Announcements

1. MP1 Extra Credit due Tonight!

Warm-Up Question: If you are searching through an infinite tree, should you use Breadth-First Search or Depth-First Search?

Iterative Deepening Search/Binary Search Trees

Learning Objectives

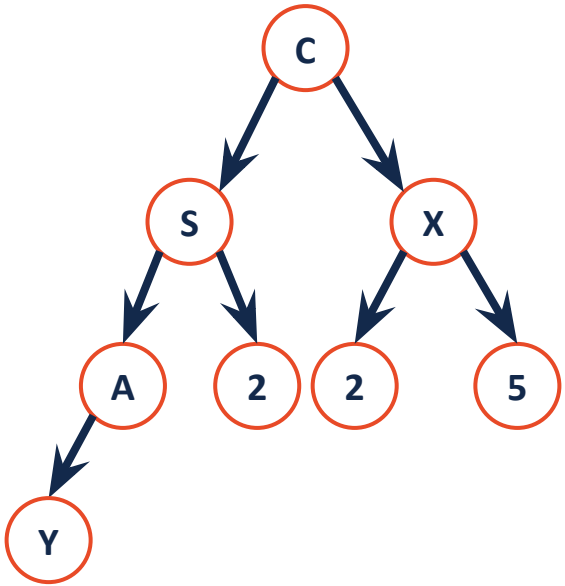
1. Understand Iterative Deepening
2. Know the runtime of iterative deepening on a binary tree
3. Understand the Dictionary ADT

Tree Search

Breadth-First -

Depth-First -

Iterative Deepening



Iterative Deepening Analysis

Dictionary

Stores key -> value pairs

Applications:

What applications of dictionaries can you think of?

Dictionary ADT

Insert ( )

Remove ( )

Find ( )

```
1  #pragma once
2
3
4  class Dictionary {
5    public:
6      void insert(const K & key, V & value);
7      V remove(const K & key);
8      V find(const K & key) const;
9      iterator begin();
10     iterator end();
11     // ...
12
13   private:
14     // ...
15
16
17
18
19 };
```

```
1  #pragma once
2
3
4  class Dictionary {
5    public:
6      void insert(const K & key, V & value);
7      V remove(const K & key);
8      V find(const K & key) const;
9      iterator begin();
10     iterator end();
11     // ...
12
13   private:
14     // ...
15     TreeNode * root_;
16
17
18
19 };
```

What if you are trying to find a key that doesn't exist?

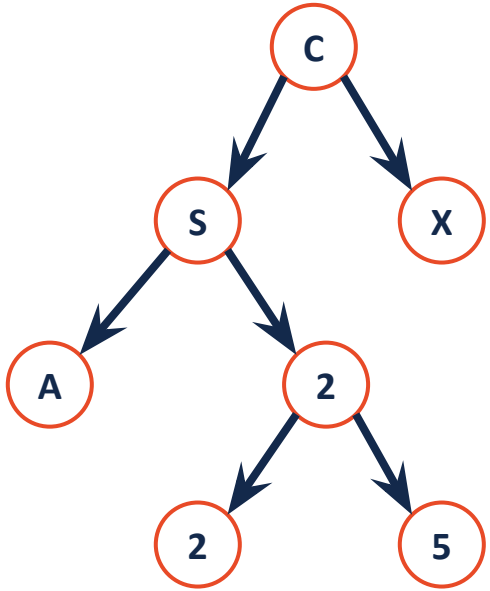
Binary Tree Definition

A *binary tree* T is either:

$$T = \emptyset$$

OR

$$T = (r, T_L, T_R)$$



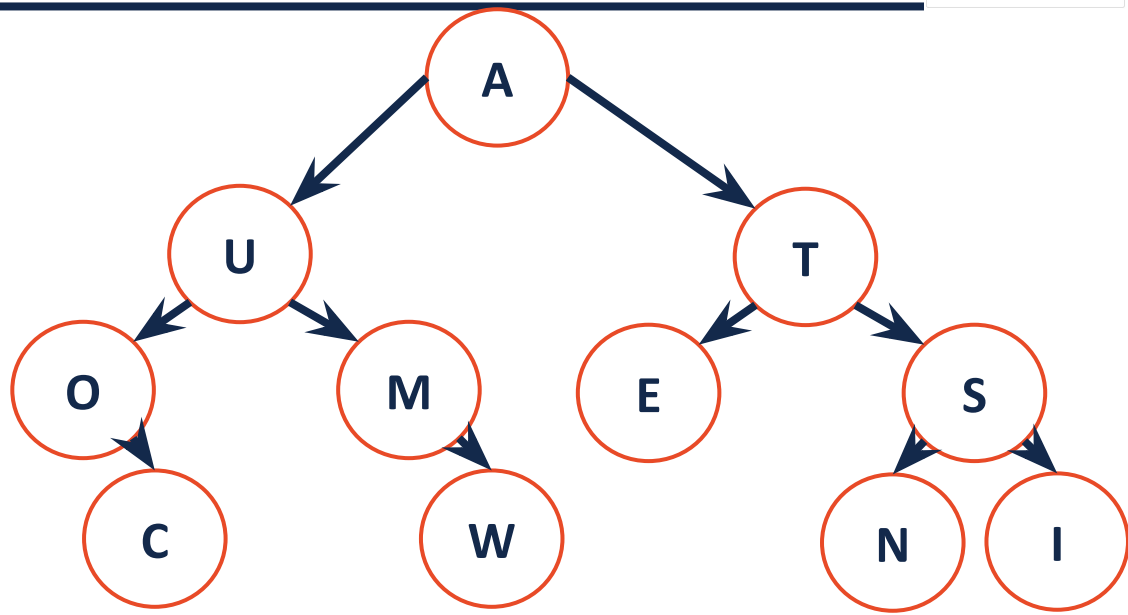
Searching through Binary Trees

Join Code: **225**

Insert (key, value)

Remove (key)

value Find (key)



Function	Runtime
Insert	
Remove	
Find	

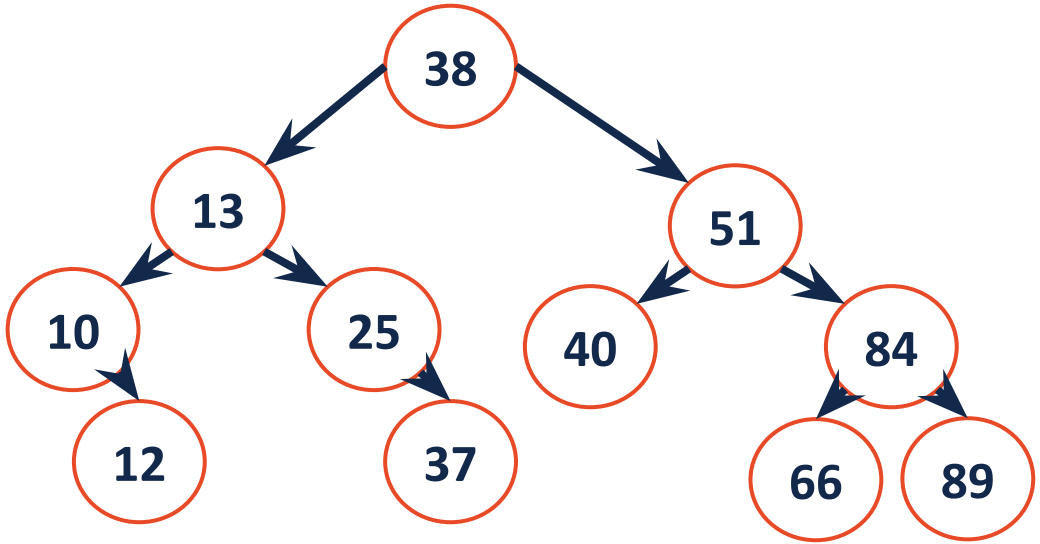
Binary Search Tree Definition

A *binary search tree* T is either:

$$T = \emptyset$$

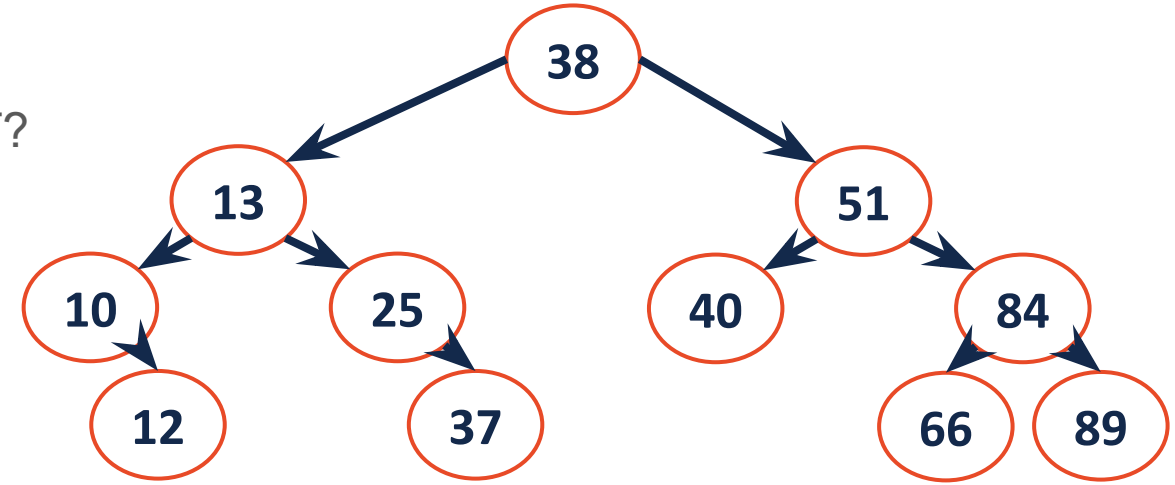
OR

$$T = (r, T_L, T_R)$$



Binary Search Trees (BSTs)

How do my functions change given the structure of the BST?



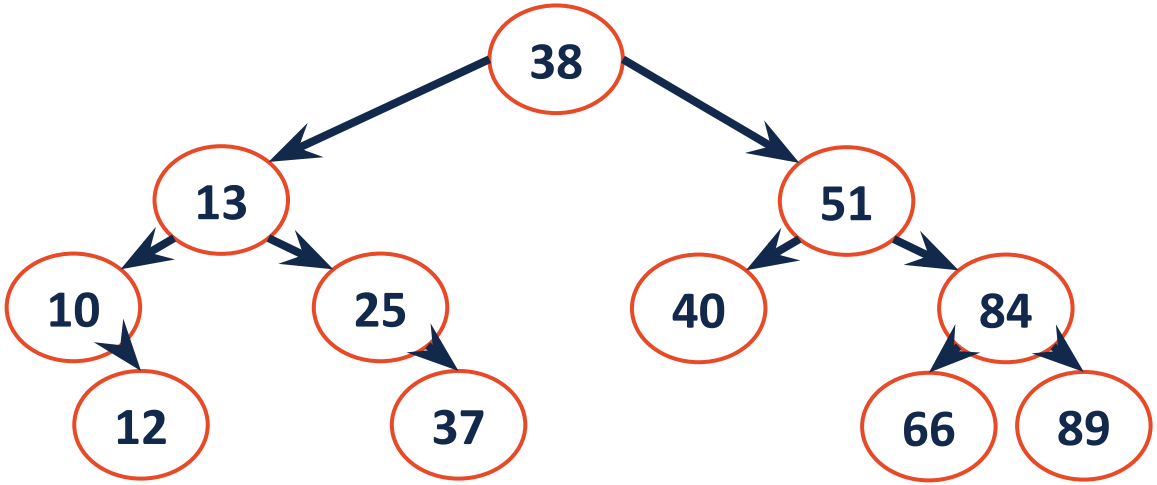
BST Runtimes

Join Code: **225**

Insert (key, value)

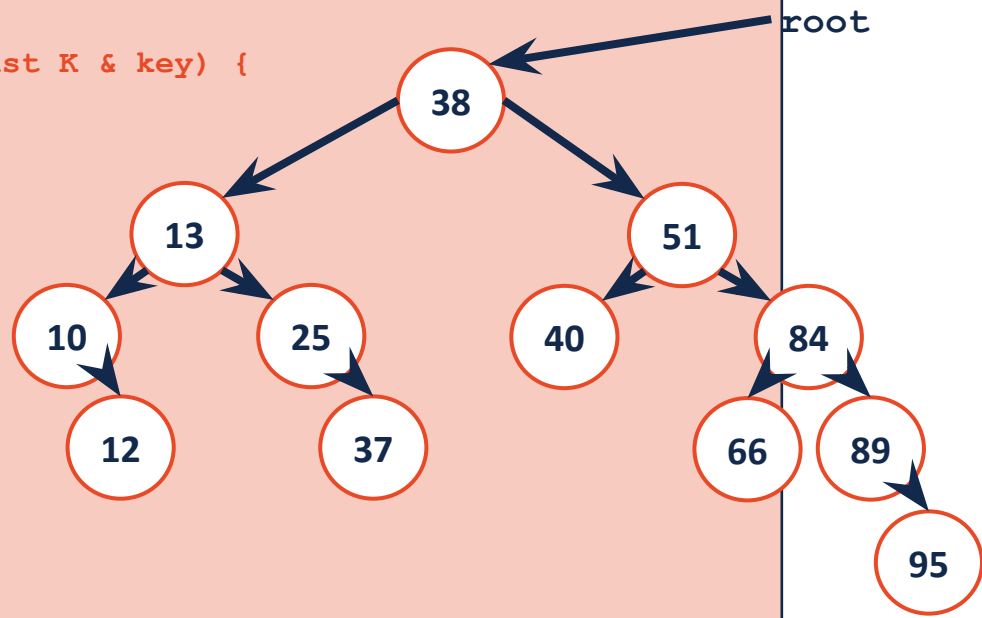
Remove (key)

value Find (key)



Function	Runtime
Insert	
Remove	
Find	

```
1  template<class K, class V>
2  TreeNode*& find(TreeNode *& root, const K & key) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 }
```



```
1  template<class K, class V>
2  TreeNode*& find(TreeNode *& root, const K & key) {
3
4       if (root == NULL){
5            return root;
6       }
7
8       if (root->key == key){
9            return root;
10      }
11
12      if(key < root->key){
13           return find(root->left, key);
14      }
15
16      if(key > root->key){
17           return find(root->right,key);
18      }
19
20
21
22
23
24
25 }
26
```