CSE 4304-Data Structures Lab. Winter 2022-23
Batch: CSE 21
Lab: 09
Date: October 16, 2023.
Target Group: All
Topic: Binary Trees

Instructions:

- Regardless of finishing the lab tasks within lab time, you must submit the solutions in the Google Classroom. If I forget to upload the tasks there, CR should contact me. The deadline will always be 11:59 PM on the day the lab has occurred.
- Task naming format: fullID_T01L07_2A.c/cpp.
- If you find any issues in the problem description/test cases, comment in the Google Classroom.
- If you find any tricky test case that I didn't include, but others might forget to handle, please comment! I'll be happy to add.
- Use appropriate comments in your code. This will help you to easily recall the solution in the future.
- Obtained marks will vary based on the efficiency of the solution.
- Do not use <bits/stdc++.h> library.
- Modified sections will be marked with BLUE color.

Group	Tasks
2A	1 2 3 4
2B	1 2 3
1B	1 2 5
1A	1 2 5
Assignment	**The tasks not covered in your lab**

__Task 1__: Basic Implementations

Implement the basic operations of a Binary Tree. Your program should include the following functions:

1. **Insert**: Insert the given numbers using the ‘**Binary Search Tree (BST)**’ policy. The first line of input will contain N, followed by N integers to be inserted in the BST. Do not write a recursive function.
2. **Print_tree**: After insertion, print the tree status using Inorder traversal. Note that the inorder traversal of a BST will always show the nodes in sorted order. (If not, there must be an error in the insertion algorithm.)
3. **Search**: Returns the node if it is present and prints its description. Otherwise, print ‘Not Found’.
4. **Height**: Given a value, search it and return the height of that node (if present). The height of a leaf node is 0. Write the insertion procedure in such a way that it considers height as an attribute for each node and updates height during insertion. Do not write the recursive height function!!
5. **Before_after**: Given the value of a node, you have to print the node that will appear before and after that node (if the numbers were organized in sorted order, but of course, don’t use any sorting algorithm!).

Input	**Output**	**Explanation**

8
100 150 50 125 135
25 40 200
	25 40 50 100 125 135 150 200	**Note**: The tree looks like
		`        100`
		`       /   \`
		`     50     150`
		`    /      /   \`
		`  25     125   200`
		`    \       \`
		`     40      135`

3 125	Present Parent(150), Left(null), Right(135)	(search 3)
3 140	Not Present	(search 1)
3 40	Present Parent(25), Left(null), Right(null)	
4 100	3	
4 125	1	
4 50	2	
5 40	25 50	
5 100	50 **125**	
5 135	125 150	Just checking subtrees is not enough. Sometimes there value may reside in ancestors as well !
5 200	150 null	
5 25	null 40	

<u>Task 2</u>: Tree Traversal Algorithms

Consider the BST given in Task 1 and write the following functions:

1. Inorder
2. Preorder
3. Postorder
4. Level_order

Insert the numbers using the **'Binary Search Tree (BST)'** policy. The first line of input will contain N, followed by N integers to be inserted in the BST.

The output must be as shown in the table.

Input	**Output**

8
100 150 50 125
135 25 40 200	

1	Inorder: 25(50) 40(25) 50(100) 100(null) 125(150) 135(125) 150(100) 200(150)
2	Preorder: 100(null) 50(100) 25(50) 40(25) 150(100) 125(150) 135(125) 200(150)
3	Postorder: 40(25) 25(50) 50(100) 135(125) 125(150) 200(150) 150(100) 100(null)
4	Level 1: 100(null) Level 2: 50(100) 150(100) Level 3: 25(50) 125(150) 200(150) Level 4: 40(25) 135(125)

Note:

- You need to modify the Level-order Traversal algorithm to print the Level ID of each node.
- Show a simulation of your code in your notebook.
- Do not write a recursive function for **insertion**. But can use recursion for the traversal algorithms.

Task 3: Tree Diameter

Given a Binary Search Tree, your task is to find the diameter. The diameter of a tree is the number of nodes in the longest path between any two leaf nodes. The longest path may not be through the root node of the tree. The longest path may be entirely in the left or right subtree, or it may pass through the root node.

The insertion process in the Binary Tree works as follows-

> **Insert:**
> Assuming each node contains a unique value.
> Input starts with a number *N* (representing the number of nodes), followed by *N* integers in the next line that are to be inserted into the BST.

Sample Test Case(s)

Input

```
7
4 2 6 1 3 5 7
```

Output

```
5
```

Input

```
13
4 2 8 1 3 7 9 6 11 5 10 12 13
```

Output

```
8
```

Note:

In the 1st test case, the tree looks as follows:

```
      4
    /   \
   2     6
  / \   / \
 1   3 5   7
```

The diameter is the number of nodes in the path starting from node 1 to node 7.

In the 2nd test case, the tree looks as follows:

```
      4
    /   \
   2     8
  / \   / \
 1   3 7   9
      /     \
     6       11
    /       / \
   5       10  12
                \
                13
```

The diameter is the number of nodes in the path starting from node 5 to node 13.

Task 4: Searching for LCA

In a Binary Search Tree (BST), find the Lowest Common Ancestor (LCA) for two given nodes, u and v, with the assumption that both nodes exist in the BST. The LCA of two nodes in a tree is formally defined as the nearest shared ancestor of those nodes.

The insertion process in the Binary Tree works as follows-

> **Insert:**
> Assuming each node contains a unique value.
> Input starts with a number N (representing the number of nodes), followed by N integers in the next line that are to be inserted into the BST.

The next line of the input will be the number of queries q.
In each of the following q lines, there will be two given nodes, u_i and v_i.
Your task is to determine $LCA(u_i, v_i)$ in each query.

Sample Test Case(s)
Input
```
7
4 2 6 1 3 5 7
5
5 2
1 3
7 3
5 7
6 2
```

Output
```
4 2 4 6 4
```

Input
```
13
4 2 8 1 3 7 9 6 11 5 10 12 13
6
9 11
11 7
1 13
10 13
1 3
13 3
```

Output
```
9 8 4 11 2 4
```

Note:
Refer to the trees in Task 3.

Task 5: Print paths between two nodes

A set of numbers is stored in a Balanced BST. Given two keys, your task is to print the path between the two nodes and the distance (number of nodes comprising that path).

At first, the numbers will be given as input. Then, there will be a series of queries. Each query contains two numbers, x & y (x<y). Print the path to reach ‘y’ starting from ‘x’. Then, count the number of nodes in that path (including x,y).

Sample Input	**Sample Output**
12 7 9 20 15 27 5 3 6 11 -1	Status: 3(0) 5(1) 6(0) 7(2) **9**(1) 11(0) 12(3) 15(0) 20(1) 27(0)
5 11	5 7 9 11 4
3 12	3 5 7 12 4
3 7	3 5 7
3 6	3 5 6
6 7	**6 5 7**
12 15	12 20 15
9 12	9 7 12
6 11	6 5 7 9 11 5
7 9	7 9 2
7 11	7 9 11 3
3 15	3 5 7 12 20 15 6

```
              12(3)
            /       \
         7(2)        20(1)
        /  \         /    \
     5(1)   9(1)  15(0)   27(0)
    /  \      \
3(0)    6(0)   11(0)
```

Hint:

- The nodes in the path DO NOT need to be in sorted order.
- Utilize the idea of the Lowest common ancestor (LCA).
- Make sure all the test cases are working. If you cannot print in the given order, you should mention it.