Project 2: Graph Algorithms

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

1. Single-source shortest path

Single source shortest path refers to the problem of finding a path between two nodes or vertices in such a way that the sum of weights of its constituent edges is minimized. Shortest path can be defined for graphs irrespective of being directed or undirected. Here we are implementing Dijkstra's algorithm for finding shortest path between source and all other vertices of the graph.

1.1. Dijkstra's algorithms

This algorithm maintains two queues, one of the visited nodes and the other of the non-visited ones. It initializes the start node with 0 and all the other nodes with infinity. Now selecting the least cost edge, the pointer moves from one node to the other, calculating the cost each time and comparing the cost to find the least one. In the end when all the nodes have been traversed, the nodes in the visited list are displayed to show the least cost path between every node.

1.2. Data structure

Main: Dijkstra's algorithm is implemented using priority queue data structure. The user is first asked to enter the path of the text file. The main class takes input from a text file using buffered reader and inputs it in a queue. It also asks the user to select either single-source shortest path or minimum spanning tree to be implemented on the provided file. Then it calculates and displays number of vertices, edges and type of graph, it requests for the source vertex in case of single source shortest path and sends it to graph function for calculating the paths.

Edge: consists of strings v1 and v2 which store the end vertices of an edge and weight variable is used for storing weight of the edges.

Graph: uses hash map for mapping vertex names to vertex objects built from the set of edges and thereby creates a graph for set of vertices passed by the main class. It also uses priority queue to store the vertices by adding v one by one. This function is also responsible for printing paths from source vertex to other vertices.

Vertex: this function is responsible for comparing costs and printing the least costing path between two vertices.

1.3. Runtime

O (E log V)

1.4. Sample I/O

Input 1:

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		

Source:A

```
Enter full path of the text file:
C:\Users\salon\Desktop\New folder\inputFile1.txt
Enter: 1. Dijkstra's Algorithm
 2. Kruskal's Algorithm
1
noOfVertices: 6
noOfEdges: 10
graphType: U
Edge{v1='A', v2='B', weight=1}
Edge{v1='A', v2='C', weight=2}
Edge{v1='B', v2='C', weight=1}
Edge{v1='B', v2='D', weight=3}
Edge{v1='B', v2='E', weight=2}
Edge{v1='C', v2='D', weight=1}
Edge{v1='C', v2='E', weight=2}
Edge{v1='D', v2='E', weight=4}
Edge{v1='D', v2='F', weight=3}
Edge{v1='E', v2='F', weight=3}
Enter start node:
A
Line 1 6 10 U
A
A => B(1)
A => C(2)
A => C(2) => D(3)
A => B(1) => E(3)
A => C(2) => D(3) => F(6)
```

Source:D

```
Enter full path of the text file:
C:\Users\salon\Desktop\New folder\inputFile1.txt
Enter: 1. Dijkstra's Algorithm
 2. Kruskal's Algorithm
1
noOfVertices: 6
noOfEdges: 10
graphType: U
Edge{v1='A', v2='B', weight=1}
Edge{v1='A', v2='C', weight=2}
Edge{v1='B', v2='C', weight=1}
Edge{v1='B', v2='D', weight=3}
Edge{v1='B', v2='E', weight=2}
```

```
Edge{v1='C', v2='D', weight=1}
Edge{v1='C', v2='E', weight=2}
Edge{v1='D', v2='E', weight=4}
Edge{v1='D', v2='F', weight=3}
Edge{v1='E', v2='F', weight=3}
Enter start node:
D
Line 1 6 10 U
D => C(1) => A(3)
D => C(1) => B(2)
D => C(1)
D
D => C(1) => E(3)
D => F(3)
```

Input 2:

10	15	U
A	B	3
A	J	4
A	G	1
B	D	10
D	J	3
D	H	11
J	G	6
G	F	8
G	E	14
F	E	2
F	I	2
F	H	4
E	I	1
H	I	6
H	C	3

Source:A

```
Enter full path of the text file:
```

```
C:\Users\salon\Desktop\New folder\inputFile2.txt
Enter: 1. Dijkstra's Algorithm
 2. Kruskal's Algorithm
1
noOfVertices: 10
noOfEdges: 15
graphType: U
Edge{v1='A', v2='B', weight=3}
Edge{v1='A', v2='J', weight=4}
Edge{v1='A', v2='G', weight=1}
Edge{v1='B', v2='D', weight=10}
Edge{v1='D', v2='J', weight=3}
Edge{v1='D', v2='H', weight=11}
Edge{v1='J', v2='G', weight=6}
Edge{v1='G', v2='F', weight=8}
Edge{v1='G', v2='E', weight=14}
Edge{v1='F', v2='H', weight=4}
Edge{v1='F', v2='I', weight=2}
Edge{v1='F', v2='E', weight=2}
Edge{v1='H', v2='I', weight=6}
Edge{v1='I', v2='E', weight=1}
Edge{v1='H', v2='C', weight=3}
Enter start node:
A
Line 1 10 15 U
A
A => B(3)
A => G(1) => F(9) => H(13) => C(16)
A => J(4) => D(7)
A => G(1) => F(9) => E(11)
A => G(1) => F(9)
A => G(1)
A => G(1) => F(9) => H(13)
A => G(1) => F(9) => I(11)
A => J(4)
```

<u>Source: J</u>

```
Enter full path of the text file:
C:\Users\salon\Desktop\New folder\inputFile2.txt
Enter: 1. Dijkstra's Algorithm
 2. Kruskal's Algorithm
1
noOfVertices: 10
noOfEdges: 15
graphType: U
Edge{v1='A', v2='B', weight=3}
Edge{v1='A', v2='J', weight=4}
Edge{v1='A', v2='G', weight=1}
Edge{v1='B', v2='D', weight=10}
Edge{v1='D', v2='J', weight=3}
Edge{v1='D', v2='H', weight=11}
Edge{v1='J', v2='G', weight=6}
Edge{v1='G', v2='F', weight=8}
Edge{v1='G', v2='E', weight=14}
Edge{v1='F', v2='H', weight=4}
```

```
Edge{v1='F', v2='I', weight=2}
Edge{v1='F', v2='E', weight=2}
Edge{v1='H', v2='I', weight=6}
Edge{v1='I', v2='E', weight=1}
Edge{v1='H', v2='C', weight=3}
Enter start node:
J
Line 1 10 15 U
J => A(4)
J => A(4) => B(7)
J => D(3) => H(14) => C(17)
J => D(3)
J => A(4) => G(5) => F(13) => E(15)
J => A(4) => G(5) => F(13)
J => A(4) => G(5)
J => D(3) => H(14)
J => A(4) => G(5) => F(13) => I(15)
J
```

Input 3:

9	16	D
A	B	4
B	C	11
B	D	9
C	A	8
D	C	7
D	E	2
D	F	6
E	B	8
F	C	1
F	E	5
E	H	4
E	G	7
G	H	14
H	I	10
G	I	9
H	F	2

Source: A

```
Enter full path of the text file:
C:\Users\salon\Desktop\New folder\inputFile3.txt
Enter: 1. Dijkstra's Algorithm
 2. Kruskal's Algorithm
1
noOfVertices: 9
noOfEdges: 16
graphType: D
Edge{v1='A', v2='B', weight=4}
Edge{v1='B', v2='C', weight=11}
Edge{v1='B', v2='D', weight=9}
Edge{v1='C', v2='A', weight=8}
Edge{v1='D', v2='C', weight=7}
Edge{v1='D', v2='E', weight=2}
Edge{v1='D', v2='F', weight=6}
Edge{v1='E', v2='B', weight=8}
Edge{v1='F', v2='C', weight=1}
Edge{v1='F', v2='E', weight=5}
Edge{v1='E', v2='H', weight=4}
Edge{v1='E', v2='G', weight=7}
Edge{v1='G', v2='H', weight=14}
Edge{v1='H', v2='I', weight=10}
Edge{v1='G', v2='I', weight=9}
Edge{v1='H', v2='F', weight=2}
Enter start node:
A
Line 1 9 16 D
A
A => B(4)
A => B(4) => C(15)
A => B(4) => D(13)
A => B(4) => D(13) => E(15)
A => B(4) => D(13) => F(19)
A => B(4) => D(13) => E(15) => G(22)
A => B(4) => D(13) => E(15) => H(19)
A => B(4) => D(13) => E(15) => H(19) => I(29)
```

Source: H

```
Enter full path of the text file:
C:\Users\salon\Desktop\New folder\inputFile3.txt
Enter: 1. Dijkstra's Algorithm
 2. Kruskal's Algorithm
1
noOfVertices: 9
noOfEdges: 16
graphType: D
Edge{v1='A', v2='B', weight=4}
Edge{v1='B', v2='C', weight=11}
Edge{v1='B', v2='D', weight=9}
```

```
Edge{v1='C', v2='A', weight=8}
Edge{v1='D', v2='C', weight=7}
Edge{v1='D', v2='E', weight=2}
Edge{v1='D', v2='F', weight=6}
Edge{v1='E', v2='B', weight=8}
Edge{v1='F', v2='C', weight=1}
Edge{v1='F', v2='E', weight=5}
Edge{v1='E', v2='H', weight=4}
Edge{v1='E', v2='G', weight=7}
Edge{v1='G', v2='H', weight=14}
Edge{v1='H', v2='I', weight=10}
Edge{v1='G', v2='I', weight=9}
Edge{v1='H', v2='F', weight=2}
Enter start node:
H
Line 1 9 16 D
H => F(2) => C(3) => A(11)
H => F(2) => E(7) => B(15)
H => F(2) => C(3)
H => F(2) => E(7) => B(15) => D(24)
H => F(2) => E(7)
H => F(2)
H => F(2) => E(7) => G(14)
H
H => I(10)
```

1.5. Instructions

On running the program, it asks for a path to locate the file, so place the input file in a separate folder and pass the path, for example, `C:\Users\salon\Desktop\New folder\inputFile3.txt`. Now, enter the option between Dijkstra's (for single-source shortest path) and Kruskal's algorithm (for minimum spanning tree). In case of Dijkstra's it asks for source node to calculate the shortest paths. On entering the vertex, the output can be seen on the screen.

2. Minimum Spanning Tree

Minimum spanning tree refers to a tree structure which comprises of a subset of edges of the entire weighted connected undirected graph such that all the vertices are connected together without cycles and with minimum edge weight. Various algorithms exist for calculating the minimum spanning tree but here we are implementing Kruskal algorithm for it.

2.1. Kruskal's algorithm

For Kruskal's algorithm is a greedy algorithm. In this algorithm, first, all the edges are sorted in a non-decreasing order of their weight. Then we pick the edge with least edge weight. If that edge forms a cycle with the spanning tree formed till now, the edge is discarded else, the edge is included in the spanning tree. This process is repeated till V-1 edges are there in the spanning tree.

2.2. Data structure

Main: Dijkstra's algorithm is implemented using priority queue data structure. The user is first asked to enter the path of the text file. The main class takes input from a text file using buffered reader and inputs it in a queue. It also asks the user to select either single-source shortest path or minimum spanning tree to be implemented on the provided file. Then it

calculates and displays number of vertices, edges and type of graph, it requests for the source vertex in case of single source shortest path and sends it to graph function for calculating the paths.
Edge: consists of strings v1 and v2 which store the end vertices of an edge and weight variable is used for storing weight of the edges.
Vertex: this function is responsible for comparing costs and printing the least costing path between two vertices.
Kruskal: The function uses hash maps for storing parent and rank information. The edges are sorted based on their edge weight. The edge with least weight is selected and put in an array list to form a set of edges after checking the formation of cycles. Once all the edges have been checked and the spanning tree consists of V-1 edges, the algorithm stops and prints are the edges that form the minimum spanning tree.

2.3. Runtime

(E log E) or (E log V)

2.4. Sample I/O

Input 1:

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		

```
Enter full path of the text file:
C:\Users\salon\Desktop\New folder\inputFile1.txt
Enter: 1. Dijkstra's Algorithm
 2. Kruskal's Algorithm
2
noOfVertices: 6
noOfEdges: 10
graphType: U
Edge{v1='A', v2='B', weight=1}
Edge{v1='A', v2='C', weight=2}
Edge{v1='B', v2='C', weight=1}
Edge{v1='B', v2='D', weight=3}
Edge{v1='B', v2='E', weight=2}
```

```
Edge{v1='C', v2='D', weight=1}
Edge{v1='C', v2='E', weight=2}
Edge{v1='D', v2='E', weight=4}
Edge{v1='D', v2='F', weight=3}
Edge{v1='E', v2='F', weight=3}
M.S.T. has the edges: [Edge{v1='A', v2='B', weight=1}, Edge{v1='B',
v2='C', weight=1}, Edge{v1='C', v2='D', weight=1}, Edge{v1='B', v2='E',
weight=2}, Edge{v1='D', v2='F', weight=3}]
```

Input 2:

10	15	U
A	B	3
A	J	4
A	G	1
B	D	10
D	J	3
D	H	11
J	G	6
G	F	8
G	E	14
F	E	2
F	I	2
F	H	4
E	I	1
H	I	6
H	C	3

```
Enter full path of the text file:
C:\Users\salon\Desktop\New folder\inputFile2.txt
Enter: 1. Dijkstra's Algorithm
 2. Kruskal's Algorithm
2
noOfVertices: 10
noOfEdges: 15
graphType: U
Edge{v1='A', v2='B', weight=3}
```

```
Edge{v1='A', v2='J', weight=4}
Edge{v1='A', v2='G', weight=1}
Edge{v1='B', v2='D', weight=10}
Edge{v1='D', v2='J', weight=3}
Edge{v1='D', v2='H', weight=11}
Edge{v1='J', v2='G', weight=6}
Edge{v1='G', v2='F', weight=8}
Edge{v1='G', v2='E', weight=14}
Edge{v1='F', v2='H', weight=4}
Edge{v1='F', v2='I', weight=2}
Edge{v1='F', v2='E', weight=2}
Edge{v1='H', v2='I', weight=6}
Edge{v1='I', v2='E', weight=1}
Edge{v1='H', v2='C', weight=3}
M.S.T. has the edges: [Edge{v1='A', v2='G', weight=1}, Edge{v1='I',
v2='E', weight=1}, Edge{v1='F', v2='I', weight=2}, Edge{v1='A', v2='B',
weight=3}, Edge{v1='D', v2='J', weight=3}, Edge{v1='H', v2='C',
weight=3}, Edge{v1='A', v2='J', weight=4}, Edge{v1='F', v2='H',
weight=4}, Edge{v1='G', v2='F', weight=8}]
```

Input 3:

11	20	U
A	B	8
A	J	12
A	I	6
A	H	10
A	G	9
A	K	3
B	K	7
B	C	10
B	E	2
C	D	9
C	K	5
D	E	13
D	F	12
E	F	10
E	G	6

F	G	8
G	H	7
H	I	3
I	J	10
J	K	8

```
Enter full path of the text file:
C:\Users\salon\Desktop\New folder\inputFile4.txt
Enter: 1. Dijkstra's Algorithm
 2. Kruskal's Algorithm
2
noOfVertices: 11
noOfEdges: 20
graphType: U
Edge{v1='A', v2='B', weight=8}
Edge{v1='A', v2='J', weight=12}
Edge{v1='A', v2='I', weight=6}
Edge{v1='A', v2='H', weight=10}
Edge{v1='A', v2='G', weight=9}
Edge{v1='A', v2='K', weight=3}
Edge{v1='B', v2='K', weight=7}
Edge{v1='B', v2='C', weight=10}
Edge{v1='B', v2='E', weight=2}
Edge{v1='C', v2='D', weight=9}
Edge{v1='C', v2='K', weight=5}
Edge{v1='D', v2='E', weight=13}
Edge{v1='D', v2='F', weight=12}
Edge{v1='E', v2='F', weight=10}
Edge{v1='E', v2='G', weight=6}
Edge{v1='F', v2='G', weight=8}
Edge{v1='G', v2='H', weight=7}
Edge{v1='H', v2='I', weight=3}
Edge{v1='I', v2='J', weight=10}
Edge{v1='J', v2='K', weight=8}
M.S.T. has the edges: [Edge{v1='B', v2='E', weight=2}, Edge{v1='A',
v2='K', weight=3}, Edge{v1='H', v2='I', weight=3}, Edge{v1='C', v2='K',
weight=5}, Edge{v1='A', v2='I', weight=6}, Edge{v1='E', v2='G',
weight=6}, Edge{v1='B', v2='K', weight=7}, Edge{v1='F', v2='G',
weight=8}, Edge{v1='J', v2='K', weight=8}, Edge{v1='C', v2='D',
weight=9}]
```

2.5. Instructions

On running the program, it asks for a path to locate the file, so place the input file in a separate folder and pass the path, for example, `C:\Users\salon\Desktop\New folder\inputFile3.txt`. Now, enter the option between Dijkstra's (for single-source shortest path) and Kruskal's algorithm (for minimum spanning tree). In case of Kruskal it prints the edges and cost on selecting the option.