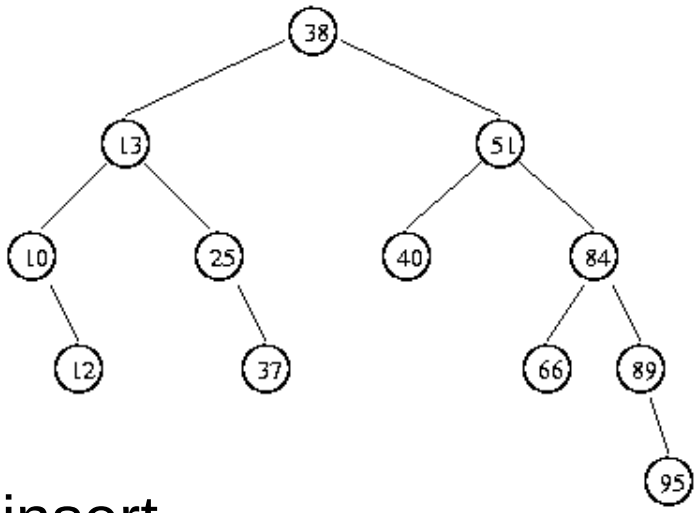
Announcements

MP4 due tonight (3/10, 11:59pm), Exam 4 starts Sunday (3/12)



insert

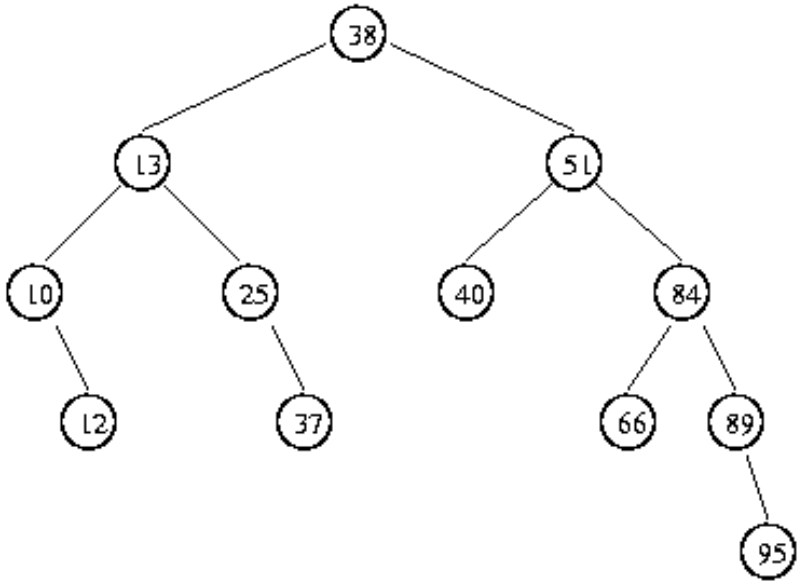
remove

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
};
```

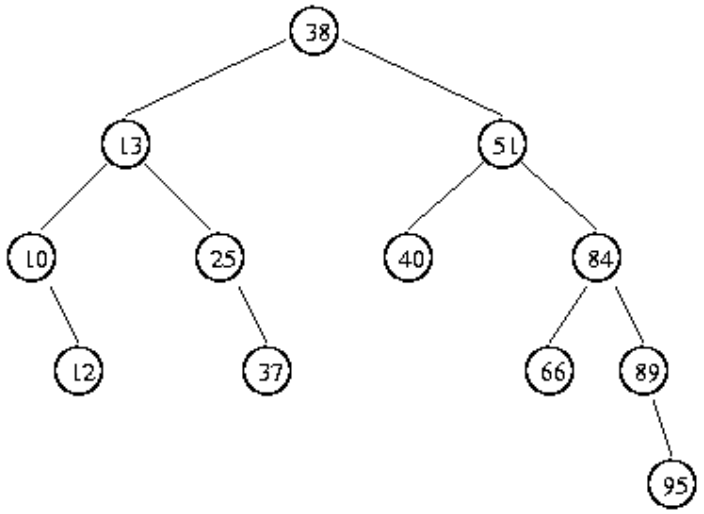
Binary Search Tree - Insert



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

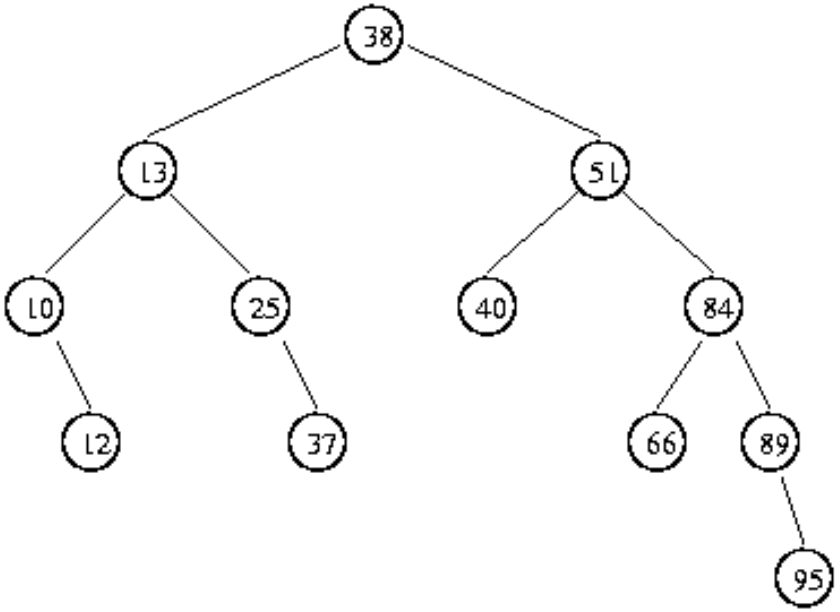



```
void Dict<K,V>::remove(treeNode * & cRoot, const K & k) {
   if (cRoot != NULL) {
      if (k == cRoot->key)
        doRemoval(cRoot);
      else if (k < cRoot->key)
        remove(cRoot->left, d);
      else
        remove(cRoot->right, d);
  }
}
```

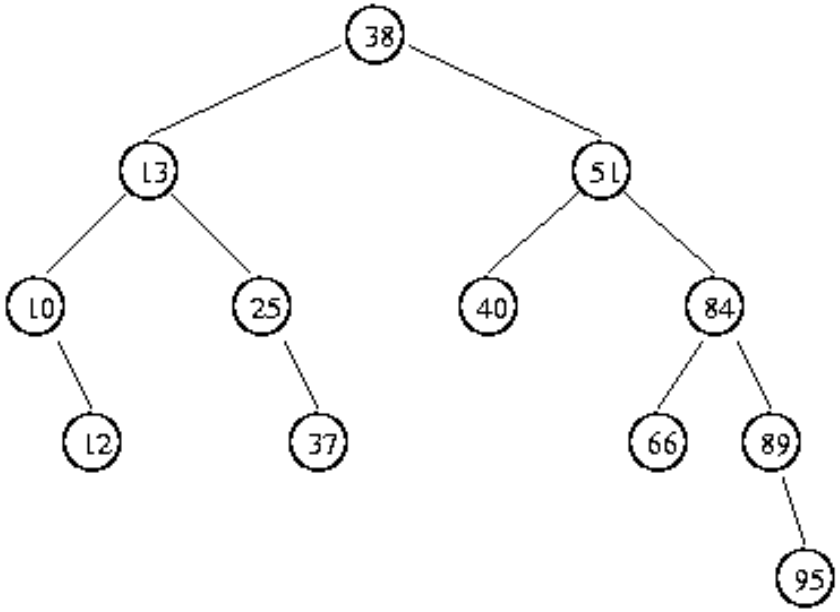
Binary Search Tree - Remove

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

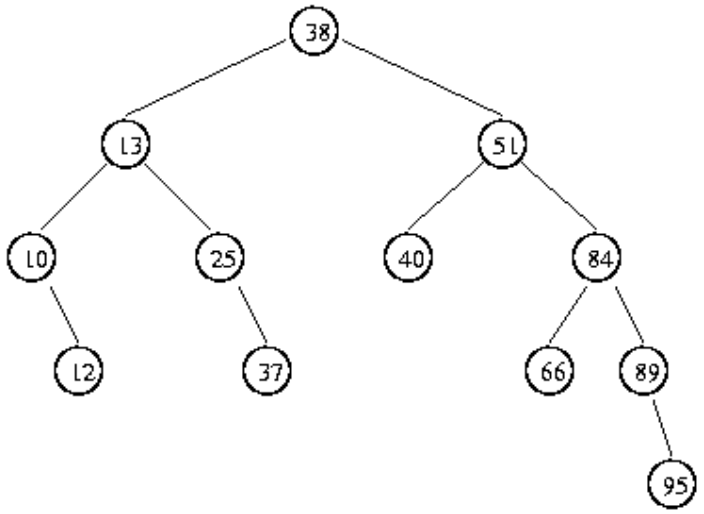
```
T.remove(10);
```




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



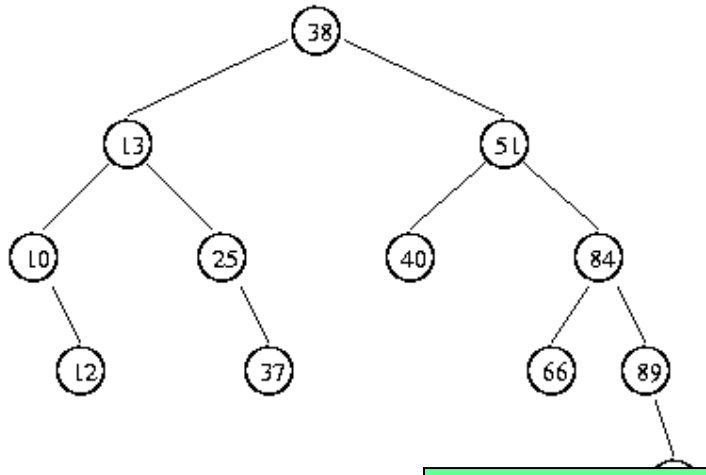
Binary Search Tree - Remove




```
void Dict<K,V>:
   if (cRoot !=
      if (cRoot
        doRem
      else if (
        remov
      else
        remov
  }
}
```

```
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



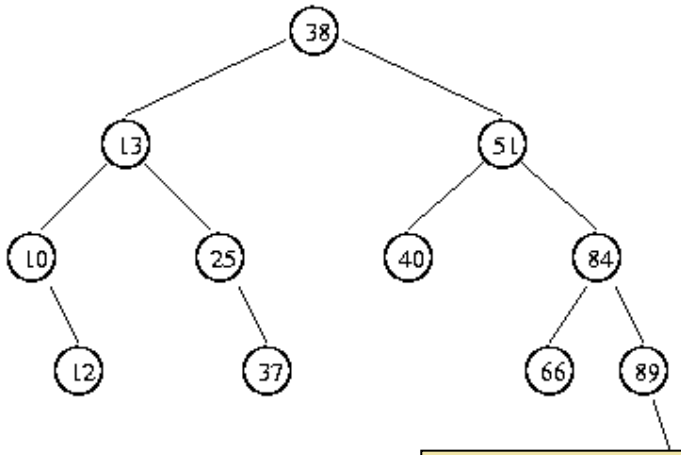

```
void Dict<K,V>:
   if (cRoot !=
      if (cRoot
        doRem
      else if (
        remov
      else
        remov
  }
}
```

```
void Dict<K,
   if ((cRoo
      noChil
   else if (
      twoChildRemove(cRoot);
   else
      oneChi
}
```

```
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>:
   if (cRoot !=
      if (cRoot
        doRem
      else if (
        remov
      else
        remov
  }
}
```

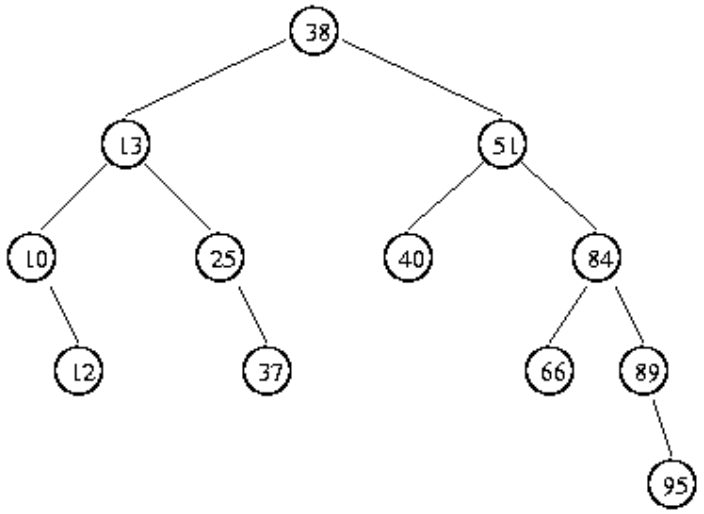
```
void Dicti
   if ((cR
      noCh
   else if
      twoC
   else
      oneChi
}
```

```
v
}
```

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

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

Binary Search Tree - Remove




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

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

   else if (cRoot->key == key)

   else if (key < cRoot->key)

   else

}
```

```
_____________________
(treeNode *  cRoot, const K & key) {
   if (cRoot == NULL)

   else if (cRoot->key == key)

   else if (key < cRoot->key)

   else

}
```