Engineering IT Town Hall

Come to the Engineering Tech Services Town Hall and let your voice be heard! The College wants your feedback on:

- **EWS Labs**
- **Remote Access**
- **Software or Hardware Resources**
- **Online Resources (such as Compass 2g)**

When and Where?
TUESDAY, MARCH 12
5:30 PM
100 MSEB

Pizza served to all attendees!

RSVP:

Or:
tinyurl.com/UIUC-IT

Announcements

MP4 available, due 3/8, 11:59p.

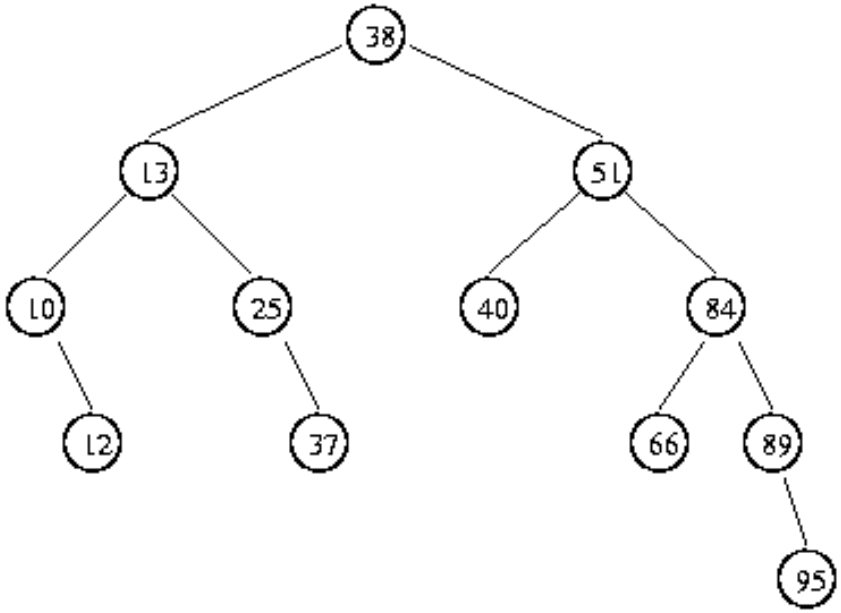
GDB tutorial: Saturday, 3/9, 3p, Siebel 0224.

Code Challenge #2: winners!

TODAY: BST insert and remove

(http://webdiis.unizar.es/asignaturas/EDA/AVLTree/avltree.html)

Dictionary ADT: (BST implementation)



insert

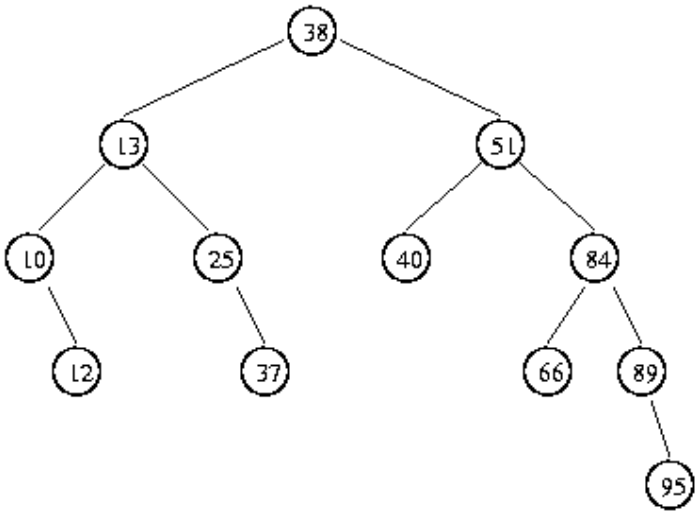
remove

find

traverse

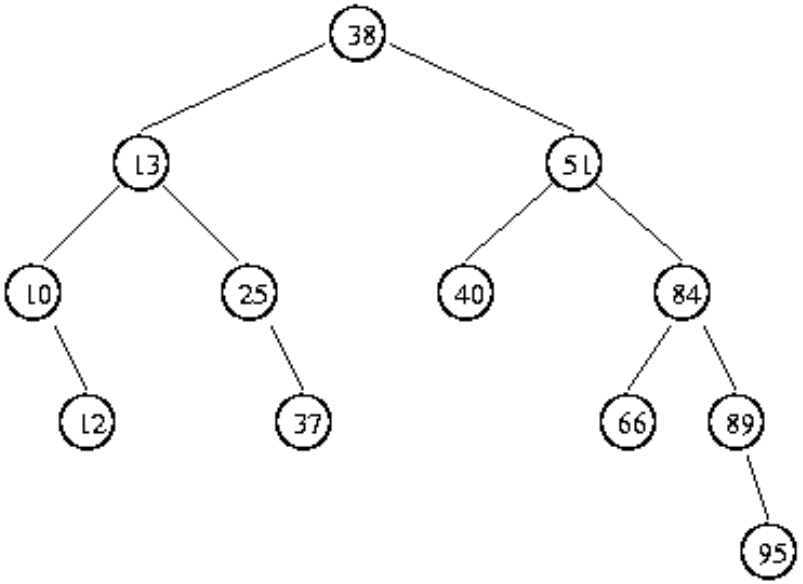
```
template <class K, class D>
class Dictionary{
public:
// constructor for empty tree.
private:
   struct treeNode{
      D data;
      K key;
      treeNode * left;
      treeNode * right;
   };
   treeNode * root
};
```

Binary Search Tree - Find



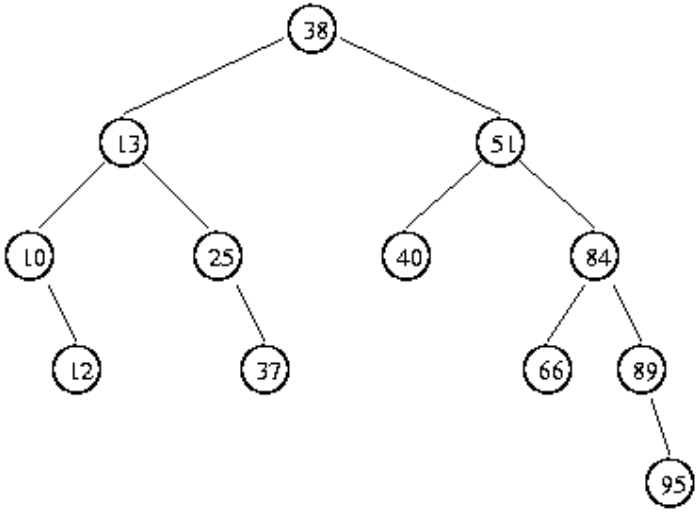

```
treeNode * BST<K,D>::find(treeNode * cRoot, const K & key)
{
   if (cRoot == NULL)
      return cRoot;
   else if (cRoot->key == key)
      return cRoot;
   else if (key < cRoot->key)
      return find(cRoot->left, key);
   else
      return find(cRoot->right, key);
}
```

Binary Search Tree - Insert




```
_____________________________
(treeNode * cRoot, const K & key, const D & data){
   if (cRoot == NULL)



   else if (cRoot->key == key)



   else if (key < cRoot->key)



   else



}
```

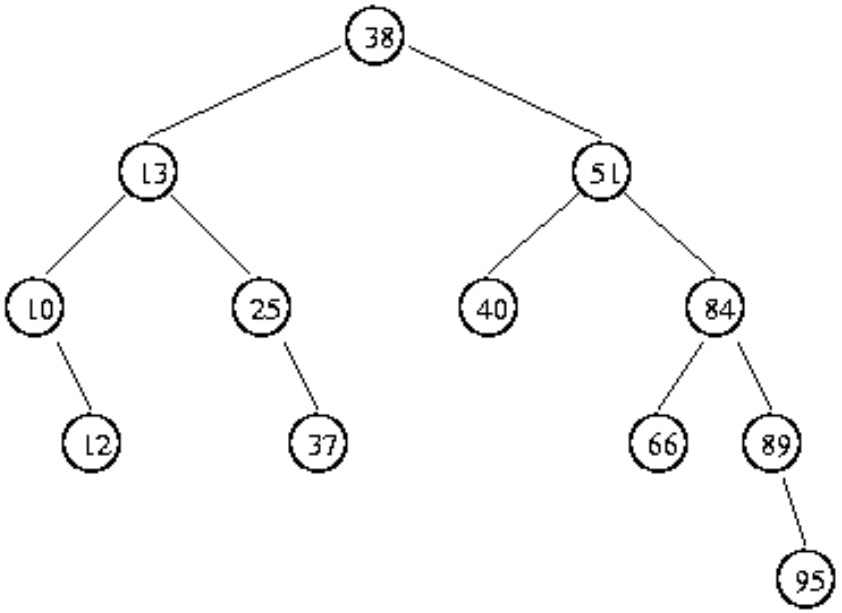
Binary Search Tree - Remove



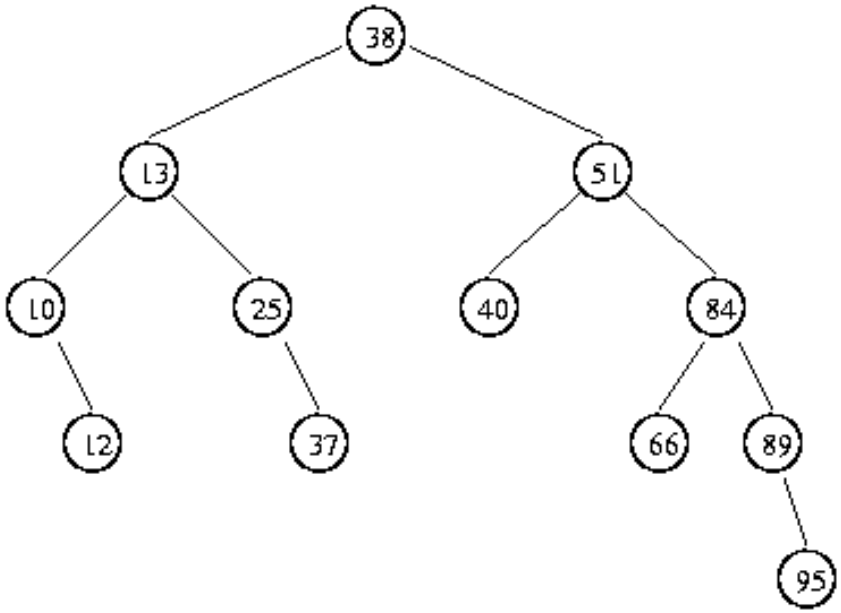

```
void BST<K>::remove(treeNode * & cRoot, const T & d) {
   if (cRoot == NULL)
     return; // no op... key not found
   else if (cRoot->key == d)
     doRemoval(cRoot);
   else if (d < cRoot->key)
     remove(cRoot->left,d);
   else
     remove(cRoot->right,d);
}
```

Binary Search Tree - Remove

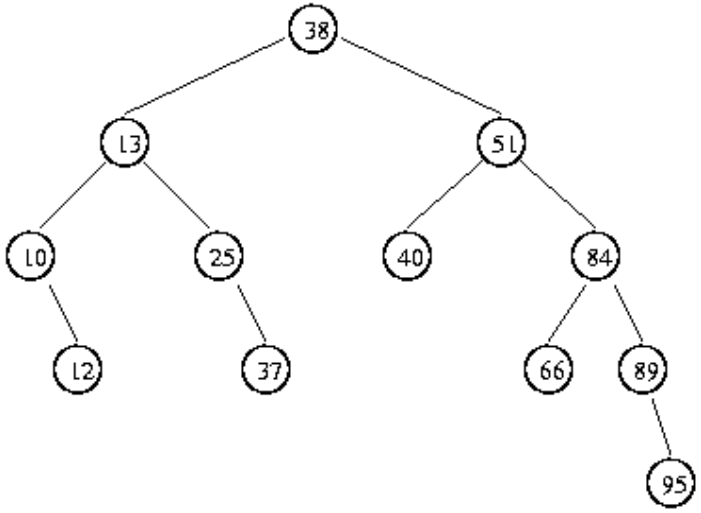
```
T.remove(37);
T.remove(10);
```




```
T.remove(13);
```



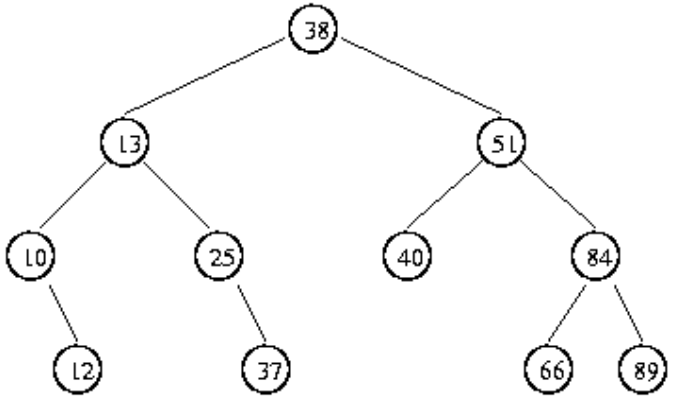
Binary Search Tree - Remove




```
void BST<K>::remo
   if (cRoot == N
      return;  //
   else if (cRoot
      doRemoval(c
   else if (d < c
      remove(cRoo
   else
      remove(cRoot->right,d);
}
```

```
void BST<K>::doRemoval(treeNode * & cRoot) {
   if ((cRoot->left == NULL) && (cRoot->right == NULL))
      noChildRemove(cRoot);
   else if ((cRoot->left != NULL) && (cRoot->right != NULL))
      twoChildRemove(cRoot);
   else
      oneChildRemove(cRoot);
}
```

Binary Search Tree - Remove

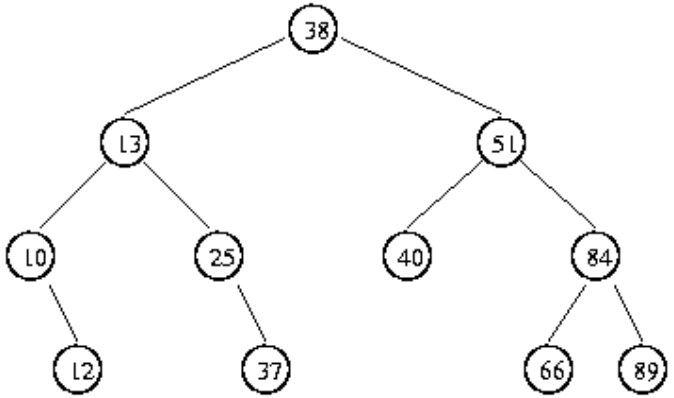



```
void BST<K>::noChildRemove(treeNode * & cRoot) {
   delete cRoot;
   cRoot = NULL;
}
```

```
void BST<K>
   if ((cRo
      noChildRemove(cRoot);
   else if ((cRoot->left != NULL) && (cRoot->right != NULL))
      twoChildRemove(cRoot);
   else
      oneChildRemove(cRoot);
}
```

```
void BST<K>::remo
   if (cRoot == N
      return;  //
   else if (cRoot
      doRemoval(c
   else if (d < c
      remove(cRoo
   else
      remove(cRoot->right,d);
}
```

Binary Search Tree - Remove

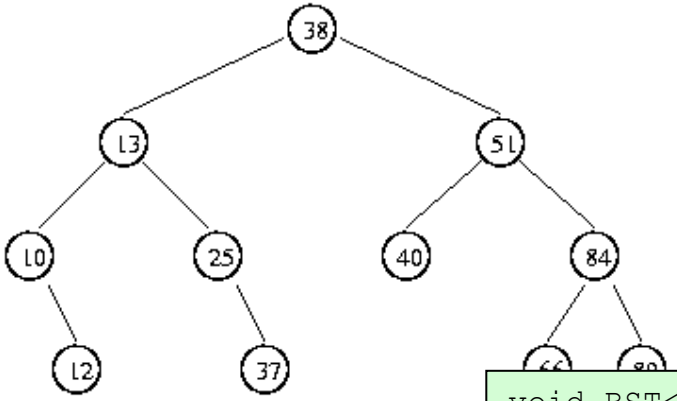



```
void BST<K>::oneChildRemove(treeNode * & cRoot) {
   treeNode * temp = cRoot;
   if (cRoot->left == NULL) cRoot = cRoot->right;
   else cRoot = cRoot->left;
   delete temp;
}
```

```
void BST<K>
   if ((cRo
      noChi
   else if
      twoChildRemove(cRoot);
   else
      oneChildRemove(cRoot);
}
```

```
void BST<K>::remo
   if (cRoot == N
      return;  //
   else if (cRoot
      doRemoval(c
   else if (d < c
      remove(cRoo
   else
      remove(cRoot->right,d);
}
```

Binary Search Tree - Remove

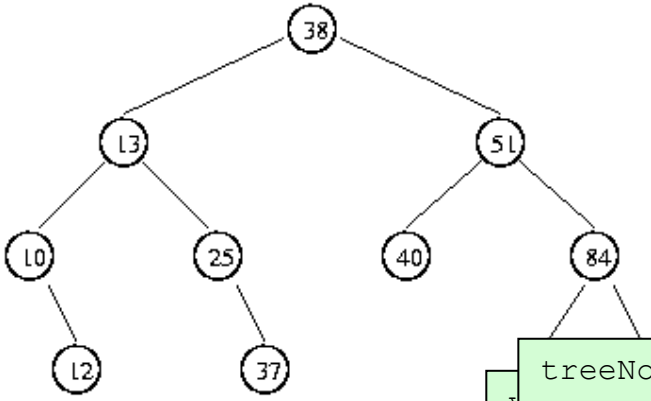



```
void BST<K>::twoChildRemove(treeNode * & cRoot) {
   treeNode * iop = IOP(cRoot);
   cRoot->key = iop->key;
   doRemoval(IOP(cRoot));
}
```

```
void BST<K>
   if ((cRo
      noChil
   else if (
      twoChildRemove(cRoot);
   else
      oneChildRemove(cRoot);
}
```

```
void BST<K>::remo
   if (cRoot == N
      return;  //
   else if (cRoot
      doRemoval(c
   else if (d < c
      remove(cRoo
   else
      remove(cRoot->right,d);
}
```

Binary Search Tree - Remove



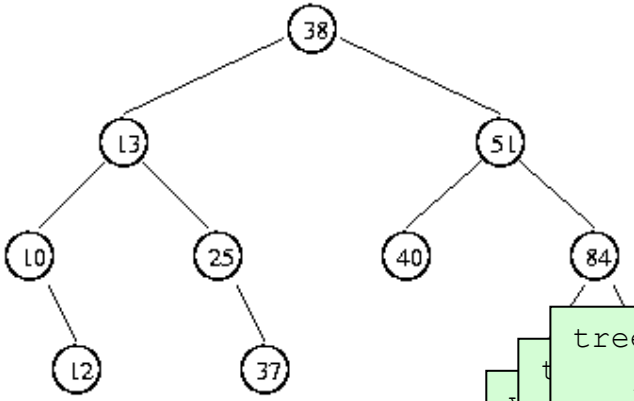

```
treeNode * & BST<K>::IOP(treeNode * & cRoot) {
   return rightMostChild(cRoot->left);
}
```

```
   doRemoval(iop);
}
```

```
void BST<K>
   if ((cRo
      noChi
   else if
      twoChildRemove(cRoot);
   else
      oneChildRemove(cRoot);
}
```

```
void BST<K>::remo
   if (cRoot == N
      return;  /
   else if (cRoo
      doRemoval(
   else if (d < c
      remove(cRo
   else
      remove(cRoot->right,d);
}
```

Binary Search Tree - Remove




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

```
void BST<K>
   if ((cRo
      noChi
   else if
      twoChildRemove(cRoot);
   else
      oneChildRemove(cRoot);
}
```

```
void BST<K>::rem
   if (cRoot == N
      return;  /
   else if (cRoo
      doRemoval(
   else if (d < c
      remove(cRo
   else
      remove(cRoot->right,d);
}
```

```
doRemoval(iop);
}
```