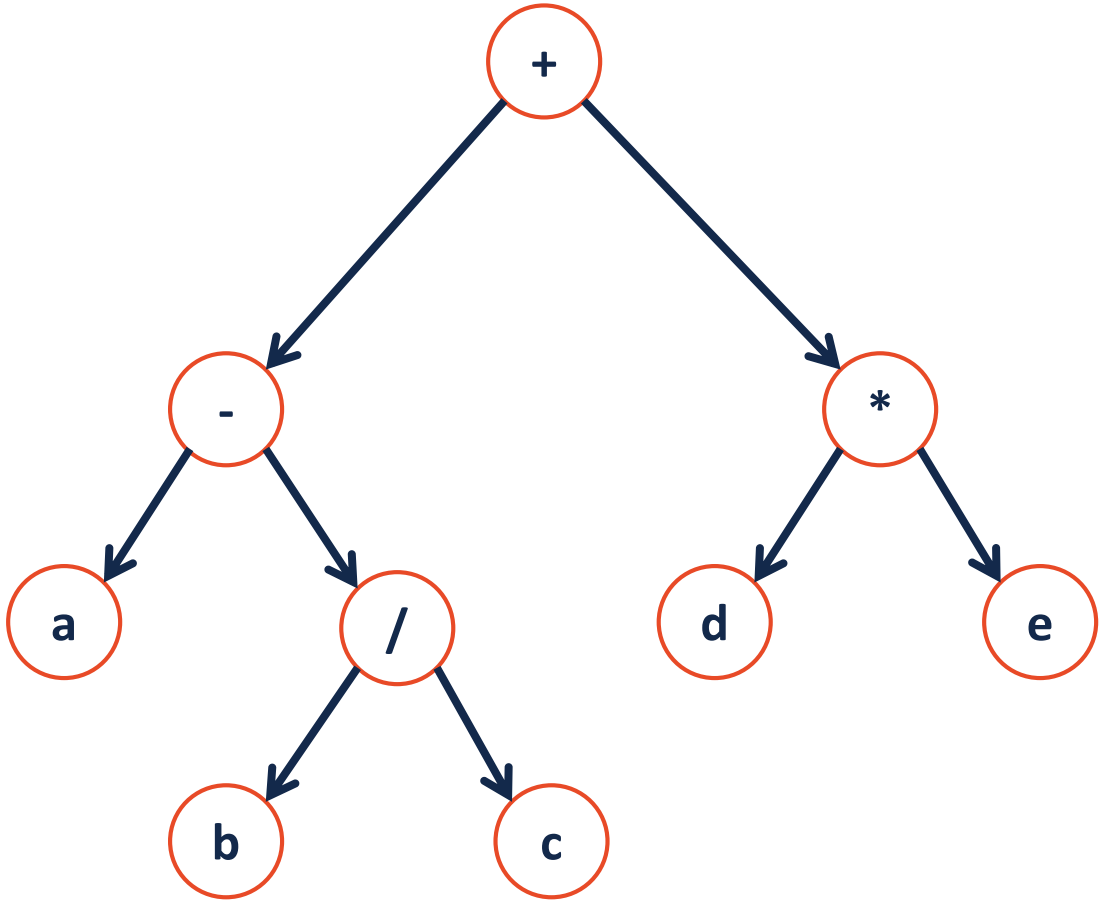
CS 225

Data Structures

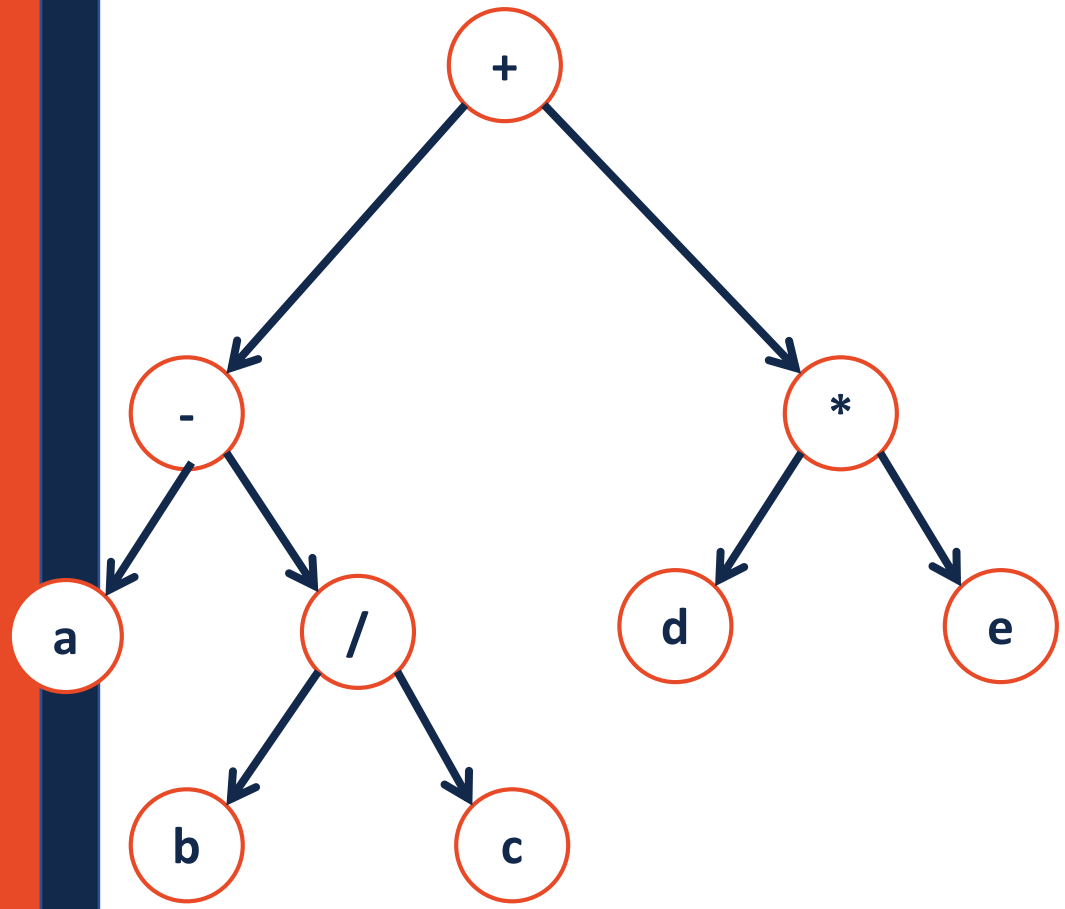
Feb. 19 – Traversal

Wade Fagen-Ulmschneider

Traversals

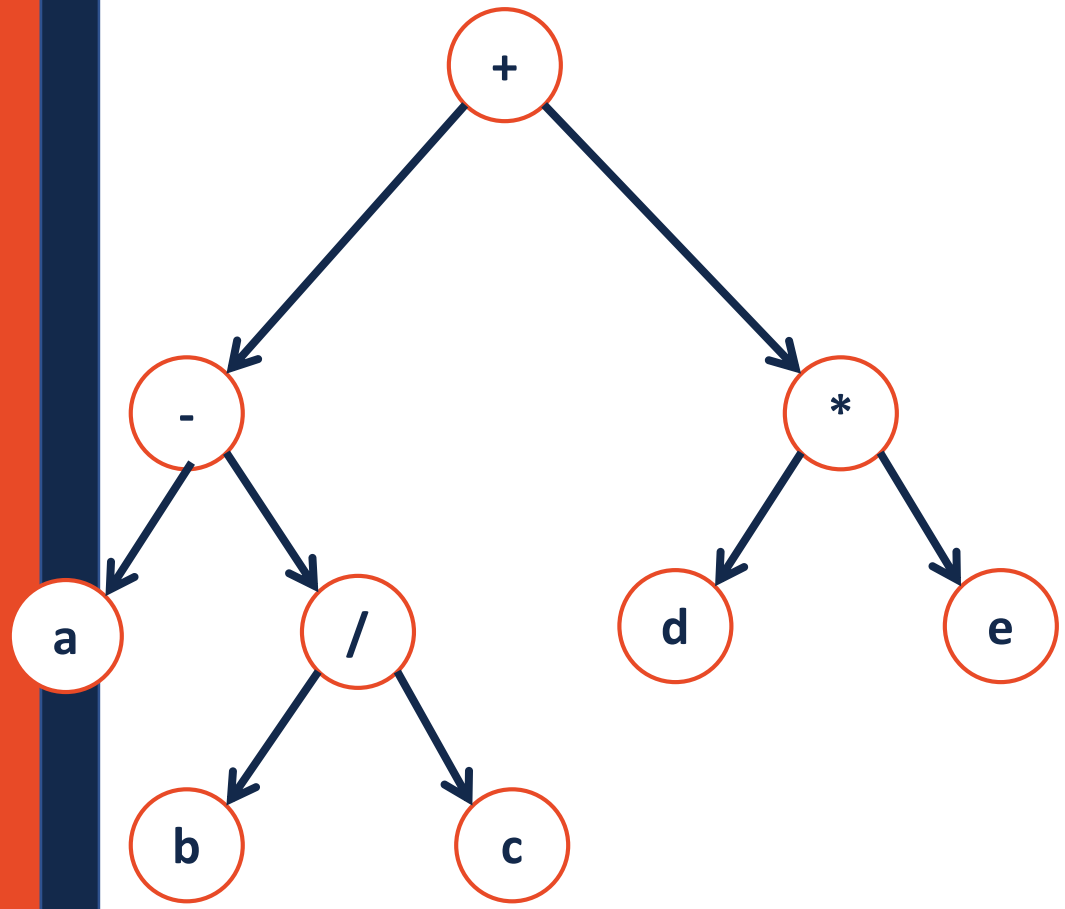


Traversals



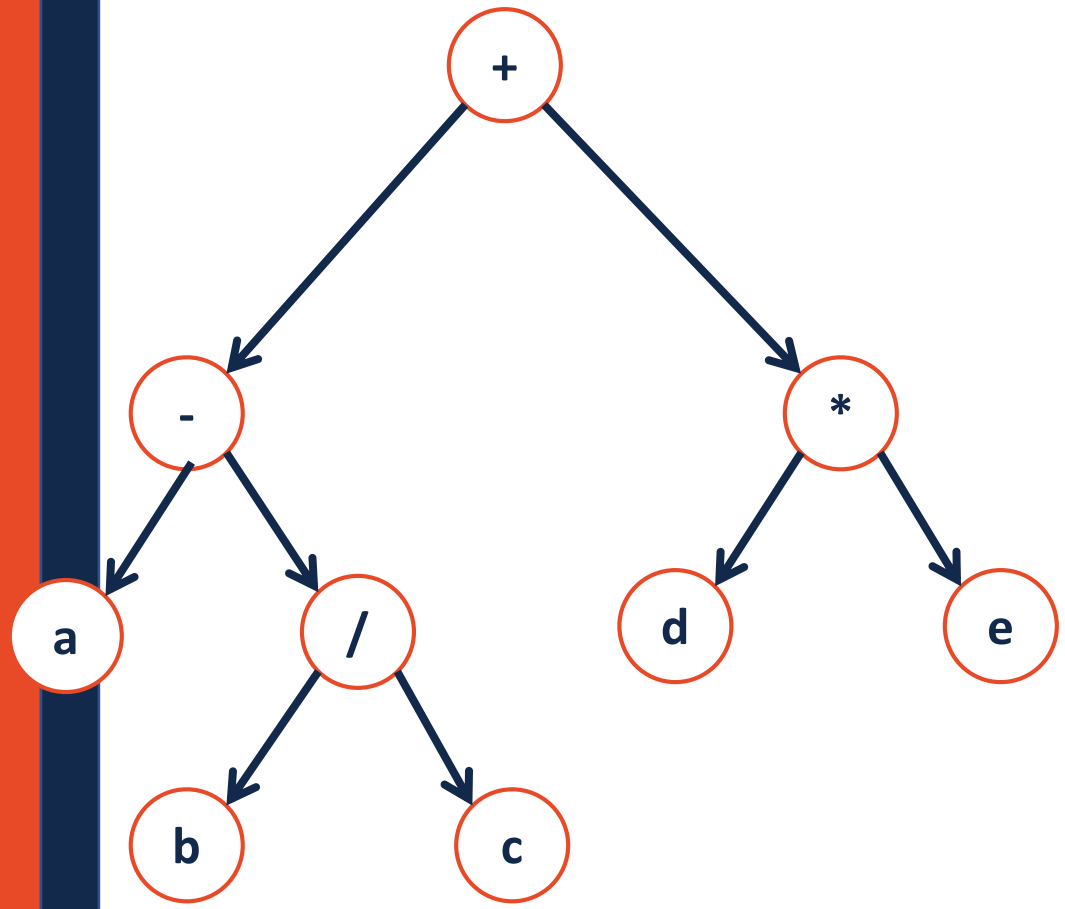

```
1  template<class T>
2  void BinaryTree<T>::__Order(TreeNode * root)
3  {
4
5
6
7
8
9
10
11
12
13
14
15
16
17 }
```

Traversals



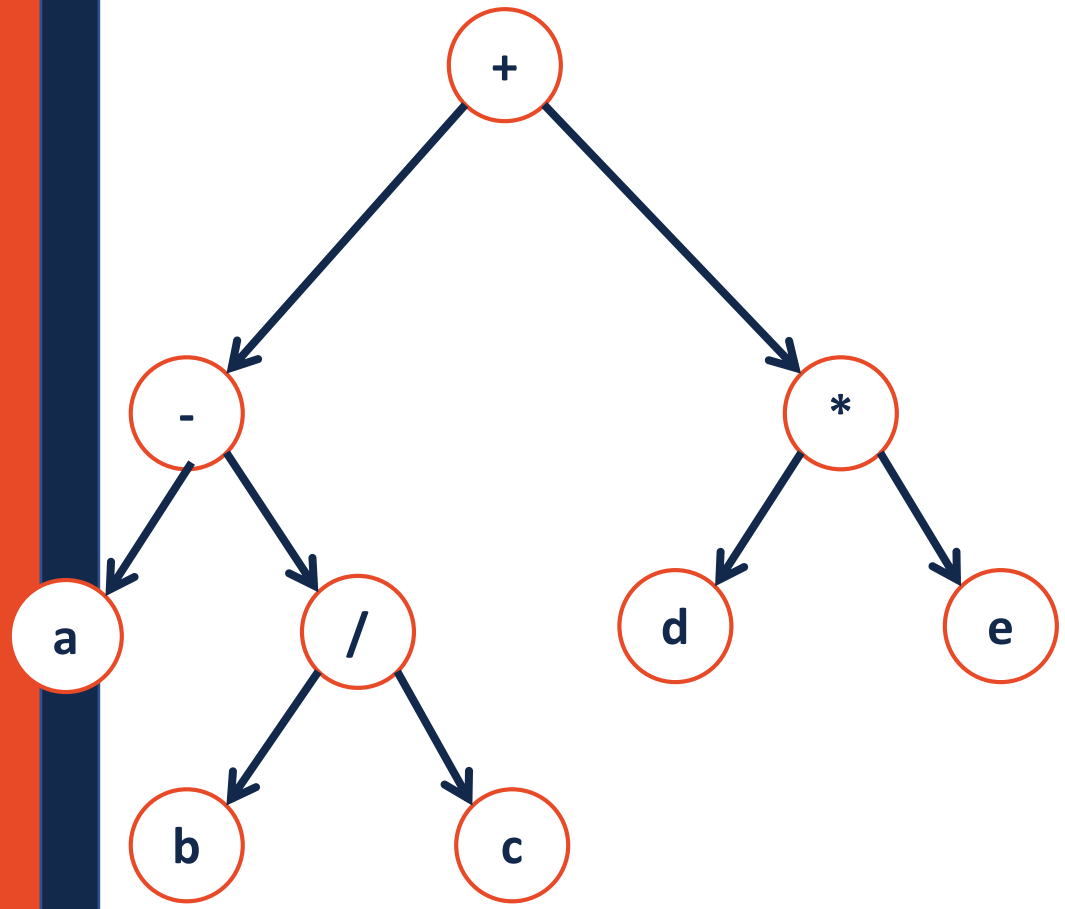

```
1  template<class T>
2  void BinaryTree<T>::__Order(TreeNode * root)
3  {
4     if (root != NULL) {
5
6        ____________________________;
7
8        ___Order(root->left);
9
10       ____________________________;
11
12       ___Order(root->right);
13
14       ____________________________;
15
16    }
17 }
```

Traversals



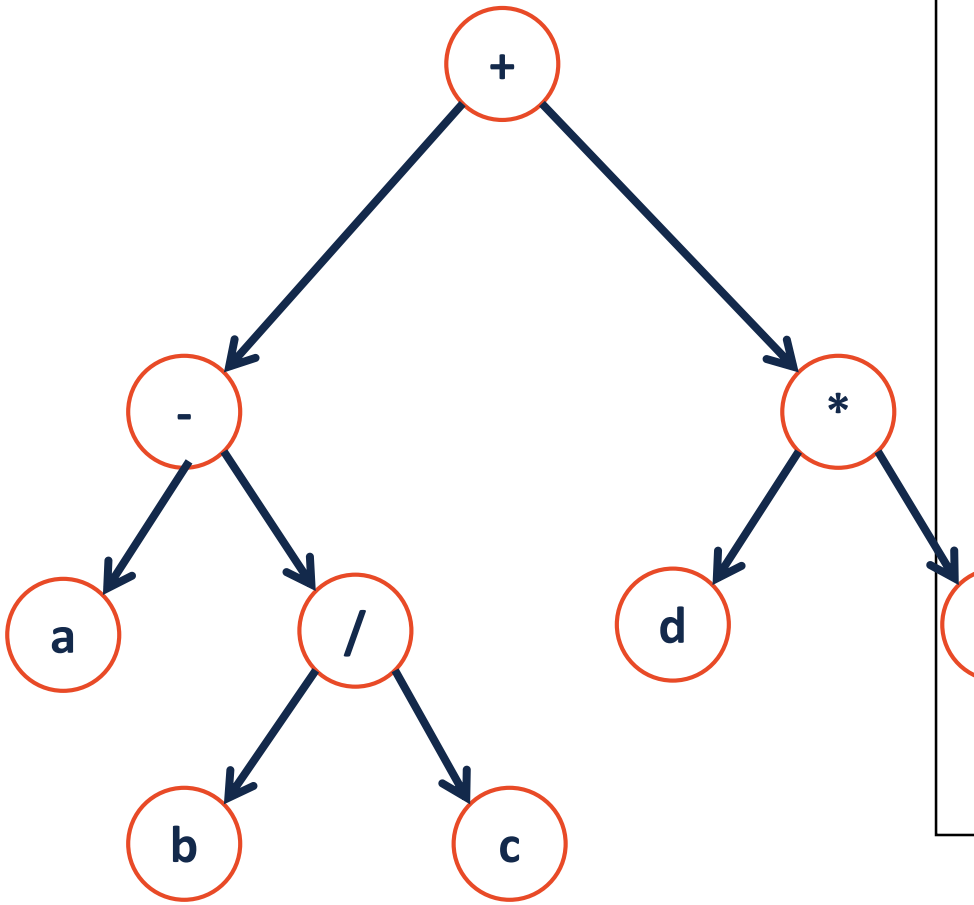

```
1  template<class T>
2  void BinaryTree<T>::__Order(TreeNode * root)
3  {
4     if (root != NULL) {
5
6        ____________________________;
7
8        ___Order(root->left);
9
10       ____________________________;
11
12       ___Order(root->right);
13
14       ____________________________;
15
16    }
17 }
```

Traversals



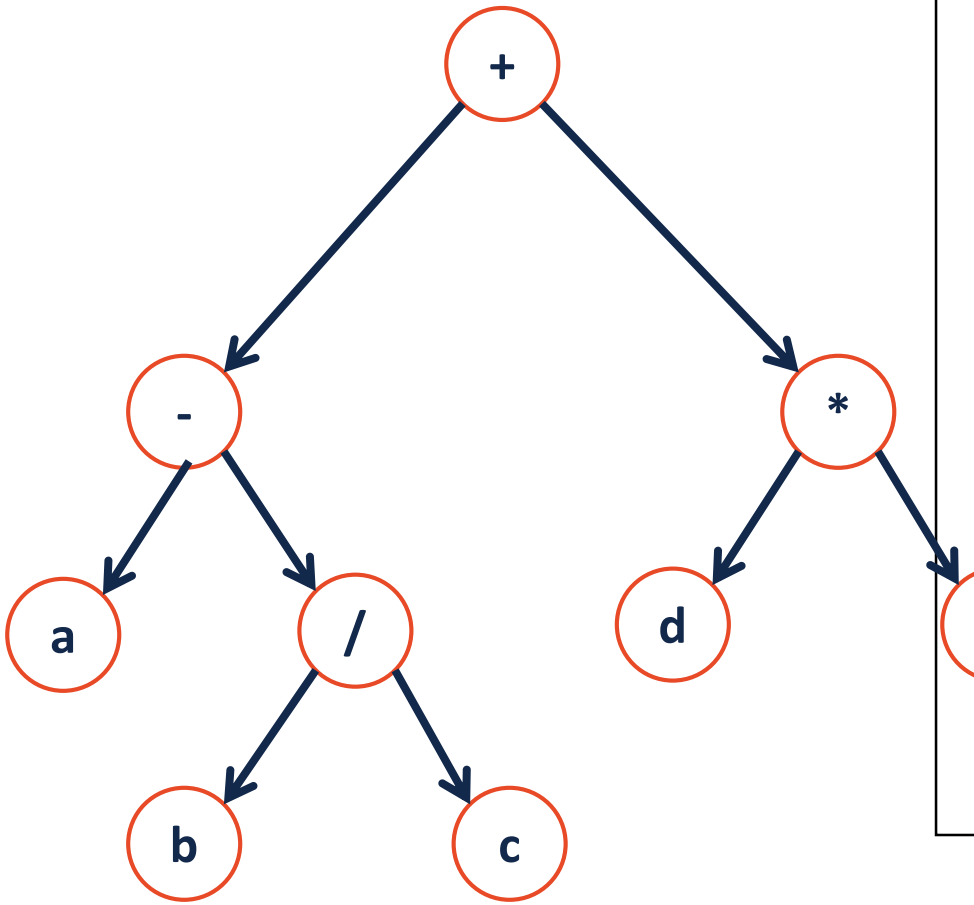

```
1  template<class T>
2  void BinaryTree<T>::__Order(TreeNode * root)
3  {
4     if (root != NULL) {
5
6        ____________________________;
7
8        ___Order(root->left);
9
10       ____________________________;
11
12       ___Order(root->right);
13
14       ____________________________;
15
16    }
17 }
```

A Broader View



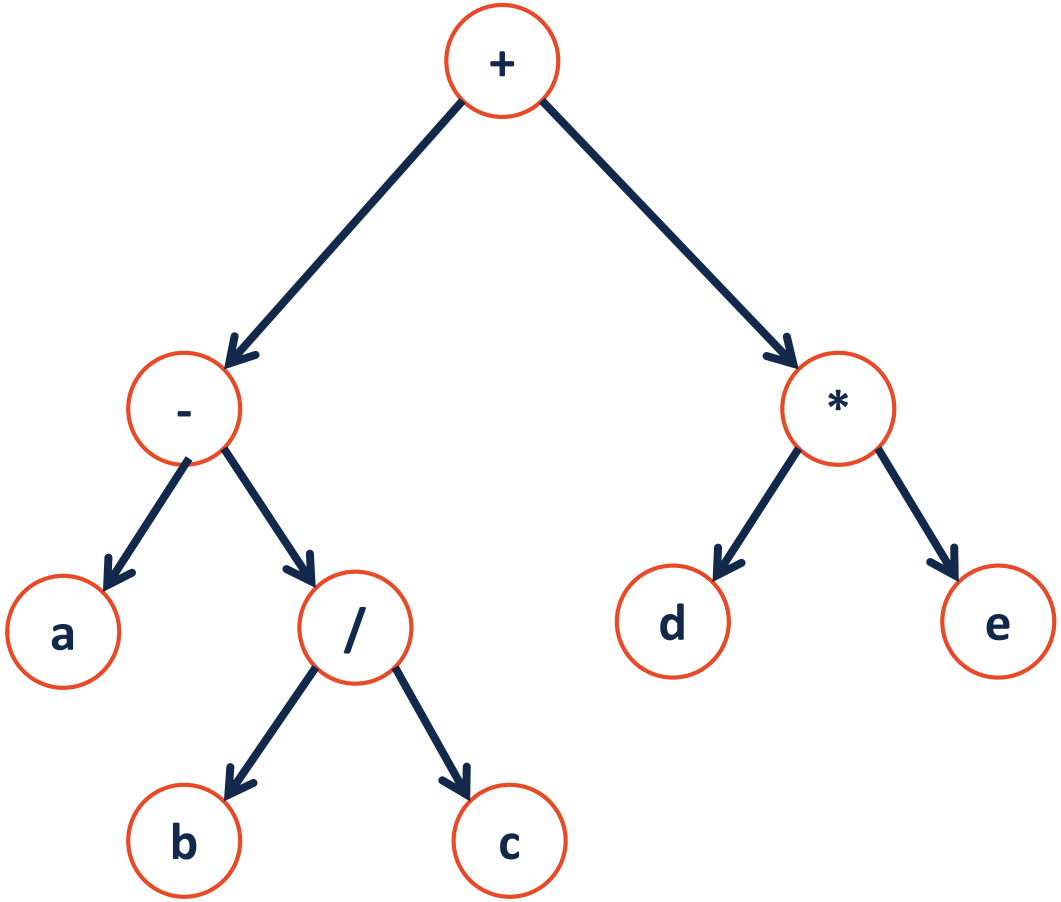

```
1  template<class T>
2  TreeNode * BinaryTree<T>::_copy
   (TreeNode * root) {
3     if (root != NULL) {
4
5
6
7
8
9
10
11
14
15    }
16 }
```

A Broader View

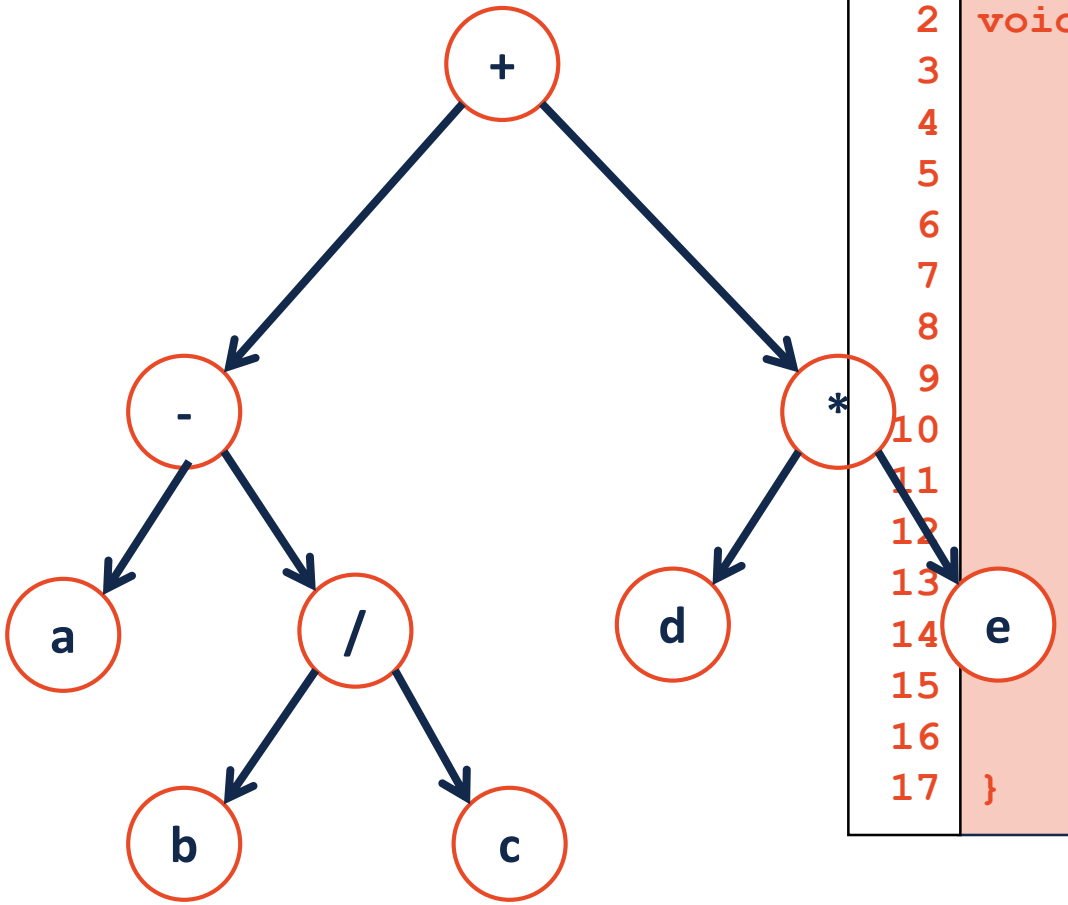



```
1  template<class T>
2  void BinaryTree<T>::_clear(TreeNode * root) {
3     if (root != NULL) {
4
5
6
7
8
9
10
11
12
15
16    }
17 }
```

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

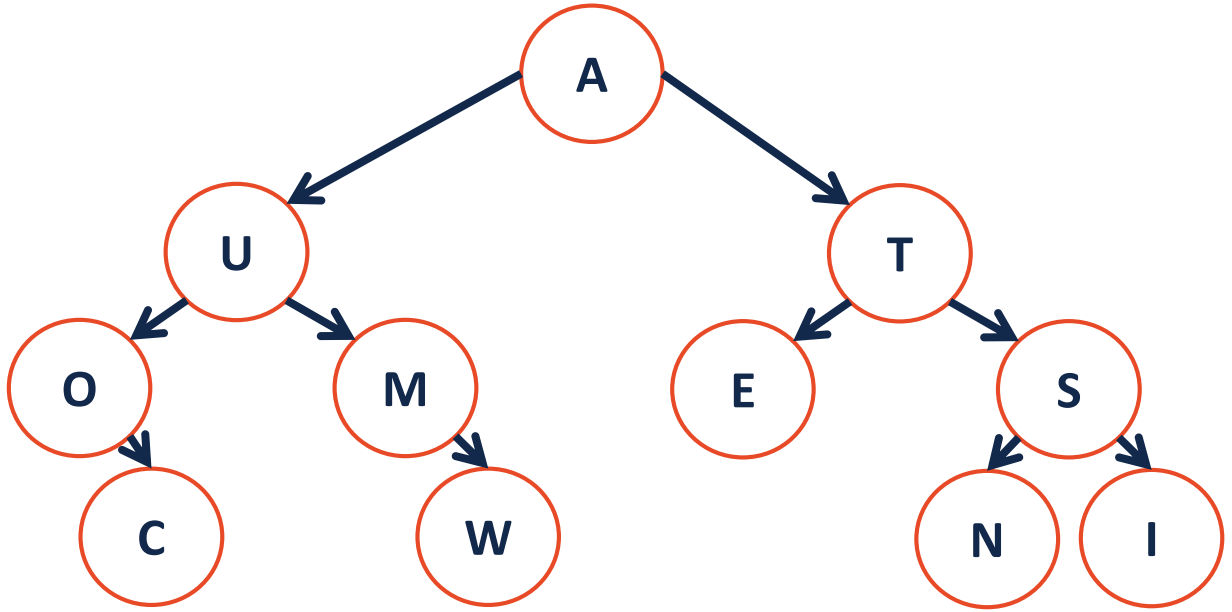
Data is often organized into key/value pairs:

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

Dictionary.h

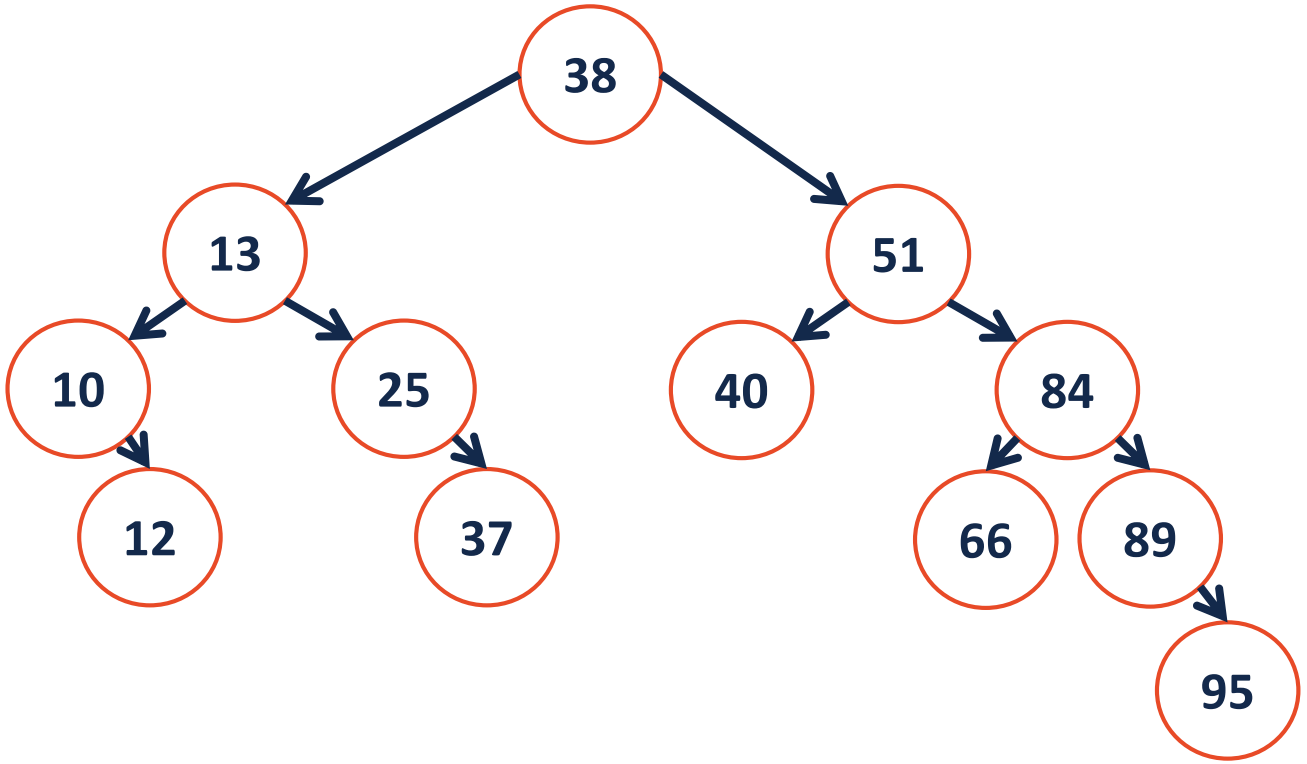
```
 1 #ifndef DICTIONARY_H
 2 #define DICTIONARY_H
 3
 4
 5 class Dictionary {
 6   public:
 7
 8
 9
10
11
12
13
14
15
16
17
18
19   private:
20 };
21
22 #endif
```

Binary Tree as a Search Structure



Binary _____ Tree (BST)

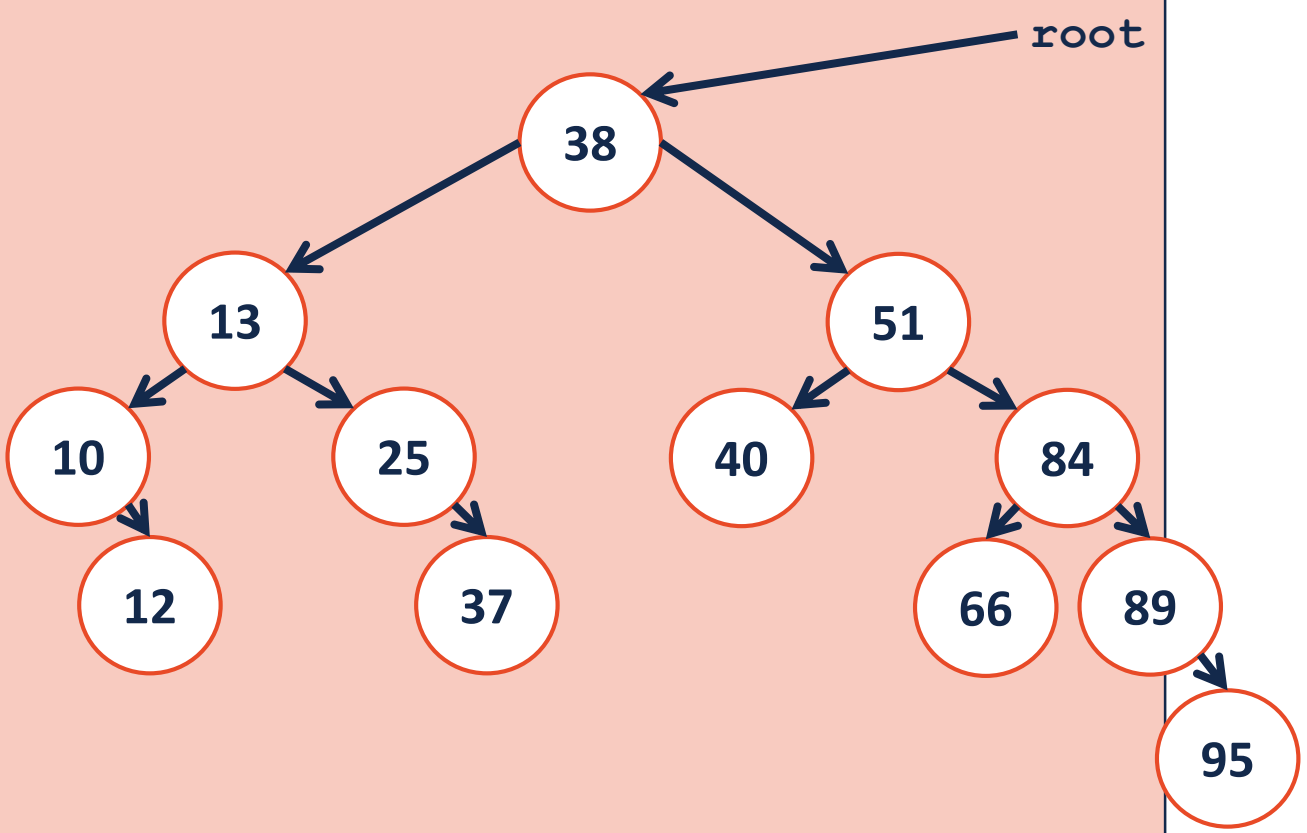
A **BST** is a binary tree **T** such that:

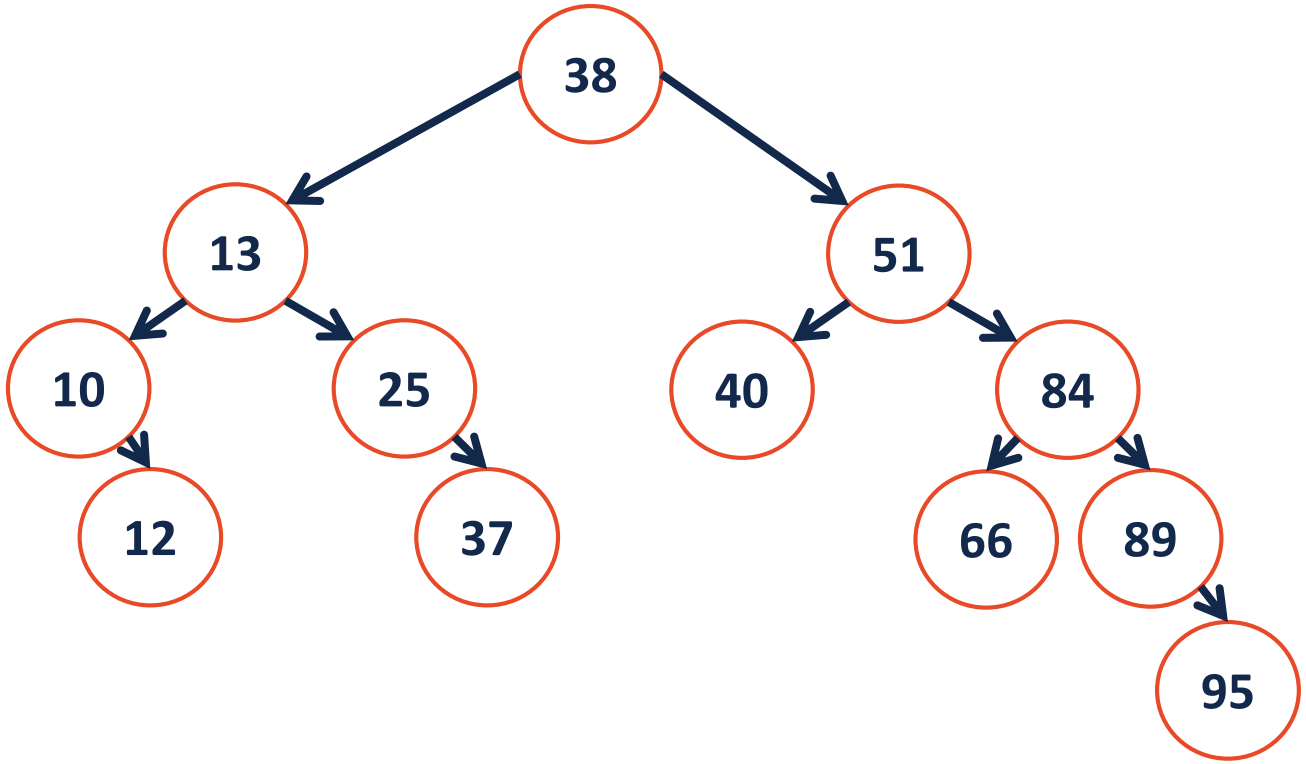


BST.h

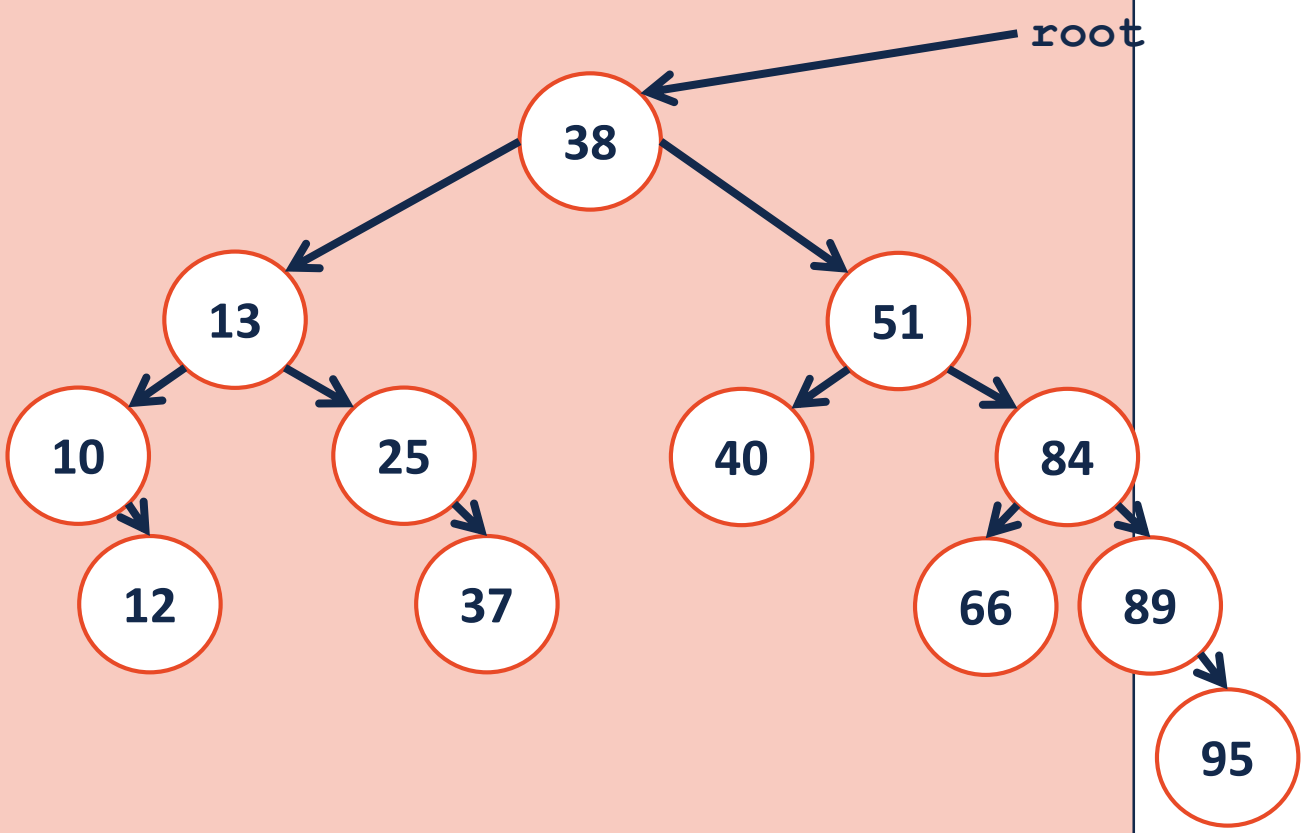
```
 1 #ifndef DICTIONARY_H
 2 #define DICTIONARY_H
 3
 4 template <class K, class V>
 5 class BST {
 6   public:
 7     BST();
 8     void insert(const K key, V value);
 9     V remove(const K & key);
10     V find(const K & key) const;
11     TreeIterator traverse() const;
12   private:
13
14
15
16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

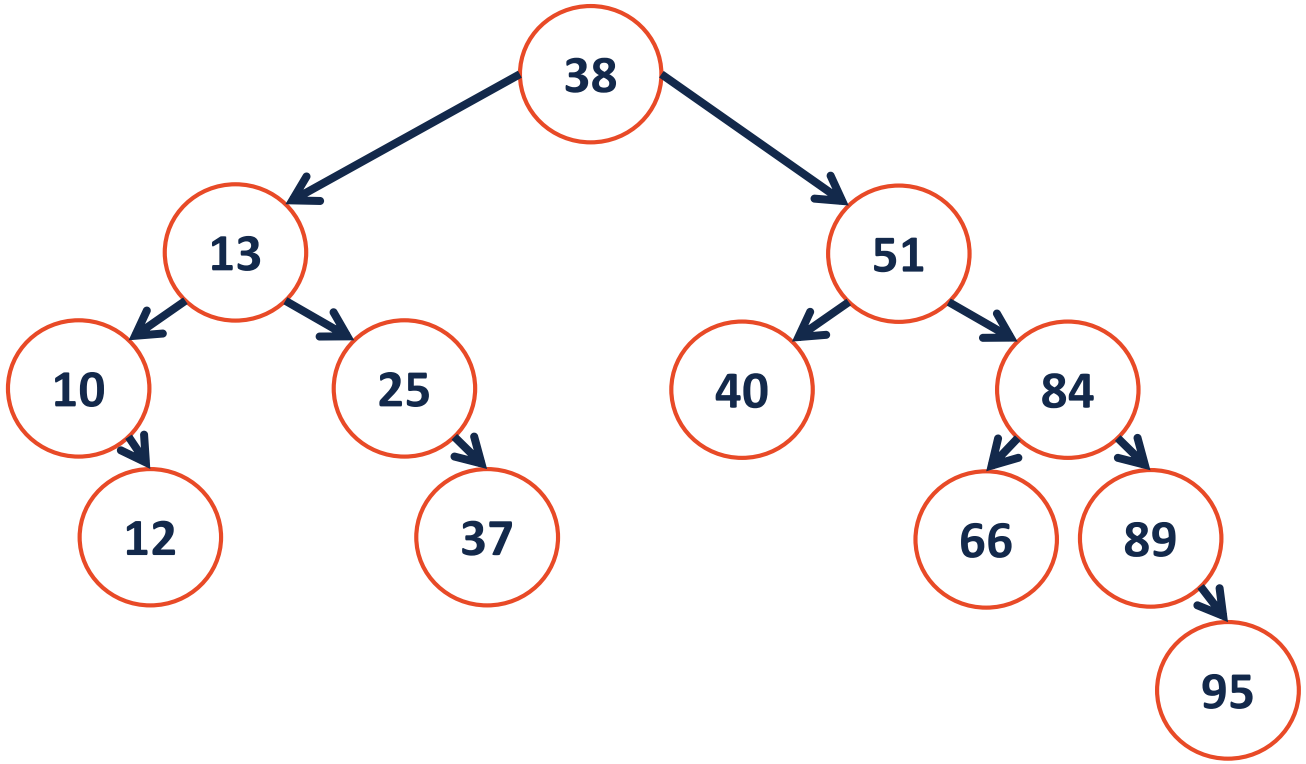
```
1  template<class K, class V>
2  _____________________________ _find(TreeNode *& root, const K & key) const {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 }
```



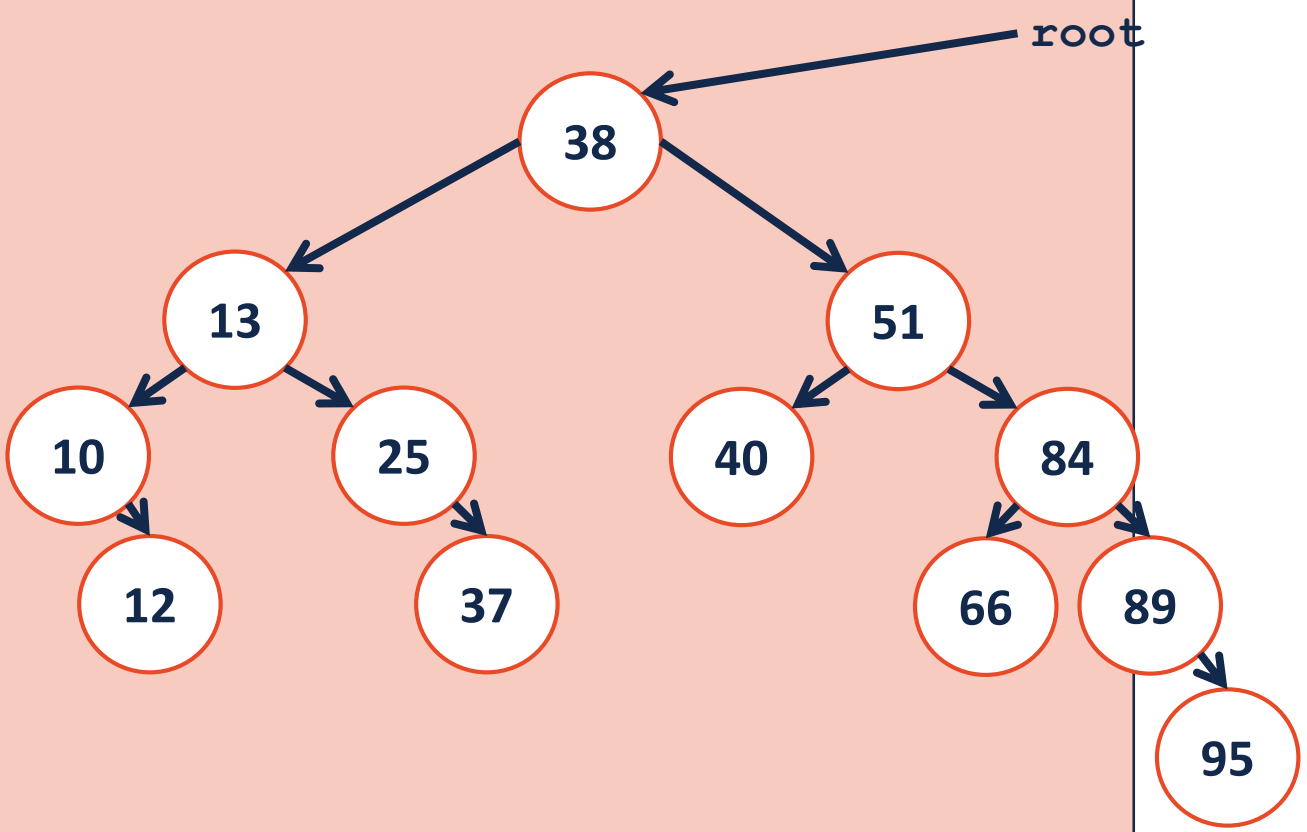

38
13
51
10
25
40
84
12
37
66
89
95

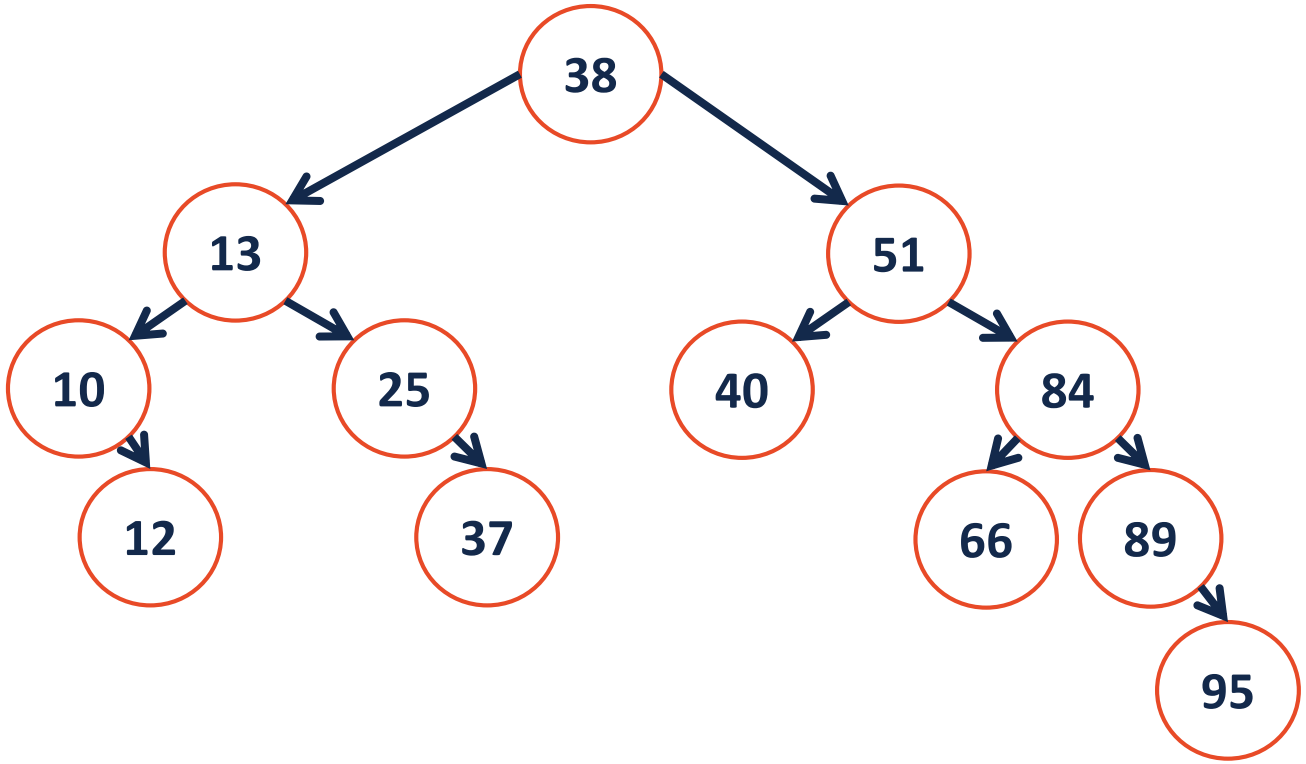
```
1  template<class K, class V>
2  _____________________________ _insert(TreeNode *& root, const K & key) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 }
```



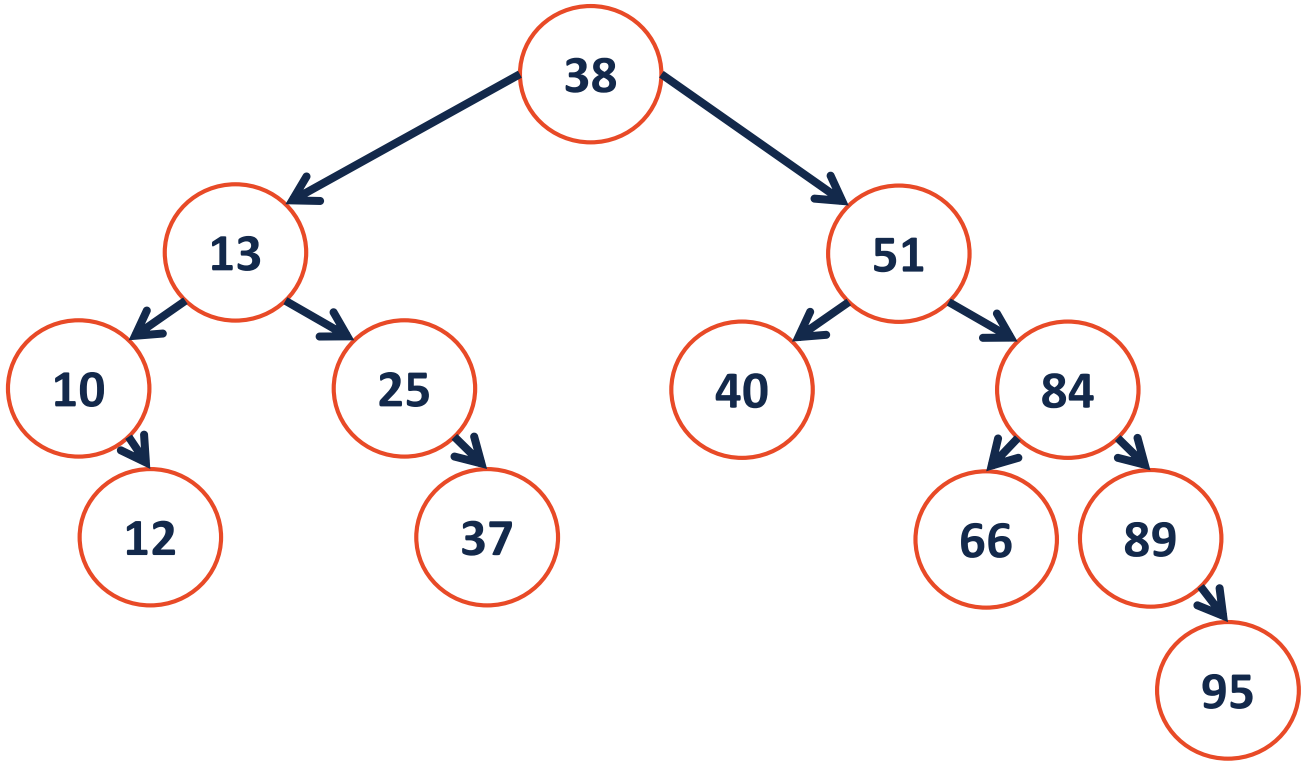

38
13
51
10
25
40
84
12
37
66
89
95

```
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 }
```

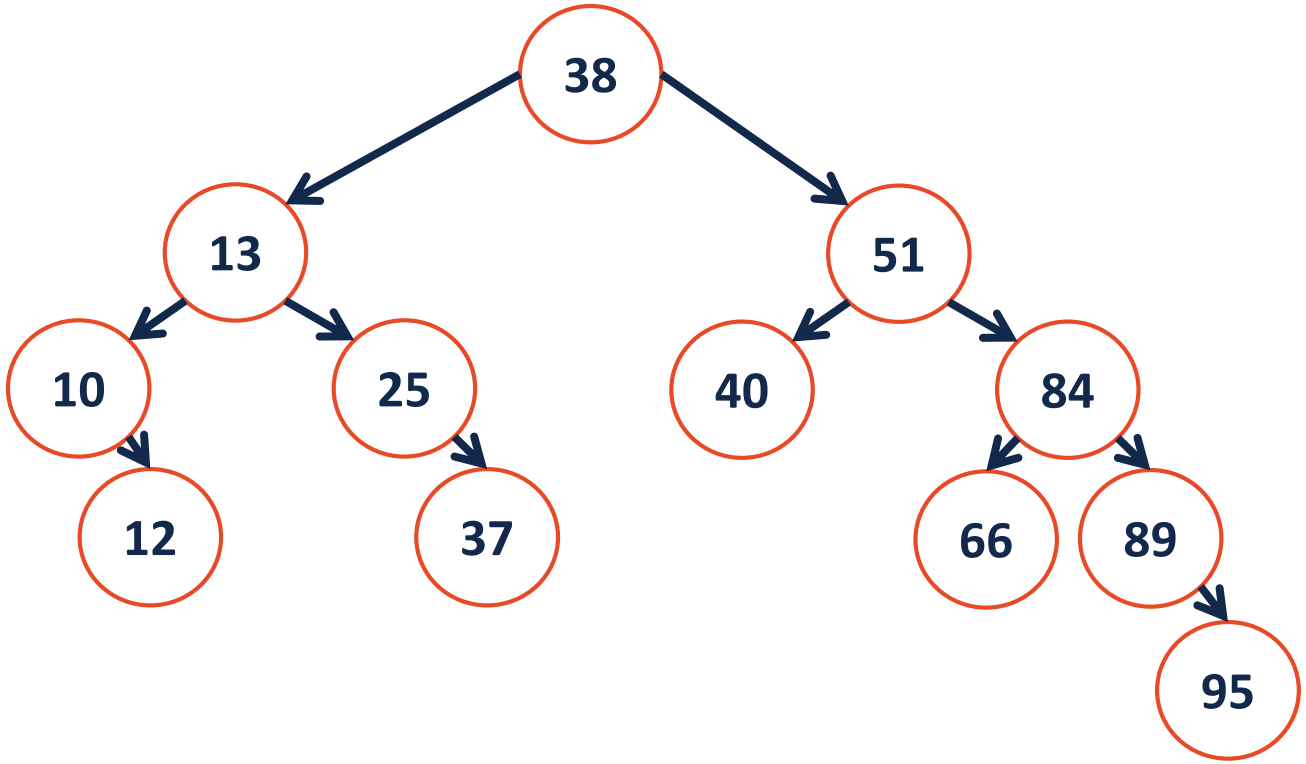




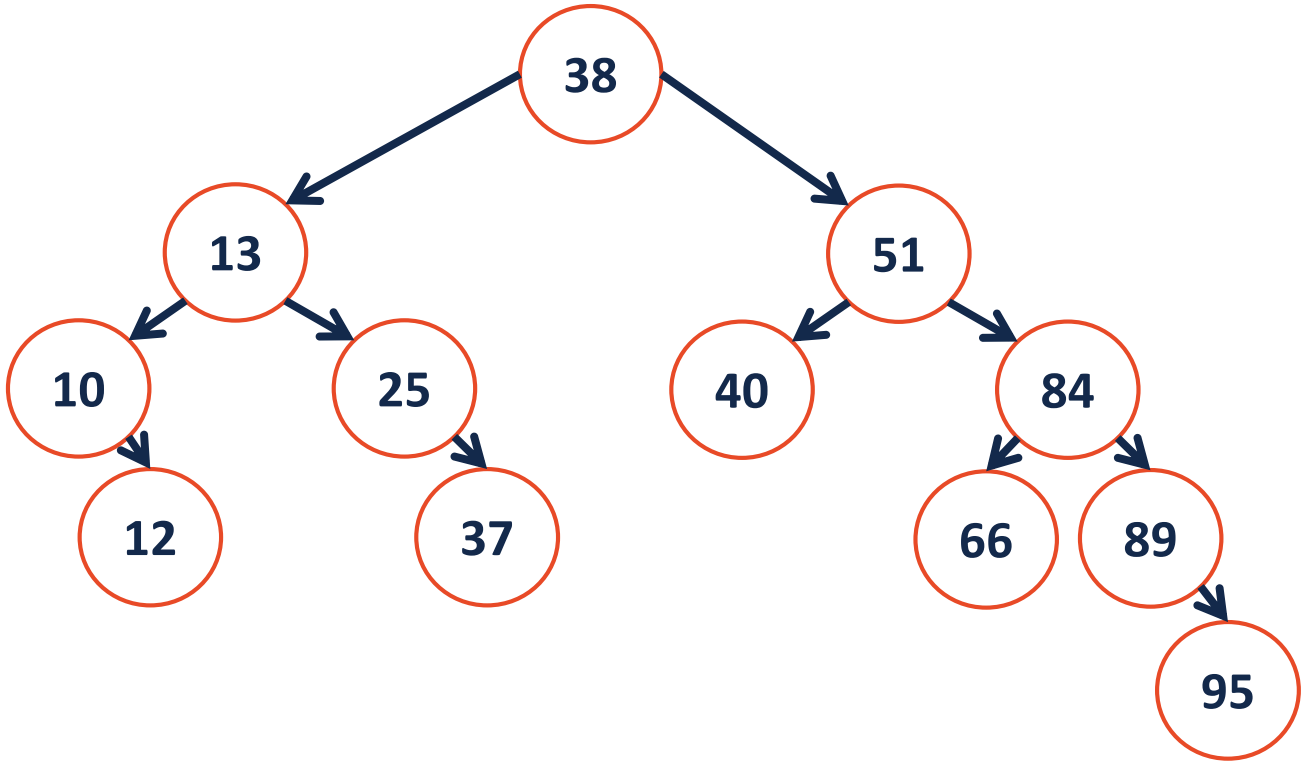

```
remove(40);
```




```
remove(25);
```




```
remove(10);
```




```
remove(13);
```