Data Structure Course

Problem 1 - Stack

In this problem, you should develop a stack class similar to that provided in the C++ STL. You cannot use any of the C++ STL classes in this problem.

Given a string path, which is an absolute path (starting with a slash '/') to a file or directory in a Unix-style file system, convert it to the simplified canonical path.

In a Unix-style file system, a period '.' refers to the current directory, a double period '..' refers to the directory up a level, and any multiple consecutive slashes (i.e. '//') are treated as a single slash '/'. For this problem, any other format of periods such as '...' are treated as file/directory names.

The canonical path should have the following format:

The path starts with a single slash '/'.

Any two directories are separated by a single slash '/'.
The path does not end with a trailing '/'.
The path only contains the directories on the path from the root directory to the target file or directory (i.e., no period '.' or double period '..')

Return the simplified canonical

path. Example 1:

Input: path = "/home/"
Output: "/home"

Example 2:

Input: path = "/../"
Output: "/"

Example 3:

Input: path = "/home//foo/"

Output: "/home/foo"

Problem 2- Queue

In this problem, you should develop a queue class similar to that provided in the C++ STL. You cannot use any of the C++ STL classes in this problem.

There are n people in a line queuing to buy tickets, where the 0th person is at the front of the line and the (n - 1)th person is at the back of the line.

You are given a 0-indexed integer array of tickets of length n where the number of tickets that the ith person would like to buy is tickets[i].

Each person takes exactly 1 second to buy a ticket. A person can only buy 1 ticket at a time and has to go back to the end of the line (which happens instantaneously)

in order to buy more tickets. If a person does not have any tickets left to buy, the person will leave the line.

Return the time taken for the person at position k (0-indexed) to finish buying tickets.

Example 1:

Input: tickets = [2,3,2], k = 2
Output: 6
Explanation:

In the first pass, everyone in the line buys a ticket and the line becomes [1, 2, 1].

In the second pass, everyone in the line buys a ticket and the line becomes [0, 1, 0].

The person at position 2 has successfully bought 2 tickets and it took 3 + 3 = 6 seconds.

Example 2:

Input: tickets = [5,1,1,1], k = 0
Output: 8

Explanation:

In the first pass, everyone in the line buys a ticket and the line becomes [4, 0, 0, 0].

In the next 4 passes, only the person in position 0 is buying tickets.

The person at position 0 has successfully bought 5 tickets and it took 4 + 1 + 1 + 1 + 1 = 8 seconds.

Problem 3 - Queue

In this problem, you must use your queue. Reimplement part of the stack data structure using only one queue as an underlying data structure. Reimplement the class for 'int' data type only and with the following functions only:

int top() – returns the top element. void pop()

– removes the top element.

void push(int value) – adds an element to the top of the stack . Write a

main function to test .

Problem 4 - Tree

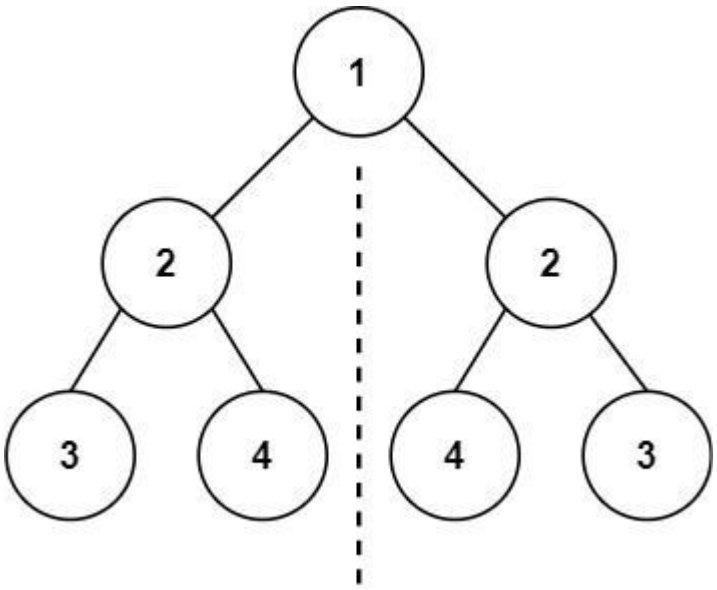
Given the following code

```
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
};

class Solution {
```

```
public:
    bool isSymmetric(TreeNode* root) {
        // write you code here
    }
};

// add main method and test cases to test your code
```

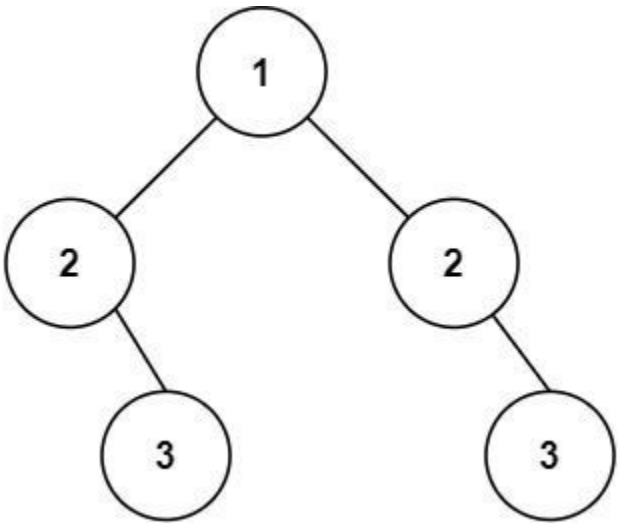
Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

Example 1:




```
Input: root = [1,2,2,3,4,4,3]

Output: true
```

Example 2:




```
Input: root =

[1,2,2,null,3,null,3] Output:

false
```

Problem 5 - Tree

Given the following code

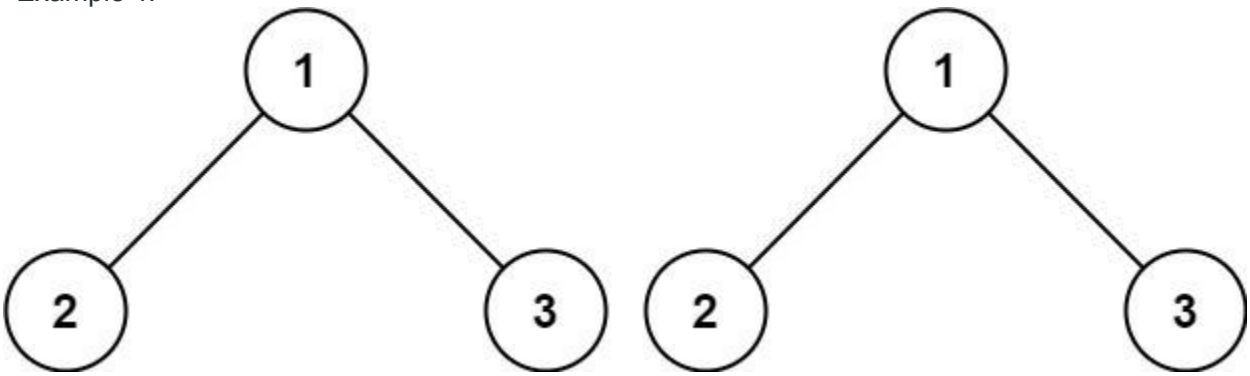
```
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
};

class Solution {
public:
   bool isSameTree(TreeNode* p, TreeNode* q) {
      // write you code here
```

```
    }
};

// add main method and test cases to test your code
```

Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

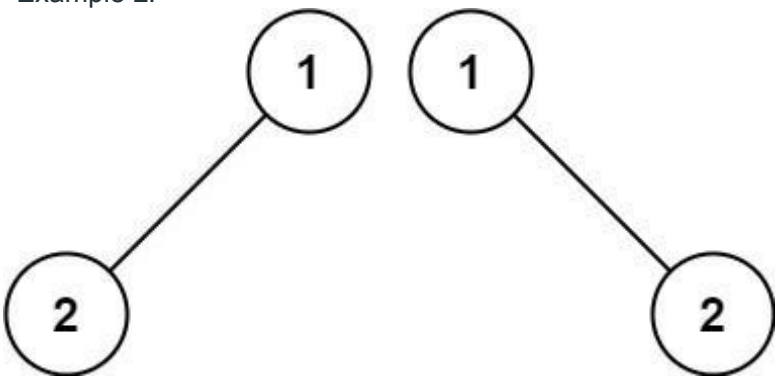
Example 1:




```
Input: p = [1,2,3], q = [1,2,3]
```

```
Output: true
```

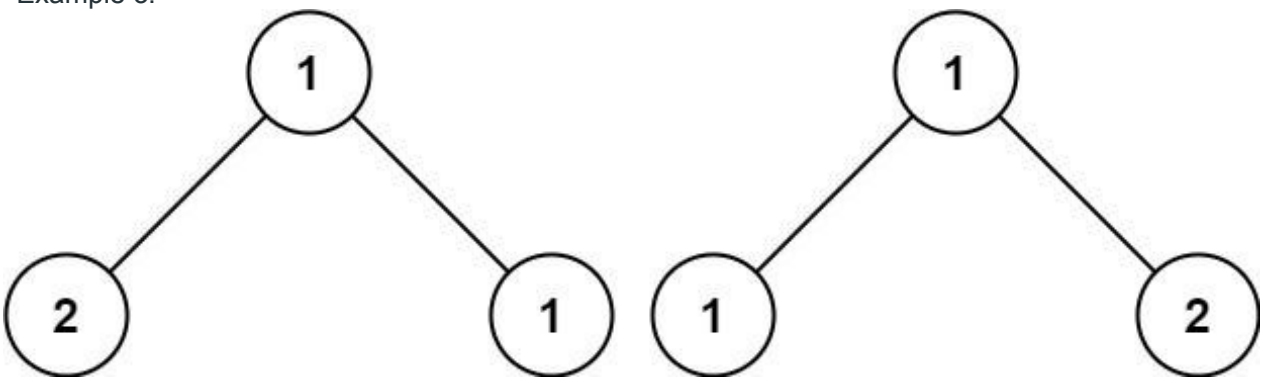
Example 2:




```
Input: p = [1,2], q = [1,null,2]
```

```
Output: false
```

Example 3:




```
Input: p = [1,2,1], q = [1,1,2]
Output: false
```

Problem 6 - Tree

Expression Tree Evaluation:

Write an algorithm that accepts an arithmetic expression written in prefix notation and builds an expression tree for it, then traverse to evaluate the expression. The evaluation should start after the whole expression has been entered.

1. Implement your algorithm as a function
2. Write five test cases for your application and run them correctly

```
expression: + 3 * 4 / 8 2
tree:
```


```
Evaluation: 19
```

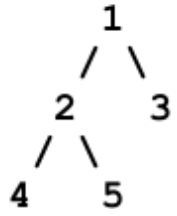
Problem 7 - Tree

Tree Flipping

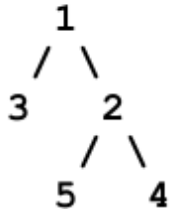
Assume a binary tree. Write a flip method that takes the node which the mirror image of the tree will start from, if no parameter is sent to the function the default value will be the root node.

3. Implement your algorithm as a function [void flip(Node* node = root)]
4. Write five test cases for your application and run them correctly

```
Example original tree:
```




```
Example new tree:
```



Problem 8 - Tree

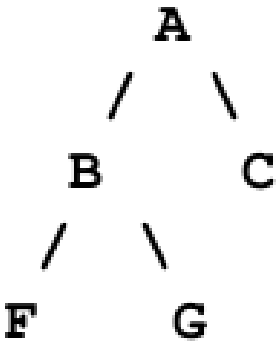
Tree Traversal

Assume a binary tree of non-repeated characters. Write a function printPostOrder that takes two strings representing the preorder traversal and the in-order traversal of the tree. Then the function prints the post-order traversal. The function prototype is: void printPostOrder (string preorder, string inorder)

A sample function call and the corresponding output is shown below:

printPostOrder ("ABFGC", "FBGAC")

=> FGBCA

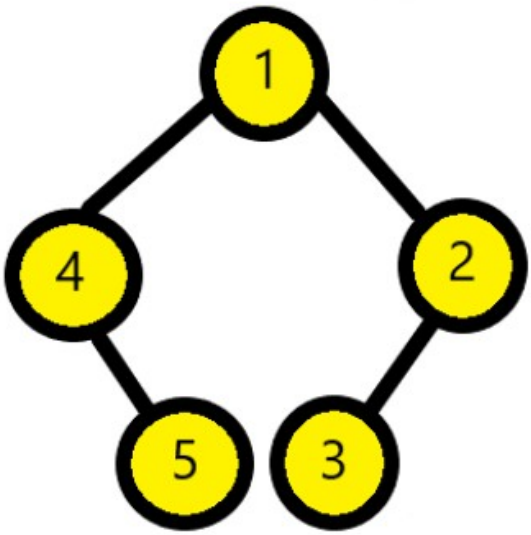


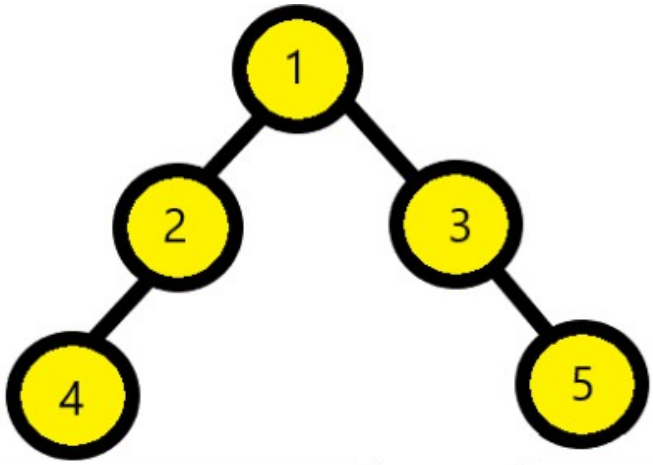
5 . Implement your algorithm as a function
6. Write five test cases for your application and run them correctly

Problem 9 - Tree

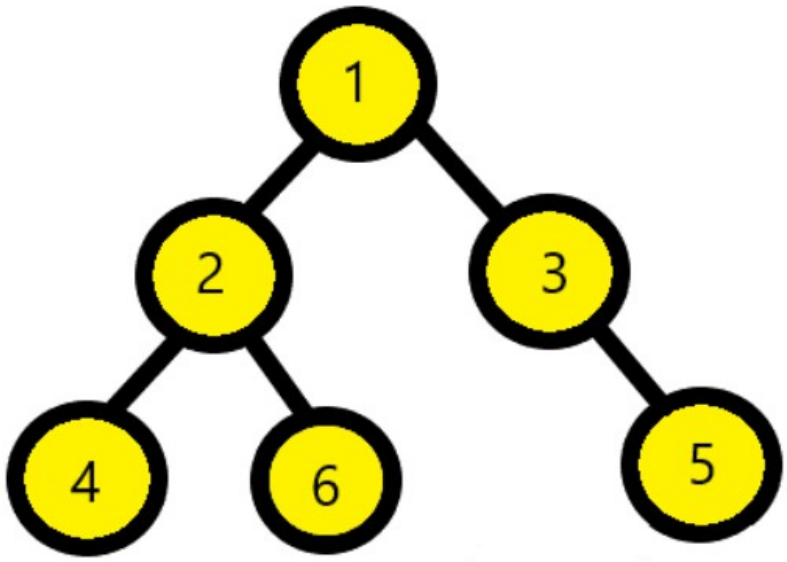
You should implement a method to check if a Binary Tree is foldable or not:

A binary tree is foldable if the left subtree and right subtree are mirror images of each other. An empty tree is also foldable. Below binary trees are foldable since both have a left subtree that is the mirror image of the right subtree, for example the following trees:





However, the below binary tree is not foldable

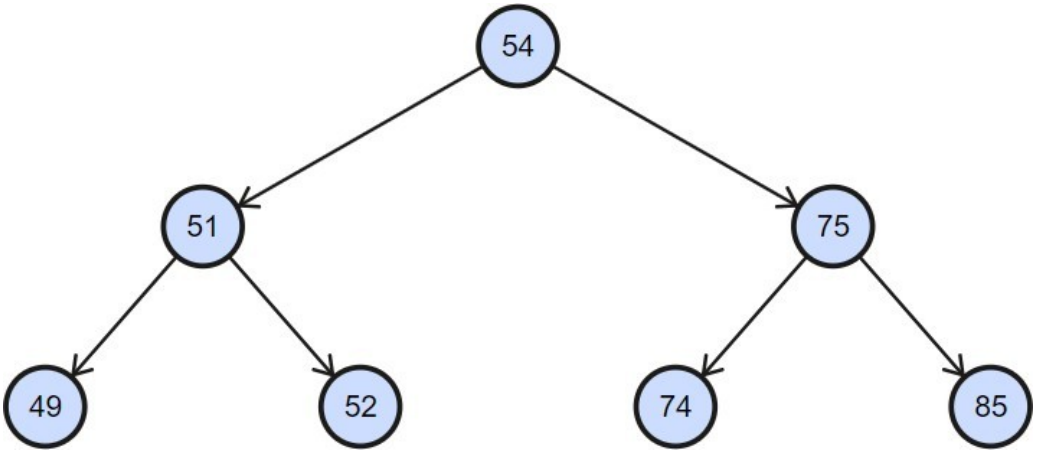


7 . Implement your algorithm as a function
8 . Write five test cases for your application and run them correctly

Problem 10 - Tree

Given a binary search tree and an integer k, our task is to implement a method to find out the sum of all the elements which is less or equal to the kth smallest element in the binary search tree.

For example, for given below binary search tree and k=3:



Explanation: so here in above example k th smallest element is 52 and sum of all the elements which is less or equal to 52 is 152.

9 . Implement your algorithm as a function
10. Calculating the sum
11. Write five test cases for your application and run them correctly

Problem 11 - Tree

- Create template binary search tree class with this name

BSTFCI and create node class with name BSTNode

- Implement BSTFCI class
- Implement BSTNode class

- Add Checking Tree Balance

A Balanced Binary Tree is a tree where the heights of the two child sub-trees of any node differ by at most one AND the left subtree is balanced AND the right subtree is balanced.

Add method called “isBalance” to BSTFCI this method will check in the BST is balanced or not.

- Implement isBalance method
- Write five test cases and run them correctly

- Tree Comparison

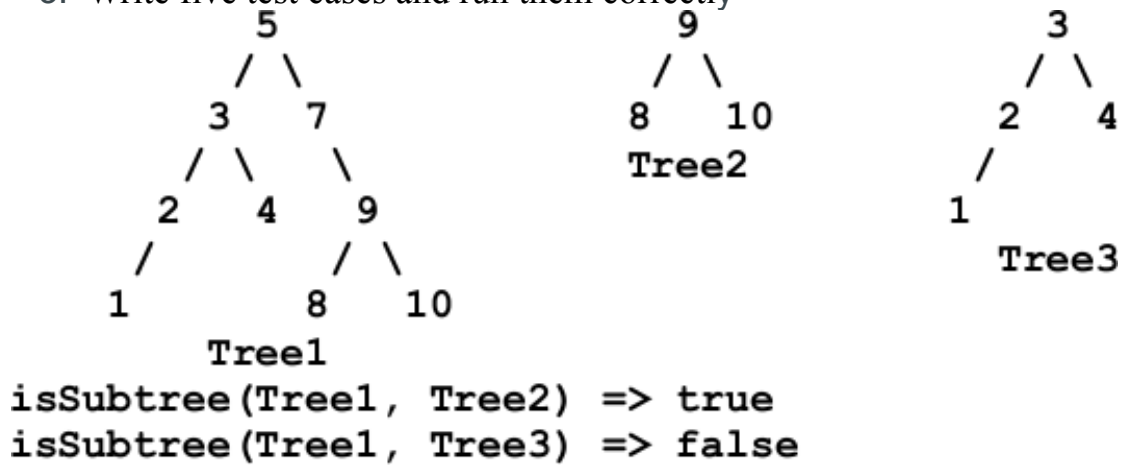
Write a function that decides if a BSTFCI T2 is a subtree of another BSTFCI T1. Prototype:

bool isSubTree(BSTFCI* t1, BSTFCI* t2);

Note: You may need to write another function that takes 2 BSTNodes and

compares their sub-trees.

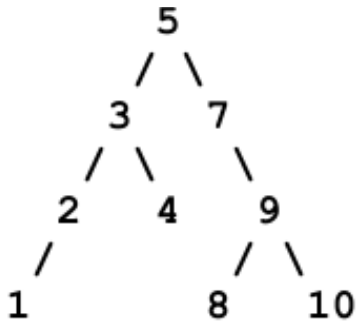
2. Implement isSubTree method
3. Write five test cases and run them correctly



- Print Range

Add a recursive function named printRange in the class BSTFCI that stores integers and given a low key value and a high key value, it prints in sorted order all records whose key values fall between the two given keys. Function printRange should visit as few nodes in the BST as possible. You should NOT traverse ALL the tree inorder and print the ones in the range. This will not be considered a correct solution. You should do smart traversal to only traverse the related parts.

- Implement printRange method
- Write five test cases and run them correctly




```
printRange(3, 6)   => [3,4,5]
printRange(8, 15)  => [8,9,10]
printRange(6,6)    => []
```