**Data Structures and Algorithms Lab**

**Project**

****

**Name:** **Hamza Ali Khan**

**Rollno: 01-134241-059**

**BSCS 3A**

**Instructor: Mohsin Javaid Butt**

**Department of Computer Sciences**

BAHRIA UNIVERSITY, ISLAMABAD

**Task**

Upload the completed Project to GitHub and provide the link to the repository.

**Project Title:**

***Telephone Directory Using AVL Trees***

**Course Details:**

* **Course Code:** CEN212L
* **Course Title:** Data Structures and Algorithms Lab
* **Instructor Name:** *[Your Instructor’s Name]*
* **Submitted By:** *[Your Full Name]*
* **Registration Number:** *[Your Registration Number]*
* **Semester:** 4th
* **Program:** BS Computer Engineering
* **Date of Submission:** *[Insert Date]*

**1. Objective**

The objective of this project is to design and implement a **Telephone Directory** using an efficient and self-balancing data structure, specifically an **AVL Tree**, to ensure fast operations such as insertion, deletion, searching, and updating contacts. The system simulates real-world telephone directory features and includes persistent file storage.

**2. Tools and Technologies Used**

* **Programming Language:** C++
* **Compiler:** GCC (via Code::Blocks/Dev C++/Visual Studio)
* **Data Structure Used:** AVL Tree (self-balancing Binary Search Tree)
* **File Handling:** Text files for persistent data storage

**3. Features of the Project**

The Telephone Directory includes the following key features:

* **Add Contact:** Insert a new contact with name, number, and group.
* **Search Contact by Name:** Efficient search using binary tree traversal.
* **Display Directory:** In-order traversal to display contacts in alphabetical order.
* **Update Contact:** Update the phone number of an existing contact.
* **Delete Contact:** Remove a contact while maintaining AVL balance.
* **Search by Group:** List all contacts belonging to a specific group.
* **Partial Name Search:** Find contacts whose names contain a specific substring.
* **Search by Number:** Locate a contact based on the phone number.
* **Clear Directory:** Delete all contacts from memory and file storage.

**4. Data Structure Design**

**AVL Tree Implementation**

Each contact is represented as a node in an AVL Tree. The node contains:

* **Name (string)**
* **Phone Number (string)**
* **Group (string)**
* **Height (int)** – to maintain AVL balance
* **Left & Right Pointers** – for the binary tree structure

The AVL Tree guarantees O(log n) time complexity for **insertion, deletion, and search operations**, making the directory efficient even with a large number of contacts.

**5. Core Algorithms**

**Insertion:**

* Insert a contact based on name (lexicographically).
* Update heights and balance the tree using left/right rotations.

**Deletion:**

* Standard BST deletion logic.
* Re-balance tree based on AVL rules after node removal.

**Search:**

* Binary search logic for name and number.
* Recursive partial match function using string::find().

**Update:**

* Locate the contact and modify the number.
* Tree rebalancing is done if needed.

**In-order Traversal:**

* To display contacts in sorted order (by name).

**File Operations:**

* **Persistent Storage:** Contacts are loaded from and written to a file named contacts.txt.
* **Temporary File:** Used for updating and deleting contact entries safely.

**6. File Structure**

* **contacts.txt** – Stores contacts in the format:  
  Name Number Group
* **temp.txt** – Temporary file used for safe file updates and deletions.

**7. Sample Outputs**

**1. Add Contact**

**Enter name: Ali**

**Enter number: 03001234567**

**Enter group: Family**

**Contact added: Ali - 03001234567 (Group: Family)**

**3. Display Directory**

**Telephone Directory:**

**Ali - 03001234567 (Group: Family)**

**7. Partial Name Search**

**Enter partial name to search: li**

**Partial matches found:**

**Ali – 03001234567**

**8. Conclusion**

**This project successfully demonstrates the use of AVL Trees in implementing a dynamic and efficient Telephone Directory. It highlights core data structure concepts such as self-balancing trees, recursive traversal, and file handling. The system is scalable, fast, and user-friendly.**

**9. Future Enhancements**

* **GUI implementation using Qt or C++/CLI.**
* **Import/export from CSV or JSON formats.**
* **Support for multiple numbers per contact.**
* **Data encryption for privacy and security.**

**10. References**

* ***Data Structures and Algorithm Analysis in C++* by Mark Allen Weiss**
* **GeeksforGeeks – AVL Tree Tutorials**
* **Cplusplus.com – File handling and string operations**