# A VISUAL AID FOR THE LEARNING OF TREE-BASED DATA STRUCTURES

BY

HUNG-CHE SHEN

Bachelor of Science

Feng Chia University

Taichung, Taiwan R.O.C.

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
May, 1994

A VISUAL AID FOR THE LEARNING OF

TREE-BASED DATA STRUCTURES

Thesis Approved:

[illegible]

Thesis Advisor

J Chandler

[illegible]

Thomas C. Collins

Dean of the Graduate College

## ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to my major advisor Dr. K. M. George for his thorough guidance and helpful advisement throughout my graduate study and writing of this thesis. Without him, the fulfillment of this thesis will be impossible.

I am grateful to Dr. J. P. Chandler for serving on my committee. He gave me invaluable suggestions and directions about the programs in this thesis. I am also grateful to other committee member, Dr. M. Neilsen, for his advisement during the course of this work.

My deepest appreciation is extended to my father, Zenyu and my mother, Chufong for their encouragement, love, and support for my graduate study here.

TABLE OF CONTENTS

## LIST OF TABLES

LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

House [10] explains the importance of graphics: "Graphics have been used for centuries to effectively communicate information among people and aid comprehension of complex information. Because patterns and shapes are inherently less abstract than numbers and languages" (p. 29). Furthermore, it is common knowledge that pictures convey information more readily and permit better retention than textual or verbal representations of the same information. And that is why graphics can have such an immense application. We can easily find graphics applications in every aspect of our daily lives.

During the past 20 years, with hardware and software techniques for creating and manipulating graphics displays advancing, computers have revolutionized their ways in creating graphics that make the applications of graphics much more efficient and widespread. Nowadays computer graphics have become one of the most exciting and rapidly growing fields in computer science. Computer graphics is a very active and rapidly changing technology. This field has greatly affected such diverse areas as business, military, industry, science, entertainment, education, and research than ever before.

The extreme value of computer graphics applications in education is easily seen by the fact pointed out by Davis [4]:

Students fooling around with ... orbits in the many-body problem that have been graphically displayed have found periodic solutions whose existence defies our keenest analytical analysis.

House [10] mentions one of the favorable merits of graphics applications in education in term of human brain. The perceptual work is done by the right hemisphere of human brain that initially processes pictorial information. Then the left hemisphere of human brain does the analysis work. If we use more pictures than words, more of the brain will be involved in processing pictorial information than in processing numerical or textual information. This results in the increased understanding, and consequently better recall, of pictorial information.

Knowing of those benefits that graphics can provide arouses our motivations of this research in using computer graphics as an aid for learning of data structures. We all know that the teaching learning process of data structures is not an easy job. If computer-assisted instruction is available, it would complement the traditional lecture course in data structures.

The primary purpose of this thesis is to devise a method of using computer graphics for helping students learning tree-based data structures. We call it "Tree-based

Data Structures Visualization" system, or TBDSV system.

Then what is visualization? Gershon [7] states that "Visualization is the process of transforming information into a visual form, enabling users to observe the information." The term "Visualization" with its basis in computer graphics is still quite a new word. As for the pioneering work of visualization done successfully in some scientific disciplines such as molecular modeling and medical imaging, the word "visualization" has gained rapid and widespread acceptance.

The primary goal of this TBDSV system is to explore how this visual aid in computer can be used to promote the learning interests and learning efficiency of data structures by providing an innovative medium of communication. In this TBDSV system, students can choose from a set of tree-based algorithms. Then they can observe and explore the dynamic behavior of data structures through graphical displays. In addition to showing the dynamic behaviors of algorithms, it also illustrates the logical organizations of data structures. Rather than using pens or drawing charts on papers, this learning method makes a fundamental improvement possible in the way we understand and think about algorithms and data structures. Besides the applications to learning and instruction, the "Data Structure Visualization" system can be used as a tool for a beginner who wants to write programs of those tree-based data structures. By observing graphical simulations of the trees' behaviors, the user can get the idea of the inner workings of those tree-based programs.

This visualization system is developed on the X Windows system, because X Windows provide a set of tools to build graphical objects. These tools are available in Oklahoma State University Computer Science Department's laboratory that is equipped with NCD X-terminals that have graphics displays connected by Ethernet.Furthermore, one important advantage of using the X Windows system lies in the fact that it provides a smooth and flexible open system user interface at a time when users are growing increasingly accustomed to window-style interfaces.

We have visualized typical data structures, namely AVL tree, Red-Black tree, B-tree, and Splay tree in our TBDSV system. These trees are important data structures that appear often in courses of data structures, file structures, etc.

The remaining chapters of this thesis are arranged as follows:

Chapter II describes related work, and illustrates the areas that this study needs. Chapter III provides an abstract system model which is used as a method of requirements derivation based on examining the system from several different viewpoints. Chapter IV is on software design. This chapter covers the software design process, design strategies and design quality. The implementation of TBDSV system and the outcome of the implementation are given in

Chapter V. Finally, summary and future work for this thesis are disscussed in Chapter VI.

# CHAPTER II

# LITERATURE REVIEW

The development process of "Algorithm Visualization" requires concepts from several areas. The areas closely related to this thesis are (1) algorithm design, (2) visualization design, and (3) user interface. In the following sections, works related to this thesis are reviewed.

## Visualization Design

Gershon [7] stated that "if one could use the flexibility of display devices to feed information through preattentive visual processes, one would enable the user to perceive the desired information efficiently and fast. Methods are based on the sensitivity of the human visual system to motion and the ease at which electronic display devices could change their display." Thus, in the visualization design, we must first take the human visual perception areas into consideration.

In order to develop the part of visualization with formal models and precise semantics, we divide it into two components. They are static displays and dynamic displays: Static displays show the image of data and their relationship, and overall structures, etc.

Dynamic displays show the behaviors of algorithms and indicate the seguences inside the codes of the algorithms.

Tree-based graphs are static structures. The difficulties of showing the tree-based graphs lie in the fact that the growing of tree can easily exceed the boundary of window. Therefore, in the graph layout algorithms, We focus upon the aspects Eades [5] has listed:

.Maximize display symmetry
.Avoid edge crossings
.Avoid bends in edges
.Keep edge lengths uniform
.Distribute vertices uniformly

There are three visualization systems listed below, which are described in the literature. They are being categorized as static because they show more of the relationships and orders among data items than of the dynamic behavior of programs.

The first system is a program of addressing Linked-List Visualization for Debugging. This system is called VIPS (Visualization and Interactive Programming Support debugging system). VIPS uses UNIX's symbolic debugger, DBX, to execute the program to be debugged. What makes this debugging tool different is that it displays linked list as syntax trees. In improving VIPS's ability to visualize linked lists, Shimomura [18] and Isoda [18] considered four requirements that VIPS should meet: 1. Easy shape recognition. 2. Easy change detection. 3. Selective display. 4. Rapid drawing.

The second system is described as "Using visualization tools to understand concurrency" by Zernik [22] et al. They point out that "programming large-scale parallel machines is daunting because parallelism makes program execution much more complex and difficult to understand." In their visualization tool, it uses graphs to provide a logical view of execution. Views are organized according to computational threads, messages, synchronization events, and so on. This tool can be used to overcome the concurrency bugs, giving the user a clear picture of concurrency.

The third is "An implementation of data structures display system" by Lee [11]. The primary functions of this system are:

1. graphically display a variety of data structures,
2. allows the users to execute and study the immediate effects of each step of an operation on a particular data structure.

The available implementations of data structures in this system include B-Tree, binary search tree, and linked-list. Since Lee's data structures display system is shown on VT100-type terminals. Its display types are more rigid compared with our TBDSV system that has graphics displays on X terminal using the X Windows environment.

In the following, we will introduce a more vivid type of display than the static displays mentioned above. It is dynamic displays. Dynamic displays show the graphic objects that change location, size, and figure. This is the area of algorithm animation.

There are a few algorithm animation systems available for education now. The well-known ones are Balsa and Balsa-II. They were used as teaching aids in learning data structures at Brown University. According to Brown [2], "the user can watch execution of an algorithm through various views. Each view is displayed in a window on the screen." Although those systems contain an extensive library of sophisticated animations and have been used for more than seven years, they are not widely used. Their drawbacks are that it requires internal Macintosh coding to create new animation view, and programs being animated must be executed in Macintosh. So the portability is not good enough.

Stasko [20] has designed and implemented a framework and system called "Tango" which facilitates algorithm animation. It is designed to support three programming activities, namely understanding programs, evaluating existing programs, and developing new programs. In his system, Stasko [20] has simplified animation design by developing an algorithm animation framework that is based on four abstract data types: locations, images, paths, and transitions. He has set up a good method for algorithm animation. In his system, to produce an animation, the user must

. annotate the program with the necessary algorithm operation.

. design animation scenes to implement the animation actions.

. create a control file specifying the mapping from the algorithm operations to the animation scenes.

Thus the main application of this system is on debugging the user's algorithms, and designing the algorithms.

## User Interface

The user interface of a system is often the criterion by which that a system is judged. If the user interface is too difficult to use or understand, it may cause this software system to be discarded, no matter how good its functionalities are. Especially for a software to be an aid for the learners, we cannot overemphasize the importance of user interface.

Sommerville [19] formulated many principles that are important for the design of user interface. They are:

(1) The interface should use terms and concepts which are familiar to the anticipated class of users.
(2) The interface should be appropriately consistent.
(3) The user should not be surprised by the system.
(4) The interface should include some mechanism which allows users to recover from their errors.
(5) The interface should incorporate some form of user guidance.

Our system is developed in the X Window environment, that is good for designing a graphical user interface. In developing the graphical user interface, we follow the guidelines listed above.

Marcus et al.[14] introduces an implementation-oriented model of graphical user interface as shown in the figure below. This model assumes that interactive application programs are running under the control of a window algorithm operations to the animation scenes.

Thus the main application of this system is on debugging the user's algorithms, and designing the algorithms.

## User Interface

The user interface of a system is often the criterion by which that a system is judged. If the user interface is too difficult to use or understand, it may cause this software system to be discarded, no matter how good its functionalities are. Especially for a software to be an aid for the learners, we cannot overemphasize the importance of user interface.

Sommerville [19] formulated many principles that are important for the design of user interface. They are:

(1) The interface should use terms and concepts which are familiar to the anticipated class of users.
(2) The interface should be appropriately consistent.
(3) The user should not be surprised by the system.
(4) The interface should include some mechanism which allows users to recover from their errors.
(5) The interface should incorporate some form of user guidance.

Our system is developed in the X Window environment, that is good for designing a graphical user interface. In developing the graphical user interface, we follow the guidelines listed above.

Marcus et al.[14] introduces an implementation-oriented model of graphical user interface as shown in the figure below. This model assumes that interactive application programs are running under the control of a window management system that manages the use of the screen and the input devices.

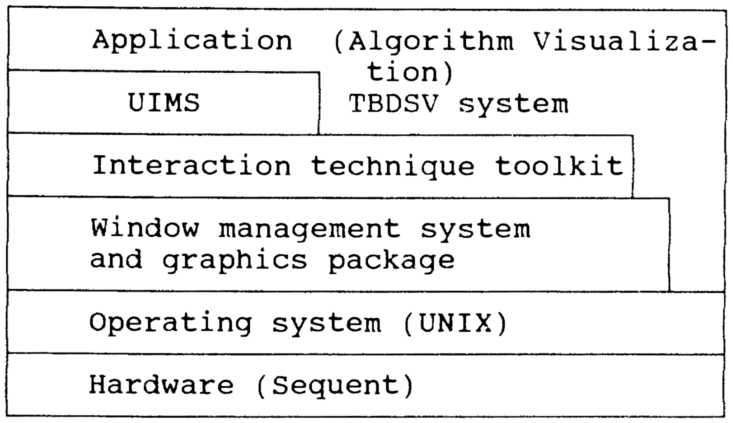

Application (Algorithm Visualization)
UIMS
TBDSV system
Interaction technique toolkit
Window management system and graphics package
Operating system (UNIX)
Hardware (Sequent)

Figure 1. The Architecture of the TBDSV System Running on Its Environment

Retting [17] points out that the process of building an interface involves two steps:

1) Write the User Interface Standards Manual.
2) Design the Interface.

In writing the user interface standards manual, one should have the symbols concepts listed below. Retting [17] states that four aspects of the symbols that make up the interface:

.Lexical Structure-What symbols are there?
.Syntactic Structure-How do they relate to each other?
.Semantics-How do they relate to the things they represent?
.Pragmatics-How do they relate to the users?

From the overall visualization systems reviewed, we can find that all the systems take some steps and require roughly the same overhead for visualizing algorithms.

In Lee's [11] data structures display system, the user's algorithm texts must be translated before the system can work on this algorithm. It requires the user to be familiar with the Algorithm Specification Language.

In Brown's [2] Balsa system, animating an algorithm involves three steps. The first step is to split the program into three components: the algorithm itself, various input generators, and various views that present the animated pictures of the algorithm in action. The second step is to implement each component. The third step is to identify for Balsa those views and input generators, and to give textual name for each algorithm, input generator, and view. In Stasko's [20] Tango system, it also takes three steps to produce an animation.

With a view to decreasing the overhead of producing the visualization for algorithm, We have developed a ready-to-use data structures visualization system in this study. However this feature will be at the cost of visualizing user's input algorithms. This system stresses mainly on an education aid but not a debugging tool for users.

The similarity between our system and the above systems mentioned is that they all provide the instructional function that helps user understand programs. The differences between our system and others are that (1) this system needs no preliminary work before the user visualizing the algorithms, (2) the visualization part of this system is developed using some X Window primitives and without the aid of any animation package and is portable, and (3) this system cannot accept user's input algorithms for visualization.

In this visual aid, we have chosen tree-based data structures - AVL tree, Splay tree, Red-Black tree and B-tree for visualization, because those are parts of structures lectured on data structure courses. Besides, this kind of visual aid can save time and efforts for the learning of tree-based data structures and their properties.

# CHAPTER III

# ABSTRACT SYSTEM MODEL

In this chapter, we describe an abstract model of the visualization system. In this abstract system model, we first characterize this visualization system as an interactive service model, then we want to establish the contextual diagrams of this system's functionalities.

## Interactive Service Model

In this interactive service model, the main goal is to provide an interactive environment that the user can get the tutorial guidance step by step. To achieve this goal, we call the users who utilize the "Tree-based Data Structure Visualization System" end-users. The end-users watch and interact with this system on the X terminal. In the end-user's model, the users of this system environment are always in a "setup-and-run" loop:

Setup: The end-user chooses from a variety of tree algorithms that he or she wants to learn in the display. The end-user also decides which operation to be performed in this algorithm, and what the input values to each of those operations should be.

Run: The end-user runs the algorithm step by step in an interactive environment. The end-user can watch the whole process of the algorithm running in the view windows on the screen. While the algorithm is running, the end-user can decide to examine any operation on this algorithm, such as insertion, deletion, or tracing back, and event replaying.

The following figures, figure 2 and figure 3 specify abstract external behaviors of the "Data Structure Visualization System".

It deals with the general dialog patterns between the system and the user.

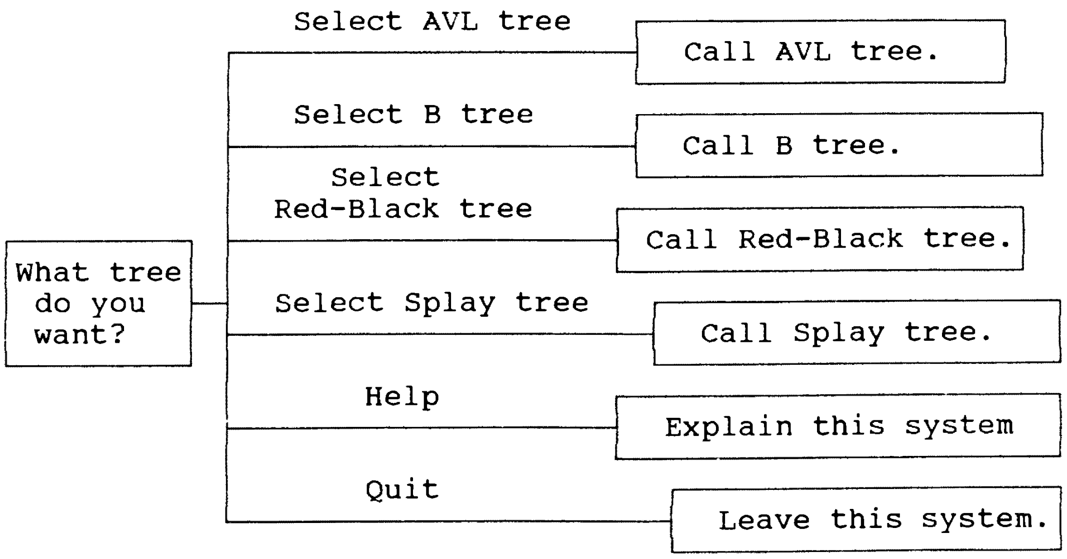


Figure 2. Main Functions of TBDSV System

Figure 2 shows the main menu of this system and the tree data structures provided for the users.

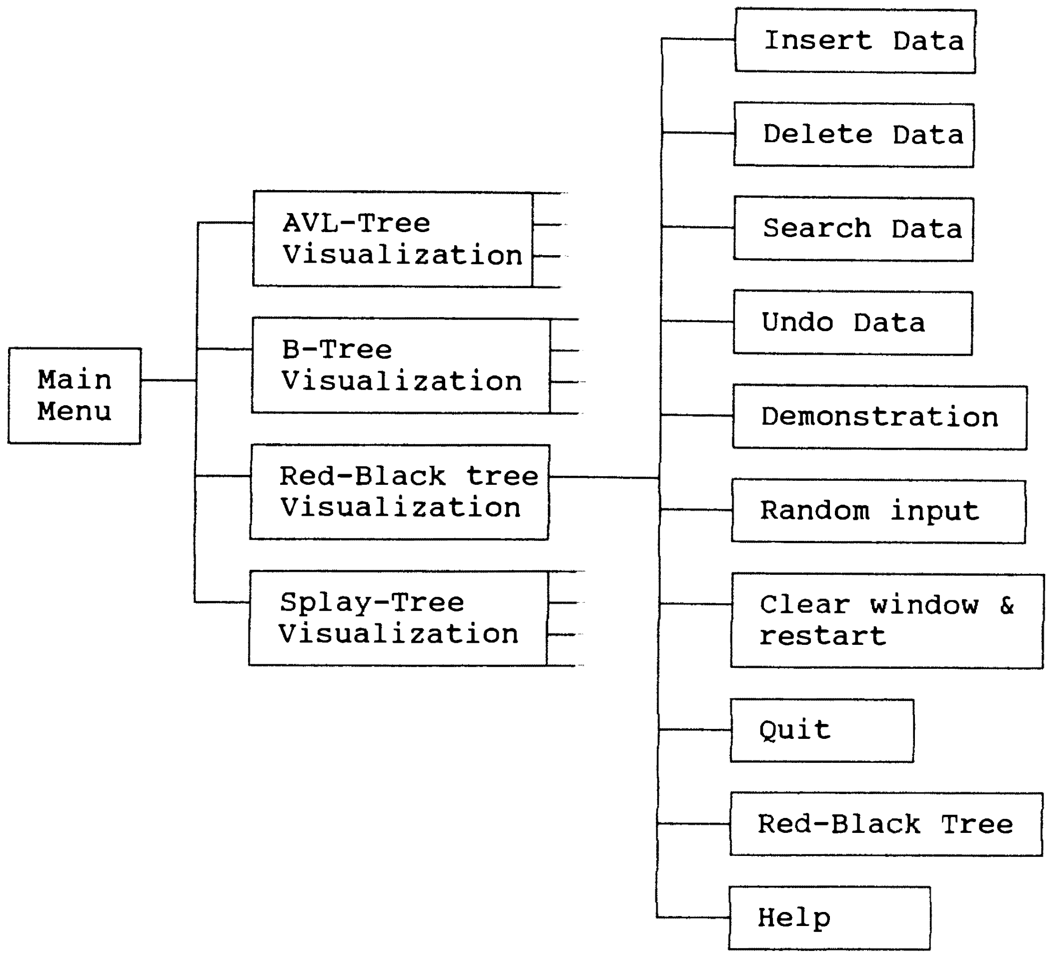


Figure 3. Trees and Their Operations

Figure 3 shows that we can choose four trees' visualization from main menu. There are many operations available for the tree's visualization. The first three functions, insert data, delete data, and search data, account for the basic operations in the tree's definition, and others are erlated to the animation. Undo data is the function that makes the users see the previous versions of tree before the current operation is issued. Demonstration is the function that shows the tree's operations and their implementations automatically without the user's input. All of the operations on the tree can be restarted by the clear function. The help function works as a documentation for the above functions, and the quit function will bring the user back to the main menu. The function named "Red-Black tree" is the one that displays a textual description about this tree.

## Contextual Diagrams of Functionality

The contextual diagrams describe the system's interface to the outside world and functionality inside this system. In the following diagrams, figure 4 is about the system inputs, the system outputs, and figure 5 is about the types of display on the screen, and the contents included in each type.

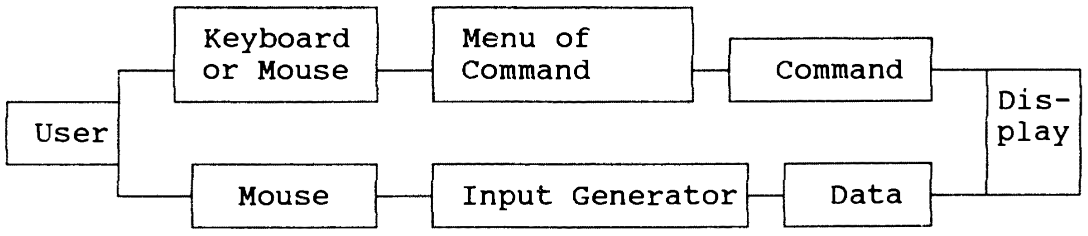


Figure 4. Input Output Diagram of TBDSV System

In figure 4, keyboard and mouse are the input devices we use. We can select the command we want from the menu by these input devices. If this command needs input data, we can use mouse to drive the input generator which produces the data for this command. Menu and input generator are collections of buttons which are made visible and to be selected by the users.

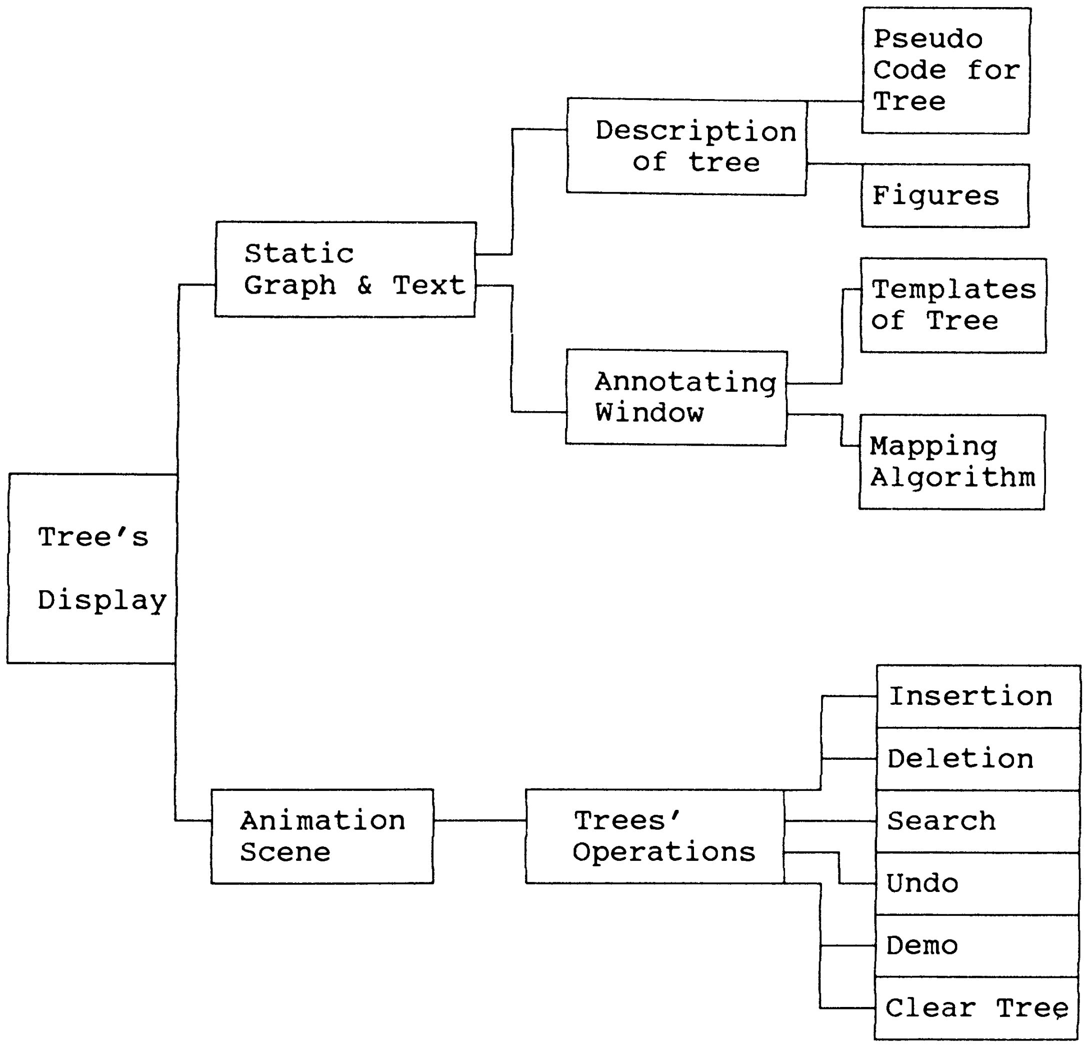


Figure 5. Display Type and Its Contents

In figure 5, we show the functionality of all kinds of displays that contribute to the tree's visualization. An animation scene is a window created for the dynamic display of tree's operations (like insertion, deletion, and search). Besides, animation scenes are annotating windows that contain graphics of tree's templates and their mapping algorithm. The description of the tree is a brief textual introduction and graphics templates of this tree. Annotating window will emerge to illustrate the behaviors shown on the animation scene.

# CHAPTER IV

# SOFTWARE DESIGN

## Development Process

The algorithm visualization system is designed by following a process model. This process model involves the activities shown below:

(1) *Data type declaration* The data type and storage structure for every tree are declared.

(2) *Algorithm decomposition* It is necessary to decompose an algorithm into a set of functions that best represents the tree's distinct behaviors.

(3) *Visualization design* We first specify four kinds of abstract data types in visualization, and use those data types to achieve trees' visualization.

(4) *Component design* This part designs the services provided by this system. As figure 3 has shown, those are services for every tree. Each service is viewed as a component.

(5) *User interface design* The user interface will rely on windows, pull-down menu and pointing devices. This design is characterized by support for graphical as

well as for textual information display.

The above activities are correlated to each other. In figure 6, we illustrate their sequential relationship. This approach allows the following: as errors or imperfections are detected, the information will be fed back to allow earlier design stages to be refined.

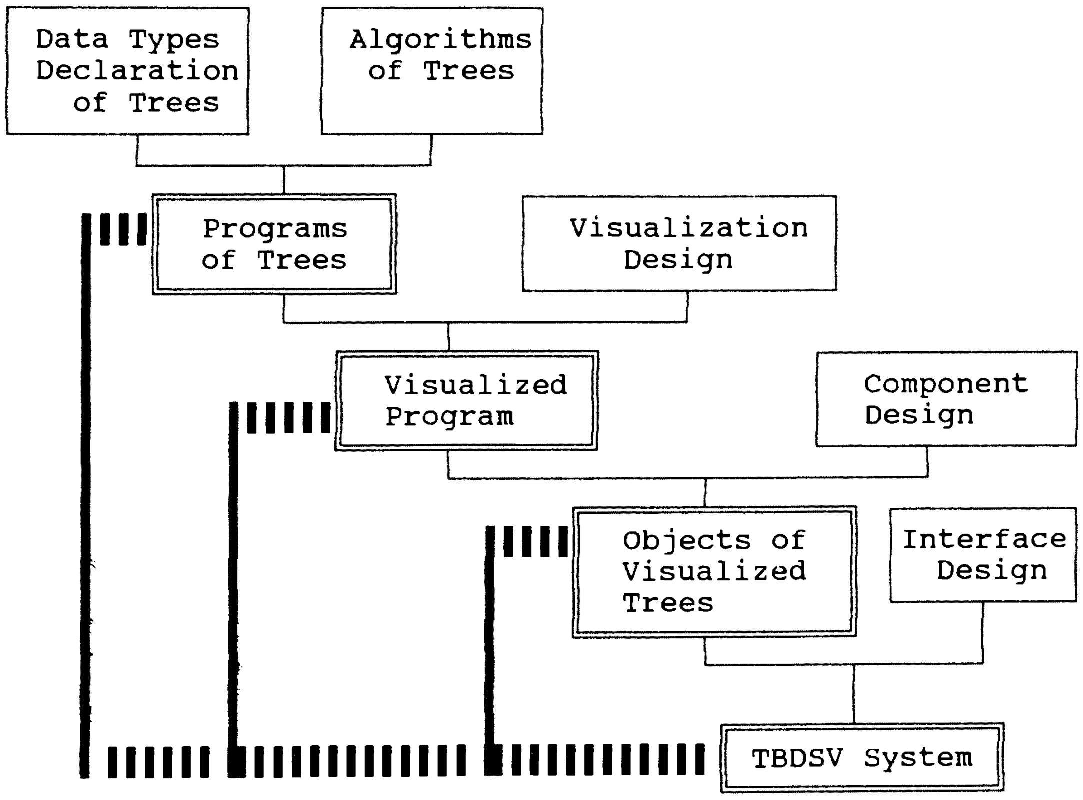


Figure 6. System Formulation

In Figure 6 the box with double lines represents the major products in this formulation. And the bold lines indicate the major products which are not finished in linear order but rather in the order with feedback.

## Data Type Declaration

Data structures and algorithms constitute a program. In this first stage of software design, we describe the storage structures used to accommodate the various possible node formats for each tree.

C language type declarations are used to specify the storage structures. Not only is the algorithm coding based on them, but also the well-defined data structures have the fundamental information for showing the images of the trees.

Figure 7 shows the declaration and type of data structure for every node in the AVL tree. The balance factor is computed using the formula:
Balance factor = left_height - right_height.

```
typedef struct tree_node { /* Build a binary tree */
  struct tree_node *left;  /* node for AVL.   */
    char data[20];   /* the balance factor is   */
    int left_height; /* left_height - right_height */
    int right_height;
  struct tree_node *right;
  }NODE;
 /*-----------------------------------------------*/
  NODE *root;
```

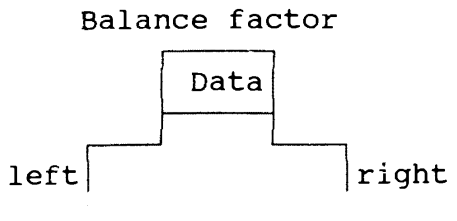


Figure 7. Storage Structure for a Node of AVL Tree and Its Declaration

Figure 8 shows the declaration and type of data structure for every node in the Red-Black tree. Color is determined by the expression : Color = r : b ? potential<0 : potential >0.

```
 typedef struct tree_node { /* Build a binary tree */
 struct tree_node *left; /* node for Red-Black tree. */
   char data[20];     /* the potential is the value */
   int potential;     /* to determine the color.   */
   int color;
 struct tree_node *right;
 }NODE;
/*-------------------------------------------------------*/
 NODE *root;
```

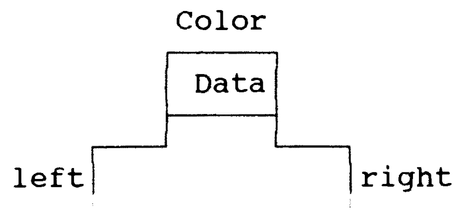
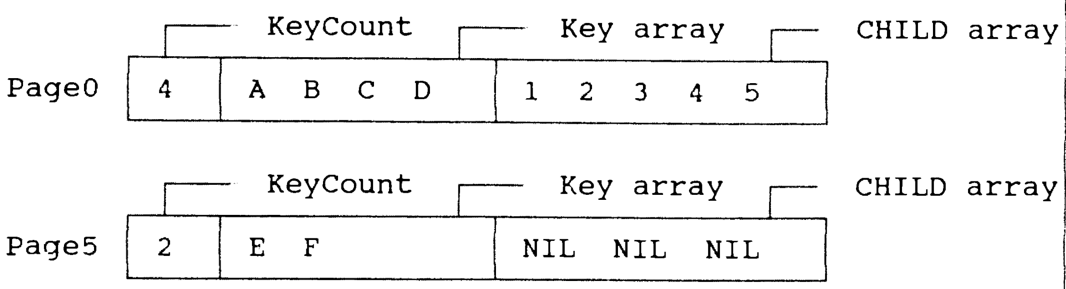


Figure 8. Storage Structure for a Node of Red-Black Tree and Its Declaration

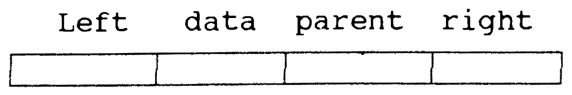
Figure 9 shows the declaration and type of data structure for every node in the B-tree.



```
typedef BTPAGE {           /* Build a B-tree node       */
 short  KEYCOUNT;          /* number of key in page     */
 char  KEY[MAXKEYS];       /* the actual keys            */
 short CHILD[MAXKEYS+1];  /* pointer to relative record */
 } PAGE;                    /* numbers of descendants    */
/*-------------------------------------------------------*/
```

Figure 9. Storage Structure for a Node of B-Tree and Its Declaration

Figure 10 shows the declaration and type of data structure for every node in the splay tree.



```
typedef struct tree_node { /* Build a binary tree node */
  struct tree_node *left;  /* for splay tree.          */
    char data[20];
    struct tree_node *parent;
    struct tree_node *right;
    }NODE;
   /*-----------------------------------------------*/
   NODE *root;
```

Figure 10. Storage Structure for a Node of Splay Tree and Its Declaration

## Algorithm Decomposition

Algorithm decomposition consists of the following steps:

Step 1: Write a C program according to the algorithm that is planned to be visualized. The program of the algorithm must be modulized according to a functional viewpoint. For example, for the AVL tree to be modulized, it will include "search", "insertion", and "deletion" modules.

Step 2: Test this program thoroughly. This process includes "Validation & Verification."

When the data type for those trees is determined, the next step is to code the algorithms for every operation of trees. The coding of the algorithms for those trees must be precise in accordance with the definitions of those trees. In addition to this point, the programs of those trees are built out of modules, which are composed of procedures and functions.

In every kind of tree-based data structure, its operations consists of many basic operations, and those operations constitute the behavior of trees.

Those basic operations include the functions of insertion, deletion, and search for the trees. But for the reason that the tree behaviors are to be visualized, the above operations must be sub-divided into a set of functions according to the tree's attributes. In table I, we list the functions that are needed for the completion of every tree's insertion. Those functions are the decompositions of insertion, and they are invoked in sequence.

TABLE I

DECOMPOSITION OF INSERTION IN TREES

| Tree type | Operation of Insertion (Including restructing) |
|---|---|
| AVL Tree | 1. Insert_key(root);<br>2. Count_balance_factor(root);<br>3. Rotation(root); |
| Red-Black Tree | 1. Insert_key(root);<br>2. Count_potential(root);<br>3. Rotation(root);<br>4. Recolor(root); |
| B-Tree | 1. Insert_key(root);<br>2. Splitting(root);<br>3. Promotion(root); |
| Splay Tree | 1. Insert_key(root);<br>2. Zig_left(root);<br>3. Zag_right(root);<br>4. Zig_and_Zag(root); |

After we have decomposed the tree operations into the functions so that they have their own peculiar behaviors, the next step is to specify each function using pre- and post- conditions. A pre-condition is a specification of the value of the function's inputs. And a post-condition is a specification of the value of the function's output. The difference between them defines how this function transforms its inputs to its outputs. Since every function in a program is closely related, one function's post-condition is another function's pre-condition.

This analysis of every function's pre- and post-condition will be very helpful for the next stage in the designing of tree's visualization. In the table 2, we specify all functions with pre- and post-conditions in the insertion module of AVL tree.

TABLE II

FUNCTIONS SPECIFIED USING PRE- AND POST-CONDITION

| Functions in the Insertion Module. | Pre-Condition. | Post-Condition. |
|---|---|---|
| 1. Insert_Key(key). | Root = NULL.<br>---------------<br>Root !=NULL.<br>(Balanced Tree) | Create Root.<br>------------------<br>Binary Tree.<br>(may be unbalanced) |
| 2. Count_Balance(key) | Binary Tree. | Binary Tree with New Balance Factor in Each Node.<br>(may be unbalanced) |
| 3. Rotation(key). | Unbalanced Binary Tree. | AVL-Tree. |

From the table II, we can visualize the first function abstractly that it may create a root node or become an unbalanced binary tree. In the second function, we can show the new balance factors of the nodes in this unbalanced tree. In the function of rotation, the focus is on the processing of transforming an unbalanced tree to an AVL tree.

Knowing the pre- and post- conditions of each function, we can thus give all functions their expected actions in the process of visualization. For the previous example of insertion in AVL tree, the algorithms of each function with its action is given in figure 11. One thing we need to point out is that the action and the actual code are all independent of each other.

```
Insert_act(root,key)
  {
     If (insert_key = any_node in this tree)
       {
        Create message window;
        Show error message;
        return root;
       }
     If (root=NULL)
       {
        Create a node;
        Draw data string in it;
        root=new_node(insert_key);
       }
     else if (root_key > Insert_key)
       {
        Blink the root key;
        root=root->left;
        Insert_act(root,key);
       }
     else if (root_key < Insert_key)
       {
        Blink the root key;
        root=root->right;
        Insert_act(root,key);
       }
      Return root;
  }
```

Figure 11. Insertion of AVL Tree with Action
(the underlined code specifies the added action to the normal code)

## Visualization Design

Of all the design activities in this system, visualization design is the most important part that determines if this visualization system is a success or not in terms of usage.

We consider the following requirements that the visualization design should meet:

(1) Easy shape recognition: It must be easy for viewers to associate the shape or color with the data type it represents.

(2) Easy change detection: The viewers must easily detect what operation has been done in each step and the transformation effected by this operation.

(3) Rapid drawing: When a figure or text is to be drawn, it must be drawn rapidly to meet the fast response time.

(4) Selective display: Each time the screen only shows the necessary figure and information for the current operation.

Developing a good visualization will involve the aesthetic knowledge and that is beyond the scope of this study. In crder to simplify animation design and provide a model that supports smooth, continuous image movement, Stasko[12] have developed an algorithm animation component that helps design animation actions to simulate the algorithm's operations.

In this component's formal model, it contains four abstract data types. They are the graphical images, the locations of images and other objects, the images' transitions, and the paths that modify those transitions. The following gives the definitions of those four abstract data types.

Images: An image is a graphical object that undergoes changes in location, size, color, etc. throughout the frames of an animation. Primary images include lines, rectangles, circles, and texts. Composite images are collections of primary images with geometric relationships to one another, as defined by a list of primary images in a local coordinate system.

<u>Locations</u>: A location is a position identified by an (x,y) coordinate pair in the animation coordinate system. The ability to save and reference particular locations is an important tool for animation design. Locations often denote a particular variable in a program, while the image at that locations denotes the variable's value.

<u>Paths</u>: A path designates the magnitude of change in image attributes from one frame to the next. Images can only be modified through paths; for example, images are moved or colored along paths, and their visibility is changed along the paths. 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. The length of a path p, denoted by $|p|$, is the number of coordinate pairs it includes.

<u>Transitions</u>: A transition uses a path parameter to modify an image's position or appearance, and to give an animation action. Like images, transitions have an extensible definition that does not restrict the framework to a predefined set of types. Simple transitions are defined by a transition type, the image being altered, and a path-argument modifier. Typical transition types include move, resize, color, fill, raise, lower, delay, and alter

visibility.

With those four abstract data types in mind, we can create animations of algorithms by assembling collections of image, location, path, transition, and association operations that accomplish desired animation actions. In table III, we list the four abstract data types used in this work to produce the animation scenes.

TABLE III

ABSTRACT DATA TYPES IN TREE ALGORITHMS VISUALIZATION

| Images | Locations | Paths | Transitions |
|---|---|---|---|
| .Lines<br>.Rectangles<br>.Circles<br>.Color<br>.Texts | .(x y)<br>Coordinate<br>Pairs | .Distance<br>Between Image<br>(rectangles,<br>circles, etc.) | .State cues<br>.Sound<br>.Highlighting<br>.Continuous<br>.Discrete |

In the trees' images, the rectangles and circles represent the nodes in the trees, and the directions of lines coming out of the nodes indicate the relationship between nodes. Besides, color is used when there are two kinds of nodes, red and black in the red-black tree. The text written or attached on the previous images is the information for the images. The following figure illustrates trees' images.

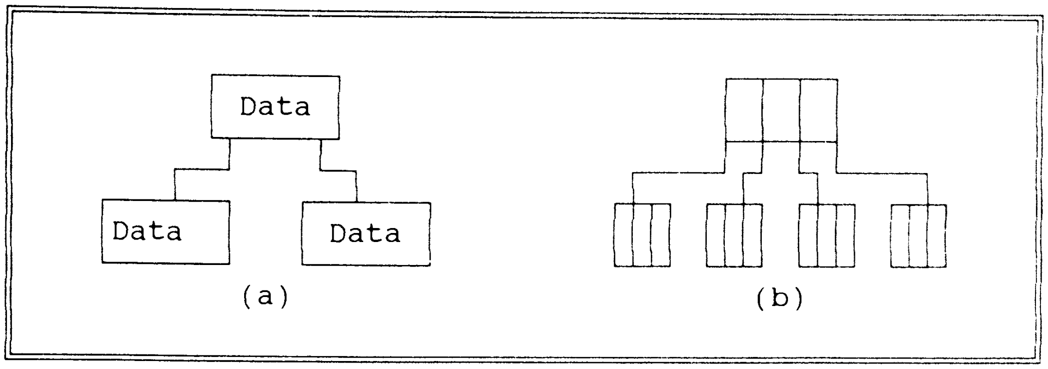


Figure 12. Tree's Hierarchical Structures (a) A Binary Tree Structure. (b) A B-tree Structure of Order 4.

Before the images can be shown, we must give them their positions in the display window. In the windowing system, the origin of this display window is located on the upper-left corner of the window.

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

Different transition types use path arguments in different ways. In the above table, we have listed four kinds of transitions in the tree algorithms' animation.

The first method of transitions is state cues. State cues are to show changes in the state of an algorithm's data structures by changing the images of their graphical representations on the screen. For example, in the function search(key), we make the tree node blinking when it is compared with the search key. The viewers can observe when and which of the nodes are being compared. That produces the effect of state cues.

The second method of transition is by sound. In the process of a tree's animation, sound can add rhythm to the motions of tree's operations. And different sounds remind the users what kind of situation happens.

The third technique of transition is by highlighting. With highlighting, we can attract the viewers' attentions to the images that have been highlighted. For example, in B tree's visualization, we can highlight the node that is going to split after inserting one more key into this node.

## Continuous Transition

Continuous transition means the displays of the algorithm's motions which are shown in a smooth way as viewed by the human eyes. London[20] states the importance of smooth updates (continuous transitions):

> Often it is even better to show smooth transitions between states; if a structure changes and the new state simply flashes on the screen, the viewer is typically startled and cannot see immediately without some mental effort how the new image evolved from the previous one.

However, achieving the smooth transitions of the algorithm in action is not an easy job. Brown[2] points out that "Unless a good animation package is provided, incremental transitions are often tedious and difficult to program." Without the animation package, we take advantage of Bresenham's line and circle algorithms to apply them in the continuous transitions.

In the trees' algorithm animation, the motions include the node being created after insertion, the nodes' rotations for balancing the height, and a series of nodes' movements after deletion of nodes. Here we use Bresenham's line-drawing algorithm to create the motions for which the images move by the straight lines. However, Bresenham's circle-drawing algorithm is used to create the motion of rotation.

But before the Bresenham's algorithms can be used in the animation, some modifications are made in the Bresenham's algorithm. Table IV lists the differences between the Bresenham's algorithm and the modified algorithm we need. And we summarize the animation algorithm in figure 13.

TABLE IV

COMPARISON OF BRESENHAM'S ALGORITHM AND ANIMATION ALGORITHM

| | Bresenham's Algorithm | Animation Algorithm |
|---|---|---|
| 1 | Set the Pixel Value on the Screen. | Plot the Image on the Screen. |
| 2 | Input a Line's Endpoints. | Input Multi-Lines(Paths) Endpoints(Locations) |
| 3 | Purpose: Draw Line | Purpose: Produce Animation |

The Bresenham's circle drawing algorithm can also be adapted to create the circular paths in continuous transition.

## Discrete Transition

In contrast to continuous transition which has a smooth motion, the discrete transition is achieved by an abrupt erase-and-repaint method. This way can be used for the efficient displays for the viewers. For example, on the

```
1. Input paths. If there are paths a, b, c, d, then
   store two endpoints of each path:(ax1,ay1), (ax2,ay2),
   (bx1,by1), (bx2,by2), (cx1,cy1), (cx2,cy2), (dx1,dy1),
   (dx2,dy2).

2. The starting point of every motion is the first endpo-
   int of its path. In path a, the starting point is
   (ax1,ay1).

3. Compute the distances in each path's both directions.
   In path a, the distances are (delta_x,delta_y).
   delta_x=|ax2-ax1|, delta_y=|ay2-ay1|.

4. Compute the direction of the increment in each path;
   an increment of 0 means either a vertical or
   horizontal line.

5. Determine which distance is greater in each path.
   If delta_x>delta_y, then distance is delta_x, else
   distance is delta_y.

6. Select the longest distance from all the path.
   For (i=0;i<=distance+1;;i++)
    {
      Move_Image_a(ax1,ay1);
      Move_Image_b(bx1,by1);
      Move_Image_c(cx1,cy1);
      Move_Image_d(dx1,dy1);
      Determine ax1+1 or ay1+1 and bx1+1 or by1+1
            and cx1+1 or cy1+1 and dx1+1 or dy1+1
    }
```

Figure 13. Algorithm for Animation

left-bottom corner of the animation scene show an instructional window. In this instructional window, the animation is achieved by discrete transition. With discrete transition in the display, the viewer can quickly get the picture frame that annotates the algorithm's each motion.

## Component Design

### Basic Operations

Insertion, deletion, and search are the basic operations for the tree algorithms. Each component must invoke the input generator for its input data, then call the insert, delete, or search function.

### Input Generators

The input generator provides data for the algorithm to manipulate the operations of insertion, deletion and search. In order for the users to control what data is provided, the input generator is designed in a graphical user interface mode in order for the user to use easily, and feel in control of the process of algorithm visualization.

The input generator is the main driver that makes the trees work. The selection of input data has a great impact on the implementation of tree and messages conveyed from the tree's visualization. Brown[19] found that "small amounts of data work best for introducing a new algorithm, whereas large amounts of data help develop an intuitive understanding of an algorithm's behavior."

In order for this visualization system to achieve those goals, We propose three kinds of input generators for the trees' operations. The first one is the number input generator that shows a specific range of numbers available for the user to choose as input data. The second one is text input generator which provides varieties of alphabet and strings for the user to choose from as input data.

To use the above two input generators, the user first uses mouse device to click on the data he wants. Then the data he chooses and the pre-set operation (like insertion, deletion, search) are combined to make the tree operate accordingly.

The third, and the last kind of input generator uses the UNIX system's built-in random number generator, rand(). This function, rand(), uses a multiplicative congruential random-number generator with period 2^32 that returns successive pseudorandom numbers in the range form 0 to (2^15)-1. Then we can use the equation, rand() mod RANGE, to get the random number in the range we want.

## Demonstration Function

This function implements a tree's algorithm automatically and produces a series of animations without accepting the user's input data. We use this function to demonstrate every kind of operations and templates to the users who have no knowledge of this tree. To familiarize the user with the tree, we can freeze the template of each operation for a few minutes in this animation scene.

## Undo Function

The undo function makes the user ignore the current operation on this tree and recover from the past version of tree. This function is useful when the user makes a wrong choice or the user wants to view the motion again. This function only permits the user to go back to the previous version of tree one step.

## Help and Instruction Functions

The help function gives the user a quick reference to the usage or purpose of every function in this system. The instruction function is designed to give a user an outline describing the tree. This outline may include the definition and attribute, and the application of this tree. Furthermore, the tree algorithms as pseudo codes are also given by this instruction function. The tree's textual algorithm associated with the algorithm animation will help the user to comprehend all aspects in this tree. In this instruction function, different fonts and some figures are used to help the user get the main points and to make the text easier to read.

## User Interface Design

The user interface for this TBDSV system is achieved by graphical user interface that is based on X Windows. Two features are provided in the user interface for this TBDSV system. They are direct manipulation and menu system.

The advantages of direct manipulation and menu system are:

. Users who are in command of the system need not fear it.

. Users can get immediate feedback and the time for users to learn to implement this system is short.

. User's input errors are minimized with the feature of menu system. And typing effort is minimized by using the input device of mouse.

In the next chapter, we provided a description of this TBDSV system as seen by a user.

# CHAPTER V

## System Overview

We dedicate the first part of this chapter to the implementation details of the TBDSV system. Then we give some snapshots from the TBDSV system running in X terminal.

Figure 14 shows the TBDSV system and the environment upon which the TBDSV system is built. This overall architecture depicts the system's interface to the users and the resources that are used by this system. The input devices for the user are mouse and keyboard. The resources used by this TBDSV system are Xlib, Xt Intrinsics, and C compiler.

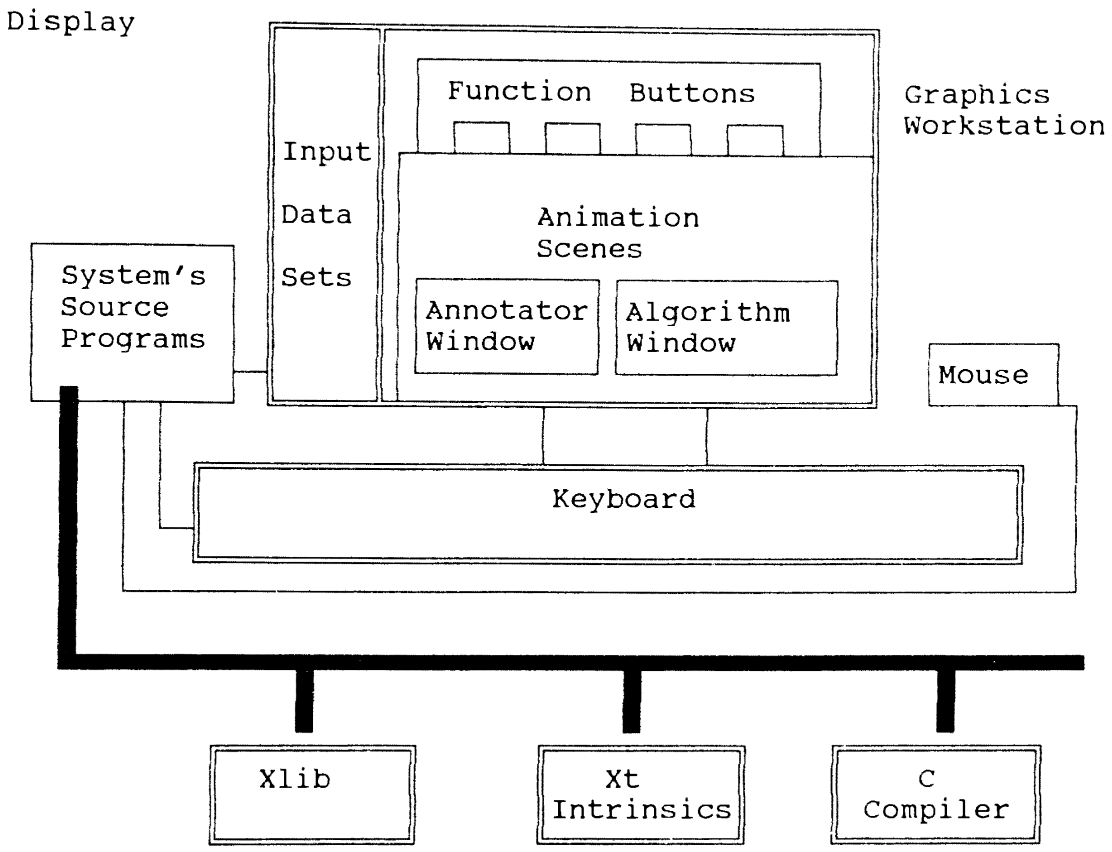


Figure 14. Overview of The TBDSV System

## Main Menu

Figure 15 shows the main menu form which user can choose the tree algorithms he wants to examine. Whenever the tree for visualization is chosen, the system will go into that tree's main loop. A typical interactive Xlib program consists of an endless loop. This endless loop is usually called the main event loop (but it is not really endless, because one of the actions would no doubt be 'quit').

The structure of such an interactive program might be summarized as:

```
 /*  structure of a main event loop  */
do FOREVER
  event = read_next_event();
  switch (type of event)
    CASE  event 1 : action 1;
    CASE  event 2 : action 2;
    CASE ...............
         ................
    CASE default : QUIT
end
```

In the tree visualization's main loop , it contains the main processing activities listed below:

Expose event: This event is always being handled. It creates the environment for activity. It is the scene before the user implements the tree's operations. In this event menu windows are displayed, and some lines and rectangles for the frame of this display are drawn.

Pointer event: When the mouse button is pressed, the pointer event is created. Each time the function window is clicked, it will call this function's component. If this function needs input data, the next pointer event is expected. When the input data is clicked, the animation scene will show the animation pictures complemented with statistics window and annotating window.

Keyboard input event: This event is created when the keyboard is pressed. Any menu items and input data can be selected by keyboard.

Keyboard mapping event: This event is created to protect the program from unexpected keyboard configuration modification.

Figure 16 shows that when the help button is pressed during tree's implementation, there will be a brief explanation of every function's usage and purpose. There are altogether 11 functions for all of the trees' implementation in this TBDSV system, They include the basic functions like insertion, deletion, search and some additional functions like undo, demo, instructional, etc. When the basic functions are chosen, the input data set must be also chosen to make those basic functions operate.

In the rest of this chapter, we will give a series of diagrams as examples of AVL tree, B-tree, Red-Black tree, and splay tree implementations.

## AVL Tree

Figure 17 shows the introductory description of AVL tree. This description includes the textual definition of AVL tree, and some figures that account for all the AVL tree rotations, and furthermore the user can also see the algorithms of elementary implementation for the AVL tree by pressing the page down button as shown in Figure 18. Figure 19 shows the AVL tree in motion for rotation when data 1, 2, and 3 are added to the AVL tree and result in unbalance. Figure 20 and figure 21 show that after balancing, the user can see the annotating window that depicts how the rotation is achieved and its mapping algorithm.

### B-Tree

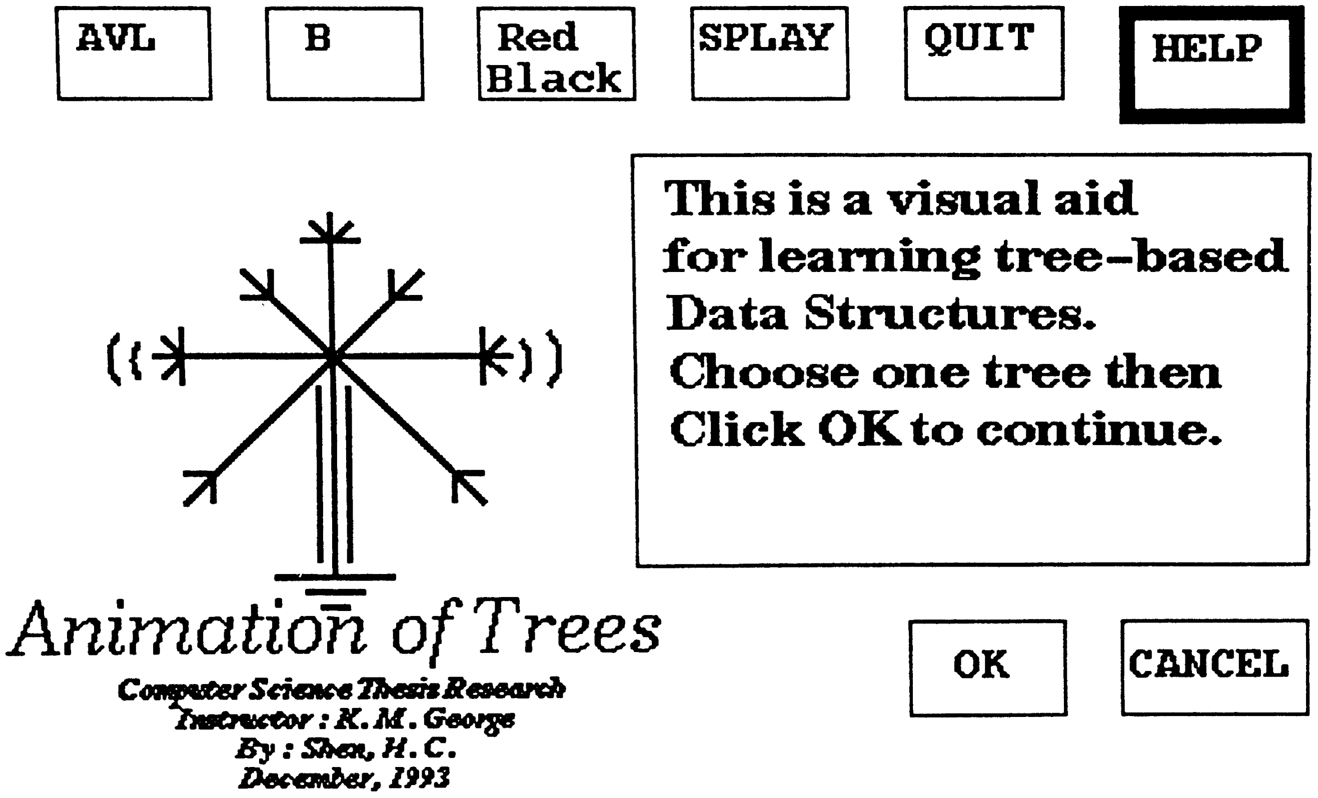
Figure 22 shows the definition and properties of B-Tree. Figure 23 and figure 24 show the pseudo codes of B-Tree's implementations. Figure 25 shows the scene for B-Tree's visualization. The nodes with highlighting mean those nodes have data in them. This tree is a fixed three level and order 5 B-Tree. The annotating window shows the information of current operation that are the key inserted, the current B-Tree by in-order traversal, and previous B-Tree by in-order traversal.

### Red-Black Tree

Figure 26 shows the instructional function provided by the Red-Black tree. This function also includes Red-Black tree's property, templates of insertion, algorithms, etc. Figure 27 shows what a Red-Black tree looks like in the display.

### Splay Tree

Figure 28 shows the instructional function provided by the splay tree. This function introduces the splay tree's purpose and the ways it is implemented, and the algorithms for the implementations. Figure 29 shows the splay tree after the insertion of data 1, 2, 3, 4, 5 , 6 and 7. Figure 30 shows what the splay tree looks like after the user applies the splay function on the node with data 7.

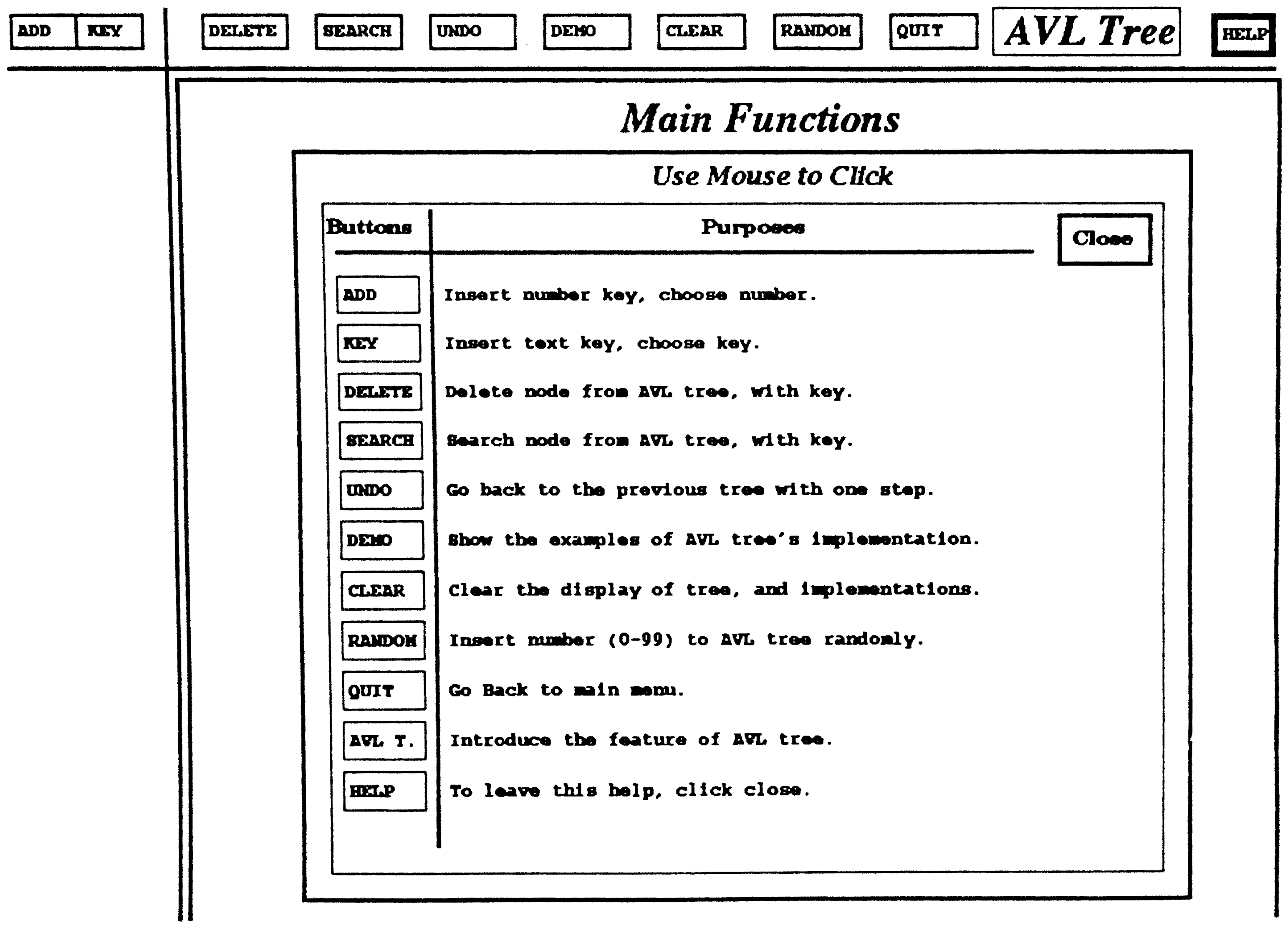


This is a visual aid for learning tree-based Data Structures. Choose one tree then Click OK to continue.

Animation of Trees

Computer Science Thesis Research
Instructor : K. M. George
By : Shen, H. C.
December, 1993

Figure 15. Main Menu

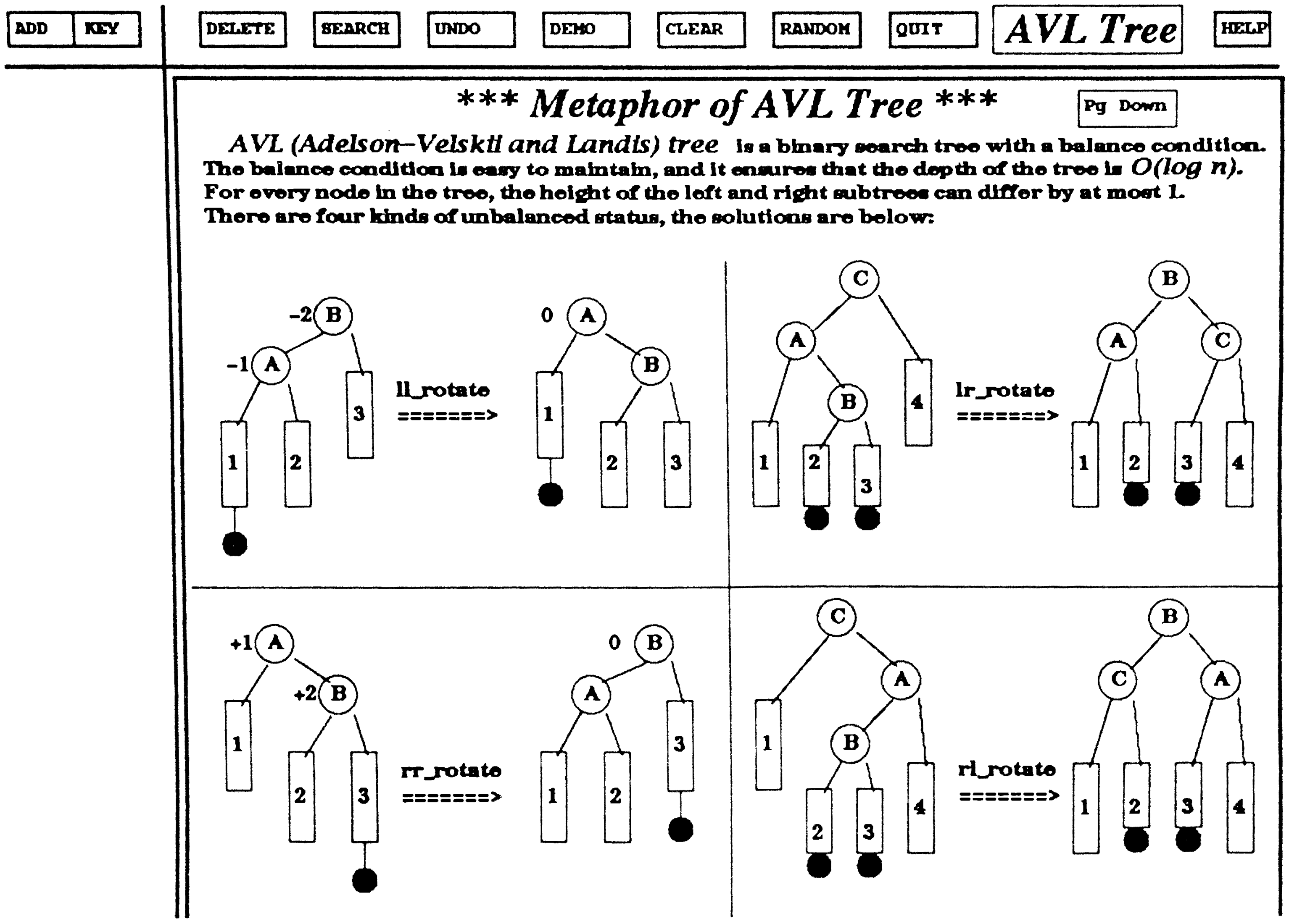


## Main Functions

*Use Mouse to Click*

| Buttons | Purposes |
|---|---|
| ADD | Insert number key, choose number. |
| KEY | Insert text key, choose key. |
| DELETE | Delete node from AVL tree, with key. |
| SEARCH | Search node from AVL tree, with key. |
| UNDO | Go back to the previous tree with one step. |
| DEMO | Show the examples of AVL tree's implementation. |
| CLEAR | Clear the display of tree, and implementations. |
| RANDOM | Insert number (0-99) to AVL tree randomly. |
| QUIT | Go Back to main menu. |
| AVL T. | Introduce the feature of AVL tree. |
| HELP | To leave this help, click close. |

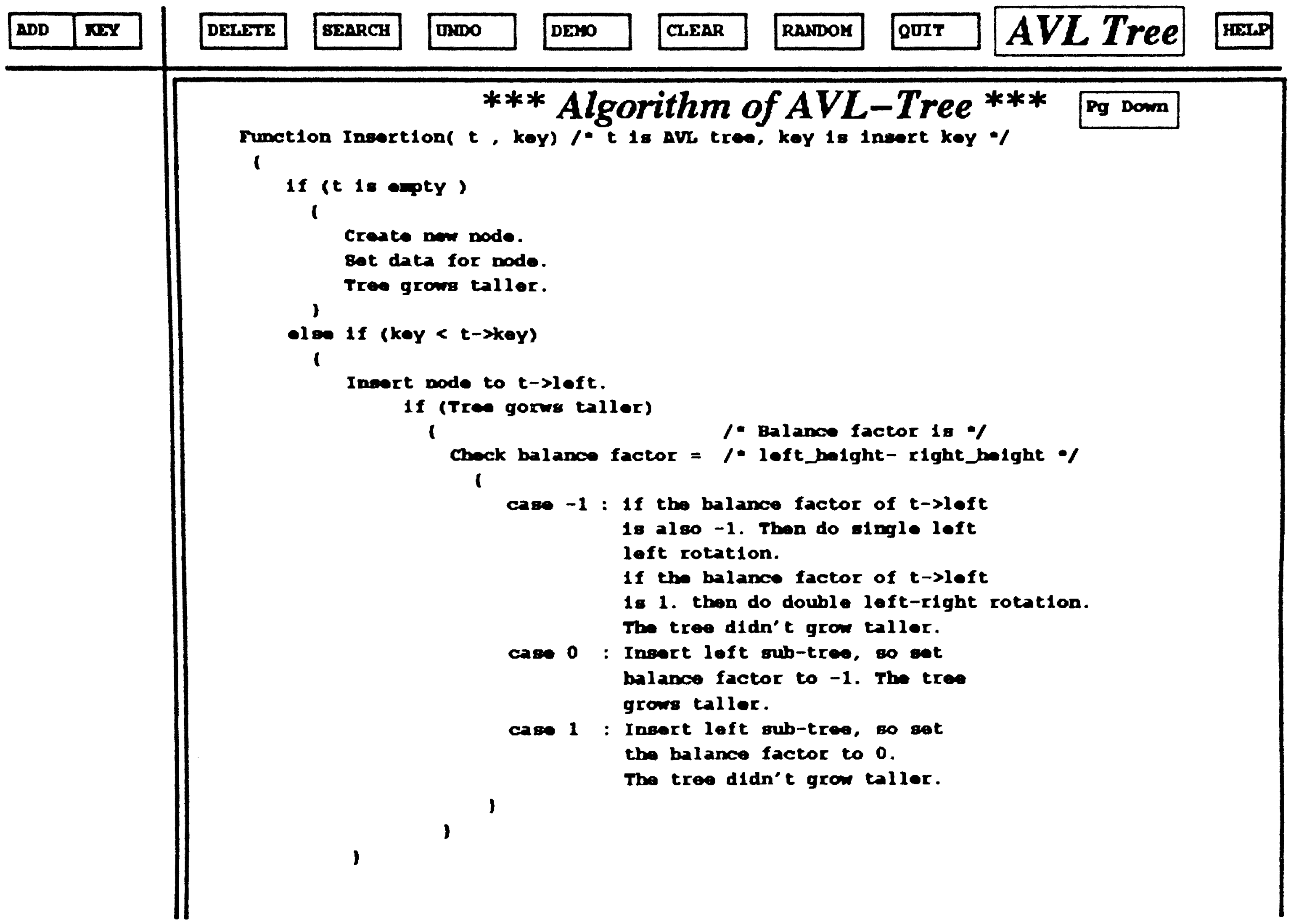
Figure 16. AVL Tree - 1



## *** *Metaphor of AVL Tree* ***

*AVL (Adelson–Velskii and Landis) tree* **is a binary search tree with a balance condition. The balance condition is easy to maintain, and it ensures that the depth of the tree is** *O(log n).* **For every node in the tree, the height of the left and right subtrees can differ by at most 1. There are four kinds of unbalanced status, the solutions are below:**

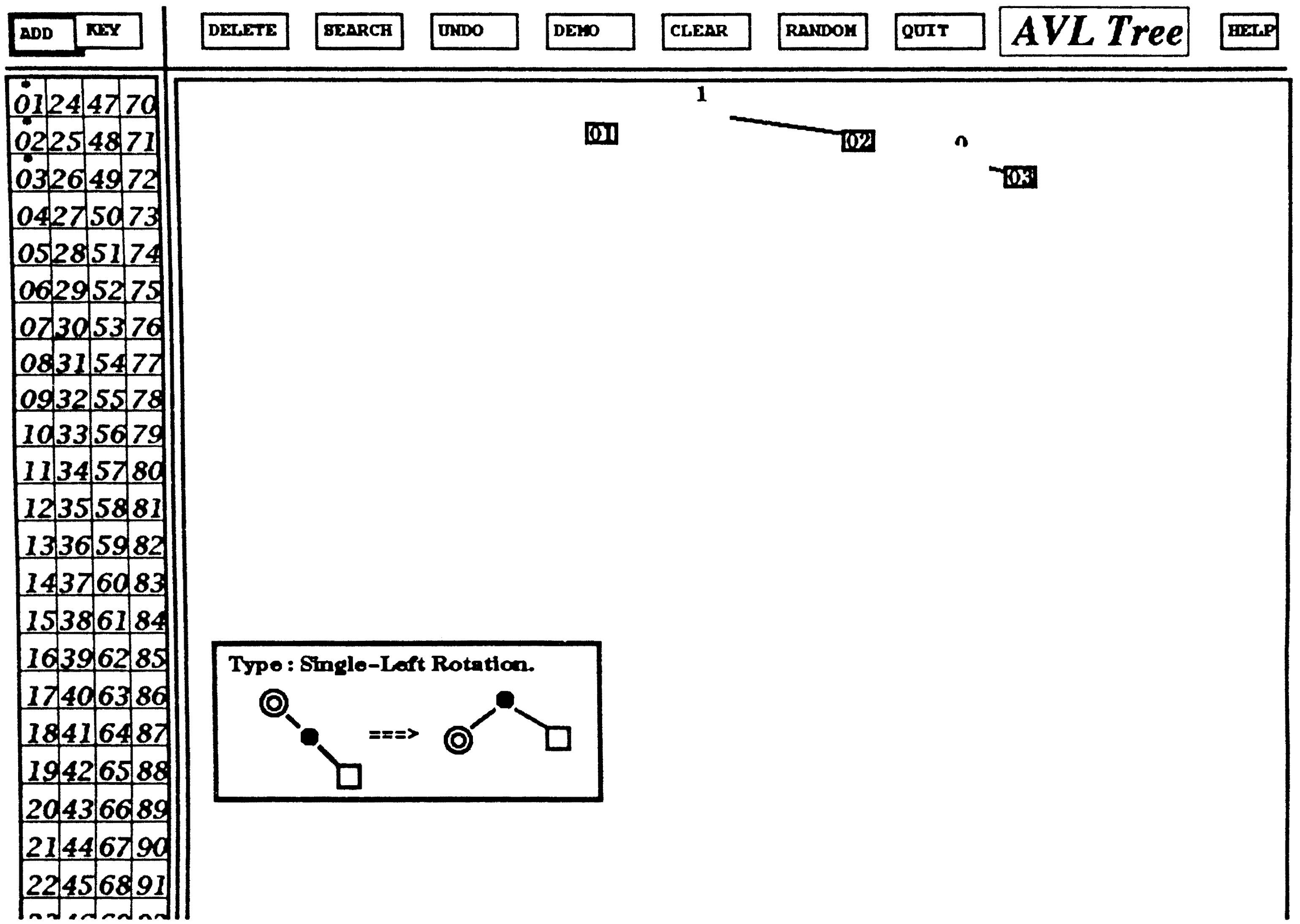
Figure 17. AVL Tree - 2



*** Algorithm of AVL-Tree ***

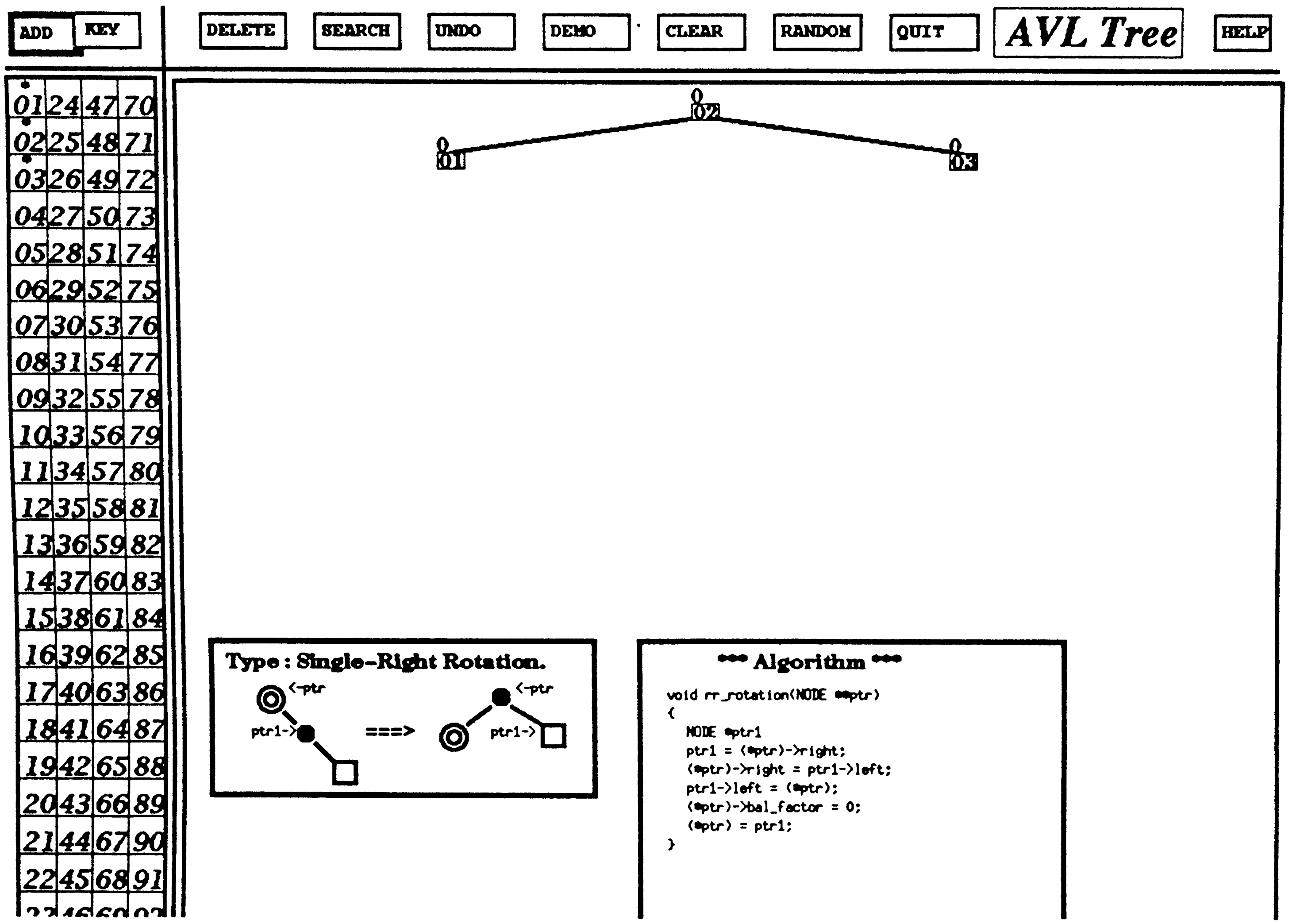
```
Function Insertion( t , key) /* t is AVL tree, key is insert key */
 {
    if (t is empty )
      {
         Create new node.
         Set data for node.
         Tree grows taller.
      }
    else if (key < t->key)
      {
         Insert node to t->left.
              if (Tree gorws taller)
                {                         /* Balance factor is */
                  Check balance factor =  /* left_height- right_height */
                    {
                       case -1 : if the balance factor of t->left
                                 is also -1. Then do single left
                                 left rotation.
                                 if the balance factor of t->left
                                 is 1. then do double left-right rotation.
                                 The tree didn't grow taller.
                       case 0  : Insert left sub-tree, so set
                                 balance factor to -1. The tree
                                 grows taller.
                       case 1  : Insert left sub-tree, so set
                                 the balance factor to 0.
                                 The tree didn't grow taller.
                    }
                }
         }
```

Figure 18. AVL Tree - 3



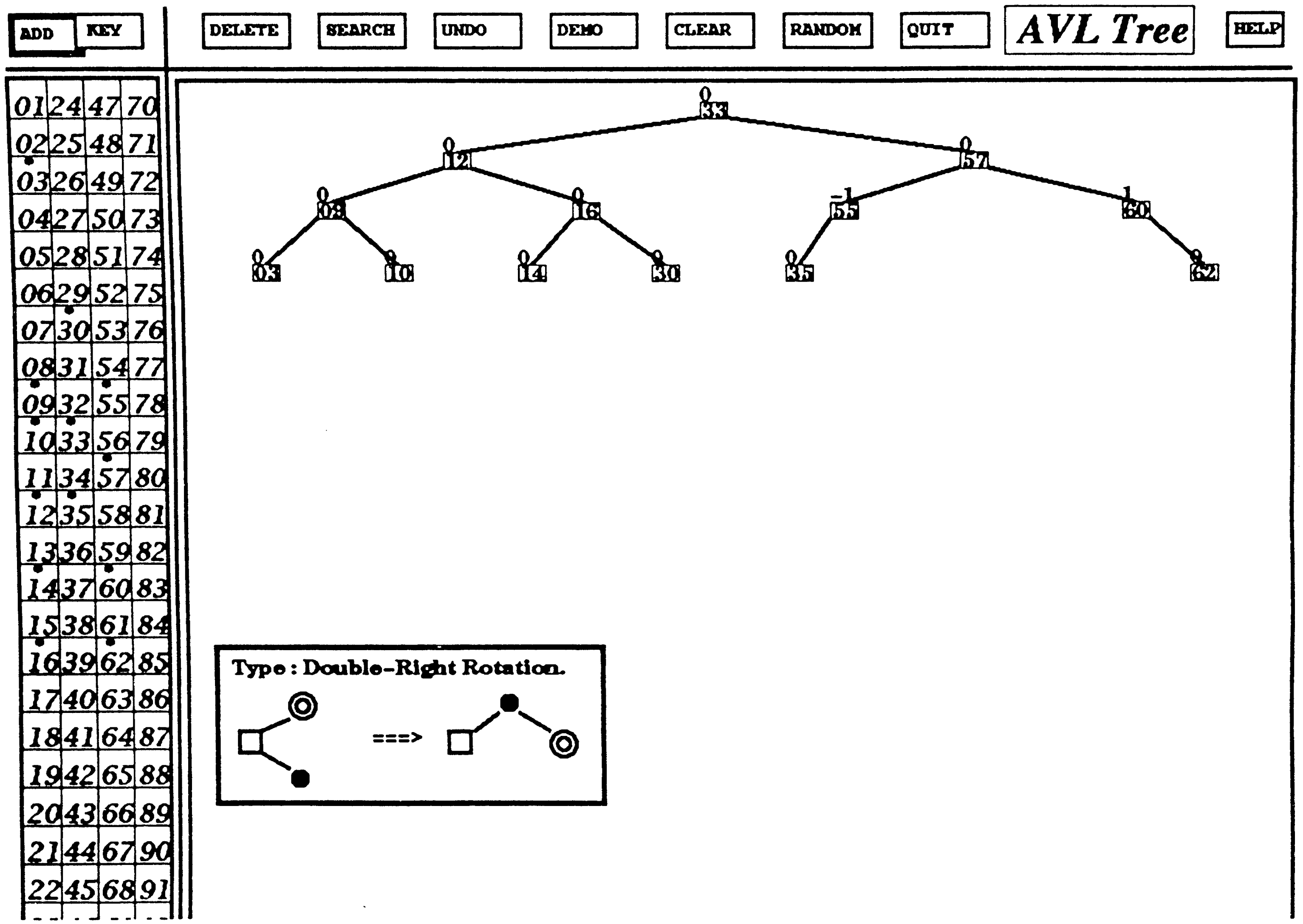
Type : Single-Left Rotation.

Figure 19. AVL Tree - 4



```
void rr_rotation(NODE **ptr)
{
  NODE *ptr1
  ptr1 = (*ptr)->right;
  (*ptr)->right = ptr1->left;
  ptr1->left = (*ptr);
  (*ptr)->bal_factor = 0;
  (*ptr) = ptr1;
}
```

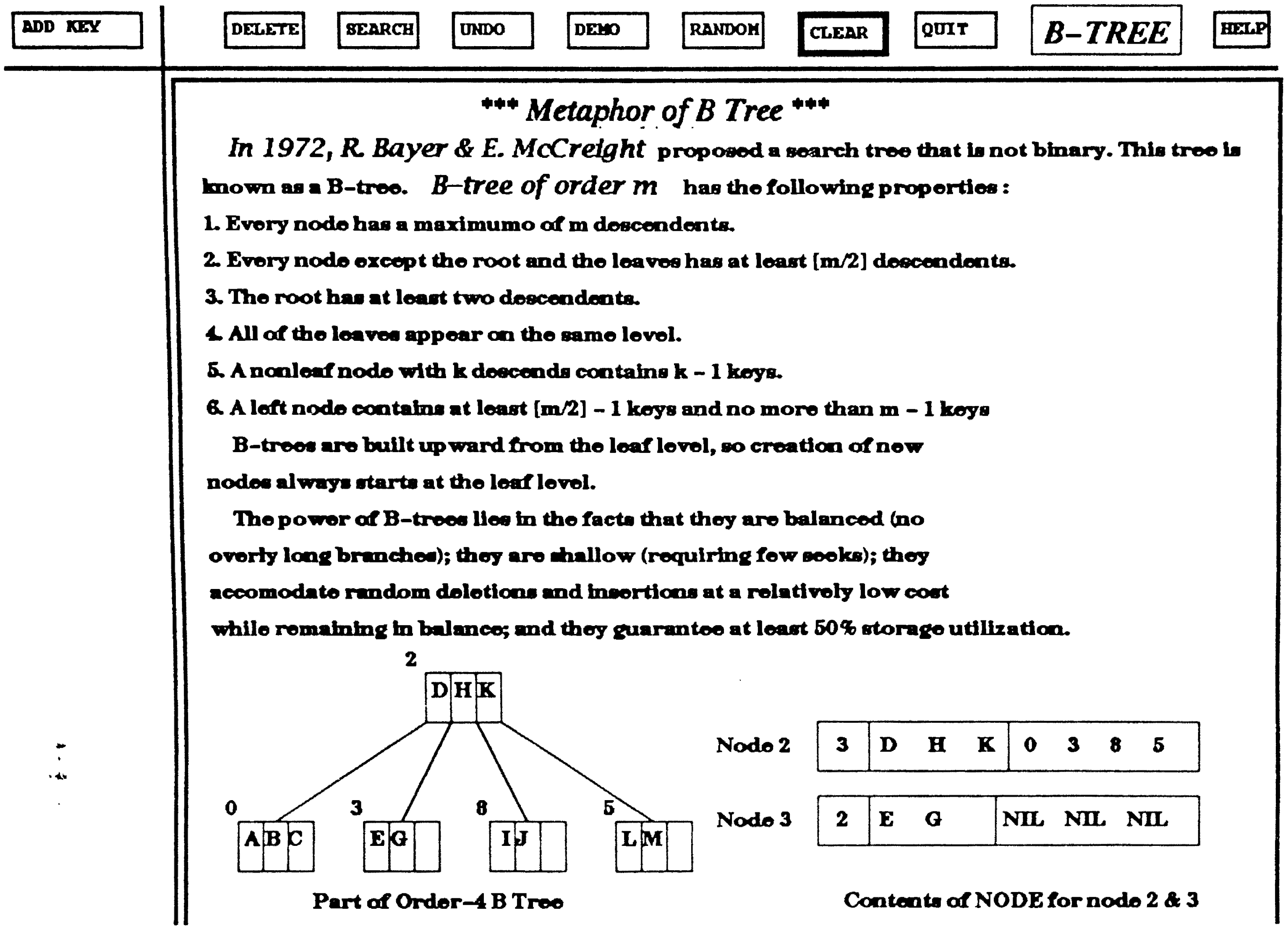
Figure 20. AVL Tree - 5



| | | | |
|---|---|---|---|
| 01 | 24 | 47 | 70 |
| 02 | 25 | 48 | 71 |
| 03 | 26 | 49 | 72 |
| 04 | 27 | 50 | 73 |
| 05 | 28 | 51 | 74 |
| 06 | 29 | 52 | 75 |
| 07 | 30 | 53 | 76 |
| 08 | 31 | 54 | 77 |
| 09 | 32 | 55 | 78 |
| 10 | 33 | 56 | 79 |
| 11 | 34 | 57 | 80 |
| 12 | 35 | 58 | 81 |
| 13 | 36 | 59 | 82 |
| 14 | 37 | 60 | 83 |
| 15 | 38 | 61 | 84 |
| 16 | 39 | 62 | 85 |
| 17 | 40 | 63 | 86 |
| 18 | 41 | 64 | 87 |
| 19 | 42 | 65 | 88 |
| 20 | 43 | 66 | 89 |
| 21 | 44 | 67 | 90 |
| 22 | 45 | 68 | 91 |

Type : Double-Right Rotation.

Figure 21. AVL Tree - 6



*** *Metaphor of B Tree* ***

*In 1972, R. Bayer & E. McCreight* **proposed a search tree that is not binary. This tree is known as a B-tree.** *B-tree of order m* **has the following properties :**

**1. Every node has a maximumo of m descendents.**

**2. Every node except the root and the leaves has at least [m/2] descendents.**

**3. The root has at least two descendents.**

**4. All of the leaves appear on the same level.**

**5. A nonleaf node with k descends contains k - 1 keys.**

**6. A left node contains at least [m/2] - 1 keys and no more than m - 1 keys**

**B-trees are built upward from the leaf level, so creation of new nodes always starts at the leaf level.**

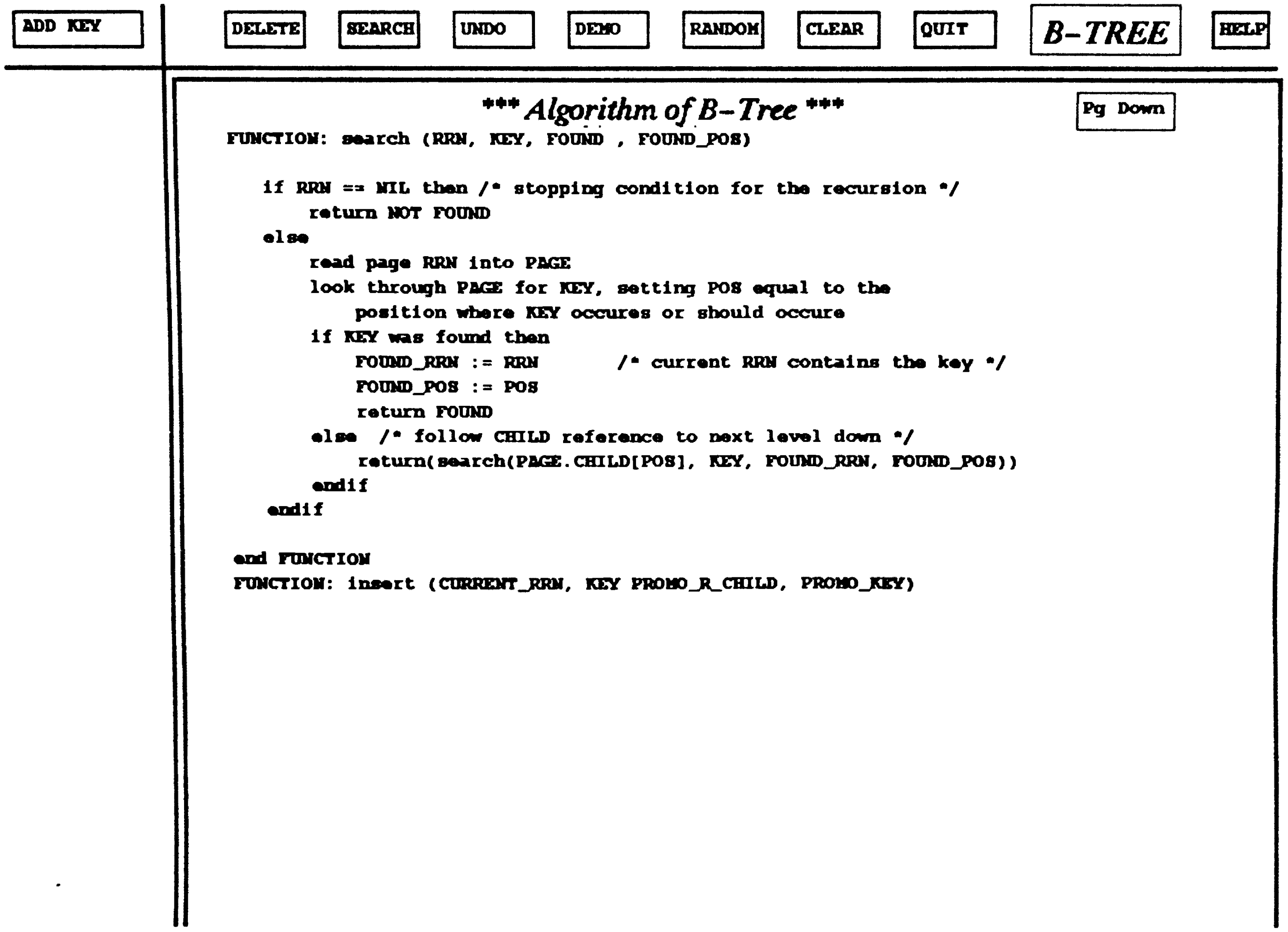
**The power of B-trees lies in the facts that they are balanced (no overly long branches); they are shallow (requiring few seeks); they accomodate random deletions and insertions at a relatively low cost while remaining in balance; and they guarantee at least 50% storage utilization.**

**Part of Order-4 B Tree**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Node 2 | 3 | D | H | K | 0 | 3 | 8 | 5 |
| Node 3 | 2 | E | G | | NIL | NIL | NIL | |

**Contents of NODE for node 2 & 3**

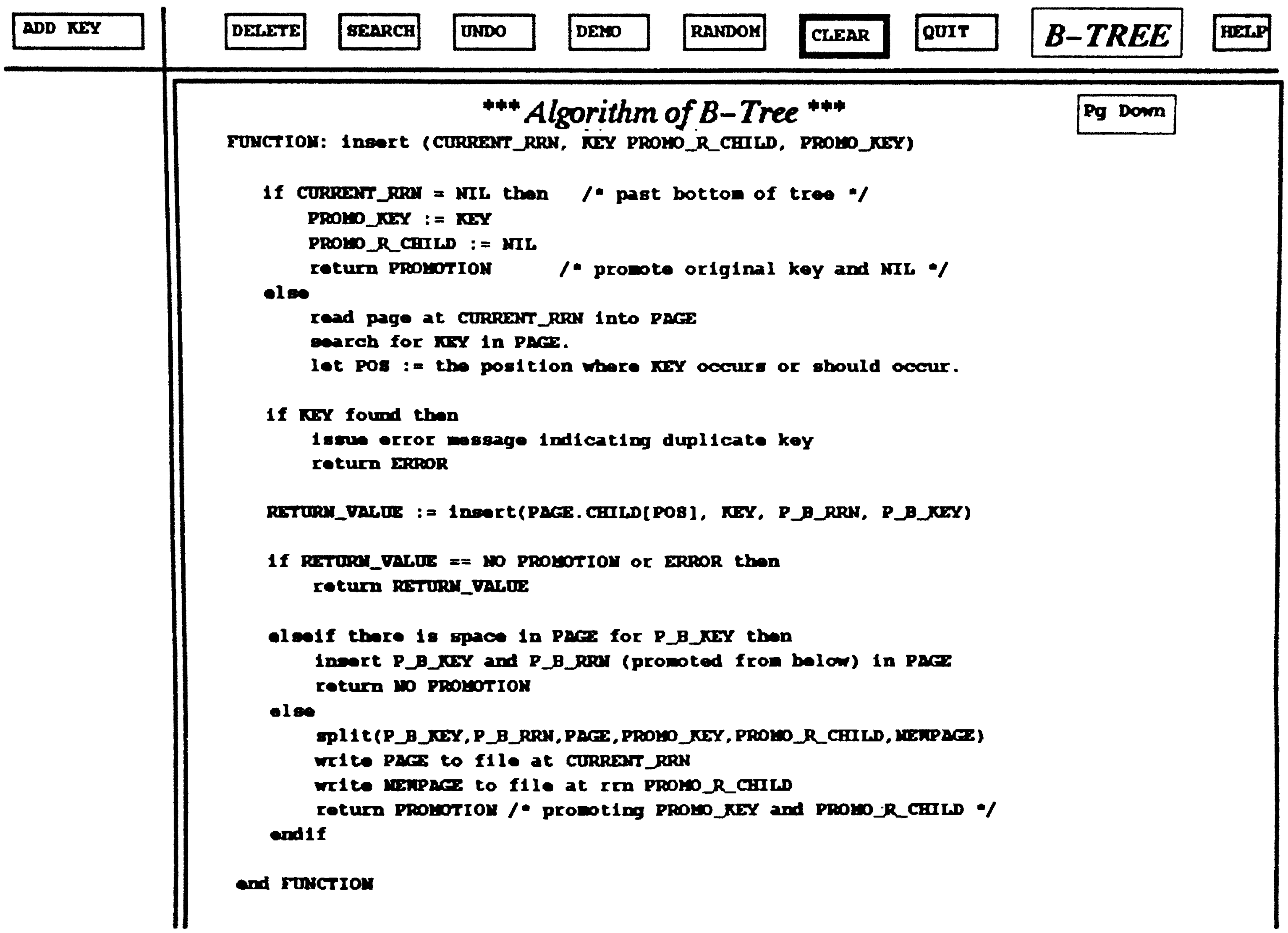
Figure 22. B Tree - 1



*** *Algorithm of B-Tree* ***

```
FUNCTION: search (RRN, KEY, FOUND , FOUND_POS)

   if RRN == NIL then /* stopping condition for the recursion */
       return NOT FOUND
   else
       read page RRN into PAGE
       look through PAGE for KEY, setting POS equal to the
           position where KEY occures or should occure
       if KEY was found then
           FOUND_RRN := RRN       /* current RRN contains the key */
           FOUND_POS := POS
           return FOUND
       else  /* follow CHILD reference to next level down */
           return(search(PAGE.CHILD[POS], KEY, FOUND_RRN, FOUND_POS))
       endif
   endif

end FUNCTION
FUNCTION: insert (CURRENT_RRN, KEY PROMO_R_CHILD, PROMO_KEY)
```

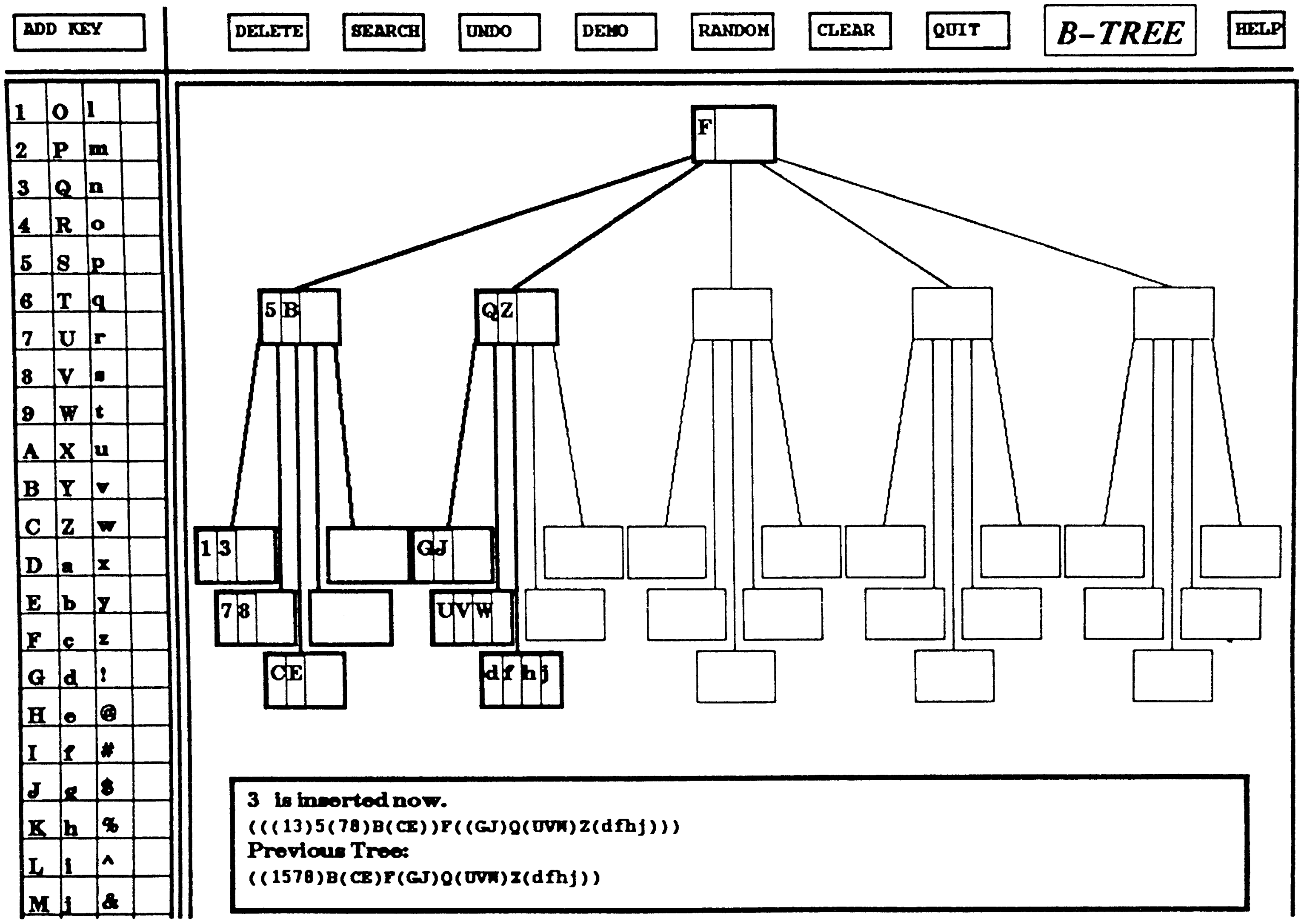
Figure 23. B Tree - 2



*** *Algorithm of B-Tree* ***

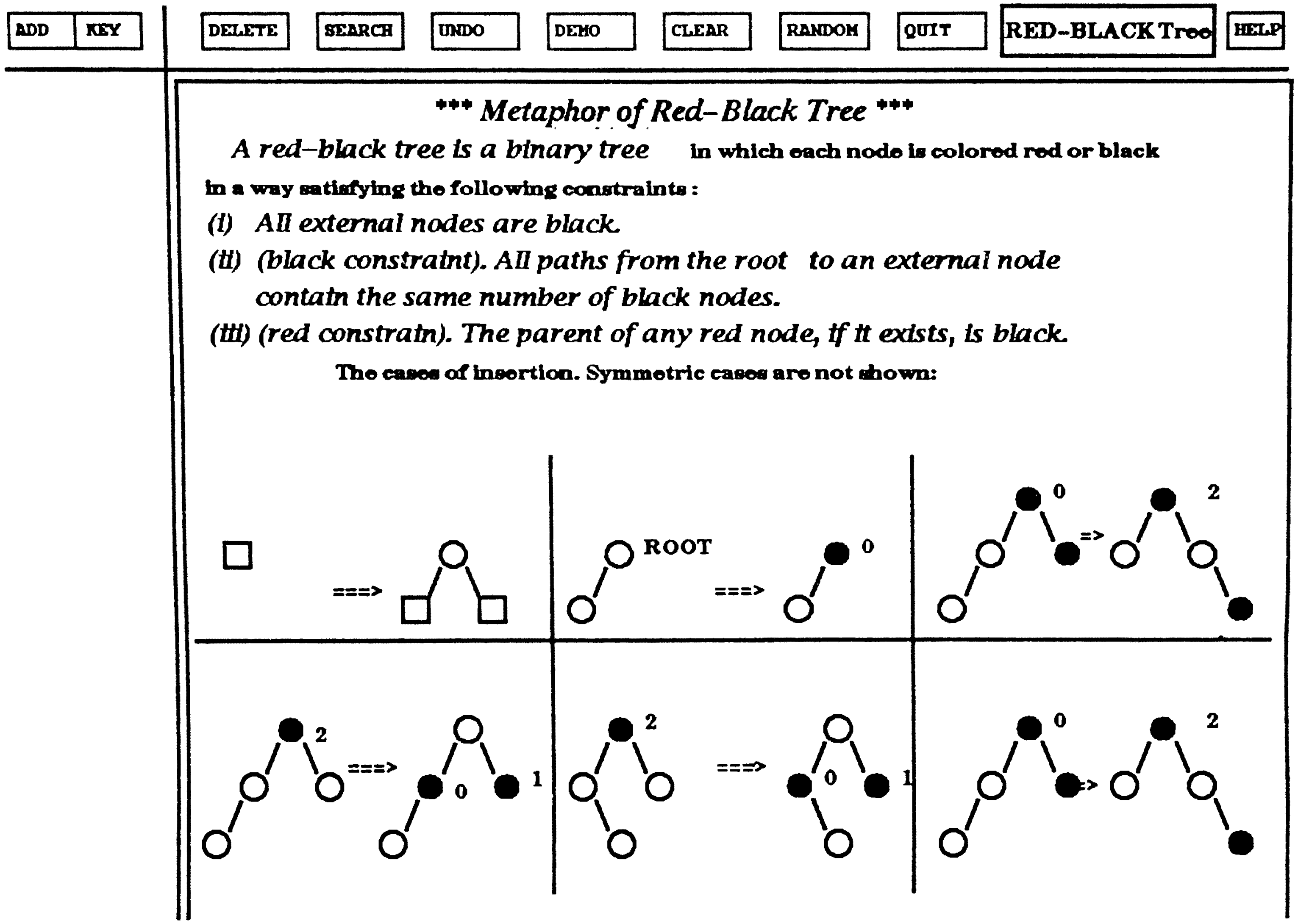
```
FUNCTION: insert (CURRENT_RRN, KEY PROMO_R_CHILD, PROMO_KEY)

   if CURRENT_RRN = NIL then   /* past bottom of tree */
       PROMO_KEY := KEY
       PROMO_R_CHILD := NIL
       return PROMOTION      /* promote original key and NIL */
   else
       read page at CURRENT_RRN into PAGE
       search for KEY in PAGE.
       let POS := the position where KEY occurs or should occur.

   if KEY found then
       issue error message indicating duplicate key
       return ERROR

   RETURN_VALUE := insert(PAGE.CHILD[POS], KEY, P_B_RRN, P_B_KEY)

   if RETURN_VALUE == NO PROMOTION or ERROR then
       return RETURN_VALUE

   elseif there is space in PAGE for P_B_KEY then
       insert P_B_KEY and P_B_RRN (promoted from below) in PAGE
       return NO PROMOTION
   else
       split(P_B_KEY,P_B_RRN,PAGE,PROMO_KEY,PROMO_R_CHILD,NEWPAGE)
       write PAGE to file at CURRENT_RRN
       write NEWPAGE to file at rrn PROMO_R_CHILD
       return PROMOTION /* promoting PROMO_KEY and PROMO_R_CHILD */
   endif

end FUNCTION
```

Figure 24. B Tree - 3



3 is inserted now.
(((13)5(78)B(CE))F((GJ)Q(UVW)Z(dfhj)))
Previous Tree:
((1578)B(CE)F(GJ)Q(UVW)Z(dfhj))

Figure 25. B Tree - 4



*** *Metaphor of Red-Black Tree* ***

*A red-black tree is a binary tree* **in which each node is colored red or black in a way satisfying the following constraints :**

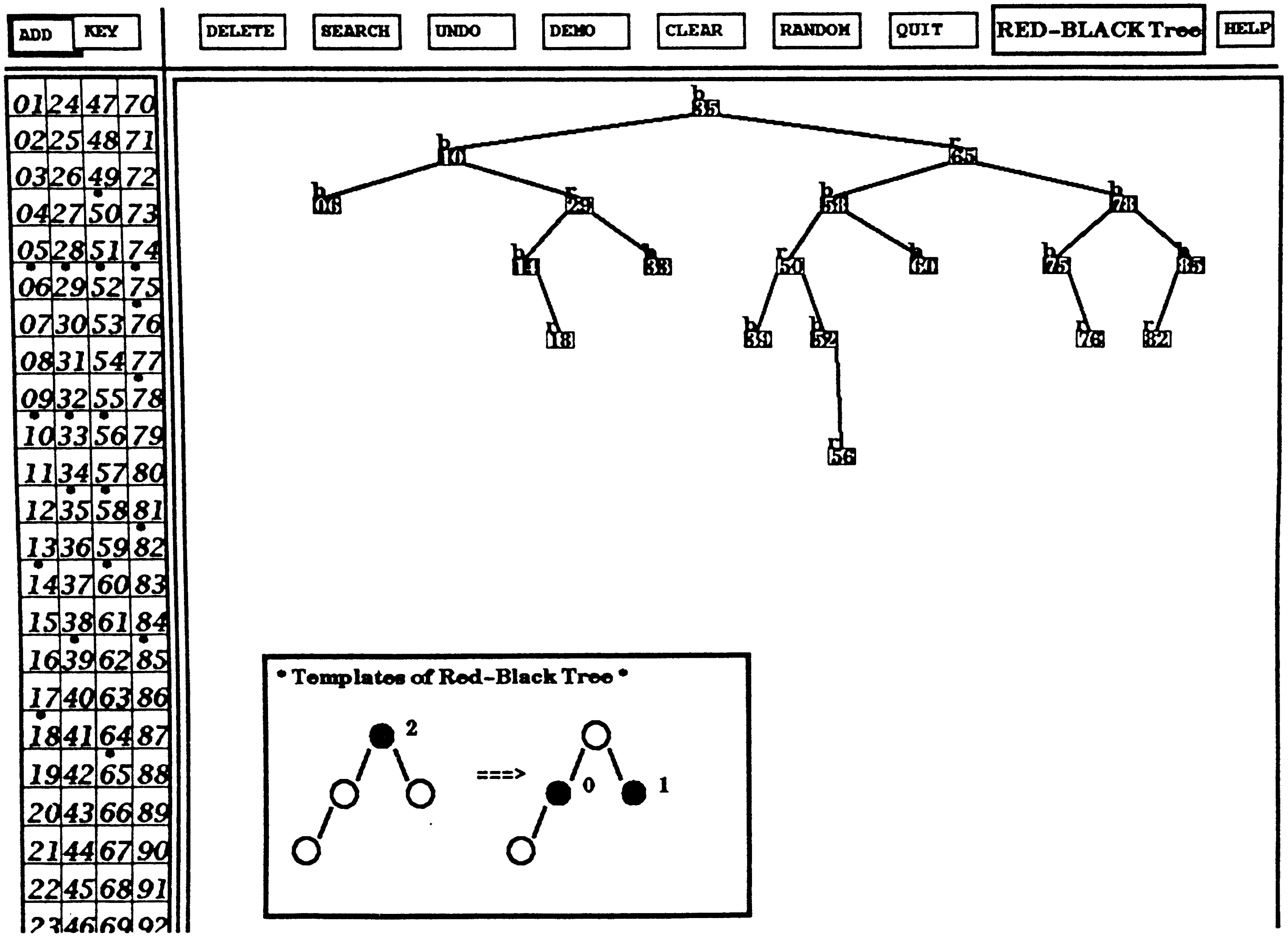
*(i) All external nodes are black.*

*(ii) (black constraint). All paths from the root to an external node contain the same number of black nodes.*

*(iii) (red constrain). The parent of any red node, if it exists, is black.*

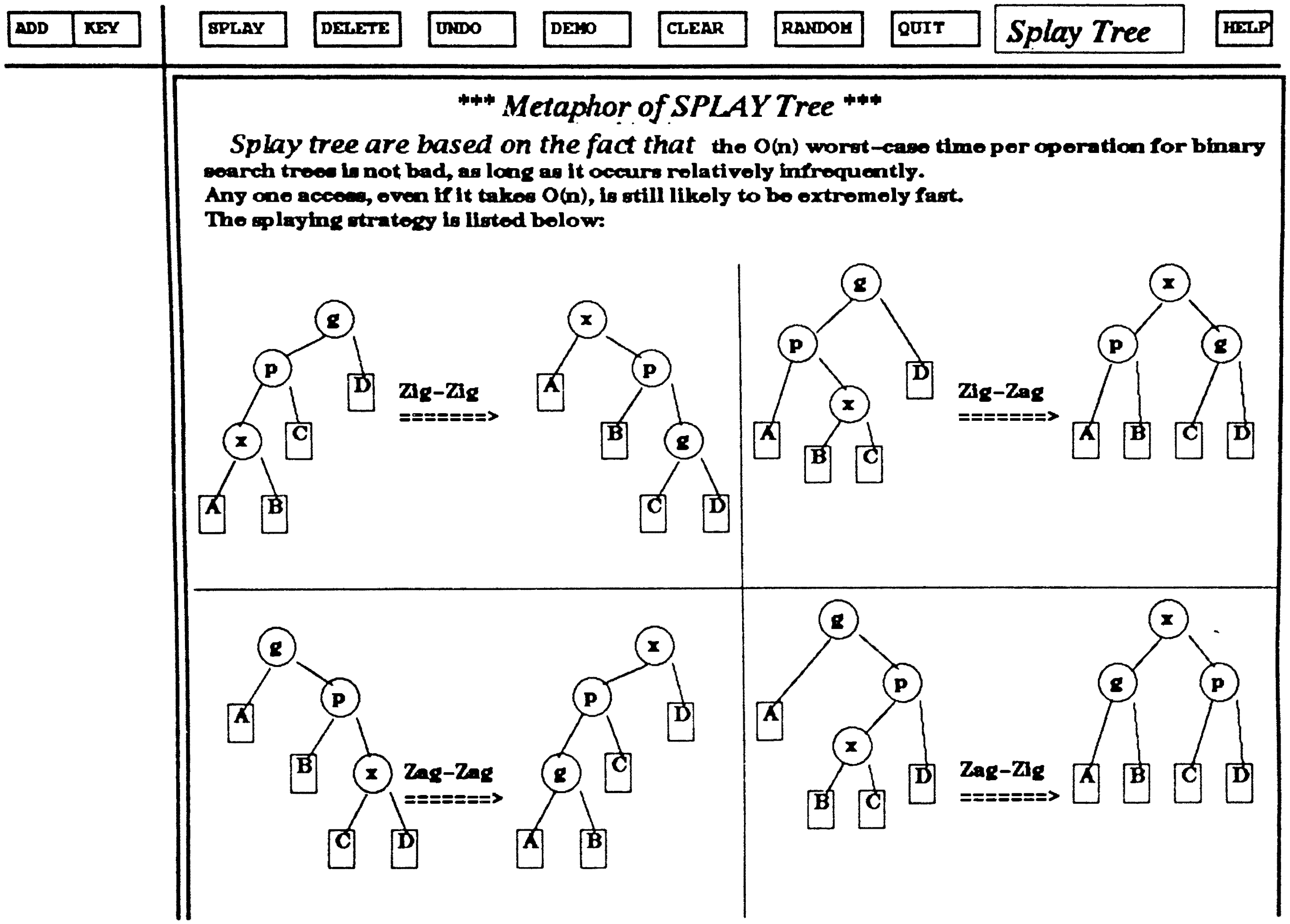
**The cases of insertion. Symmetric cases are not shown:**

Figure 26. Red-Black Tree - 1



| | | | |
|---|---|---|---|
| 01 | 24 | 47 | 70 |
| 02 | 25 | 48 | 71 |
| 03 | 26 | 49 | 72 |
| 04 | 27 | 50 | 73 |
| 05 | 28 | 51 | 74 |
| 06 | 29 | 52 | 75 |
| 07 | 30 | 53 | 76 |
| 08 | 31 | 54 | 77 |
| 09 | 32 | 55 | 78 |
| 10 | 33 | 56 | 79 |
| 11 | 34 | 57 | 80 |
| 12 | 35 | 58 | 81 |
| 13 | 36 | 59 | 82 |
| 14 | 37 | 60 | 83 |
| 15 | 38 | 61 | 84 |
| 16 | 39 | 62 | 85 |
| 17 | 40 | 63 | 86 |
| 18 | 41 | 64 | 87 |
| 19 | 42 | 65 | 88 |
| 20 | 43 | 66 | 89 |
| 21 | 44 | 67 | 90 |
| 22 | 45 | 68 | 91 |
| 23 | 46 | 69 | 92 |

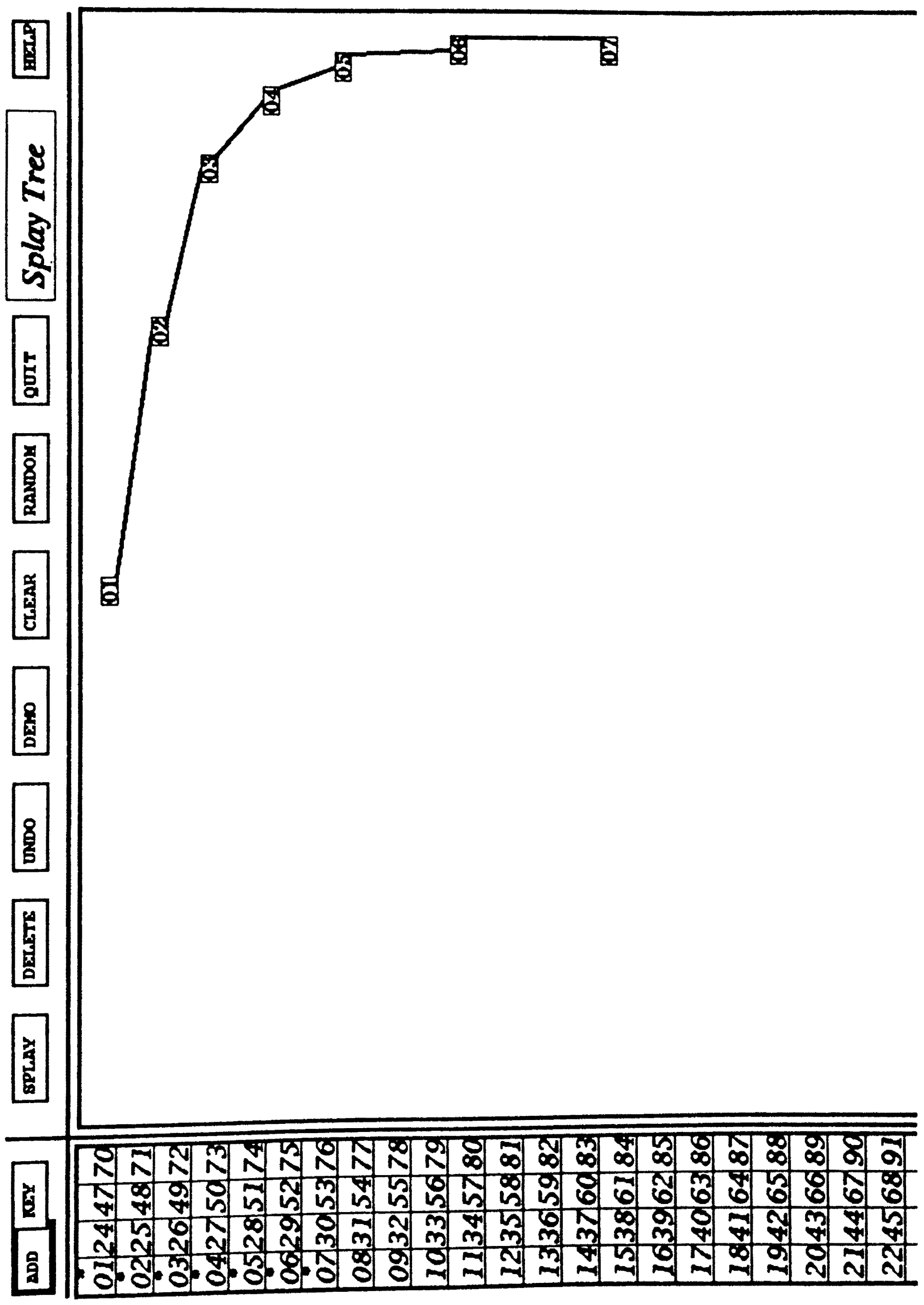
Figure 27. Red-Black Tree - 2



*** Metaphor of SPLAY Tree ***

*Splay tree are based on the fact that* the O(n) worst-case time per operation for binary search trees is not bad, as long as it occurs relatively infrequently.
Any one access, even if it takes O(n), is still likely to be extremely fast.
The splaying strategy is listed below:

Figure 28. Splay Tree - 1



| | | | |
|---|---|---|---|
| 01 | 24 | 47 | 70 |
| 02 | 25 | 48 | 71 |
| 03 | 26 | 49 | 72 |
| 04 | 27 | 50 | 73 |
| 05 | 28 | 51 | 74 |
| 06 | 29 | 52 | 75 |
| 07 | 30 | 53 | 76 |
| 08 | 31 | 54 | 77 |
| 09 | 32 | 55 | 78 |
| 10 | 33 | 56 | 79 |
| 11 | 34 | 57 | 80 |
| 12 | 35 | 58 | 81 |
| 13 | 36 | 59 | 82 |
| 14 | 37 | 60 | 83 |
| 15 | 38 | 61 | 84 |
| 16 | 39 | 62 | 85 |
| 17 | 40 | 63 | 86 |
| 18 | 41 | 64 | 87 |
| 19 | 42 | 65 | 88 |
| 20 | 43 | 66 | 89 |
| 21 | 44 | 67 | 90 |
| 22 | 45 | 68 | 91 |

Figure 29. Splay Tree - 2

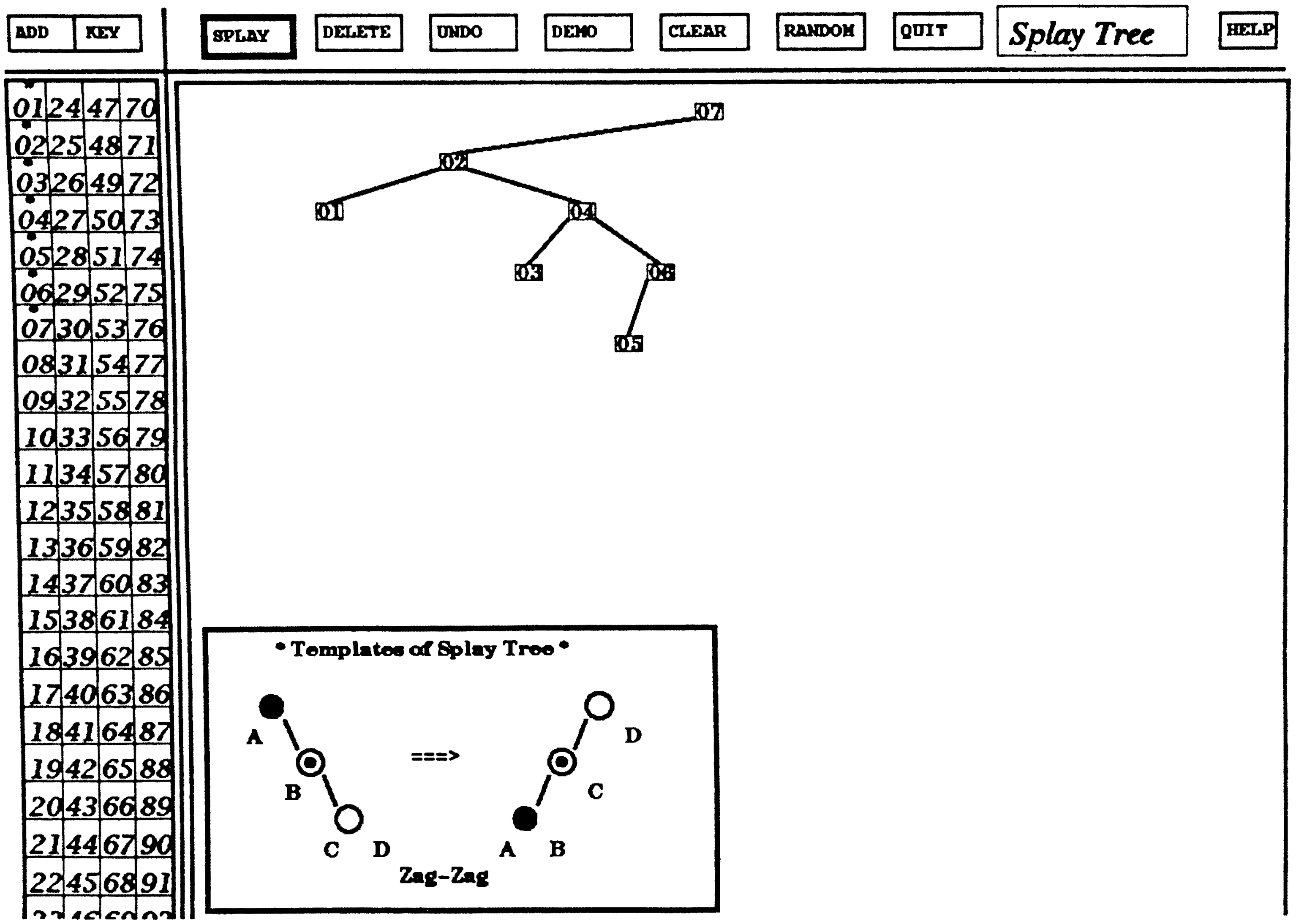


Figure 30. Splay Tree - 3

# CHAPTER VI

# SUMMARY AND FUTURE WORK

Due to the advance of technology in computer graphics and the advent of windowing techniques, visualization is applied immensely in every area of science and engineering.

In view of the fact that teaching and learning of data structures and algorithms in the classroom is a process that takes much time and is not effective sometimes, many systems for data structures and algorithms visualization have been invented.

In this study, using a systematic design process we have implemented a visualization system called TBDSV system. Since the first AVL tree visualization is finished, due to the design process, it only takes very limited time to develop the following Red-Black tree's, B-tree's, and Splay tree's visualization.

The TBDSV system is easy to use. We devise the strategy of modified Bresenham's Line Drawing algorithm to achieve the animation pictures. Using only integer arithmetic, this animation algorithm is proved to be efficient, and the animation pictures which it produces are very smooth.

The source code for this TBDSV system is available through the department of Computer Science. Information on the source code can be obtained by sending a request by e-mail to the address: kmg@a.cs.okstate.edu.

Due to the limited time, this system primarily focused on the visualization of AVL tree, Red-Black tree, B tree, and Splay tree algorithms. In addition to those tree-based algorithms, there are still many kinds of algorithms that need to be visualized if we want to ease the learning of them such as the sorting algorithms, the searching algorithms, the string processing algorithms, the graph algorithms, the geometric algorithms, and the mathematical algorithms. Visualizing other algorithms are considered future work.

## REFERENCES

[1] Atkinson, M. P., Baily, P. J., Chisholm, K. J., Cockshott, W. P. and Morrison, R. "The Persistent Object Management System," Soft. Pract. Experience, Vol 13, pp. 56-72, (1983).

[2] Brown, M. H., Algorithm Animation, The MIT Press, (1987).

[3] Brown, M. H.; Hershberger, J., Color and Sound in Algorithm Animation, IEEE Computer Graphics and Applications, Vol 12, pp. 52-63, (1992).

[4] Davis, P.J. "Visual Geometry, Computer Graphics and Theorems of Perceived Type," presented at the Missoula Conf. on the Influence of Computing on Mathematical Research and Education, August, (1973).

[5] Eades, P.; Tamassia, R., Algorithms For Drawing Graphs: An Annotated Bibliography, Unpublished Technical Report, Brown University, Department of Computer Science, (1989).

[6] Folk, M,J. and Zoellick, B., File Structures, Addison-Wesley Pub. Co., (1992).

[7] Gershon, N.D. From Perception To Visualization, Computer Graphics, Vol 27, pp. 414-417, (1992).

[8] Hearn, D. and Baker, M.P., Computer Graphics, Prentice-Hall, Inc., Englewood Cliffs, N.J., (1986).

[9] Horowitz, E.; Shani, S., Fundamentals of Data Structures in Pascal, Pitman Pub. Ltd, pp. 226-294, (1984).

[10] House, W.C., Interactive Computer Graphics Systems, Petrocelli Books, Inc, (1982).

[11] Lee, W. "An Implementatation of A Data Structures Display System," Unpublished Master's thesis, Oklahoma State University, (1988)

[12] Litwinowicz, P.C., Inkwell: A 2.5-D Animation System, Computer Graphics, Vol 25, pp. 113-122, (1991).

[13] London, R. L.; Duisberg, Animating Programs Using Smalltalk, Computer, Vol. 18, pp. 61-71, (Aug, 1985).

[14] Marcus, A.; Dam, A. V., User-Interface Developments for the Nineties, Computer, Vol 24, pp. 49-57, (1991).

[15] Mendez, R.H., Visualization in Supercomputing, Springer-Verlag, (1990).

[16] Myers, E. W. Efficient Applicative Data Types. In Conference Record Eleventh Annual ACM Symposium on Principles of Programming Languages, pp. 66-75. 1984.

[17] Retting, M., Interface Design When You Don't Know How, Communications of the ACM, Vol 35, pp. 29-34, (1992).

[18] Shimomura, T.; Isoda, S., Linked-List Visualization for Debugging, IEEE Software, Vol 17, pp. 44-51, (1991).

[19] Sommerville, I., Software Engineering, Addison-Wesley Pub. Co., (1992).

[20] Stasko, J. T., Tango: A Framework and System for Algorithm Animation, Computer, Vol 23, pp. 27-38, (1990).

[21] Thalmann, D., Scientific Visualization and Graphic Simulation, John Wiley & Sons, (1990).

[22] Zernik, D,; Snir, M.; Malki, D., Using Visualization Tools To Understand Concurrency, IEEE Software, Vol 18, pp. 87-92, (1992).

# APPENDIX

# USER'S MANUAL

## Main Menu

The TBDSV system has full mouse support. Once the system has been running, the main menu is as figure 15 shows. And you will be asked to select the buttons by mouse. They are:

| | |
|---|---|
| **<u>AVL</u>** : | Click this button to observe AVL tree. |
| **<u>B</u>** : | Click this button to observe B-tree. |
| **<u>Red-Black</u>** : | Click this button to observe Red-Black tree. |
| **<u>Splay</u>** : | Click this button to observe Splay tree. |
| **<u>Quit</u>** : | Click this button to leave TBDSV system. |
| **<u>Help</u>** : | Click this button to get textual explnation. |
| **<u>Ok</u>** : | When any button for tree is select, click this button to go into the tree's implementation. |
| **<u>Cancel</u>** : | Before you click OK button, you can use this button and select other tree again. |

## Tree Windows

When you go into one of the trees, you will have the following buttons for this tree's implementation.

They are:

| | |
|---|---|
| <u>ADD</u> : | When you click this button, you will have a table that contains many numbers, then click any number to insert key to this tree. |
| <u>Key</u> : | As the button of ADD, this button insert key in character but not digital. |
| <u>Delete</u> : | When you click this button, you will have a table that contains numbers or characters, then click any numbers or characters to delete key from this tree. |

**Search** : When you click this button, you will have a table that contains numbers or characters, then click any numbers or characters to search for that key in this tree.

**Undo** : Click this button to go back to the previous tree with one step. For example, when you insert a key to this tree, you can use undo to go back to the previous tree without the key inserted.

**Demo** : Click this button, and the system will show this tree's implementations automatically.

**Clear** : Click this buootn to clear screen and start again.

**Random** : Click this button to insert a number (from 0 to 99) to this tree randomly.

**Quit** : Click this button to go back to main menu.

**AVL T.** : In AVL tree, click this button to get the feature of AVL tree.

**Help** : Click this button to get textual explanation of every button.

[illegible]

VITA

Hung-Che Shen

Candidate of the Degree of

Master of Science

Thesis: A VISUAL AID FOR THE LEARNING OF TREE-BASED DATA STRUCTURES

Major Field: Computer Science

Biographical:

Personal Data: Born in Pingtung, Taiwan, October 4, 1967, the son of Zenyu Shen and Chufong Lin.

Education: Received Bachelor of Science Degree in Computer Science from Feng Chia University at Taichung, Taiwan in May, 1989; completed requirements for the Master of Science degree at Oklahoma State University in May, 1994.