CS 225

Data Structures

February 16 – Trees and Traversal

G Carl Evans

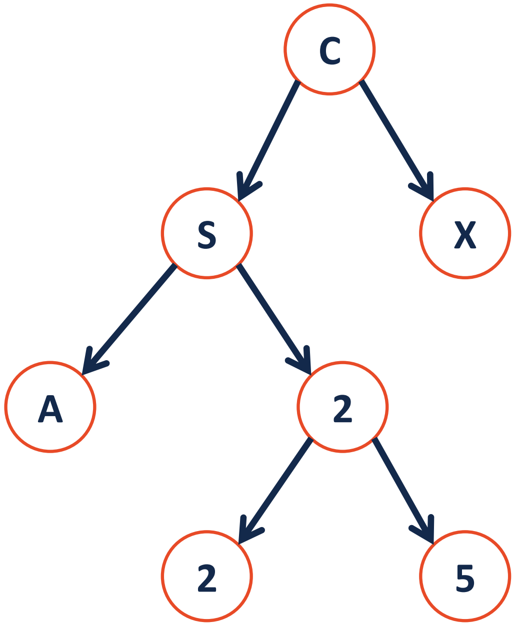
Binary Tree – Defined

A *binary tree* T is either:

-

OR

-

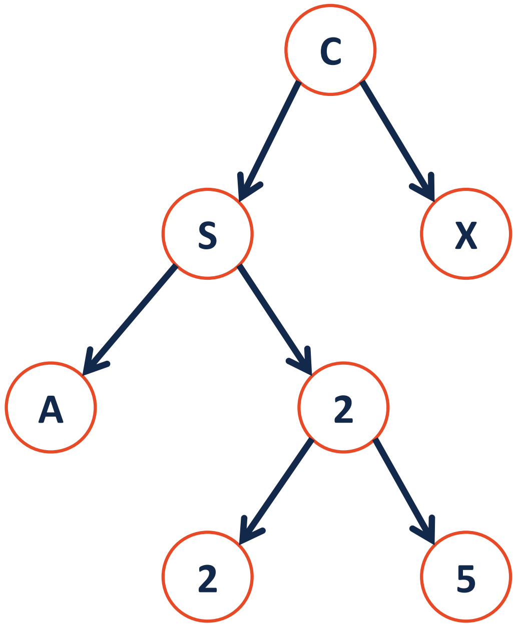


Tree Property: height

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

Given a binary tree T:

height(T) =

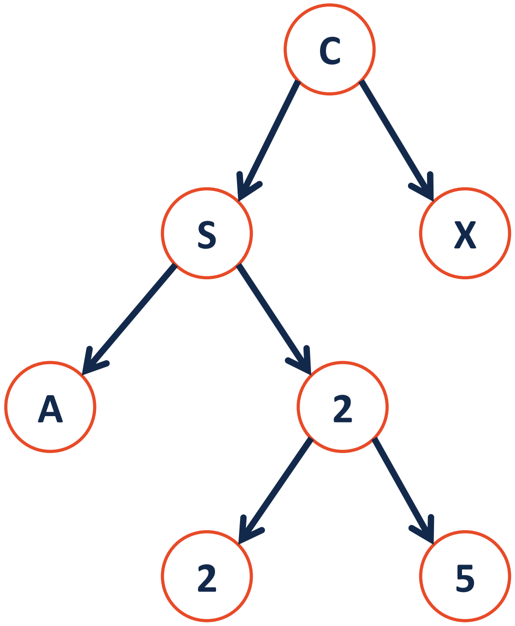


Tree Property: full

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

1.

2.

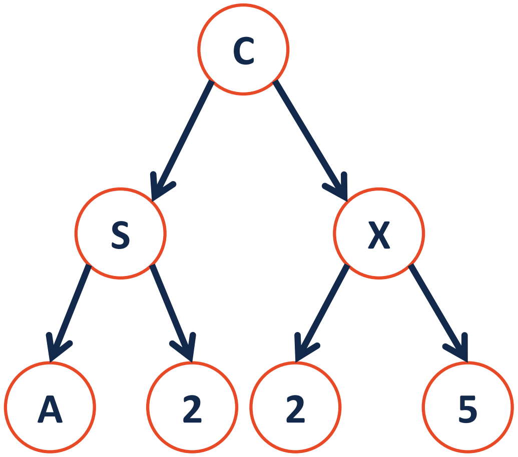


Tree Property: perfect

A **perfect** tree ***P*** is:

1.

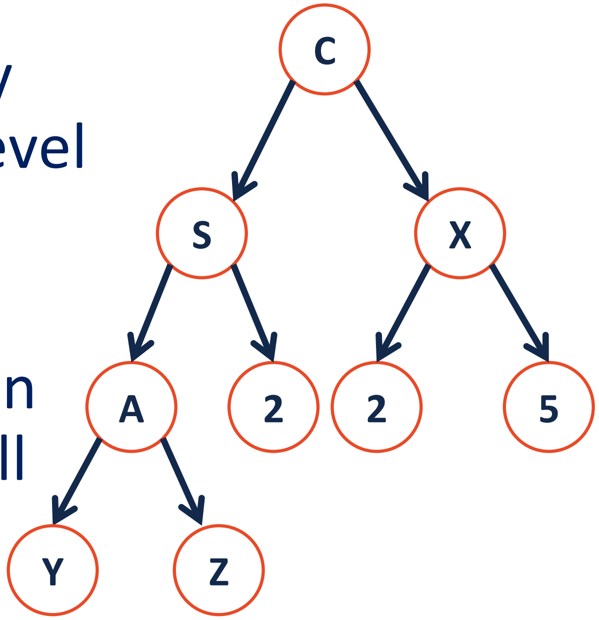
2.



Tree Property: complete

Conceptually: A perfect tree for every level except the last, where the last level if "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".



Tree Property: complete

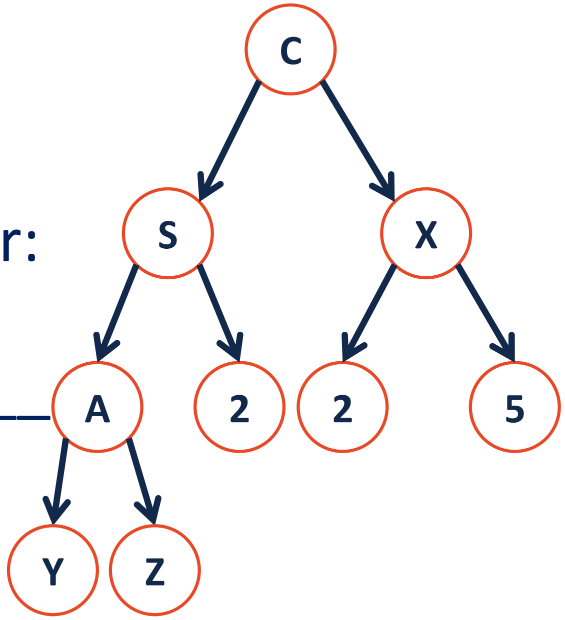
A **complete** tree ***C*** of height **h**, **C_h**:

1. **C_{-1}** = {}
2. **C_h** *(where h>0)* **= {r, T_L, T_R}** and either:

T_L is ___________ and T_R is ___________

OR

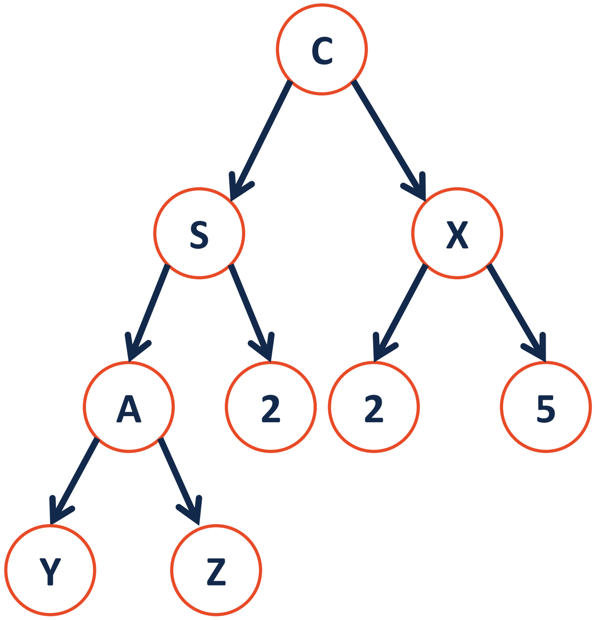
T_L is ___________ and T_R is ___________



Tree Property: complete

Is every **full** tree **complete**?

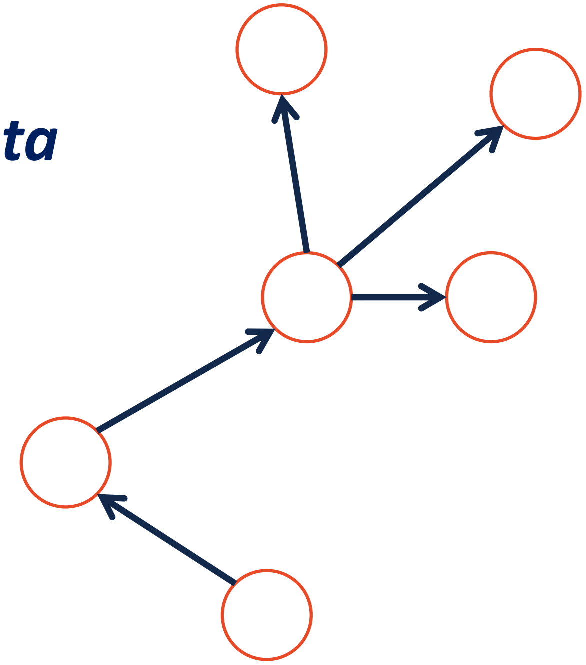
If every **complete** tree **full**?



Trees

“The most important non-linear data structure in computer science.”

- David Knuth, The Art of Programming, Vol. 1



A tree is:

-
-

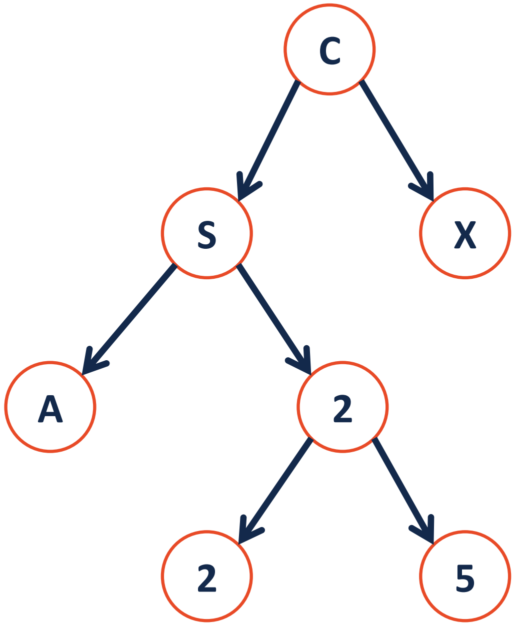
Binary Tree – Defined

A *binary tree* T is either:

-

OR

-

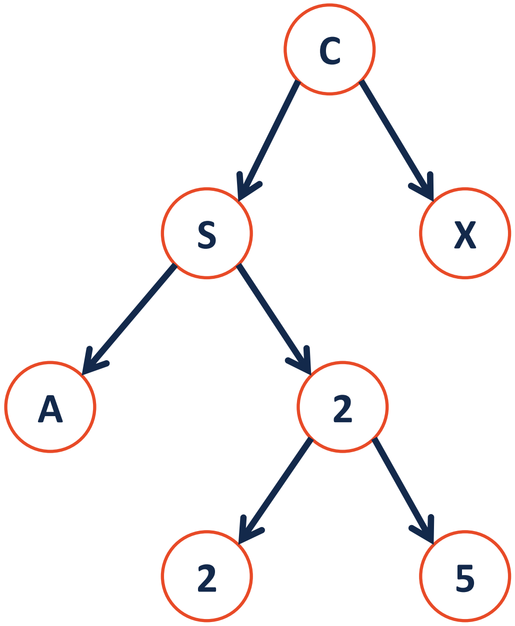


Tree Property: height

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

Given a binary tree T:

height(T) =

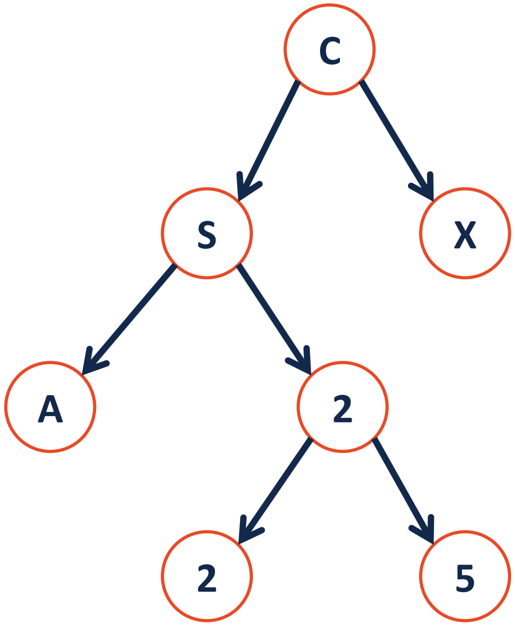


Tree Property: full

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

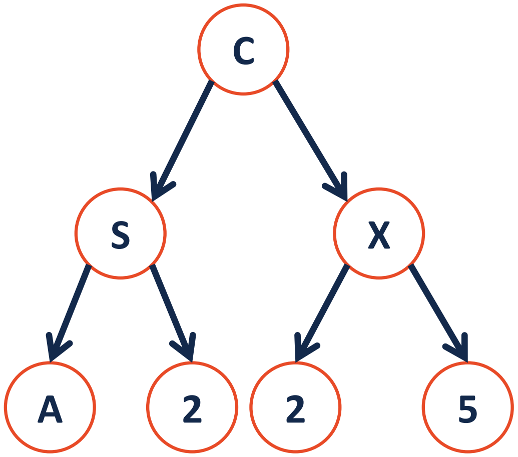
1.

2.



Tree Property: perfect

A **perfect** tree ***P*** is defined in terms of the tree’s height.



Let $\mathbf{P_h}$ be a perfect tree of height **h**, and:

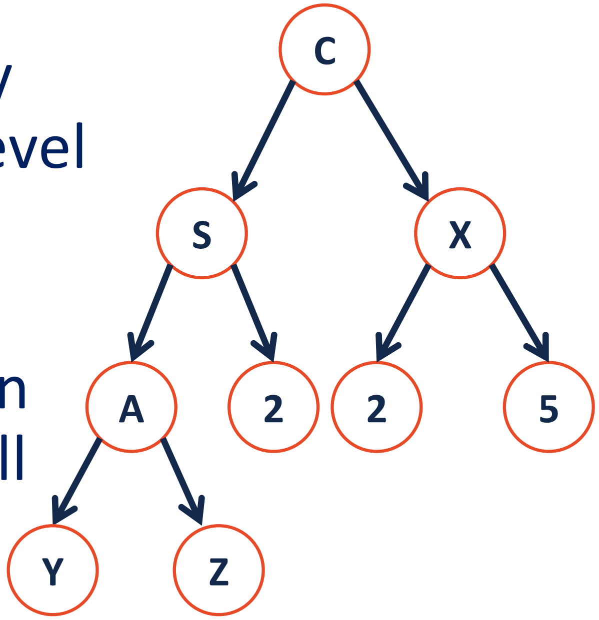
1.

2.

Tree Property: complete

Conceptually: A perfect tree for every level except the last, where the last level if "pushed to the left".

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



Tree Property: complete

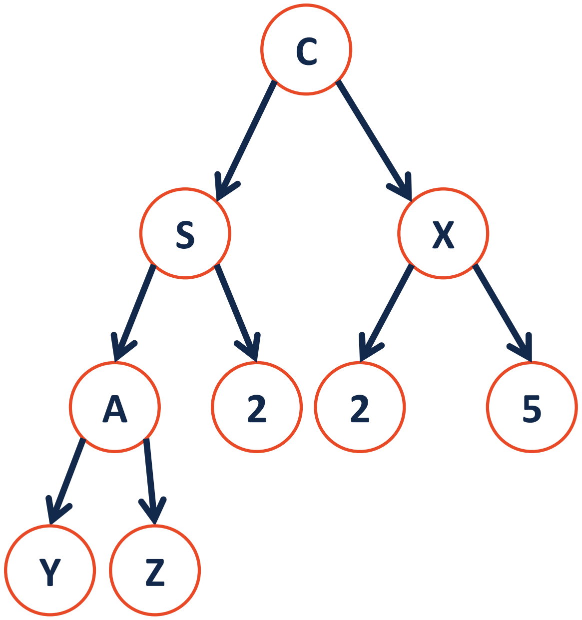
A **complete** tree ***C*** of height **h**, $\mathbf{C_h}$:

1. $\mathbf{C_{-1}}$ = {}
2. $\mathbf{C_h}$ *(where h>0)* = **{r, T_L, T_R}** and either:

T_L is ____________ and T_R is ___________

OR

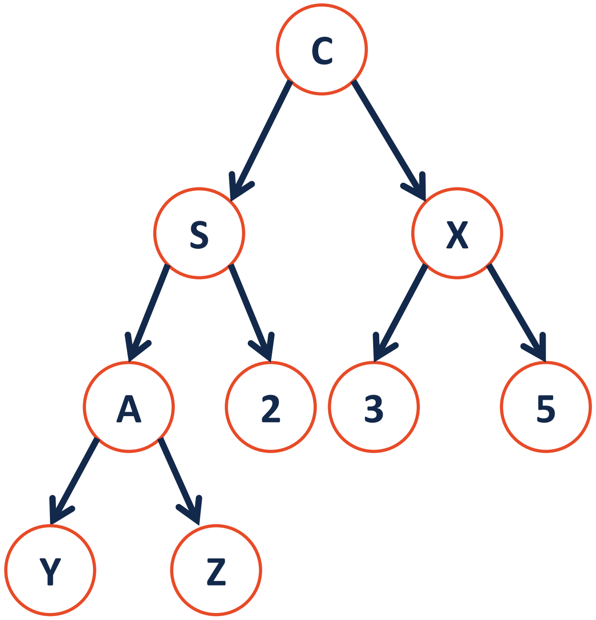
T_L is ____________ and T_R is ___________



Tree Property: complete

Is every **full** tree **complete**?

If every **complete** tree **full**?



Tree ADT

Tree ADT

insert, inserts an element to the tree.

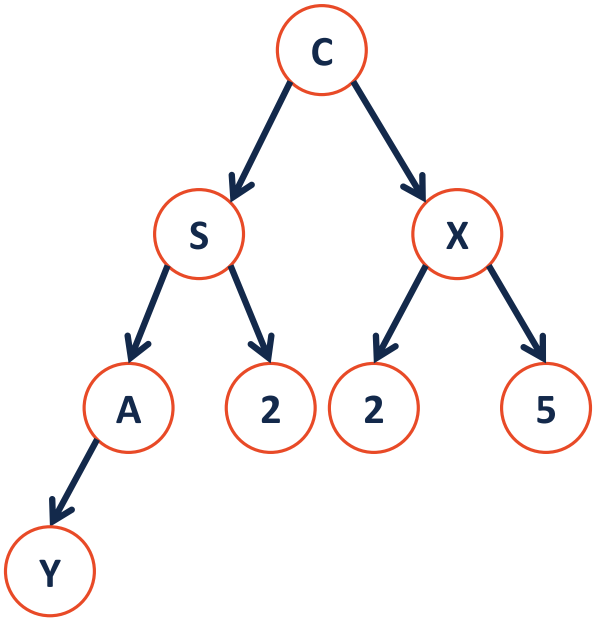
remove, removes an element from the tree.

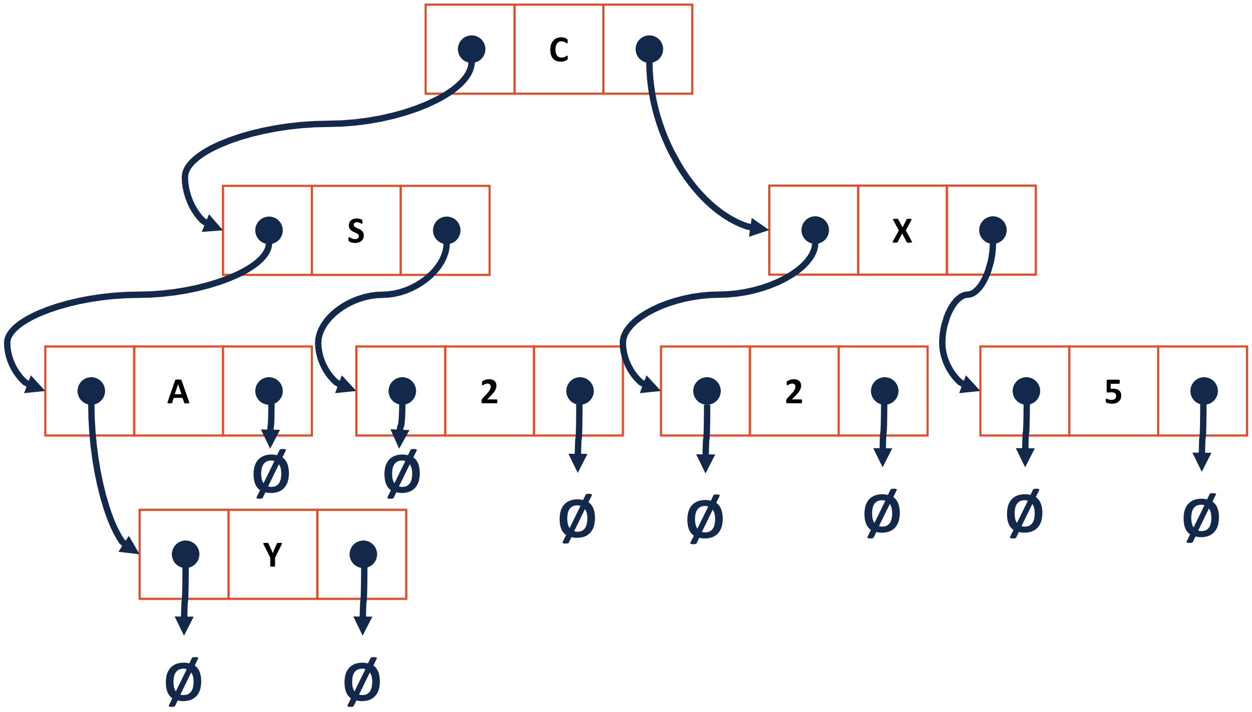
traverse,

BinaryTree.h

```
#pragma once

template <class T>
class BinaryTree {
  public:
    /* ... */

  private:










};
```

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):

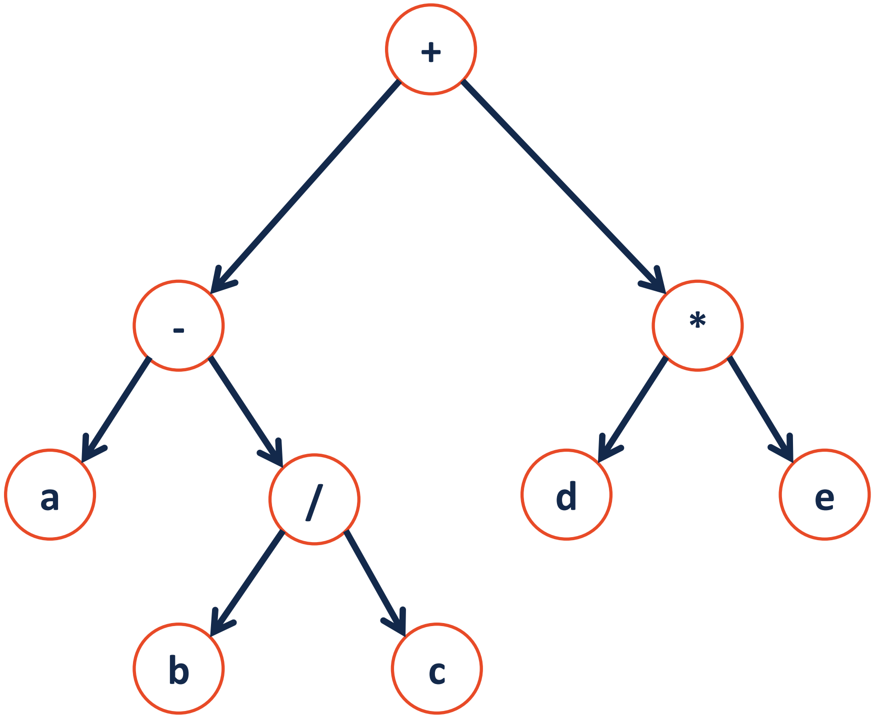
How many nullptrs?

Induction Hypothesis:

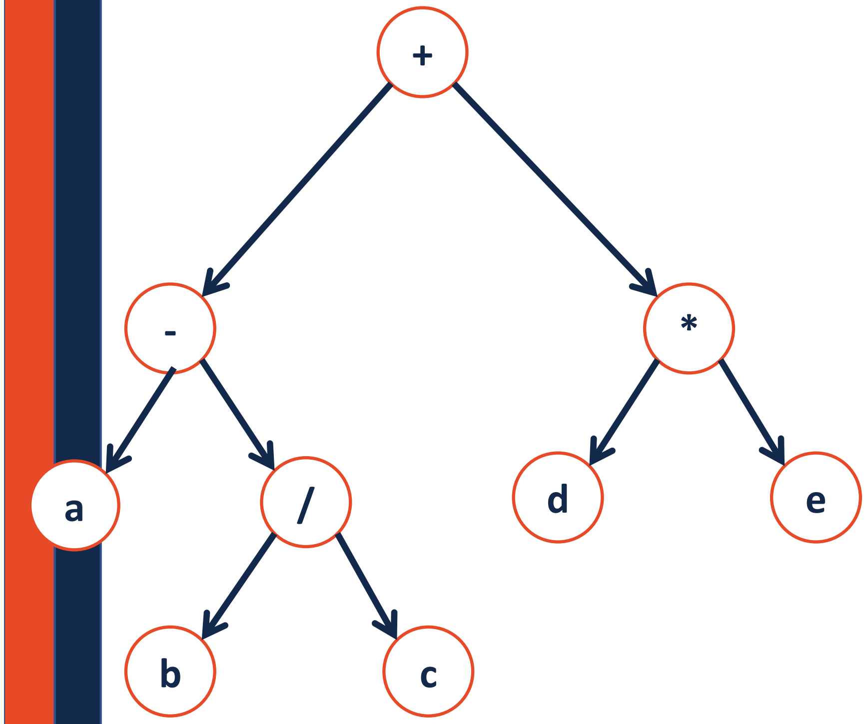
How many nullptrs?

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

Traversals

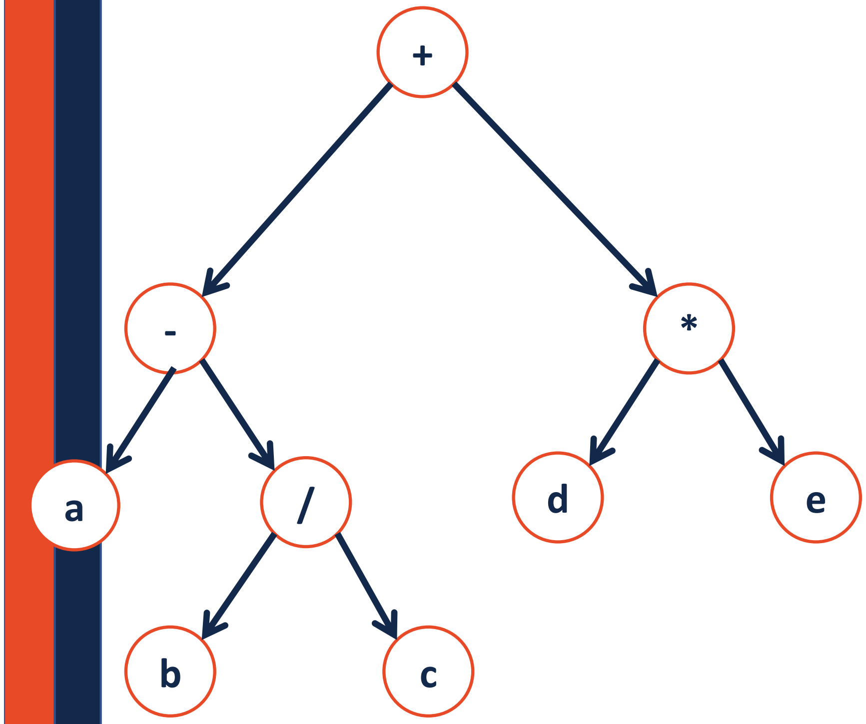


Traversals



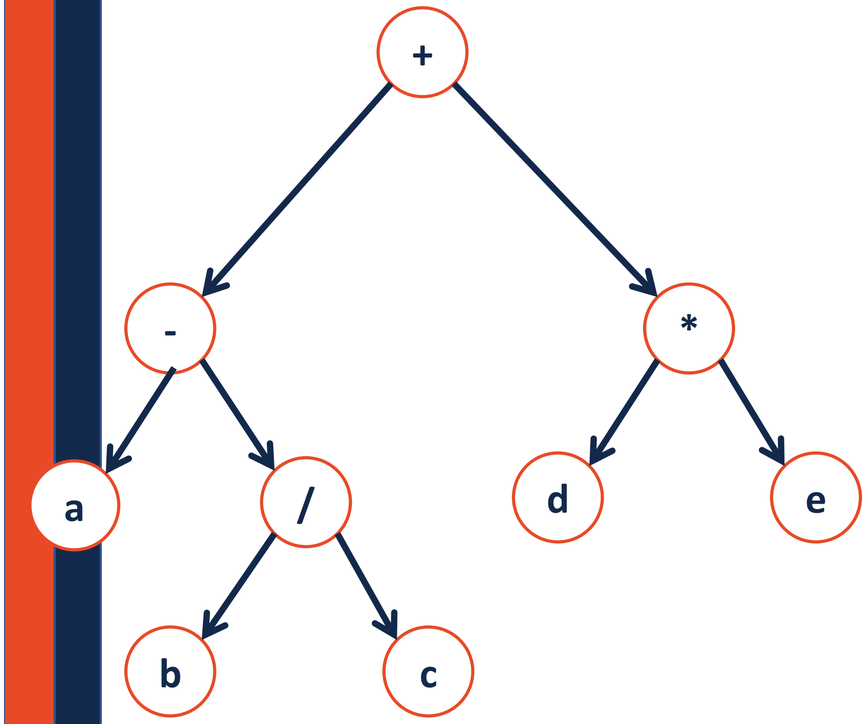
```
template<class T>
void BinaryTree<T>::__Order(TreeNode * cur)
{






}
```

Traversals



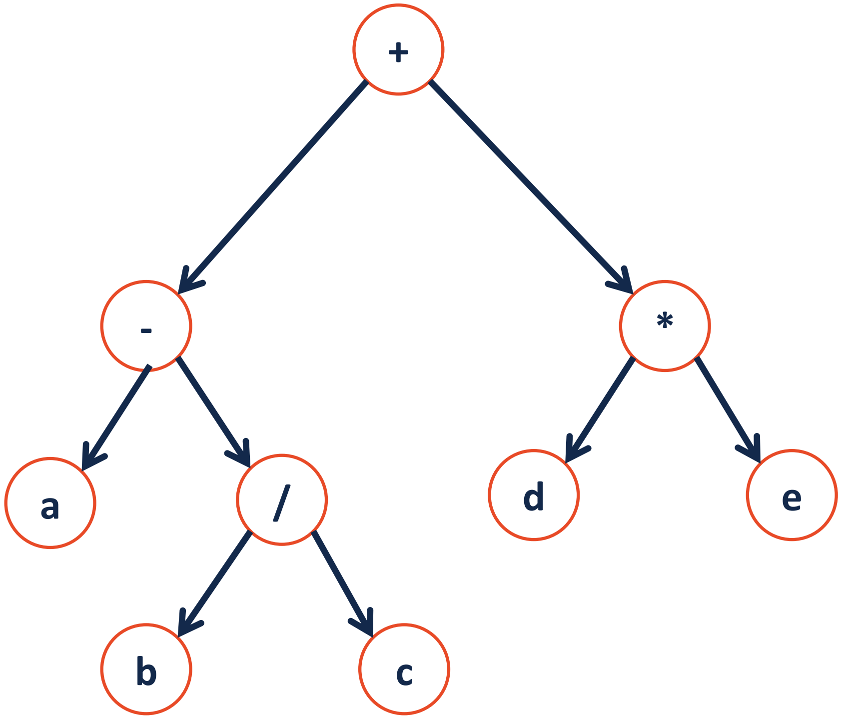
```
template<class T>
void BinaryTree<T>::___Order(TreeNode * cur) {
   if (cur != NULL) {

      ______________________;

      ___Order(cur->left);

      ______________________;

      ___Order(cur->right);

      ______________________;

   }
}
```

Traversals

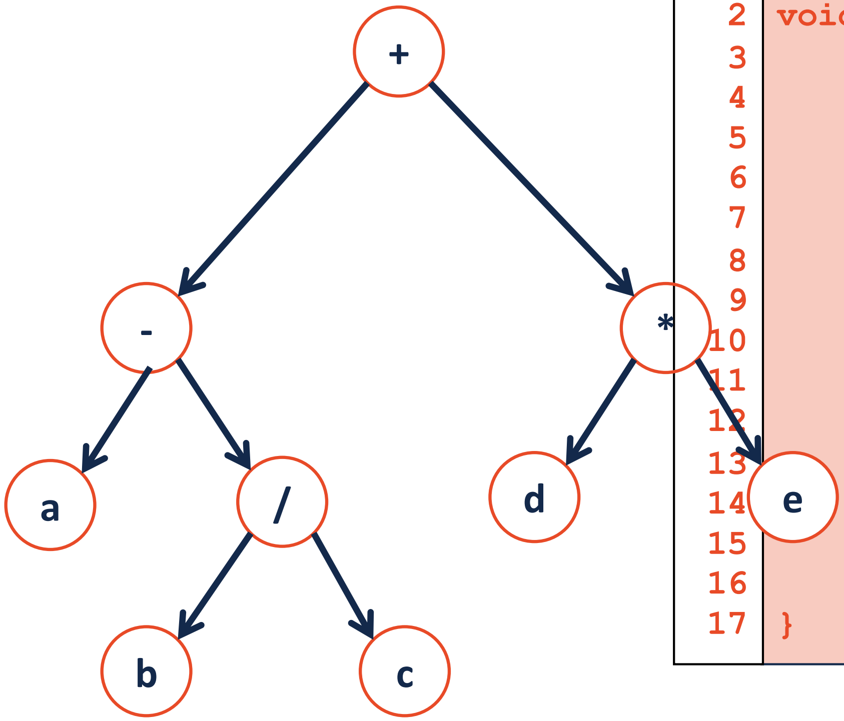


```
template<class T>
void BinaryTree<T>::___Order(TreeNode * cur) {
   if (cur != NULL) {
      ______________________;
      ___Order(cur->left);
      ______________________;
      ___Order(cur->right);
      ______________________;
  }
}
```

A Different Type of Traversal



A Different Type of Traversal



```
template<class T>
void BinaryTree<T>::levelOrder(TreeNode * root) {














}
```

Traversal vs. Search

Traversal

Search

Search: Breadth First vs. Depth First

Strategy: Breadth First Search (BFS)

Strategy: Depth First Search (DFS)

Dictionary ADT

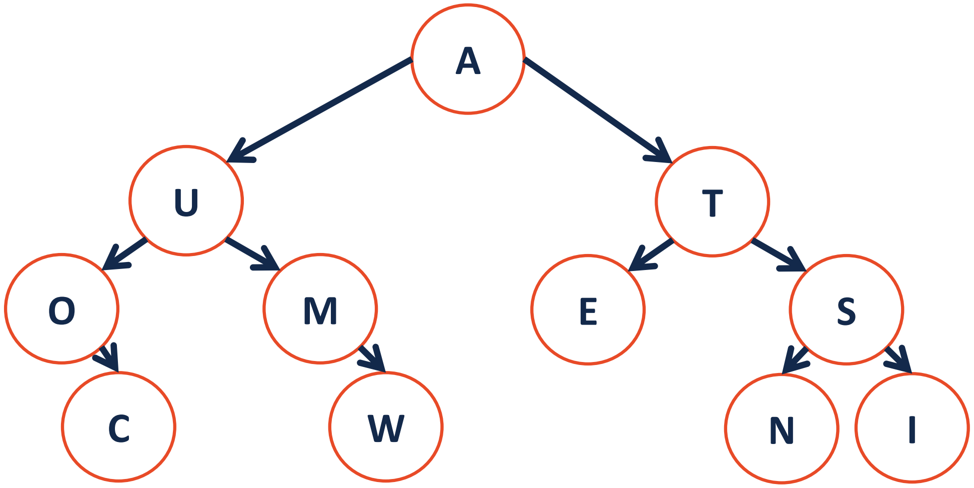
Data is often organized into key/value pairs:

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

Dictionary.h

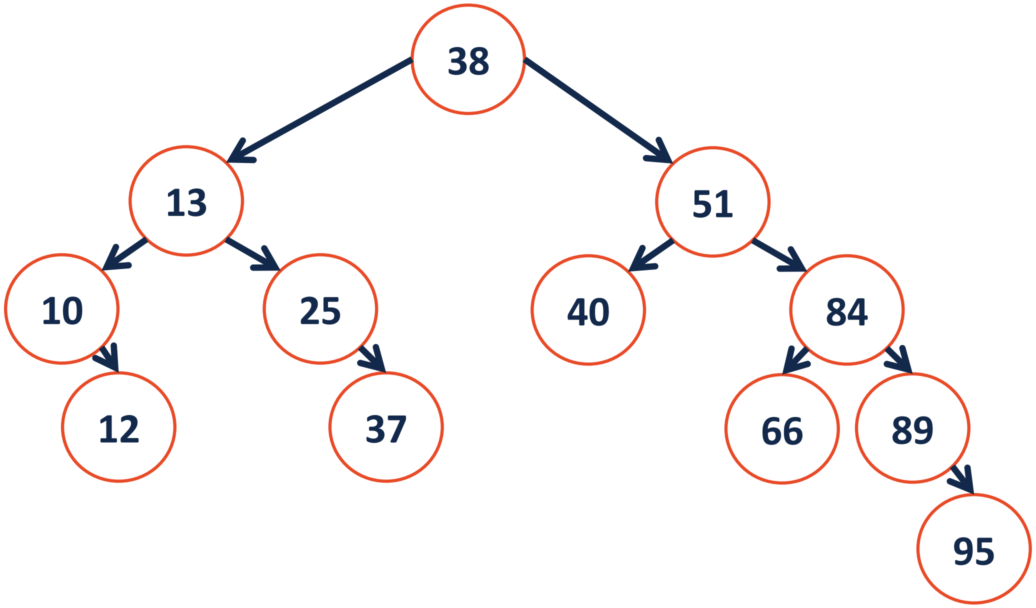
```
#pragma once


class Dictionary {
  public:














  private:
    // ...
};
```

Binary Tree as a Search Structure



Binary ______________ Tree (BST)

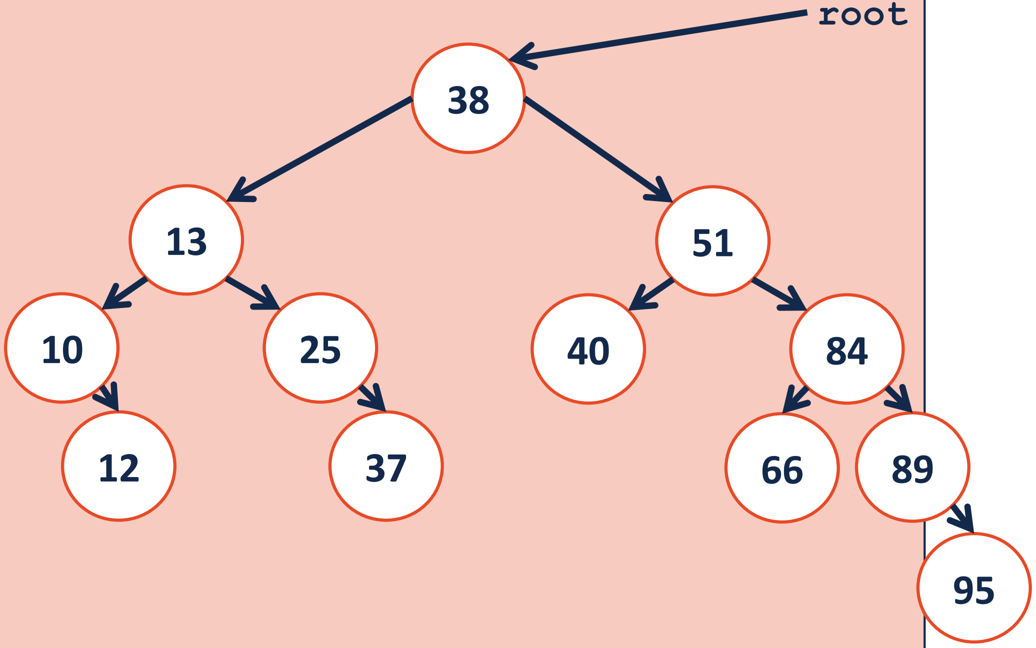
A **BST** is a binary tree **T** such that:

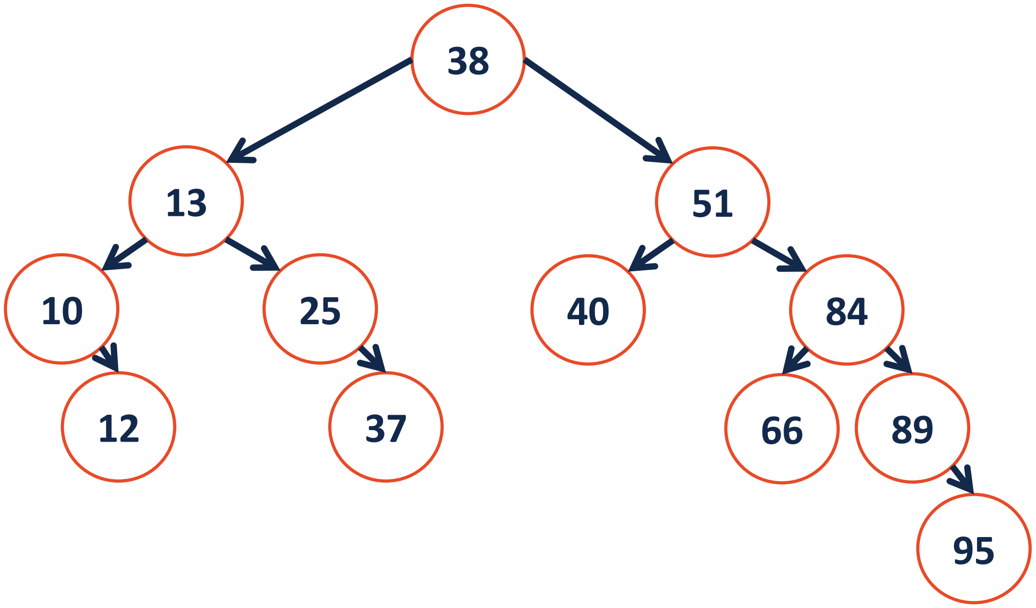


BST.h

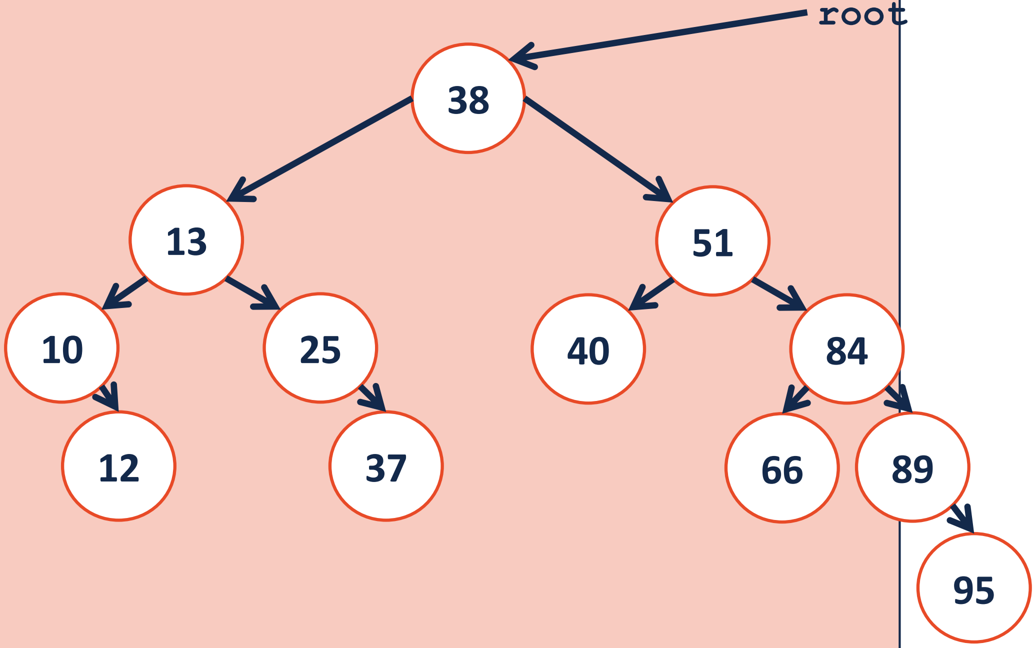
```
#pragma once

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:









};
```

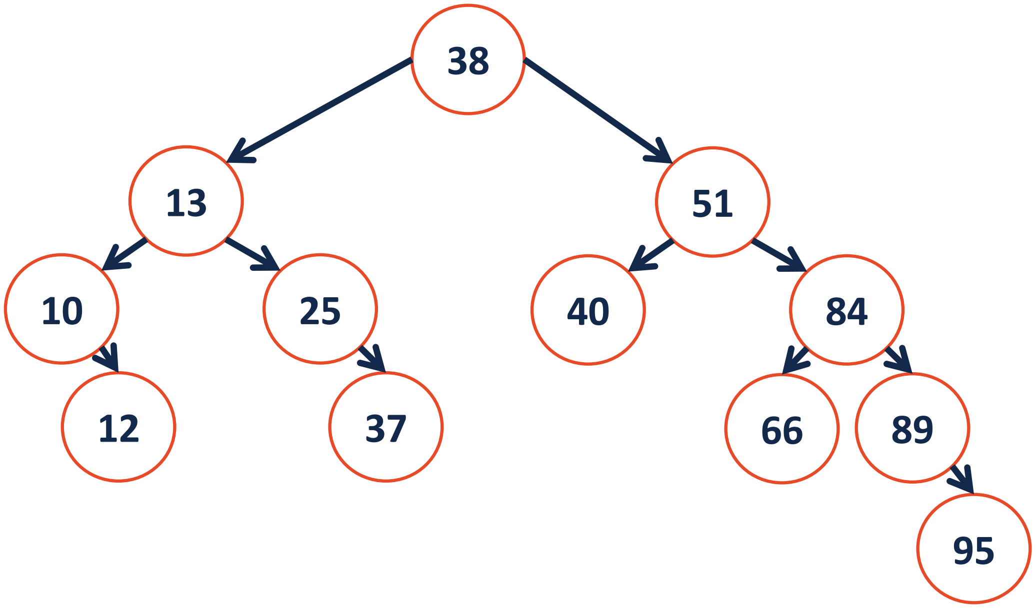
```
template<class K, class V>

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
























}
```



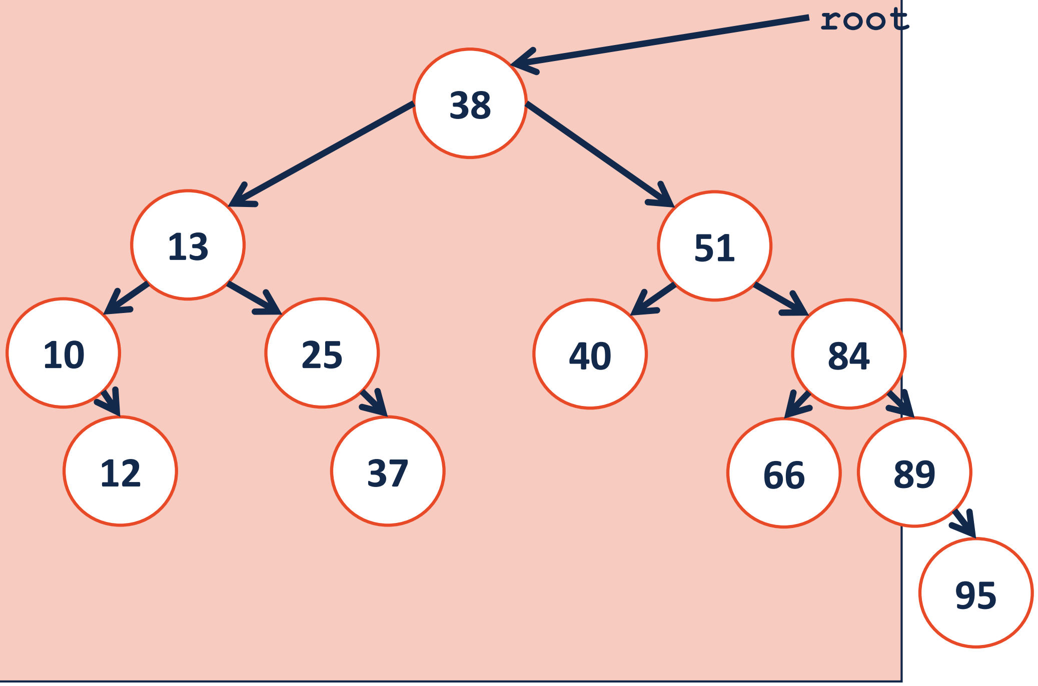

38
13
51
10
25
40
84
12
37
66
89
95

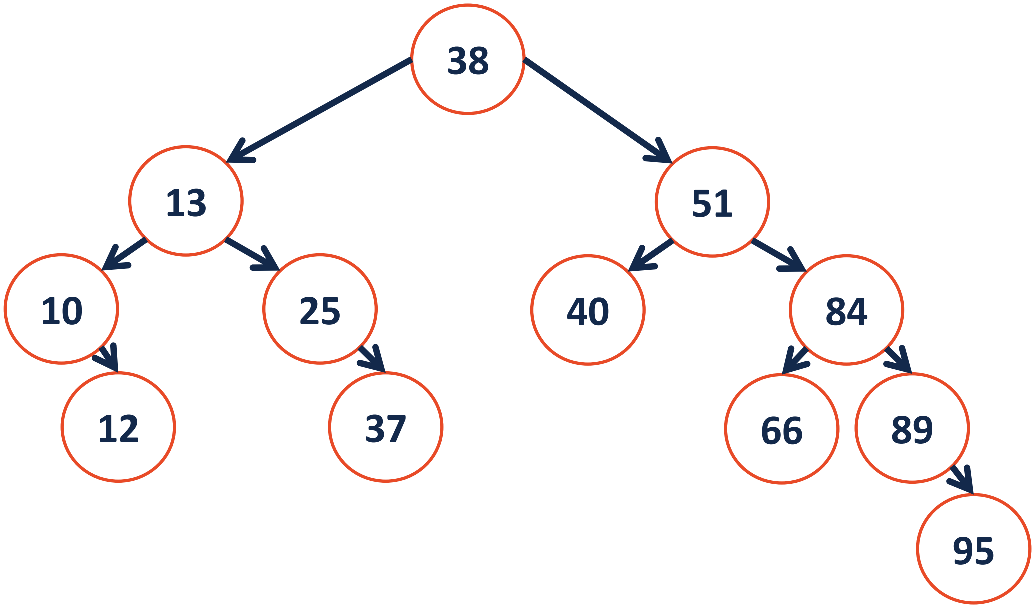
```
template<class K, class V>

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























}
```



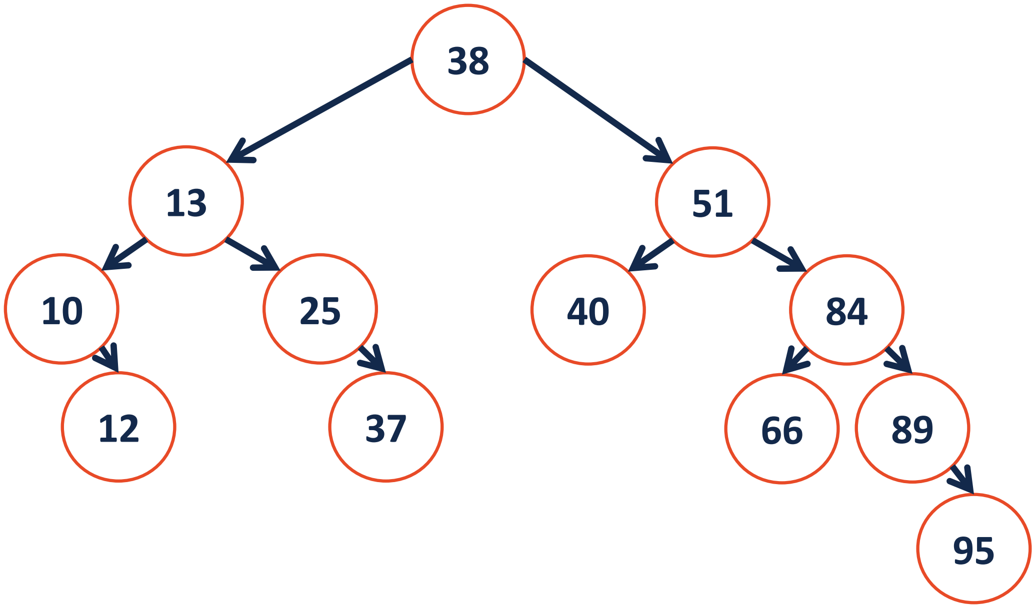

38
13
51
10
25
40
84
12
37
66
89
95

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
























}
```

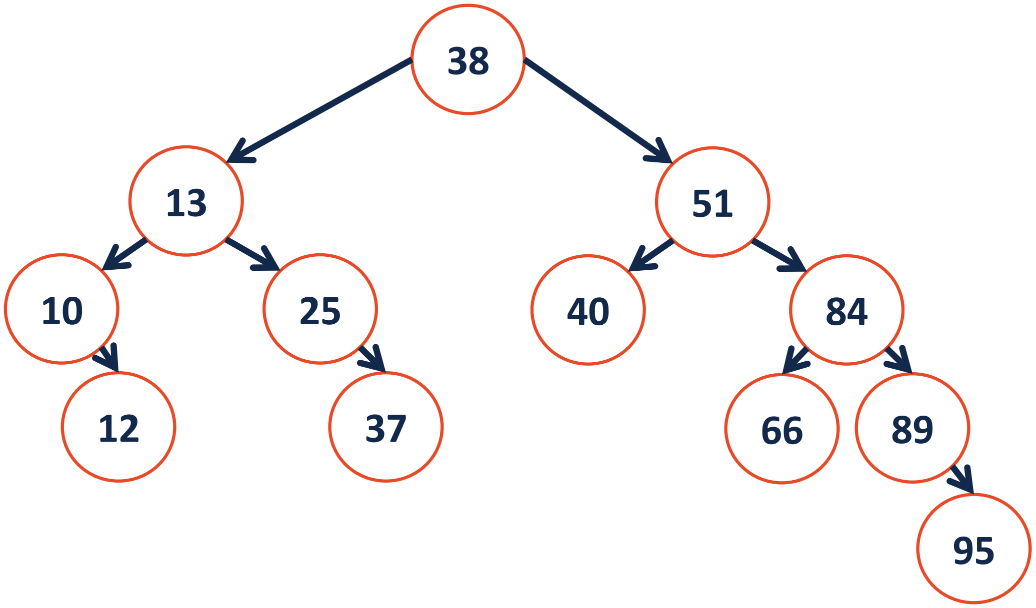




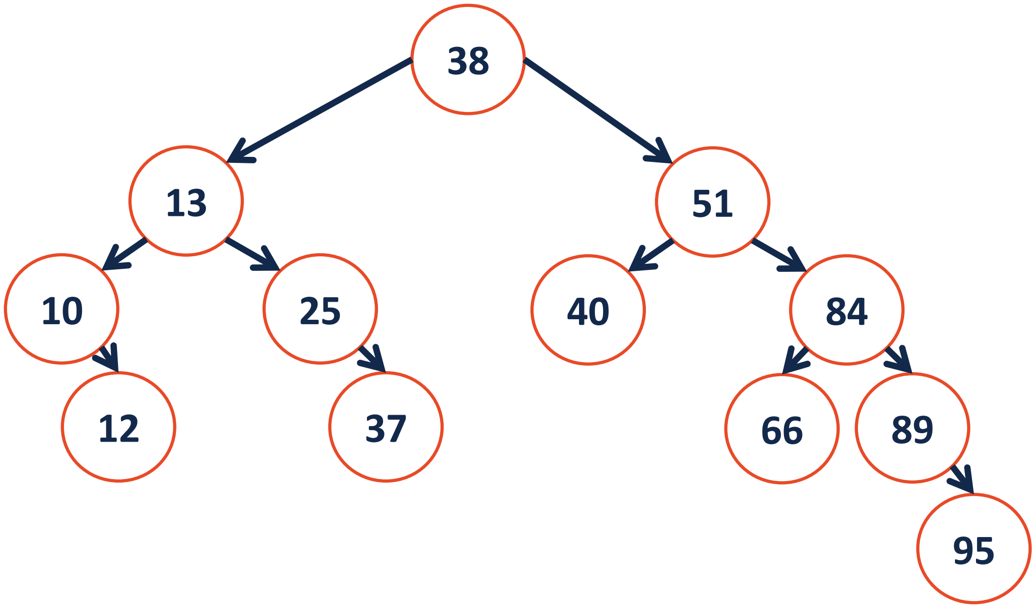
`remove(40);`



`remove(25);`



`remove(10);`



`remove(13);`