Assignment # 5

Binary Search Trees

Submission Dead Line: 09/11/2022

Question: Write a C++ class BST (binary search tree) with the following functions:

1. **Constructor, Destructor**
2. **Search**
3. **Insert**
4. **Delete**
5. **Print,** (Inorder, Postorder and Preorder), by using both recursive and iterative algorithms and level order by using iterative algorithm.

Note: The time complexity of search, insert and delete should be O (log N), and that of destructor and print should be O (N).

6. **Update-Key**
   It will take two keys as input key1 and key2 and finds the node with key1, then changes it to key2. This may disturb the BST property (of having smaller things to the left and bigger to the right); this function should make appropriate changes in tree so that it becomes BST again.
7. **Overloaded Assignment Operator:** This function should make a copy of the whole tree (deep copy). Time complexity should be O (N).
8. **Equality operator (==)** that compares two BST's. It returns true only when both trees have a same structure and same data values. Complexity should be O (N).
9. **Count_Nodes(Node * root)** Write a non-recursive function to count the total nodes in the tree.
10. **Mirror_BST(Node * root)**
    Changes the tree to a mirror image of itself. So, all right sub trees get exchanged by left sub trees for all the nodes (not just the root node).
11. **Find_width(Node * root)**
    Returns the number of nodes in the fullest level of the tree (the maximum number of nodes in any level).
12. **PathSum(vector<T> & sums)**
    Determines the sums of all nodes in each path of the tree and return that in the vector sums. The total number of sums is equal to the number of unique paths in the tree, in other words, the number of leaves in the tree.
13. **Make_Skew(Node * root)**
    Converts the existing BST into a completely skewed tree with the smallest node at the top. This function must not allocate any new nodes; change the structure of the existing tree to completely skewed one.
14. **bool IsBST(Node * root)**
    This function will take root of a binary tree as input and will check recursively whether the tree is a valid BST or not.
15. **void Common_ Ancestors (Node * n1, Node * n2)**
    This function will take two nodes of BST as input and will find and print the common ancestors of both nodes.