Algorithm and Data Structures

Project 2

Graph Algorithms
Singles-source shortest path and Minimum Spanning Tree (MST)

PROJECT MEMBERS
Varshitha Yataluru
Neela Ayshwaria Alagappan

Shortest Path (Dijkstra’s):

In Single source shortest path, we need to find the shortest path for each node from the source node with least path cost. For this we can use Dijkstra’s Algorithm.

Dijkstra's algorithm (or Dijkstra's Shortest Path First algorithm, SPF algorithm) is an algorithm for finding the shortest paths between nodes in a graph.

Program Structure:

1. Set distance of source node to 0 and other nodes to infinity.
2. For the current node, calculate the tentative distances of all its neighboring unvisited nodes from source node.
3. If the tentative distance is less than the current value of the neighboring unvisited node, then update the new tentative distance value.
4. Mark the current node as visited when you are done visiting all its neighboring nodes.
5. Repeat 2 and 3 steps until all nodes are visited.
6. For a node, print its shortest path from the source node and the minimum cost to reach the current node from the source node.

Time complexity

The time complexity of the algorithm here is O(E log V)

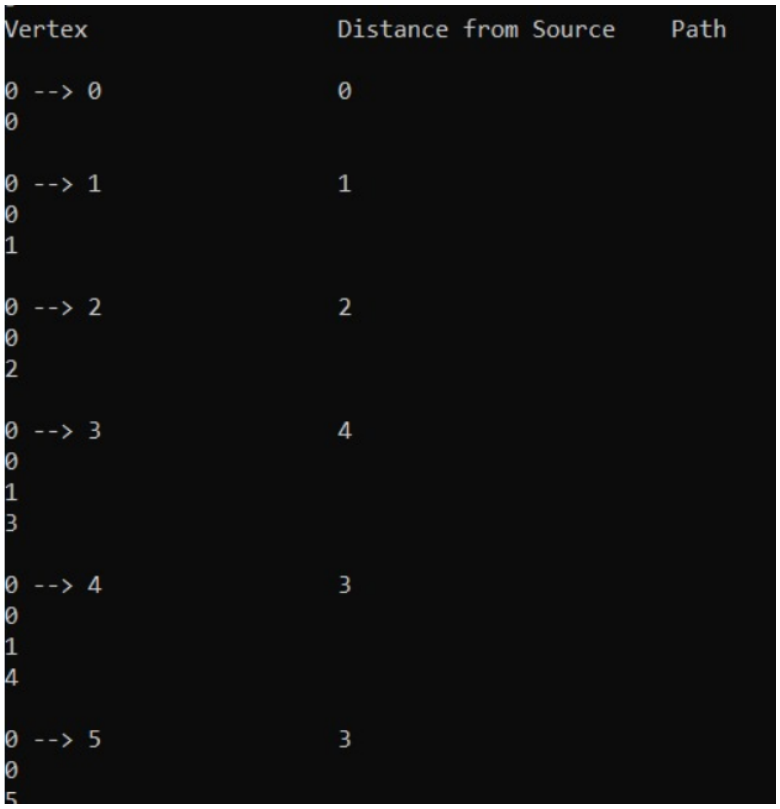
Data structure used

The data structure used is min heap.

Sample Input for Undirected Graph:

```
6 10 U
0 1 4
0 2 2
1 2 10
1 3 9
1 4 15
2 3 7
2 4 2
3 4 8
3 5 3
0 5 6
0
```

Sample Output for Undirected Graph

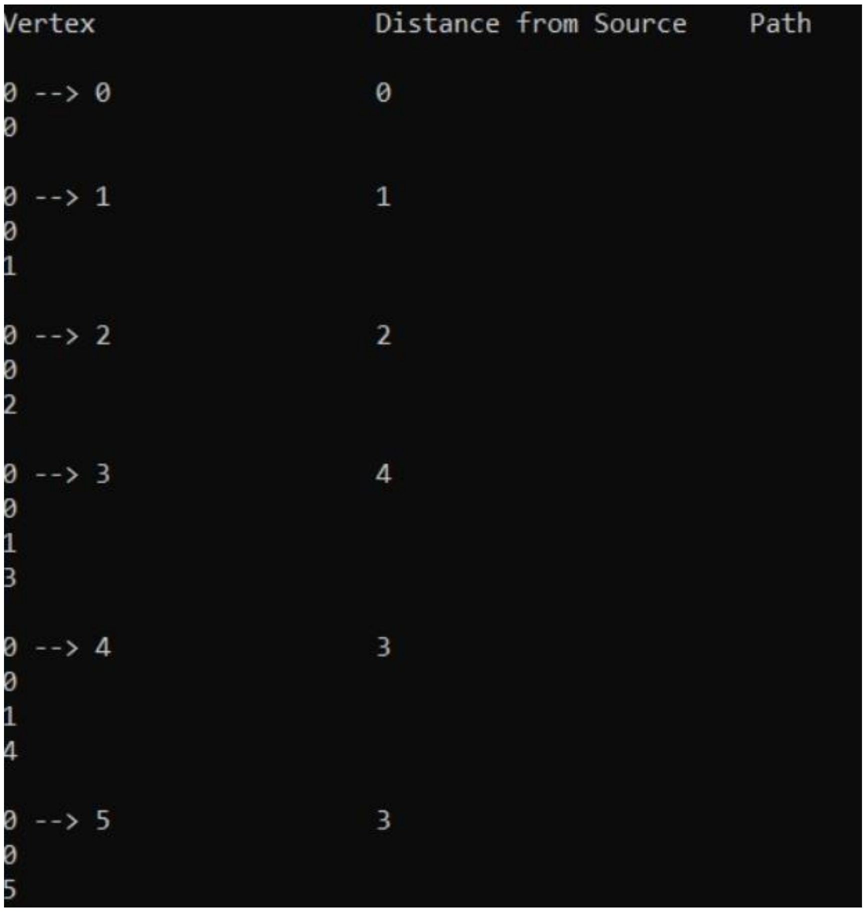


```
Vertex                     Distance from Source    Path

0 --> 0                    0
0

0 --> 1                    1
0
1

0 --> 2                    2
0
2

0 --> 3                    4
0
1
3

0 --> 4                    3
0
1
4

0 --> 5                    3
0
5
```

Sample Input for Directed Graph:

```
6 10 D
0 1 1
0 2 2
1 2 1
1 3 3
1 4 2
2 3 1
2 4 2
3 4 4
3 5 3
0 5 3
0
```

Sample Output for Directed Graph



```
Vertex                  Distance from Source    Path

0 --> 0                 0
0

0 --> 1                 1
0
1

0 --> 2                 2
0
2

0 --> 3                 4
0
1
3

0 --> 4                 3
0
1
4

0 --> 5                 3
0
5
```

Run Time for Dijkstra’s:
0.0031049251556396484 ms

Minimum Spanning Tree (Kruskal’s)

The minimum spanning tree is nothing but a subset of the graph,
where it connects all V vertices with V-1 edges of least path cost. So, the minimum spanning tree can be achieved using Kruskal’s algorithm.
In this algorithm, the edges are sorted in ascending order. If an edge between two nodes does not form a cycle, then consider that pair of nodes along with their weight. If it forms a cycle, then do not consider that pair and continue with next pair of nodes in increasing order. Continue this process V-1 times, where V is the number of vertices in the graph.

Program Structure:

1. The edges of the graph is sorted in ascending order based on the weight of each edge.
2. For every edge (u, v), parent method is invoked.
3. No cycle will be formed by including the edge (u,v) in the Minimum spanning tree, otherwise the edge (u,v) is not considered to be a part of MST.
4. The process of including edges in ascending order is continued till V-1 edges are obtained.
5. The total cost of the MST is computed by adding the weights of the edges that are included in the MST and it is printed on the screen.

Time complexity

The time complexity of the algorithm here is O(E log V)

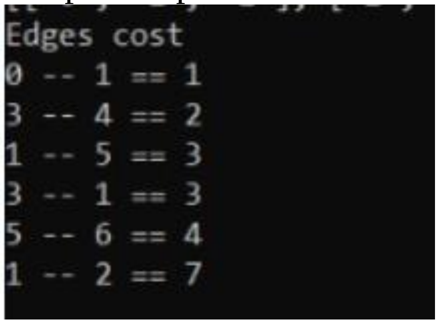
Data structure used

The data structure used here is disjoint sets.

Sample Input file:

```
0 1 1
1 2 7
6 0 11
1 6 5
1 5 3
5 6 4
2 5 9
2 3 10
3 4 2
4 5 8
3 1 3
```

Sample Output file:



```
Edges cost
0 -- 1 == 1
3 -- 4 == 2
1 -- 5 == 3
3 -- 1 == 3
5 -- 6 == 4
1 -- 2 == 7
```

Run Time for Kruskal's:
0.011966466903686523 ms

Instruction to run Program:
Provide an input file in the following format

1. The first character should denote the number of vertices.
2. The second character should denote the number of edges.
3. The first character should denote whether the graph is directed(D) or undirected(U).
4. The next three pairs of characters should denote the nodes (u,v) and their corresponding weights.

Eg: consider a directed graph with 3 vertices and 2 edges
The input file should be as follows
6 10 D
A B 1
B C 2