

RED-BLACK TREE ALGORITHM ANIMATION  
USING JAVA

By

PENGCHENG CHEN

Bachelor of Science  
East China Normal University  
Shanghai, P. R. China  
1986

Master of Science  
East China Normal University  
Shanghai, P. R. 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

RED-BLACK TREE ALGORITHM ANIMATION  
USING JAVA

Thesis Approved:

*Jacques E. LaFrance*  
\_\_\_\_\_  
Thesis Advisor

*J. Chandler*  
\_\_\_\_\_

*D. E. Hedman*  
\_\_\_\_\_

*Wayne B. Powell*  
\_\_\_\_\_  
Dean of the Graduate College

## ACKNOWLEDGMENT

I wish to express my sincere appreciation to my advisor Dr. Jacques. Lafrance for his guidance, patience, kindness and encouragement throughout my studies and finishing my thesis. I am deeply grateful to Dr. J. P. Chandler and Dr. G. E. Hedrick for their serving on my graduate committee and providing me invaluable advice and suggestions.

I am greatly indebted to my parents for their unending love and care throughout my life. Special thanks go to my wife, Jing Yu, for her love, understanding encouragement and support, both spiritual and financial. Without her love and support, I would not have completed my study today.

## TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION .....	1
2. RELATED WORKS .....	3
3. JAVA LANGUAGE .....	4
4. RED-BLACK TREE .....	5
5. DESIGN AND IMPLEMENTATION .....	14
5.1 Development Process.....	14
5.2 Data Type Declaration .....	15
5.3 Decomposition Algorithm .....	18
5.4 Visualization Design .....	22
5.5 Component Design .....	27
5.6 User Interface .....	31
5.7 Sound Display .....	39
6. SYSTEM OVERVIEW.....	40
7. SUMMARY AND FUTURE WORK .....	50
8. REFERENCES .....	52

### LIST OF FIGURES

Figure	Page
1. Red-black Tree Color Flip.....	7
2. Variations of Nodes being Inserted .....	9
3. Insert Node Into the Tree .....	9
4. Parent is Red and X is Inside Child .....	11
5. Outside Grandchild Lowers its Level .....	12
6. Inside Grandchild on the Way Down .....	13
7. The Designed System .....	13
8. The System Design Flow Chart .....	15
9. Storage Structure for A Node of Red Black Tree and its Declaration .....	16
10. The Decomposition of the Algorithm .....	23
11. The Designed Image Represent in the Screen .....	24
12. Java Coordinate System .....	24
13. A Portion of the Java .awt Inheritance Hierarchy .....	32
14. The Designed Starting Window .....	33
15. The Designed Main Window .....	36
16. The Designed Help Window in Main Window .....	37
17. The Designed Input Window .....	37
18. Designed Text Field in the Main Window .....	38
19. Designed Exiting Window .....	39
20. The DOS Command to Run The System .....	40
21. The Starting Window of the System .....	41
22. The Help Window from Starting Window .....	42
23. The Main Window of the System .....	43

24. The Help Window in the Main Window .....	44
25. The Scene of the 'Demo' Button was Pressed in Main Window .....	45
26. The Input Window .....	44
27. The Exiting Window .....	45
28. Step 1: The Node 13 Was Inserted Into the Tree .....	46
29. Step 2: Insert 45 Into the Tree .....	47
30. Step 3: Insert 23. The Tree is Unbalanced .....	47
31. Step 4: the Node Will Rotate With 45 .....	48
32. Step 5: The Action Note Showed Need Color Flip.....	48
33. Step 6: After Color Flip .....	49
34. Step 7: Finally the Tree Was Balanced .....	49

# I. INTRODUCTION

Data structures are ways in which data is arranged in the computer's memory (or stored on disk). Algorithms are the procedures that a software program uses to manipulate the data in these structures. Almost every computer program, even a simple one, uses data structures and algorithms. Data structures include linked lists, stacks, binary trees, hash tables, and others. Algorithms manipulate the data in these structures in various ways, such as searching for a particular data item and sorting the data. It is very important for computer science students to understand thoroughly the concepts and principles of various basic data structures, and apply them to their work. The red-black tree is a very important data structure and very efficient. But it is difficult for students to understand its implementing algorithms. According to Weiss (Weiss, 1997), the red-black tree is an advanced data structure. The goal of this study is to design and implement an animated presentation of red-black tree animation using the Java language. It will allow the people to explore visually different scenarios in implementing the different operations of red-black trees.

“Visualization is a part of our daily lives” (Schroeder, 1996). From weather maps to the exciting computer graphics of the entertainment industry, examples of visualization abound. Informally, visualization is the transformation of data or information into pictures. Visualization engages the primary human sensory apparatus, vision, as well as the processing power of the human mind. The result is a simple and effective medium for communicating complex and /or voluminous information. Visualization is the process of exploring, transforming, and viewing data as images (or other sensory forms)

to gain understanding and insight into the data. A picture is worth a thousand words.

Visualization transforms the data code into an image, enabling people to observe simulations and computations. Combined with multimedia, graphics, and graphical user interfaces, visualization uses carefully designed representations to help people to understand. In courses dealing with algorithms and data structures, this can be a very useful tool to demonstrate concepts. Students can exercise with the system by viewing animations for different input values, which can reveal some special cases that were not known and understood by the student before.

Animated courseware is also very useful for teachers in explaining to their students what the algorithm does and what it is supposed to do. For example, a student can very easily compare the advantages and disadvantages of different sorting methods through animated images (after entering the same set of values) and they can compare the underlying algorithms more easily. Animation includes both dynamic display and static display. The present study aims at dynamically displaying the changes in the red-black tree data structure. Such displays can lead to a better understanding of algorithms and data structures.



## II. RELATED WORK

Visualization as a method to explain the algorithms and programs has been studied since the 1960s. In the early days, most of the visualization was done with tapes and films and they were static. The bad part of such a system was that it did not allow users to explore different situations. In 1966, Knowlton produced the first computer-generated movie (Knowlton, 1966). It showed how an assembly-level list processing language works. Later other people also made data structure films, such as Hopgood's film on hashing table algorithms, Booth's on "PQ-tree" (Booth, 1974).

Since the mid-1970s there has been more and more research on animation of data structures. Several systems were built that automatically produced a static graphical display of a program's data structures (Thomas, 1985; Yarwood, 1974; Stasko, 1990, etc). These systems gave a static picture of the data structure. But they could not represent what operation was being performed and how the data structure changed. They only showed the image of the data structure before and after the operation.

In recent years more and more work has been done in visualizing the data structure through various tools and environments. Shen (Shen, 1994) designed a visual system for learning of tree-based data structures; Lin (Lin, 1997) implemented an object-oriented graphic user interface for visualization of B-trees' animation; Harvick designed rule based data structure animation (Harvick, 1997). An animated construction of a B-tree was created using Macromedia Director. This has also been done using C++ (Lin, 1997). Work were also done on stacks, queues and lists (Xu, 1997), and binary search trees (Shen, 1997). All the above research show the data structure more clearly and directly and some of them include multimedia techniques.

The future of computer visualization and graphics appears to be explosive. Just a few decades ago, the field of data visualization did not exist; more and more computer graphics was viewed as an offshoot of the more formal discipline of computer science. As techniques were created and computer power increased, scientists and other researchers began to use graphics to understand and communicate information. At the same time, user interface tools were developed. These forces have now converged to the point where we expect computers to adapt to humans rather than the other way around. Now using the visualization window, we can extract information from data, analyze, understand, and manage more complex systems than ever before.

### **III. JAVA LANGUAGE**

Java is a new language. In 1991, Sun Microsystems funded an internal corporate research project code-named Green in 1991. The project resulted in the development of a C and C++ based language that its creator, James Gosling, called Oak after an Oak tree outside his window at Sun. It was later discovered that there was already a computer language called Oak. When a group of Sun people visited a local coffee place, the name Java was suggested and it stuck. In May 1995, Sun formally announced Java.

Java allows programmers to create applets, programs that may be “embedded” inside Web pages; On the other hand, it is a full-fledged object-oriented language which lets you implement object-oriented applications very efficiently. The Java Development Kit provides a wealth of APIs to help one develop applications quickly. These APIs support developing networking applications, building platform-independent GUIs, multithreading for efficient use of system resources, implementing applets to launch

from Web browsers, handling input and output streams, and many others that will help you build applets as well as stand-alone applications.

An applet is a custom interface component object, similar in concept to a windows custom control, or a X-window widget. Applet-aware applications can load and construct applet objects from URLs pointing to CLASS files anywhere on a network. Using the graphical capabilities of Java, applets are visually executing multimedia elements. Through objects of the class Java.awt, Graphics applets can create graphical content on screen. Because of all these features, applets have become the good method for computer graphics animation and distributing interactive content on the World Wide Web.

#### **IV. RED-BLACK TREE**

A red-black tree is a balanced binary search tree; It is a binary search tree with an additional field--color. Ordinary binary search trees offer important advantages as data storage devices: you can quickly search for an item with a given key, and you can also quickly insert or delete an item. Other data storage structures, such as arrays, sorted arrays, and linked lists, perform one or the other of these activities slowly. Thus binary search trees might appear to be the ideal data storage structure. Unfortunately, ordinary binary search trees suffer from a troublesome problem. They work well if the data is inserted into the tree in random order. However, they become much slower if data is inserted in already sorted order. When the values to be inserted are already ordered, a binary tree becomes unbalanced. At this situation, the search becomes slow. The advantage of the red-black tree structure is that it can keep the tree balanced.

The characteristics of red-black tree:

- Every node in a red-black tree is either black or red
- Every null leaf is black
- No path from a leaf to a root can have two consecutive red nodes – i. e. the children of a red node must be black
- Every path from a node,  $x$ , to a descendant leaf contains the same number of black node – the “black height” of node  $x$ .

Characteristics third and fourth guarantee that a red-black tree is efficient. Since every path to a leaf contains the same number of black nodes, and any red node has black children, the longest possible path is composed of  $[2\lg N]+1$  nodes that alternate red and black. Since the smallest possible height for a binary tree with  $N$  nodes is  $[\lg N]$ , red-black trees offer performance that is twice optimal in the worst case.

Algorithms for red-black tree construction:

The construction of a red-black tree is more complicated than the construction of an ordinary binary search tree. The tree must follow the red-black rule. There are more cases that must be followed. In the discussion that follows,  $X$ ,  $P$ ,  $G$  are used to designate a pattern of related nodes.  $X$  is the node that has caused a rule violation;  $P$  is the parent of  $X$ .  $G$  is the grandparent of  $X$  (the parent of  $P$ ). The insertion routine in a red-black tree starts off doing essentially the same thing that it does in an ordinary binary search tree: It follows a path from the root to the place where the node should be inserted, going left or right at each node depending on the relative size of the node’s key and the search keys .

However, in a red-black tree, getting to the insertion point is complicated by color flips and rotations. To make sure the color rules are not broken, it needs to perform

color flips when necessary. The rule is: every time after inserting a new node, the tree encounters a black node that has two red children, it must change the children to black and the parent to red (unless the parent is the root, which always remains black). This can help to keep the black height unchanged. Figure 1 shows the nodes after the color flip. The flip leaves unchanged the number of black nodes on the path from the root on down through P to the leaf or null nodes. All such paths go through P, and then through either X1 or X2. Before the flip, only P is black, so the triangle (consisting of P, X1, X2) adds one black node to each of these paths. After the flip, P is no longer black, but both L and R are, so again the triangle contributes one black node to every path that passes through it. So a color flip won't cause rule fourth to be violated.

Color flips are helpful because they make red leaf nodes into black leaf nodes. This makes it easier to attach new red node without violating Rule 3.

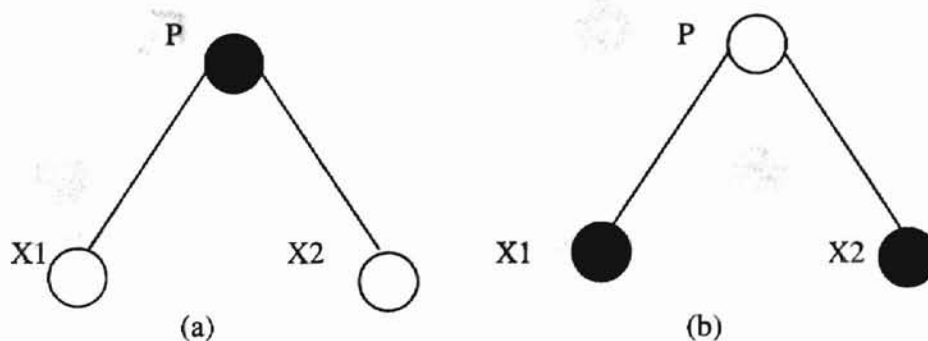


Fig 1 Red-black tree color flips

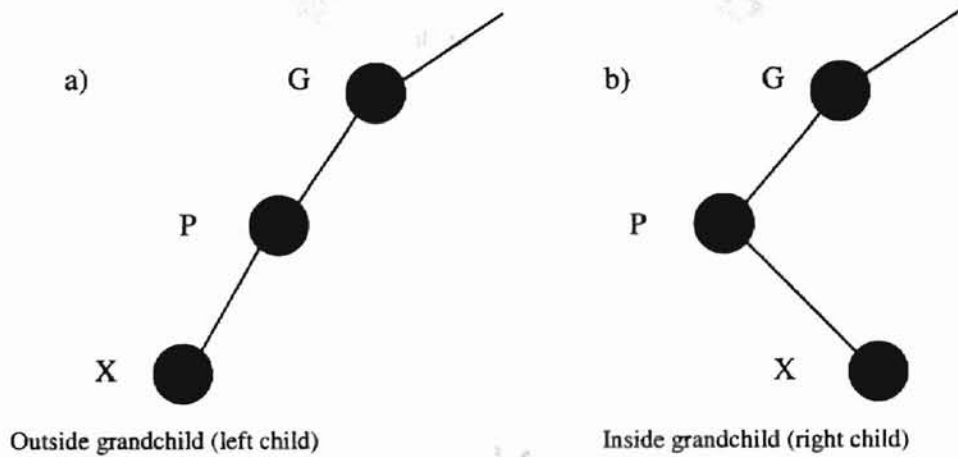
The insertion of a new node may cause the red-black rules to be violated. Therefore, following the insertion, we must check for rule violations and take appropriate steps.

A node X is an outside grandchild if it's on the same side of its parent P that P is of its parent G. Conversely, X is an inside grandchild if it's on the opposite side of its

parent P that P is of its parent G. If X is an outside grandchild, it may be either the left or right child of P, depending on whether P is the left or right child of G. Two similar possibilities exist if X is an inside grandchild (Fig 2.).

The color and configuration of X and its relatives determine the action we take to restore the red-black rules.

case 1: P is black. If P is black, we don't need to do anything else, just insert the node to the tree as a red node.



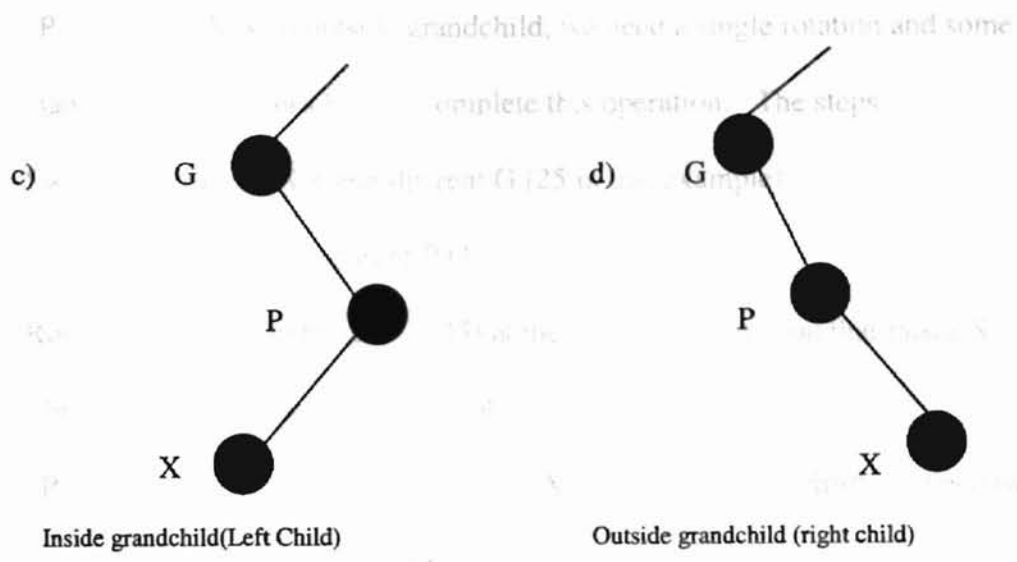


Fig 2 Variations of nodes being inserted

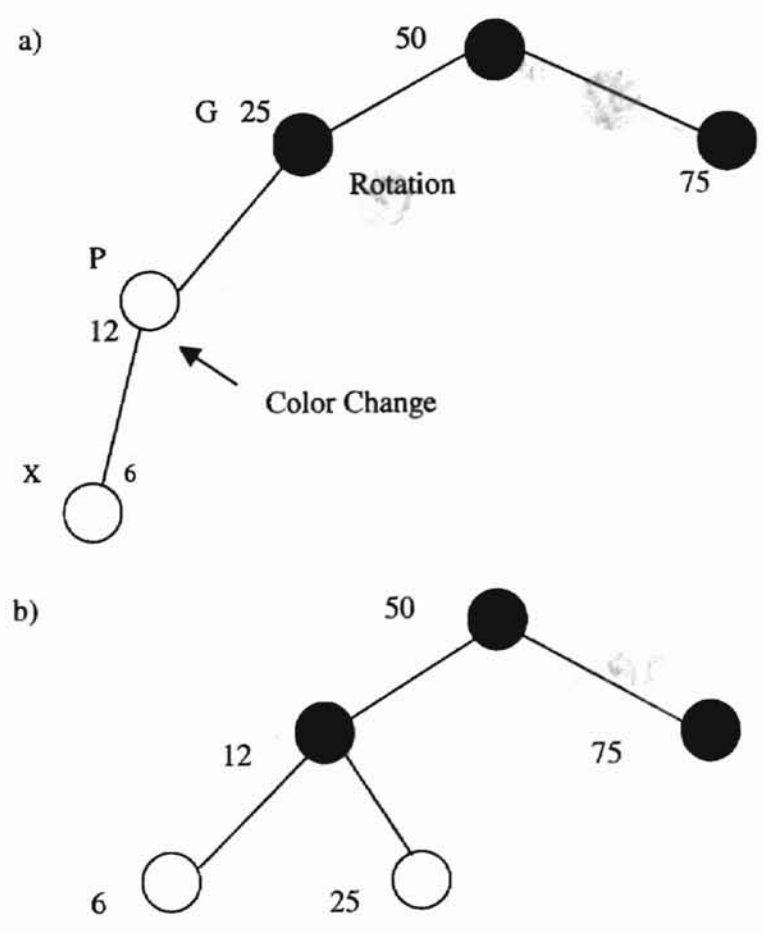


Fig 3. Insert node into the tree. P is red, X is Outside Child. a) before rotation. b) after rotation.

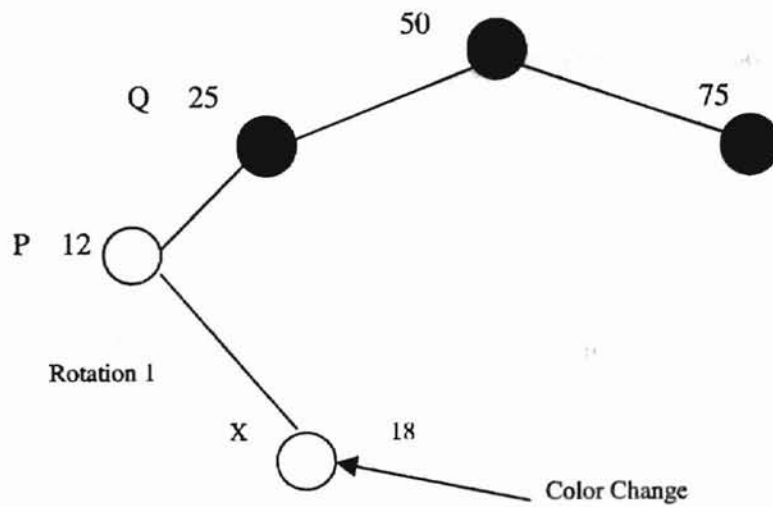
Case 2: P is red and X is an outside grandchild, we need a single rotation and some color changes. Fig 3. shows how to complete this operation. The steps:

1. Switch the color of X's grandparent G (25 in this example).
2. Switch the color of X's parent P (12).
3. Rotate with X's grandparent G (25) at the top, in the direction that raises X(6).

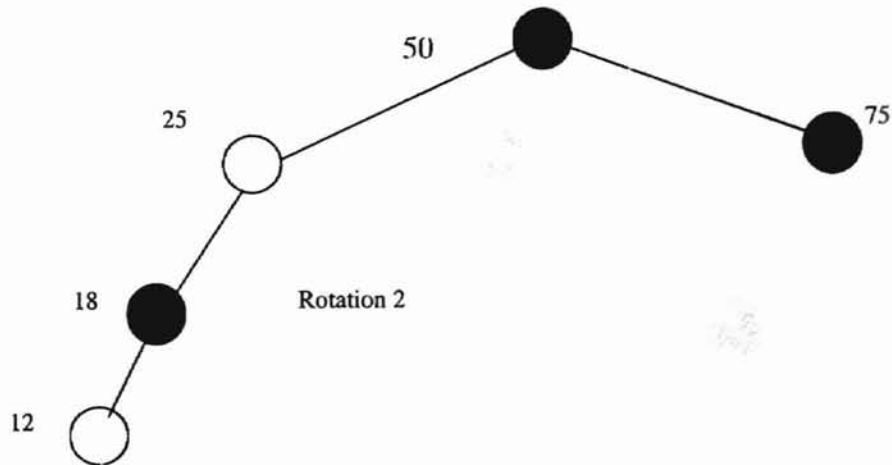
This is a right rotation in the example.

Case 3: P is red and X is inside: If P is red and X is an inside grandchild, we need two rotations and some color changes. (Fig. 4).

a)



b)





c)

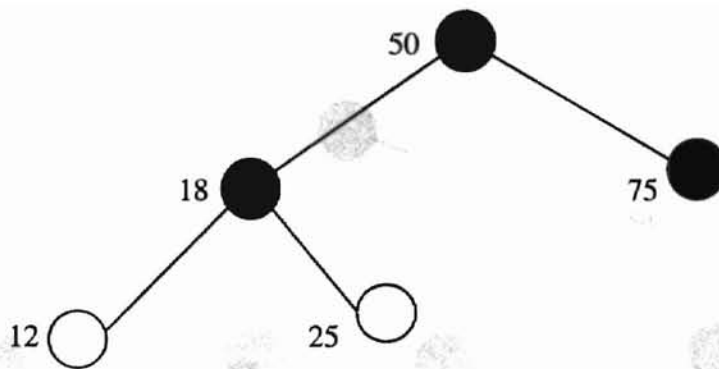


Fig 4. Parent is red and X is inside Child.

Rotation on the way down:

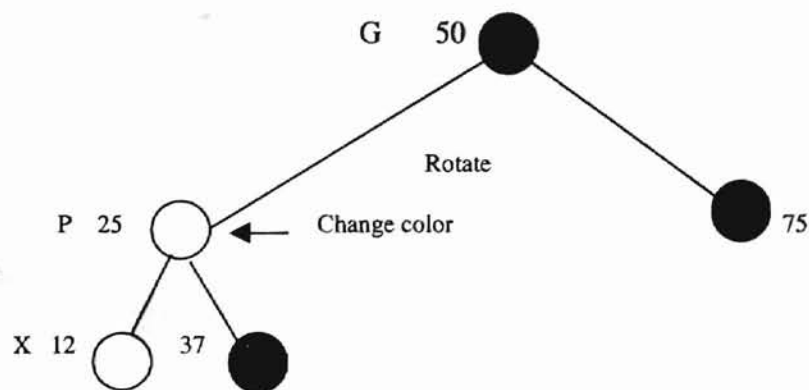
This operations to make rotations on the way down to the insertion point. It is possible for a color flip to cause a violation of Rule 3 (a parent and child can not both be red). A rotation can fix this problem.

Case 1: The Grandchild is in outside. (Fig. 5).

Steps:

1. Switch the color of X's grandparent G. Ignore the message that the root must be black.
2. Switch the color of X's parent P (25).
3. Rotate with X's grandparent at the top, in the direction that raises X.

a)



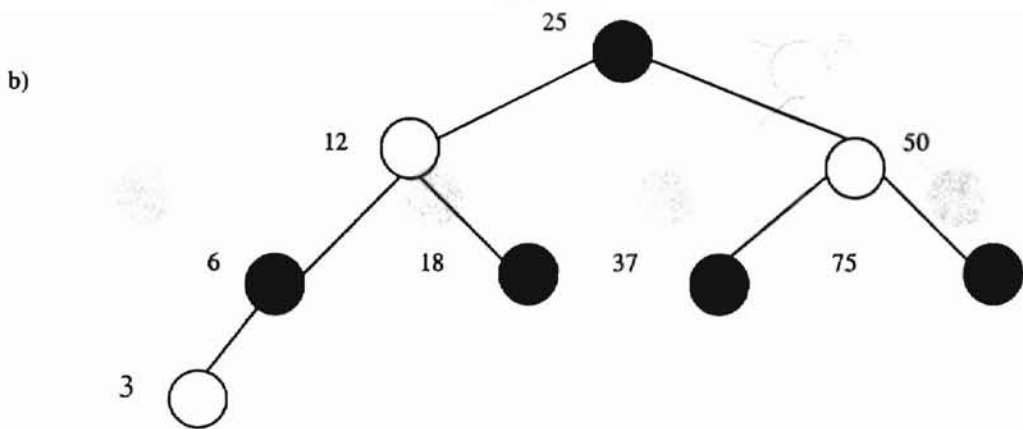
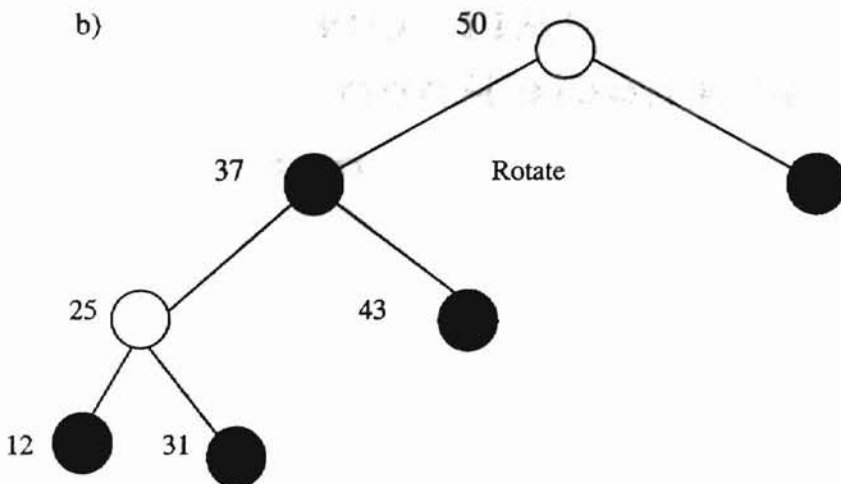
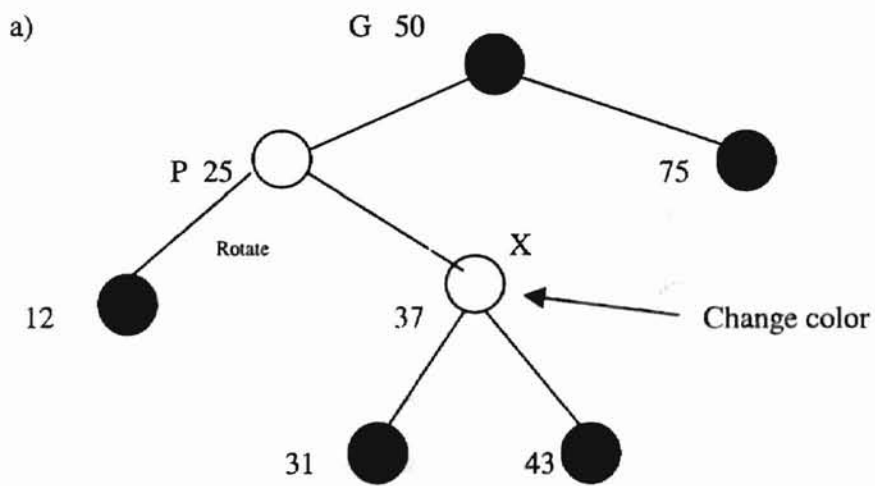
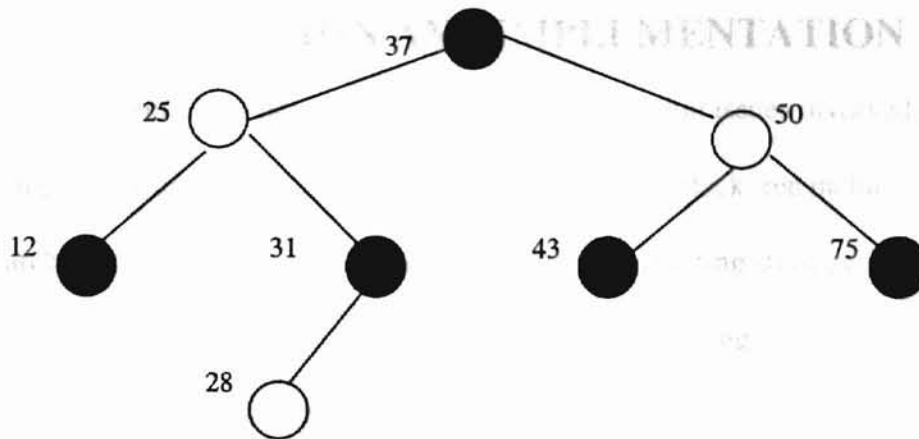


Fig 5. Outside grandchild lowers its level. a) Original tree. b) Final tree.



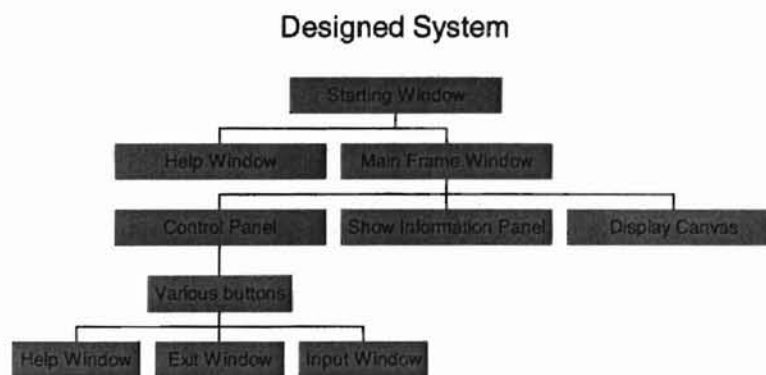
c)



*Fig. 6 Inside grandchild on the way down. a) original tree. b) changing tree. c) final tree.*

Case 2: The Grandchild is in inside (Fig 6, insert 28).

1. Change the color of G.
2. Change the color of X.
3. Rotate with P as the top, in the direction that raises X. (Fig 6. b).
4. Rotate with G as the top in the direction that raises X.



**Fig 7. The designed system**

## V. DESIGN AND IMPLEMENTATION

This chapter examines the design and implementation issues involved in developing the data structure animation program of red-black tree including the class hierarchies, the application framework, inserting and deleting strategy, dynamic display, and other related issues. The program was implemented using Java programming language, following Java and object-oriented programming and design conventions.

### v.1. Development Process

The algorithm animation system is designed by following the process model. This process model includes the activities shown below:

1. **Data type declaration:** the data type and storage structure for the Red-Black tree simulation system was defined clearly.
2. **Visualization design:** several classes were specified for visualization, and they were used to achieve Red-Black tree's algorithm visualization.
3. **Algorithm decomposition:** In order to show the algorithm to the user in a step by step mode, it is necessary to decompose the algorithms (both insertion and deletion) into a set of different parts that best represents the tree's behaviors during insertion and deletion operations. This is the key designing step for the system.
4. **Component design:** This part designs the services by this system. As figure 7 has shown, those are the services for the tree.

5. **User interface design:** The user interface will rely on windows, buttons, choice and mouse. This design is characterized by support for graphical as well as for textual information display.
6. **Sound display:** Sound is an important part of the system. This system use sound to explain some important information and use background music while the user operating the system.

The above activities are correlated to each other. In figure 8, we illustrate their designed relationship.

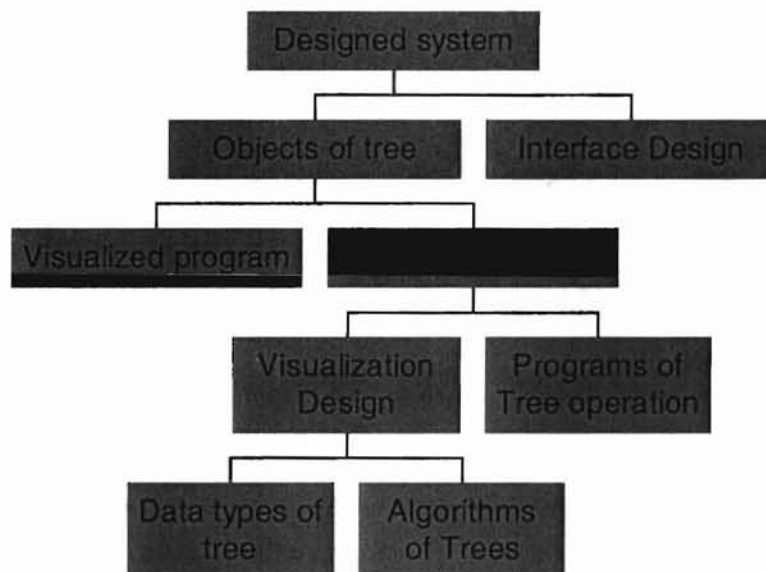


Figure. 8 *The system design flow chart (bottom up).*

## v.2. Data Type Declaration

Java language type declarations are used to specify the storage structures. This structure is very important for the whole program. First, the algorithm coding based on them, also the well-defined data structures have the fundamental information for

showing the images of the Red-black tree. Figure 9 shows the data type declaration for nodes in the tree.

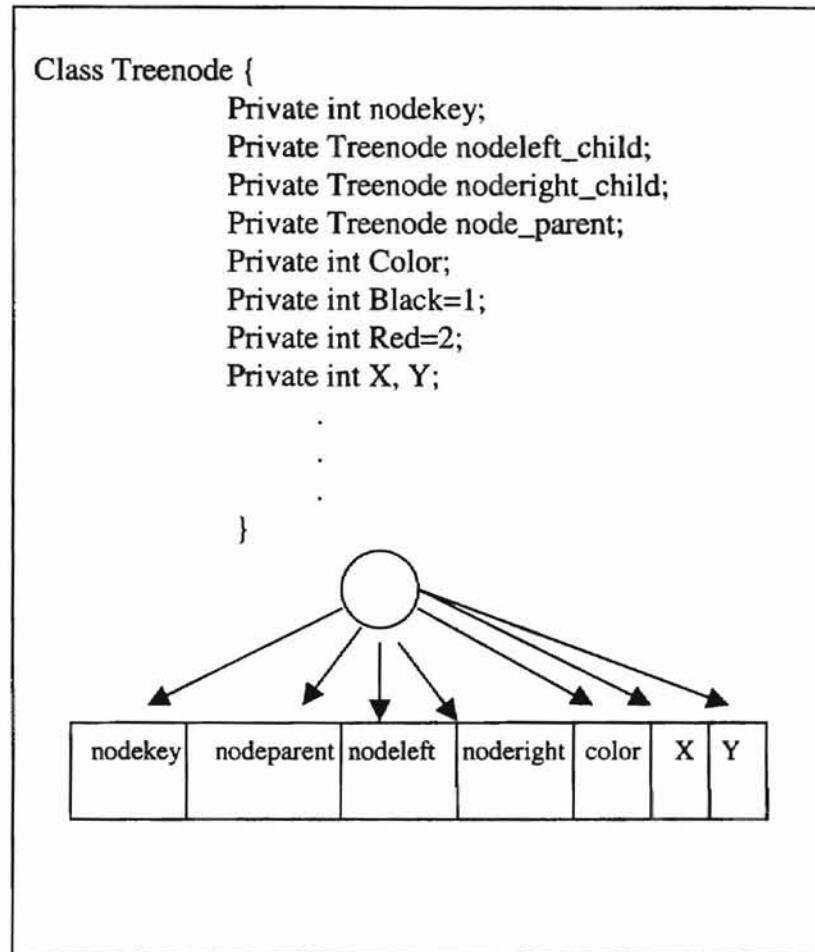


Fig. 9. Storage structure for a node of Red Black tree and its declaration.

In Fig 9, nodekey means the key value of the node. In this design, we use non-negative integer as input key value. Nodeleft\_child was used to represent its left child; Noderight\_child was used to represent its right child; Color was used to represent its color; X, Y was used to represent its location in the screen (X, Y). There are also some necessary initializations and node operations (such as set color, get color, set location, get location...) in this class as below.

```

class Treenode{
    public final static int L_flag=0;
    public final static int R_flag=1;
    public final static int P_flag=2;
    public final static int Black=1;
    public final static int Red=2;
    private int Color;

    private int nodekey;
    private Treenode nodeleft;
    private Treenode noderight;
    private Treenode nodeparent;
    private int X, Y;

    public Treenode()
    {
        Color=Black;
    }
    /* This function used to initialize the node */

    public Treenode(int key){
        Color=Red;
        nodekey=key;
    }

    /* This function used to set the node to the tree */

    public void node_seting(int keynode, Treenode node)
    {
        if(keynode==0)
            nodeleft=node;
        else if(keynode==1)
            noderight=node;
        else
            nodeparent=node;
    }

    /* This function used to find the related node */

    public Treenode Findnod(int keynode)
    {
        if(keynode==0)
            return nodeleft;
        else if(keynode==1)
            return noderight;
        else
            return nodeparent;
    }

    /* This function used to get the color. */

    public int getColor(){
        if(Color==Black)
            return 1;
        else
            return 2;
    }

    /* This function used to set the color of the node. */

    public void setColor(int color)

```

```

        {
            if(color==1)
                Color=Black;
            else
                Color=Red;
        }

    /* This function used to get the key value of the node */
public int getKey()
    {
        return nodekey;
    }

    /* This function used to set the key value of the node. */
public void setKey(int key){
        nodekey=key;
    }

    /* This function used to set the X and Y positions of the node */
public void setx(int x)
    {
        X =x;
    }

public void sety(int y)
    {
        Y =y;
    }
/* The function used to get the X and Y positions of the node. */
public int getx()
    {
        return x;
    }

public int gety()
    {
        return y;
    }
}

```

### v.3. Decomposition of the algorithm

The red-black tree operation in this simulation including insertion and deletion. The basic algorithms we used are according to Cormen (Cormen, H. T., 1997). The detailed algorithms are below:

Insertion:

Tree-Insert(T,x)



color[x] ← Red

While x ≠ root[T] and color[p[x]] = Red

Do if p[x] = left[p[p[x]]]

Then y ← right[p[p[x]]]

If color[y] = red

Then color[p[p[x]]] ← Black

color[y] ← Black

color[p[p[x]]] ← Red

x ← p[p[x]]

Else if x = right[p[x]]

Then x ← p[x]

Left-Rotate(T, x)

color[p[x]] ← Black

color[p[p[x]]] ← Red

Right-Rotate(T, p[p[x]])

Else (same as then clause with “right” and “left” exchanged)

Color[root[T]] ← Black.

Deletion:

RB-Delete(T, z)

If left[z] = nil[T] or right[z] = nil[T]

Then y ← z

Else y ← Tree-Successor(z)

If left[y] ≠ nil[T]

```

Then x ← left[y]      If color[left[y]] = Black
Else x ← right [y]    Then color[right[y]] ← Black
P[x] ← p[y]           color[w] ← Red
If p[y] = nil[T]      Then color[x] ← Red
Then root[T] ← x
Else if y = left[p[y]]
    Then left [p[y]] ← x
    Else right [p[y]] ← x
If y!=z
    Then key[z] ← key[y]
If color[y] =Black
    Then
        While x != root [T] and color =Black
            Do if x = left[p[x]]
                Then w ← right[p[x]]
                If color[w] = Red
                    Then color[w] ← Black
                    color[p[x]] ← Red
                    Left_Rotate ( T, p[x])
                    W ← right[p[x]]
                If color[left[w]] =Black and color[right[w]] = Black
                    Then color[w] ← Red
                    X ← p[x]

```

Else if  $\text{color}[\text{right}[w]] = \text{Black}$  we will break the

Then  $\text{color}[\text{left}[w]] \leftarrow \text{Black}$  recolor after

$\text{color}[w] \leftarrow \text{Red}$  case 1)

$\text{Right-Rotate}(T, w)$

$W \leftarrow \text{right}[p[x]]$

$\text{color}[w] \leftarrow \text{color}[p[x]]$

$\text{color}[p[x]] \leftarrow \text{Black}$

$\text{color}[\text{right}[w]] \leftarrow \text{Black}$

$\text{Left\_Rotate}(T, p[x])$

$X \leftarrow \text{root}[T]$

Else (same as then clause with 'right' and 'left'

exchange)

$\text{color}[x] \leftarrow \text{Black}$

According to above algorithms, we write insertion and deletion modules. Those basic operations include the functions of insertion and deletion. But for the reasons that the tree behaviors are to be visualized, the above operations must be sub-divided into a set of parts according to the tree attributes in order to make the user understand the algorithm step by step. We decomposed insertion and deletion into four parts:

- Insert or delete key.
- Check the balance of the tree.
- Rotation
- Recolor.

In this program, in each point of the program meet above conditions we will break the operation and show a command to the user what's the next operation (Ex. " need color flip or need rotate..") should be. User can push a button to operate next step. Fig 10 showed the decomposition procedure of the algorithms.

In this java program we use a specify button to operate this step by step operation, the screen will prompt user each step he should go and what the next step will be (Ex. Rotate, color flip,..).

#### V.4. Visualization design

The most important and difficult designing activities in this animation system are the visualization design. It will determine whether this animation is success or not in terms of its usage.

This system aims at animating the red-black tree algorithm. We are basically dealing with the nodes' movement to achieve the animation. With this, user could obtain a graphical picture of the data structure of the tree. As operations are being performed on a data structure the changes in the data structure are shown as movements and transitions of graphical objects that represent the data structure. The transitions

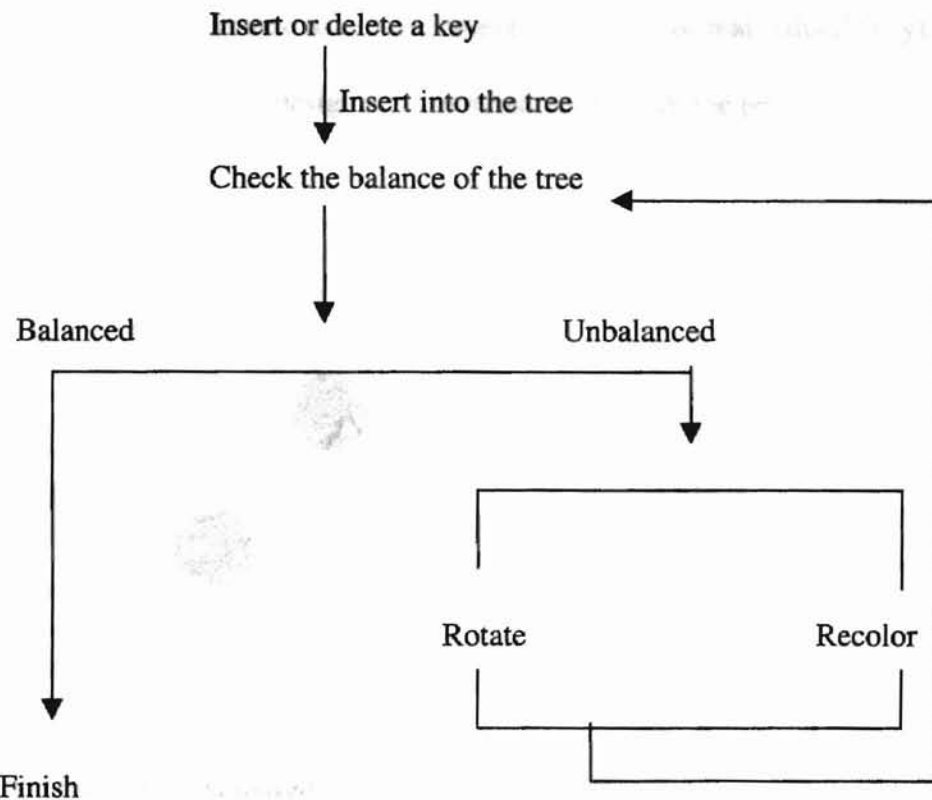


Fig. 10 *The decomposition of the algorithms.*

include changes in node location, color, etc. The operation will cause the tree nodes to rearrange and move to new locations or change color (red or black).

There are four major parts of visualization—graphical image, the locations of images, the images' transitions, and the paths that modify those transitions. An image is a graphical object that undergoes changing in location, color, etc. throughout the frames of the animation. In this system, the image includes lines, rectangles, and texts. Composite images are collections of primary images with geometric relationships to one another in a Java applet (Fig 11). A location is a position identified by an  $(x, y)$  coordinate pair in the Java coordinate system. (Fig.12). The ability to save and reference particular locations is an important part for animation design. Location  $(x, y)$  is stored in Treenode

class; A path designates the magnitude of change in image attributes from one frame to the next. A path is formally defined as a finite ordered sequence of real-valued  $(x, y)$  coordinate pairs, where each pair designates a relative offset from the previous position;

A

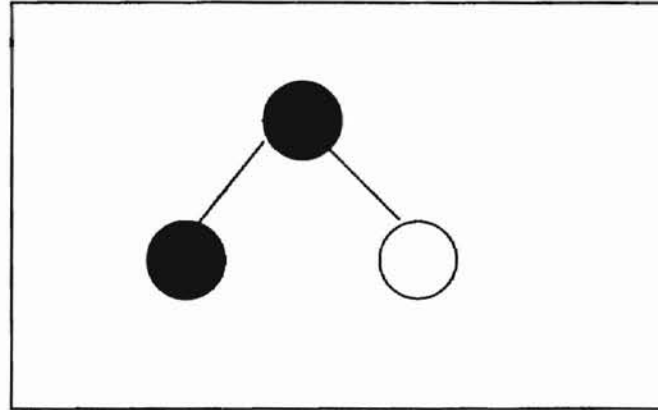


Fig 11. *The designed image represent in the screen.*

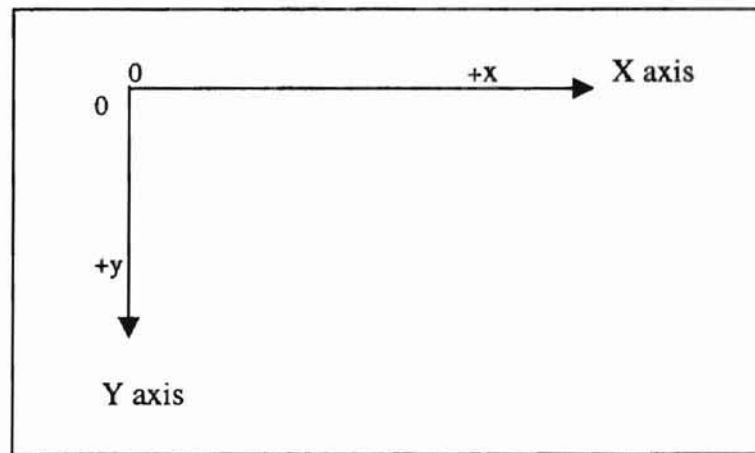


Fig.12 *Java coordinate system. Units are measured in pixels.*

transition uses a path parameter to modify an image's position or appearance, and to give an animation action. Typical transition types include move, resize, color, delay, alter visible. We can create animations of algorithms by assembling collections of image,

location, path, transition, and association operations that accomplish desired animation actions.

In the tree images, the rectangles show the nodes in the tree, and the lines coming out of the nodes indicate the relationship between nodes. Color is used to represent the color of the node (red or black). The text includes the key value in the nodes and the next step instruction. The following example Java code will draw the tree image (including node rectangular, line and node color):

m – diameter of rectangular, wide—width of the screen, verticle – verticle of the tree.

```

public void Node_drawing(Treenode node,int x,int y)
{
    . . .
    wide = wide / 2;
    if (node.Findnod(Treenode.L_flag) != nullnode)
    {
        offGraphic.setColor(Color.green);
        offGraphic.drawLine(x,y,Math.round(x-
wide),Math.round(y+verticle));
        /* Used to draw line between two nodes. */

        Node_drawing(node.Findnod(Treenode.L_flag),
            Math.round(x-wide),Math.round(y+verticle));
        /* recursively drawing the nodes on the
screen*/
    }
    if (node.getColor() == Treenode.Black)
    {
        offGraphic.setColor(Color.black);
        offGraphic.fillRect(x-m,y-m,2m,2m);
        /* used to draw rectangular as a node */
        offGraphic.setColor(Color.red);
        /* used to set the node's color */
    }

    if (node.getKey() < 10)
offGraphic.drawString(Integer.toString(node.getKey()),x-a,y+b);
/* used to draw key value(key <10) in the node */
    else
        offGraphic.drawString(Integer.toString(node.getKey()),x-
a,y+b);
    vertical = 2*vertical;
    node.setx(x); // update the node location .
    node.sety(y);
}

```

The paths are set up for the use of transitions. Once the locations are determined, the paths are set. For some path operations, such as insertion---receive locations from the root node down to the leaf node, and deletion--- receive two locations and create a path between them. In animation, the location of the node to be deleted is the motion's ending point and the location of the deleted node's successor is the motion's starting point.

Below is the Java code to operate the node transition:

```

...

startX = startnode.getx();          /* Starting point x */
startY = startnode.gety();          /* Starting point y */
endX = endnode.getx();              /* Ending point x */
endY = endnode.gety();              /* Ending point y */

LY = startY;
LX = startX;

offGraphic.fillRect(LX-m,LY-m,2m,2m);

Graphic.drawString(Integer.toString(startnode.getKey()),LX-
a,LY+b);
}

update(g);
for (cX= startX+i ; (cX*i) <= (destinationX*i) ; cX+=i)
{
    /* This for loop will move the node
    from starting point to ending point */
    cY = Math.round((cX-startX)*n) + startY;
    if (shape == F)
    {
        offGraphic.fillRect(LX-m,LY-m,2m,2m);
        offGraphic.fillRect(cX-m,cY-m,2m,2m);
    }
    else
    {
        offGraphic.fillRect(LX-m,LY-m,2m,2m);
    }
}

offGraphic.drawString(Integer.toString(startnode.getKey()),LX-a,LY+a);
offGraphic.fillRect(cX-m,cY-m,2m,2m);

offGraphic.drawString(Integer.toString(startnode.getKey()),cX-a,cY+a);
}
update(g);          /* used to clear the screen for next
drawing */
LX = cX;
LY = cY;
}

```



```

        if (status == Finish)
            offGraphic.fillRect(LX-m,LY-m,2m,2m);
        else
        {
            offGraphic.fillRect(LX-m,LY-m,2m,2m);
        }
        offGraphic.drawString(Integer.toString(startnode.getKey()), LX-a, LY+a);
    }
    update(g);
}

```

Continuous transition means the display of the node's motions which are shown in a smooth way. In contrast to smooth motion, the discrete transition is obtained by an abrupt erase-and-repaint method. This way can be used for the efficient displays for the user. In this system, the insertion motion uses this method; the deletion motion uses smooth motion.

## v.5. Component Design

In this project, we are basically dealing with the tree nodes' movement to achieve the animation. In order to get this goal, several components are designed in the system.

All these components are controlled by button.

**Input Window:** The input window provides input data for the algorithm to manipulate the operations of insertion and deletion. In order for the user to control what data is provided, the input window is designed in a pop up window mode in order for the user to use it easily, and feel in control of the process of algorithm animation. The input window is the important driver that makes the tree work. The user must first input a data then the system will implement its operation. The selection of input data has a great impact on the implementation of tree algorithm animation.

In this system, we use the input window to input a non-negative integer for further use. If the number is entered, you can choose enter or cancel in case something wrong. The following Java code creates input window:

```
class input_window extends Dialog
{
    private TextField data;
    private Button    ok, cancel;
    private int      Inputflag;
    private String    header;
    private Label     message = new Label("Enter an integer for
insertion.");

    public key_in( )
    {
        super(p, true);
        data = new TextField(6);
        ok = new Button("Enter");
        cancel = new Button("Cancel");

        Font f = new Font("TimesRoman", Font.BOLD, 16);
        ok.setFont(f);
        cancel.setFont(f);
        message.setFont(f);
        data.setFont(f);
        ok.setForeground(Color.yellow);
        cancel.setForeground(Color.red);

        add(message);
        add(data);
        add(ok);
        add(cancel);
        setLayout(null);
        message.reshape(20, 30, 280, 60);
        data.reshape(120, 100, 70, 40);
        ok.reshape(50, 160, 60, 40);
        cancel.reshape(160, 160, 60, 40);

        resize(300, 220);
    }
}
```

2.Exit Window: Exit window used to exit from the simulation area. This window includes two buttons. "Exit" and "Cancel".

```
class exit_Window extends Dialog
{
    Button    yes, no;
    public exitWin(String title)
    {
        super(title, true);
    }
}
```

```

yes = new Button("Yes");
no = new Button("No");
Label l=new Label(" Are you really want to exit?");
add(l);
Font f = new Font("TimesRomn",Font.BOLD,18);
Font f1 = new Font("TimesRomn",Font.BOLD,14);
yes.setFont(f1);
no.setFont(f1);
yes.setForeground(Color.yellow);
no.setForeground(Color.red);

add(yes);

add(no);
setLayout(null);

l.reshape(40, 20, 200, 30);
yes.reshape(50, 70, 50, 30);
no.reshape(140, 70, 50, 30);

resize(220,140);
}

```

3. Display Area: This class defines the display area on which the tree is drawn, and how the tree will be drawn. All nodes will be drawn in this field. Following Java code will be define this class:

```

class showarea extends Canvas
{
    private Image    offImage;    // for double buffering use
    private Graphics offGraphic;
    private float    H,V;    // use for the Redblacktree drawing, tell
the horizontal & vertical gaps between the nodes.

public synchronized void paint(Graphics g)
    {
        tree_drawing();
    }

    public void update(Graphics g)
    {
        g.drawImage(offImage,0,0,this);
    }
}

```

3. Main Window: This is the main window class. It includes the control panel, information panel and display panel. It also contains the tree object. It is used as a

container of the display area and control panel. It is a mean for the object communication. The following Java code create this main window:

```

class mWindow extends Frame
{
    private RedBlack P_flag;
    showarea showarea;
    Redblacktree Redblacktree;
    Controlbutton Controlbutton;
    infoarea infoarea;

    public mWindow(RedBlack p)
    {
        P_flag = p;
        Redblacktree = new Redblacktree(this);
        showarea = new showarea(this);
        Controlbutton = new Controlbutton(this);
        infoarea = new infoarea(this);

        add("North",Controlbutton);
        add("Center",showarea);
        add("South",infoarea);
    }

    public void dispose()
    {
        P_flag.start.enable();
        Controlbutton.DisposeWin();
        super.dispose();
    }

    public void ClearTree()
    {
        Redblacktree = new Redblacktree(this);
        showarea.tree_drawing();
        infoarea.setmsg("");
    }
}

```

5. Help Window: This kind of window used to present the help information to help the user to get some useful knowledge for algorithm or how to use this system. The help window of the system gives the user a quick reference to the usage or purpose of every function in this system. The following Java code defined the help window.

```

class HelpDlg extends Dialog
{
    mWindow P_flag;
    Button yes,Exit;
    Label message = new Label(" THE FUNCTION OF VARIOUS
BUTTONS:");
}

```

```

Label      11=new Label("INSERT:      Open input window to enter a
value for inseration.");
Label      12=new Label("DELETE:      Open input window to enter a
value for deletion.");
Label      13=new Label("NEXT/RB/ROT: Implementing actual ins/del,
color flip and Rotation.");
Label      14=new Label("CLEAR:       Clear the exist treetree .");
Label      15=new Label("EXIT:       Exit the window.");
Label      16=new Label("HELP:       Open Help window to enter a
value.");

```

```

public HelpDlg(mWindow p,String title)
{
    super(p,title,true);
    P_flag = p;

    yes = new Button("Yes");
    Exit= new Button("Exit");

    Font f = new Font("TimesRomn",Font.BOLD,16);
    Font f1= new Font("TimesRomn",Font.BOLD,20);
    Exit.setFont(f);
    message.setFont(f1);
    message.setForeground(Color.red);
    yes.setForeground(Color.red);
    Exit.setForeground(Color.red);
    setBackground(Color.yellow);

    add(message);
    add(11);
    add(12);
    add(13);
    add(14);
    add(15);
    add(16);
    add(Exit);
    setLayout(null);
    Exit.reshape(250, 300, 70, 40);
    message.reshape(10, 30, 600, 20);
    11.reshape(10,70, 600, 20);
    12.reshape(10,100, 550, 20);
    13.reshape(10,130, 550, 20);
    14.reshape(10,160, 550, 20);
    15.reshape(10,190, 550, 20);
    16.reshape(10,220, 550, 20);
    11.setFont(f);
    12.setFont(f);
    13.setFont(f);
    14.setFont(f);
    15.setFont(f);
    16.setFont(f);
    resize(600, 400);
}

```

## v.6. User Interface Design

A graphical user interface (GUI) presents a pictorial interface to a program. A GUI gives a program a distinctive 'look' and 'feel'. The user interface design for this

system is based on Java .awt (Abstract Window Toolkit) package. In order to effectively use GUI components, the awt inheritance hierarchy includes component class and container class. Much of each component's functionality is derived from one or both of these classes ( Fig. 13)

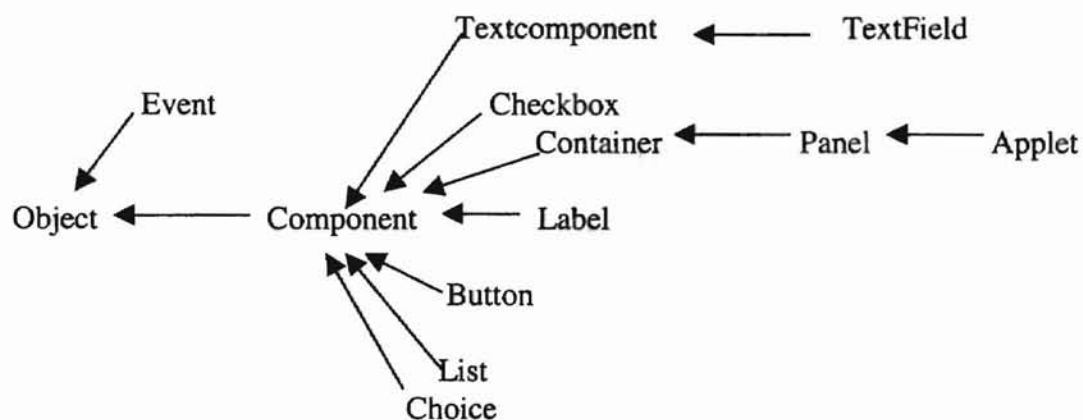


Fig. 13 A portion of the Java .awt inheritance hierarchy.

The Graphical User Interface of this system uses direct manipulation and uses buttons to control the operations. The advantage of this user system is obvious: users who are in command of the system can use it easily; User can get immediate feedback and the time for user to learn to implement this system is short. There are two main window in the designed system: one is the main window refereed above, it is derived from Java Frame; The other is a starting window, it a derived from Java applet. From here, user can enter the main window to operate the red-black tree algorithms , enter help window or play sound. The designed starting window and Java code is as below:

```

public class RedBlack extends Java.applet.Applet
{
    public AudioClip sound;

    Button start, help, play;
    private Image treeimage, oimage, buf1, buf2;
    private Graphics gContext1, gContext2;
    mWindow main1;
    Font font1, font2, font3, x;
  
```

```
helpWin f;
```

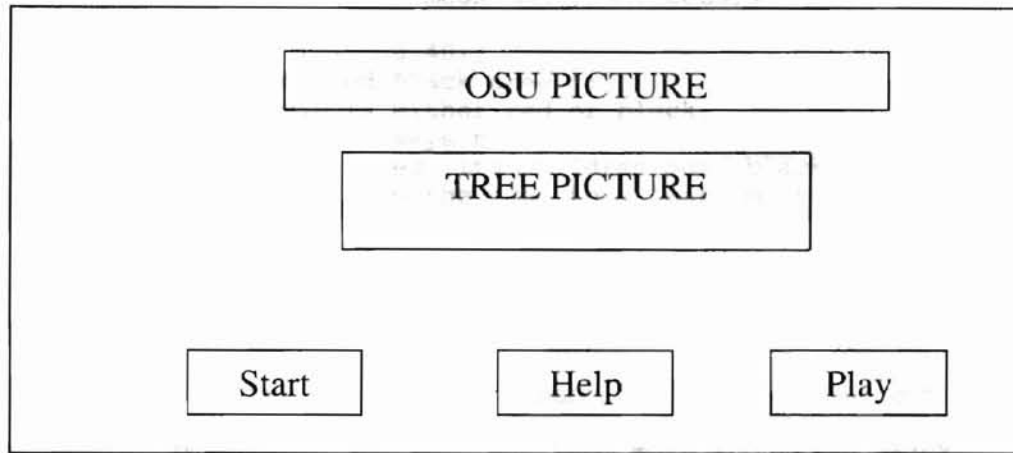


Fig. 14 The designed starting window. Start Button—Start into the main window. Help Button—Enter the help window; Play Button—Play sound.

```
private Label l1, l2, l3, l4, l5, l6;  
private String s1, s2, s3, s4, s5, s6;  
  
public void init ()  
{  
  
    sound = getAudioClip(getDocumentBase(), "s.AU");  
    sound.play();  
    treeimage= getImage(getDocumentBase(), "fig157.gif");  
    oimage= getImage(getDocumentBase(), "o.jpg");  
    buf1=createImage(700, 500);  
    buf2=createImage(40,1);  
    gContext1=buf1.getGraphics();  
    gContext1.setColor(Color.green);  
    gContext1.fillRect(0,0, 700, 500);  
    gContext2=buf2.getGraphics();  
    gContext2.setColor(Color.green);  
    gContext2.fillRect(0, 2, 600, 100);  
  
    font1=new Font("TimesRoman", Font.BOLD, 24);  
    x=new Font("TimesRoman", Font.BOLD, 14);  
    setBackground(Color.green);  
    start = new Button("Start");  
    start.setForeground(Color.black);  
    start.setBackground(Color.red);  
    start.setFont(new Font("TimesRoman",Font.BOLD,30));  
    setLayout(null);  
    add(start);  
    start.reshape(200, 435,80,40);  
    help = new Button("Help");  
    help.setForeground(Color.black);  
    help.setBackground(Color.red);  
    help.setFont(new Font("TimesRoman",Font.BOLD,30));  
    add(help);  
    help.reshape(340, 435,80,40);  
    play = new Button("Play");
```

```

        play.setForeground(Color.black);
        play.setBackground(Color.red);
        play.setFont(new Font("TimesRoman", Font.BOLD, 30));
        add(play);
        play.reshape(480, 435, 80, 40);
        s1= " The Rules of Red Black tree:";
        s2= " 1. Every node is either red or black.";
        s3= " 2. The Root is always black.";
        s4= " 3. If a node is red, its children must black.";
        s5= " 4. Every path from the root to a leaf, must";
        s6= "    contain the same number of black node.";

    }

    public void start (Graphics g)
    {
        gContext1.drawString("Welcome to Redblack tree simulation", 160,
100);
        gContext1.drawString("If you want know more about red black tree,
press Help.", 100, 400);
        gContext1.drawString("If you want to enter the simulation, press
Start", 100, 520);
        /*gContext1.drawImage(treeimage, 140, 120, this);
        gContext1.drawImage(oimage, 80, 1, this);*/

    }

    public void paint(Graphics g){
        g.drawImage(buf1, 0,0, this);
        gContext1.fillRect(0,0, 700, 500);
        gContext1.drawImage(treeimage, 140, 120, this);
        gContext1.drawImage(oimage, 80, 1, this);

    }

    public boolean action(Event e, Object o)
    {
        Object target = e.target;

        if (target == start)
        {
            main1 = new mWindow(this);
            main1.resize(800,600);
            main1.setResizable(false);
            main1.show();
            start.disable();
            sound.play();
        }
        if (target == help){
            if (f != null){
                f.hide();
                f.dispose();
            }

            f= new helpWin(" Red black Redblacktree help ");
            l1= new Label(s1);
            l1.setFont(font1);
            l2= new Label(s2);
            l2.setFont(x);
        }
    }

```



```

        l3= new Label(s3);
        l3.setFont(x);
        l4= new Label(s4);
        l4.setFont(x);
        l5= new Label(s5);
        l5.setFont(x);
        l6= new Label(s6);
        l6.setFont(x);

        f.setLayout( new GridLayout(8, 1));

        f.add(l1);
        f.add(l2);
        f.add(l3);
        f.add(l4);
        f.add(l5);
        f.add(l6);
        f.resize(400, 300);
        f.show();
    }

    return true;
}
}

```

The major graphical user interface contents are included in the main window. ( Fig. 15). It includes various buttons. Through these buttons, the user can control the animation and implement the red-black tree algorithms.

- **Help Button:** This button is used to open the help window. (Fig. 16). This window will show the various functions of different buttons in main window.
- **Input Button:** This button is used to open the input window. Through this window, the user can input a non-negative integer for insertion or deletion into/from the tree. Fig. 17 shows the designed input window.
- **Demo:** if the user push this button, a red-black tree will constructed automatically. Through this button, user will have a direct knowledge what the red-black tree looks like and how the insertion and deletion algorithm implementation. In the program,

we use a array to contain a series integer, and use a for loop to input this integer continuously.

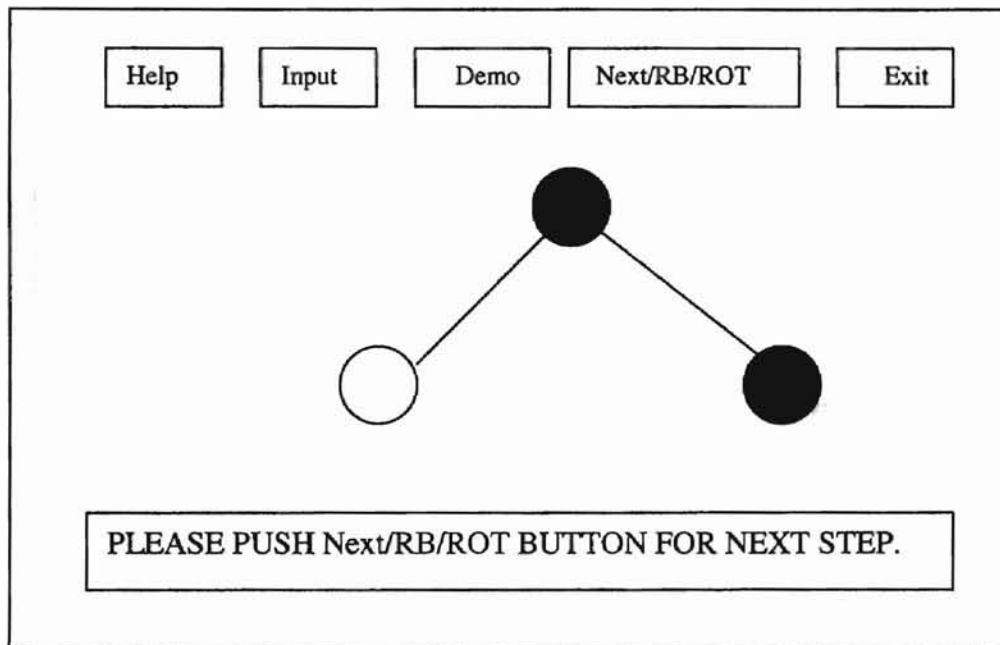


Fig. 15 *The designed main window.*

```
private void Demobutton()  
{  
    int c[] = {50,45,32,23,76,97,88,11, 64,45,45,65, 47,3,32,45};  
  
    for(int i=0; i<c.length; i++){  
        if((i==9) || (i==14) || (i==15))  
            P_flag.Redblacktree.Delete(c[i]);  
        else  
            P_flag.Redblacktree.insert(c[i]);  
    }  
}
```

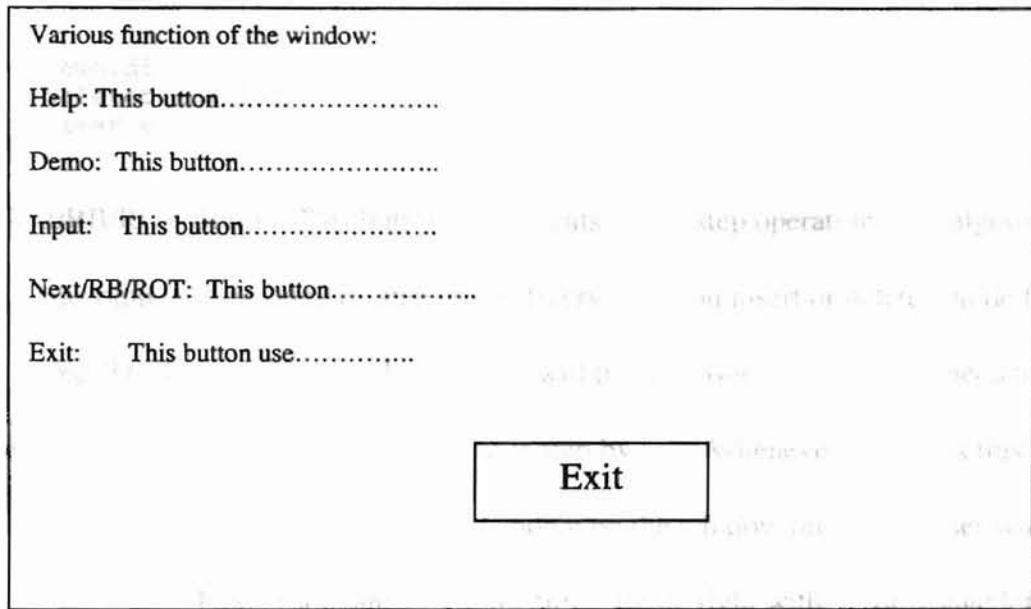


Fig. 16 *The designed help window in main window. Exit button—Exit from this window.*

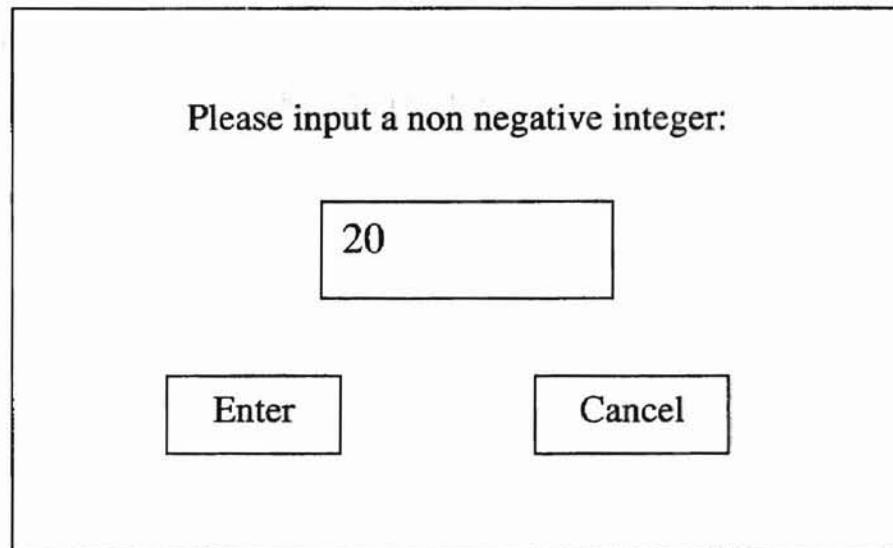


Fig. 17 *The designed input window. 20 – input integer; Enter button – enter the value; Cancel—cancel the key-in value.*

```
demo.disable();
deletet.enable();
clear.enable();
}
```

- **Next/RB/ROT Button:** This button implements 'next' step operation. The algorithm was decomposed into step by step mode. Every time you insert or delete a node from the tree. The balance of the red-black tree will be destroyed. The tree will need to restore this balance. This restore course is step by step. Whenever you press this button for next step, a sentence will be shown on the window prompting user what the next step will be , for example: 'color flip', 'rotate right with...' or 'rotate left with...'.
- **Exit Button:** This button used to exit from the main window and return to the starting window. Because the main window is a Java frame, it need a specified procedure to exit (the starting window is a Java applet, it include exit function.). Fig. 19 showed the designed exiting window.

**ACTION NOTE:**

NODE 5 NEED ROTATE LEFT WITH NODE 10

Fig. 18 *Designed text field in the main field.*

- **Action note Field:** This is a Java Text Field, in this field there will be a sentence showing what the next step will be. (Fig. 18).

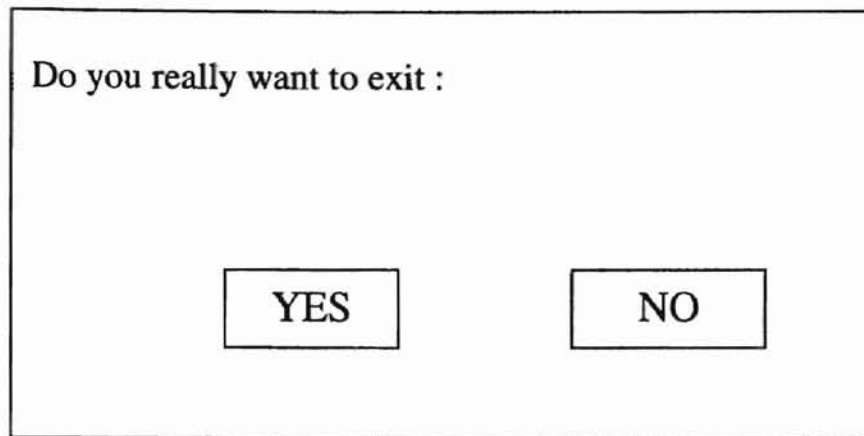


Fig. 19. *Designed Exiting window.*

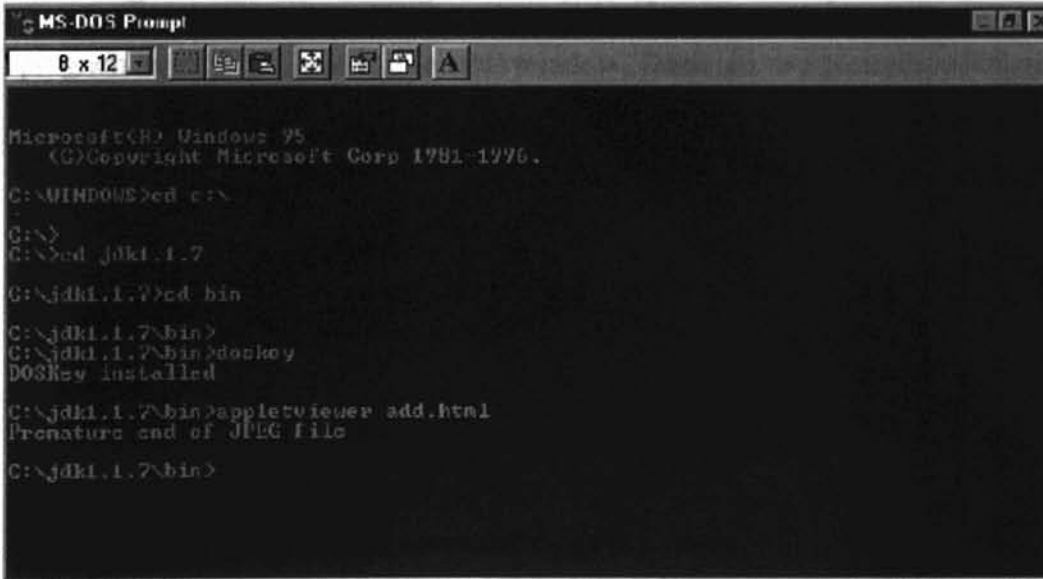
### V.7. Sound Display

Java program can manipulate and play audio clips. It is easy for users to capture their own audio clips. In this system, we used sound to explain some important features of this design and add some background sound. At present time the ability to load and use sound of Java is quite limited--- you can only load sounds in the .au format. We use Cool Edit 96 to make the sound clips. Cool Edit 96 (Cool Edit™ 96 ) is a digital audio editor for Windows 95 and Windows NT.

## VI. SYSTEM OVERVIEW

In last chapter, we described the designing details of this red-black tree animation system. In this chapter we give some snapshots from this system running in Java Developing Kit 1.1.7. These snapshot will show you what the running results are and how the visualization pictures look like.

The designed system can running under all kinds of Sun Java Development Kit and Microsoft Visual J++ environment. We implement it in JDK 1.1.7 under DOS system. Fig. 20 shows the DOS command of this program. Fig. 21, Fig. 23 shows the two main window structure of the system. Fig. 21 showed the starting window, it includes three buttons. If we press the 'Start' button, the main window will be open. It includes most GUI components and user can implement red-black tree algorithms.



```
MS-DOS Prompt
8 x 12
Microsoft(R) Windows 95
(C)Copyright Microsoft Corp 1981-1996.
C:\WINDOWS>cd c:\
C:\>
C:\>cd jdk1.1.7
C:\jdk1.1.7>cd bin
C:\jdk1.1.7\bin>
C:\jdk1.1.7\bin>doskey
DOSkey installed
C:\jdk1.1.7\bin>appletviewer add.html
Premature end of JPEG file
C:\jdk1.1.7\bin>
```

Fig. 20 The DOS command to run this system.

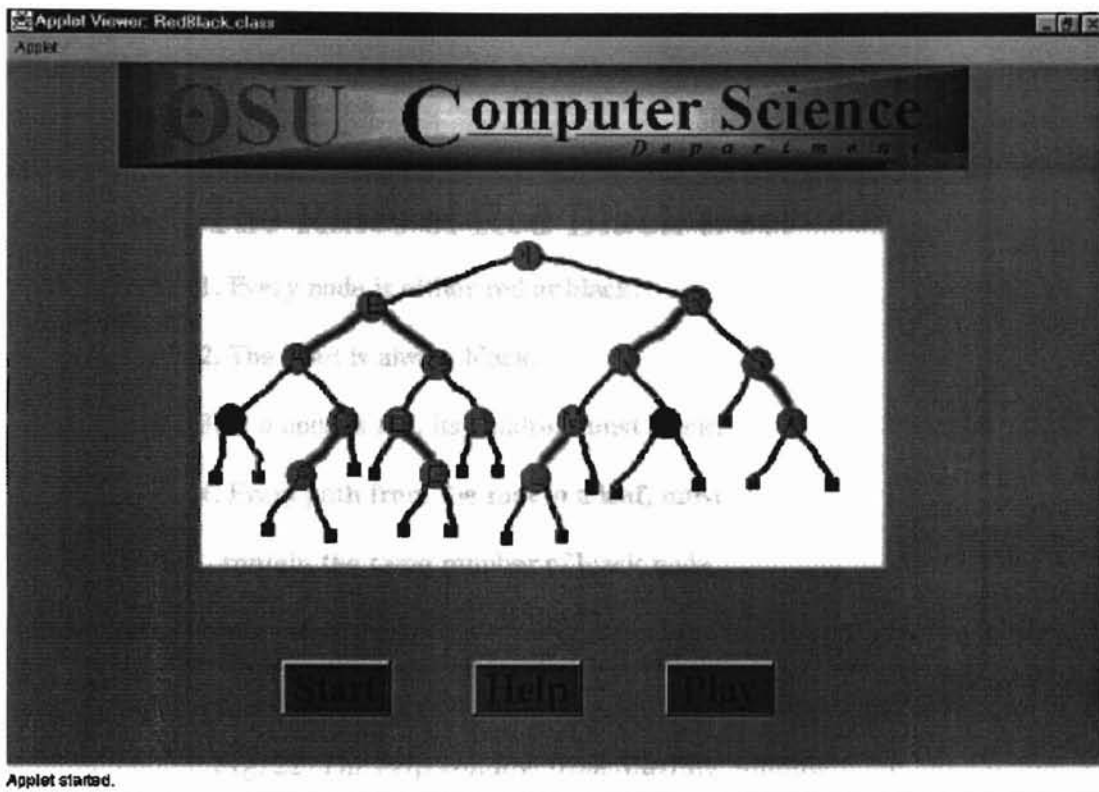


Fig. 21 The starting window of the system. *Start Button* – enter the main Window; *Help Button* – open the help window; *Play Button* – sound display.

Fig. 21 shows the starting window of the system. When you first run this Java program in jdk 1.1.7, you will see this window. There are two pictures and three buttons on it. If 'Help' button is pushed, user will see Fig. 22, it will show the four rules of red-black tree for the user; 'Play' button is used for sound display. If you push this button, there will be a sound to explain basic knowledge of red-black tree and how to use this system. Sound is an important part of this system, it can attract the user's attention; If user push 'Start' button, the main window will be open and user can began his exercise about red-black tree algorithms.

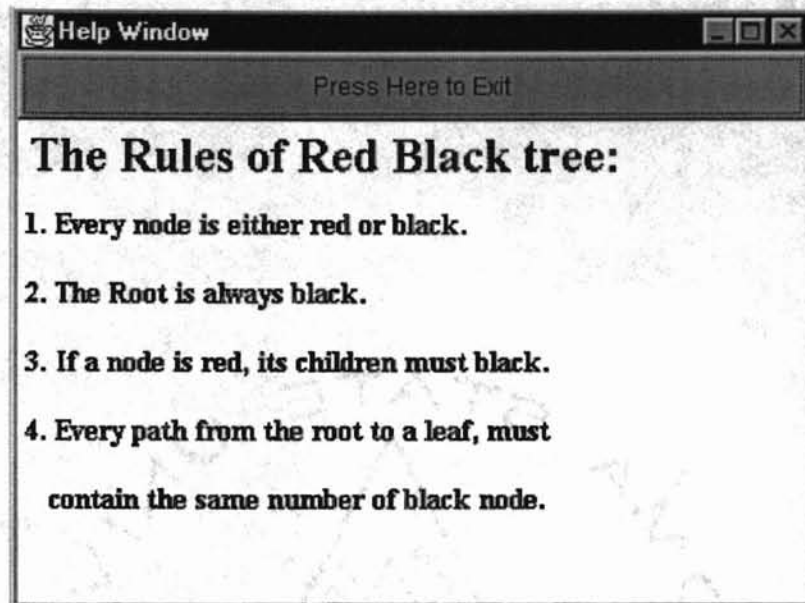


Fig. 22 The help window from Starting window.

Fig. 23 shows the main window of the system. Whenever the 'Start' button on the starting window is pushed, the system will enter this window. All the interactive buttons about the tree algorithms are in this window. If you push these buttons, there will be various events happen.

Fig. 24 shows when the help button is pushed during tree's implementation, there will be a brief explanation of every button function's usage and purpose. There are seven buttons for the tree implementation in this system, they include the basic functions like insertion, deletion and some additional functions such as demo, clear, exit, etc. When the basic functions are chosen, the input data must also be chosen to make those basic functions operate.



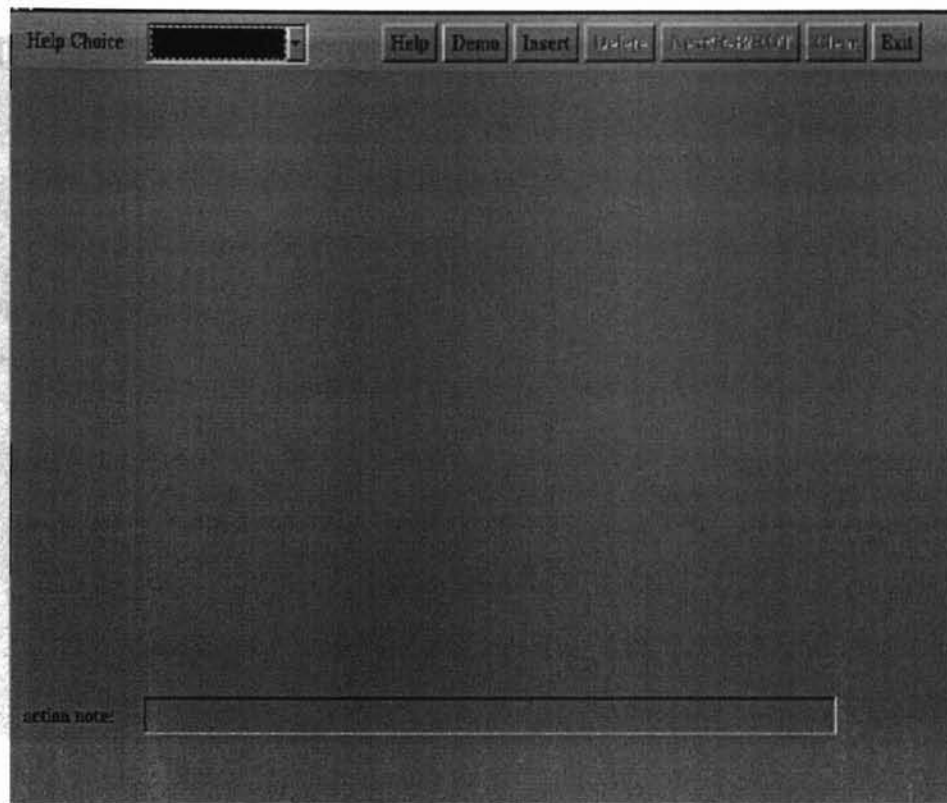


Fig. 23. *The main window of the system. Help button – open the help window; Exit button – open the exit window; Insert or delete button – open the window for insert/delete; NEXT/RB/ROT – next step; Demo button – implement a demo tree; Clear button – clear the exist tree from the screen.*

Fig. 25 shows that when the demo button is pushed during tree implementation, there will a continuous operations to construct a red-black tree ( include insertion and deletion). This function is used to demonstrate every kind of operations and templates to the users who have no direct knowledge of red-black tree.

If the user press insertion or deletion button, the input window (Fig. 26) will show up, and you can enter a non negative integer into the system for operation through the key board. There are two buttons in this window: “enter”— enter the integer into the system; “cancel” – cancel the input integer for some reasons.

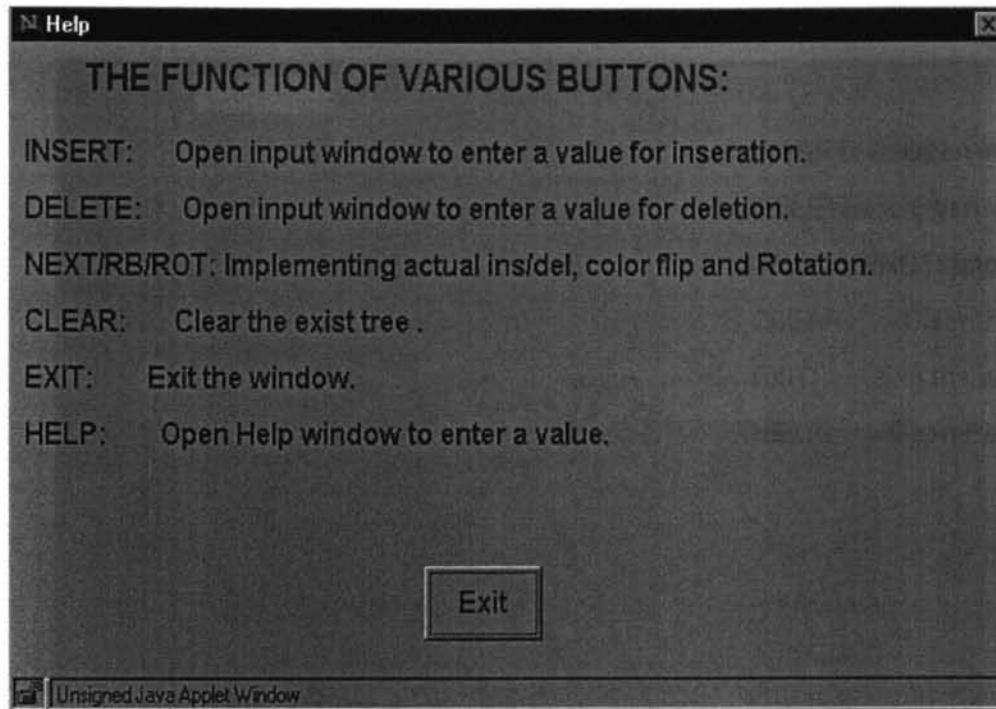


Fig. 24 *The help window in main window.*

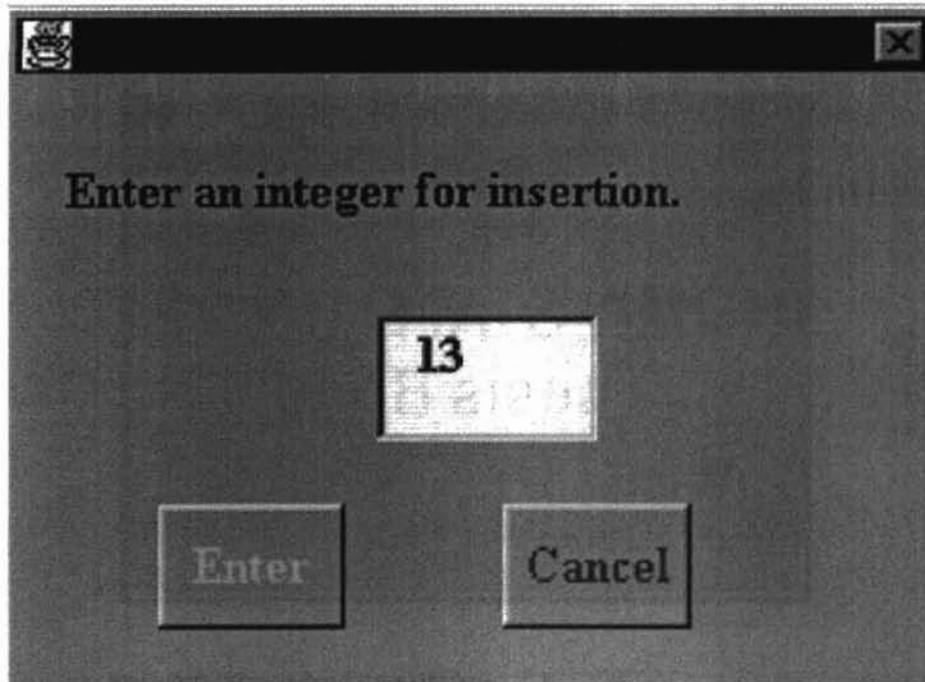


Fig. 26 *The input window.*

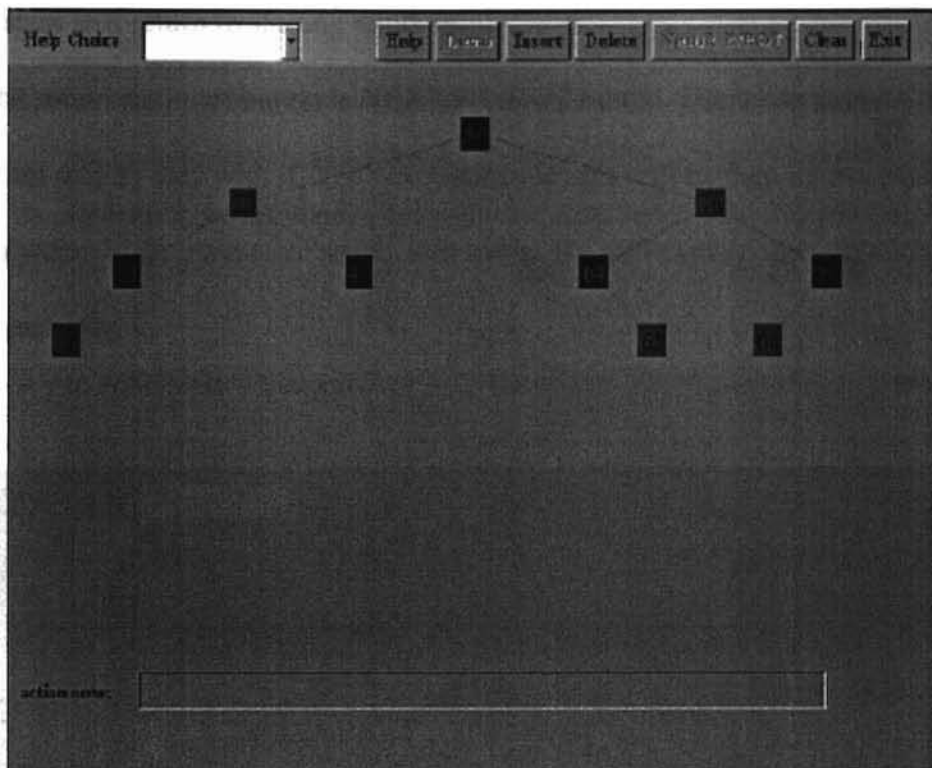


Fig. 25 The scene of the 'Demo' button was pressed in main window.

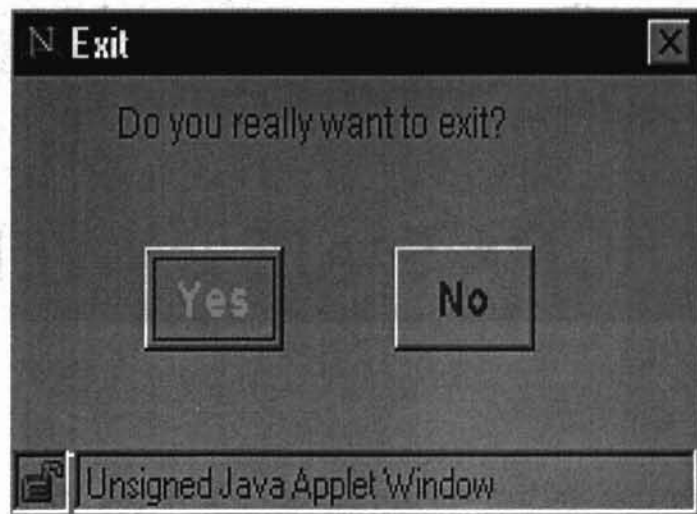


Fig. 27 The exiting window.

Fig. 27 shows the exit window. If you push exit button this window will show up and system will return to starting window.

The most important button is NEXT/RB/ROT button. Users can operate the tree algorithm step by step with it. The next series pictures will explain its usage and how the system works in an interactive mode with users. We will explain the operation procedure under each step.

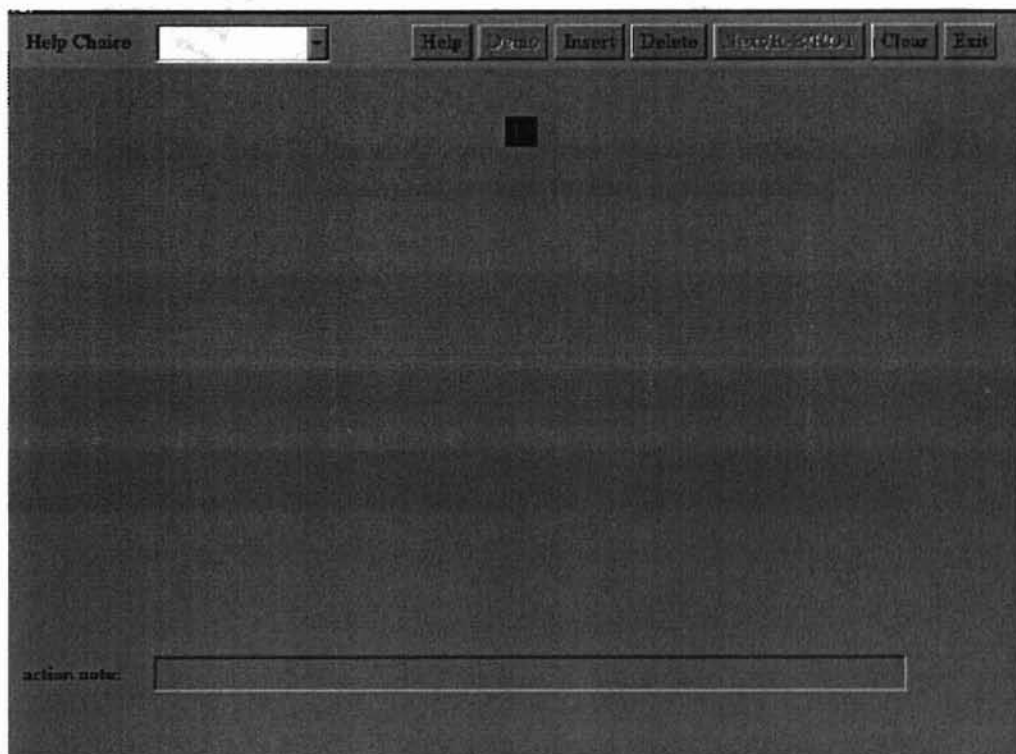


Fig. 28 Step 1: The node 13 was inserted into the tree. (first node).

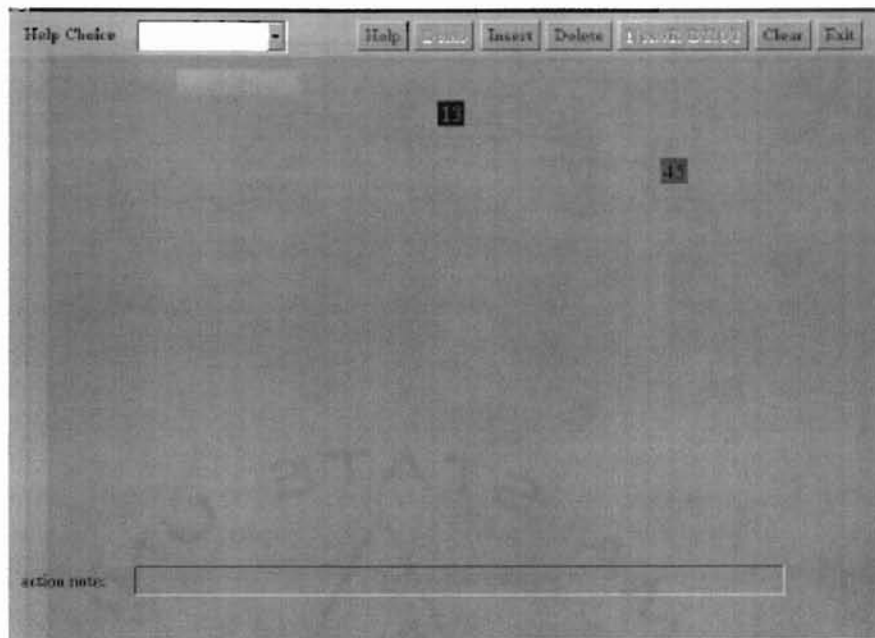


Fig. 29 Step 2: Insert 45 into the tree. the node initialize is red. The tree is Balanced and no further action needed.

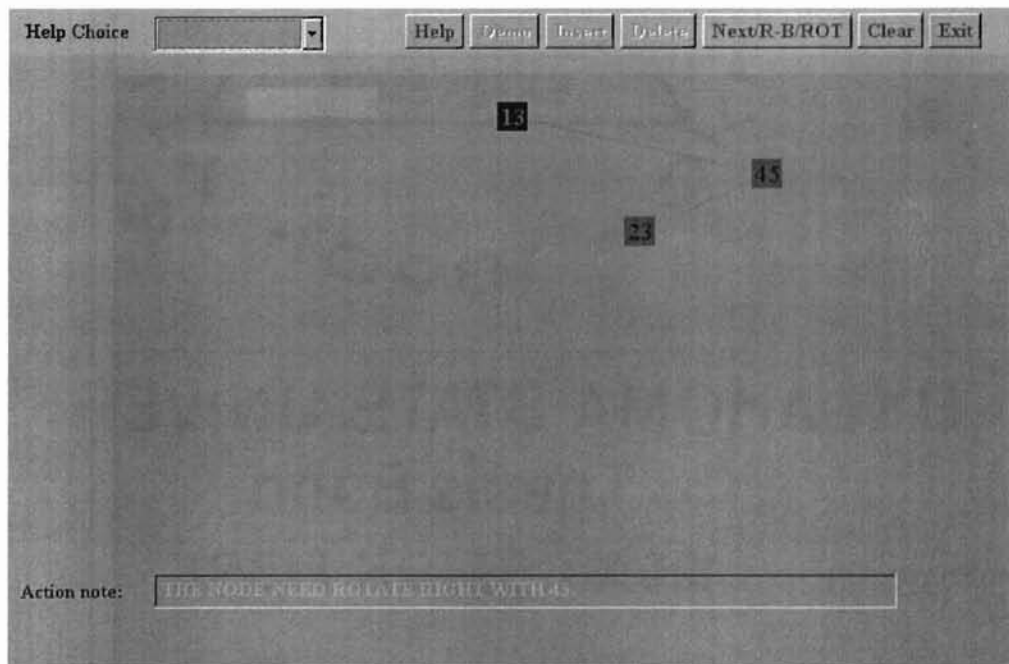


Fig. 30 Step 3: Insert 23. The tree is unbalanced now. The action note showed 'THE NODE NEED ROTATE WITH 45'. At that time, the user need to press the 'NEXT/RB/ROT' button for next step according to the red-black tree algorithm.

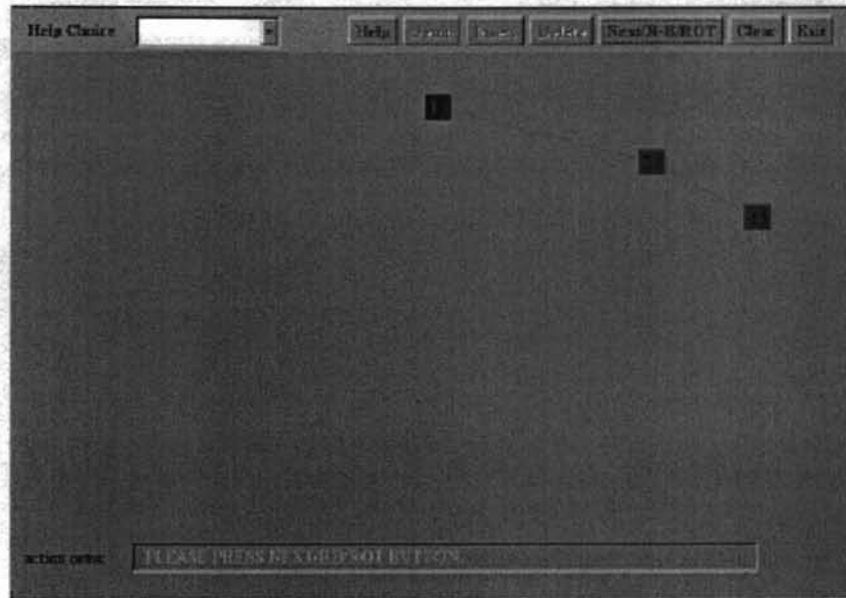


Fig. 31 *Step 4: After press the 'NEXT..' button the node will rotate with 45. But the tree still unbalanced. We still need to press this button.*

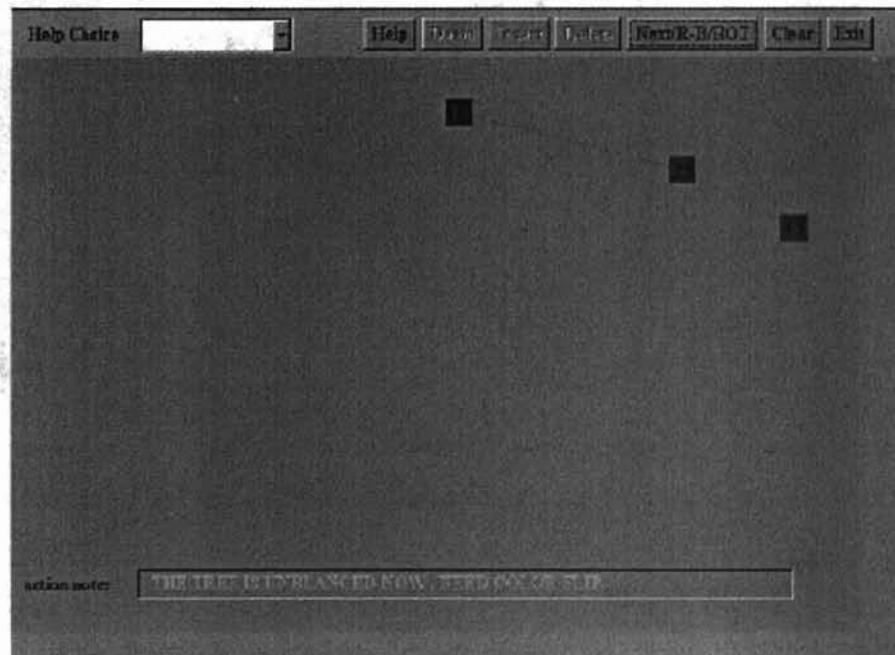


Fig. 32 *Step5: The action note shows need color flip. So press "NEXT.." button.*

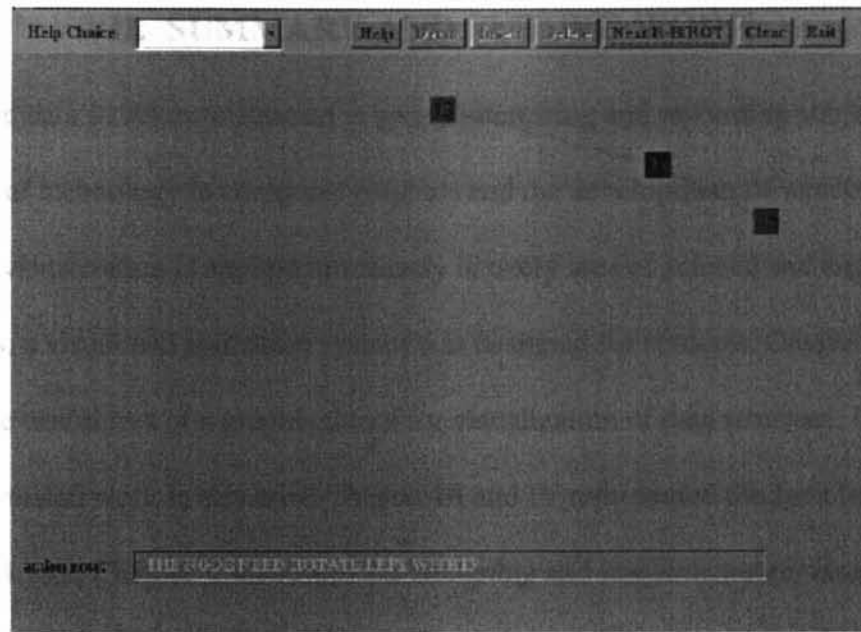


Fig. 33 Step 6 : After color flip. The tree still unbalanced. The action showed " THE NODE NEED ROTATE LEFT WITH 13", so we need press 'NEXT button' again.

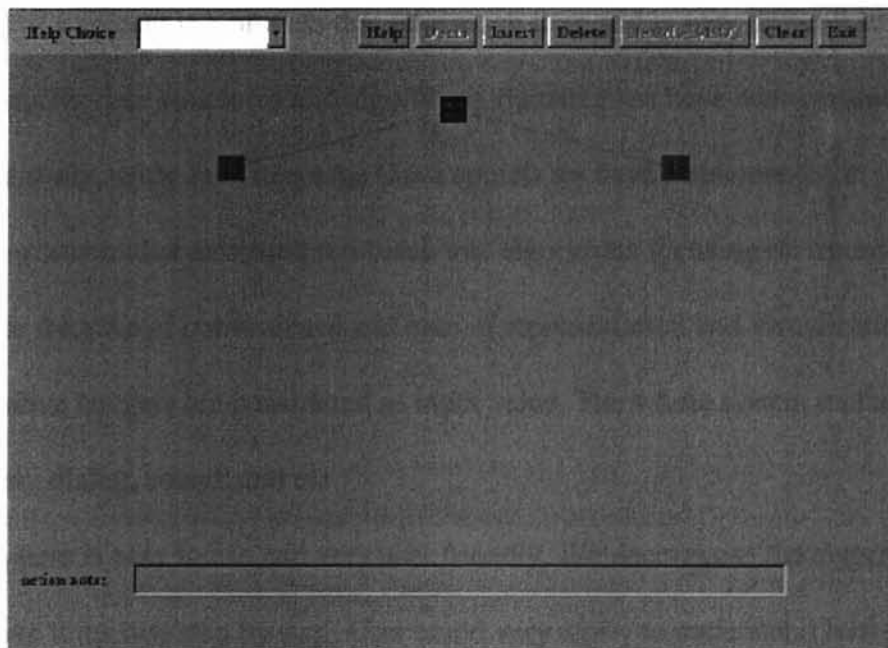


Fig. 34 Step 7: Finally the tree was balanced. The action note showed nothing.

## VII. SUMMARY AND FUTURE WORK<sub>n of red black tree</sub>

Dynamic data structure animation is a very interesting and rewarding subject. Due to the advance of technology in computer graphics and the development of windowing technology, visualization is applied immensely in every area of science and engineering. In this thesis, a visualized animation system was designed for students. Chapter I discussed the usefulness of a graphical tool for visualization of data structure; Chapter II showed the related work in this area; Chapter III and IV represented the Java language and red-black tree; Chapter V discussed the designing and implementation issue; Chapter VI overview the designed system and explained how to use this tool.

The algorithm animation system will make data structure program easy to understand. In view of the fact that teaching and learning of data structures and algorithms for a student is a process that take much time and is very difficult sometimes, many systems for data structures and algorithms visualization have been created.

In this study, using Java language (Java applet) we have implemented a visualization system that animated red-black tree algorithms focusing on insertion and deletion. For the sake of convenience and ease of representation and visualization, only the nonnegative integers are considered as input value. The whole system includes Java applet, frame, dialog, sound, and etc.

The system is easy to use and very user friendly. We decompose the algorithms in order to make it operate step by step. User could very easily to understand how the algorithm works and how the red-black tree structure changes during the insertion or deletion operation.



Due to the limited of time, this system focused on the visualization of red-black tree. There are still many kinds of tree based algorithms that can be visualized such as: AVL tree, B tree, Splay tree, and etc. We can put all these data structure's visualization in one system. In this design, we make choice list for further use. We can very easily add new contents of other tree algorithms. Visualizing other algorithms in this system are considered future work.

## VIII. REFERENCES

- Venners, B., *Inside the Java Virtual Machine*, McGraw-Hill, New York, 1998.
- Cormen, H. T., Leiserson, E. C., Rivest L. R., *Introduction To Algorithms*. The MIT Press, Cambridge, Massachusettes , 1997.
- Deitel, H. M. , Deitel P. J. , *Java: How to Program*, Prentice Hall, Upper Saddle River, NJ, 1997.
- Harvick, L. H. , *Rule Based Data Structures Animation*. M. S. Thesis, Dept. Computer Science. Oklahoma State University, 1996.
- Knowlton, K. C. , *L6: Bell Telephone Laboratories Low Level Linked List Language*, Two Black and White Films, Bell Labs, Murray Hill, NJ, 1966.
- Laurence, V. *Master Java 1. 1*, Second Edition, SYBEX, Alameda, California, 1997.
- Lin, B. H. , *An Object-Oriented Graphic User Interface for Visualization of B-Trees' Animator*. M. S. Thesis, Dept. Computer Science. Oklahoma State University, 1997.
- Nataraj, N., Arvind S., *Java Networking and AWT API Superbible*. Waite Group Press, Corte Madera, California, 1996.
- Schroeder, W, Lorensen. B., *Visualization Toolkit*, Simon & Schuster, Upper Saddle River, New Jersey, 1996.
- Shen, H. , *A Visual Aid for the Language of Tree Based Data Structures*. M. S. Thesis, Dept. Computer Science. Oklahoma State University, 1994.
- Stasko, J. T. , *A Practical Animation Language for Software Development*. *Proc. of*

*IEEE 1990 Intl Conf. On Computer Languages.* 8, 1990 , 1-10.

Thomas, M. G. , *PROVIDE: A Process Visualization and Debugging Environment Technical Report*, Department of Computer Science, University of Illinois at Chicago, Chicago, IL, July 1985.

Weiss, M. A. , *Data Structure and Algorithm Analysis In C* , Addison-Wesley, Menlo Park , CA. 1997.

Weiss, M. A. , *Data Structure and Algorithm Analysis In C++* , Addison-Wesley, Menlo Park , CA. 1997.

Xu, C. , *Multimedia Visualization of Abstract Data Type*, M. S. Thesis, Dept. Computer Science. Oklahoma State University, 1997.

Yarwood, E. , *Towards Program Illustration.* M. S. Thesis, Department of Computer Science, University of Toronto, Toronto, ON, 1974.

VITA

Pengcheng chen

Candidate for the Degree of

Master of Science

Thesis: RED-BLACK TREE ALGORITHM ANIMATION USING JAVA

Major Field: Computer Science

Biographical:

Personal Date: Born in Yantai, Shandong, P. R. China, Married to Jing Yu.

Education: Received Bachelor of Science degree in Biology from East China Normal University, Shanghai, P. R. China, in July 1986; received Master of Science degree in Biology from East China Normal University, Shanghai, P. R. China, in July 1989. Completed the requirements for the Master of Science degree with a major in computer Science at Oklahoma State University in July, 1999.

Experience: Employed by Environment Protection Agency of Shandong Province, Jinan, P. R. China from September 1986 to August 1992; employed by the Shanghai Fishery University, P. R. China, assistant professor from July 1995 to November 1996.