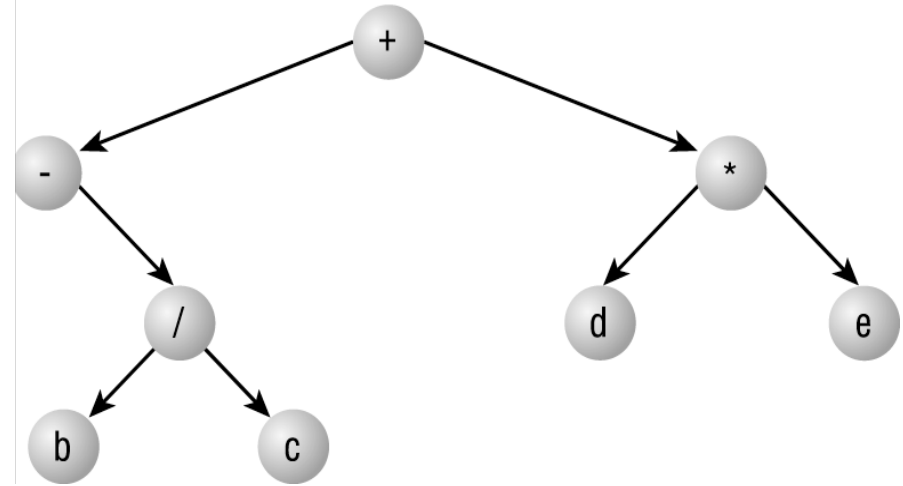


Announcements

MP4 available, due 10/14, 11:59p.



Running time:

```
template<class T>
void binaryTree<T>::levelOrder(treeNode * croot){
    queue<treeNode *> Q;
    Q.enqueue(croot);
    while( !Q.isEmpty()){
        treeNode * t = Q.dequeue();

        yell(t->data);
        Q.enqueue(t->left);
        Q.enqueue(t->right);
    }
}
```

ADT Dictionary:

Suppose we have the following data...

ID #	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:

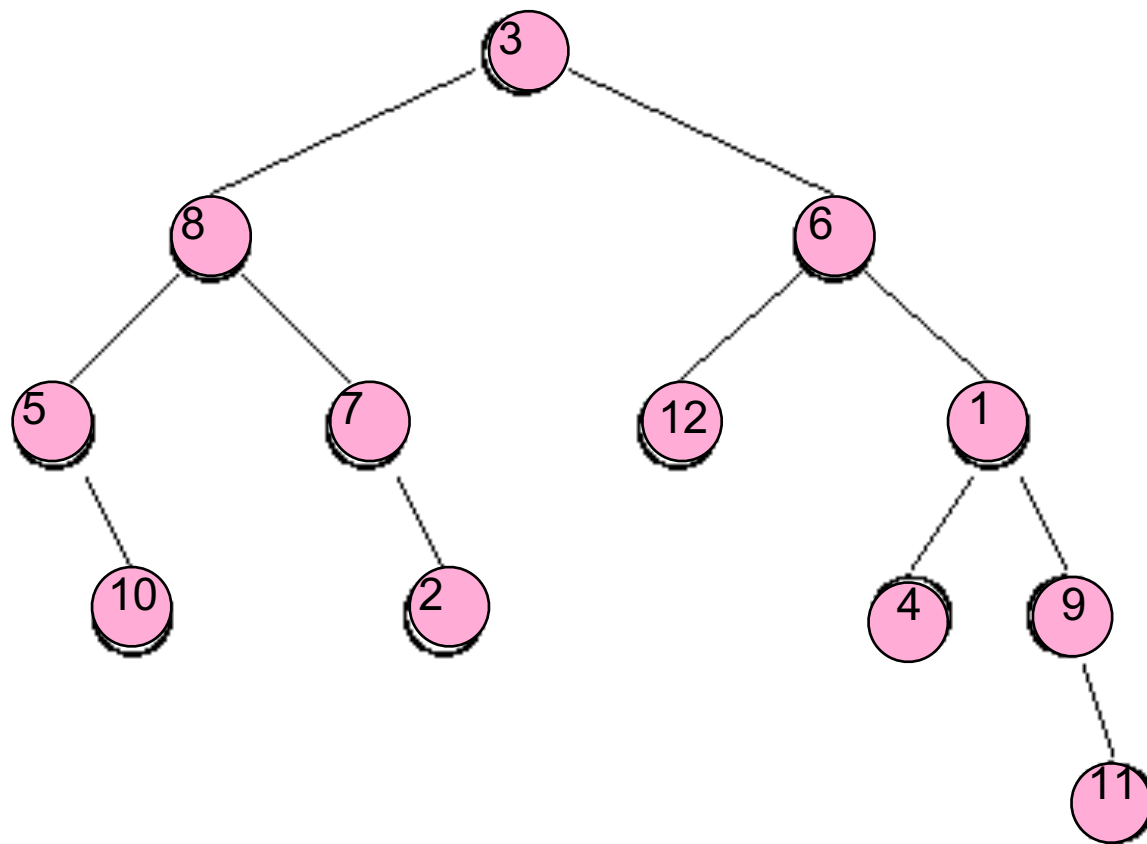
```
void insert(kType & k, dType & d)
```

```
void remove(kType & k)
```

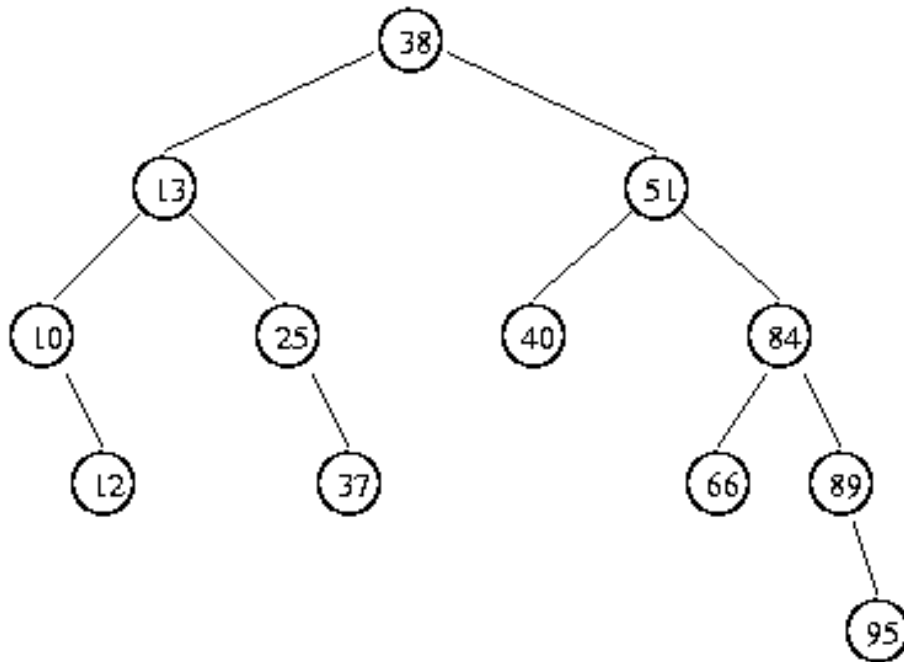
```
dType find(kType & k)
```

Binary Trees as a search structure (Dictionary)

Find me ...



Binary _____ Tree



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

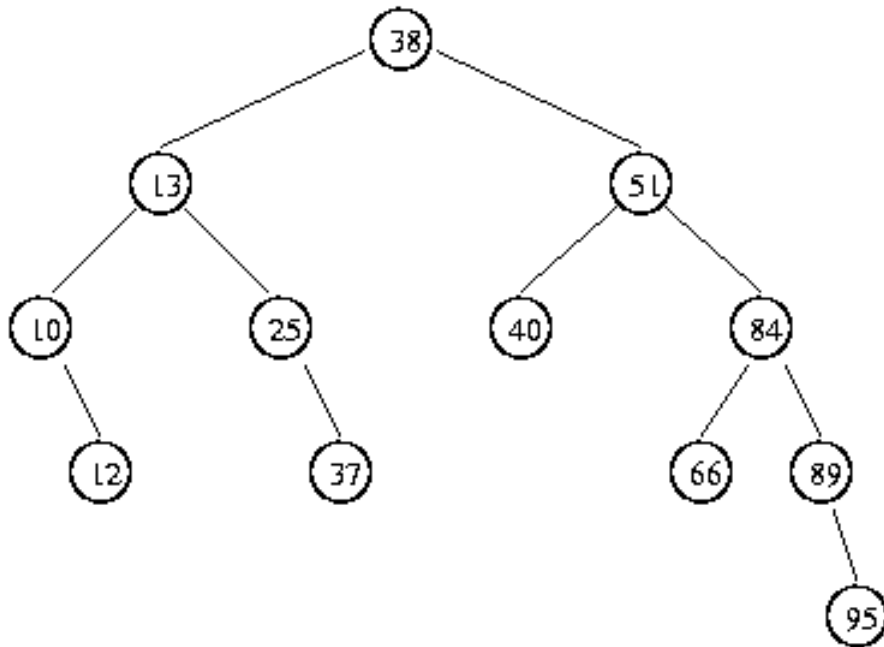
- _____, OR
- $T = \{r, T_L, T_R\}$ and

$x \in T_L \rightarrow$ _____

$x \in T_R \rightarrow$ _____

and

Dictionary ADT: (BST implementation)



insert

remove

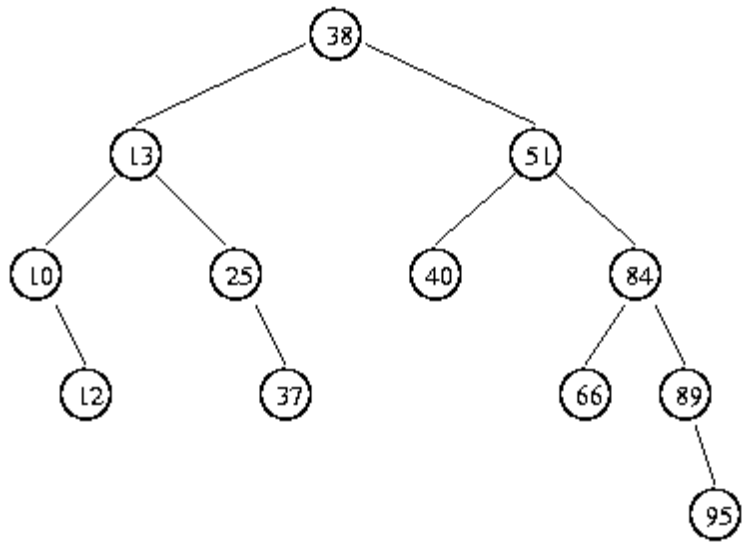
find

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
};
```

<https://visualgo.net/bst>

Binary Search Tree - Find



Running time:

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

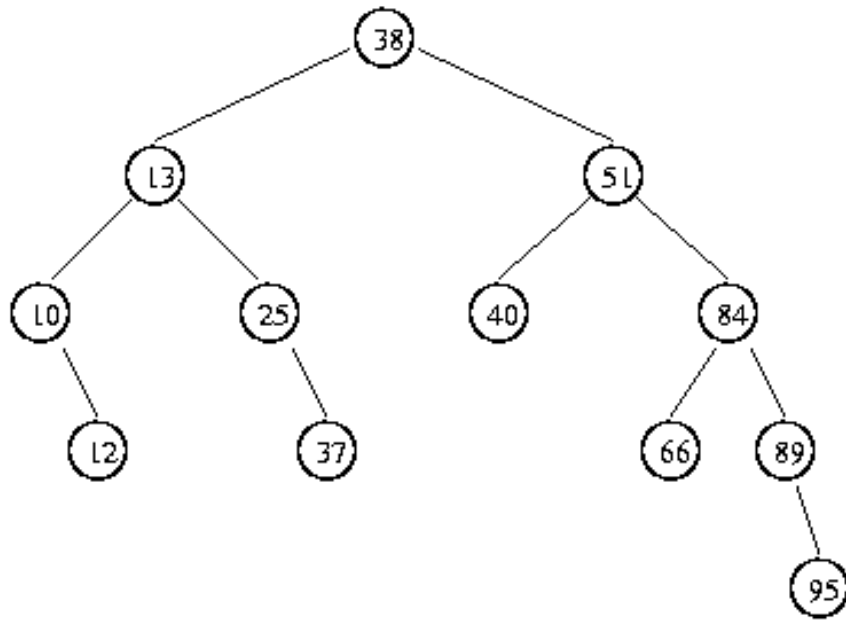
    else if (cRoot->key == key)

    else if

    else

}
```

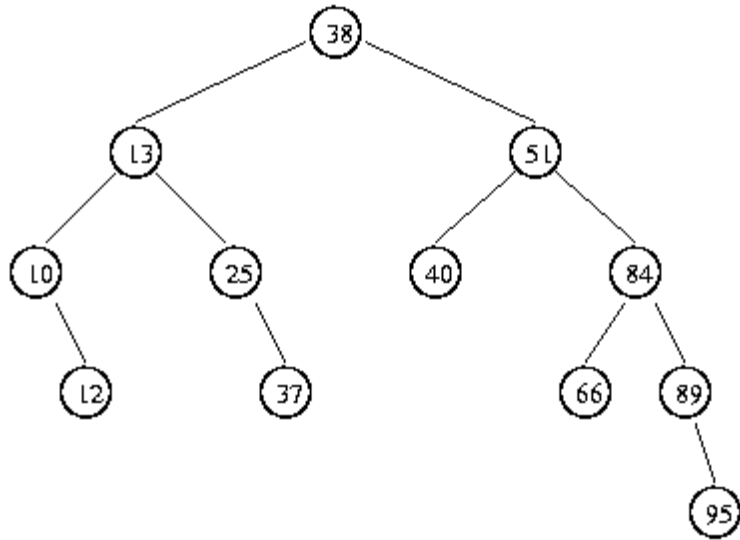
Binary Search Tree - Insert



Running time:

```
(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



Running time:

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