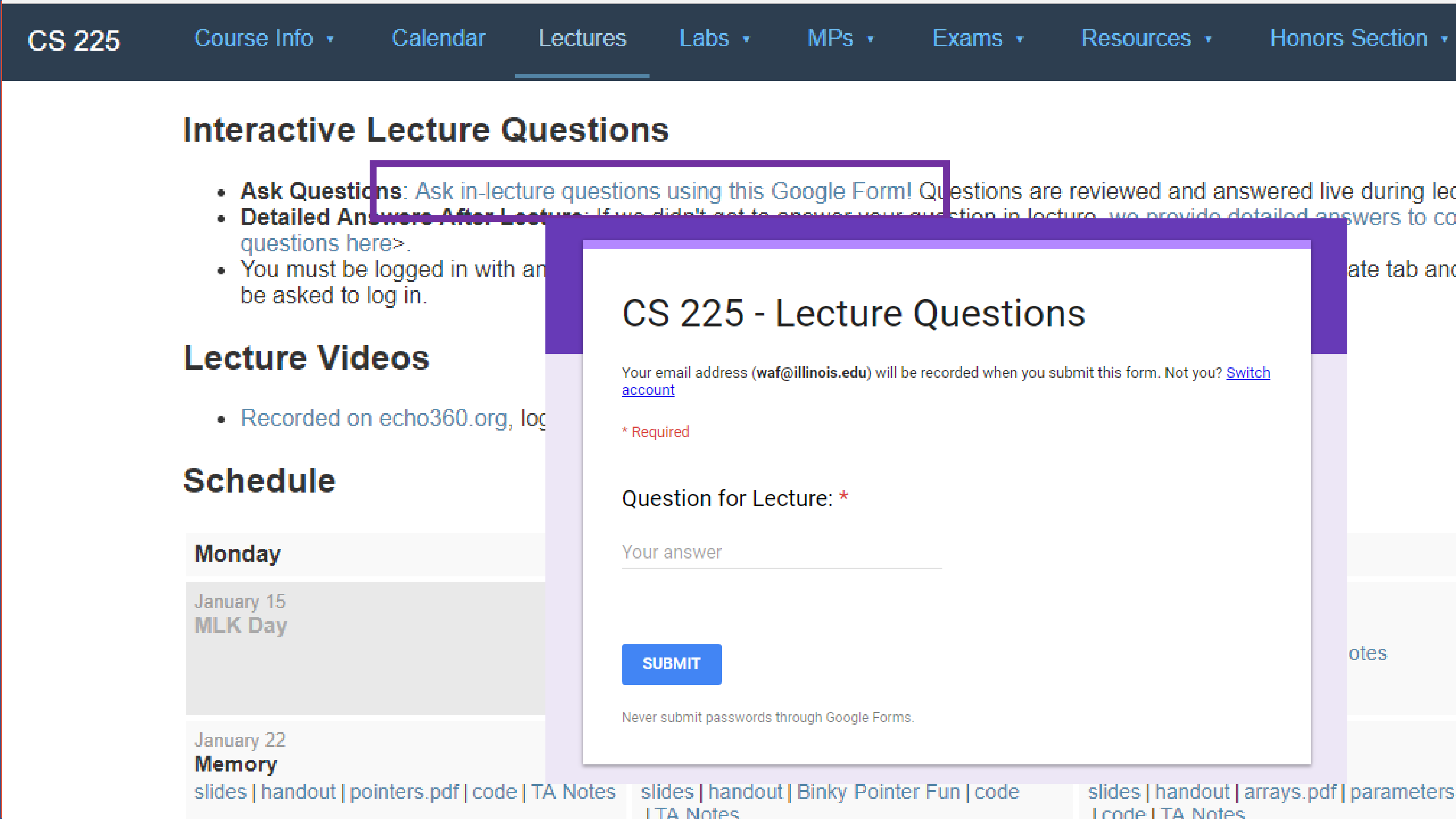
CS 225

Data Structures

Feb. 21 – Binary Search Tree

Wade Fagen-Ulmschneider



Interactive Lecture Questions

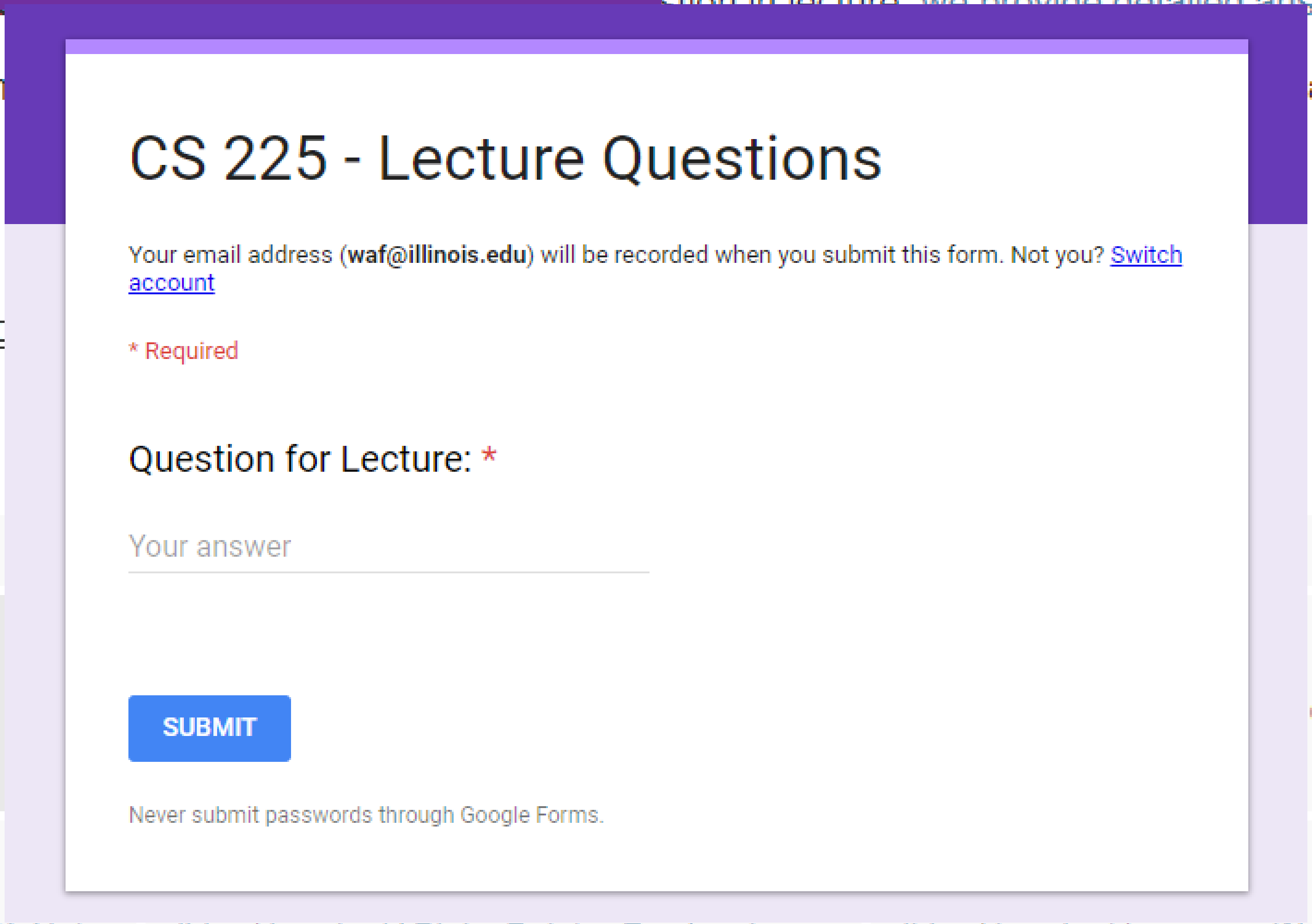
- **Ask Questions**: Ask in-lecture questions using this Google Form! Questions are reviewed and answered live during lec
- **Detailed Answers After Lecture**: If we didn't get to answer your question in lecture, we provide detailed answers to co questions here>.
- You must be logged in with an ate tab and be asked to log in.

Lecture Videos

- Recorded on echo360.org, log

Schedule

Monday		
January 15 **MLK Day**		otes
January 22 **Memory** slides | handout | pointers.pdf | code | TA Notes	slides | handout | Binky Pointer Fun | code | TA Notes	slides | handout | arrays.pdf | parameters | code | TA Notes



CS 225 - Lecture Questions

Your email address (**waf@illinois.edu**) will be recorded when you submit this form. Not you? Switch account

* Required

Question for Lecture: *

Your answer

Never submit passwords through Google Forms.

Traversal vs. Search

Traversal vs. Search:

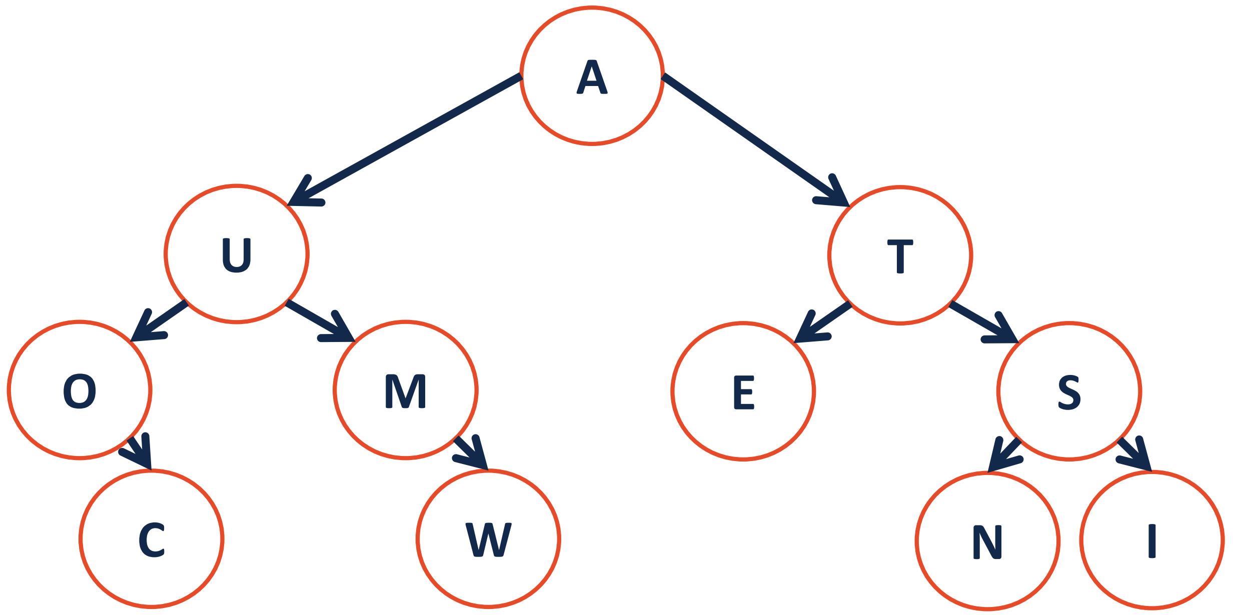
- **Traversal** visits every node in the tree exactly once.
- **Search** finds one element in the tree.

Search: Breadth First vs. Depth First

Strategy: Breadth First Search (BFS) / Traversal

Strategy: Depth First Search (DFS) / Traversal

Running Times on a Binary Tree



Dictionary ADT

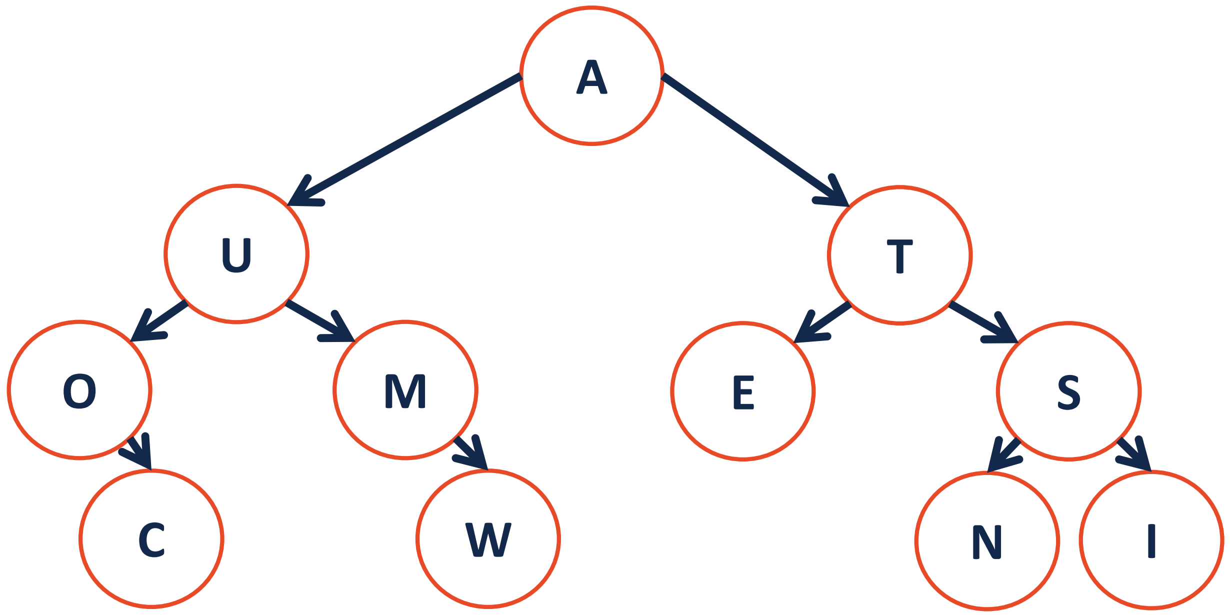
Data is often organized into key/value pairs:

UIN ➔ Advising Record
Course Number ➔ Lecture/Lab Schedule
Node ➔ Incident Edges
Flight Number ➔ Arrival Information
URL ➔ HTML Page
…

Dictionary.h

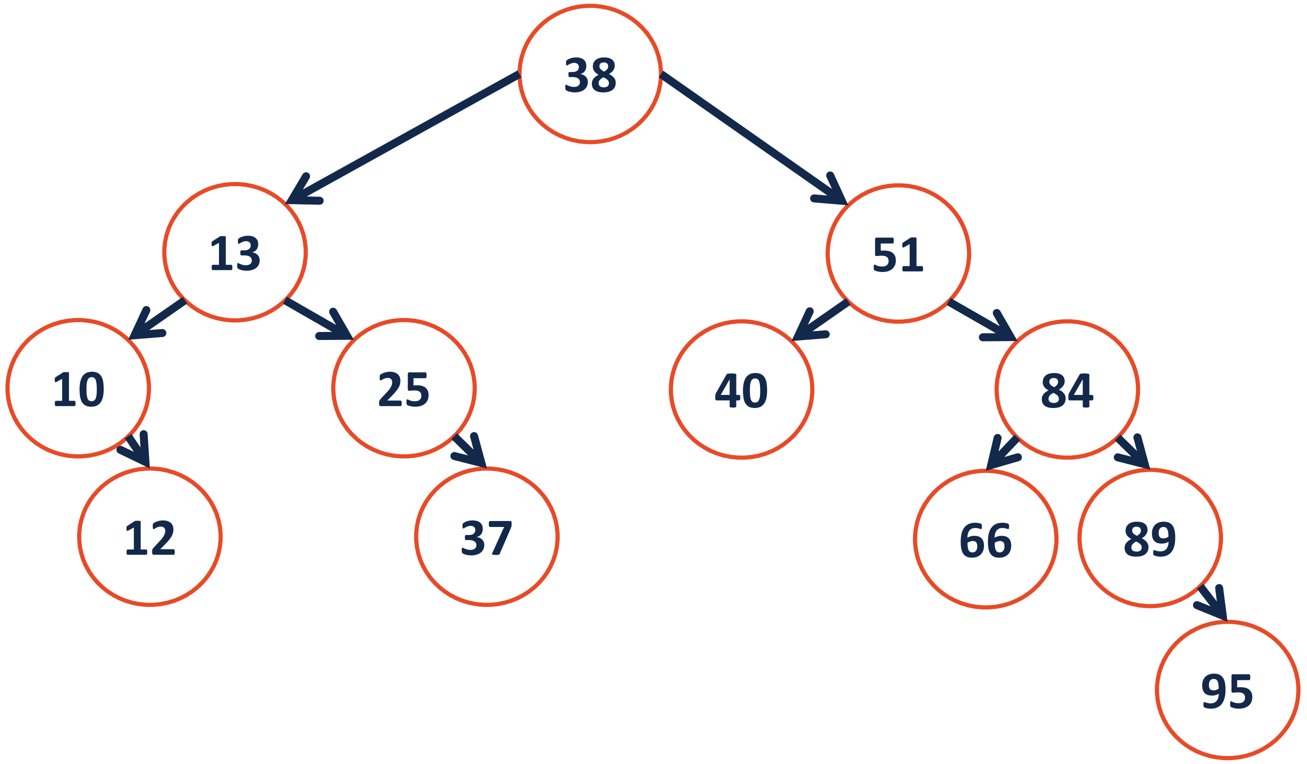
```
#ifndef DICTIONARY_H
#define DICTIONARY_H


class Dictionary {
  public:









  private:



};

#endif
```

Binary Tree as a Search Structure



Binary ______________ Tree (BST)

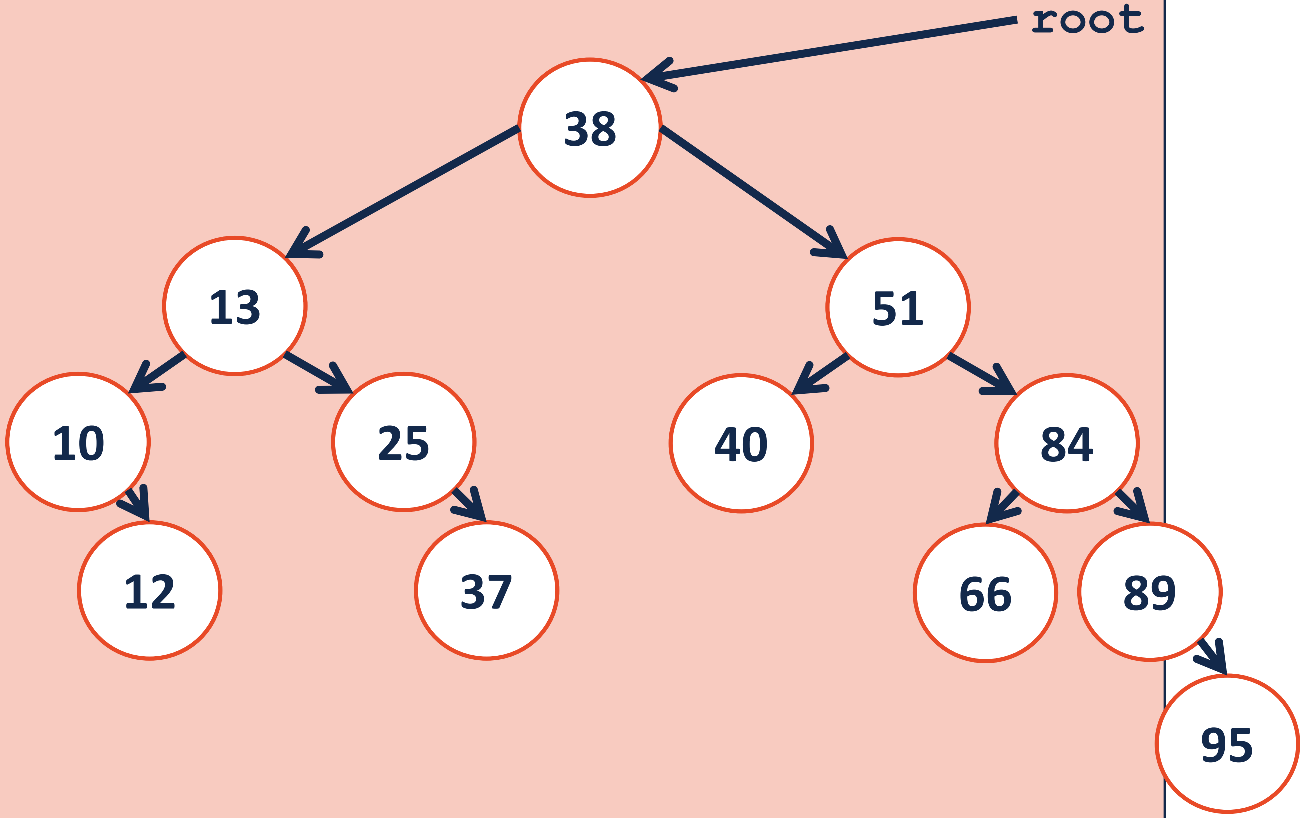
A **BST** is a binary tree **T** such that:

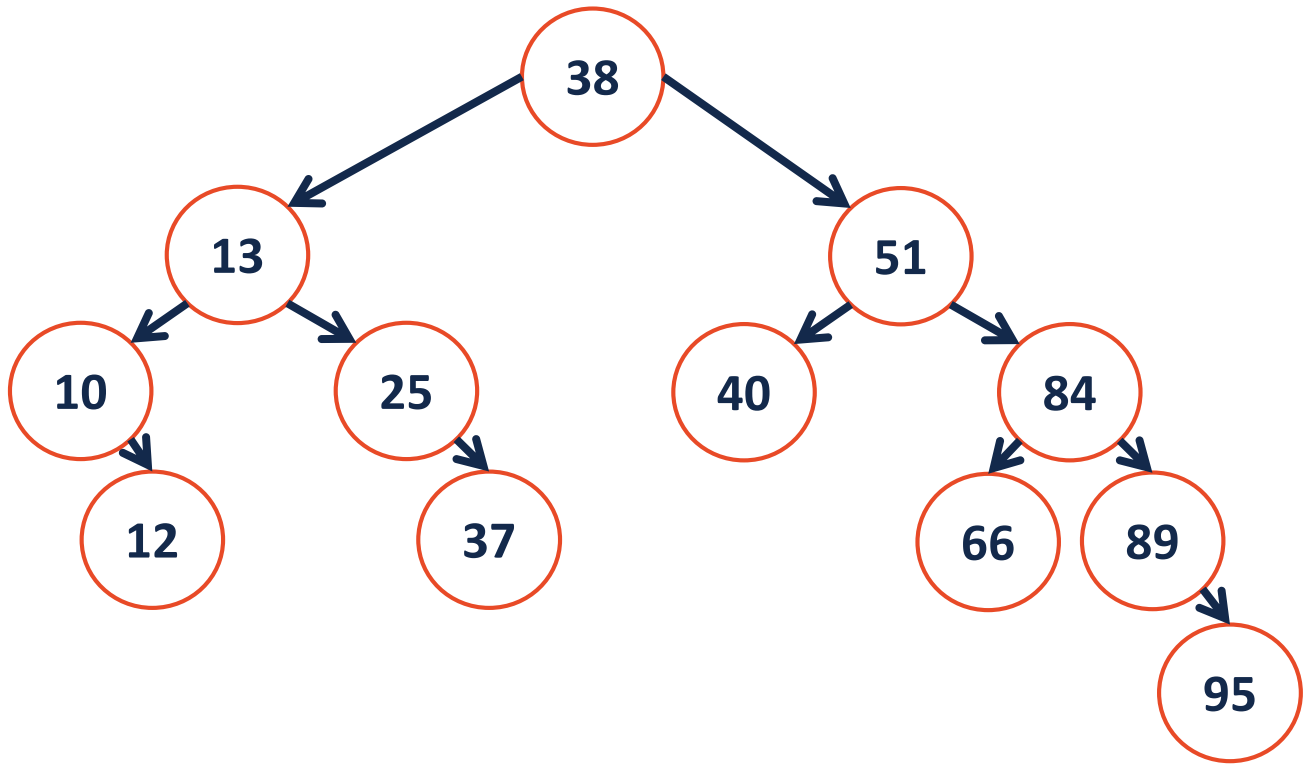


BST.h

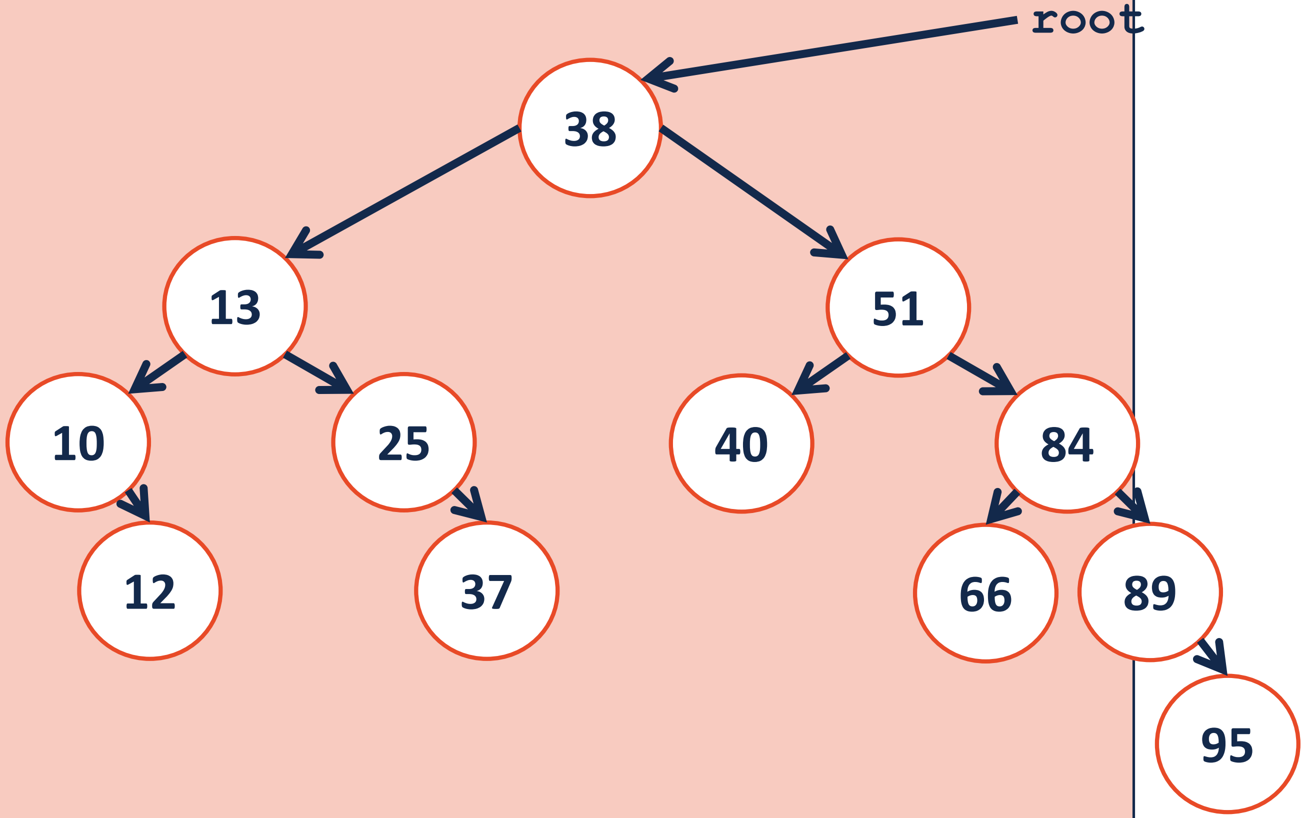
```
#ifndef DICTIONARY_H
#define DICTIONARY_H

template <class K, class V>
class BST {
  public:
    BST();
    void insert(const K key, V value);
    V remove(const K & key);
    V find(const K & key) const;
    TreeIterator traverse() const;
  private:
    struct TreeNode {
       TreeNode *left, *right;
       K & key;
       V & value;
       TreeNode(K & k, V & v) : key(k), value(v), left(NULL),
          right(NULL) { }
    };
};

#endif
```

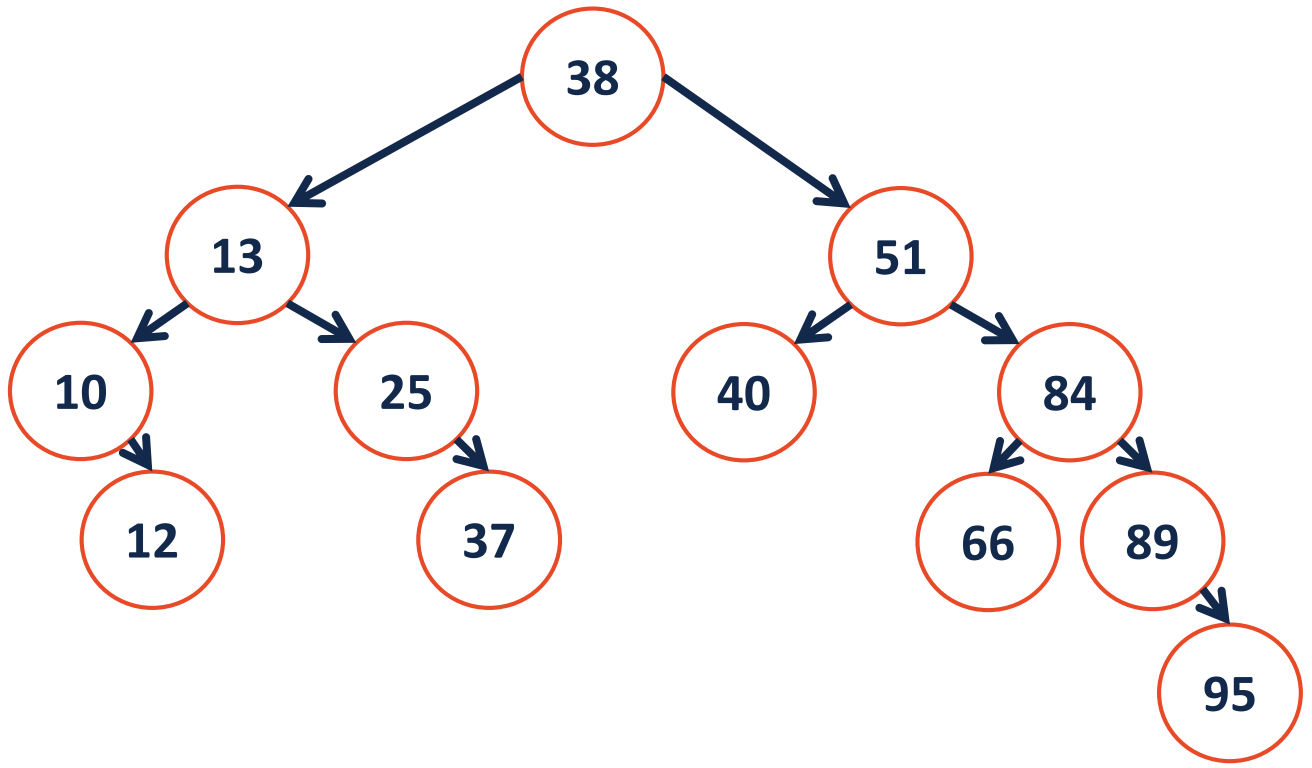
```
template<typename K, typename V>

_______________________ _find(TreeNode *& root, const K & key) const {
























}
```

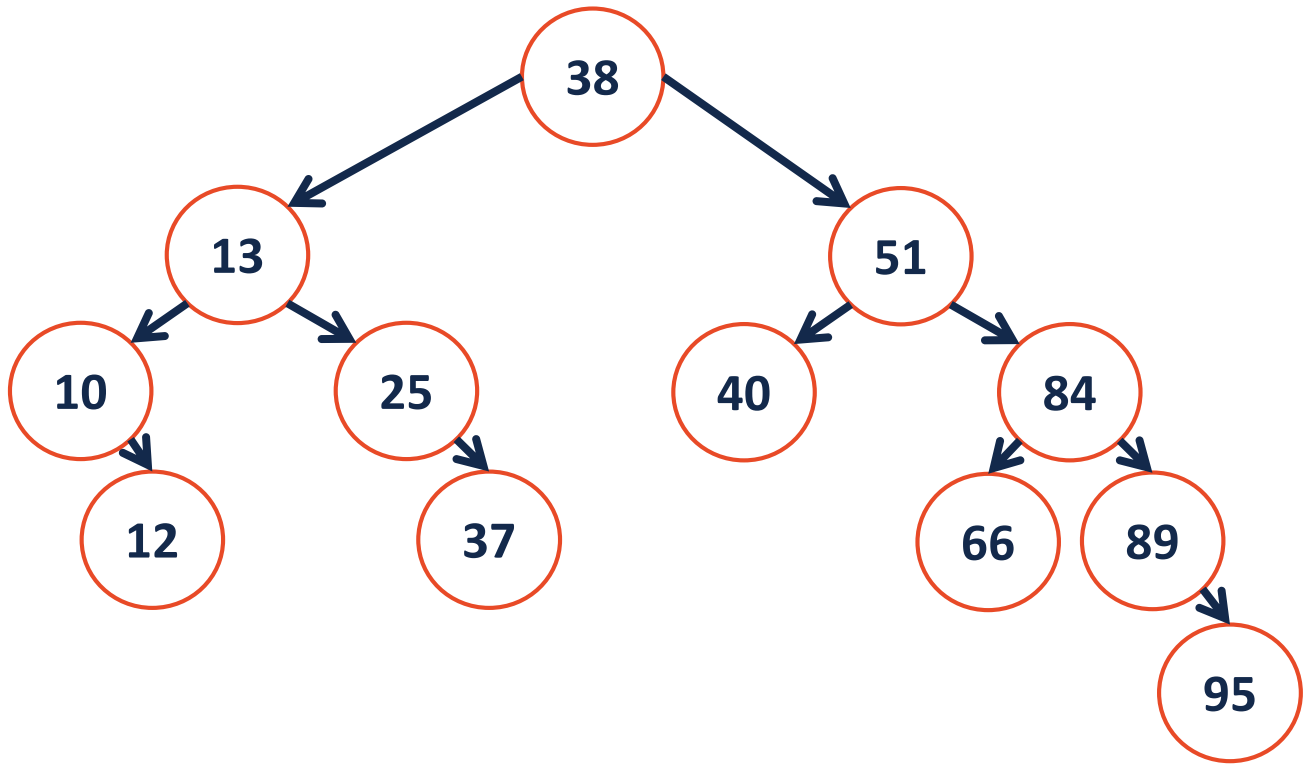



38
13
51
10
25
40
84
12
37
66
89
95

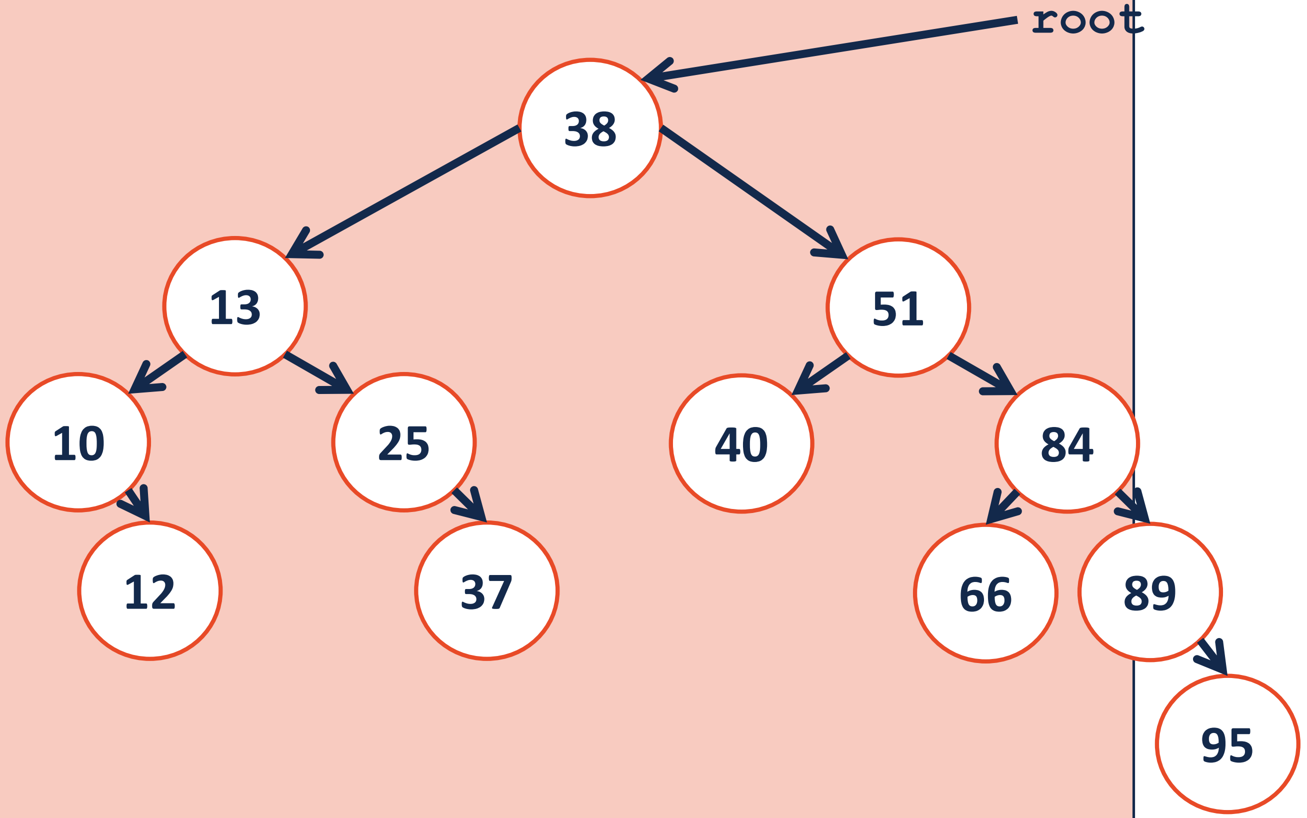
```
template<typename K, typename V>

_______________________ _insert(TreeNode *& root, const K & key) {
























}
```

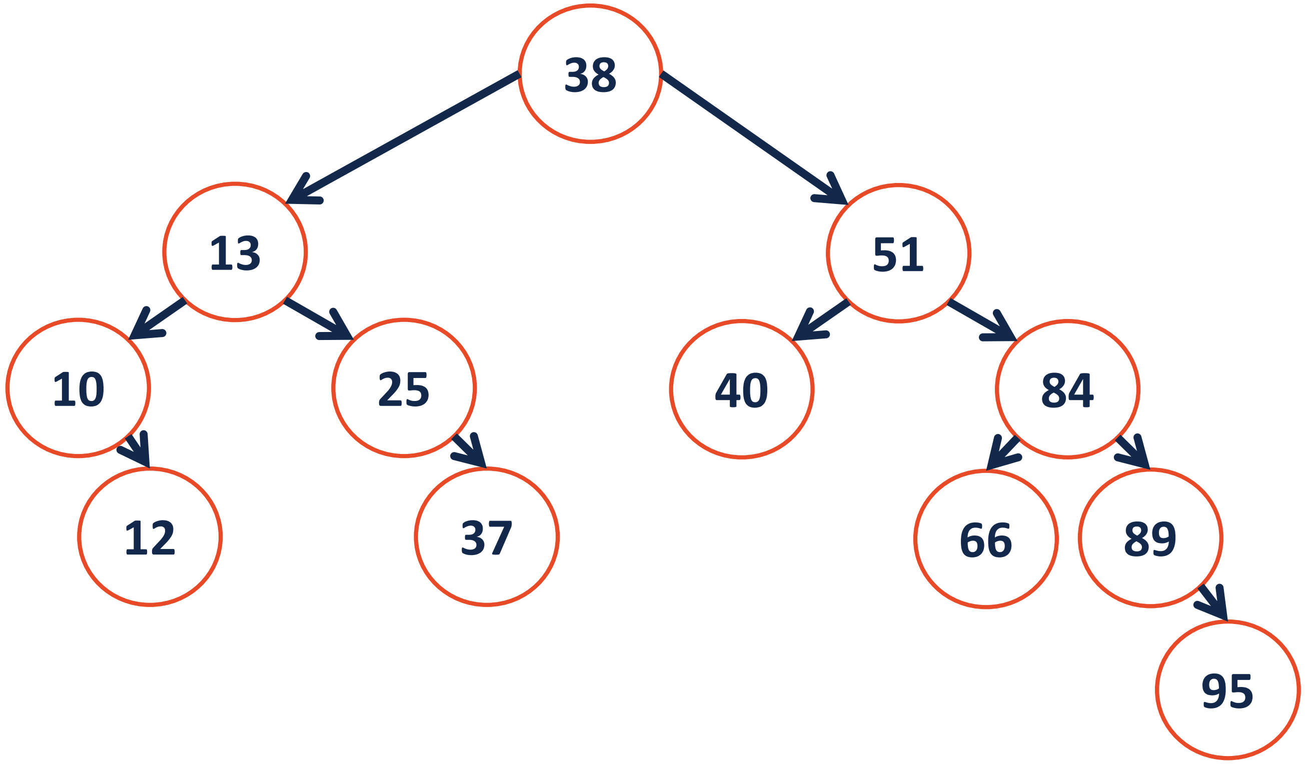



38
13
51
10
25
40
84
12
37
66
89
95

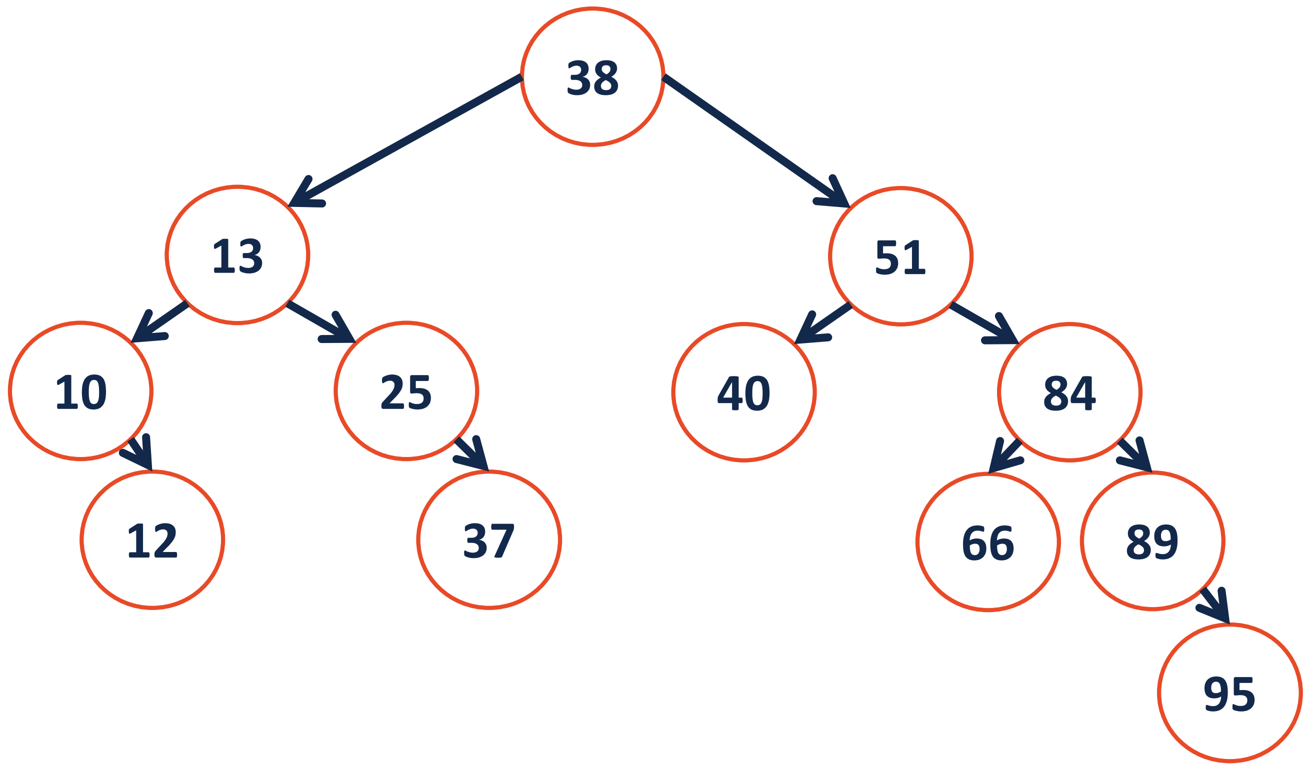

38
13
51
10
25
40
84
12
37
66
89
95

```
template<typename K, typename V>
_________________________ _remove(TreeNode *& root, const K & key) {
























}
```

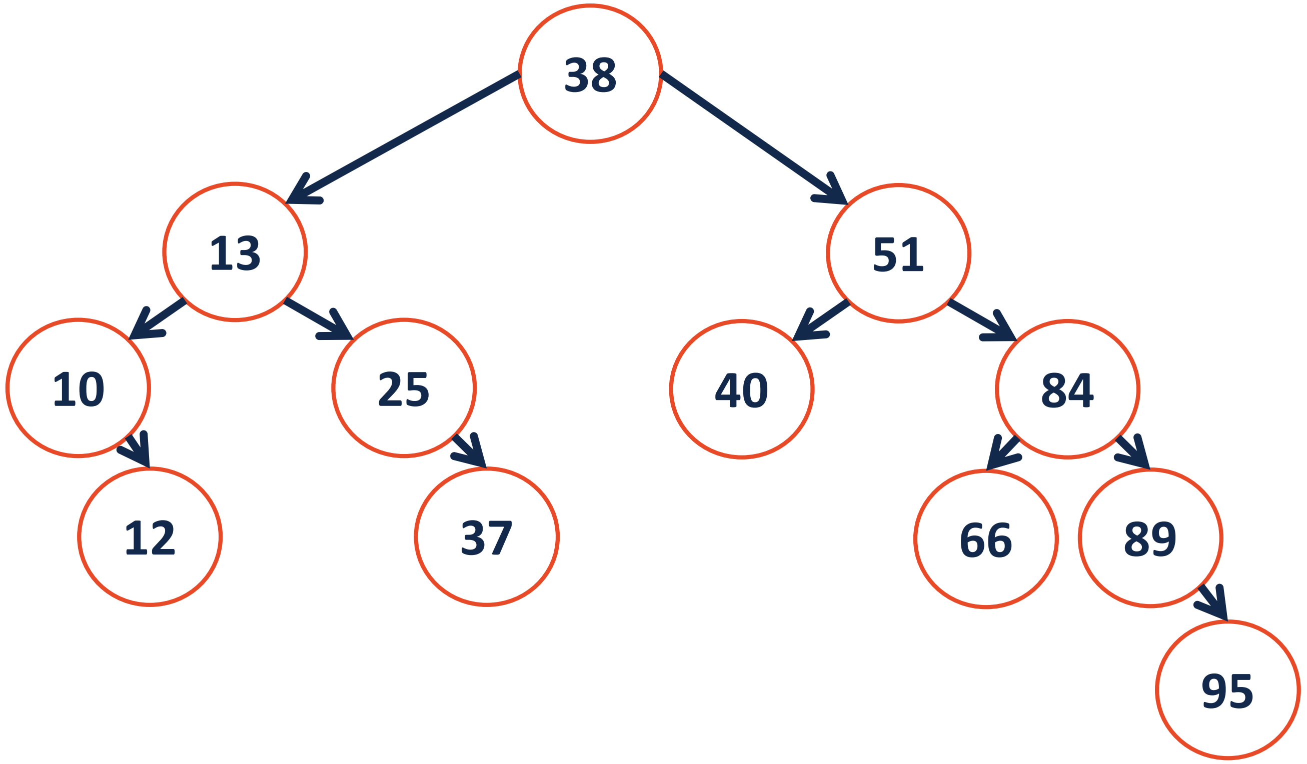




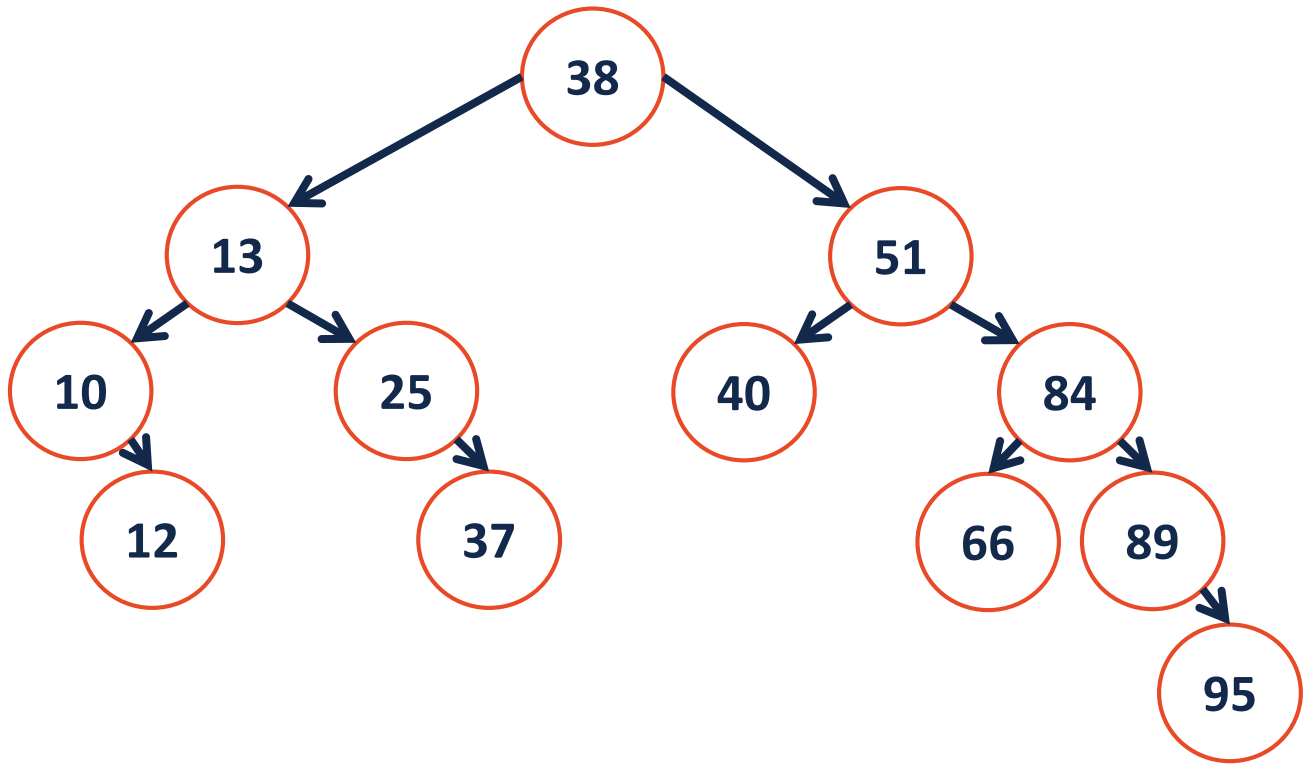
```
remove(40);
```



```
remove(25);
```



```
remove(10);
```



```
remove(13);
```