Dijkstra's Algorithm:

Consider a weighted directed graph G.

It starts with the source node and finds the shortest path between source node to all other nodes in the graph.

Define weighted distance matrix $D = (d_{ij})$ as follows;

$$d_{ii} = 0,$$

$d_{ij} = \infty; if\ v_i$ is not adjacent to v_j.

d_{ij} =weight assigned to the edge $if\ v_i$ is adjacent to v_j.

Define 2 sets K and U, where K consists of those vertices which have been fully investigated and between which the best path is known and U is the set of vertices which have not been fully identified.

Let $r \in K$. Then define an array best d[i] which is the length of the shortest path from r ro vertex I and another array tree[i] which is the just previous vertex to I on the shortest path.

Step 1: Let $K = \{r\}$ and U={all other vertices}

Set $best\ d[i] = d_{ir}$ and tree[i]=r.

Step 2: Find the vertex s in U with minimum value in best d[i] and put it in K.

Step 3: For each t in U, find bestd[s]+d_{st} and if this is less than best d[t], replace best d[t] with this new value and update tree[t]=s.

Step 4: repeat step 2,3 until U becomes an empty set.

Example 1: Let G be the graph as shown in the figure 2.. Consider the vertex B as the source vertex. Find the shortest paths from the source vertex to all the remaining vertices of G using Dijkstra's algorithm.

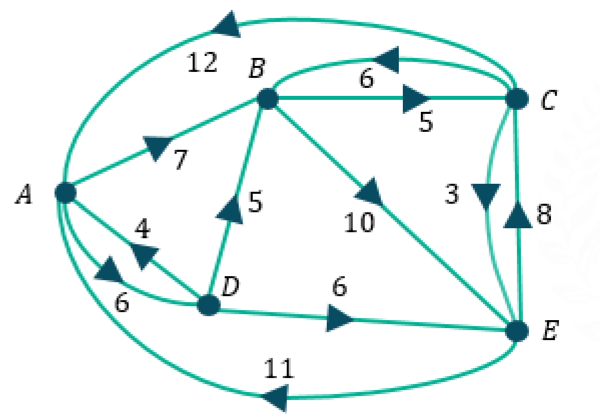


Figure 2: A directed graph G

Solution:

$$D(G) = \begin{bmatrix} 0 & 7 & \infty & 6 & \infty \\ \infty & 0 & 5 & \infty & 10 \\ 12 & 6 & 0 & \infty & 3 \\ 4 & 5 & \infty & 0 & 6 \\ 11 & \infty & 8 & \infty & 0 \end{bmatrix}$$

K={B},U={A,C,D,E}

	A	C	D	E
Best d	∞	**5**	∞	10
tree	B	B	B	B

Best(C)=5 and tree (C)=B

K={B,C},U={A,D,E}

	A	D	E

Best d	17	∞	**8**
tree	C	B	**C**

Bestd(E)=8 and tree (E)=C.

K={B,C,E},U={A,D}

	A	D
Best d	**17**	∞
tree	**C**	B

Bestd(A)=17 and tree (A)=C

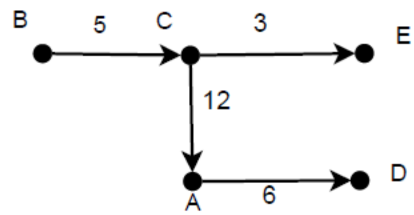
K={B,C,E,A},U={D}

	D
Best d	23
tree	A

Bestd(D)=23 and tree (D)=A

	A	C	D	E
Best d	17	5	23	8

tree	C	B	A	



Tree

2. Find the distance from vertex B to G for the graph G as shown in Figure 1.

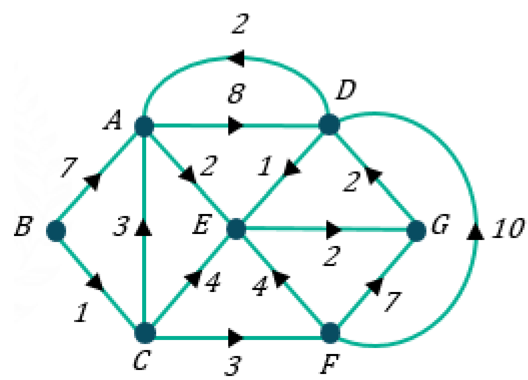
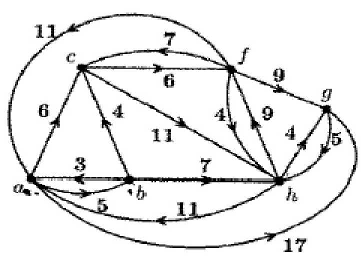
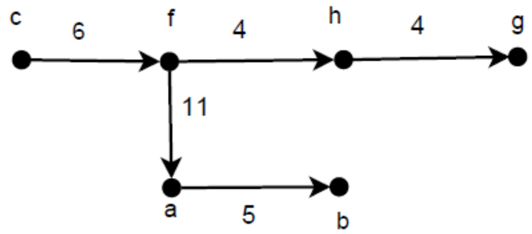


Figure 1: Directed graph G

Example: Implement Dijkstra's algorithm to find shortest path from c to all other vertices of the following network.



$$D(G) = \begin{bmatrix} 0 & 5 & 6 & \infty & 17 & \infty \\ 3 & 0 & 4 & \infty & \infty & 7 \\ \infty & \infty & 0 & 6 & \infty & 11 \\ 11 & \infty & 7 & 0 & 9 & 4 \\ \infty & \infty & \infty & \infty & 0 & 5 \\ 11 & \infty & \infty & 9 & 4 & 0 \end{bmatrix}$$



Tree

	a	b	f	g	h
$bestd$	17	22	6	14	10
$tree$	f	a	c	h	f

Floyd Warshall Algorithm

The Floyd Warshall Algorithm is an all pair shortest path algorithm.

Algorithm:

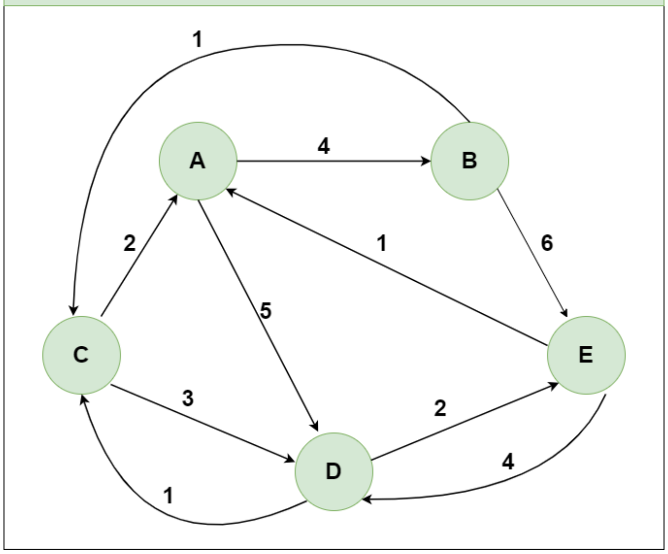
- Initialize the solution matrix same as the input graph matrix as a first step.
- Then update the solution matrix by considering all vertices as an intermediate vertex.
- The idea is to pick all vertices one by one and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.
- When we pick vertex number **k** as an intermediate vertex, we already have considered vertices **{0, 1, 2, .. k-1}** as intermediate vertices.
- For every pair **(i, j)** of the source and destination vertices respectively, there are two possible cases.
    - **k** is not an intermediate vertex in shortest path from **i** to **j**. We keep the value of **dist[i][j]** as it is.
    - **k** is an intermediate vertex in shortest path from **i** to **j**. We update the value of **dist[i][j]** as **dist[i][k] + dist[k][j],** if **dist[i][j] > dist[i][k] + dist[k][j]**

Pseudo-Code:

For k = 0 to n – 1
For i = 0 to n – 1
For j = 0 to n – 1
Distance[i, j] = min(Distance[i, j], Distance[i, k] + Distance[k, j])
where i = source Node, j = Destination Node, k = Intermediate Node

1. Find the shortest path from every node to every other nodes using Floyd warshall algorithm.

Example Graph



<u>Step 1</u>: *Initialize the Distance[][] matrix using the input graph such that Distance[i][j]= weight of edge from **i** to **j**, also Distance[i][j] = Infinity if there is no edge from **i** to **j**.*

Step1: Initializing Distance[ ][ ] using the Input Graph

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	∞	0	3	∞
D	∞	∞	1	0	2
E	1	∞	∞	4	0

Step 2: *Treat node **A** as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to **A**) + (Distance from **A** to j))*

*= Distance[i][j] = minimum (Distance[i][j], Distance[i][**A**] + Distance[**A**][j])*

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	?	?	?	?
C	2	?	?	?	?
D	∞	?	?	?	?
E	1	?	?	?	?

→

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	6	0	3	∞
D	∞	∞	1	0	2
E	1	5	∞	4	0

Step 3: *Treat node **B** as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to **B**) + (Distance from **B** to j))*

*= Distance[i][j] = minimum (Distance[i][j], Distance[i][**B**] + Distance[**B**][j])*

Step 3: Using Node B as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][B] + Distance[B][j])

	A	B	C	D	E
A	?	4	?	?	?
B	∞	0	1	∞	6
C	?	6	?	?	?
D	?	∞	?	?	?
E	?	5	?	?	?

	A	B	C	D	E
A	0	4	5	5	10
B	∞	0	1	∞	6
C	2	6	0	3	12
D	∞	∞	1	0	2
E	1	5	6	4	0

Step 4*: Treat node **C** as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to **C**) + (Distance from **C** to j))*

*= Distance[i][j] = minimum (Distance[i][j], Distance[i][**C**] + Distance[**C**][j])*

Step 4: Using Node C as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][C] + Distance[C][j])

	A	B	C	D	E
A	?	?	5	?	?
B	?	?	1	?	?
C	2	6	0	3	12
D	?	?	1	?	?
E	?	?	6	?	?

	A	B	C	D	E
A	0	4	5	5	10
B	3	0	1	4	6
C	2	6	0	3	12
D	3	7	1	0	2
E	1	5	6	4	0

Step 5*: Treat node **D** as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to **D**) + (Distance from **D** to j))*

*= Distance[i][j] = minimum (Distance[i][j], Distance[i][**D**] + Distance[**D**][j])*

Step 5: Using Node D as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][D] + Distance[D][j])

	A	B	C	D	E
A	?	?	?	5	?
B	?	?	?	4	?
C	?	?	?	3	?
D	3	7	1	0	2
E	?	?	?	4	?

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Step 6: *Treat node* ***E*** *as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

= Distance[i][j] = minimum (Distance[i][j], (Distance from i to ***E****) + (Distance from* ***E*** *to j))*

*= Distance[i][j] = minimum (Distance[i][j], Distance[i][****E****] + Distance[****E****][j])*

Step 6: Using Node E as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][E] + Distance[E][j])

	A	B	C	D	E
A	?	?	?	?	7
B	?	?	?	?	6
C	?	?	?	?	5
D	?	?	?	?	2
E	1	5	5	4	0

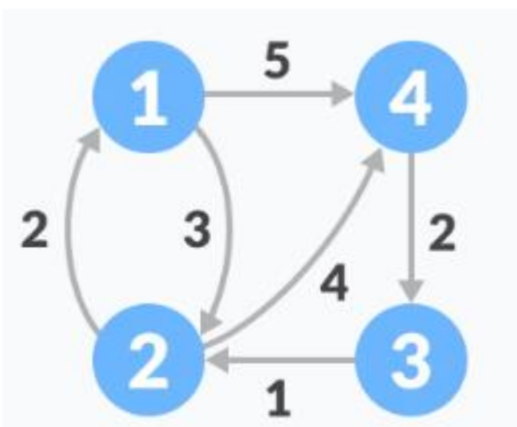
	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Step 7: *Since all the nodes have been treated as an intermediate node, we can now return the updated Distance[][] matrix as our answer matrix.*

Step 7: Return Distance[ ][ ] matrix as the result

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Example 2: Find the shortest path from every node to every other nodes using Floyd warshall algorithm.



Solution:

***Step 1**: Initialize the Distance[][] matrix using the input graph such that Distance[i][j]= weight of edge from **i** to **j**, also Distance[i][j] = Infinity if there is no edge from **i** to **j**.*

Step1: Initializing Distance[ ][ ] using the Input Graph

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline \begin{array}{c}1\\2\\3\\4\end{array} & \multicolumn{4}{c}{\begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix}} \end{array}$$

Step 2: treat 1 as an intermediate vertex and calculate the distance[][] for every {i,j} pair using the formula

$$distance[i][j] = \min\left(distance[i][j], (distance[i][1] + distance[1][j])\right)$$

	1	2	3	4
1	0	3	∞	5
2	2	0	∞	4
3	∞	1	0	∞
4	∞	∞	2	0

Step 3: treat 2 as an intermediate vertex and calculate the distance[][] for every {i,j} pair using the formula

$$distance[i][j] = \min\left(distance[i][j], (distance[i][2] + distance[2][j])\right)$$

	1	2	3	4
1	0	3	∞	5
2	2	0	∞	4
3	3	1	0	5
4	∞	∞	2	0

Step 4: treat 3 as an intermediate vertex and calculate the distance[][] for every {i,j} pair using the formula

$$distance[i][j] = \min\left(distance[i][j], (distance[i][3] + distance[3][j])\right)$$

	1	2	3	4
1	0	3	∞	5
2	2	0	∞	4
3	3	1	0	5
4	∞	3	2	0

Step 5: treat 4 as an intermediate vertex and calculate the distance[][] for every {i,j} pair using the formula

$$distance[i][j] = \min\left(distance[i][j], (distance[i][4] + distance[4][j])\right)$$

	1	2	3	4
1	0	3	7	5
2	2	0	6	4
3	3	1	0	5
4	5	3	2	0

Kruskal's Algorithm

This algorithm is used to derive the minimal spanning tree of the given graph is introduced by **Joseph Kruskal** (1956).

Let G be a graph with n vertices. Then, the minimal spanning tree is found as follows;

Step 1: Remove all the self-loops and parallel edges of G.

Step 2: List all the edges of the graph G in the order of non-decreasing weight

Step 3: Select the smallest edge of G, say e_k.

Step 4: Select another smallest edge e_l such that e_l makes no cycle with e_k.

Step 5: Select another smallest edge, say e_m such that e_m makes no cycle with e_k and e_l.

Step 6: Continue this process of selecting the smallest edges (from all the remaining edges of G) which make no cycle with previously selected edges until all $n - 1$ edges have been selected.

These edges constitute the desired minimal spanning tree.

Question 1: Find Minimal Spanning Tree of G Using Kruskal's Algorithm for the graph given in Figure 1.

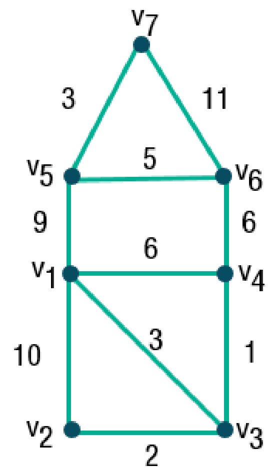


Figure 1: Graph G

Solution: First we will list all the edges of the graph G in the order of non-decreasing weight as follows; (v_3, v_4): 1, (v_2, v_3): 2, (v_1, v_3): 3, (v_5, v_7): 3, (v_5, v_6): 5, (v_4, v_6): 6, (v_1, v_4): 6, (v_1, v_5): 9, (v_1, v_2): 10

and (v_6, v_7): 11.

The smallest edge with minimum weight is selected first i.e., edge (v_3, v_4) with weight 1. The next edge with minimum weight i.e., (v_2, v_3) has weight 2. Add this edge to the spanning tree as it does not create any cycle with the previously added edge (v_3, v_4) as shown in the Figure 2.

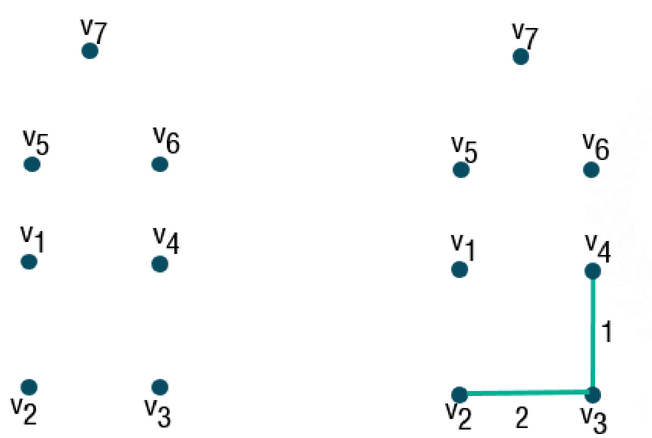


Figure 2: An empty graph and a graph with 2 edges with minimum weight

Next we add the edges (v_1, v_3): 3, (v_5, v_7): 3, (v_5, v_6): 5, (v_4, v_6): 6. The minimal spanning tree is as shown in the figure 3. Total weight is **1+2+3+3+5+6=20**

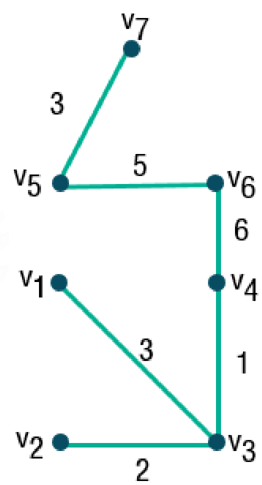


Figure 3: Minimal Spanning tree

Question 2: Find Minimal Spanning Tree of G Using Kruskal's Algorithm for the graph as shown in the figure 3.

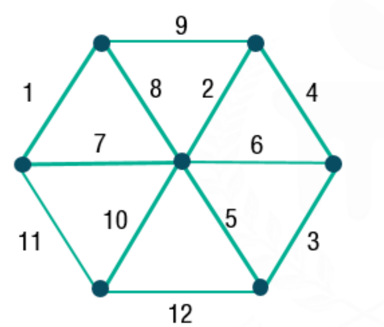


Figure 3: Graph G

Solution: The four edges with the lowest weight are selected (weights 1,2 3,4) as shown in figure 4.

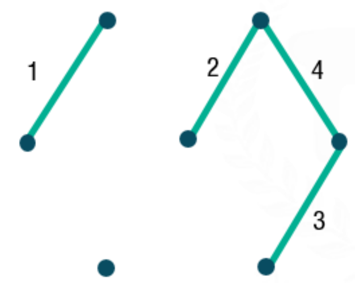


Figure 4: A graph with four edges with lowest weight

Edges with weight 5 or 6 cannot be included as they create a cycle.

Edge with weight 7 can be taken. But, edges with weight 8 or 9 cannot be taken.

Finally, an edge with weight 10 is selected to get a minimal spanning tree as shown in the figure 5, whose total weight is 1+2+3+4+7+10=27.

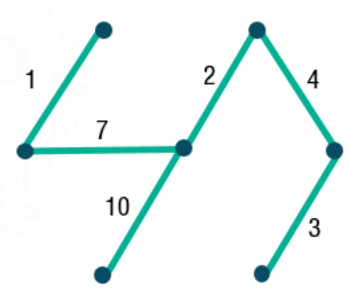


Figure 5: Minimal Spanning tree

Prim's Algorithm

Let G be a graph with n vertices. Then, the minimal spanning tree is found as follows:

Step I: Tabulate the given weights of the edges of G in an n-by-n table.

(Then the table is symmetric with respect to the diagonal, and the diagonal is empty). Set the weights of nonexistent edges as very large.

Step II: Draw n isolated vertices and label them $v_1, v_2, v_3, \dots, v_n$.

Step III: Start from vertex v_1 and connect it to the vertex, say v_k, which has the smallest entry in row 1 of the table.

Step IV: Consider v_1 and v_k as one subgraph, and connect this subgraph to a vertex, other than v_1 and v_k, which has the smallest entry among all the entries of the rows 1 and k. Let this new vertex be v_i.

Step V: Regard v_1, v_k, v_i as one subgraph, and continue this procedure until all n vertices have been connected by $n-1$ edges.

Question 1: Find the Shortest Spanning Tree of a Graph G Using Prim's Algorithm for the graph as shown in the Figure 1.

Solution: First create the 7×7 table showing the weights of every edge.

	$\boldsymbol{v_1}$	v_2	v_3	v_4	v_5	v_6	v_7
v_1	0	10	$\underline{\mathbf{3}}$	6	9	∞	∞
v_2	10	0	2	∞	∞	∞	∞
v_3	3	$\underline{\mathbf{2}}$	0	$\underline{\mathbf{1}}$	∞	∞	∞
v_4	6	∞	1	0	∞	$\underline{\mathbf{6}}$	∞
v_5	9	∞	∞	∞	0	5	$\underline{\mathbf{3}}$
v_6	∞	∞	∞	6	$\underline{\mathbf{5}}$	0	11
v_7	∞	∞	∞	∞	3	11	0

Start from vertex v_1 and connect it to the vertex which has the smallest entry (3) in row 1 of the table i.e., v_3. We get the 2[nd] graph shown in the Figure 6. Next select the minimum entry in 3[rd] row i.e., row corresponds to vertex v_3. The vertex v_4 is the one with minimum entry. Connect v_4 to v_3. Then we obtain the 3[rd] graph as shown in the Figure 6.

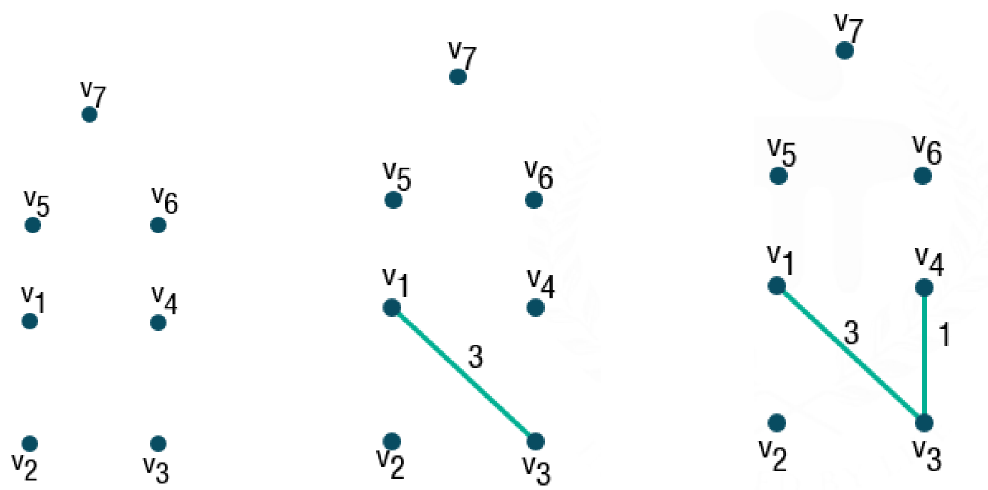


Figure 6: Empty graph, Graph with one edge, graph with 2 edge

Continuing like this we get the Minimal spanning tree as shown in the Figure 7. The total weight of the spanning tree is **20**.

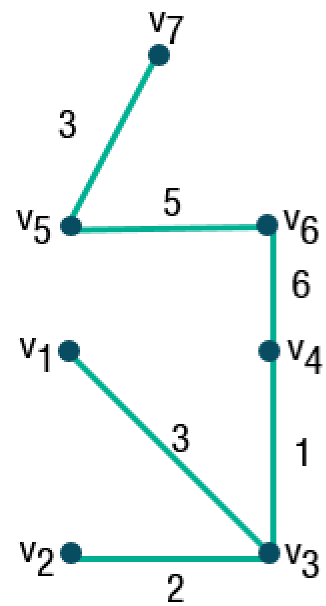


Figure 7: Minimal spanning tree

Application: Suppose that we have to connect n cities $v_1, v_2, \dots, v_n$ through a **network** of roads, given that c_{ij} is the cost of building a direct road between the cities v_i and v_j. Then, the problem of finding the least expensive **network** that connects all the cities is the same as finding a minimal spanning tree.

Comparison Between Prim’s and Kruskal’s Algorithms

The running time for large graphs depends on the time to sort m numbers. With this cost included, Prim’s algorithm may be faster than Kruskal's Algorithm.

Prim's and Kruskal's Algorithms have similar running times when edges are pre-sorted by weight.

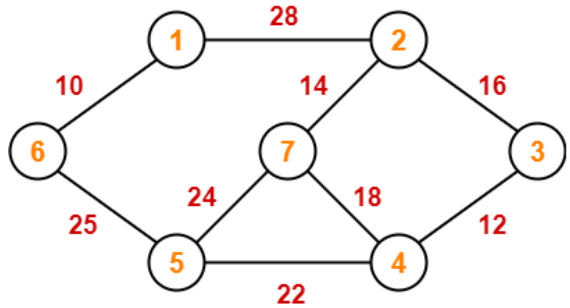
Both are greedy Algorithms

Prim’s algorithm can start with any vertex, but Kruskal’s algorithm starts with a vertex that carries minimum weight.

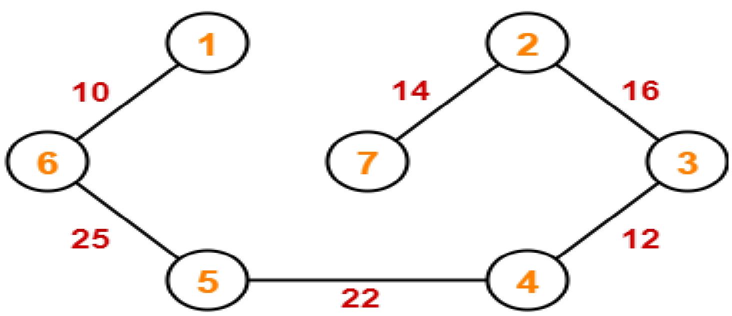
In Kruskal’s algorithm, it traverses one vertex more than once, whereas in Prim’s algorithm it traverses one vertex only once.

Questions:

1. For the graph in the figure below find minimal spanning tree using (i) Kruskal's algorithm (ii) Prims Algorithm.



Answer: Spanning tree



Sum of all edge weights=99.