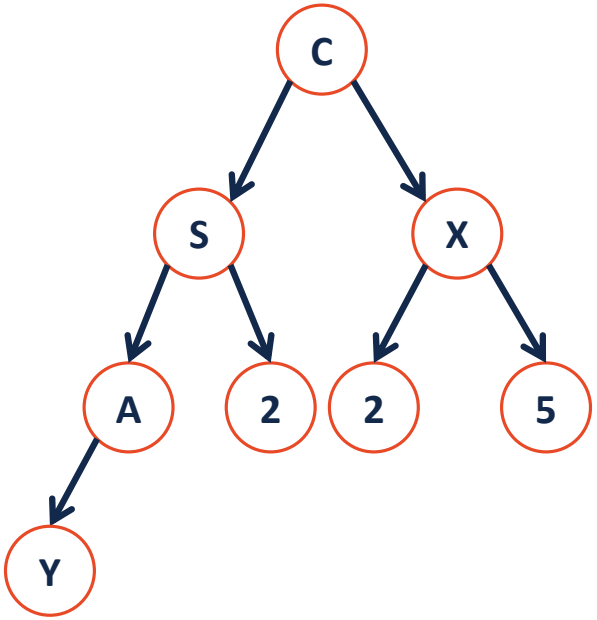
CS 225

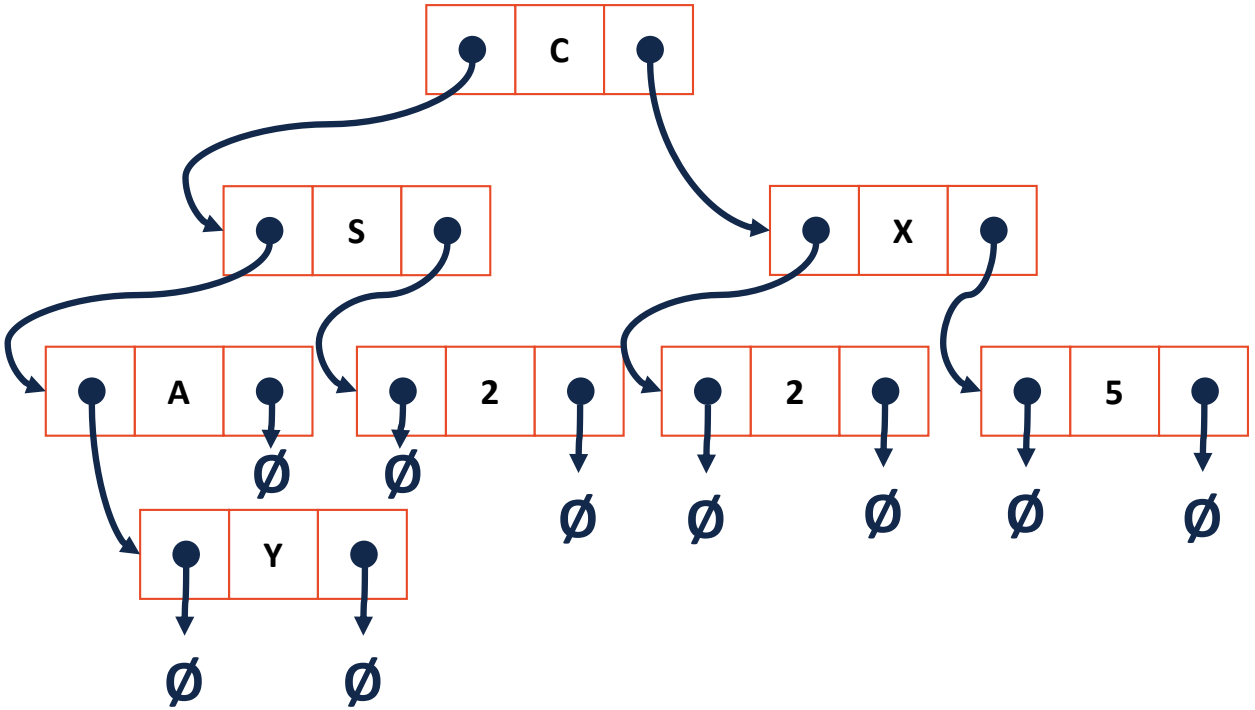
Data Structures

March 1 – Proof and Traversal

G Carl Evans

Trees aren't new:





How many nullptrs?

Theorem: If there are **n** data items in our representation of a binary tree, then there are _____ **nullptr**s.

How many nullptrs?

Base Cases:

NULLS(0):

NULLS(1):

NULLS(2):

How many nullptrs?

Base Cases:

NULLS(3):

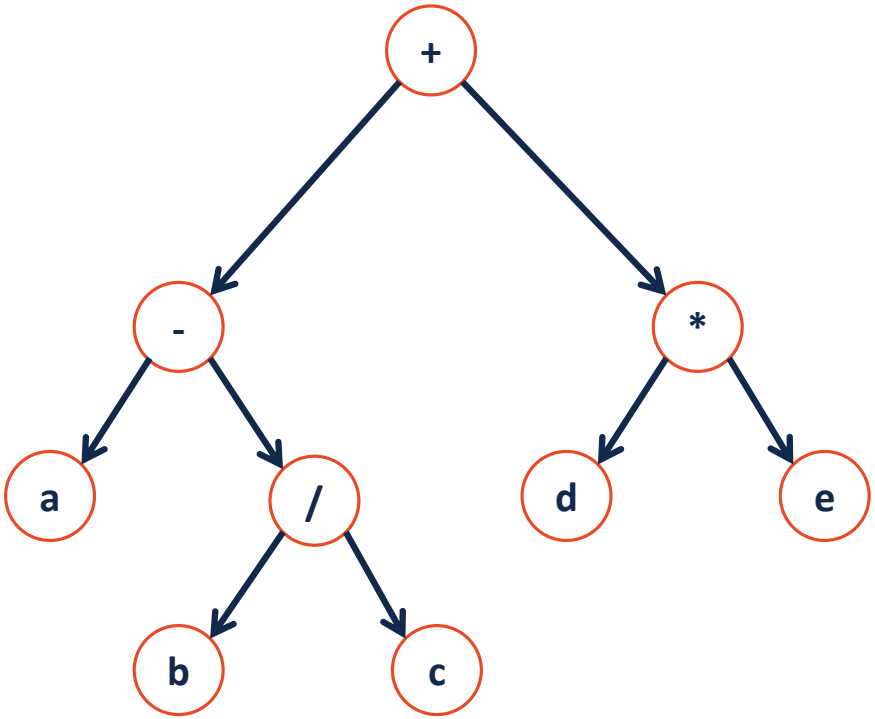
How many nullptrs?

Induction Hypothesis:

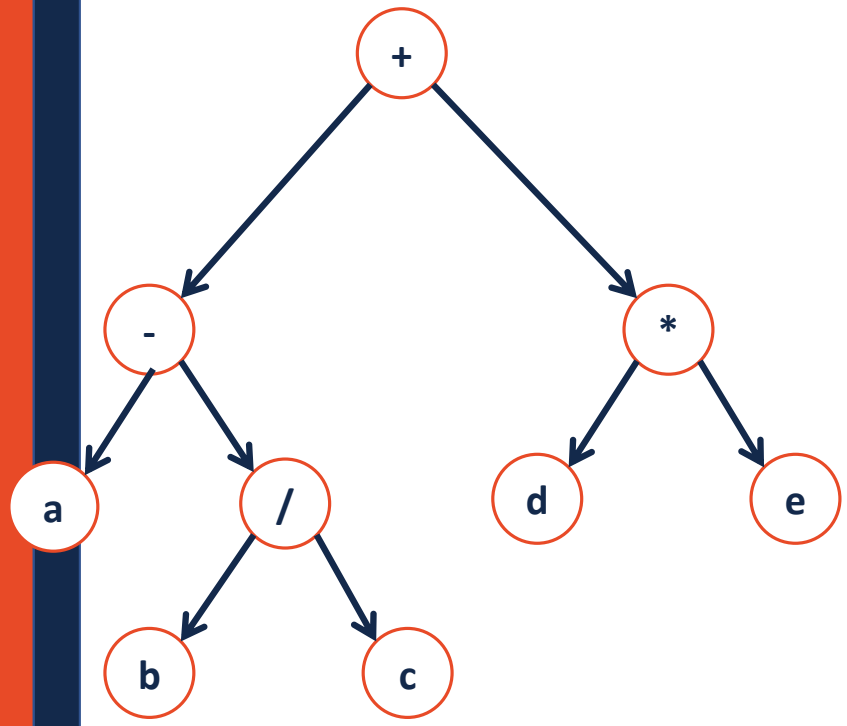
How many nullptrs?

Consider an arbitrary tree **T** containing **k** nodes:

Traversals

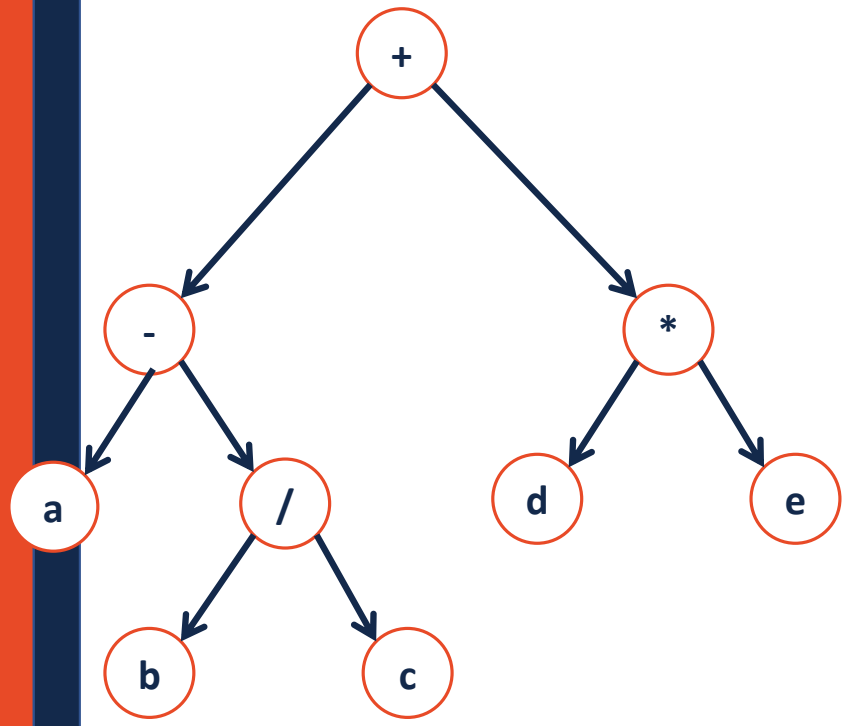


Traversals



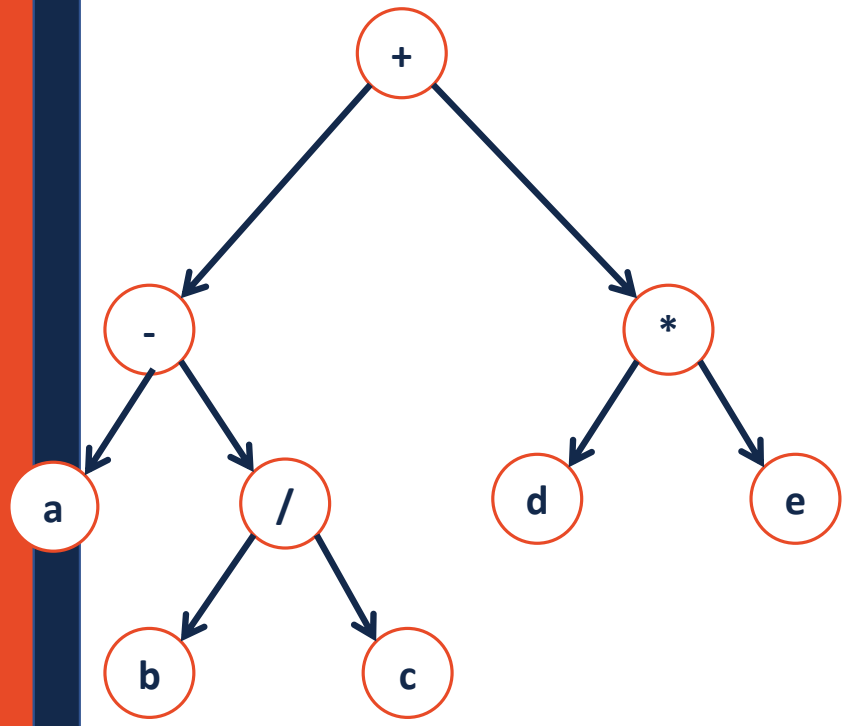

```
49  template<class T>
50  void BinaryTree<T>::__Order(TreeNode * cur)
51  {
52
53
54
55
56
57
58  }
```

Traversals



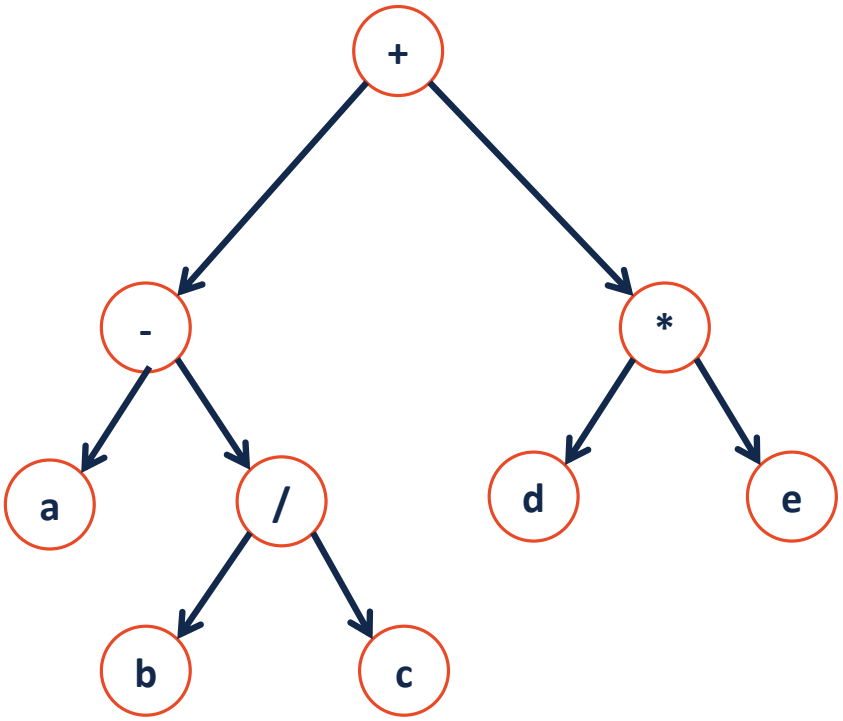

```
49  template<class T>
50  void BinaryTree<T>::___Order(TreeNode * cur) {
51     if (cur != NULL) {

52        ____________________________;

53        ___Order(cur->left);

54        ____________________________;

55        ___Order(cur->right);

56        ____________________________;

57     }
58  }
```

Traversals

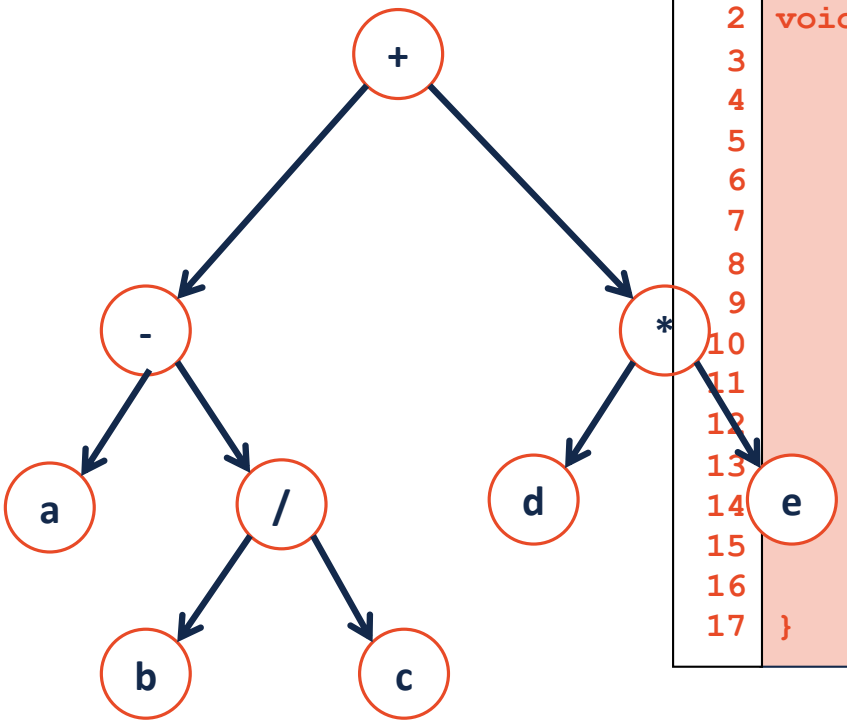



```
49  template<class T>
50  void BinaryTree<T>::___Order(TreeNode * cur) {
51     if (cur != NULL) {

52        ____________________________;

53        ___Order(cur->left);

54        ____________________________;

55        ___Order(cur->right);

56        ____________________________;

57     }
58  }
```

A Different Type of Traversal



A Different Type of Traversal




```
1  template<class T>
2  void BinaryTree<T>::levelOrder(TreeNode * root) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17 }
```

Traversal vs. Search

Traversal

Search

Search: Breadth First vs. Depth First

Strategy: Breadth First Search (BFS)

Strategy: Depth First Search (DFS)

Dictionary ADT

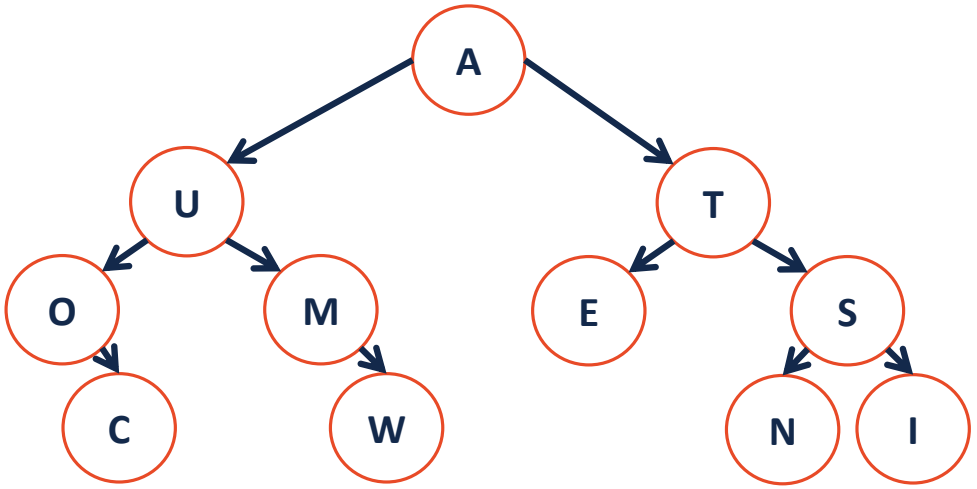
Data is often organized into key/value pairs:

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

Dictionary.h

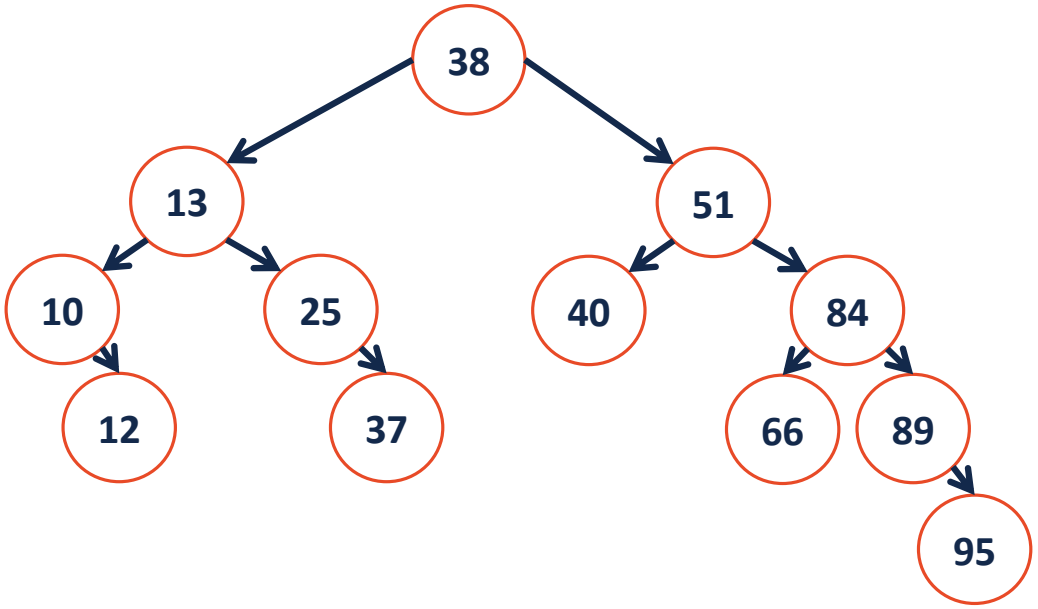
```
1  #pragma once
2
3
4  class Dictionary {
5    public:
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20   private:
21     // ...
22 };
```

Binary Tree as a Search Structure



Binary _____ Tree (BST)

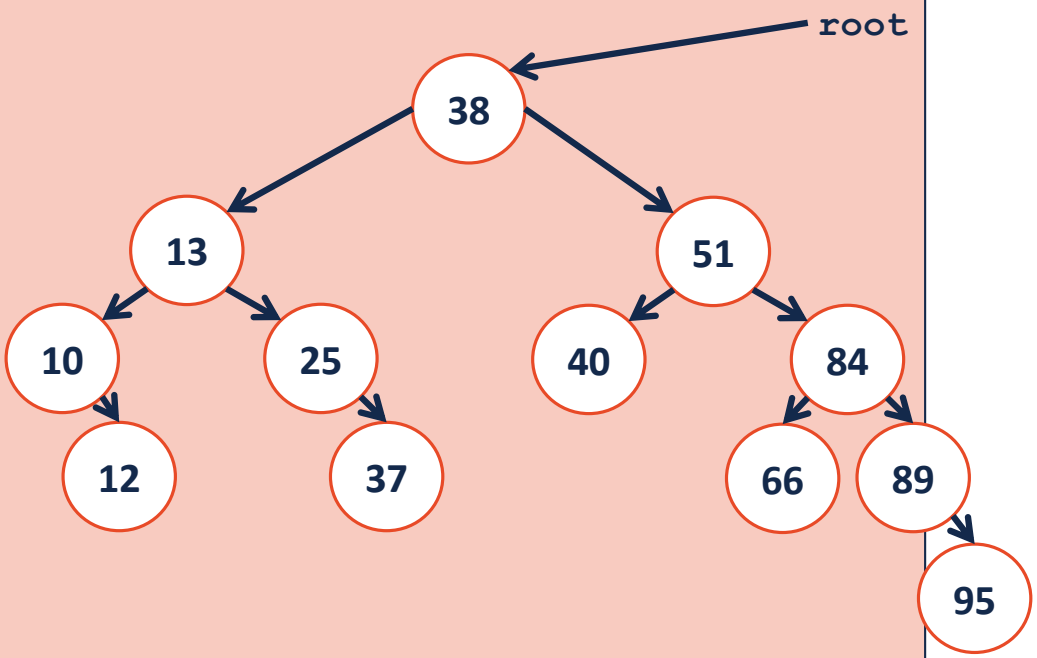
A **BST** is a binary tree **T** such that:

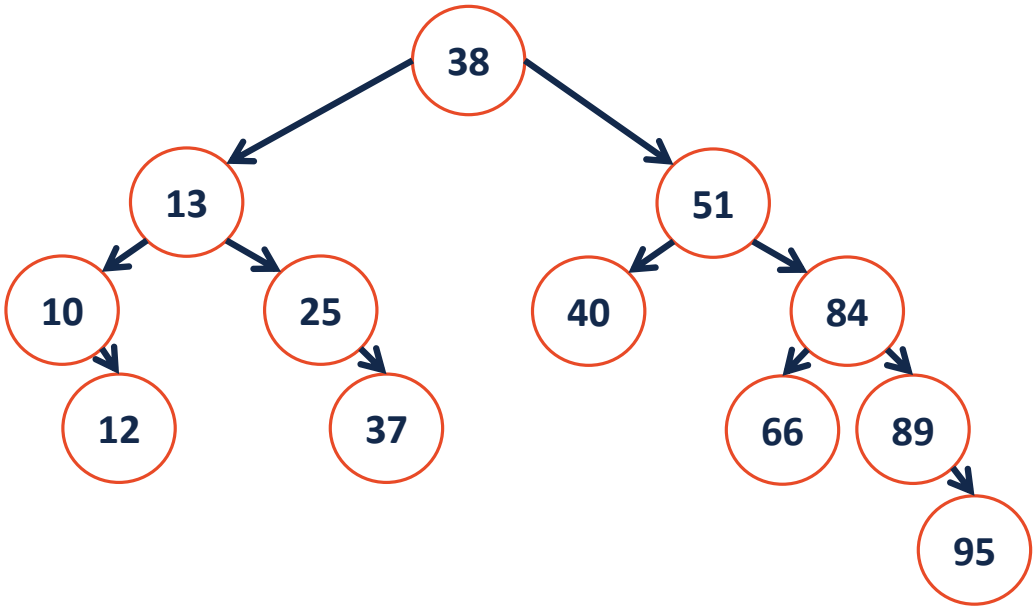


BST.h

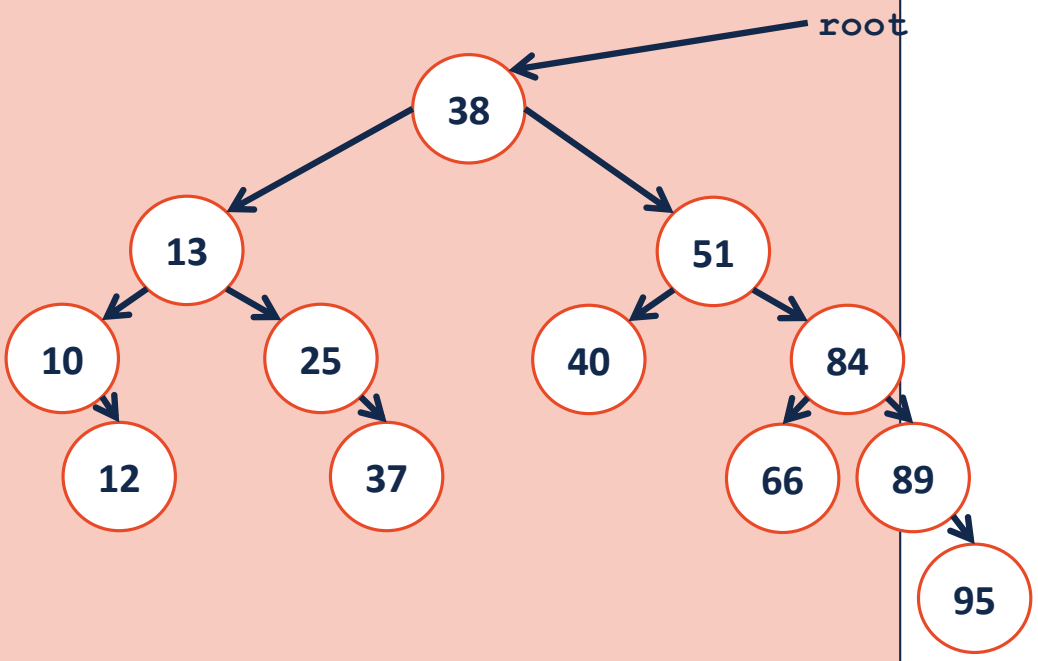
```
 1 #pragma once
 2
 3 template <class K, class V>
 4 class BST {
 5   public:
 6     BST();
 7     void insert(const K key, V value);
 8     V remove(const K & key);
 9     V find(const K & key) const;
10     TreeIterator traverse() const;
11
12   private:
13
14
15
16
17
18
19
20
21
22 };
```

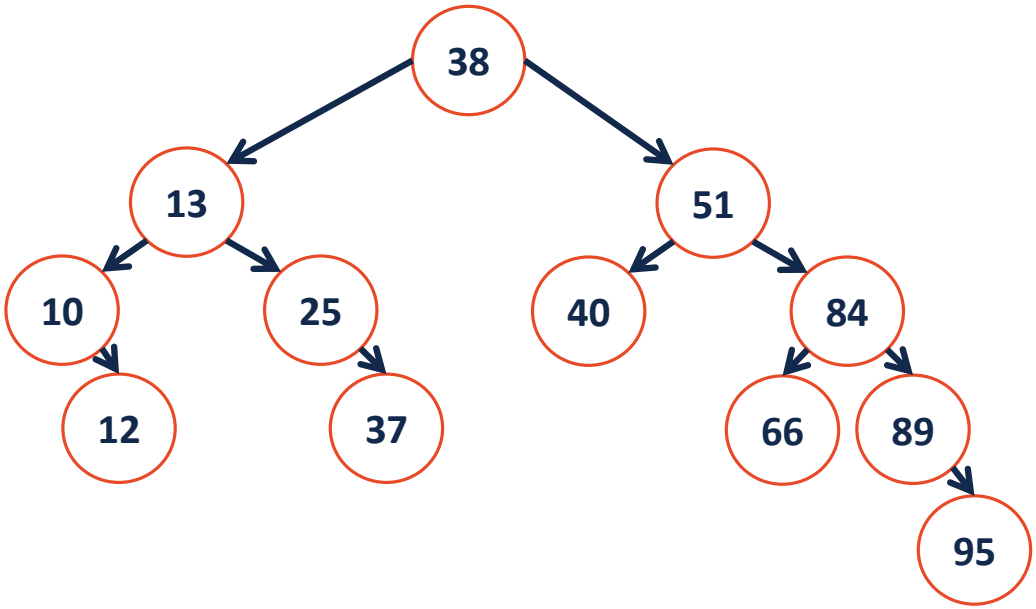
```
1  template<class K, class V>
2  _____________________________ _find(TreeNode *& root, const K & key) const {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 }
```



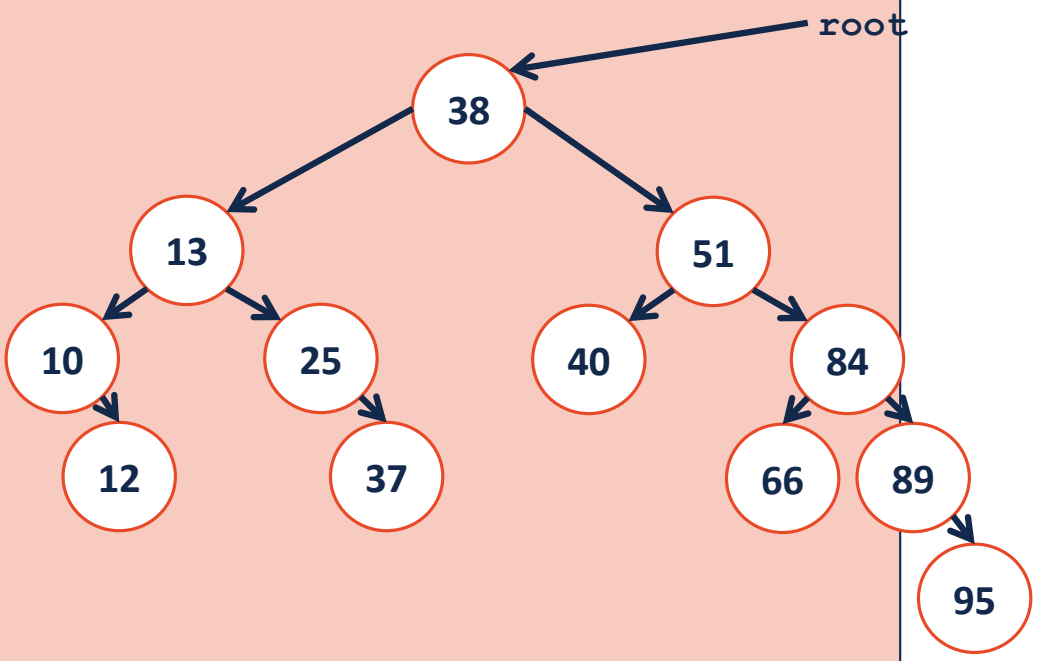

38
13
51
10
25
40
84
12
37
66
89
95

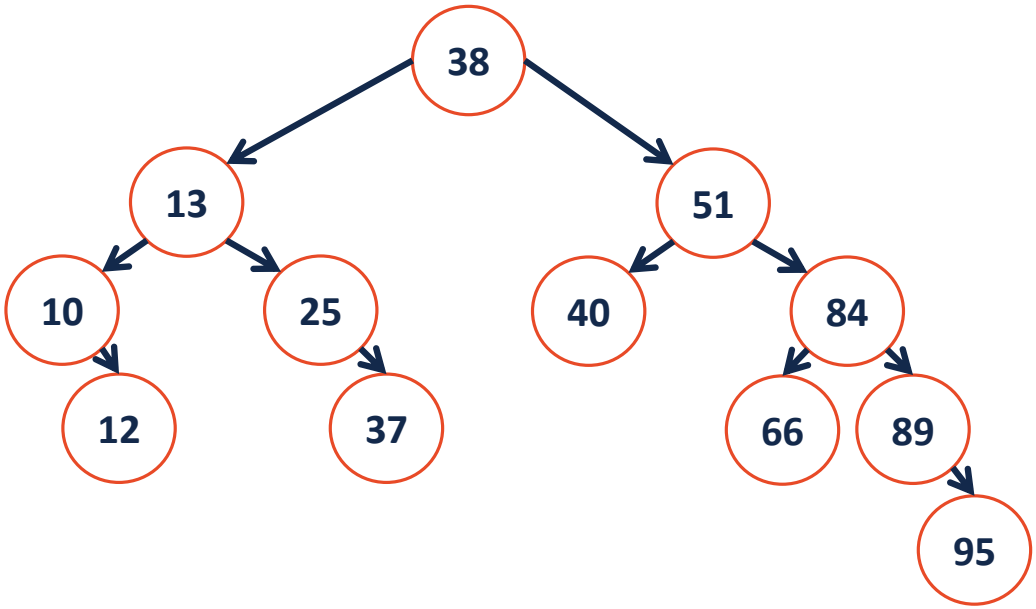
```
1  template<class K, class V>
2  _____________________________ _insert(TreeNode *& root, const K & key) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 }
```



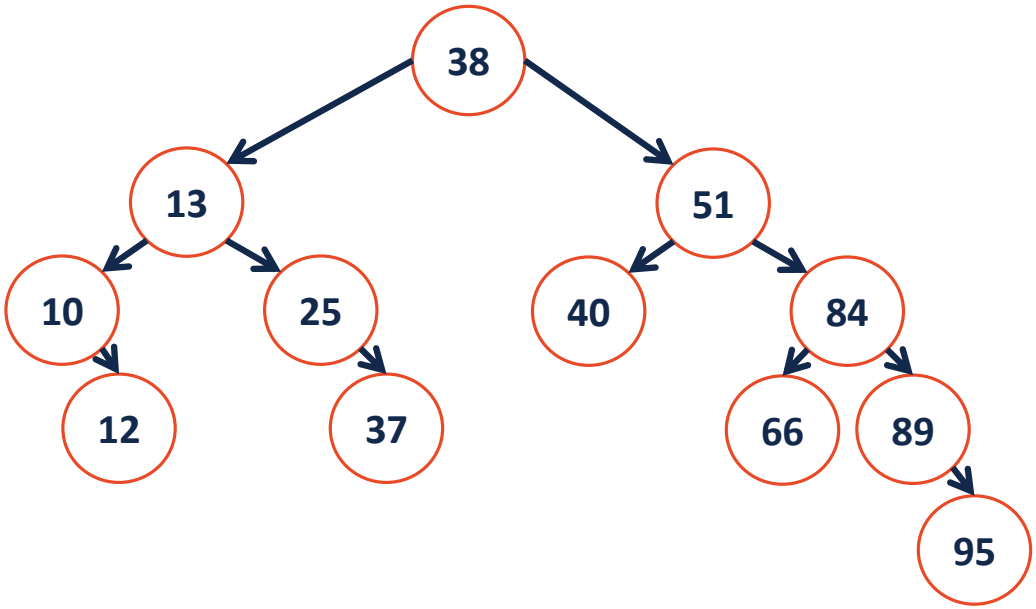

38
13
51
10
25
40
84
12
37
66
89
95

```
1  template<class K, class V>
2  _____________________________ _remove(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 }
```

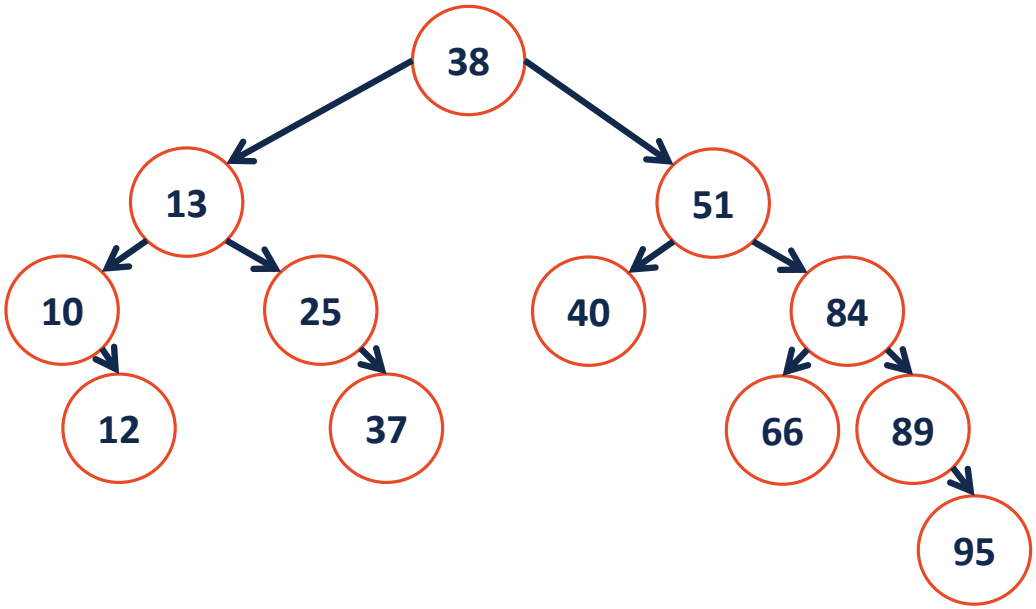




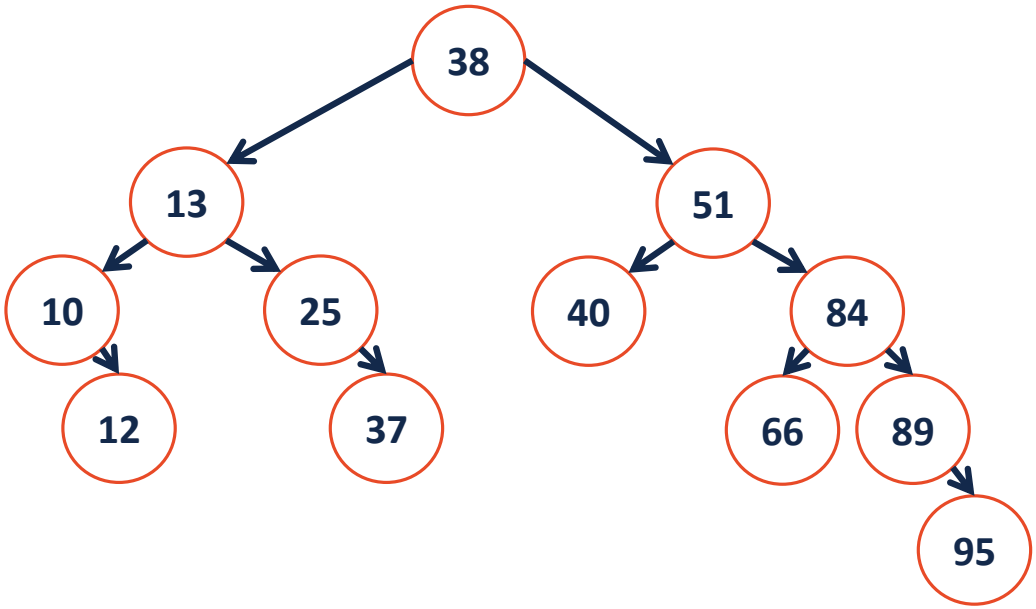

```
remove(40);
```




```
remove(25);
```




```
remove(10);
```




```
remove(13);
```