CS 260
Deleting BST Nodes

Overview

There are two ways to delete a node from a BST. One way is to have a Boolean value in the node that indicates if it is present or not. The second is to find the largest smaller value or the smallest larger value and use it as a replacement.

Deletion by Marking

The simplest way to remove a node is to keep a Boolean value indicating if it is present. The node itself remains in the tree to maintain the structure of its subtree. The following pseudo code illustrates this algorithm. Note that it is necessary to also modify the find method to verify a node is present before reporting that it was found.

Remove Method

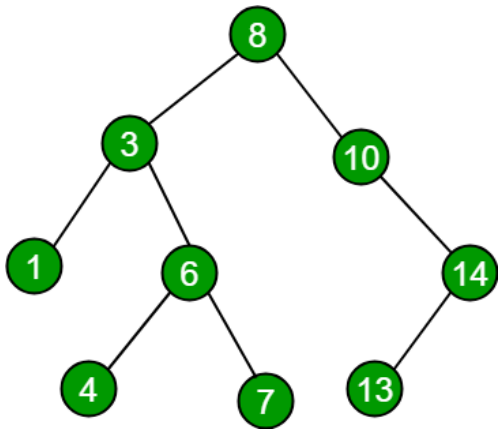
```
bool remove(value)
    ptr = root

    while ptr != nullptr

        if ptr.value == value and ptr.present
            ptr.present = false
            return true

        if value < ptr.value
            ptr = ptr.left
        else
            ptr = ptr.right

    return false
```

Find Method

```
bool find(value)
    ptr = root

    while ptr != nullptr

        if ptr.value == value and ptr.present
            return true

        if value < ptr.value
            ptr = ptr.left
        else
            ptr = ptr.right

    return false
```

Deletion by replacement

For the second method, consider the following tree:



To delete the root, either replace it with 7 (the largest smaller number) or 10 (the smallest larger number). This discussion will use the smallest larger number, which is normally referred to as the inorder successor. To find this node, start at the node to be deleted, go to its left subchild, then go as far to the right as possible. When there are no more right children, the inorder successor has been found.

The algorithm to solve this has four cases:

1. The node to be deleted has no children, in this case simply delete the node.
2. The node has one child, simply replace the connection from the parent to the node with a connection from the parent to the child.
3. The node has two children.
    a. The successor is the right child of the node to be deleted (replacing 8 with 10). Simply link the right subtree to the parent (connect 8 to 14 above).
4. The successor is a left descendent of the node to be deleted (replacing 3 with 4 above).
    a. Attach right child of successor as left child of successor's parent.
    b. Attach right child of node to be deleted as right child of successor.
    c. Attach left child of node to be deleted to left child of successor
    d. Attach successor to parent of node to be deleted

Deletion by replacement

The following pseudo code algorithm does this recursively. Note that it does it by copying the value from the successor to the node being deleted and then deleting the successor.

```
// delete by removing
bool removeItem(int value)

   // see if it is there
   bool found = recFind(value, root)

   // if it was found call recursive delete
   // returns root of modified tree
   if (found)
```

```
        {
            root =  recDelete(root, value);
        }

        return found


    // return the minValueNode (smallest value in subtree)
    Node minValueNode(Node ptr)

        Node current = ptr

        // walk down left most branch to end
        while (current->getLeft() != nullptr)
            current = current->getLeft()

        return current


// recursive delete function itself
    Node recDelete(Node ptr, int value)

        // base case
        if (ptr == nullptr)
            return nullptr

        // if node to delete is to the left, go there
        if (ptr->getValue() > value)
            ptr->setLeft(recDelete(ptr->getLeft(), value))

        // if it is to the right, go there
        else if (ptr->getValue() < value)
            ptr->setRight(recDelete(ptr->getRight(), value))


        // this is actually the node; delete it
        else

                // first case, no left child
            // return the right subtree for insertion
            if (ptr->getLeft() == nullptr)
                Node temp = ptr->getRight()
                delete ptr // C++
                return temp

            // second case, no right child
                // return the left subtree for insertion
            if (ptr->getRight() == nullptr)
                Node temp = ptr->getRight()
                delete ptr  // C++
                return temp


            // third case, two children, replace with inorder successor
     // find smallest value in right subtree (inorder successor)
            Node temp = minValueNode(ptr->getRight())

                // copy that value to this node
```

```
    ptr->setValue(temp->getValue())

        // now recursively delete that node
    ptr->setRight(recDelete(ptr->getRight(), temp->getValue()))


       // all done, return the updated link
return ptr
```