Deleting in a Binary Search Tree

When writing a delete function for a binary search tree, you can separate the task into two parts. The first part which is finding the item to delete can be done recursively just like a “find” function.

Find ( key, root )

if root is null

Key is not found

else if root->info is the key

Key is found

else if key < root->info

Find (key, root->leftPtr)

else

Find (key, root->rightPtr)

The difference is that when we find the node, we call a function to do the second part: delete the node form the tree.

Delete ( key, root )

if root is null

Key is not found

else if root->info is the key

DeleteNode (root)

else if key < root->info

Delete (key, root->leftPtr)

else

Delete (key, root->rightPtr)

So how do you write DeleteNode (root)? You need to take care of 3 cases. In the first two cases, either the left pointer is null or the right pointer is null. In these cases, it is exactly like deleting a node from a linked list. The following shows the case of left pointer being null.

root

root

root

12

12

9

deletePtr

12

deletePtr

9

9

root

deletePtr

12

9

What about the case where neither the left pointer or the right pointer is null? In this case, deleting the node will disturb the tree structure too much. What we will do is to replace the info field with another value already in the tree that will keep the binary search tree structure mostly intact.

Suppose we want to replace 17 in the following binary search tree. What value could you use that would leave the tree largely intact?

root



You can use the largest value in the left subtree (15) or the smallest value in the right subtree (21). What happens to the node the value came from? Notice that the node with the largest value in a subtree will have its right pointer be null and the node with the smallest value in a subtree will have its left pointer be null. Either node will be easy to delete.

Suppose we take the largest value in the left subtree to replace the value at the root. The tree will end up looking like the following.

root



11

15

Partial pseudocode for deleteNode:

deleteNode (root)

if (root-> leftPtr is null)

{ write code for sequence of drawings on page 1 and 2)

else if (root-> rightPtr is null)

{very similar to code above)

else

{ TreeNode \* temp = root->leftPtr;

if (temp->rightPtr is null) // the node pointed to by root->leftPtr { // has the largest value in the subtree

root->info = temp->info // copy largest value in left subtree // to root node

root->leftPtr = **temp->leftPtr;** // bypass temp node **Error corrected**

delete temp;

}

else

{ while (temp->rightPtr->rightPtr is not null)

temp= temp->rightPtr; // look for the node with largest value

TreeNode \* deletePtr = temp->rightPtr; // deletePtr points to node // with largest value

root->info = deletePtr->info; //copy largest value to root node

temp->rightPtr = deletePtr->leftPtr; // bypass node to be deleted

delete deletePtr; // delete node

}

}

Try tracing through this code on the binary search tree example above. Before the values are copied and the node deleted, it should look like the following. If you don’t get to this picture, please see the instructor.

root



deletePtr

temp