Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Report on
COMP 202: Data Structures and Algorithms
Mini Project

Submitted by:
Ishar Maharjan(29)
Aron Shrestha(42)
Pranima Kansakar(20)

Submitted to:
Dr. Rajani Chulyadyo
Department of Computer Science and Engineering

Submission Date:
January 19, 2019

Task:

To implement Kruskal's algorithm for finding a minimum spanning tree, apply it on at least 3 graphs and find the time complexity of its implementation.

Implementation:

Algorithm:

Input: A weighted, connected and undirected graph.
Output: A minimum spanning tree for the graph.
Steps:
1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle in the spanning tree formed so far. If no cycle is formed, include this edge. Otherwise, discard it.
3. Repeat step#2 until it is a spanning tree.

The step#2 uses ***Union-Find algorithm*** to detect a cycle. This algorithm performs two operations to complete the objective.
a. ***Find:*** Determine which subset(sub-tree) a particular element(vertex) is in. This can be used for determining if two elements(vertices) are in the same subset(sub-tree).
b. ***Union:*** Join two subsets(sub-trees) into a single subset(sub-tree).

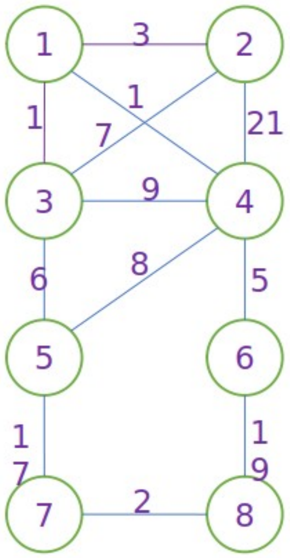
C++ Source Code:

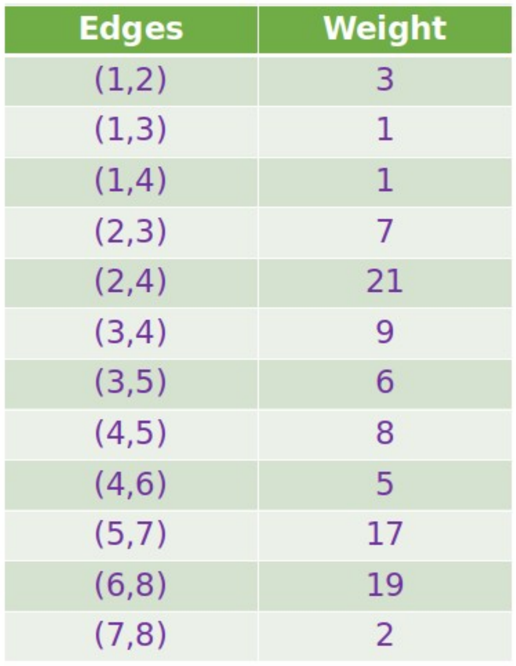
In the source code, two classes 'Edge' and 'Graph' have been used to create graphs. An in-built function 'qsort' has been used to implement comparison sort in order to sort the edges of the graphs in terms of their weights. Similarly, the third class 'Makeset' has been used in order to implement *Union-Find algorithm* to detect cycles in the graph. For the same purpose, two functions 'find-parent' and 'Union' have also been created. Further, the function 'KruskalAlgoForMST' has been created to implement the use of all the classes and functions to finally find a MST.

Test Program (Manually done according to the algorithm)

Graph 1:

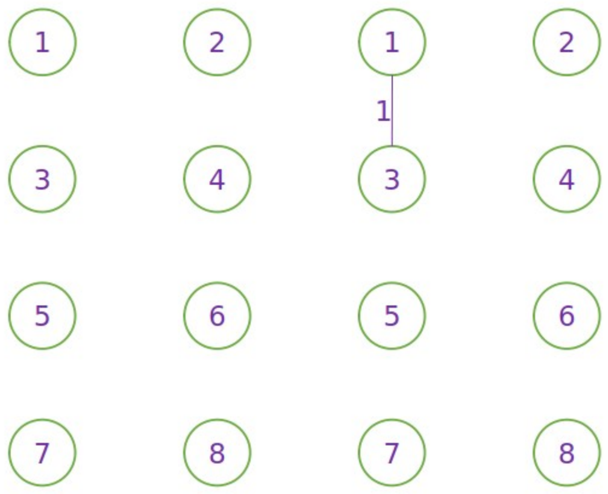
The graph below is the **input**.

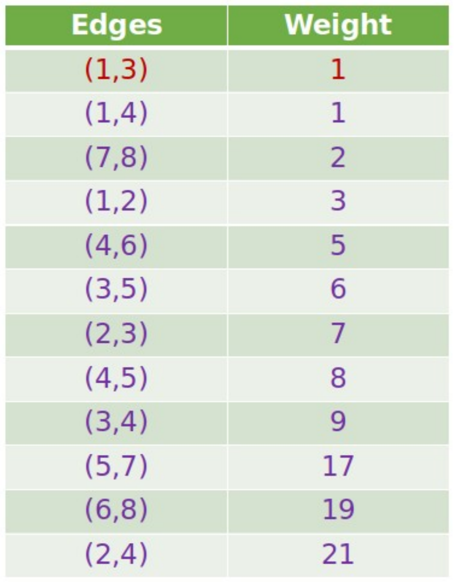




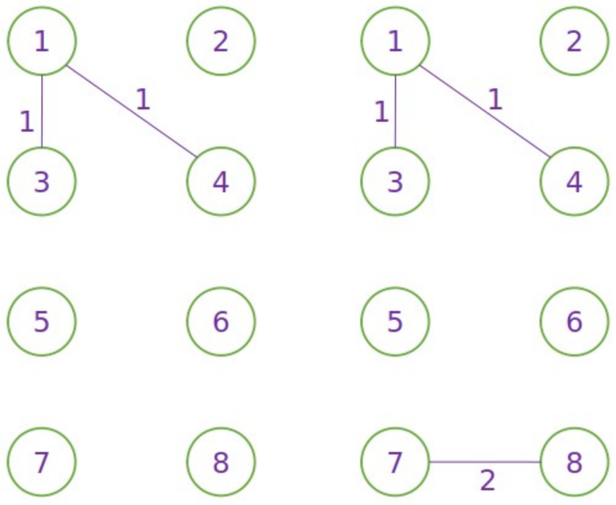
Edges	Weight
(1,2)	3
(1,3)	1
(1,4)	1
(2,3)	7
(2,4)	21
(3,4)	9
(3,5)	6
(4,5)	8
(4,6)	5
(5,7)	17
(6,8)	19
(7,8)	2

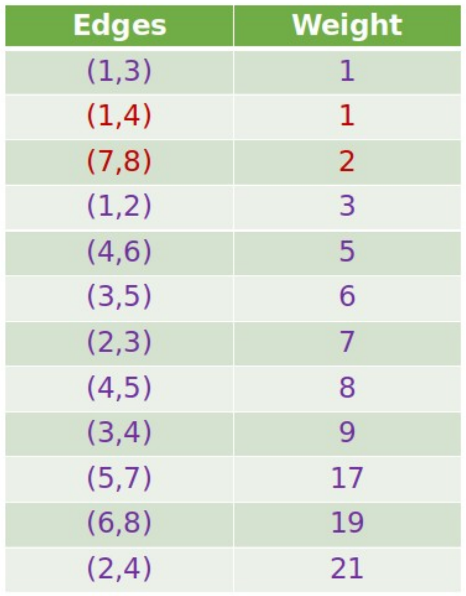
Using Kruskal’s algorithm:





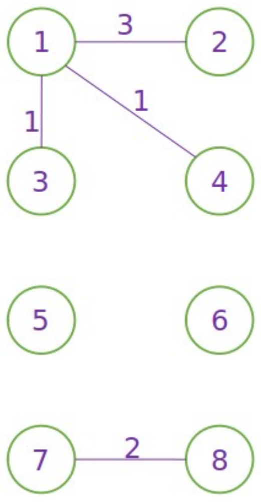
Edges	Weight
(1,3)	1
(1,4)	1
(7,8)	2
(1,2)	3
(4,6)	5
(3,5)	6
(2,3)	7
(4,5)	8
(3,4)	9
(5,7)	17
(6,8)	19
(2,4)	21

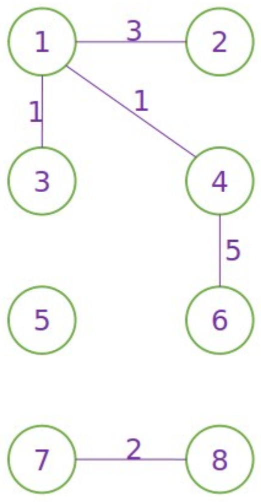


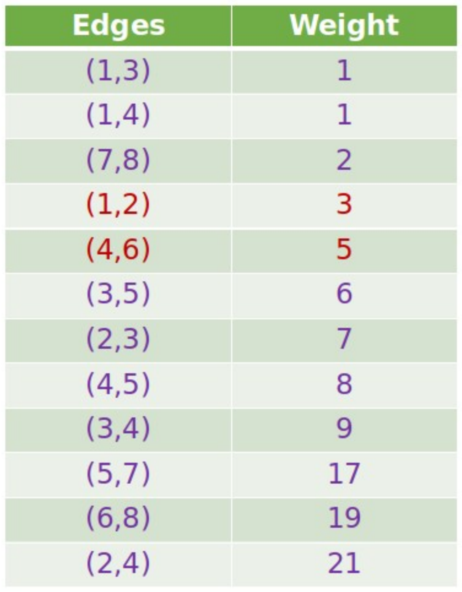


Edges	Weight
(1,3)	1
(1,4)	1
(7,8)	2
(1,2)	3
(4,6)	5
(3,5)	6
(2,3)	7
(4,5)	8
(3,4)	9
(5,7)	17
(6,8)	19
(2,4)	21

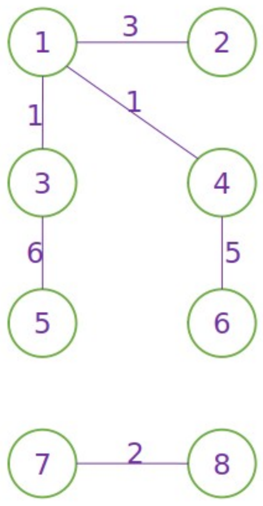
The MST evaluated above is the **expected output**.

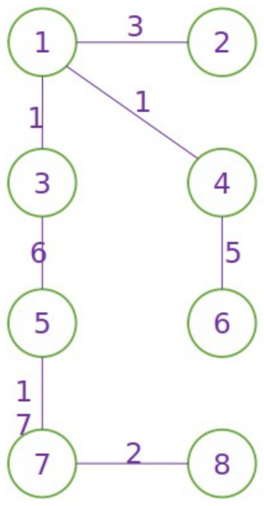


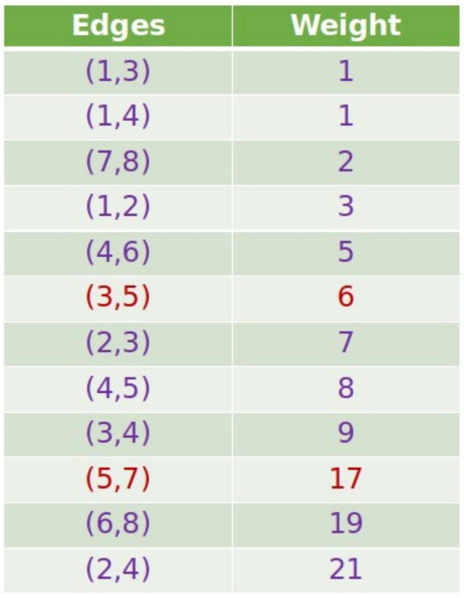




Edges	Weight
(1,3)	1
(1,4)	1
(7,8)	2
(1,2)	3
(4,6)	5
(3,5)	6
(2,3)	7
(4,5)	8
(3,4)	9
(5,7)	17
(6,8)	19
(2,4)	21



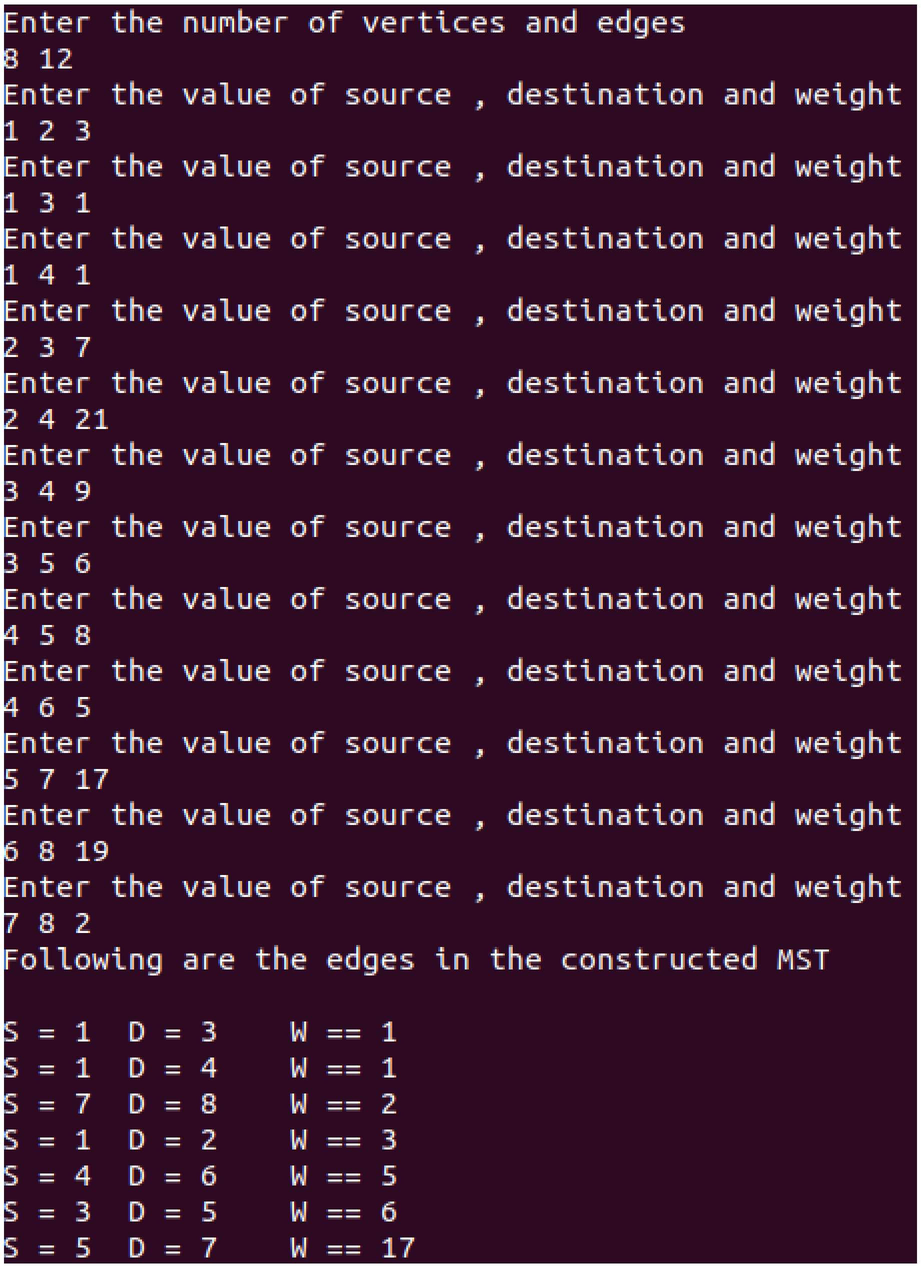




Edges	Weight
(1,3)	1
(1,4)	1
(7,8)	2
(1,2)	3
(4,6)	5
(3,5)	6
(2,3)	7
(4,5)	8
(3,4)	9
(5,7)	17
(6,8)	19
(2,4)	21

The MST evaluated above is the **expected output**.

Output from the program:

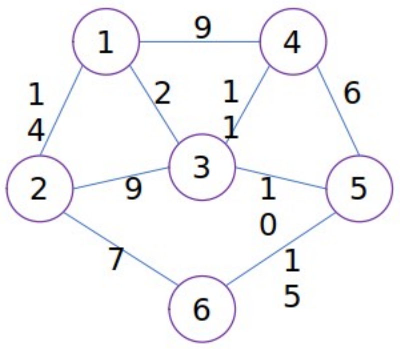


```
Enter the number of vertices and edges
8 12
Enter the value of source , destination and weight
1 2 3
Enter the value of source , destination and weight
1 3 1
Enter the value of source , destination and weight
1 4 1
Enter the value of source , destination and weight
2 3 7
Enter the value of source , destination and weight
2 4 21
Enter the value of source , destination and weight
3 4 9
Enter the value of source , destination and weight
3 5 6
Enter the value of source , destination and weight
4 5 8
Enter the value of source , destination and weight
4 6 5
Enter the value of source , destination and weight
5 7 17
Enter the value of source , destination and weight
6 8 19
Enter the value of source , destination and weight
7 8 2
Following are the edges in the constructed MST

S = 1  D = 3    W == 1
S = 1  D = 4    W == 1
S = 7  D = 8    W == 2
S = 1  D = 2    W == 3
S = 4  D = 6    W == 5
S = 3  D = 5    W == 6
S = 5  D = 7    W == 17
```

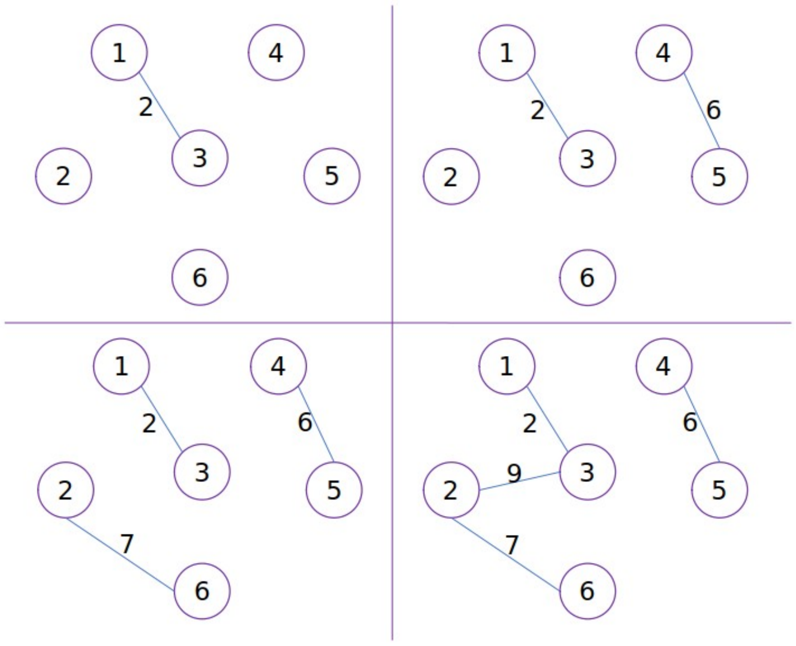
Graph 2:

The graph below is the **input**.

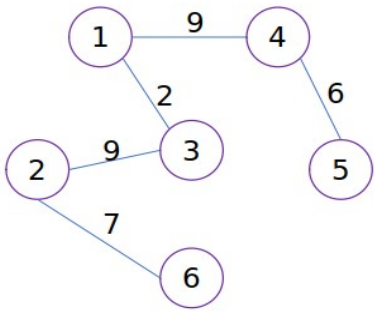


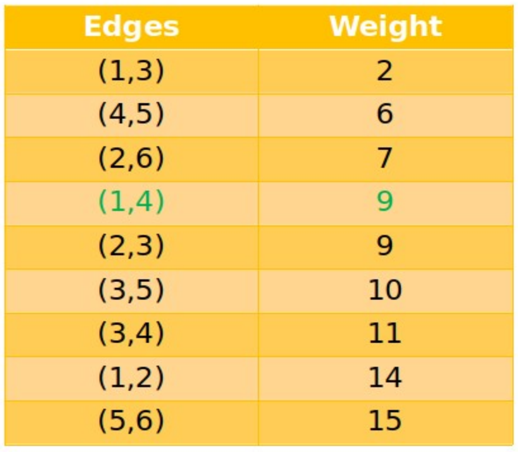
Edges	Weight
(1,2)	14
(1,3)	2
(1,4)	9
(2,3)	9
(2,6)	7
(3,4)	11
(3,5)	10
(4,5)	6
(5,6)	15

Using Kruskal's algorithm:



Edges	Weight
(1,3)	2
(4,5)	6
(2,6)	7
(1,4)	9
(2,3)	9
(3,5)	10
(3,4)	11
(1,2)	14
(5,6)	15

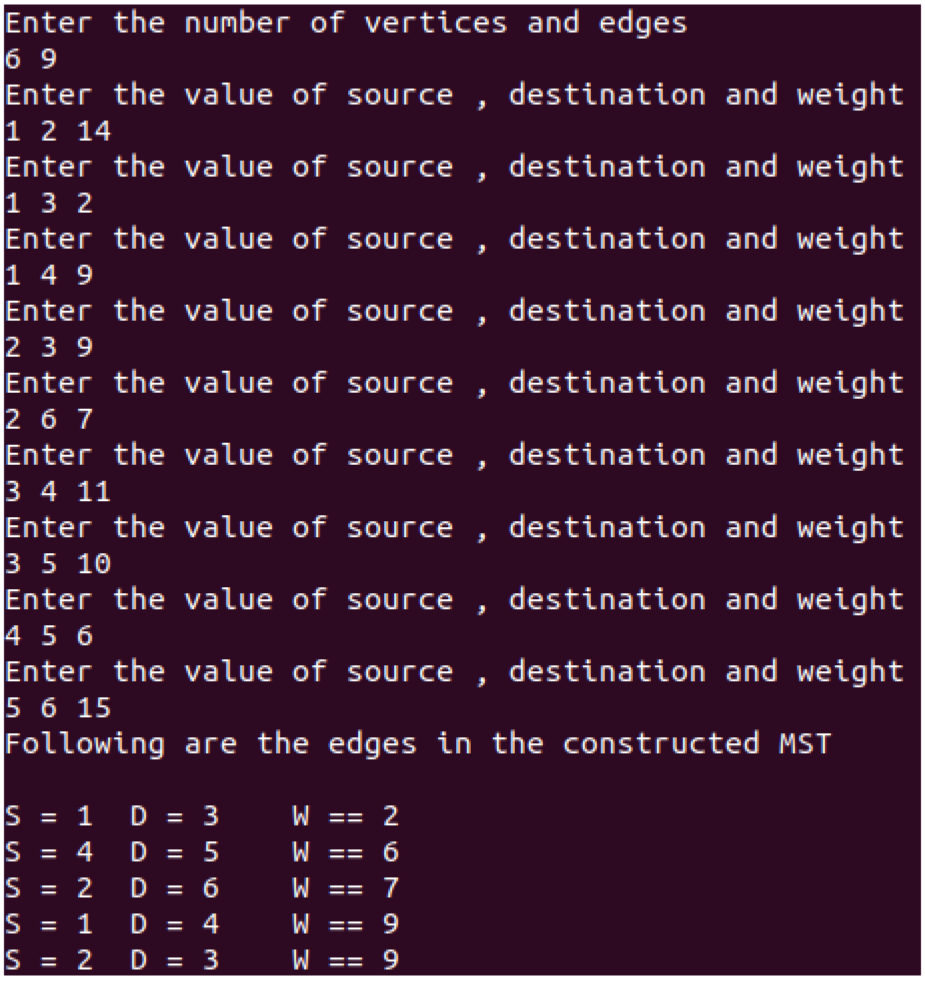




Edges	Weight
(1,3)	2
(4,5)	6
(2,6)	7
(1,4)	9
(2,3)	9
(3,5)	10
(3,4)	11
(1,2)	14
(5,6)	15

The MST evaluated above is the **expected output**.

Output from the program:

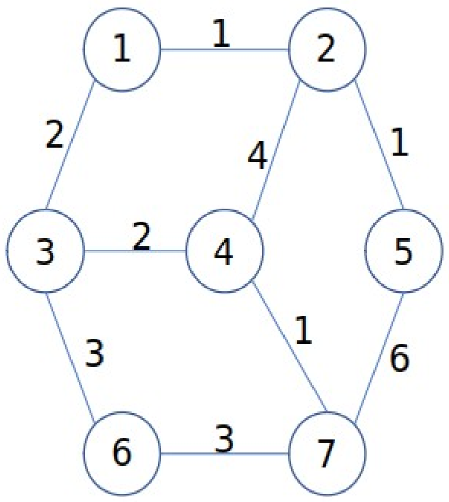


```
Enter the number of vertices and edges
6 9
Enter the value of source , destination and weight
1 2 14
Enter the value of source , destination and weight
1 3 2
Enter the value of source , destination and weight
1 4 9
Enter the value of source , destination and weight
2 3 9
Enter the value of source , destination and weight
2 6 7
Enter the value of source , destination and weight
3 4 11
Enter the value of source , destination and weight
3 5 10
Enter the value of source , destination and weight
4 5 6
Enter the value of source , destination and weight
5 6 15
Following are the edges in the constructed MST

S = 1  D = 3    W == 2
S = 4  D = 5    W == 6
S = 2  D = 6    W == 7
S = 1  D = 4    W == 9
S = 2  D = 3    W == 9
```

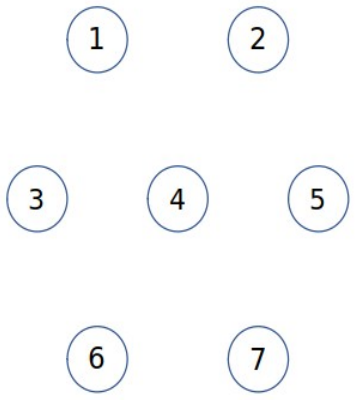
Graph 3:

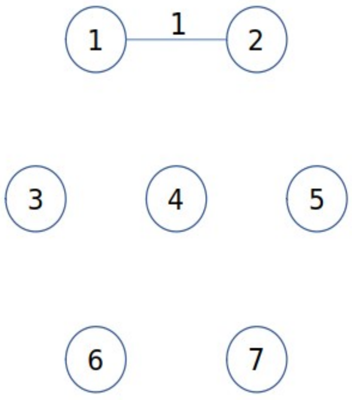
The graph below is the **input**.



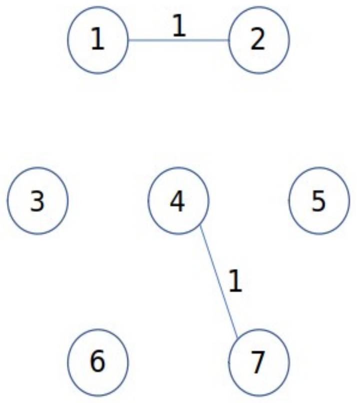
Edges	Weight
(1,2)	1
(4,7)	1
(2,5)	1
(1,3)	2
(3,4)	2
(3,6)	3
(6,7)	3
(2,4)	4
(7,5)	6

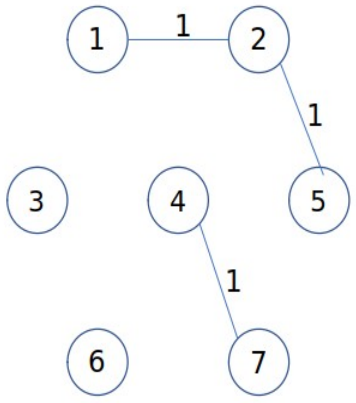
Using Kruskal's algorithm:



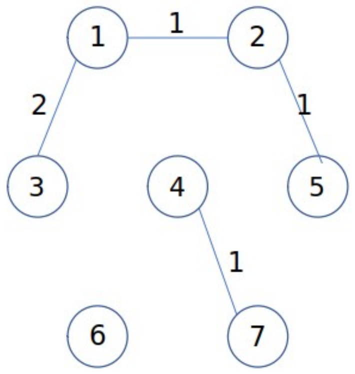


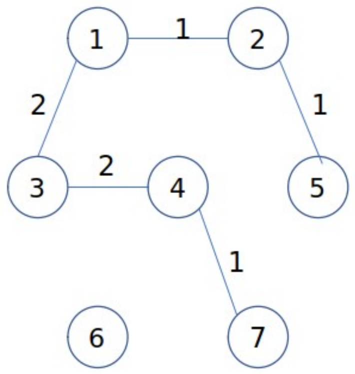
Edges	Weight
(1,2)	1
(4,7)	1
(2,5)	1
(1,3)	2
(3,4)	2
(3,6)	3



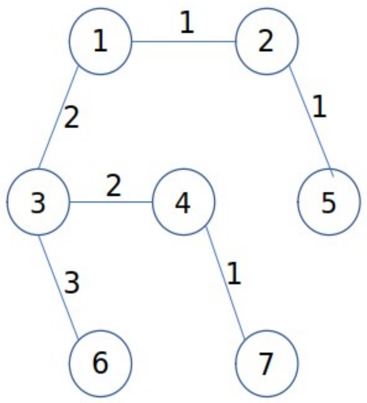


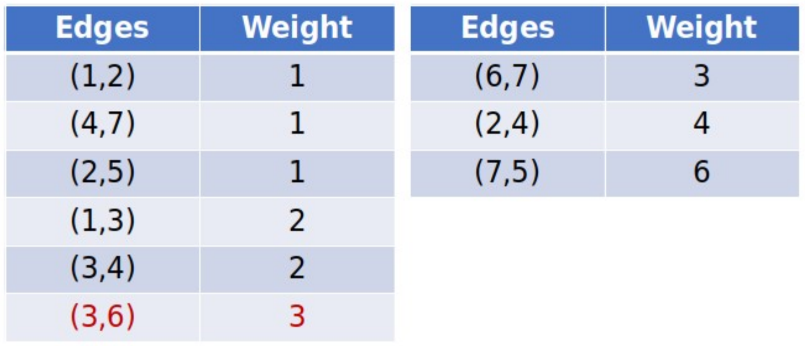
Edges	Weight
(1,2)	1
(4,7)	1
(2,5)	1
(1,3)	2
(3,4)	2
(3,6)	3





Edges	Weight
(1,2)	1
(4,7)	1
(2,5)	1
(1,3)	2
(3,4)	2
(3,6)	3

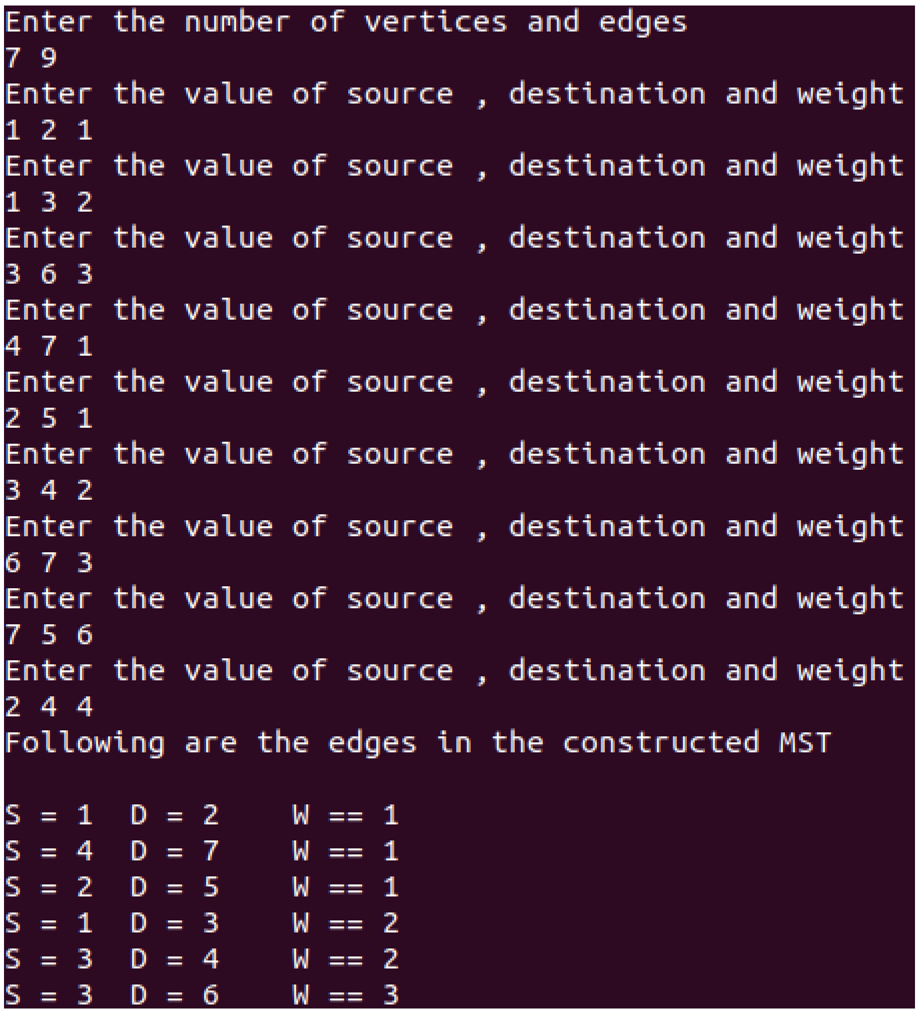




Edges	Weight	Edges	Weight
(1,2)	1	(6,7)	3
(4,7)	1	(2,4)	4
(2,5)	1	(7,5)	6
(1,3)	2		
(3,4)	2		
(3,6)	3		

The MST evaluated above is the **expected output**.

Output from the program:



```
Enter the number of vertices and edges
7 9
Enter the value of source , destination and weight
1 2 1
Enter the value of source , destination and weight
1 3 2
Enter the value of source , destination and weight
3 6 3
Enter the value of source , destination and weight
4 7 1
Enter the value of source , destination and weight
2 5 1
Enter the value of source , destination and weight
3 4 2
Enter the value of source , destination and weight
6 7 3
Enter the value of source , destination and weight
7 5 6
Enter the value of source , destination and weight
2 4 4
Following are the edges in the constructed MST

S = 1  D = 2    W == 1
S = 4  D = 7    W == 1
S = 2  D = 5    W == 1
S = 1  D = 3    W == 2
S = 3  D = 4    W == 2
S = 3  D = 6    W == 3
```

Hence, the implementation works correctly.

Time Complexity of the Program:

The time complexity of the program is *O(E log E)* or equivalently, *O(E log V)*. Here, E represents the number of edges and V represents the number of vertices of a graph.

At first, sorting the edges by weight using a comparison sort takes $O(E \log E)$ time. Next, we iterate through all the edges and apply find-union algorithm to detect which vertices are in which edges. Since in each iteration, we connect one vertex to the spanning tree, and perform two find operations along with possibly one union operation, we need to perform O(V) operations which can take at most $O(V \log V)$ time.

Now, since the value of E is at most V^2, $\log E = \log V^2 = 2 \log V$. Hence, $O(E \log E) = O(E \log V)$.

Similarly, each isolated vertex is a separate component of the minimum spanning forest. If we ignore isolated vertices we obtain $V \leq 2E$. Hence, $O(V \log V) = O(E \log V)$.

So overall time complexity is ***O(E log E)* or *O(E log V)***.