Minimum Spanning Tree (MST) Project

1 Introduction

This report presents the implementation of two fundamental algorithms—**Prim's** and **Kruskal's**—to find the **Minimum Spanning Tree (MST)** of a graph. The MST is a subset of edges that connects all vertices in a graph with the minimum possible total edge weight and no cycles, ensuring a connected and optimal structure.

The program utilizes a **priority queue** for Prim's algorithm and a **Union Find** data structure for Kruskal's algorithm. These algorithms are widely used in network design, including communication networks and circuit design, where minimizing connection costs is critical. The graph is represented using an adjacency matrix and an edge list, providing flexibility for both algorithms. The report details the design, execution, and performance analysis of both approaches, highlighting their efficiency and practical applications.

2 File Structure

The following files are included in this project:

- **GraphADT.h** and **GraphADT.cpp**: Implements the Graph Abstract Data Type (ADT), which represents the graph using an adjacency matrix and an edge list. It provides functions for graph creation, edge management, and memory handling.
- **PriorityQueue.h** and **PriorityQueue.cpp**: Implements a Priority Queue (min-heap) for efficient edge selection in Prim's algorithm. The queue supports insertions, deletions, and key updates while maintaining the heap property.
- **Prims.h** and **Prims.cpp**: Contains the function for Prim's algorithm, which takes a Graph instance as input and computes the Minimum Spanning Tree (MST) using a greedy approach. It relies on the priority queue for efficient edge selection.
- **UnionFind.h** and **UnionFind.cpp**: Implements the Union-Find (disjoint set) data structure, which is used for efficient cycle detection in Kruskal's algorithm. It supports union by rank and path compression optimizations.
- **Kruskals.h** and **Kruskals.cpp**: Contains the function for Kruskal's algorithm, which takes a sorted list of edges and uses the DisjointSet to avoid cycles. It computes the MST by adding edges in increasing order of weight.
- **main.cpp**: The top-level program that takes as input a cost matrix, initializes a graph, and calls both Prim's and Kruskal's algorithms. The results, including the MST edges, total costs, and runtimes, are printed in the specified format.

3 Input Format

The input is read from a text file named `input.txt`. The file should contain only the adjacency matrix of the graph. Each row represents a node, and the columns represent the weights of edges to other nodes. For non-existent edges or no connection between nodes, use the value `-1` or `0`.

The following is an example of the contents of `input.txt`:

```
0 4 -1 -1 -1 5
4 0 8 -1 -1 -1
-1 8 0 2 -1 -1
-1 -1 2 0 1 -1
-1 -1 -1 1 0 1
5 -1 -1 -1 1 0
```

4 Setup, Compilation, and Running the Program

4.1 Requirements

To compile and run the project, you need:

- A C++ compiler supporting C++11 or later (e.g., `g++`).
- The Standard Library for C++.

4.2 Compilation

To compile all files, use the following command:

```
g++ main.cpp GraphADT.cpp PriorityQueue.cpp Kruskals.cpp Prims.cpp UnionFind.cpp
```

This command generates an executable named `a.exe`.

4.3 Running the Program

To run the program, use the following command:

```
./a
```

The program will:

- Run Prim's algorithm on the graph and print the MST, total cost, and runtime.
- Run Kruskal's algorithm on the graph and print the MST, total cost, and runtime.

5 Output Format

The output format includes:

- `MST via Prim's Algorithm`: A list of edges in the MST with their respective weights.
- `Total Cost of MST via Prim's Algorithm`: The sum of the weights of all edges in the MST.
- `Runtime of Prim's Algorithm`: The time taken to compute the MST.

- `MST via Kruskal's Algorithm`: A list of edges in the MST with their respective weights.
- `Total Cost of MST via Kruskal's Algorithm`: The sum of the weights of all edges in the MST.
- `Runtime of Kruskal's Algorithm`: The time taken to compute the MST.

6 Example

Suppose you have an input file named `input.txt` with the following content:

```
0 4 -1 -1 -1 5
4 0 8 -1 -1 -1
-1 8 0 2 -1 -1
-1 -1 2 0 1 -1
-1 -1 -1 1 0 1
5 -1 -1 -1 1 0
```

Run the following command:

./a

The output will display the MSTs generated by Prim's and Kruskal's algorithms, the total cost of each MST, and their respective runtimes. Sample output:

```
Prim's algorithm MST:
Total cost- 13
(2 - 1)
(3 - 4)
(4 - 5)
(5 - 6)
(6 - 1)
Prim's Algorithm running time: 0.00408 seconds

Kruskal's algorithm MST:
Total cost- 13
(4 - 5)
(5 - 6)
(3 - 4)
(1 - 2)
(1 - 6)
Kruskal's Algorithm running time: 0.0015055 seconds
```