Project 4

Minimum Spanning Tree

Name: **Sakshi Patel**
Student ID: **1002198852**
Project: **4 - Minimum Spanning Tree**
CSE 5311: Design and Analysis of Algorithms

Minimum Spanning Tree:

- In the field of graph theory, a **spanning tree T** of an **undirected graph G** is a subgraph that is a tree which includes all the vertices of G, with a minimum possible number of edges.
- A **minimum spanning tree (MST)** is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.
- It is a spanning tree whose sum of **edge weights is as small as possible**.
- This **project** is to find a subset T of the edges of G such that all the nodes remain connected when only the edges in T are used. The sum of the lengths of the edges in T is as small as possible.
- A connected graph with n nodes must have at least **n-1 edges**, so this is the minimum number of edges there can be in T.
- There are two algorithms to find minimum spanning tree in a graph
  1. Kruskal Algorithm
  2. Prim's Algorithm
- Implementation of these algorithms is done using Java Programming language. My project file contains two java files first is **MST.java** and second is **MSTtest.java.**
- **MST.java** contains both algorithms with **user input**. In this user can select which algorithm to use. Users need to give the number of vertices and edges with the weight between those edges. Output of this java program gives you the selected edges with its weight which make minimum spanning tree. It also give you the total weight of minimum spanning tree.

```
The selected edges are...
0->2=1
0->1=3
1->4=3
2->5=4
5->3=2
The cost of minimum Spanning tree is 13
Execution time: 0.009 Second
```

- **MSTtest.java** file is to **compare** Kruskal algorithm and Prim's algorithm. User need to select the input size of data (data of 8 vertices, 250 vertices and 1000 vertices). Output of this java program give you the selected edges of MST with its weight and total weight of MST for both Kruskal Algorithm and Prim's Algorithm. You can also see the running time of both algorithms and compare it.

```
*****Select the option from below list*****
Option 1: 8 vertices
Option 2: 250 vertices
Option 3: 1000 vertices
Your option is: 1
*****Kruskal Algorithm*****
The selected edges are...
Edge: 7 , 0 Weight: 0.16
Edge: 3 , 2 Weight: 0.17
Edge: 7 , 1 Weight: 0.19
Edge: 2 , 0 Weight: 0.26
Edge: 7 , 5 Weight: 0.28
Edge: 5 , 4 Weight: 0.35
Edge: 6 , 2 Weight: 0.4
The cost of minimum Spanning tree is 1.81
Execution time: 0.014 Second
*****Prims Algorithm*****
The selected edges are...
0->7=0.16
7->1=0.19
0->2=0.26
2->3=0.17
7->5=0.28
5->4=0.35
2->6=0.4
The cost of minimum Spanning tree is 1.81
Execution time: 0.002 Second
0.014 > 0.002
Kruskal Algorithm takes more time than Prim's Algorithm
```

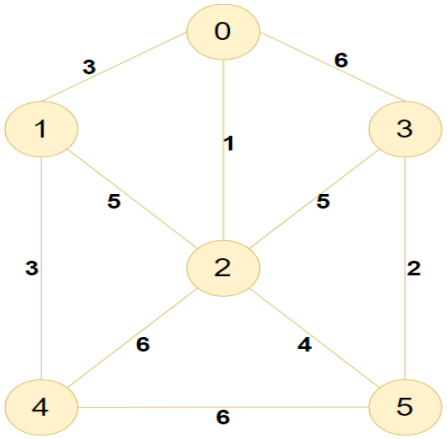
- In this both algorithms I used Two-Dimensional Array Data Structure.
- Java Development Kit is required to run the code.

1 Kruskal Algorithm:

- In this algorithm always minimum cost edge has to be selected. But it is not necessary that selected optimum edge is adjacent.
- From selected all vertices, edge with optimum weight is selected. Care should be taken for **not forming cycle**.

Algorithm:

1. First **ui()** method is called which take input from user with source and destination vertices with its weight. It use two dimensional array to store these values which create **graph[v][v]**. Where v in number of vertices. If there is no edge between that two vertices, then instead of infinity I take 999 value.
2. I am taking one **array[v]** of size as same as number of vertices and initially store 0 in that array. This array is for handling merging of to sets which contain vertices. This array is help to find that selected edges is not make a cycle in the MST.
3. One by one I select minimum edges and then call **find()** method which tells that does selected edge creates cycle or not in MST.
4. After selecting that edge I print it and set value 0.
5. At last it print total minimum cost of MST.



Step	Selected edge	Connected Components	Array[6]						Cost
			0	1	2	3	4	5	
Initialization	-	{0}{1}{2}{3}{4}{5}	0	0	0	0	0	0	
1	{2,0}	{0,2}{1}{3}{4}{5}	1	0	1	0	0	0	1
2	{5,3}	{0,2}{1}{4}{3,5}	1	0	1	2	0	2	2
3	{4,1}	{0,2}{1,4}{3,5}	1	3	1	2	3	2	3
4	{1,0}	{0,1,2,4}{3,5}	1	1	1	2	1	2	3
5	{5,2}	{0,1,2,3,4,5}	1	1	1	1	1	1	4
Total Minimum Cost									13

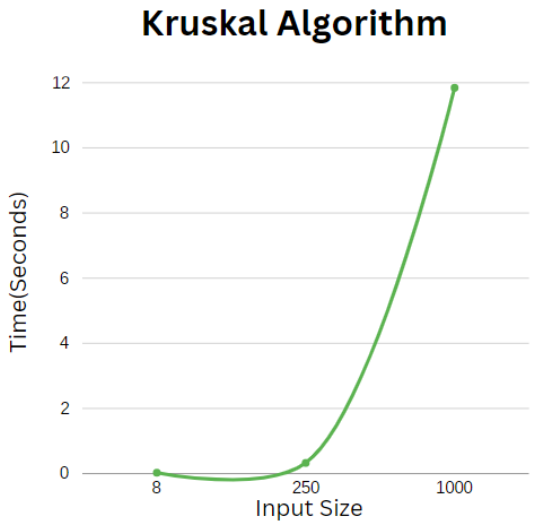
Output:

```
The selected edges are...
Edge: 2 , 0 Weight: 1
Edge: 5 , 3 Weight: 2
Edge: 4 , 1 Weight: 3
Edge: 1 , 0 Weight: 3
Edge: 5 , 2 Weight: 4
The cost of minimum Spanning tree is 13
Execution time: 0.009 Second
```

Time taken for large input size data:

Number of vertices	Number of edges	Time (Seconds)
8	16	0.028
250	1273	0.328
1000	8433	11.844

Comparison of input size and time taken to run the code in graphical form:

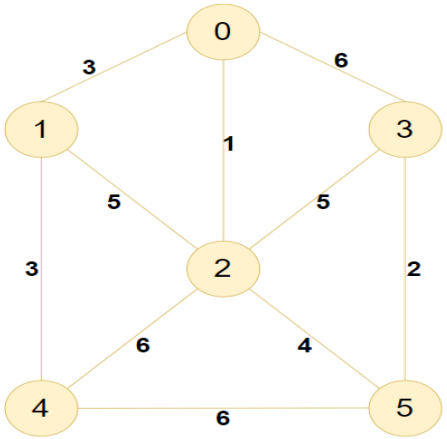


2 Prim's Algorithm:

- In Prim's algorithm all the vertices consider first. Then we will select an edge with minimum weight.
- The algorithm proceeds by selecting adjacent edges with minimum weight. Care should be taken for **not forming cycle**.

Algorithm:

1. First **ui()** method is called which take input from user with source and destination vertices with its weight. It use two dimensional array to store these values which create **graph[v][v]**. Where v in number of vertices. If there is no edge between that two vertices then instead of infinity I take 999 value.
2. After storing values in graph, user can define source vertex from where it start to select adjacent edges.
3. I am taking one **array[v]** of size as same as number of vertices and initially store 0 in that array. This array is for handling visited vertices in graph. If vertex in a graph is visited, then it's store 1 in that array else 0.
4. In **algo()** method we find minimum adjacent edges and with considering previous edges and take minimum value from that all edges with the help of **array[]** and display that selected edges with its weight. Algorithm stops when all the nodes have been reached.
5. After visiting all vertices **array[]** store 1 value for each index and at last it print total minimum cost of MST.



Step	Selected edge	Connected Components	Array[6]						Cost
			0	1	2	3	4	5	
Initialization	-	{0}	0	0	0	0	0	0	
1	{0,2}	{0,2}	1	0	1	0	0	0	1
2	{0,1}	{0,1,2}	1	1	1	0	0	0	3
3	{1,4}	{0,1,2,4}	1	1	1	0	1	0	3
4	{2,5}	{0,1,2,4,5}	1	1	1	0	1	1	4
5	{5,3}	{0,1,2,3,4,5}	1	1	1	1	1	1	2
								Total Minimum Cost	13

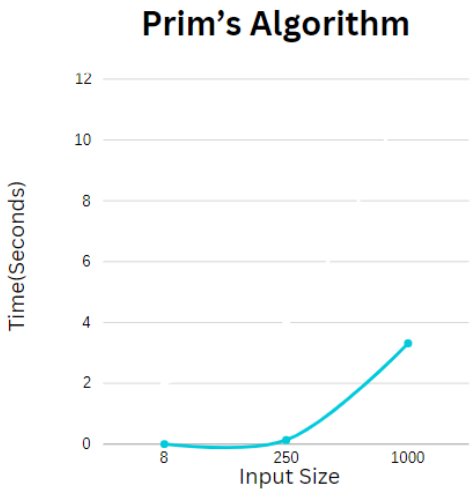
Output:

```
Enter the Source vertex :- 0
The selected edges are...
0->2=1
0->1=3
1->4=3
2->5=4
5->3=2
The cost of minimum Spanning tree is 13
Execution time: 0.014 Second
```

Time taken for large input size data:

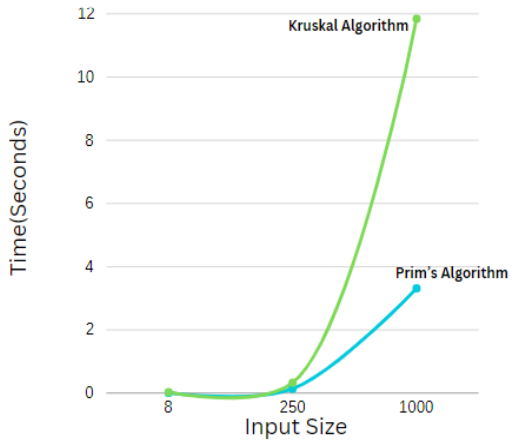
Number of vertices	Number of edges	Time (Seconds)
8	16	0.004
250	1273	0.136
1000	8433	3.314

Comparison of input size and time taken to run the code in graphical form:



❖ Comparison of Kruskal Algorithm and Prim's Algorithm:

Number of vertices	Number of edges	Time (Seconds)		
		Kruskal Algorithm		Prim's Algorithm
8	16	0.028	>	0.004
250	1273	0.328	>	0.136
1000	8433	11.844	>	3.314



How Kruskal and Prim's algorithm's running times change with respect to data size?

According to the size of input, when input size is increase Prim's algorithm perform better than Kruskal algorithm.

How their speed compare to each other in the cases with different data size?

When we run the code for large size of input we can see that Prim's algorithm run faster than Kruskal algorithm.

Can you improve their running time with your discovery?

I used liner search to find minimum edges instead of that if we use effective sorting algorithm we can improve running time.

Which one is better in terms what conditions?

Kruskal algorithm is preferable when there is small number of edges and large number of vertex.

Prim's algorithm is preferable when number of vertices need to be kept to a minimum in addition to the number of edges.