**REPORT**

**Project 2: Graph Algorithms**

**Single-source shortest path algorithm and Minimum Spanning Tree (MST)**

**Submitted By**

Abdullah Al Raqibul Islam (ID# 801151189)

**Code Repository Overview**

I have used C++ as the programming language for this project. The project structure looks like this

//fig

**Problem 1: Single Source Shortest Path**

**Short description:** In this task, I have implemented Dijkstra’s algorithm for finding the shortest paths from a source node to all the other nodes of the graph. This algorithm will work on both directed and undirected weighted graph. The limitation of this algorithm is that, it will not work on graph contains negative cycle reachable from the source vertex.

Dijkstra’s algorithm applies greedy strategy to find shortest path from a single source node. Dijkstra’s algorithm use priority queue to decide (based on the already discovered path-cost) which node to be considered to relax the shortest path from the source node, is typically considered as a greedy approach. By nature, single source shortest path problem has satisfactory optimization substructure since if node A is connected to node B, node B is connected to node C, and the path must go through node A and node B to reach the destination node C, then the shortest path from node A to node B and the shortest path from node B to node C must be a part of the shortest path from node A to node C. So, the optimal answers from the subproblems do contribute to the optimal answer for the total problem.

**Data-structure:** In my solution,I have used priority queue from C++ Standard Template Library with custom comparator, where it will cost O(log n) to push/pop an item into/from it. To store graph, I used array of C++ vector, which acts as an adjacency list. I have used array to store other supporting data, i.e. graph node visit mark, distance of nodes from the sources, and shortest path of nodes from the source node, etc.

**Runtime analysis:**

**Sample I/O specification:** I have chosen the following directed and undirected weighted graphs to test my solution for this problem,

* Undirected complete graph (Fig 4)
* Directed line graph (Fig 1)
* Directed graph with negative edge but no negative cycle (Fig 2)
* Undirected random graph

|  |  |
| --- | --- |
|  |  |
| Fig 1: Directed line graph | Fig 2: Directed graph with negative edge, but no negative cycle |
|  |  |
| Fig 3: Undirected line graph | Fig 4: Undirected complete graph |

**Problem 2: Minimum Spanning Tree (MST)**

**Short description:** For this task, I have implemented Kruskal’s algorithm to find minimum spanning tree of an undirected graph. This algorithm will only work on undirected graph. The working principal of Kruskal’s algorithm is to choose the edge with minimum weight (greedy approach) that connects any two trees in the forest. Eventually it will become a single tree (only if it is a connected graph) which connects all the nodes of the graph.

**Data-structure:** I have used disjoint set data-structure to track the set-id of a node. Initially every node assigned to his own set-id. When we found an edge, which connects two nodes from two separate sets, we make a connection between them by applying Union operation. I applied two technique called **path-compression** in find and **union-by-rank** to reduce the time complexity of each operation done on disjoint set data-structure. This two techniques actually complement each other and make the amortized time complexity even smaller than O(log n).

To store edges, I used vector from C++ Standard Template Library (STL). To sort the edge list by weight, I use sort function provided by C++ standard library which requires O(n log(n)) in the worst case scenario.

**Runtime analysis:**

**Sample I/O specification:** I have chosen the following connected, undirected, weighted graphs to test my solution for this problem,

* Undirected complete graph (Fig 4)
* Undirected line graph (Fig 3)
* Undirected graph with multiple MSTs
* Undirected random graph

Here is the graphical demonstration of my sample input graphs,

**Test platform:**

* Processor: Intel(R) Xeon(R) CPU E5-2620 2.00GHz (12 Core)
* Linux version: 5.0.0-27-generic
* g++ version: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0

**References:**

* Online graph drawing tool: <https://csacademy.com/app/graph_editor/>