CSE 4304-Data Structures Lab. Winter 23-24
Batch: CSE 22
Date: October 09, 2024
Target Group: All
Topic: Binary Trees, Binary Search Trees

Instructions:

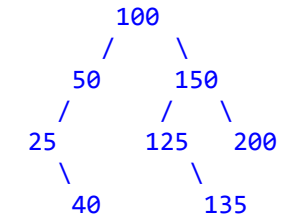
- Regardless of how you finish the lab tasks, you must submit the solutions in Google Classroom. In case I forget to upload the tasks there, CR should contact me. The deadline will always be 11:59 PM on the day the lab took place.
- Task naming format: fullID_T01L01_2A.c/cpp
- If you find any issues in the problem description/test cases, comment in the Google Classroom.
- If you find any tricky test cases that I didn't include but that others might forget to handle, please comment! I'll be happy to add them.
- Use appropriate comments in your code. This will help you to recall the solution in the future easily.
- 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.
- You are allowed to use the STL stack unless it's specifically mentioned to use manual functions.

Group	**Tasks**
2A	1 2 3
1B	1 2 3
1A	4 5 6
2B	4 5 6
Assignments	2A/1B: 1A/2B:

Task 1: Basic Implementations

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

1. **Insert**: Insert the given numbers maintaining the properties of 'Binary Search Tree (BST)'. The first line of input will contain N, followed by N integers to be inserted in the BST. Do not write a recursive function for insertion.
2. **Print_tree**: After insertion, print the 'status of the tree' 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 implementation.)
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 during inorder traversal (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 
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	

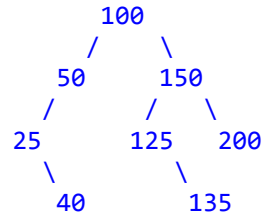
Task 2: Tree Traversal Algorithms

Consider the Binary Search Tree 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)' insertion 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. Print the parent of each node beside them. Note that, you have to store the parent of each node during insertion.

Input	**Output**
8 100 150 50 125 135 25 40 200	**Clarification**: The tree looks like (Not part of output) 
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.

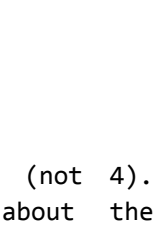
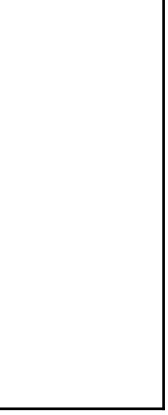
Task 3: Searching for LCA

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.

Input	**Output**	**Explanation**
7 4 2 6 1 3 5 7 5 5 2 1 3 7 3 5 7 6 2	 4 2 4 6 4	 Note: the LCA of 5,7 is 6 (not 4). Because LCA doesn't care about the magnitude, rather checks which common ancestor is the nearest one!
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	 9 8 4 11 2 4	

Task 4: Basic Implementations-II

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

1. **Insert**: Insert the given numbers maintaining the properties of 'Binary Search Tree (BST)'. The first line of input will contain N integers to be inserted in the BST (until you get -1). Do not write a recursive function for insertion.
2. **Print**: After each insertion, print the 'status of the tree' using Inorder traversal. Print the height of each node in the bracket. Note that the inorder traversal of a BST will always show the nodes in sorted order. Otherwise, there must be an error in the implementation.
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, you have to print the node that will appear before and after that node during inorder traversal (don't use any sorting algorithm!).
6. **Max_min**: Print the maximum and minimum value of the tree.

Input	**Output**	**Explanation**
8 100 150 50 125 135 25 40 200 -1	100(0) 100(1) 150(0) 50(0) 100(1) 150(0) 50(0) 100(2) 125(0) 150(1) 50(0) 100(3) 125(1) 135(0) 150(2) 25(0) 50(1) 100(3) 125(1) 135(0) 150(2) 25(1) 40(0) 50(2) 100(3) 125(1) 135(0) 150(2) 25(1) 40(0) 50(2) 100(3) 125(1) 135(0) 150(2) 200(0)	**Note**: 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	Just checking subtrees is not enough. Sometimes their value may reside in ancestors as well !
5 100	50 **125**	
5 135	125 150	
5 200	150 null	
5 25	null 40	
6	25 200	

Task 5: Tree Traversal Algorithms

Consider the Binary Search Tree given in Task 1 and write the following functions:

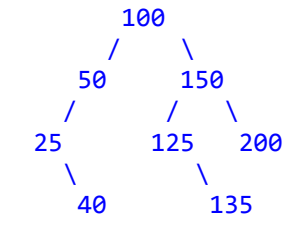
1. Inorder
2. Preorder
3. Postorder
4. Level_order

Insert the numbers using the 'Binary Search Tree (BST)' insertion 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 following table.

Type of Traversal	Output
Inorder	Node(parent)
Preorder	Node(left child)
Postorder	Node(right child)
Levelorder	Node(height)

Test Cases :

Input	Output
8 100 150 50 125 135 25 40 200	**Clarification**: The tree looks like (Not part of output) 
1	Inorder: 25(50) 40(25) 50(100) 100(null) 125(150) 135(125) 150(100) 200(150)
2	Preorder: 100(50) 50(25) 25(null) 40(null) 150(125) 125(null) 135(null) 200(null)
3	Postorder: 40(null) 25(40) 50(null) 135(null) 125(135) 200(null) 150(200) 100(150)
4	100(3) 50(2) 150(2) 25(1) 125(1) 200(1) 40(25) 135(125)

Note:

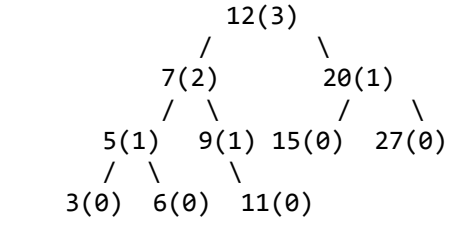
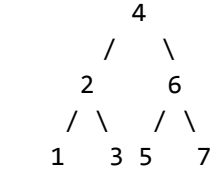
- You need to modify the Level-order Traversal algorithm to print the Level ID of each node.
- Show a simulation of your code in your notebook.
- Do not write a recursive function for **insertion**. But can use recursion for the traversal algorithms.

Task 6: 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 along with the length of the path (number of nodes comprising that path).

At first, values will be inserted inside BST as long as -1 isn't given. Once the insertion is over, print the status of the tree using inorder traversal. Each node will show its height information in brackets. Your insertion algorithm should take care of the height values and store with each node. Don't call the recursive height function for each node!

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**	**Clarification**
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)	 • Utilize the idea of the Lowest Common Ancestor (LCA). The LCA of two nodes in a tree is formally defined as the nearest shared ancestor of those nodes.  The LCA of 5,7 is 6 (not 4). Because LCA doesn't care about the magnitude, rather checks which common ancestor is the nearest one! • You don't print the path in sorted order. • Make sure all the test cases are working. If you cannot print in the given order, you should mention it.
5 11	5 7 9 11 4	
3 12	3 5 7 12 4	
3 7	3 5 7 3	
3 6	3 5 6 3	
6 7	6 5 7 3	
12 15	12 20 15 3	
9 12	9 7 12 3	
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	