**Graph Algorithms Implementation - Shortest Path using BFS**

# Authors:

Muhammad Murtaza (Reg No. 2022402)

Hasan Abbas (Reg No. 2022204)

Hassan Ahmed (Reg No. 2022)

University: Ghulam Ishaq Khan Institute of Engineering Sciences and Technology (GIKI)

Course: Data Structures and Algorithms (CS221)

# Contributions:

## Muhammad Murtaza (Reg No. 2022402):

Function Implementation: Implemented the core functionality of the graph class, including methods for adding edges, printing the graph, and performing BFS.

Algorithm Design: Contributed to the design of the BFS algorithm for finding the shortest path.

Collaboration: Worked closely with team members to integrate different parts of the code seamlessly.

## Hasan Abbas (Reg No. 2022204):

Algorithm Refinement: Assisted in refining the BFS method and ensured its correctness.

Testing and Debugging: Conducted testing and debugging of the code to identify and fix issues.

Collaboration: Collaborated with team members to ensure the compatibility of different components.

**Hassan Ahmed (Reg No. 2022):**

File Reading Functionality: Implemented the readGraphFromFile method to read graph data from a file.

Testing and Insights: Assisted in testing the code and provided insights into potential improvements.

Collaboration: Worked collaboratively with team members to integrate file reading functionality with the existing code.

# Collaborative Efforts:

## Collaborative Logic Design:

The team collaboratively contributed to the overall design and logic of the code, pooling diverse perspectives and ideas to create a robust foundation for the project.

Through joint brainstorming sessions, each team member brought valuable insights that influenced the design decisions, ensuring a comprehensive and effective approach

Design Discussions: The team collectively participated in discussions to design the overall structure and logic of the code.

Code Integration: Collaborated to integrate individual components seamlessly, ensuring the functionality of the entire program.

Problem-Solving: Worked together to address challenges and solve issues encountered during the development process.

# Language Used: C++

Introduction: The project focuses on implementing graph algorithms, with a specific emphasis on finding the shortest path between two vertices using the breadth-first search (BFS) technique. The code is written in C++ due to its versatility, efficiency, and strong support for data structures and algorithms.

**Code Structure:**

**Class Definition (graph):**

adj: An unordered map to represent the adjacency list of the graph.

Methods:

addedge: Adds edges to the graph.

printGraph: Prints the graph with vertices and their connected vertices.

BFS: Implements breadth-first search to find the shortest path.

readGraphFromFile: Reads graph data from a file.

**Main Function:**

Creates an instance of the graph class.

Attempts to open a file named "cities.txt" to read graph data.

User input prompts to specify the starting and destination cities.

Calls the BFS method to find and print the shortest path.

Functions Overview:

**addedge Function:**

Adds an edge between two vertices with an optional weight.

If the graph is undirected, it adds the reverse edge as well.

**printGraph Function:**

Prints the graph with each vertex and its connected vertices along with edge weights.

**BFS Function:**

Performs breadth-first search to find the shortest path between two specified vertices.

Utilizes a queue, visited, parent, and distance maps to keep track of exploration progress.

**readGraphFromFile Function:**

Reads graph data from a file named "cities.txt."

Expects each line to contain two vertices and an edge weight.

Language Choice: The choice of C++ for this project is motivated by several factors:

Efficiency: C++ is known for its performance and efficiency, making it suitable for handling graph-related computations and algorithms.

Standard Template Library (STL): C++ STL provides robust support for data structures (like unordered\_map, queue, and stack) and algorithms, facilitating the implementation of graph-related operations.

Object-Oriented Design: C++ supports object-oriented programming, allowing for the creation of a well-organized and modular code structure through the use of classes.

Versatility: C++ is a versatile language, allowing low-level memory manipulation when needed, which can be crucial for optimizing certain algorithms

**Interface:**

