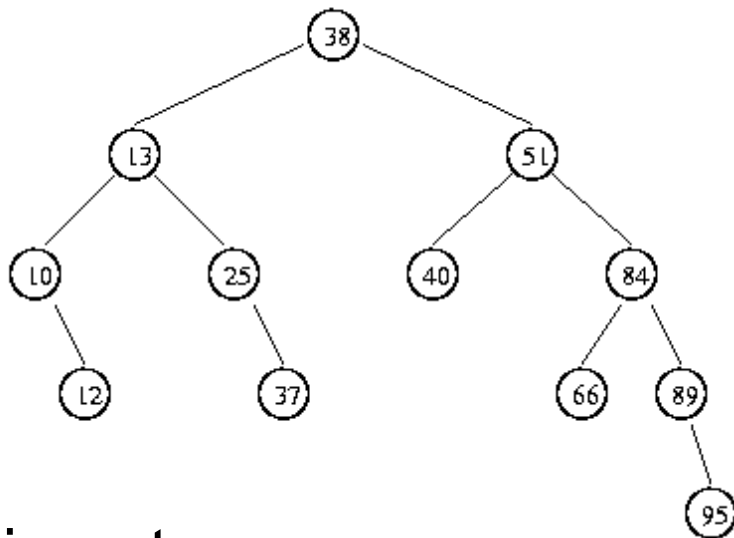


Announcements

MP5 available, due 11/1, 11:59p. EC due 10/25, 11:59p.

<http://www.qmatica.com/DataStructures/Trees/AVL/AVLTree.html>



insert

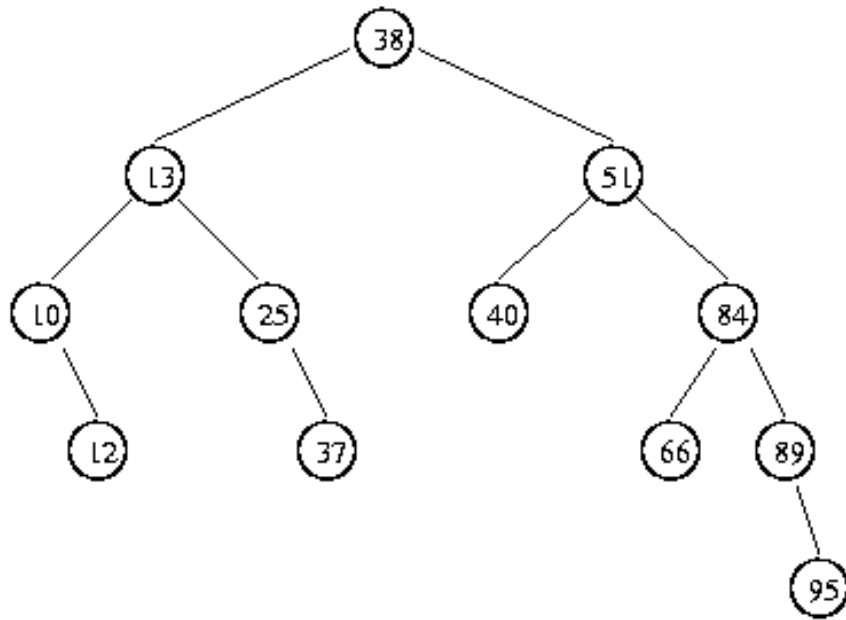
remove

find

traverse

```
template <class K, class V>
class Dictionary{
public:
    // ctor for empty tree, + ...
private:
    struct treeNode{
        V value; // data
        K key;   // identifier
        treeNode * left;
        treeNode * right;
    };
    treeNode * root
};
```

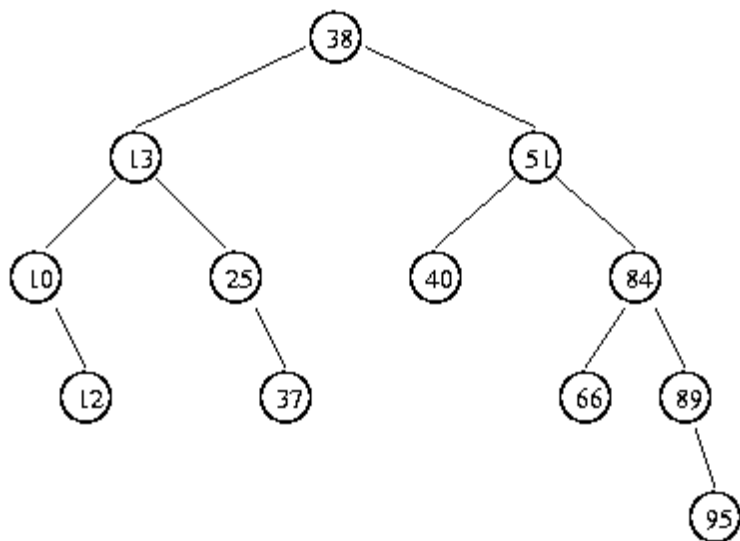
Binary Search Tree - Insert



Running time:

```
(treeNode * cRoot, const K & key, const V & data){  
    if (cRoot == NULL)  
  
        else if (cRoot->key == key)  
  
        else if (key < cRoot->key)  
  
        else  
  
}
```

Binary Search Tree - Remove

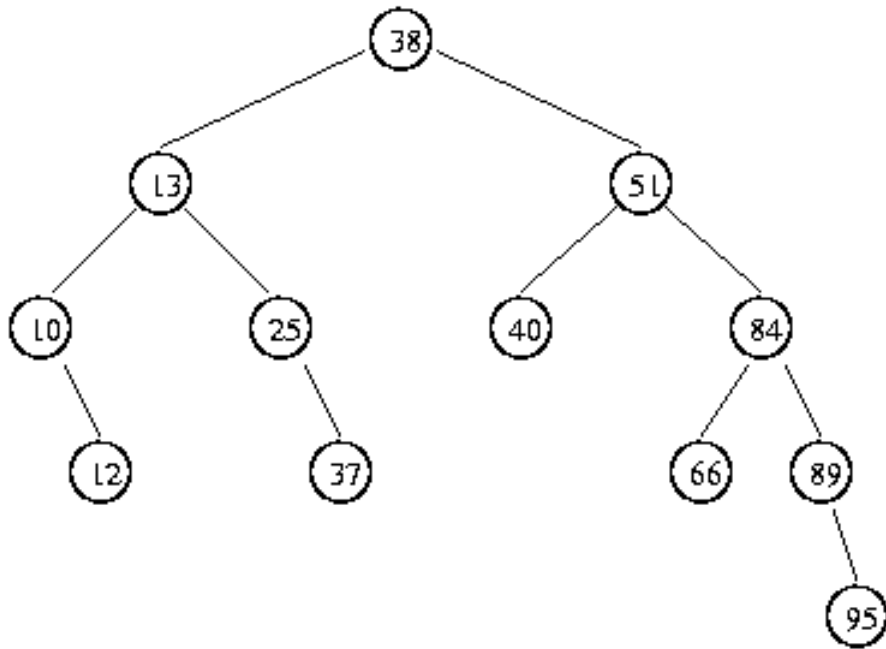


```
void Dict<K,V>::remove(treeNode * & cRoot, const K & d) {  
    if (cRoot != NULL) {  
        if (cRoot->key == d)  
            doRemoval(cRoot);  
        else if (k < cRoot->key)  
            remove(cRoot->left, d);  
        else  
            remove(cRoot->right, d);  
    }  
}
```

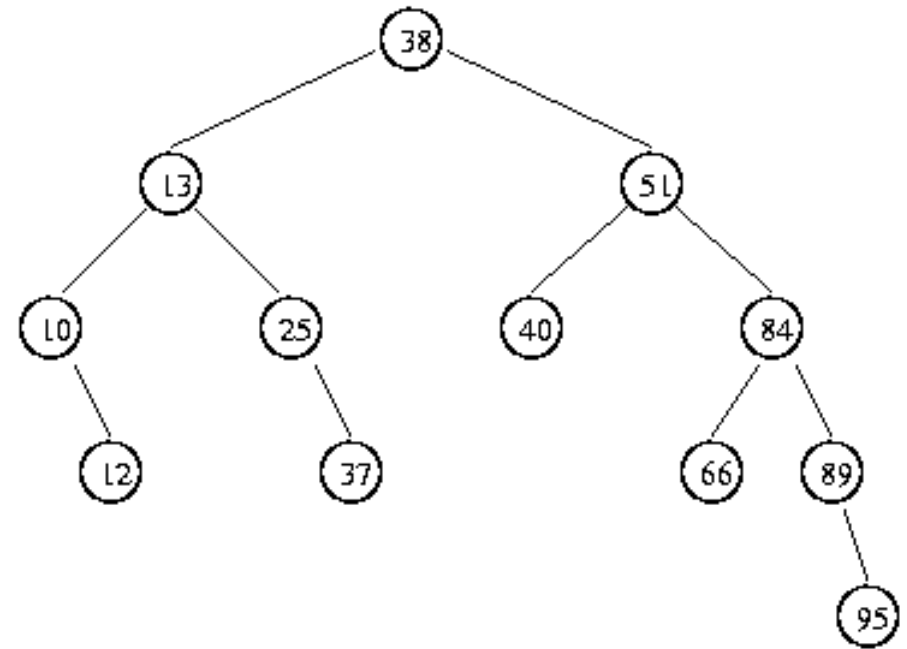
Binary Search Tree - Remove

`T.remove(37);`

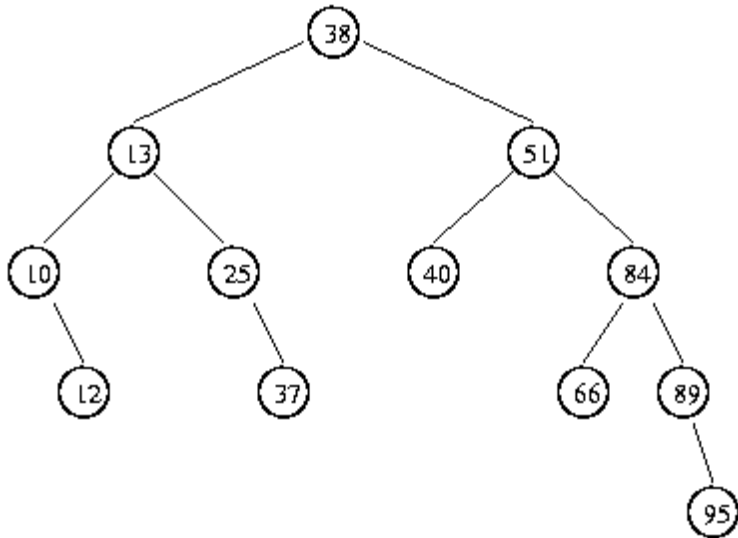
`T.remove(10);`



`T.remove(13);`

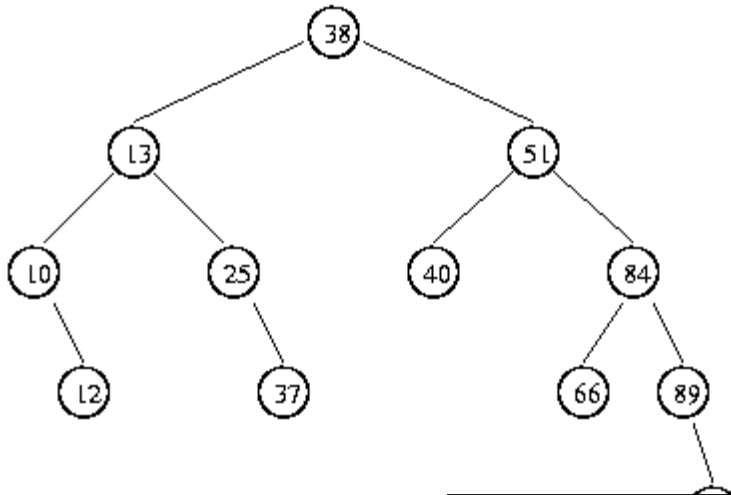


Binary Search Tree - Remove



```
void Dict<K,V>::doRemoval(treeNode * & cRoot) {  
    if ((cRoot->left == NULL) && (cRoot->right == NULL))  
        ____ChildRemove(cRoot);  
    else if ((cRoot->left != NULL) && (cRoot->right != NULL))  
        ____ChildRemove(cRoot);  
    else  
        ____ChildRemove(cRoot);  
}  
}
```

Binary Search Tree - Remove



```

void Dict<K,V>::noChildRemove(treeNode * & cRoot) {
    if (cRoot != NULL) {
        if (cRoot->left == NULL & cRoot->right == NULL)
            doRemove(cRoot);
        else if (cRoot->left != NULL & cRoot->right == NULL)
            remove(cRoot->left, cRoot->left->right);
        else if (cRoot->left == NULL & cRoot->right != NULL)
            remove(cRoot->right, cRoot->right->left);
        else
            remove(cRoot->left, cRoot->right);
    }
}

```

```

void Dict<K,V>::oneChildRemove(treeNode * & cRoot) {
    if ((cRoot->left == NULL & cRoot->right != NULL) ||
        (cRoot->left != NULL & cRoot->right == NULL))
        noChildRemove(cRoot);
    else if (cRoot->left != NULL & cRoot->right != NULL)
        twoChildRemove(cRoot);
    else
        oneChildRemove(cRoot);
}

```

```

void Dict<K,V>::noChildRemove(treeNode * & cRoot) {
    treeNode * temp = cRoot;
    cRoot = NULL;
    delete temp;
}

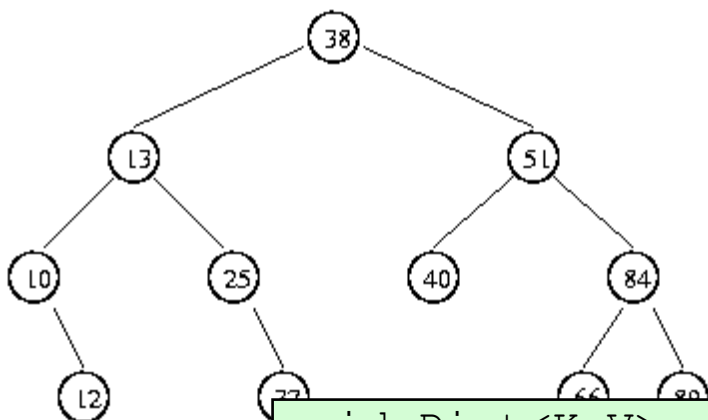
```

```

void Dict<K,V>::oneChildRemove(treeNode * & cRoot) {
    treeNode * temp = cRoot;
    if (cRoot->left == NULL) cRoot = cRoot->right;
    else cRoot = cRoot->left;
    delete temp;
}

```

Binary Search Tree - Remove



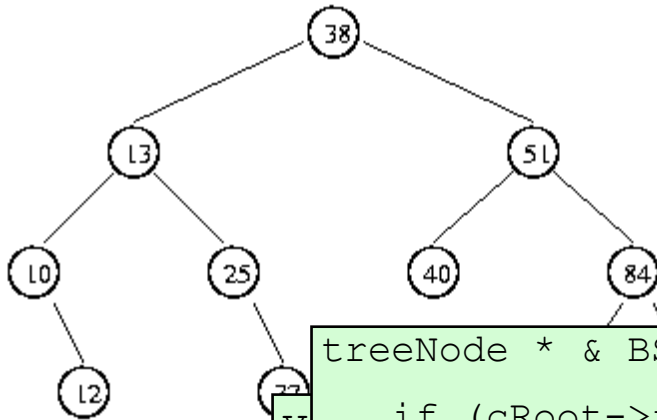
```

void Dict<K,V>::twoChildRemove(treeNode * & cRoot) {
    treeNode * iop = rightMostChild(cRoot->left);
    cRoot->key = iop->key;
    doRemoval(iop);
}
    
```

```

void Dict<K,V>::doRemoval(treeNode * cRoot) {
    if (cRoot != NULL) {
        if (cRoot->left == NULL && cRoot->right == NULL) {
            delete cRoot;
            cRoot = NULL;
        }
        else if (cRoot->left == NULL) {
            cRoot->key = cRoot->right->key;
            doRemoval(cRoot->right);
        }
        else if (cRoot->right == NULL) {
            cRoot->key = cRoot->left->key;
            doRemoval(cRoot->left);
        }
        else {
            twoChildRemove(cRoot);
        }
    }
}
    
```

Binary Search Tree - Remove



```

treeNode * & BST<K>::rightMostChild(treeNode * & cRoot) {
    if (cRoot->right == NULL) return cRoot;
    else return rightMostChild(cRoot->right);
}
  
```

```

void BST<K>::remove(int iop) {
    if (cRoot != NULL) {
        if (cRoot->key == iop)
            doRemoval(iop);
        else if (cRoot->left != NULL)
            remove(iop);
        else if (cRoot->right != NULL)
            remove(iop);
        else
            remove(iop);
    }
}
  
```

```

void BST<K>::doRemoval(int iop) {
    if (cRoot->key == iop) {
        if (cRoot->left != NULL)
            delete cRoot;
            cRoot = cRoot->left;
        else if (cRoot->right != NULL)
            delete cRoot;
            cRoot = cRoot->right;
        else
            delete cRoot;
            cRoot = NULL;
    }
}
  
```

```

void BST<K>::oneChildRemove(int iop) {
    if (cRoot->key == iop) {
        if (cRoot->left != NULL)
            delete cRoot;
            cRoot = cRoot->left;
        else if (cRoot->right != NULL)
            delete cRoot;
            cRoot = cRoot->right;
        else
            delete cRoot;
            cRoot = NULL;
    }
}
  
```

```

void BST<K>::twoChildRemove(int iop) {
    if (cRoot->key == iop) {
        treeNode * temp = rightMostChild(cRoot->right);
        delete cRoot;
        cRoot = temp;
    }
}
  
```