+4
  else if subRootNum>=6 and subRootNum<8 then
    set newNum=subRootNum+6
  else if subRootNum>8 and subRootNum<12 then
    set newNum=subRootNum+8
  else if subRootNum>=12 and subRootNum<16 then
    set newNum=subRootNum+12
  else
    decline
    exit
  end if
  nodeMoving(newNum, subRootNum)
end

-- ********************************************************************
on LRmoveSubTreeDown2 subRootNum
  set leftChildNum=2*subRootNum
  set rightChildNum=2*subRootNum+1
  if subRootNum<16 then
    if the visible of sprite (3*leftChildNum-1)=TRUE and (the text of member (the
memberNum of sprite (3*leftChildNum-1)))<>"" then
      LRmoveSubTreeDown2(leftChildNum)
    end if

    if the visible of sprite (3*rightChildNum-1)=TRUE and (the text of member (the
memberNum of sprite (3*rightChildNum-1)))<>"" then
      LRmoveSubTreeDown2(rightChildNum)
    end if
  end if

  if subRootNum=10 then
    set newNum=subRootNum+10
  else if subRootNum=14 then
    set newNum=subRootNum+14
  else
    decline
    exit
  end if

  nodeMoving(newNum, subRootNum)
end

-- ********************************************************************
on decline
  alert "Out of stage! Please rebuild the tree."
  rebuild
end

-- ********************************************************************
on ColorRootBlack
  longwait
  puppetSprite 1, TRUE
  set the foreColor of sprite 1 to 255
  updateStage
end

-- ********************************************************************
on cleardata
  set the text of member "FieldInput" to ""
  set the text of member "movingNode" to ""
end cleardata

-- ********************************************************************
on toggleVisible channel, state
  set the visible of sprite channel to state
```

```
end

-- ********************************************************************
on wait
  repeat with x=1 to 15000
    nothing
  end repeat
end

-- ********************************************************************
on longWait
  repeat with x=1 to 5
   wait
  end repeat
end

-- ********************************************************************
on invisible
  repeat with x=1 to 94
   toggleVisible(x, FALSE)
  end repeat
end

-- ********************************************************************
on getParentNum child
  set parentNum=child/2

  return parentNum
end

-- ********************************************************************
on getGrandparentNum grandChild
  set child=getParentNum(grandChild)
  return getParentNum(child)
end

-- ********************************************************************
on getColor nodeNum
  set nodeSprite=(3*nodeNum-1)-1
  if the visible of sprite nodeSprite=0 then
    set color=255
 else
    set color=the foreColor of sprite nodeSprite
 end if
 return color
end

-- ********************************************************************
on colorBlack nodeNum
 set nodeSprite = 3*nodeNum-1
 puppetSprite nodeSprite-1, TRUE
 set the foreColor of sprite nodeSprite-1 to 255
 updateStage
end
```

```
-- **********************************************************************
on colorRed nodeNum
  set nodeSprite = 3*nodeNum-1
  puppetSprite nodeSprite-1, TRUE
  set the foreColor of sprite nodeSprite-1 to 35
 updateStage
end

-- **********************************************************************
-- *                          Deletion Routine                           *
-- **********************************************************************
on delete
  global gNodeList, gOperationList

  if the text of member "FieldInput" = "" then
    alert "Please enter an integer in the box."
    exit
  end if

  set deletion=value(the text of member "FieldInput")
  if getOne(gNodeList, deletion)=0 then
    alert "Key not found, please select another one."
    exit
  end if

  set keyNum=findKeyNum(deletion)

  if the visible of sprite (3*(2*keyNum)-1)=FALSE or the visible of sprite
(3*(2*keyNum+1)-1)=FALSE then
    set delNum=keyNum
  else
    set delNum=findsuccessor(keyNum)
  end if

  if the visible of sprite (3*2*delNum-1)=TRUE then
    set shortness=2*delNum
  else
    set shortness=2*delNum+1
  end if

  set the text of member (the memberNum of sprite (3*keyNum-1)) to ""
  wait
  set delColor=the foreColor of sprite (3*delNum-2)
  set the visible of sprite 3*delNum-1 to FALSE
  set the visible of sprite 3*delNum-2 to FALSE
  set the visible of sprite 3*delNum-3 to FALSE
  wait

  if keyNum<>delNum then
    set height=the locV of sprite (3*delNum-1)-the locV of sprite (3*keyNum-1)
    set width=the locH of sprite (3*delNum-1)-the locH of sprite (3*keyNum-1)
    set times=integer(height/3)-1
    puppetSprite 3*keyNum-1, TRUE
    puppetSprite 93, TRUE
    set the text of member "movingNode"= the text of member (the member of sprite
(3*delNum-1))
    set the foreColor of sprite 93= the foreColor of sprite (3*keyNum-1)
    set the loc of sprite 93 to the loc of sprite (3*delNum-1)
    set the visible of sprite 93 to TRUE
    updateStage
    repeat with x=1 to times
      set the locV of sprite 93 = the locV of sprite 93 - 3
      set the locH of sprite 93=the locH of sprite 93-(integer(3*width/height))
     wait
     updateStage
    end repeat

    set the text of member (the memberNum of sprite (3*keyNum-1)) to the text of member
"movingNode"
    set the visible of sprite 93 to FALSE
    updateStage
  else
    if the foreColor of sprite (3*delNum-2)=35 then
      --the visible of sprite (3*2*delNum-1)=FALSE and the visible of sprite
(3*(2*delNum+1)-1)=FALSE then
      set the visible of sprite (3*delNum-1)=FALSE
      deleteOne gNodeList, deletion
      add gOperationList, "Delete"&& deletion
      cleardata
      exit
    end if
  end if

  set shortnessColor=getColor(shortness)
  if delColor=255  then
    puppetSprite 94, TRUE
    puppetSprite (3*delNum-2), TRUE
    if shortnessColor=35 then
      puppetSprite (3*delNum-1), TRUE
      set the foreColor of sprite (3*delNum-2) to shortnessColor
      set the text of member (the memberNum of sprite (3*delNum-1)) to the text of member
(the memberNum of sprite (3*shortness-1))
      set the visible of sprite 3*shortness-1 to FALSE
      set the visible of sprite 3*shortness-2 to FALSE
      set the visible of sprite 3*shortness-3 to FALSE
      set the visible of sprite 3*delNum-1 to TRUE
      set the visible of sprite 3*delNum-2 to TRUE
      set the visible of sprite 3*delNum-3 to TRUE
    else
      set the foreColor of sprite (3*delNum-2)=255
      set the visible of sprite (3*delNum-2)=TRUE
    end if

    set the loc of sprite 94 to point(the locH of sprite (3*delNum-1)-16, the locV of sprite
(3*delNum-1)-17)
    set the visible of sprite 94 to TRUE
    updateStage
    longwait
    fixup(delNum)

  end if

  deleteOne gNodeList, deletion
  add gOperationList, "Delete"&& deletion
  cleardata
end delete

-- ********************************************************************
on findKeyNum key
  set nodeNum=1
  set nodeKeySprite=3*nodeNum-1

  repeat while the visible of sprite nodeKeySprite=TRUE
    set nodeKey=value(the text of member (the memberNum of sprite nodeKeySprite))

    if nodeKey=key then
      set theNum=nodeNum
      exit repeat

    else if nodeKey>key then
      set nodeNum=nodeNum*2
    else
      set nodeNum=2*nodeNum+1
    end if

    set nodeKeySprite=3*nodeNum-1
  end repeat

  return theNum
end

-- ********************************************************************
on findSuccessor keyNum
  set successor=2*keyNum+1
  set theLeft=2*successor

  repeat while the visible of sprite (3*theLeft-1)=TRUE
    set successor=theLeft
    set theLeft=theLeft*2
  end repeat

  return successor
end

-- ********************************************************************
-- *                    Fix-Up handler of deletion routine                    *
-- ********************************************************************
on fixup shortNum
  set times =1
  repeat while shortNum<>1 and getColor(shortNum)=255
```

```
   if (shortNum mod 2)=0 then
     set theNum=shortNum+1
     if getColor(theNum)=35 then
       colorBlack(theNum)
       colorRed(shortNum/2)
        leftRotate(shortNum/2)
       set shortNum=2*shortNum
        set theNum=2*(theNum-1)+1
    end if

      if getColor(theNum*2)=255 and getColor(theNum*2+1)=255 then
       colorRed(theNum)
       if times=1 then
         set the visible of sprite (3*shortNum-1) to FALSE
         set the visible of sprite (3*shortNum-2) to FALSE
         set the visible of sprite (3*shortNum-3) to FALSE
      end if

       set times=times+1
       set shortNum=shortNum/2

       puppetSprite 94, TRUE
        set the loc of sprite 94 to point(the locH of sprite (3*shortNum-1)-16, the locV of
sprite  (3*shortNum-1)-17)
       set the visible of sprite 94 to TRUE
      updateStage
      longwait

    else
       if getColor(2*theNum+1)=255 then
        colorBlack(2*theNum)
        colorRed(theNum)
         rightRotate(theNum)
       --         set theNum=theNum*2+1
      end if

       puppetSprite (3*theNum-2), TRUE
        set the foreColor of sprite (3*theNum-2) to the foreColor of sprite (3*(shortNum/2)-2)
      updateStage
       colorBlack(shortNum/2)
       colorBlack(2*theNum+1)
        leftRotate(shortNum/2)
      set shortNum=1
    end if

  else
     set theNum=shortNum-1
     if getColor(theNum)=35 then
       colorBlack(theNum)
       colorRed(shortNum/2)
        rightRotate(shortNum/2)
       set shortNum=2*shortNum+1
       set theNum=2*(theNum+1)
    end if
```

```
      if getColor(theNum*2)=255 and getColor(theNum*2+1)=255 then
       colorRed(theNum)
       if times=1 then
         set the visible of sprite (3*shortNum-1) to FALSE
         set the visible of sprite (3*shortNum-2) to FALSE
         set the visible of sprite (3*shortNum-3) to FALSE
      end if
       set shortNum=shortNum/2
       set times=times+1

       puppetSprite 94, TRUE
        set the loc of sprite 94 to point(the locH of sprite (3*shortNum-1)-16, the locV of
sprite (3*shortNum-1)-17)
       set the visible of sprite 94 to TRUE
      updateStage
      longwait

    else
       if getColor(2*theNum)=255 then
         colorBlack(2*theNum+1)
        colorRed(theNum)
         leftRotate(theNum)
      end if

       puppetSprite (3*theNum-2), TRUE
        set the foreColor of sprite (3*theNum-2) to the foreColor of sprite (3*(shortNum/2)-2)
      updateStage
       colorBlack(shortNum/2)
       colorBlack(2*theNum)
       rightRotate(shortNum/2)
      set shortNum=1
    end if
  end if
 end repeat

 colorBlack(shortNum)
 set the visible of sprite 94 to FALSE
end
```

2

VITA

Yongzhong Zhang

Candidate for the Degree of

Master of Science

Thesis: RED-BLACK TREE ANIMATION

Major Field: Computer Science

Biographical:

Personal Data: Born in Guangzhou, Guangdong, P. R. China, on August, 1968, the son of Baojuan Gao and Hailin Zhang, and grandson of Muzhen Diao and Daolong Zhang; married to Tong Xie in 1995.

Education: Graduated from No. 16 High School, Guangzhou, Guangdong, P. R. China in June, 1986. Received Bachelor of Science degree in Chemistry from Zhongshan University, Guangzhou, Guangdong, P. R. China in July, 1990. Completed the requirements for the Master of Science degree with a major in Computer Sciences at Oklahoma State University in December, 1998.

Experience: Employed as Assistant Engineer by Nanzhong Organic Chemical Plant, Guangzhou, Guangdong, P. R. China, 1990 - 1992, Employed as Technical Representative by Coates (Guangzhou) PRC Ltd., Guangzhou, Guangdong, P. R. China, 1992 - 1994, Employed by ICI Swire Paints (China) Ltd. as a Senior Technical Sales Representative, Guangzhou, Guangdong, P. R. China, 1994 - 1996.