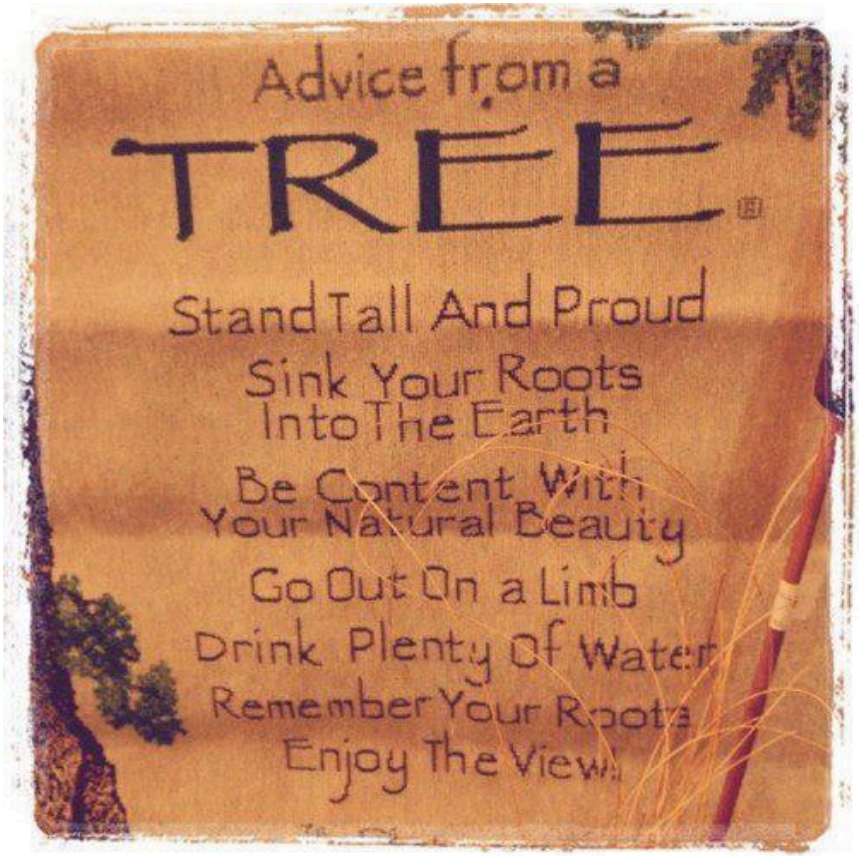
Announcements

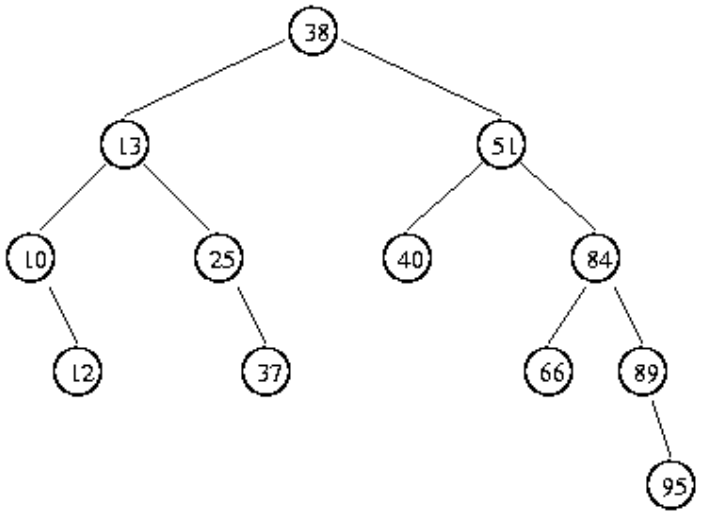
MP4 available, due 3/8, 11:59p. EC due 3/1, 11:59p.

Code Challenge #2: 3/6, 9p, Siebel 0224.

TODAY: tree traversals
ADT Dictionary
BST

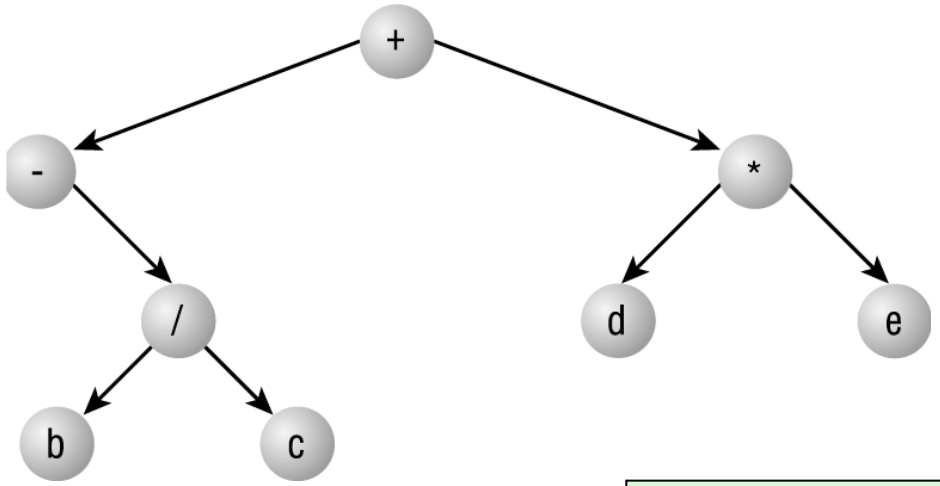


Traversals: another broader view…




```
template<class T>
void binaryTree<T>::clear(treeNode * croot){









}
```

Traversals: something totally different…



Running time:

```
template<class T>
void binaryTree<T>::levelOrder(treeNode * croot){









}
```

ADT Dictionary:

Suppose we have the following data…

Locker Numbe	Name
103	Jay Hathaway
92	Linda Stencel
330	Bonnie Cook
46	Rick Brown
124	Kim Petersen
…	…

…and we want to be able to retrieve a name, given a locker number.

More examples of key/value pairs:

UIN -> Advising Record

Course Number -> Schedule info

Color -> BMP

Vertex -> Set of incident edges

Flight number -> arrival information

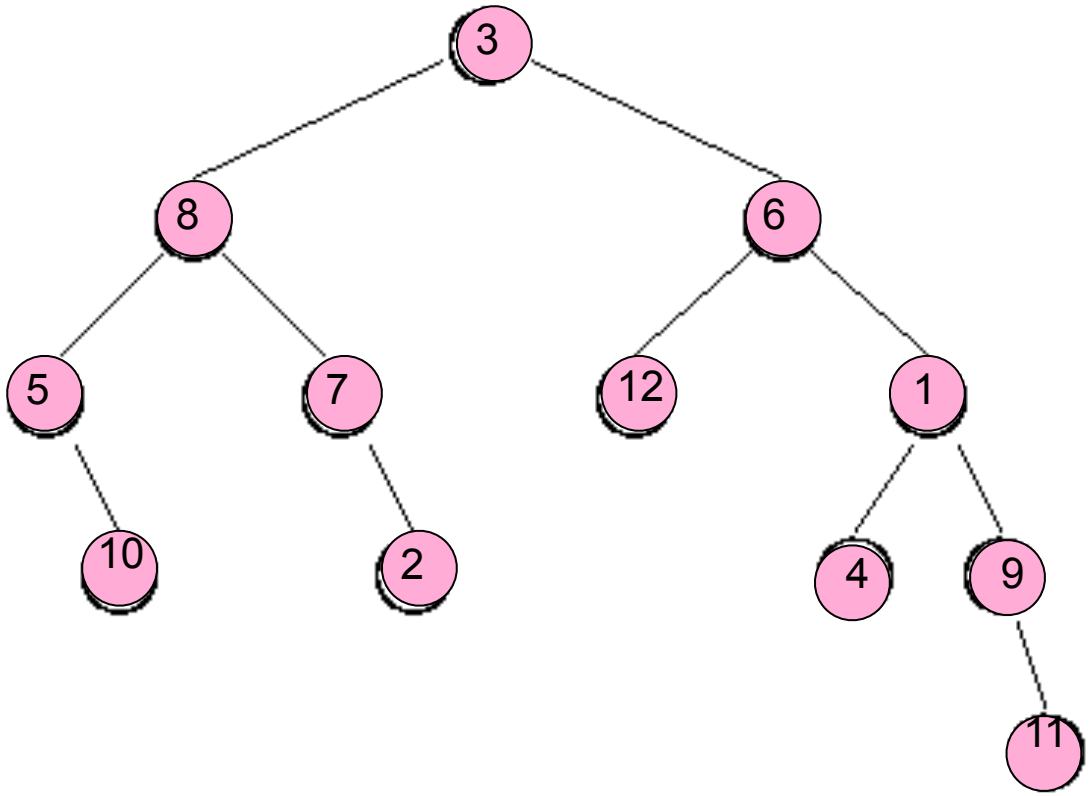
URL -> html page

A *dictionary* is a structure supporting the following:

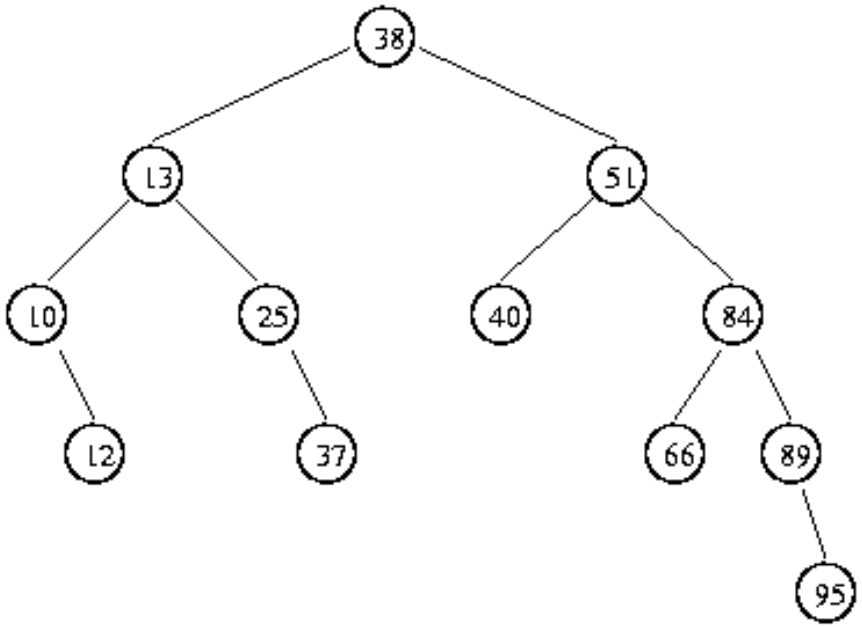
```
void insert(kType & k, dType & d)
void remove(kType & k)
dType find(kType & k)
```

Binary Trees as a search structure (Dictionary)

Find me ...



Binary _____ Tree



A Binary Search Tree (BST) is a binary tree, T, such that:

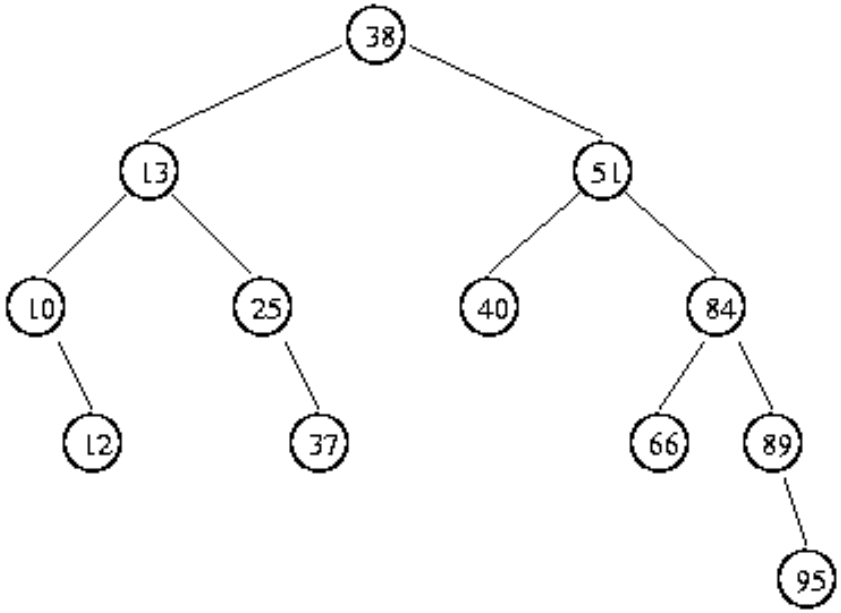
- _____, OR
- $T = \{r, T_L, T_R\}$ and

  $x \in T_L \rightarrow$ _____

  $x \in T_R \rightarrow$ _____

and

Dictionary ADT: (BST implementation)



insert

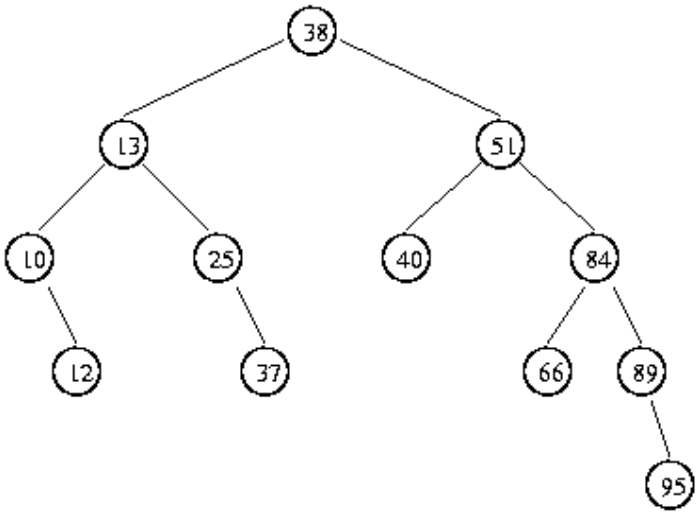
remove

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



   else if (cRoot->key == key)



   else if



   else



}
```