Laboratory practice No. 4: Hash Tables and Trees

Mariana Quintero Montoya
Universidad Eafit
Medellín, Colombia
mquintero3@eafit.edu.co

Isabella Pérez Serna
Universidad Eafit
Medellín, Colombia
iperezs2@eafit.edu.co

3) Practice for final project defense presentation

3.1

To solve this exercise, a data structure called octree is used, which is based on a tree with 8 secondary nodes at each node, through a list of 8-place matrices, where each position is a linked list. The method that calculates the collisions receives one of these linked lists as a parameter and iterates each element through a loop, so its complexity is O (n), where n is the number of elements in the list.

3.2 [OPC]

3.3 [OPC]

2.1) In this class is the method of insertion in the binary tree, which begins from the root node and evaluates that the smaller or equal value will be to the left and the greater value will be to the right, this is done through the recursion until arriving to an empty node, so the value that passes the parameter becomes the new node. Then, to print in posOrder through recursion, first print the left sub-tree, then print the right sub-tree, and finally print the root.

3.4

3.5

2.1) n is the number of nodes that the binary tree has.

4) Practice for midterms

4.1

4.1.1 B
4.1.2 D

4.2 [OPC] C

4.3

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

Line 3: return false;
Line 5: return suma == a.value;
Line 7: return sumaElCamino(a.left, suma)
Line 8: | | sumaElCamino(a.rigth, suma);

4.4 [OPC]
4.4.1 B
4.4.2 A
4.4.3 D
4.4.4 A

4.5 [OPC]
Line 4: if (p == toInsert)
Line 6: if (p > toInsert)

4.6 [OPC]
4.6.1 D
4.6.2 return 0;
4.6.3 if(raiz.hijos.size() > 0) // Si suma es 0

4.7 [OPC]
4.7.1 A
4.7.2 B
4.7.3 D

4.8 [OPC]

4.9 A

4.10 Empty

4.11 [OPC]
4.11.1 B
4.11.2 A
4.11.3 B

4.12 [OPC]
4.12.1 J
4.12.2 A
4.12.3 B

4.13
4.13.1 suma[raíz.id] = suma[e.id] + suma[raíz.id];

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

4.13.2 A

5) Recommended reading (optional)

Chapter 8: Binary Tree

Why use binary trees?

You can search for a tree quickly, as you can do an ordered array, and you can also insert and delete items quickly, as you can do with a linked list.

Slow insertion into an ordered matrix

As we learned, it is quick to search for such an array for a given value, using a binary search. The application of this process repeatedly finds the object in O time. It is also fast to go through an ordered array in iteration, visiting each object in order. On the other hand, if you want to insert a new object in an ordered array, you must first find where the object will go and then move all the objects with major keys one space in the array to make room for it. If you are going to do a lot of inserting and deleting, an ordered array is a bad choice.

Slow search in a linked list

On the other hand, as we saw in Chapter 7, "Advanced Sorting", insertions and deletions are quick to make in a linked list. Unfortunately, finding a specified item in a linked list is not so easy. You must start at the beginning of the list and visit each item until you find the one you're looking for. You might think that you could speed things up by using an ordered linked list, where items are organized in order, but this doesn't help.

Trees to the rescue

It would be good if there was a data structure with the quick insertion and removal of a linked list, and also the quick search of an ordered matrix.

What is a tree?

We'll be interested primarily in a particular type of tree called a binary tree, but let's start by discussing trees in general before moving on to the details of binary trees. A tree consists of nodes connected by edges. In such a tree image, the nodes are represented as circles and the edges as lines that connect the circles.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

Trees have been extensively studied as abstract mathematical entities, so there is a lot of theoretical knowledge about them. A tree is actually an instance of a more general category called a graph, but we don't have to worry about that here. Usually there is a node in the top row of a tree, with lines that are connected to more nodes in the second row, even more in the third, and so on. So, the trees are small at the top and large at the bottom.

This may seem upside down compared to real trees, but usually a program starts an operation at the small end, and it is more natural to think about going up and down, like in text conferencing. There are different types of trees. Trees have more than two secondary elements per node. However, in this chapter we are going to discuss a specialized form of tree called a binary tree.

Each node in a binary tree maintains a maximum of two secondary elements. More general trees, where nodes can have more than two children, are called multimotion trees. A generally found tree is the hierarchical structure of files in a computer system. The root directory of a given device is the root of the tree. Clearly a hierarchical file structure is not a binary tree, because a directory can have many secondary elements. The terms used for file structure, such as root and path, were borrowed from tree theory. A hierarchical file structure differs significantly from the trees we will discuss here.

In a tree, each node contains data.

How do binary trees work?

Let's see how to carry out the common binary tree operations of finding a node with a certain key, inserting a new node, crossing the tree and removing a node.

The Applet of the Tree Workshop

However, because the tree in the Workshop applet will generate randomly, it will not look exactly like the tree in the figure.

Using the Applet

Of course, in a real tree, there would probably be a larger range of key values. This constraint ensures that all nodes in the tree will be visible on the screen. In a real tree the number of levels is unlimited. With the Workshop applet, you can create a new tree whenever you want.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

A message will ask you to enter the number of nodes in the tree. This can vary from 1 to 31, but 15 will give a representative tree. After entering the number, press Fill twice more to generate the new tree.

Unbalanced trees

If these key values are inserted randomly, the tree will be more or less balanced. However, if you generate an ascending sequence or a descending sequence, all values will be correct secondary or left secondary elements and the tree will be unbalanced. When you learn how to insert elements into the tree in the Workshop applet, you can try to create a tree by inserting an ordered sequence of elements and see what happens. If you request a large number of knots when you use Fill to create a tree, you may not get as many knots as you requested.

Depending on how unbalanced the tree is, some branches may not be able to contain a full number of knots. If a tree is created by data elements whose key values arrive in a random order, the problem of unbalanced trees may not be too much of a problem for larger trees, because when this happens, the efficiency of the tree can seriously degrade.

Tree representation in Java code

Let's see how we could implement a binary tree in Java. As with other data structures, there are several approaches to representing a tree in the computer's memory. It is also possible to represent a tree in memory as an array, with nodes in specific positions stored in the corresponding positions in the array.

The kind of tree

We will call this class Tree. You don't need fields for the other nodes because you access them all from the root.

The TreeApp class

Finally, we need a way to perform operations on the tree. This is how you can write a class with a main routine to create a tree, insert three knots in it, and then find one of them.

Find a node

Finding a node with a specific key is the simplest of the main tree operations, so let's start with that. Remember that the nodes in a binary search tree correspond to objects that contain information.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

Using the Applet Workshop to find a node

Look at the applet Workshop choose a node, preferably one near the bottom of the tree. Of course, when you run the Workshop applet you will get a different tree and you will have to choose a different key rating.

Java code to find a node

The key to the argument is the value to be found. The routine starts at the root. That is, it establishes in the current root. Continuously, in the while loop, it compares the value that is going to be found, key, with the value of the iData field in the current node.

If the key is smaller than this field, it is set to the left secondary element of the node. If the key is larger than the iData field on the node, it is set to the right subelement of the node.

Found It

We return the node so that the routine that called found can access any of the data of the node.

Java code to insert a node

The insert function starts by creating the new node, using the data provided as arguments. Then, insert must determine where to insert the new node. This is done using approximately the same code as finding a node. The difference is that when you simply bring it up against a node, you know that the node you are looking for does not exist, so you can return immediately.

When you try to insert a node, you insert it before returning. The value that you are going to look for is the data element passed in the argument identifier. We use a new variable, parent, to remember the last non-null node we found. This is necessary because it is set to null in the process of detecting that its previous value did not have a suitable secondary element.

To insert the new node, change the appropriate secondary pointer on the primary element to point to the new node.

Crossing the Tree

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

Crossing a tree means visiting each node in a specified order. But crossing a tree is useful in some circumstances and the algorithm is interesting. There are three simple ways to traverse a tree.

Inorder Traversal

It is called has a recursive method to go through the whole tree with a node as an argument. Initially, this node is the root. Calling itself to go through the right sub-tree of the node Traversals work with any binary tree, not just binary search trees.

Java code for traversals

The routine, inOrder, performs the three steps already described. The visit to the node consists in showing the content of the node. In inOrder this happens when the node passed as an argument is null.

Crossing a 3-node tree

In a binary search tree this would be the order of the ascending keys. The inOrder function calls itself for each node, until it has worked its way through the whole tree.

The Traversing applet with the Workshop

This is where it happens when you use the Tree Workshop applet to go through the tree in order. This is a little more than the 3-node tree seen above.

Traversing pre-order and post-order

It's pretty clear why you might want to go through a tree in order, but the motivation for pre-order and post-order tours is darker. A binary tree can be used to represent an algebraic expression involving the binary arithmetic operators, -, /, and *. Traversing the order of the tree will generate the correct order sequence A * BC.

* A + BC

This is called prefix notation. Inserting that into the original expression * A + BC gives us A * inorder. As described in chapter 4, "Stacks and Tails", it means "apply the last operator in the expression, *, to the first and second things". The first thing is A, and the second thing is BC +. BC + means "apply the last operator in the expression, to the first and second thing. The first thing is B and the second thing is C, so this gives us in infix. Inserting this in the original expression ABC + * gives us A * postfix. The code contains methods for pre-order and post-order paths, as well as for inorder.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

Searching for maximum and minimum values

We must take into account how easy it is to find the maximum and minimum values in a binary search tree. To obtain the maximum value in the tree, follow the same procedure, but go from the right child to the right child until you find a node without a suitable child. However, deletion is important in many tree applications, and studying the details creates character.

Case 1: The node to be eliminated has no children

The node will still exist, but will no longer be part of the tree. The node to be eliminated has a child, you want to "cut" the node out of this sequence by connecting its primary element directly to its secondary element. This involves changing the appropriate reference on the primary element so that it points to the secondary element of the removed node. Node 71 has only one child, 63. Its place is taken by its left occupied child, 63. In fact, the entire sub-tree of which 63 is the root moves up and connects as the new right child of 52. Find the subtree whose root is the secondary element of the deleted node. No matter how complicated this sub-tree is, it simply moves up and connects as the new secondary element of the primary element of the deleted node.

Note that working with references makes it easier to move an entire sub-tree. To do this, simply disconnect the old reference to the sub-tree and create a new reference to it elsewhere. Although there may be many nodes in the sub-tree, you don't have to worry about moving them individually. In fact, they only "move" in the sense of being conceptually in different positions in relation to the other nodes. As far as the program is concerned, only the reference to the root of the sub-tree has changed.

The node to be eliminated has two children

If the deleted node has two secondary elements, you cannot replace it with one of these secondary elements, at least if the child has its own secondary elements. Imagine eliminating node 25 and replacing it with its right sub-tree, whose root is 35. Remember that in a binary search tree, nodes are organized in order of ascending keys. For each node, the node with the next highest key is called its successor in order, or simply its successor.

Node 30 is the successor to node 25, a deleted node that is being replaced by its successor. Note that the nodes are still in order.

Finding the Successor

Just take a quick look at the tree and find the next largest number after the key of the node you are going to delete. The successor to 25 is 30. First, the program goes to the

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

right side element of the original node, which must have a larger key than the node. The last secondary left element in this path is the successor to the original node.

What we are really looking for is the smallest of the set of nodes that is larger than the original node. When you go to the right secondary element of the original node, all the nodes in the result from subtree are larger than the original node. This is how you define a binary search tree. If the right secondary element of the original node has no left secondary elements, this right secondary element is itself the successor.

You may want to make sure that the successor has no children of its own. If you do, the situation looks more complicated because whole sub-trees move around, rather than a single node. Once you have chosen a node to delete, click on the Del button. You will be asked for the key value of the node to be deleted.

When you have specified it, repeated clicks of the Del button will display the red arrow that searches the tree for the designated knot. When the node is deleted, it is replaced with a successor node. When the while loop is closed, the successor contains the delNode successor. Once we have found the successor, we may need to access its primary element, so inside the loop while also tracking the current node's primary element.

The getSuccessor routine performs two additional operations in addition to finding the successor. As we have seen, the successor node can occupy one of the two possible positions in relation to the current one, the node to be eliminated. The successor can be the right hand child of the current one, where it can be one of the left hand descendants of this right hand child.

Plug the right secondary element of the node to be deleted into the rightChild field of the successor.

Here is the code for these four steps

Step 1 replaces in effect the successor with its right sub-tree. Step 2 keeps the correct secondary element of the deleted node in its proper place. Steps 1 and 2 are carried out in the if statement that ends the getSuccessor method. These steps are more convenient to perform here than in removing, because in getSuccessor it is easy to find out where the parent of the successor is while we are descending the tree to find the successor.

Is elimination necessary?

The elimination of a node does not change the structure of the tree. This approach is a bit of a cop-out, but can be appropriate when there won't be many deletions in a tree.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

The efficiency of binary trees

Most tree operations involve descending the tree from level to level to find a particular node. In a complete tree, about half of the nodes are at the lowest level. Therefore, the time needed to carry out the common operations of the tree are proportional to the base register N-2. If the tree is not full, the analysis is difficult. We can say that, for a tree with a certain number of levels, the average search times will be shorter for the not full tree than the full tree because less search will proceed to lower levels.

For example, a tree with 1,000,000 elements requires 20 comparisons. Inserting an element into a tree with 1,000,000 elements requires 20 or fewer comparisons, plus a small amount of time to connect the element. Similarly, removing an element from a tree of 1. 000. 000 elements requires moving an average of 500. 000 elements, while removing an element from a tree of 1. 000. 000 nodes requires 20 or fewer comparisons to find the element, plus some more comparisons to find a successor, plus a short amount of time to disconnect the element and connect a successor.

Therefore, a tree provides high efficiency for all common data storage operations. They are most appropriate when a tree is used as an aid to analyze algebraic or similar expressions, which are probably not too long anyway.

Trees represented as matrices

Our code examples are based on the idea that the edges of a tree are represented by the references leftChild and rightChild in each node. The position of the node in the array corresponds to its position in the tree. The node in index 0 is the root, the node in index 1 is the secondary left element of the root, and so on, progressing from left to right along each level of the tree. Each position in the tree, regardless of what an existing node represents or not, corresponds to one cell in the array.

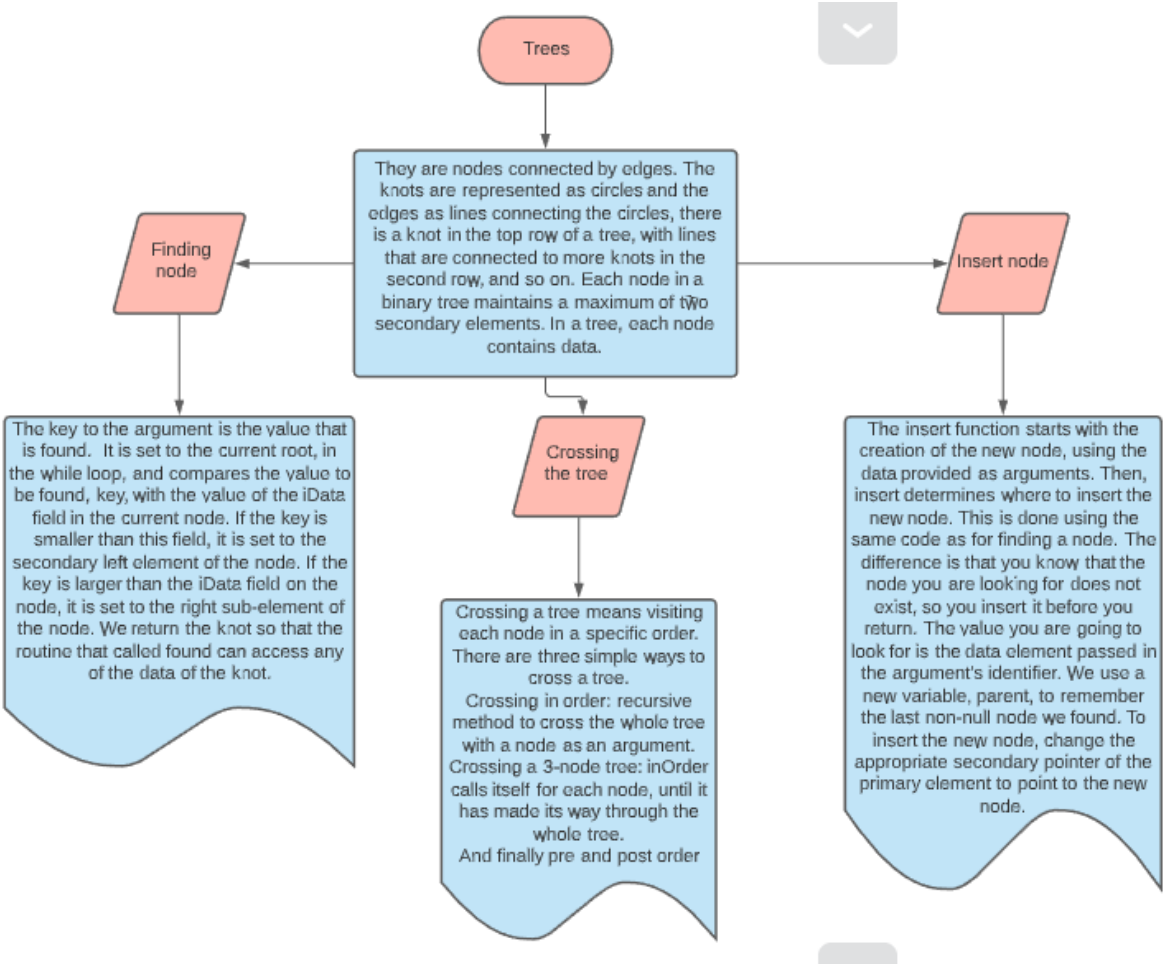
Adding a node at a given position in the tree means inserting the node into the equivalent cell in the array. Cells representing tree positions without nodes are filled with zero or null. In most situations, representing a tree with an array is not very effective. The tree in the Workshop applet, for example, is represented internally as an array to make it easier to assign the nodes in the array to fixed locations on the screen.

Duplicate keys

In the code shown for insertion, and in the Workshop applet, you will insert a node with a duplicate key as the correct secondary element of its twin. The problem is that the find routine will find only the first of two duplicate nodes. The find routine could be modified

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

to check an additional data element, to distinguish data elements even when the keys were the same, but this would take a lot of time.



PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473