National University of Computer and Emerging Sciences

Laboratory Manual

for

Data Structures Lab

Course Instructor	Mr. Saad Farooq
Lab Instructor(s)	Hasnain Iqbal Usama Hassan
Section	CS-E
Date	6-Dec-2021
Semester	Fall 2021

Department of Computer Science

Objectives:

In this lab, students will practice:

1. Insert operation on AVL Trees
2. Delete operation on AVL Trees

Question 1: For this lab, use your BST code implemented in the previous lab.In this lab, you have to create a height-balanced tree class named "AVL". Inherit the BST class publicly in your new "AVL" class. You can add "height" variable in your existing TNode struct implementation.

Implement the following methods for AVL class:

a. A default Constructor which calls the default constructor of base class (BST class).

b. Override the insert method of base class (BST class) in your AVL class, so that the AVL tree remains height-balanced after insertion of a new node.

c. Override the delete method of base class (BST class) in your AVL class, so that the AVL tree remains height-balanced after deletion of a node.

d. A function "height" which returns the height of the tree. `int height()const`

e. A function "search" which returns a pointer to the value of the node containing the required key b

Question 2: Now run the following main program.

```
int main()
{
	AVL<int, int> tree;


	for (int i = 1; i <= 100; i++)
		tree.insert(i, i);

	for (int i = -1; i >= -100; i--)
		tree.insert(i, i);

	for (int i = 150; i > 100; i--)
		tree.insert(i, i);

	for (int i = -150; i < -100; i++)
		tree.insert(i, i);

	for (int i = 150; i > 100; i--)
		tree.delete(i);
```

```
        tree.inorderPrintKeys();
        cout << endl << endl;
        cout <<"Tree Height: "<< tree.height() << endl;

        int *val = tree.search(-100);

        if (val != nullptr)
        {
                cout <<"Key= -100 found"<< endl;
        }

        val = tree.search(-151);
        if (val == nullptr)
        {
                cout <<"Key= -151 not found"<< endl;
        }

        system("pause");
}
```

Question 3: Provide the implementation of following functions in your BST class.

1. Add a member function **printCommonAncestors** which is passed two keys (k1 and k2) as parmeters. The function then prints the keys of all ancestor nodes that are common between node n1 (containing key k1) and node n2 (containing key k2).
   void printCommonAncestors(K k1, K k2) const;

2. Add a member function **getTreeWidth** which returns the count of number of nodes of that level that has the maximum number of nodes. **int getTreeWidth() const;**

3. Add a member function **kthMaxKey** which is passed a positive integer (let's call it K) as a parameter. The function then finds the kth maximum key from the binary search tree. If the number of nodes is less than k, then throw an exception**. K kthMaxKey(int const key) const;**

4. Add a member function is **isCompleteBST** which returns true if the binary search tree is a complete binary search tree. A complete binary search tree is a binary search tree in which all levels, except possibly the last level, are completely filled (they form a perfect binary tree); and the nodes in the last level are as far left as possible. **bool isCompleteBST() const;**