Group Members: Chaitanya Darade, Tanvi Rasam

Project 2: Graph Algorithms

Project Overview:

In this project, the following algorithms are implemented in Python language:

- Singles-source shortest path
- Minimum Spanning Tree (MST)

Data Structures Used:

1. Lists
2. Numpy Arrays
3. Numpy Matrix
4. Strings
5. Heap

Dijkstra's shortest path algorithm

Dijkstra's algorithm is an greedy algorithm for finding the shortest paths between nodes in a graph, for example, road networks. A common variant of it fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree. Thus, as it maintains a shortest-path-tree, Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree.

The idea is to traverse all vertices of graph using BFS and use a Min Heap to store the vertices for which shortest distance is not finalized yet

This is done by following data structures:

- Heap : min-heap, Every node of min heap contains vertex number and distance value of the vertex from the source ie. it is made up of dist and pos array. It is initialized as: dist(src) = 0; and for all other nodes v, dist(v) = ∞. This is done at the beginning because as the algorithm proceeds, the dist from the source to each node v in the graph will be recalculated and finalized when the shortest distance to v is found. Min Heap is used as a priority queue to get the minimum distance vertex from set of not yet included vertices.

- sptSet: list to keep track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty
- parent : list to display the final path. Keeps track of node which last updated that node
- Adjacency Matrix: to keep track of edges and weights

The core logic is- to Extract the vertex with minimum distance value node from Min Heap. Let the extracted vertex be u. Then for every adjacent vertex v of u, check if v is in Min Heap. If v is in Min Heap

and distance value is more than weight of u-v plus distance value of u, then update the distance value of v. Update the parent of every v to u. Repeat this till Min Heap is not empty.

Dijkstra's algorithm doesn't work for graphs with negative weight edges. For graphs with negative weight edges, Bellman–Ford algorithm can be used

decreaseKey - is used to update distance value in min heap and to perform heapify.

Thus, time complexity of operations like extract-min and decrease-key value is O(LogV) for Min Heap.

Complexity Analysis-

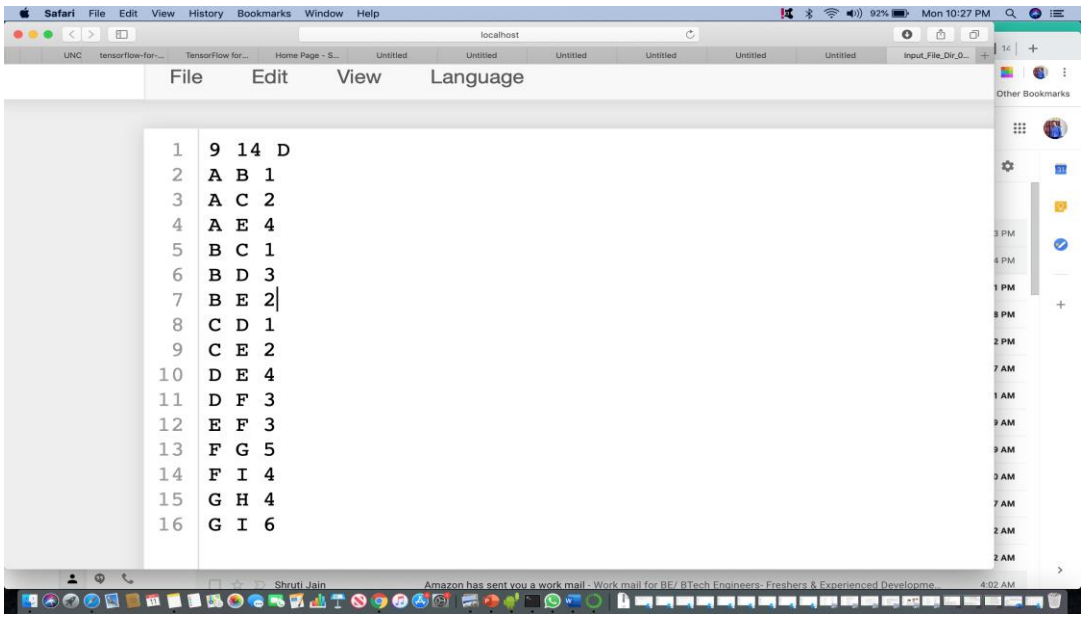
If Dijkstra implemented using an unsorted array. Thus it was O(1) for Insert and Update, but O(|V|) for Find/Delete minimum, resulting in O(|E| + |V|^2), but since |E| < |V|^2, we have O(|V|^2).

If a binary min-heap is used to implement, we have log(|v|) for all operations, resulting in O(|E|log|V| + |V|log|V|), but since it is reasonable to assume |E| > |V|, you have O(|E|log|V|).

Each extractMin() operation then takes time O (logV). There will be V such operations, thus O(VlogV). Each decreaseKey operation takes time O(logV), and there are at most E such operations, thus, O(ElogV). Hence, the total running time is therefore O((V+E) logV), which is **O(ElogV)** if all vertices are reachable from the source. (sparse graph)

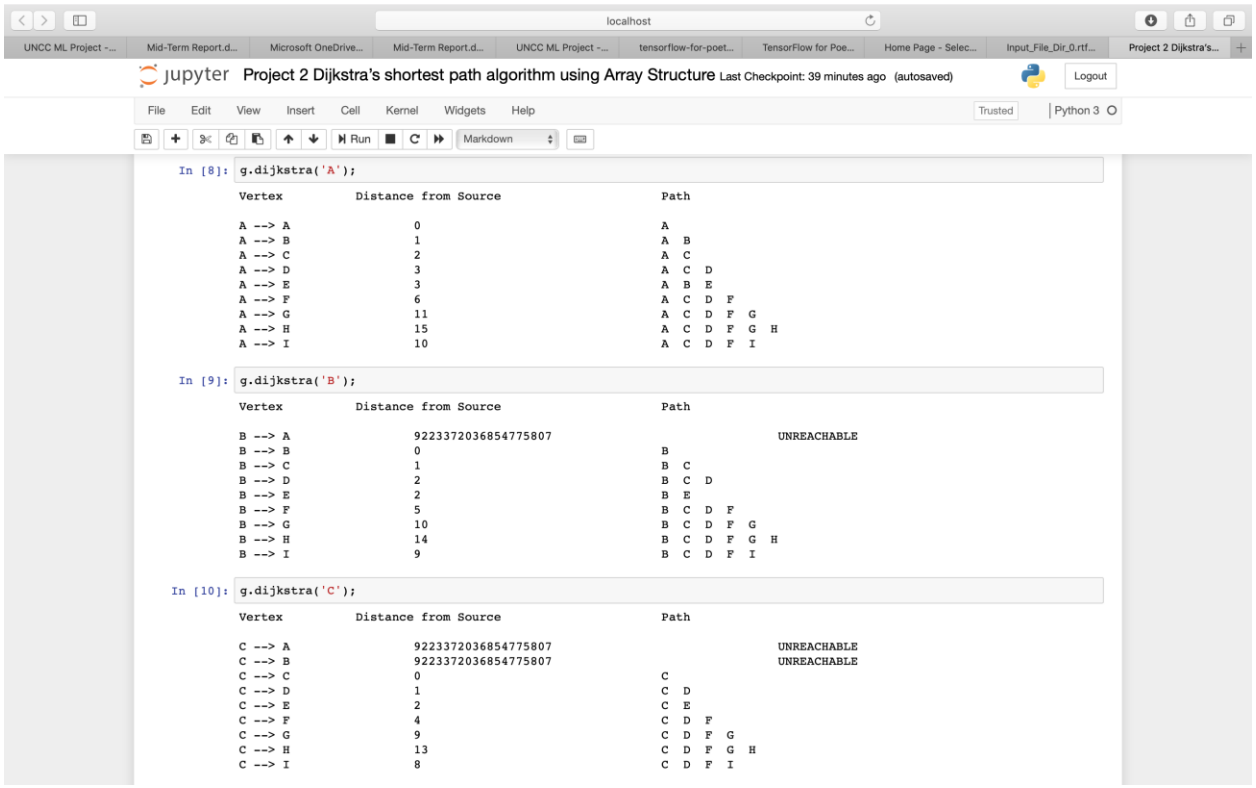
Input And Output

Input File 1 (Directed):

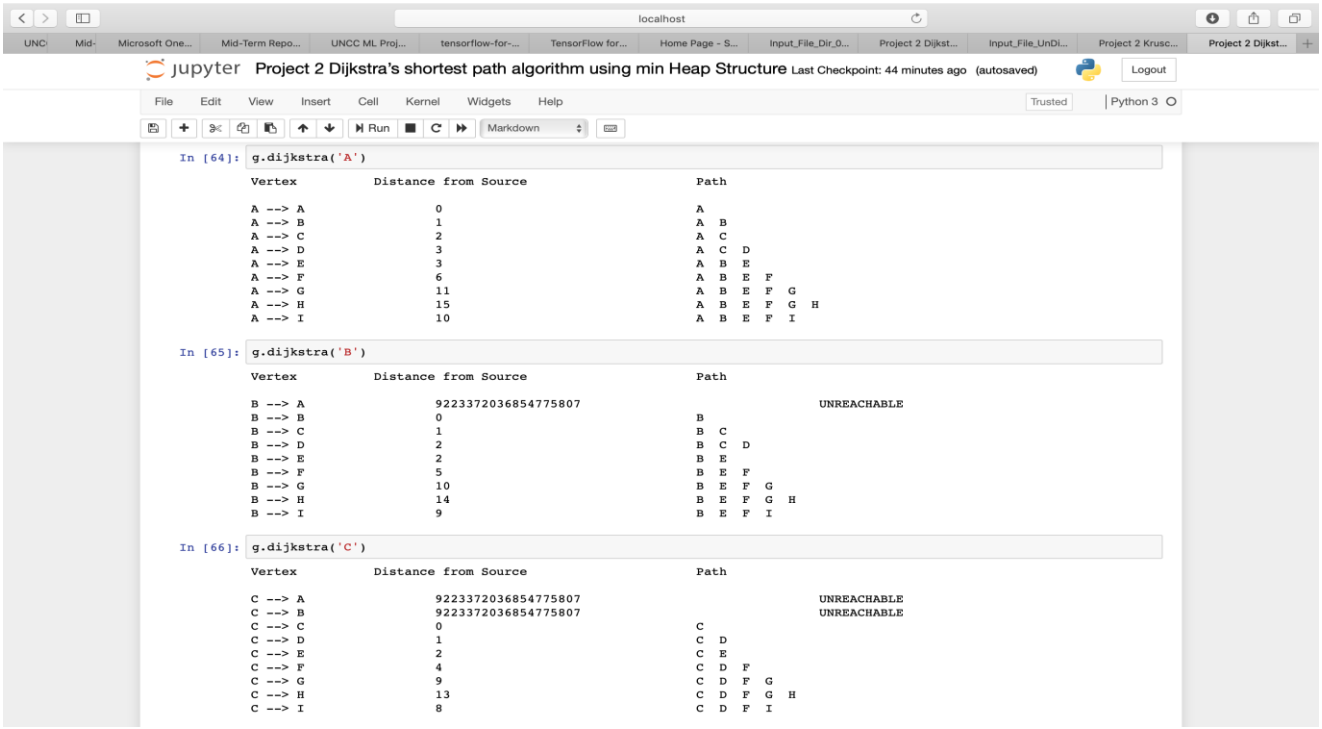


Output:

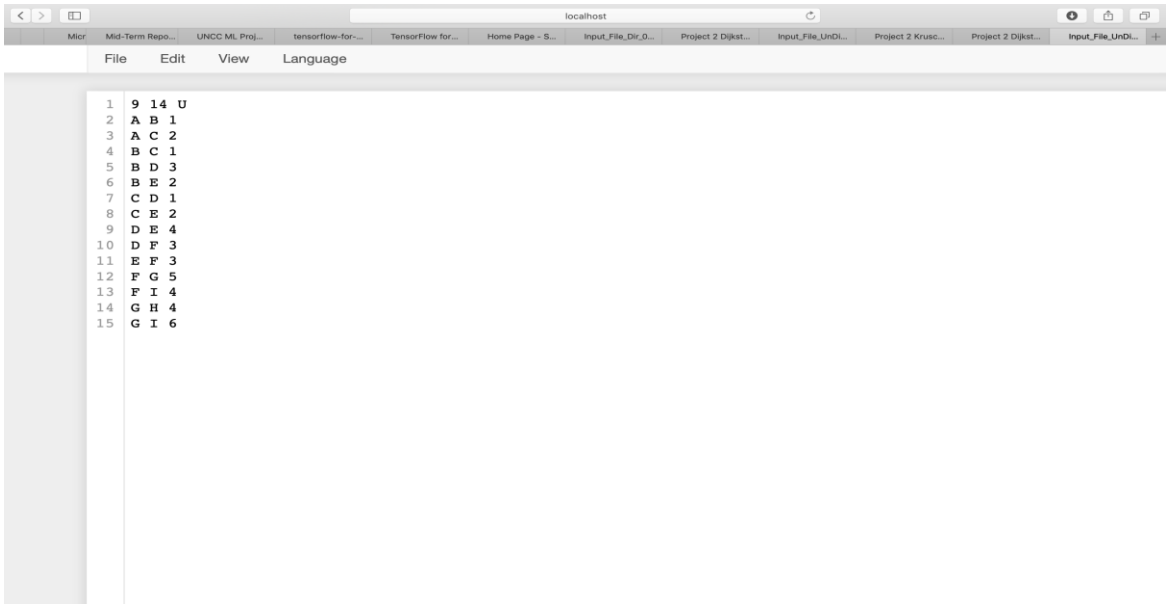
1. **Dijkstra's Algorithm using Array Data Structure:**
   Paths drawn from source nodes A, B and C



2. **Dijkstra's Algorithm using Array Min Heap Structure:**
   Paths drawn from source nodes A, B and C



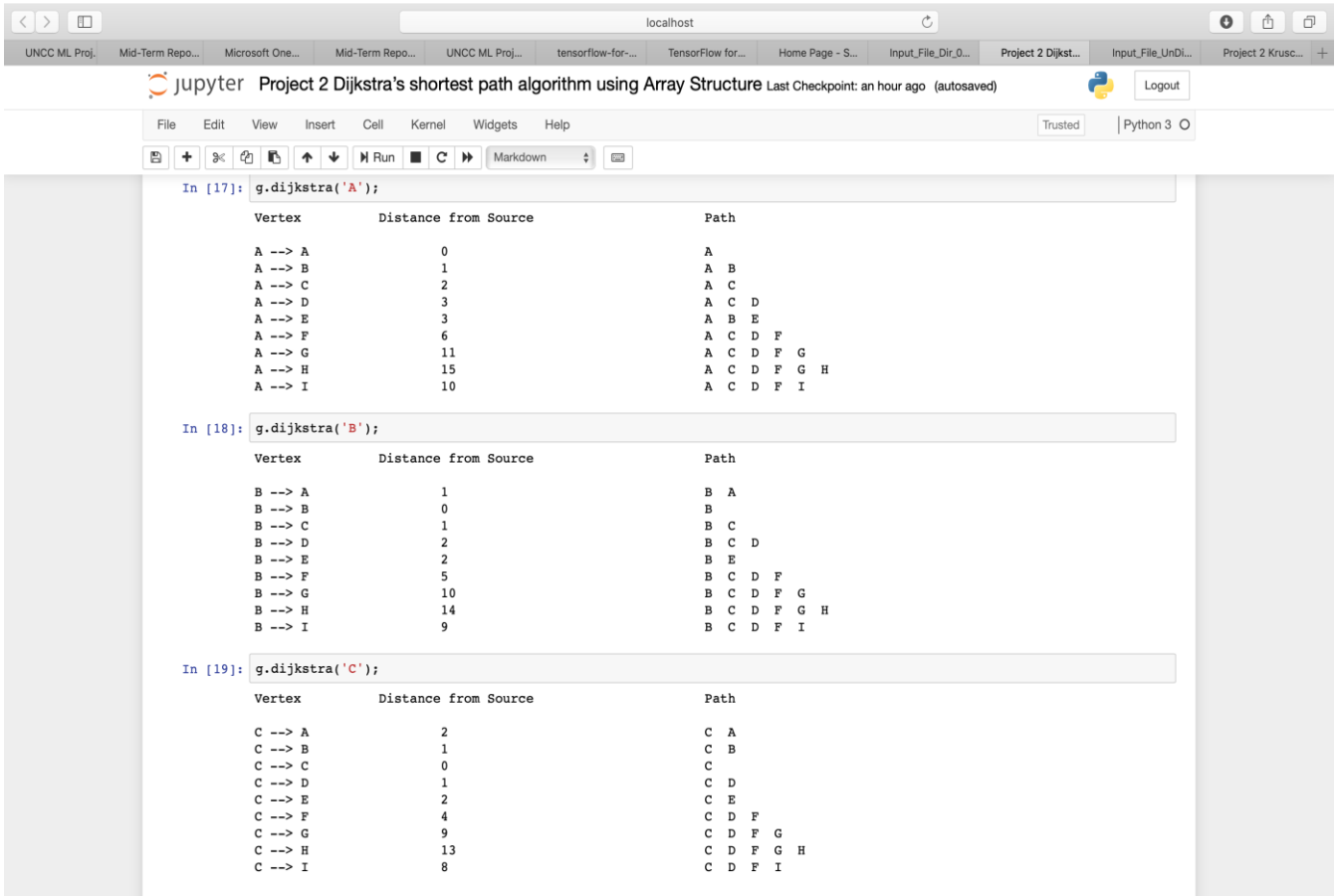
Input File 1 (UnDirectced):



Output:

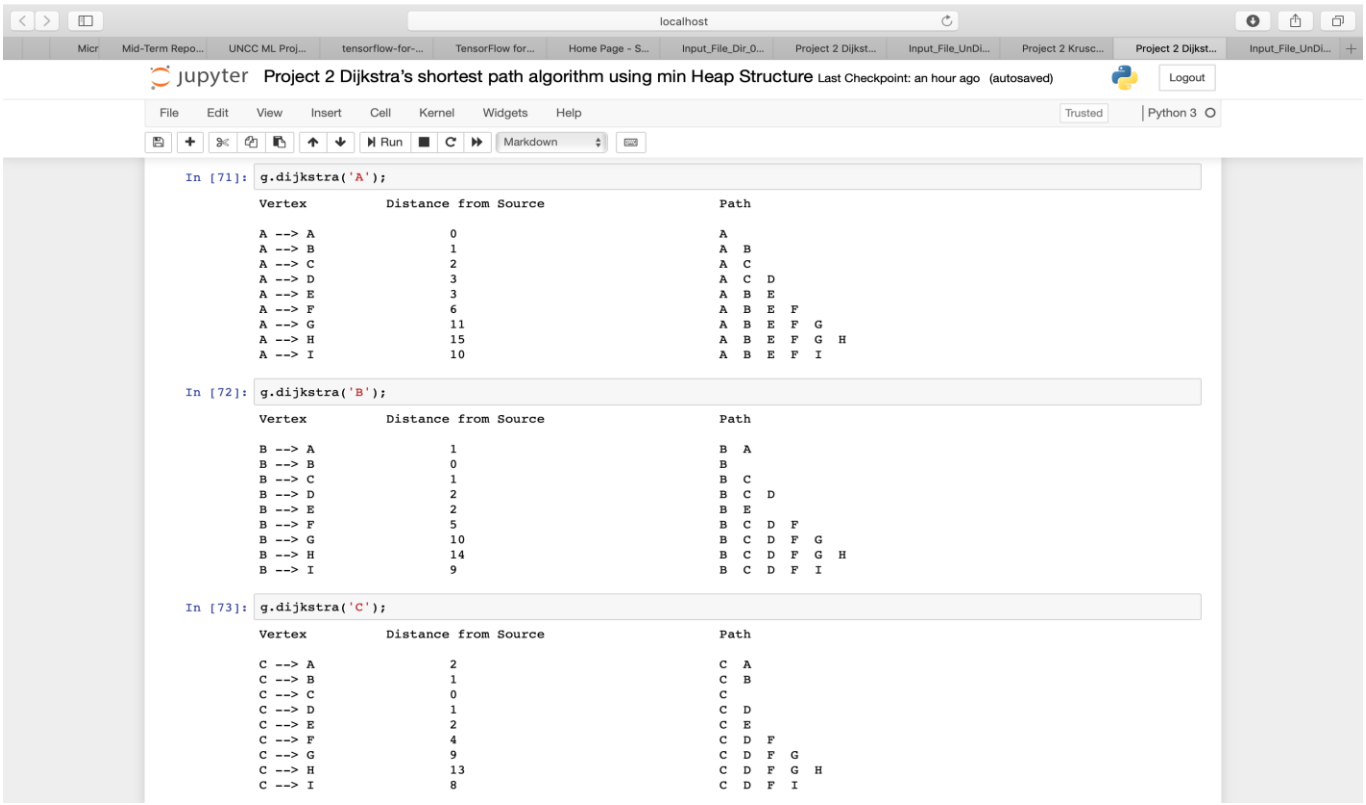
1. **Dijkstra's Algorithm using Array Data Structure**:

Paths drawn from source nodes A, B and C



2. Dijkstra's Algorithm using Array Min Heap Structure:

Paths drawn from source nodes A, B and C



Kruskal's Minimum Spanning Tree Algorithm

Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree. A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph. The algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

The flow is- Sort all the edges in non-decreasing order of their weight. Pick the smallest edge . Check if it forms a cycle with the spanning tree formed so far using uses Union-Find algorithm. If cycle is not formed, include this edge. Else, discard it. Repeat step 2 until there are (V-1) edges in the spanning tree.

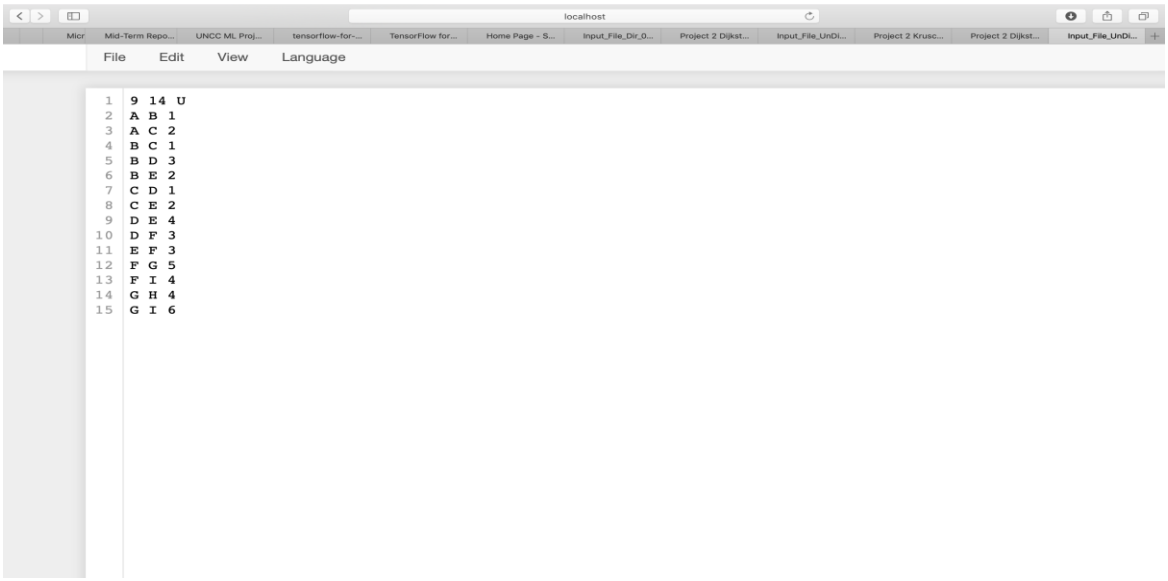
Data Structures used- Adjacency Matrix, List, Arrays

Time Complexity:

Sorting of edges takes O(ELogE) time using heap based priority queue. After sorting, we iterate through all edges and apply find-union algorithm. The find and union operations can take atmost O(LogV) time. So overall complexity is O(ELogE + ELogV) time. The value of E can be atmost $O(V^2)$, so O(LogV) and **O(LogE)** are same. Therefore, overall time complexity is O(ElogE) or O(ElogV). Thus, it is asymptotically same as Prim's Algorithm.

Input And Output:

Input File 1 (UnDirectced):



Output:

Kruscal's Algorithm to find MST:

Minimum path edges are shown with respective cost values.

Total Cost of the MST : 21