



CS 225

Data Structures

February 18 – Traversals

G Carl Evans

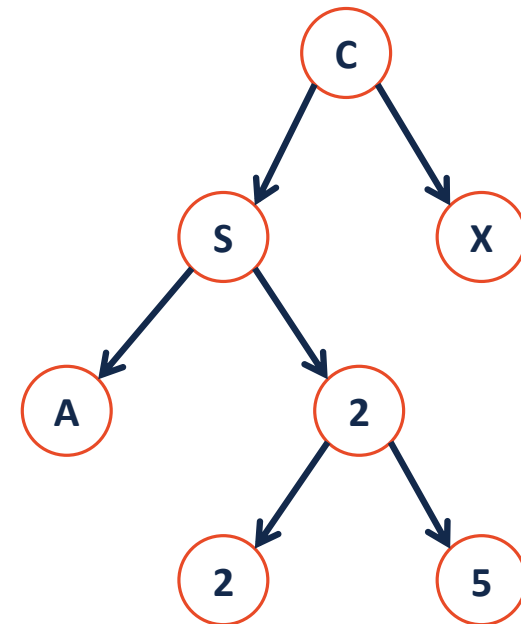
Binary Tree – Defined

A binary tree T is either:

-

OR

-

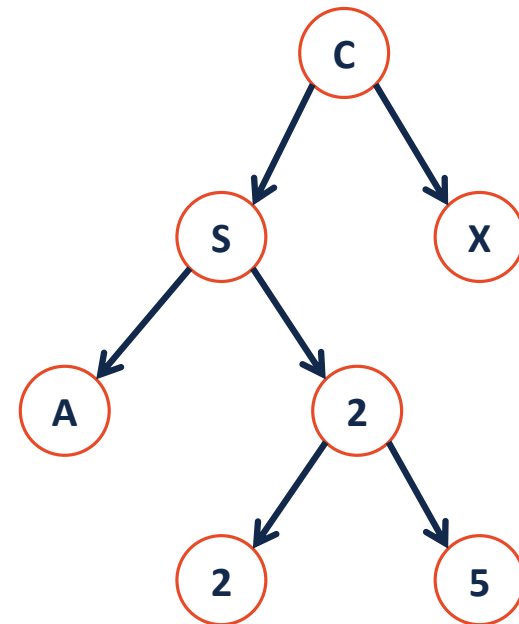


Tree Property: height

height(T): length of the longest path from the root to a leaf

Given a binary tree T:

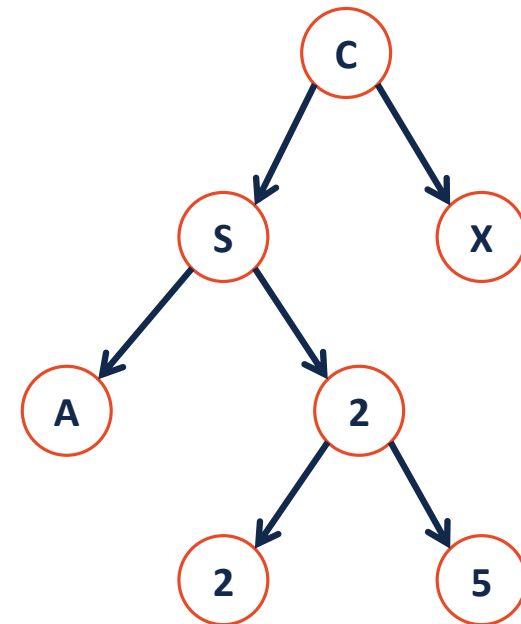
height(T) =



Tree Property: full

A tree F is **full** if and only if:

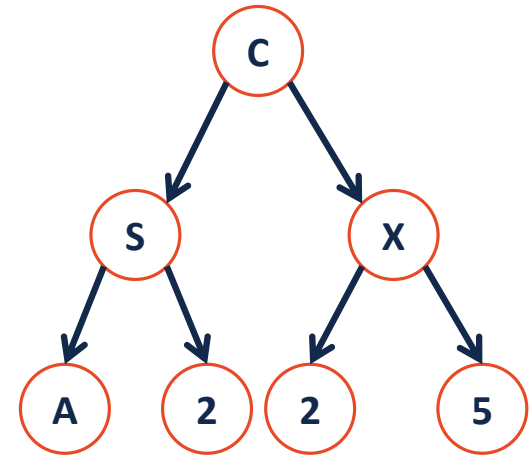
- 1.
- 2.



Tree Property: perfect

A **perfect** tree P is:

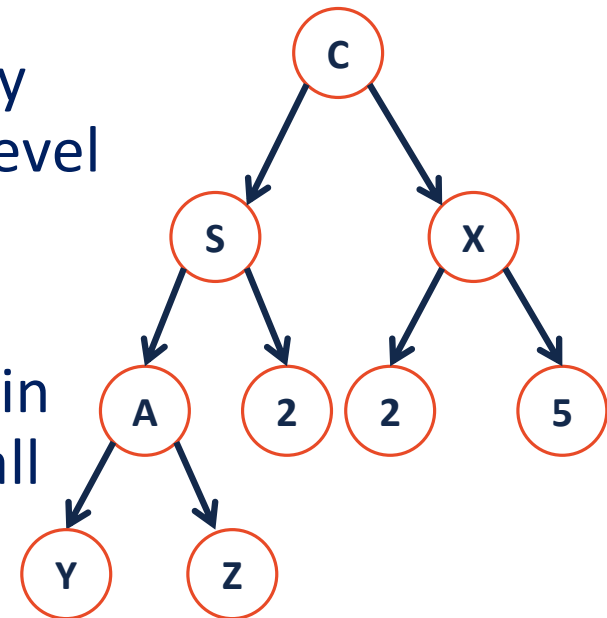
- 1.
- 2.



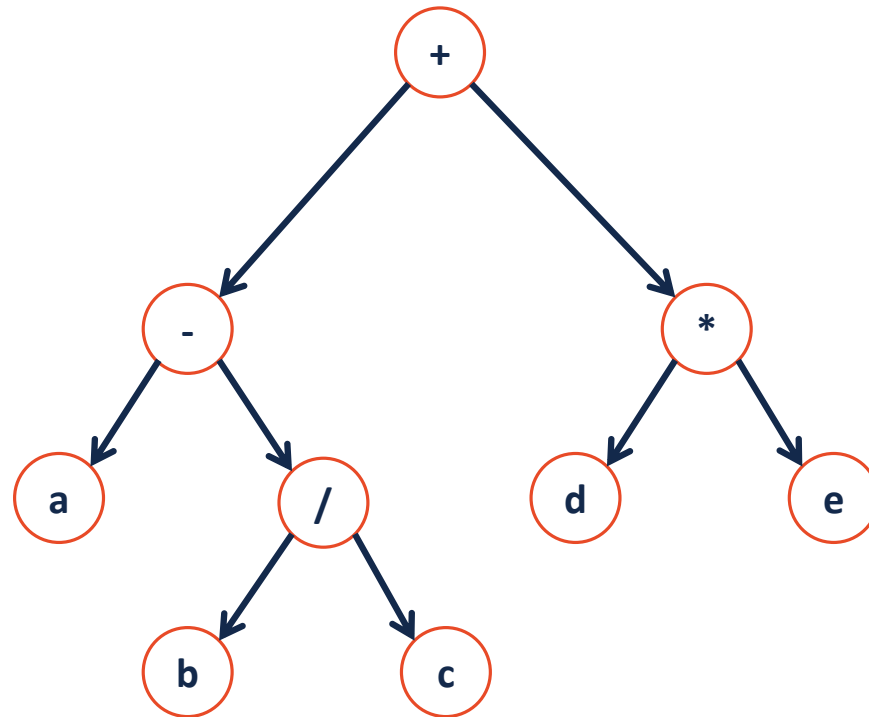
Tree Property: complete

Conceptually: A perfect tree for every level except the last, where the last level is “pushed to the left”.

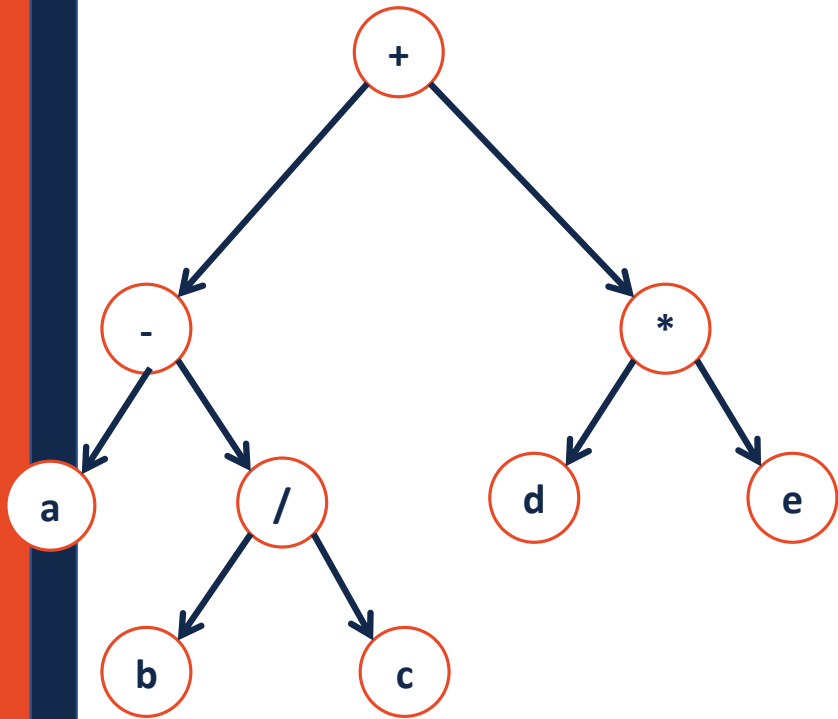
Slightly more formal: For any level k in $[0, h-1]$, k has 2^k nodes. For level h , all nodes are “pushed to the left”.



Traversals

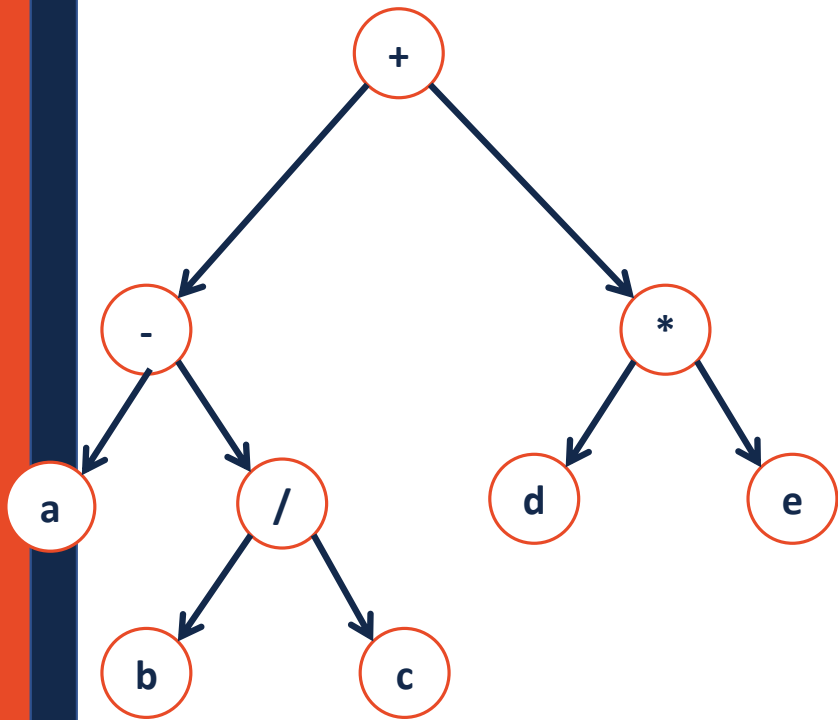


Traversals



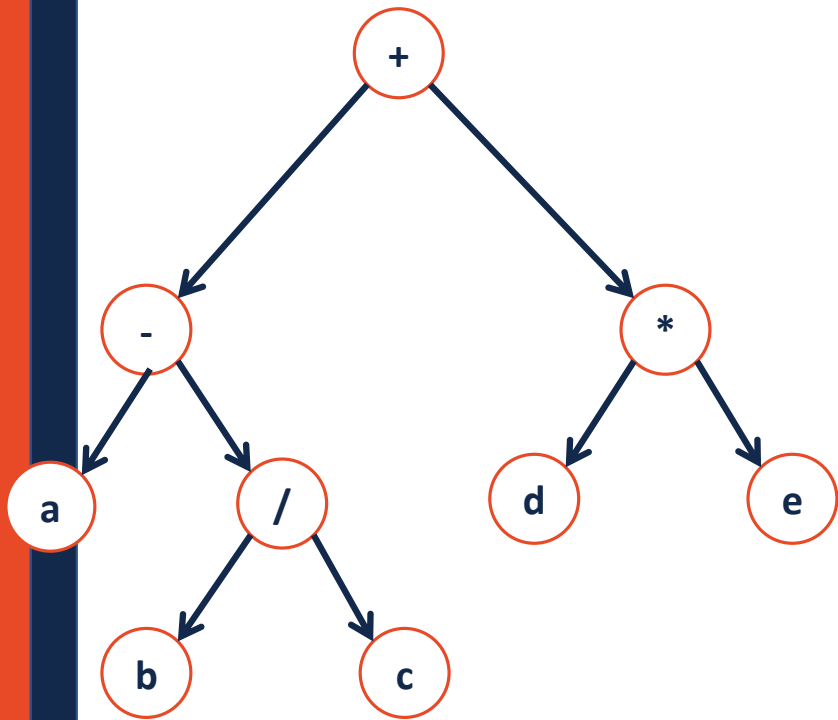
```
49 template<class T>
50 void BinaryTree<T>::__Order(TreeNode * cur)
51 {
52
53
54
55
56
57
58 }
```


Traversals



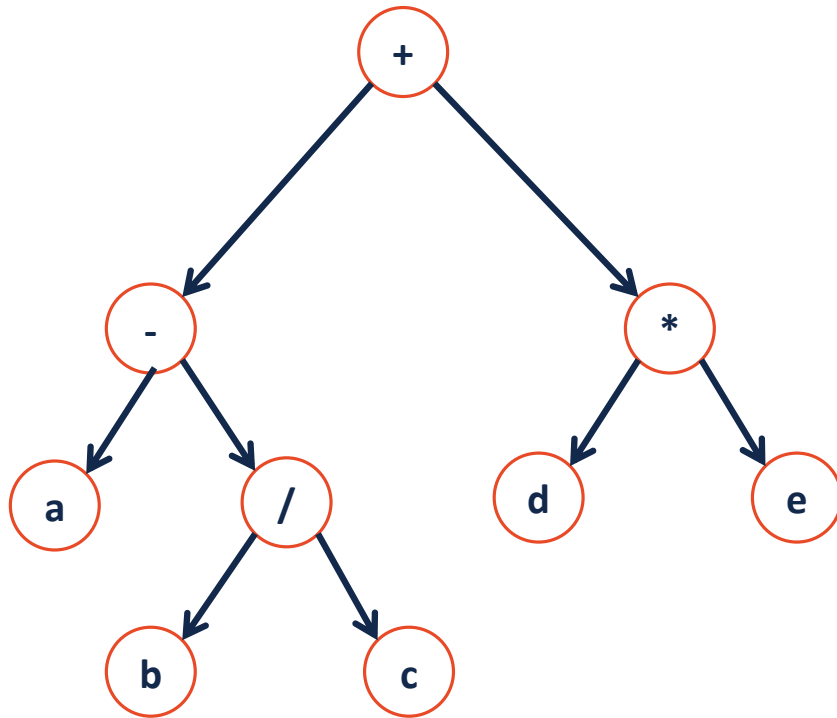
```
49 template<class T>
50 void BinaryTree<T>::__Order(TreeNode * cur) {
51     if (cur != NULL) {
52         _____;
53         __Order(cur->left);
54         _____;
55         __Order(cur->right);
56         _____;
57     }
58 }
```

Traversals

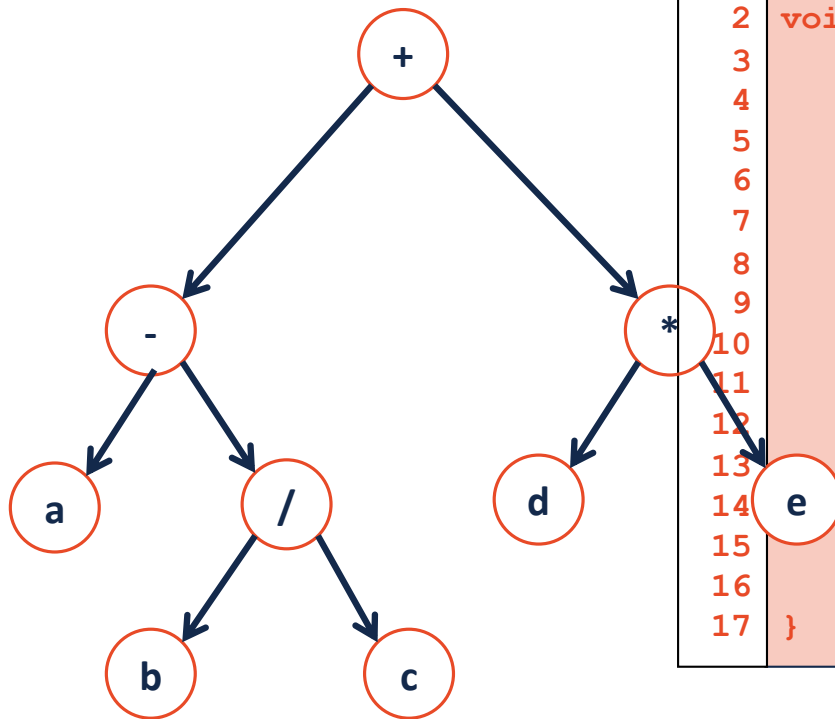


```
49 template<class T>
50 void BinaryTree<T>::__Order(TreeNode * cur) {
51     if (cur != NULL) {
52         _____;
53         __Order(cur->left);
54         _____;
55         __Order(cur->right);
56         _____;
57     }
58 }
```

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  e
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

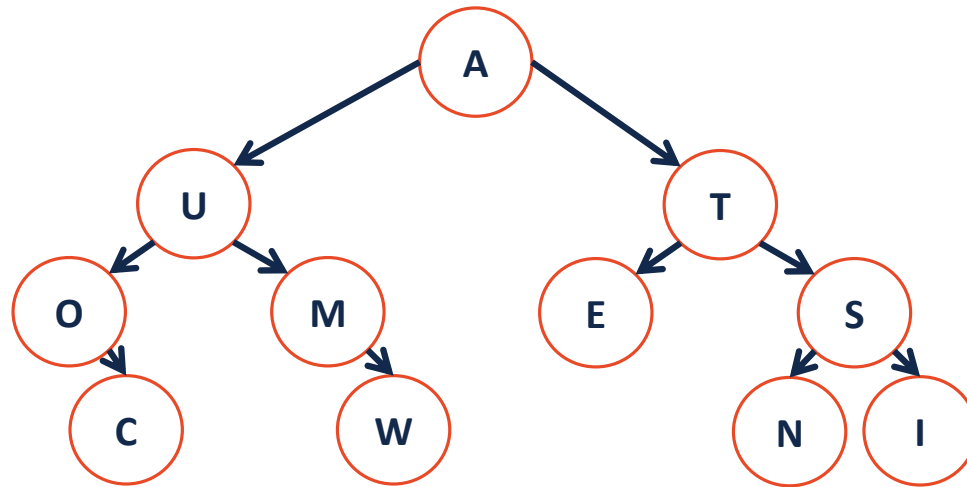
URL → HTML Page

...

Dictionary.h

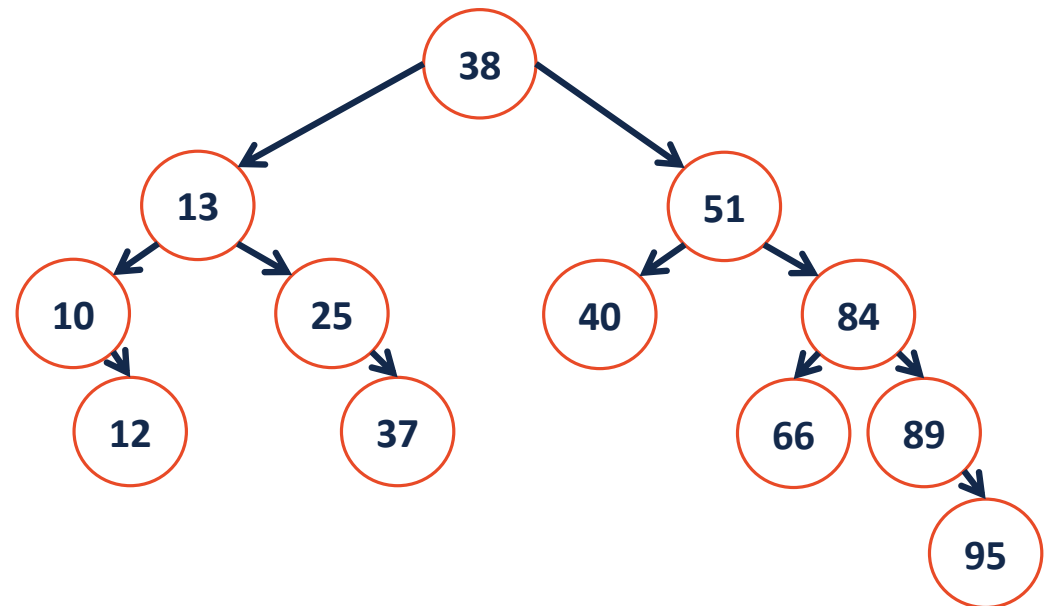
```
1 #pragma once
2
3
4 class Dictionary {
5     public:
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20     private:
21         // ...
22 };
```


Binary Tree as a Search Structure



Binary _____ Tree (BST)

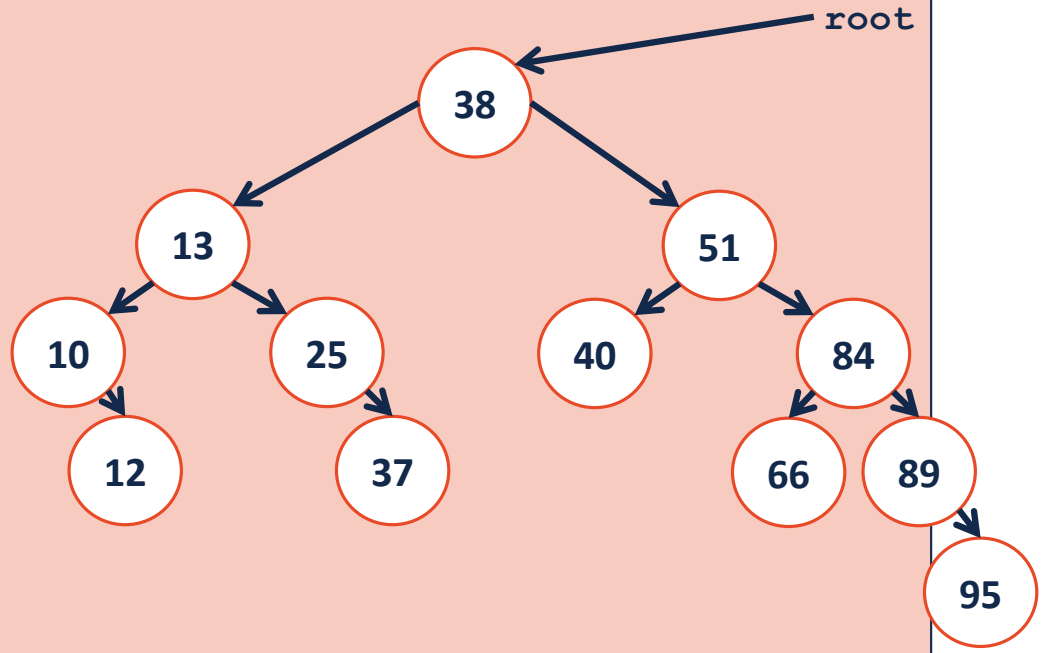
A **BST** is a binary tree **T** such that:

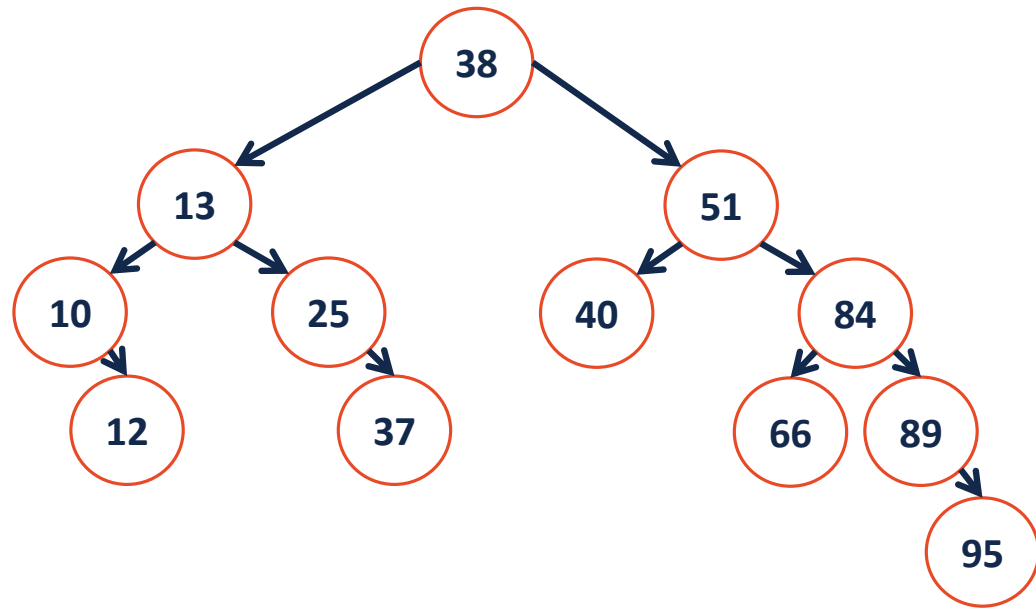


BST.h

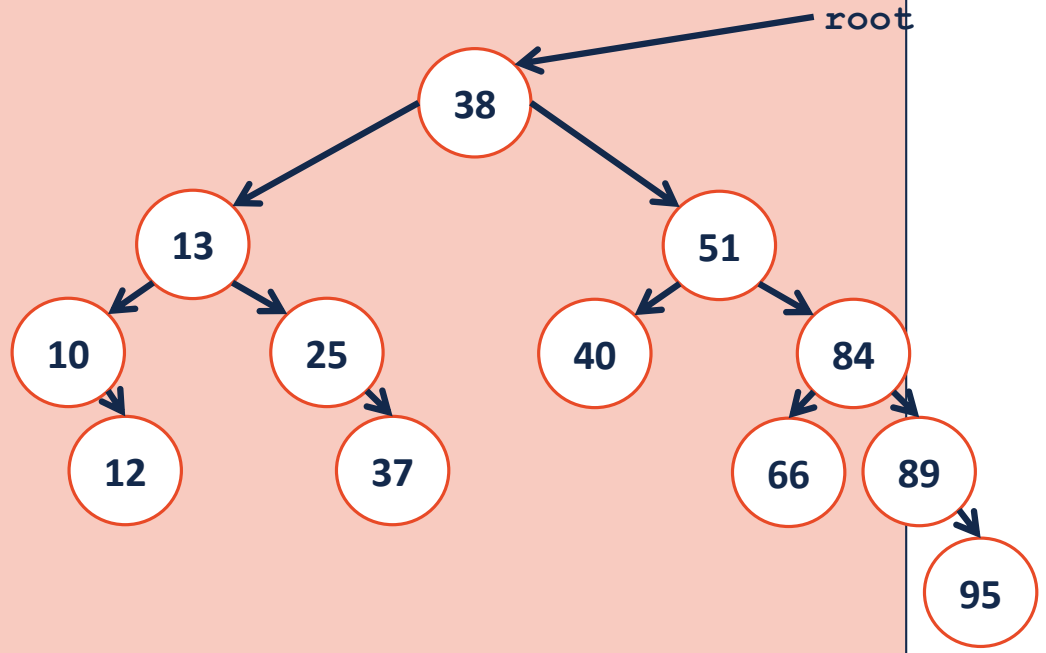
```
1 #pragma once
2
3 template <class K, class V>
4 class BST {
5     public:
6         BST();
7         void insert(const K key, V value);
8         V remove(const K & key);
9         V find(const K & key) const;
10        TreeIterator traverse() const;
11
12    private:
13
14
15
16
17
18
19
20
21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

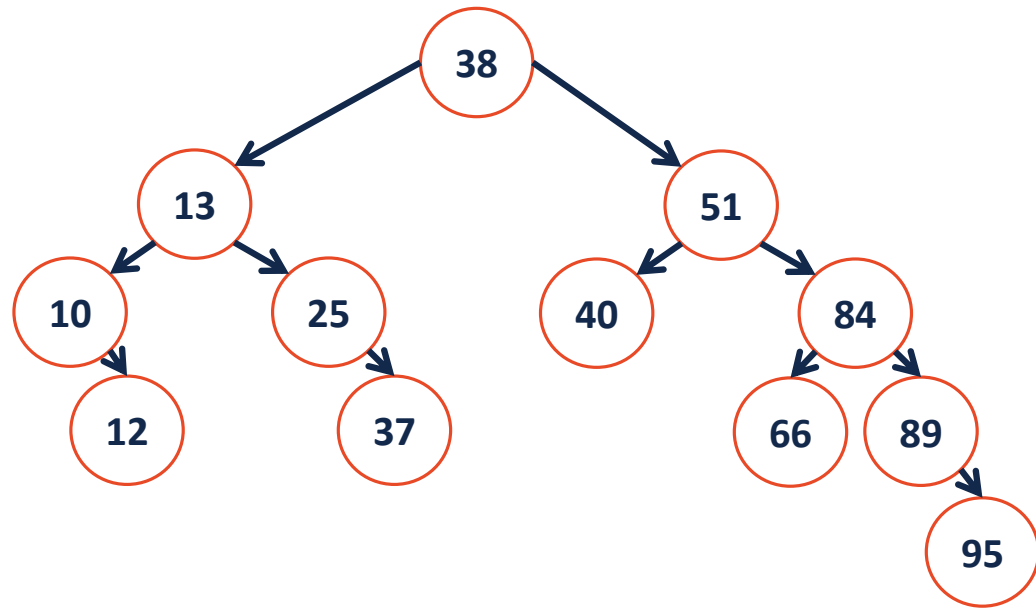
```
1  template<class K, class V>
2  _____ _find(TreeNode *& root, const K & key) const {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 }
```



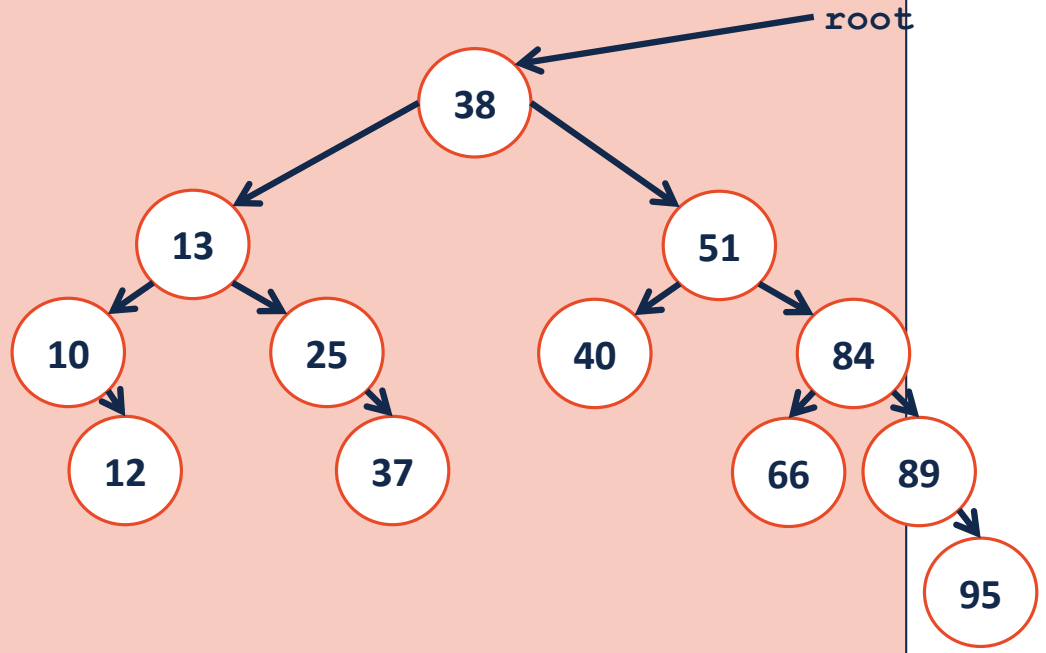


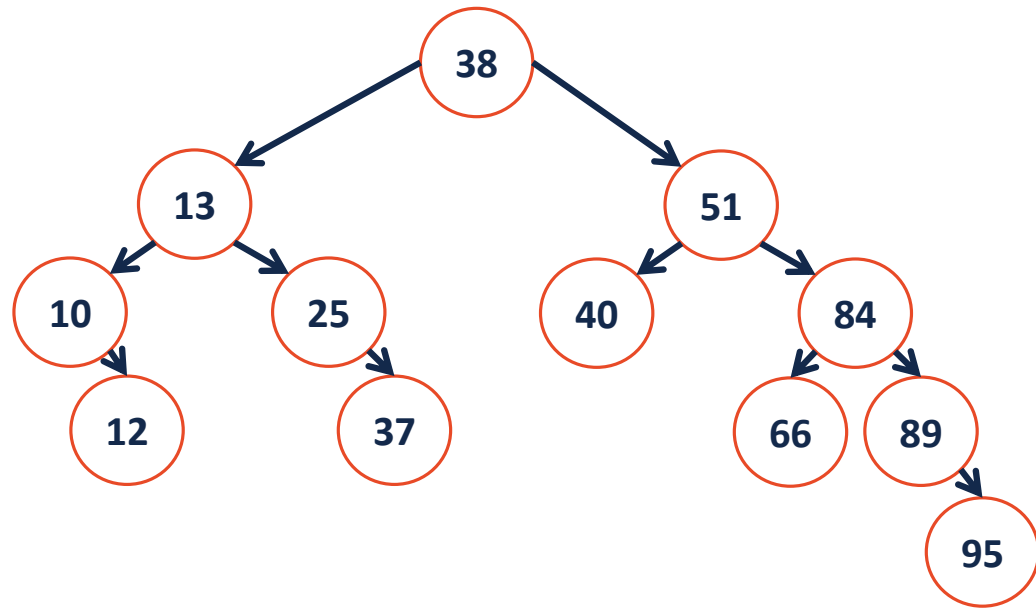
```
1 template<class K, class V>
2 _____ _insert(TreeNode *& root, const K & key) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 }
```



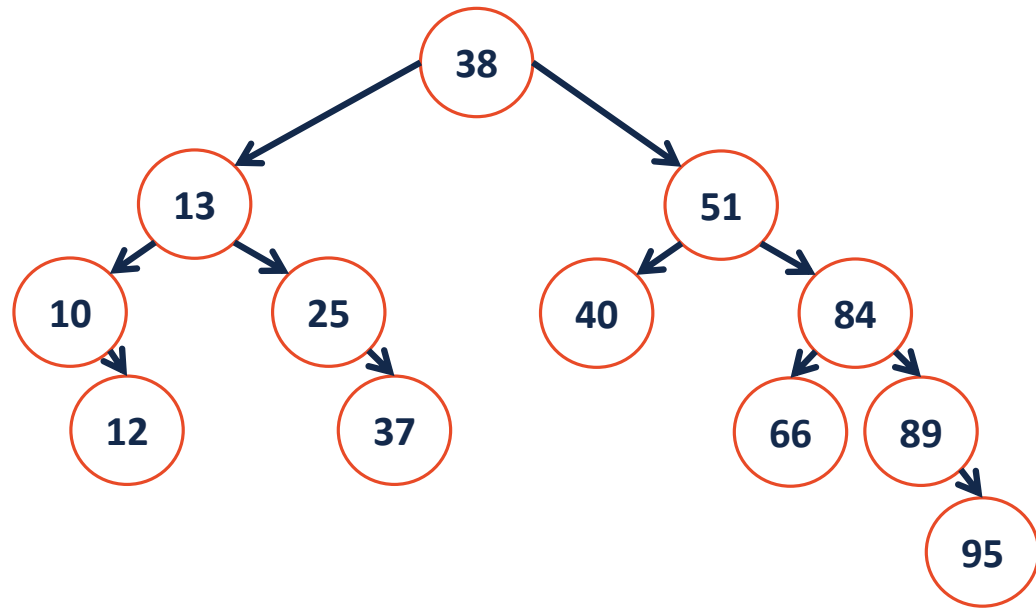


```
1  template<class K, class V>
2  _____ _remove(TreeNode *& root, const K & key) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 }
```

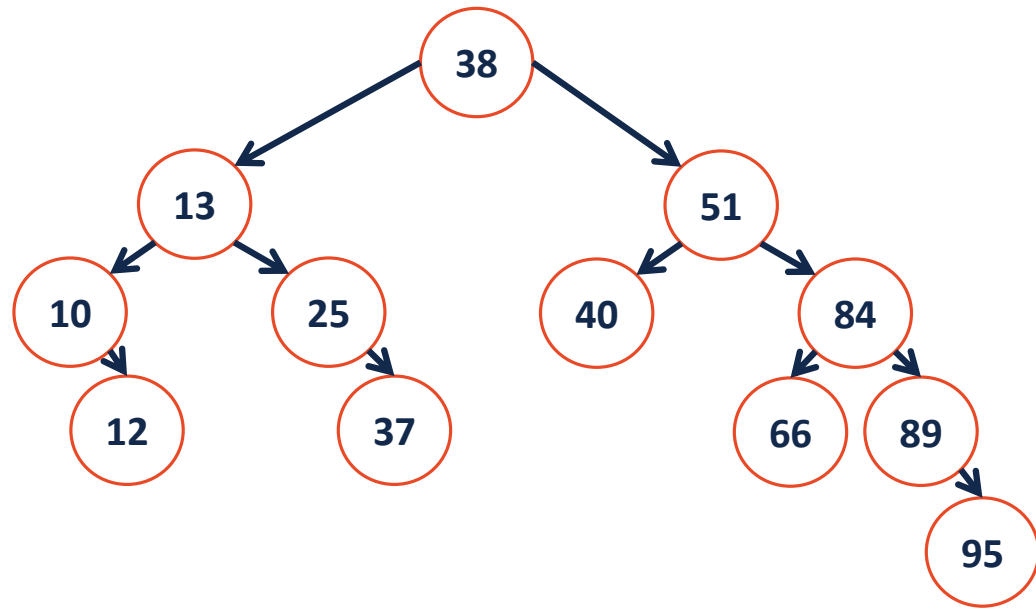




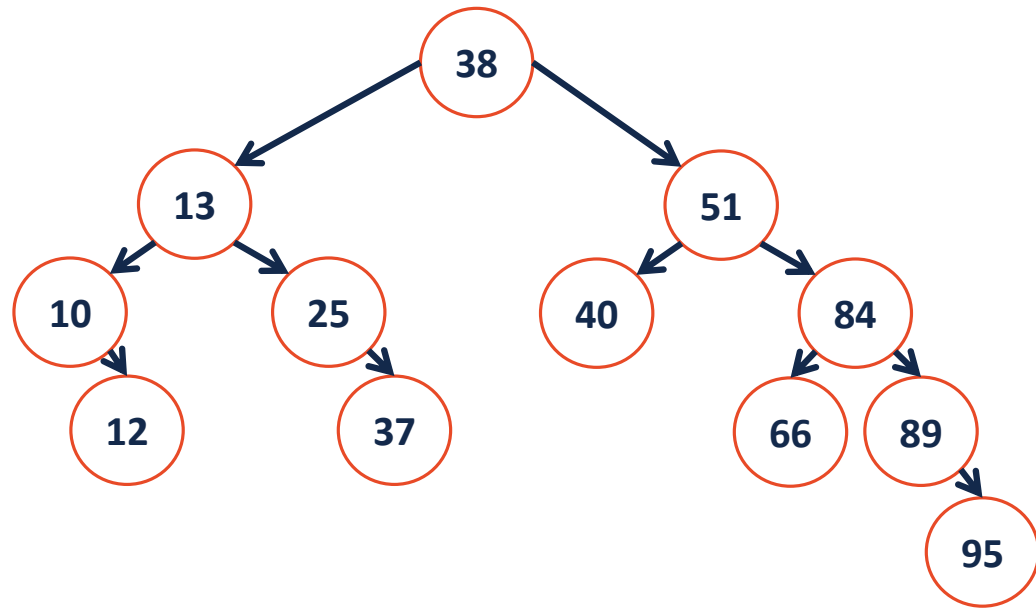
`remove (40) ;`



remove (25) ;



`remove(10);`



`remove (13) ;`