Deletion from a Binary Tree

Last time we outlined how to delete a node from a binary tree. We broke up our work into three cases:

1) Deleting a leaf node
2) Deleting a node with one child
3) Deleting a node with two children.

Keep these three in mind, as well as how we determined the delete would work in these cases when looking at this code.

For a leaf node, our job isn't too hard. The key will be identifying the parent of the node to delete. (Actually, we'll do this in the other cases as well.) Once we do this, then we just have to set the appropriate node to NULL:

```
parert->left = NULL or

parent->right = NULL
```

For the second task, we'll need to find the parent node of the node to be deleted, along with the sole child node of the node to be deleted and "connect them." Here are the four possibilities:

```
        10                        10
      /     \                   /    \
     5       15   ====>      1      15            (delete 5)
    /
   1
```

```
     10                    10
  /      \               /    \
5          15   ====>   7      15            (delete 5)
  \
   7

     10                    10
  /      \               /    \
5          15   ====>   5      12            (delete 15)
          /
        12

     10                    10
  /     \                /    \
5          15   ====>   7      23            (delete 15)
             \
              23
```

In each of these two main cases, I am still glossing over a couple details:

1) What happens if the node to delete IS the root node of the tree and has no parent?

2) How do we free memory for the deleted node?

We'll deal with both of these issues when we see the code.

Finally, we have to deal with the case where a node to be deleted has two children. The nice thing here is that we will not physically do the delete of the chosen node. If we did, deciding what gets to be the root node, and what takes its place etc. could be unusually complicated.

Could there be another node we physically delete so that we could essentially maintain the structure of the tree?

Consider the following example:

```
               30
             /      \
         15          45
       /      \       /    \
     5      23    39    50
             /      /
          18    36
```

If I were deleting 30, what two nodes could I replace the 30 with and still essentially maintain the binary tree property?

Since all the terms to the left of the 30 must be less than it, and all to the right must be greater than it, the only values we can put at the root w/o serious repercussions are

1) The maximum value in the left subtree (in this case 23)

OR

2) The minimum value in the right subtree (in this case 36)

Once we pick one of these two to replace with the root node, we must simply delete the node corresponding to the value we pick. We are guaranteed that this node has at most one child.

Why is that?

Now, let's look at the picture of us deleting 30 from this node, when we replace 30 with 23:

```
            23
          /    \
       15        45
     /    \     /   \
    5     18   39   50
               /
             36
```

Some Auxiliary Functions

In aiding the delete function, I have written several auxiliary functions:

1) parent - finds the parent of a given node in a given binary tree.

2) minVal - finds the minimum value in a given binary tree.

3) maxVal - finds the maximum value in a given binary tree.

4) isLeaf - determines if a node is a leaf node or not.

5) hasOnlyLeftChild - determines if a node ONLY has a left child or not.

6) hasOnlyRightChild - determines if a node ONLY has a right child or not.

7) findNode - returns a pointer to a node in a given tree that stores a particular value