**National University of Computer and Emerging Sciences**



**Laboratory Manual**

*for*

# Data Structures Lab

|  |  |
| --- | --- |
| Course Instructor | Mr. Uzair Naqvi |
| Lab Instructor | Mr. Durraiz Waseem |
| Semester | Fall 2024 |

# Department of Computer Science

FAST-NU, Lahore, Pakistan

**Objectives:**

In this lab, students will practice AVL and Heap Trees

**Problem 1: AVL Implementation:**

class AVL; class Node { friend class AVL; private:

int key; // data to be stored in a node

int height; // To store the height of each node with it

Node\* left;

Node\* right;

}; class AVL { private: Node\* root; int avlHeight (Node\*); int max (int a, int b);

int getBalance(Node\*); // get balance factor of a node

Node\* SingleRightRotation(Node\*); //LL case

Node\* SingleLeftRotation(Node\*); //RR case

Node\*LeftRightRotation(Node\*); //LR case

Node\*RightLeftRotation(Node\*); //RL case

Node\* insert (int data, Node\*); Node\* delete (int data, Node\*); void display\_inorder(Node\*);

public:

// driver functions for insert, delete, display

};

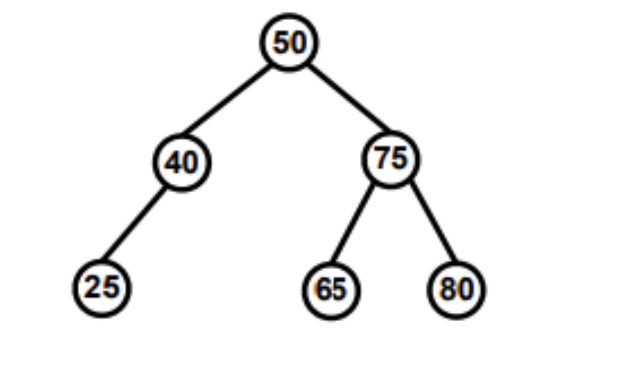
**Implement a template-based AVL tree and create the functions described below:**

1. Create a function ‘**insert()’** to insert a node into AVL keeping in view that the tree remains balanced after insertion. For this task, you need to compute the **Balancing Factor (BF)** of each node.
   * **Node \*insert(Node \*node, int key)**
   * **Int balanceFactor(Node \*n)**
2. Implement deletion of a given node from the AVL Tree. This function should delete the node with the key from the AVL tree rooted at node **‘n’**. Think about its logic now and don’t forget to balance your tree afterwards just like you did in insertion.
   * **Node\* delete (int data, Node\* n);**
3. Implement functions for four types of rotation namely **RR, LL, LR,** and **RL.**
   * **Node \*RotateRight(Node \*y)**
   * **Node \*RotateLeft(Node \*x)**
4. Create a bool function **‘isAVL()’** which takes a tree as an argument and tells whether the tree is an AVL or not.
5. A **‘search’** function to search an element in the tree **Bool search(int key)**
6. A **recursive** function ‘**findmin()’** that finds the minimum element using recursion.
7. A **‘print’** function to print the **inorder, preorder,** and **postorder** traversal.
8. **Print all possible paths from root to leaf.** Implement a public member function of the

AVLTree class which prints all the root-to-leaf paths of a given AVLTree. The prototype of your function can be

**● void printAllPaths ()**

For example, if we call this function on the following Tree:

For example, if we call this function on the following Tree:

It should print the following paths (in this order):

**50 -> 40 -> 25**

**50 -> 75 -> 65**

**50 -> 75 -> 80**

**Hint 1:** You will need to implement a private recursive helper function.

**● void printAllPathsRecursive(Node\*curr, int path[], int pathLength)**

**Hint 2**: You can assume that the maximum length of a root-to-leaf path will be 100.

**Note: Do NOT use any global or static variables in your implementation.**

**Problem 2: Heap Trees:   
  
1**. In this task, you are going to implement a class for Max Heap. Each node of this Max Heap will contain the roll number, and CGPA of a student. The heap will be organized on the basis of students’ CGPAs i.e. the student having the maximum CGPA will be at the root of the heap. The class definitions will look like above.

class StudentMaxHeap;

class Student {

friend class StudentMaxHeap;

private: int rollNo; // Student’s roll number

double cgpa; // Student’s CGPA };

class StudentMaxHeap {

private: Student\* st; // Array of students which will be arranged like a MaxHeap

int currSize; // Current number of students present in the heap

int maxSize; // Maximum no. of students that can be present in heap

public: StudentMaxHeap (int size); // Constructor

~StudentMaxHeap(); // Destructor

bool isEmpty(); // Checks whether the heap is empty or not bool

isFull(); // Checks whether the heap is full or not

};

**2.** Implement a public member function of the StudentMaxHeap class which inserts the record of a new student (with the given roll number and CGPA) in the Max Heap. The prototype of your function should be:   
**bool insert (int rollNo, double cgpa);**   
This function should return true if the record was successfully inserted in the heap and it should return false otherwise. The worst-case time complexity of this function should be (𝒍𝒈 𝒏). If two students have the same CGPA then their records should be stored in a way such that at the time of removal if two (or more) students have the same highest CGPA then the student with smaller roll number should be removed before the students with larger roll numbers.

**3.** Now, implement a public member function to remove that student’s record from the Max Heap which has the highest CGPA. The prototype of your function should be:   
**bool removeBestStudent (int& rollNo, double& cgpa);**   
Before removing the student’s record, this function will store the roll number and CGPA of the removed student in its two reference parameters. It should return true if the removal was successful and it should return false otherwise. The worst-case time complexity of this function should also be (𝒍𝒈 𝒏).