Teaching of Tree Data Structures using Microcomputer Graphics

G. Scott Owen
Dept. of Mathematics and Computer Science
Georgia State University
Atlanta, Ga 30303

Abstract

A set of procedures to graphically display ordered and unordered trees has been developed. The procedures have been used in several class demonstration programs to illustrate tree insertion, deletion, and balancing algorithms. The procedures are available for inclusion in student programs so that they can determine if their programs are working correctly. The procedures are written in Turbo Pascal for an IBM PC.

Introduction

In our course on data structures stacks, queues, graphs, and trees are covered, with a heavy emphasis on trees. The students grasp the concept of stacks and queues easily as these are simple one dimensional structures, i.e. they just get bigger or smaller. However, it is much more difficult for them to visualize a tree structure as elements are added to and/or deleted from the tree.

A very sophisticated system for algorithm animation has been recently reported(1) but we do not have the necessary resources to use this system. Since our computer science program is based largely on microcomputers (IBM PC's and compatibles) it was decided to take advantage of the graphics cabability of these machines. Thus, procedures to graphically display binary ordered and unordered trees were developed in Turbo Pascal.

The procedures draw the trees with small rectangles representing the nodes and with edges connecting parent nodes to their children. There is space in the rectangles for a two or three character display of information. The procedure to display binary ordered trees, named GraphTree, was used in several demonstration programs to illustrate tree insertion, deletion, and balancing. The students incorporated this procedure into their own programs and were able to use it to quickly determine if their programs were working correctly. The procedure to display unordered trees, named GraphHeap, was used in demonstrating the construction of a heap from an array and in performing a heap sort. Both procedures are incorporated into programs by using the Turbo Pascal include file compiler directive, e.g. {$I Graftree.inc }, where Graftree.inc is a file containing the GraphTree procedure.

In the following I will discuss the two procedures GraphTree and GraphHeap, and give examples of their use.

GraphTree

The input to the GraphTree procedure consists of the pointer to the root node of the original tree, of type TreePtr. The tree nodes must have a field labeled InfoType, which contains the item to be displayed in the tree graph boxes. This type should be three characters or less to fit into the tree boxes. An example might be the declaration: InfoType = String[2]. When invoked, GraphTree creates a new tree with pertinent graphics information, displays it, and disposes of it when the user is finished viewing the tree.

The graphics tree has, in addition to the InfoType and left and right children pointer fields, two additional fields with graphics information. The first field is the node level with the root having a level value of one, it's children at level two, etc.. The second field is the position of the node at that level, starting at zero and incrementing from left to right. For example, the root has

position zero, it's left child position zero, and it's right child position one. On the third level the positions vary from zero on the far left to three on the far right.

The procedure only displays five levels even though the tree can have up to eleven levels. For any levels past five the boxes and edges are not drawn but the Info field is displayed on the bottom line of the screen. The graph tree declarations are included in Listing 1 and a full five level tree and the associated position numbers is given in Figure 1.

Creation of the Graphics Tree

The Graphics Tree is created in the procedure PreTravBuild. This is a recursive procedure which performs a pre-order traversal of the original tree. As each node of the original tree is encountered the corresponding node of the graphics tree is created. In the procedure GetNode each graph node has it's Info field initialized to the value of the Info field of the original tree node, it's Level field is initialized to one (the root level) and it's Position field is initialized to zero.

In the recursive procedure AddNode the Level and Position fields of the graph tree node are determined. When the procedure AddNode is reentered if the correct insertion spot has not yet been found then the Level field is incremented. If the search moves to the right then the old Position is multiplied by two and one is added, and for a left move the old Position is just multiplied by two.

Plotting of the Graphics Tree

The graphics tree is plotted in the procedure PreOrderGraph, which performs a recursive pre-order traversal of the tree and draws the rectangles and parent-children edges. The graphics mode used has a resolution of 640 x 200 pixels and the root is always centered. Thus, the root box is located at X = 320 and Y = 0. On the next level the boxes are at positions 160, 40 and 480, 40. For the third level the positions are (80, 80), (240, 80), (400, 80), and (560, 80). The X and Y positions at any level are given by the equations

$$X = 320 \times 2^{(Level-1)} + Position \times 2 \times 320 \times 2^{(Level-1)}$$

$$Y = 40 \times (Level - 1).$$

In the program the Turbo Pascal shift right (shr) operation is used instead of exlicitly raising two to the power of (Level - 1).

The boxes and connecting lines are drawn in the procedure DrawEdgeAndRectangle. A vertical fudge factor (Yfudge) is used to correctly align the characters in the boxes.

After the user has finished viewing the graphed tree it is destroyed by using a post-order traversal in the procedure PosTravDispose.

Sample Programs

The GraphTree procedure was incorporated into several class demonstration programs. The importance of maintaining a balanced tree was emphasized and the use of GraphTree enabled the students to quickly grasp the effect of insertions and deletions in a tree. An example is given in Figures 2, 3, and 4 which shows the two general ways to delete a node with two children, in this case the root node with Info value 'D'. Figure 2 shows the tree before any deletions and Figures 3 and 4 show the tree after the root node, containing 'D', is deleted.

In the first deletion method (Figure 3) the root node is not actually deleted. The tree is searched for the node with an Info field value which is the immediate predecessor of the node to be deleted and the Info field of the root node is replaced with this value. The immediate predecessor node is then deleted. In the second deletion method (Figure 4) the root node is actually deleted. The left subtree of the root node is attached to the root node's right subtree and then the root node is deleted. It can be easily seen from Figures 3 and 4 that the second deletion algorithm unbalances the tree more than the first algorithm.

GraphTree was also used to demonstrate the tree balancing algorithm of Kruse(2). This algorithm builds a new balanced tree from the old tree and then destroys the old tree.

The general principles of AVL trees are covered in the course, but not the detailed algorithms. GraphTree is used to demonstrate the rotations that occur in the AVL trees with insertions and deletions of nodes. An example using AVL trees is given in Figures 5 and 6. Figure 5 is the initial AVL tree and figure 6 shows the tree after insertion of the node with an Info value of 'ta'.

GraphHeap

A second application of this technique was in displaying an unordered binary tree. This was used in a demonstration of the Heap Sort. The program first generated an array of random integers and the resulting non heap tree was displayed. Then as the heap was being created the tree was displayed for each pass through the array. Finally, the tree was displayed at each step of the sort.

Since the GraphTree procedure was designed to create and display an ordered binary tree, a slightly modified version, called GraphHeap, was written to display the unordered trees involved in the heap sort process. Whereas the original tree root was passed to GraphTree, the array of integers was passed to GraphHeap. The only other difference between GraphTree and GraphHeap was in the procedure to build the tree, called RebuildTree in the GraphHeap procedure.

RebuildTree consists of a for loop which goes through all of the array elements. It calls two procedures, GetNodeFromArray and AddNode. The procedure GetNodeFromArray performs the same function as GetNode in GraphTree except that the Info field of the node takes its value from the data array.

The procedure AddNode determines the Level and Position fields of the new node and also makes the parent of the new node point to the new node. In converting an array into a tree the first element is the root, the next is the left child of the root, then the right child of the root, then the left child of the first left child, etc. So for any array element the corresponding tree level is one plus the highest power of two that divides the array index (computed in the function Power2). For example, array element number seven is the right child of the right child of the root. This places it on level three. The highest power of two which divides seven is two which plus one equals three.

The node position is the array index mod the index raised to the highest power of two that divides the index (computedin Exp2). For example, array element four has position zero (four mod four), element five has position one, element six has position two, and element seven has position three.

The final task of AddNode is to determine the correct parent of the new node and whether the new node is a left or right child. The array index of the left child of a node with index I is two x I and the right child is (two x I) + one. Therefore, the parent of a node with array index I is I div two. If I is odd then it is a right child and if I is even then it is a left child.

One or Two Monitors

There are two slightly different versions of the above procedures. My IBM PC has two monitors, a monochrome monitor and a color graphics monitor. Therefore, I use one for text and the other for graphics. To switch between the two from inside a program the public domain procedure SwitchAdapter, which is included in the procedure GraphTree listing, was used. In the classroom there was one large display screen attached to a PC with a graphics card, so the SwitchAdapter commands were commented out for classroom use.

Conclusion

Procedures have been developed to graphically display ordered binary search trees and unordered binary trees derived from arrays. These procedures were used in classroom demonstrations and incorporated into student programs. The use of these procedure allowed the students to quickly view the results of various tree operations such as insertions, deletions, tree rebalancing, building a heap and performing a heap sort. The student response to these procedures was very positive.

These procedures can be easily adapted to other microcomputer graphics systems, e.g. the Apple II Pascal system. The procedures can be modified to display more levels by changing the Height constant in PreOrderGraph. This may require the changing of the character fudge factor to align the characters in the boxes. Complete Tree programs using these procedures are available from the author.

References

1.) M. H. Brown and R. Sedgewick, "Techniques for Algorithm Animation", IEEE Software, Vol. 2, No. 1, p. 28, (January, 1985).

2.) R. L. Kruse, "Data Structures & Program Design", Prentice Hall, 1984.

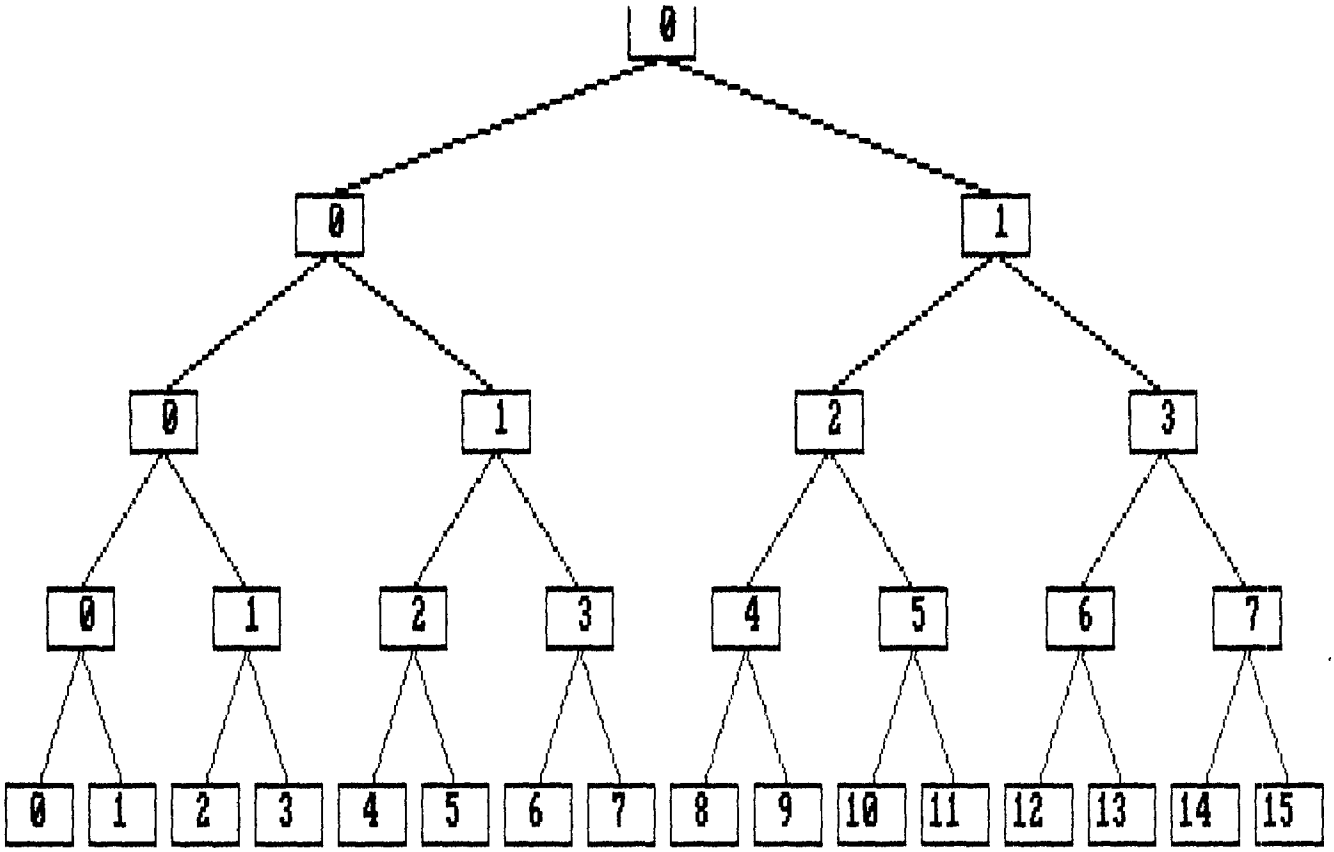

Figure 1

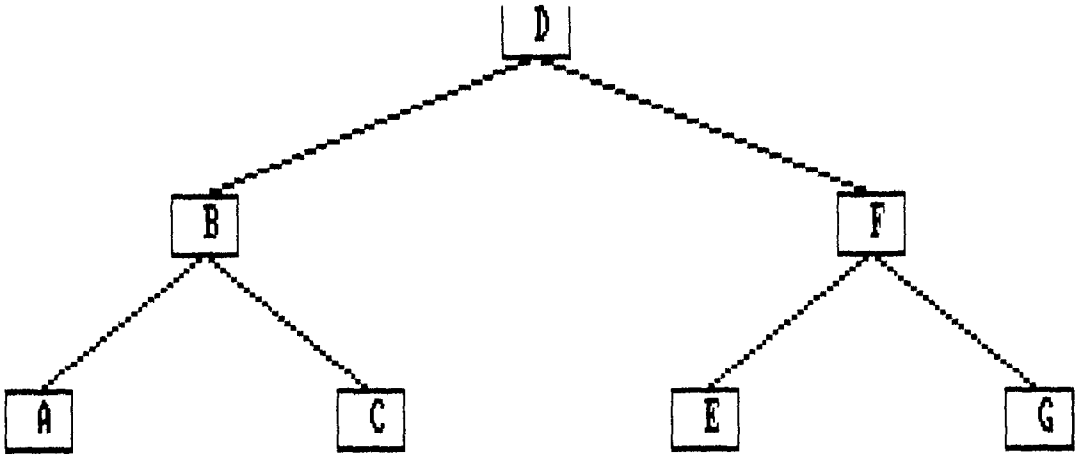

Figure 2

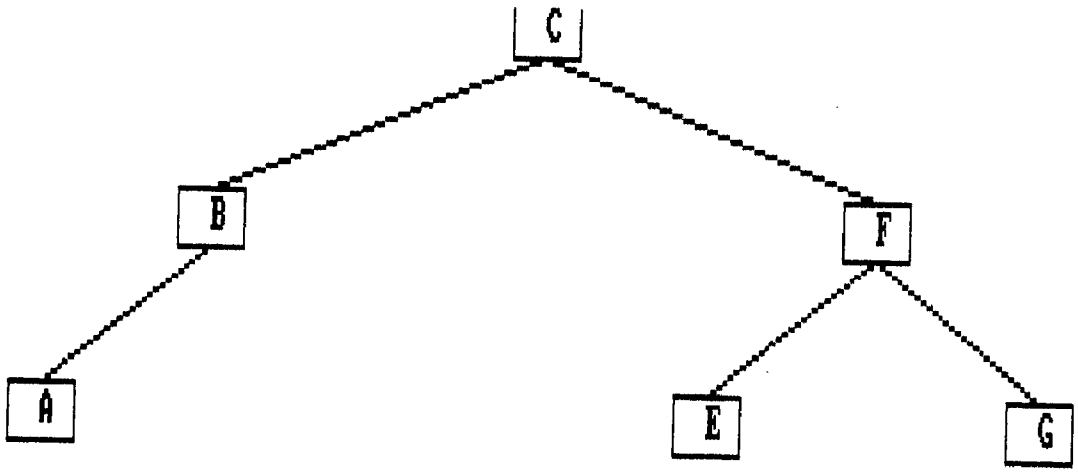

Figure 3

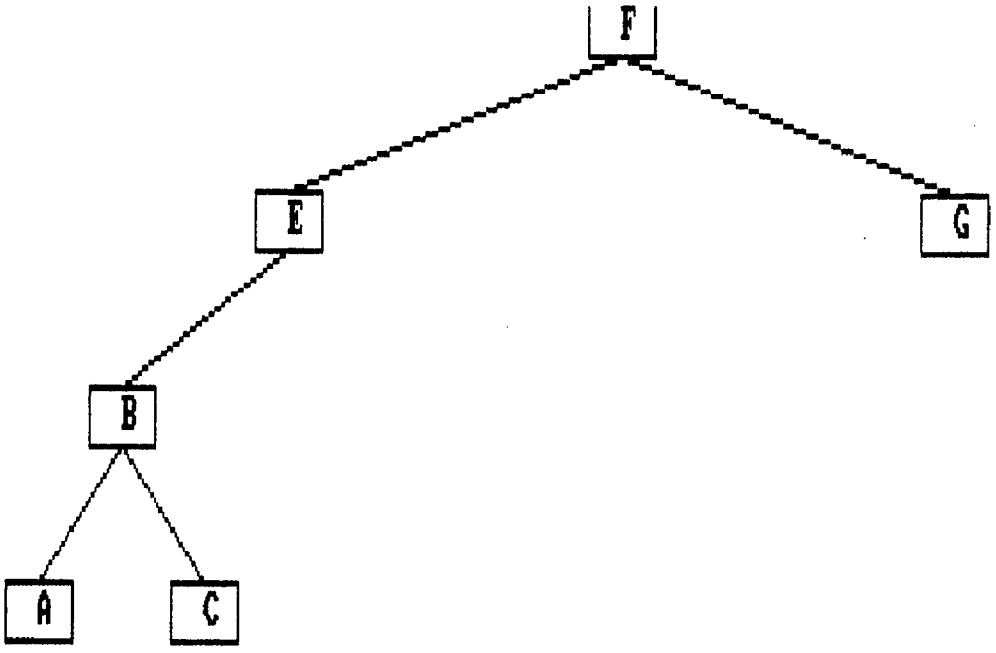

Figure 4

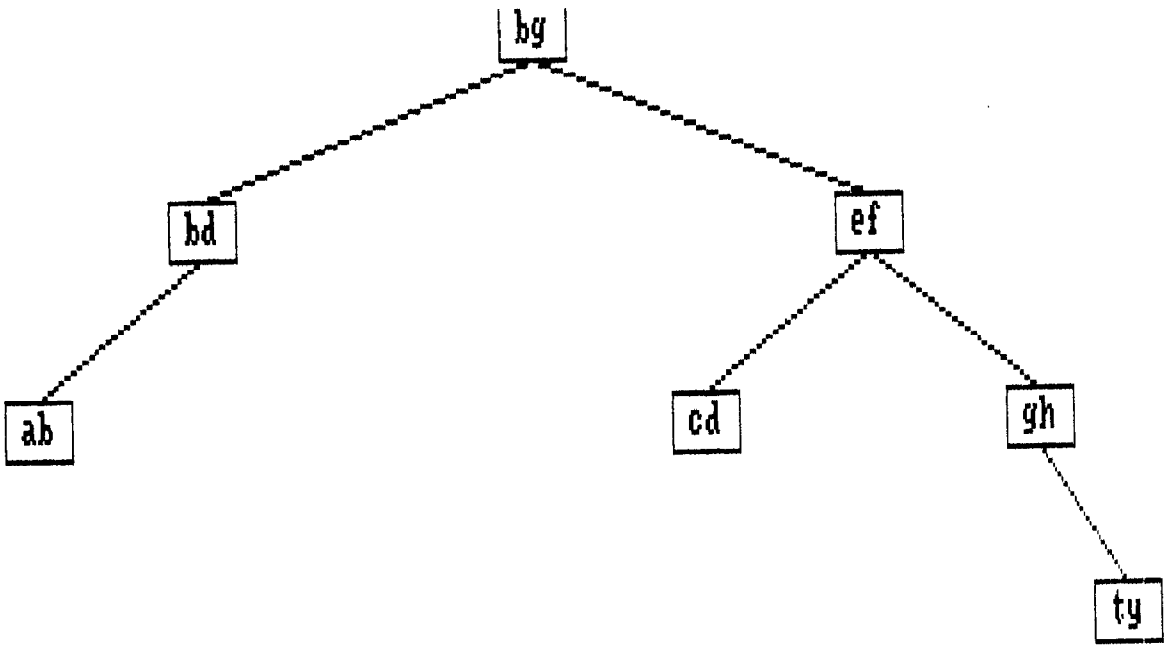

Figure 5

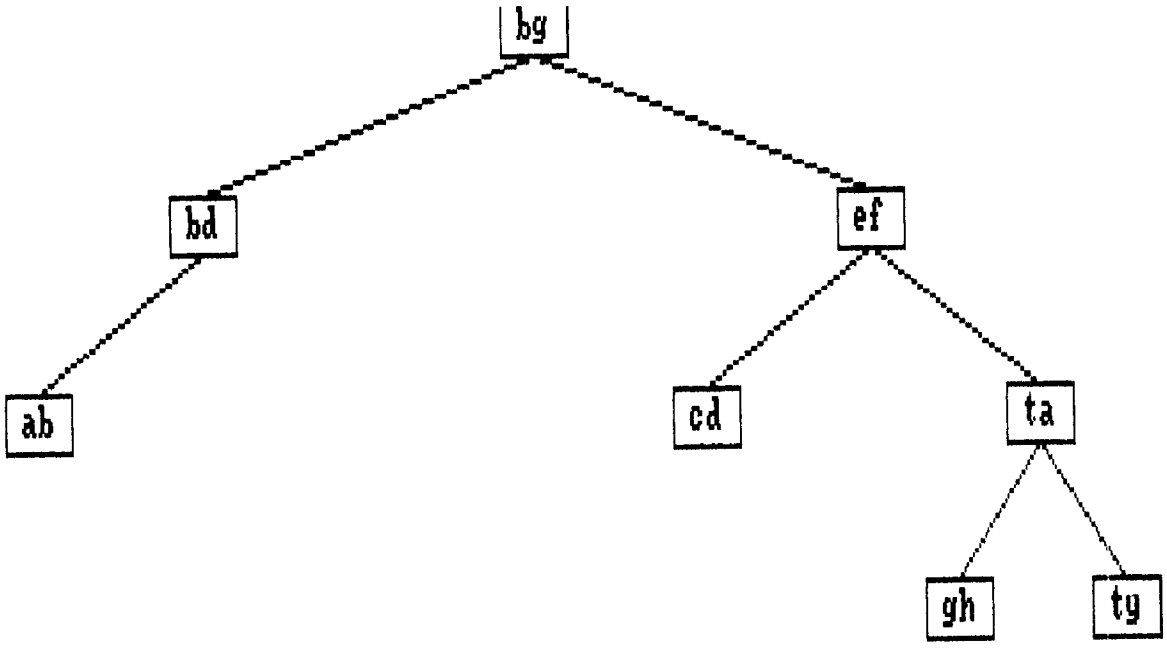

Figure 6