**REPORT**

**Project 2: Graph Algorithms (Single source shortest path & Minimum Spanning Tree)**

**Submitted By**

Jawad Chowdhury, (ID# 801135477)

**Overview of Algorithms (short description)**

Here, I basically used 2 different algorithms for 2 problems.

**Algorithm 1 (Dijkstra’s algorithm):**

The first problem was to find out the shortest path for a given graph with a source node. The graph might be directed or it can also be undirected. Here, the edges on the graph were non-negative. So for every node andif there is an weight then,

**.**

For this problem, I have implemented the Dijkstra’s algorithm which is basically one of the greedy algorithm. It basically starts with a cloud or set of vertices at initial only having the source node. Then it tries to execute edge relaxation which basically tries to minimize the adjacency node’s distance from the source based on current cloud or set of visited nodes. So, according to the algorithm,

In the end, the cloud ends up having all the vertices with the shortest path cost in that graph.

**Algorithm 2 (MST : Kruskal’s algorithm):**

The second problem was to find out the minimum spanning tree for an undirected graph.

For this problem, as per the instruction, I have implemented the Kruskal’s algorithm. This algorithm itself is a greedy algorithm also. Here, for each vertex of the graph, the algorithm considers a tree and the whole graph as a forest. Then it takes the edges according to the non-decreasing order and if the two vertices of that edge are from different tree than it takes that edge as a part of the MST and Union those edges.

In the end, we have the set of edges from the graph, which forms the MST.

**Data Structure Chosen**

Here, for the implementation, I have used the basic data structures such as Vertex, Edge and Graph.

For a Vertex v, I used to keep its own information like v.name, v.parent and v.distance\_from\_source which I utilized on the shortest path algorithm.

For an Edge e, I used to keep the information of the e.from\_vertex, e.to\_vertex and the weight of the edge as e.weight.

In the Graph g, I have all the information related to the graph like the list of vertices as g.vertices, the list of edges as g.edges.

Also we have a worker class P2, where I have put the extra information such as the type of the algorithm, the nature of the graph (directed or undirected), the source node etc.

Here, one thing is that, for extracting the min I haven’t implemented any heap or priority queue, rather used the default brute-force method, which will definitely increase the time but as for these samples, I don’t have a large dataset so I haven’t tried to reduce that part, just implemented the basic algorithms, but there is scope to improve the time complexity.

**Runtime of Code**

**Dijkstra’s algorithm**

As I have mentioned, there is scope to improve the time complexity with the help of an efficient data structure (heap or priority queue) where the extraction of the minimum distance from source is possible with a lesser time which will lead to a time complexity of this algorithm to:

**O((E+V) logV)O(E logV)**

but I have not used that in my implementation rather used the brute-force search for the minimum, and that leads to the runtime of my code to:

**O().**

**MST Kruskal’s algorithm**

For the kruskal’s algorithm, the runtime basically depends upon the sorting of the edges, and here for the runtime of the implemented it leads to :

**O(E logV)**

**Sample Input & Output**

**1. Shortest path:**

**Input 1 (dijkstra\_graph\_1.txt):**

6 10 U

A B 1

A C 2

B C 1

B D 3

B E 2

C D 1

C E 2

D E 4

D F 3

E F 3

A

**Output 1:**

Source Node: A

Node: A, Cost: 0, Path: A

Node: B, Cost: 1, Path: A -> B

Node: C, Cost: 2, Path: A -> C

Node: D, Cost: 3, Path: A -> C -> D

Node: E, Cost: 3, Path: A -> B -> E

Node: F, Cost: 6, Path: A -> C -> D -> F

**Input 2 (dijkstra\_graph\_2.txt):**

9 14 U

A B 4

A H 8

B H 11

B C 8

H I 7

H G 1

C I 2

I G 6

C D 7

C F 4

G F 2

D F 14

D E 9

E F 10

A

**Output 2:**

Source Node: A

Node: A, Cost: 0, Path: A

Node: B, Cost: 4, Path: A -> B

Node: H, Cost: 8, Path: A -> H

Node: C, Cost: 12, Path: A -> B -> C

Node: I, Cost: 14, Path: A -> B -> C -> I

Node: G, Cost: 9, Path: A -> H -> G

Node: D, Cost: 19, Path: A -> B -> C -> D

Node: F, Cost: 11, Path: A -> H -> G -> F

Node: E, Cost: 21, Path: A -> H -> G -> F -> E

**Input 3 (dijkstra\_graph\_3.txt):**

7 21 U

A B 2

A C 4

A D 5

A E 7

A F 9

A G 12

B C 7

B D 13

B E 16

B F 5

B G 3

C D 9

C E 11

C F 8

C G 7

D E 5

D F 3

D G 7

E F 9

E G 14

F G 19

A

**Output 3:**

Source Node: A

Node: A, Cost: 0, Path: A

Node: B, Cost: 2, Path: A -> B

Node: C, Cost: 4, Path: A -> C

Node: D, Cost: 5, Path: A -> D

Node: E, Cost: 7, Path: A -> E

Node: F, Cost: 7, Path: A -> B -> F

Node: G, Cost: 5, Path: A -> B -> G

**Input 4 (dijkstra\_graph\_4.txt):**

10 9 D

A B 2

B C 2

C D 2

D E 2

E F 2

F G 2

G H 2

H I 2

I J 2

A

**Output 4:**

Source Node: A

Node: A, Cost: 0, Path: A

Node: B, Cost: 2, Path: A -> B

Node: C, Cost: 4, Path: A -> B -> C

Node: D, Cost: 6, Path: A -> B -> C -> D

Node: E, Cost: 8, Path: A -> B -> C -> D -> E

Node: F, Cost: 10, Path: A -> B -> C -> D -> E -> F

Node: G, Cost: 12, Path: A -> B -> C -> D -> E -> F -> G

Node: H, Cost: 14, Path: A -> B -> C -> D -> E -> F -> G -> H

Node: I, Cost: 16, Path: A -> B -> C -> D -> E -> F -> G -> H -> I

Node: J, Cost: 18, Path: A -> B -> C -> D -> E -> F -> G -> H -> I -> J

**2. MST (Kruskal):**

**Input 1 (mst\_graph\_1.txt):**

6 10 U

A B 1

A C 2

B C 1

B D 3

B E 2

C D 1

C E 2

D E 4

D F 3

E F 3

**Output 1:**

A -> B

B -> C

C -> D

B -> E

D -> F

total cost: 8

**Input 2 (mst\_graph\_2.txt):**

9 14 U

A B 4

A H 8

B H 11

B C 8

H I 7

H G 1

C I 2

I G 6

C D 7

C F 4

G F 2

D F 14

D E 9

E F 10

**Output 2:**

H -> G

C -> I

G -> F

A -> B

C -> F

C -> D

A -> H

D -> E

total cost: 37

**Input 3 (mst\_graph\_3.txt):**

7 21 U

A B 2

A C 4

A D 5

A E 7

A F 9

A G 12

B C 7

B D 13

B E 16

B F 5

B G 3

C D 9

C E 11

C F 8

C G 7

D E 5

D F 3

D G 7

E F 9

E G 14

F G 19

**Output 3:**

A -> B

B -> G

D -> F

A -> C

A -> D

D -> E

total cost: 22

**Input 4 (mst\_graph\_4.txt):**

10 9 U

A B 2

B C 2

C D 2

D E 2

E F 2

F G 2

G H 2

H I 2

I J 2

**Output 4:**

A -> B

B -> C

C -> D

D -> E

E -> F

F -> G

G -> H

H -> I

I -> J

total cost: 18

**Instruction to Run Code**

The running procedure of my implementation is pretty much simple.

I have only one python script named as **project\_2.py.**

Here is this script,

I basically kept a list for the input file names for both shortest path implementation and MST Kruskal implementation ( i.e.

file\_list = [**'dijkstra\_graph\_1.txt'**, **'dijkstra\_graph\_2.txt'**, **'dijkstra\_graph\_3.txt'**, **'dijkstra\_graph\_4.txt'**]

)

So updating this list with the file name/s I want to run and running the script will do the work.