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 nlog(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

2.1) nlog(n).

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

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

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

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

What is a tree?

They are nodes connected by edges. The knots are represented as circles and the edges as lines connecting the circles, there is a knot in the top row of a tree, with lines that are connected to more knots in the second row, and so on. Each node in a binary tree maintains a maximum of two secondary elements. In a tree, each node contains data.

Tree representation in Java code

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.

Find a node

Finding a node with a specific key is the simplest of the main tree operations, remember that the nodes in a binary search tree correspond to objects that contain information.

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

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.

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.

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.

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.

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 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. When you go to the right secondary element of the original node, all the nodes in the result from

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

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.

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.

Trees represented as matrices

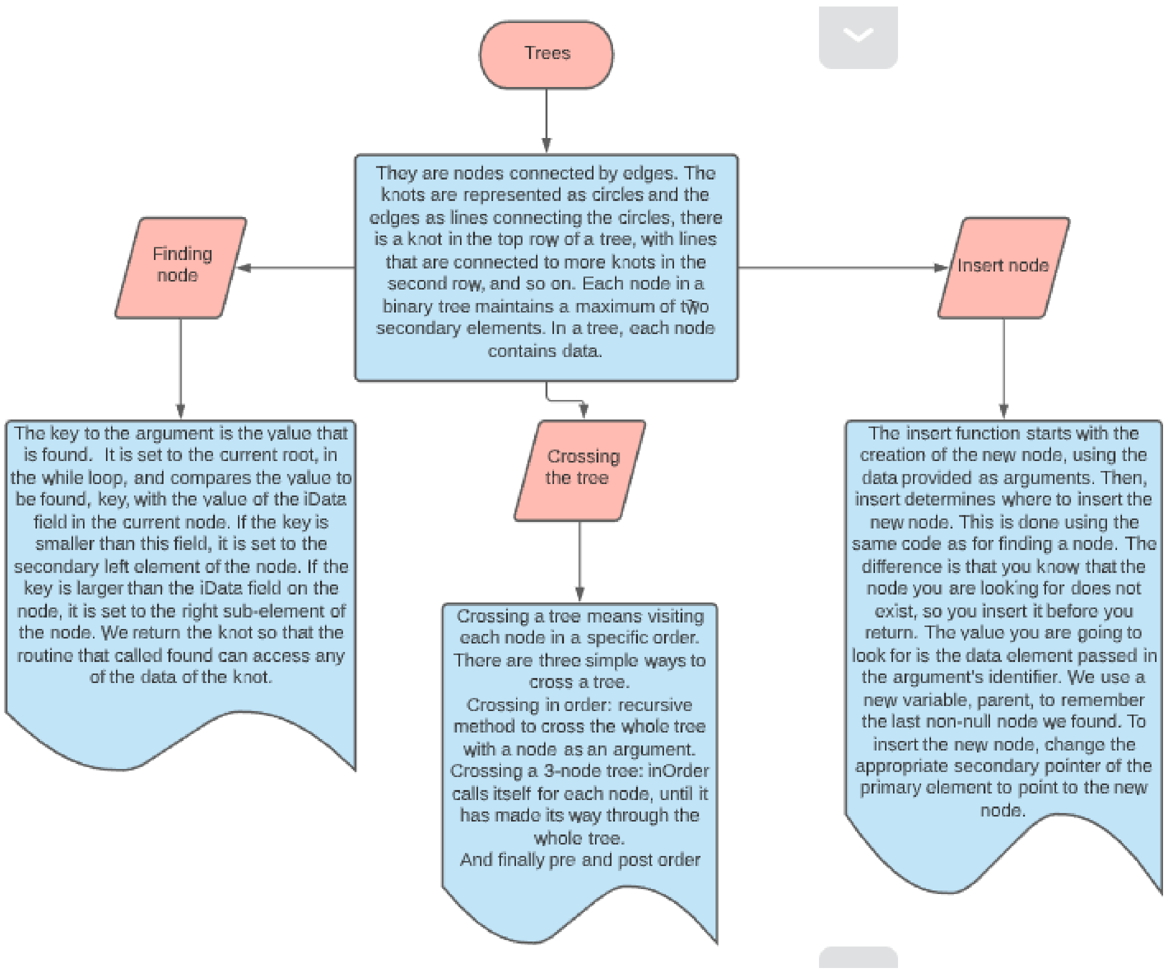
Our code examples are based on the idea that the edges of a tree are represented by the references leftChild and rightChild in each node. The position of the node in the array corresponds to its position in the tree. The node in index 0 is the root, the node in index 1 is the secondary left element of 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.

Duplicate keys

In the code shown for insertion, 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 to check an additional data

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

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


Trees
They are nodes connected by edges. The knots are represented as circles and the edges as lines connecting the circles, there is a knot in the top row of a tree, with lines that are connected to more knots in the second row, and so on. Each node in a binary tree maintains a maximum of two secondary elements. In a tree, each node contains data.
Finding node
Insert node
The key to the argument is the value that is found. It is set to the current root, in the while loop, and compares the value 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 secondary left element of the node. If the key is larger than the iData field on the node, it is set to the right sub-element of the node. We return the knot so that the routine that called found can access any of the data of the knot.
Crossing the tree
Crossing a tree means visiting each node in a specific order. There are three simple ways to cross a tree. Crossing in order: recursive method to cross the whole tree with a node as an argument. Crossing a 3-node tree: inOrder calls itself for each node, until it has made its way through the whole tree. And finally pre and post order
The insert function starts with the creation of the new node, using the data provided as arguments. Then, insert determines where to insert the new node. This is done using the same code as for finding a node. The difference is that you know that the node you are looking for does not exist, so you insert it before you return. The value you are going to look for is the data element passed in the argument's identifier. We use a new variable, parent, to remember the last non-null node we found. To insert the new node, change the appropriate secondary pointer of the primary element to point to the new node.

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