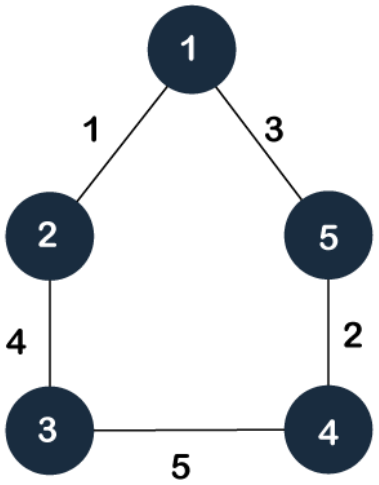
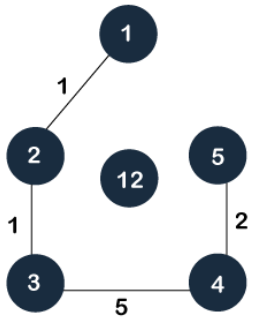
Spanning Tree

Defination: Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together.

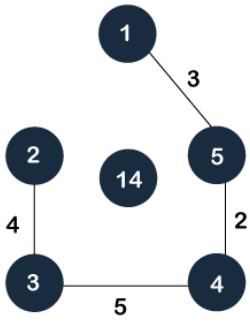
- A single graph can have many different spanning trees.
- A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree.
- The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.
- A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.
- Suppose we want to create the spanning tree of the graph, which is shown below:



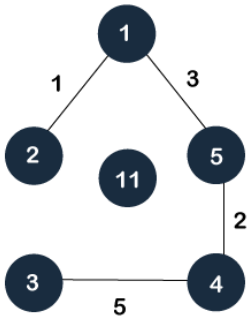
- spanning tree contains the same number of vertices as the graph, so the total number of vertices in the graph is 5; therefore, the spanning tree will also contain the 5 vertices.
- The edges in the spanning tree are equal to the number of vertices in the graph minus 1; therefore, the number of edges is 4. Three spanning trees can be created, which are shown below:



Spanning tree 1



Spanning tree 2



Spanning tree 3

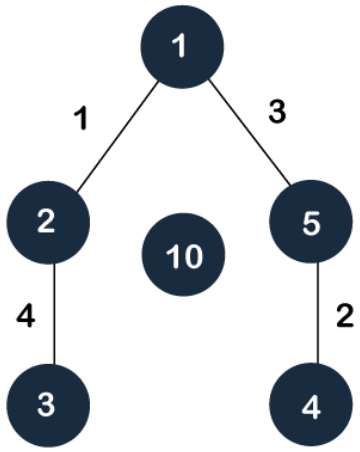
Minimum Spanning Trees

The minimum spanning tree is the tree whose sum of the edge weights is minimum.

From the above spanning trees, the total edge weight of the spanning tree 1 is 12, the total edge weight of the spanning tree 2 is 14, and the total edge weight of the spanning tree 3 is 11;

the total edge weight of the spanning tree 3 is minimum.

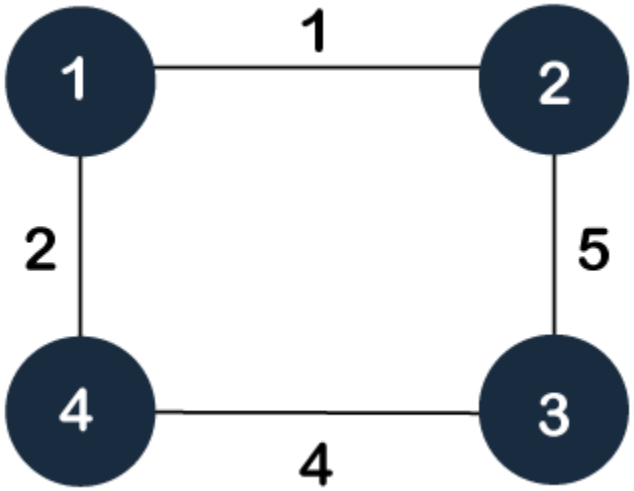
From the above graph, we can also create one more spanning tree as shown below:



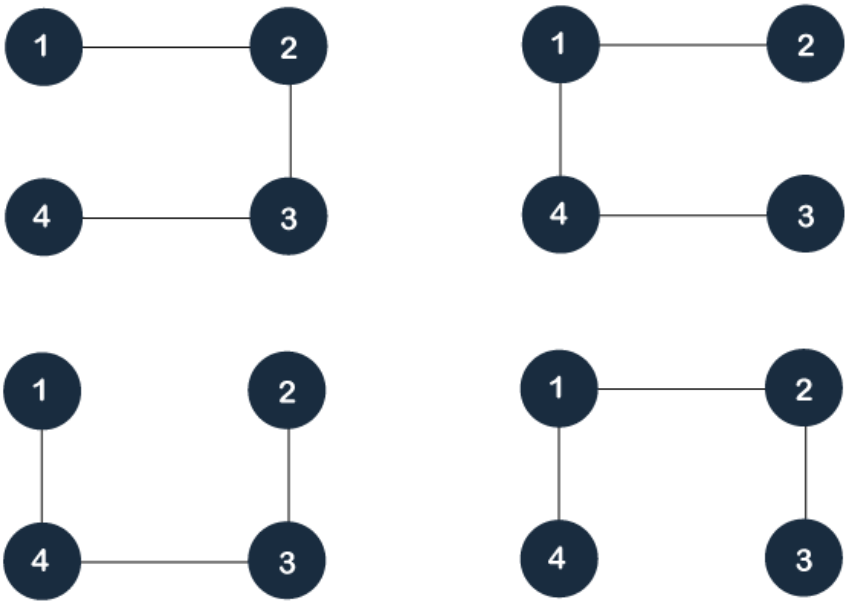
In the above tree, the total edge weight is 10 which is less than the above spanning trees; therefore, the minimum spanning tree is a tree which is having an edge weight, i.e., 10.

Properties of Spanning tree

- A connected graph can contain more than one spanning tree. The spanning trees which are minimally connected or we can say that the tree which is having a minimum total edge weight would be considered as the minimum spanning tree.
- All the possible spanning trees that can be created from the given graph G would have the same number of vertices, but the number of edges in the spanning tree would be equal to the number of vertices in the given graph minus 1.
- The spanning tree does not contain any cycle. Let's understand this property through an example.
  Suppose we have the graph which is given below:



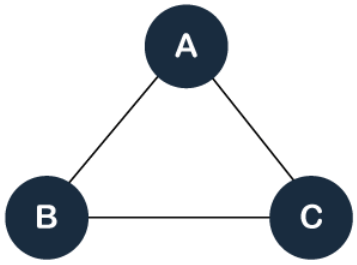
We can create the spanning trees of the above graph, which are shown below:



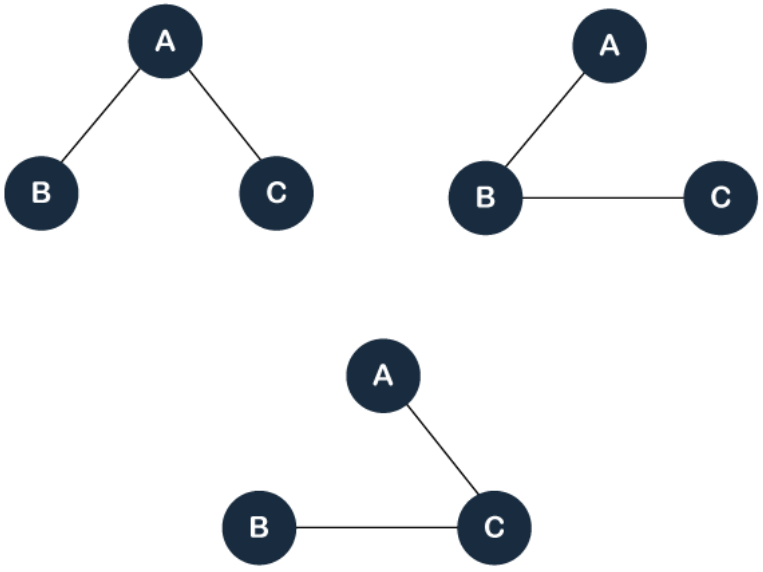
As we can observe in the above spanning trees that one edge has been removed. If we do not remove one edge from the graph, then the tree will form a cycle, and that tree will not be considered as the spanning tree.

- The spanning tree cannot be disconnected. If we remove one more edge from any of the above spanning trees as shown below:
  The above tree is not a spanning tree because it is disconnected now.
- If two or three edges have the same edge weight, then there would be more than two minimum spanning trees. If each edge has a distinct weight, then there will be only one or unique minimum spanning tree.

- A complete undirected graph can have n^{n-2} number of spanning trees where n is the number of vertices in the graph. For example, the value of n is 5 then the number of spanning trees would be equal to 125.
- Each connected and undirected graph contains at least one spanning tree.
- The disconnected graph does not contain any spanning tree, which we have already discussed.
- If the graph is a complete graph, then the spanning tree can be constructed by removing maximum (e-n+1) edges. Let's understand this property through an example. A complete graph is a graph in which each pair of vertices are connected. Consider the complete graph having 3 vertices, which is shown below:



- We can create three spanning trees from the above graph shown as below:



- According to this property, the maximum number of edges from the graph can be formulated as (e-n+1) where e is the number of edges, n is the number of vertices. When we substitute the value of e and n in the formula, then we get 1 value. It means that we can remove maximum 1 edge from the graph to make a spanning tree. In the above spanning trees, the one edge has been removed.

Applications of Spanning trees

The following are the applications of the spanning trees:

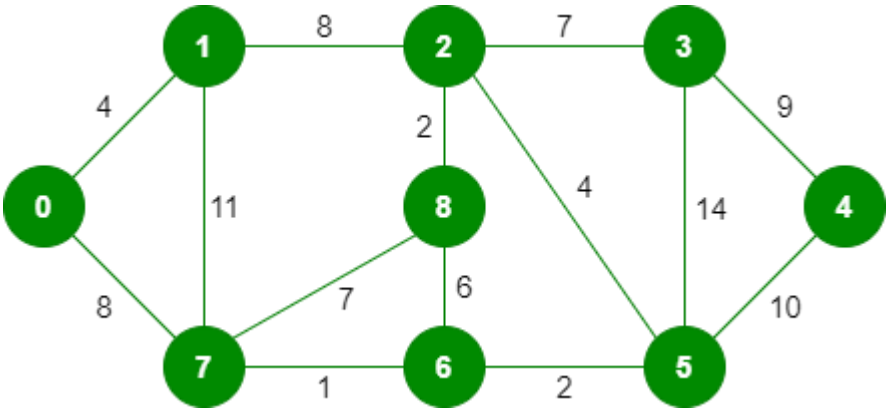
- **Building a network:** Suppose there are many routers in the network connected to each other, so there might be a possibility that it forms a loop. So, to avoid the formation of the loop, we use the tree data structure to connect the routers, and a minimum spanning tree is used to minimally connect them.
- **Clustering:** Here, clustering means that grouping the set of objects in such a way that similar objects belong to the same group than to the different group. Our goal is to divide the n objects into k groups such that the distance between the different groups gets maximized.

Krushkal's Algorithm

1. ***Sort all the edges in non-decreasing order of their weight.***
2. ***Pick the smallest edge.***
    a. ***Check if it forms a cycle with the spanning tree formed so far.***
    b. ***If cycle is not formed, include this edge. Else, discard it.***
3. ***Repeat step#2 until there are (V-1) edges in the spanning tree.***

The algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far. Let us understand it with an example: Consider the below input graph.

Example:



The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

After sorting:

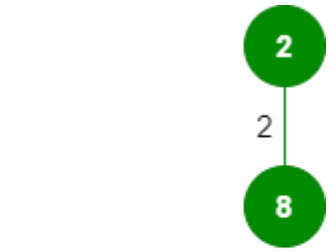
Weight Src Dest

1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

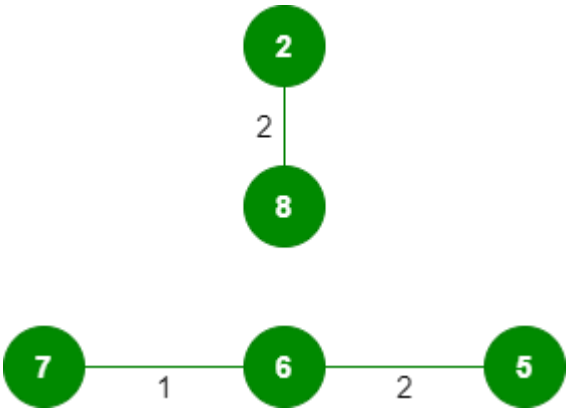
Now pick all edges one by one from the sorted list of edges

1. *Pick edge 7-6:* No cycle is formed, include it.

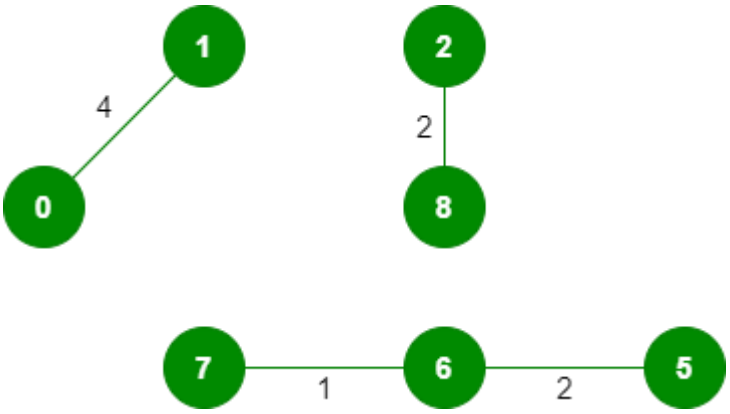
2. *Pick edge 8-2:* No cycle is formed, include it.



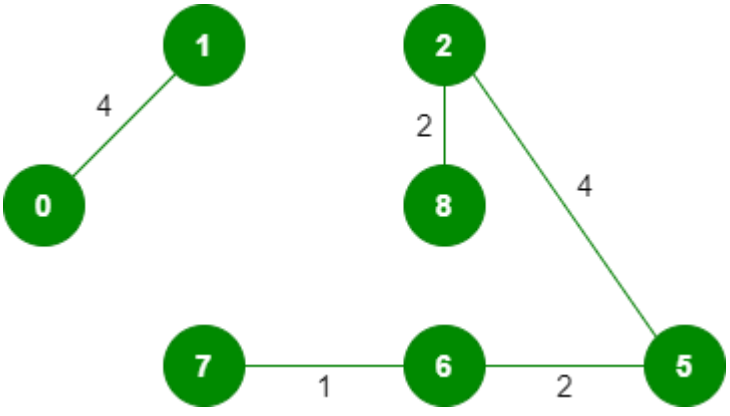
3. *Pick edge 6-5:* No cycle is formed, include it.



4. *Pick edge 0-1:* No cycle is formed, include it.

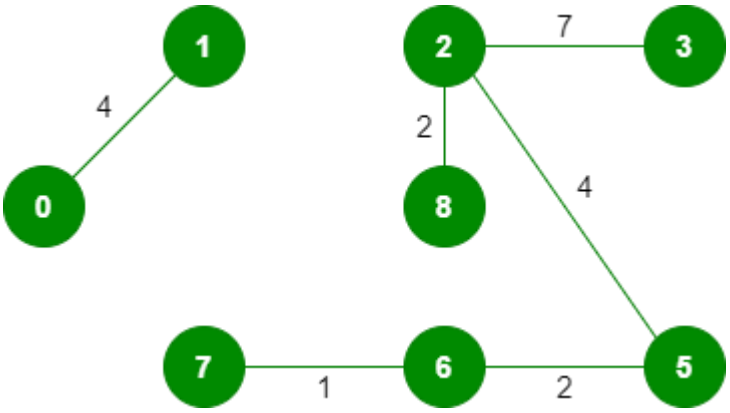


5. *Pick edge 2-5:* No cycle is formed, include it.



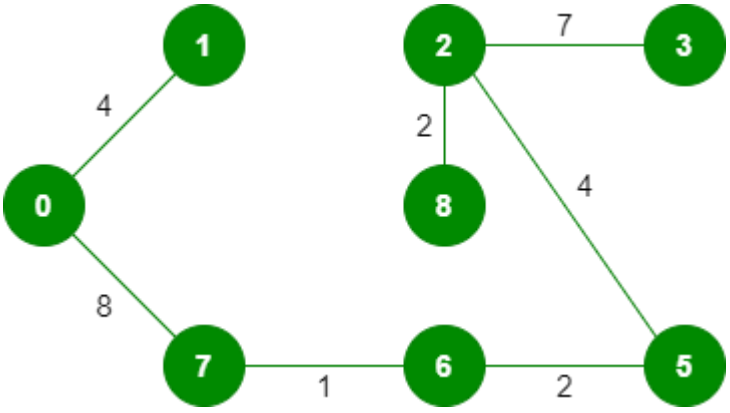
6. *Pick edge 8-6:* Since including this edge results in the cycle, discard it.

7. *Pick edge 2-3:* No cycle is formed, include it.



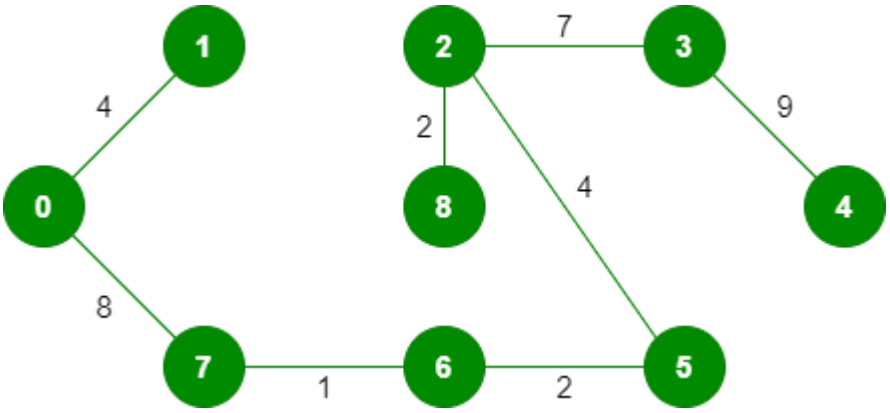
8. *Pick edge 7-8:* Since including this edge results in the cycle, discard it.

9. *Pick edge 0-7:* No cycle is formed, include it.



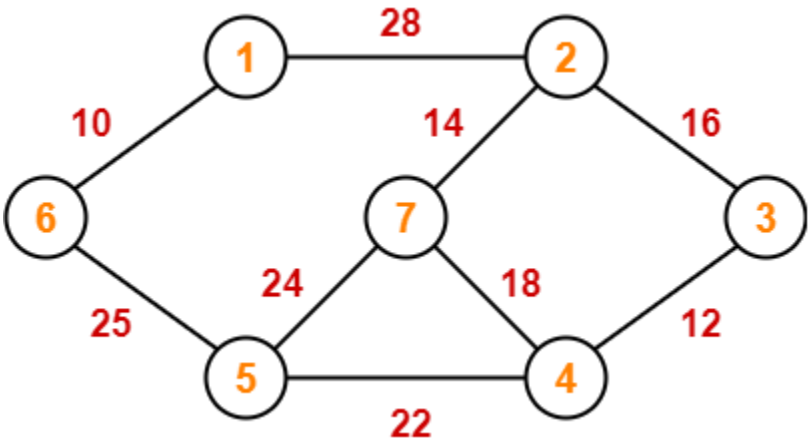
10. *Pick edge 1-2:* Since including this edge results in the cycle, discard it.

11. *Pick edge 3-4:* No cycle is formed, include it.



Since the number of edges included equals (V – 1), the algorithm stops here.

Example 2:

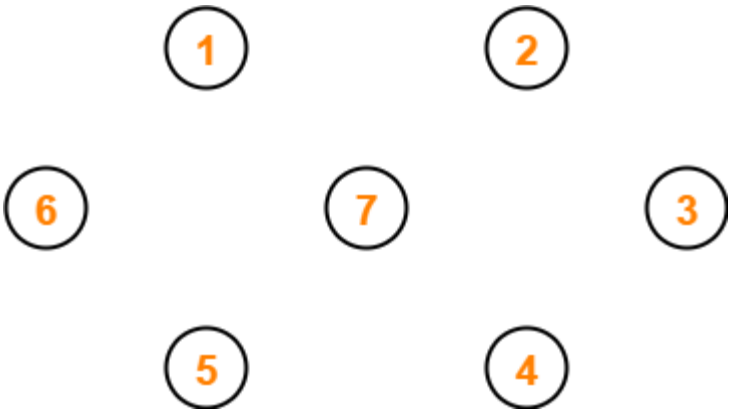


Solution-

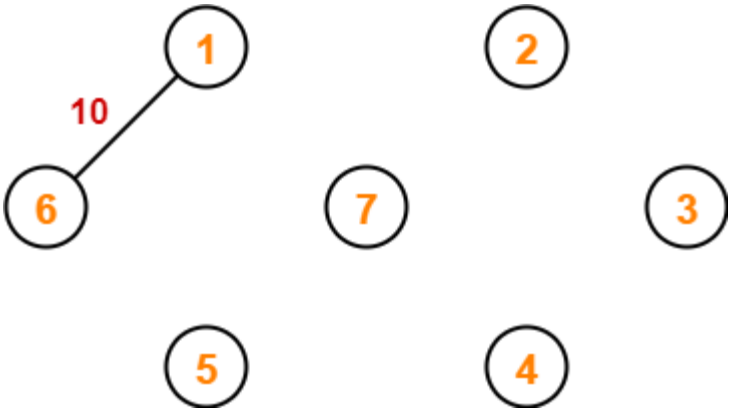
To construct MST using Kruskal's Algorithm,

- Simply draw all the vertices on the paper.
- Connect these vertices using edges with minimum weights such that no cycle gets formed.

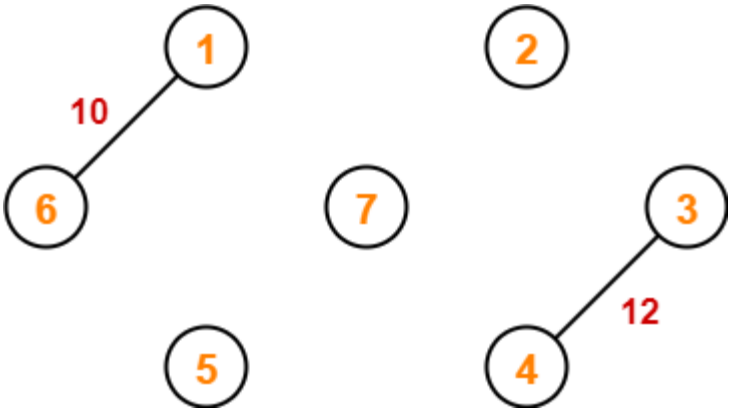
Step-01:



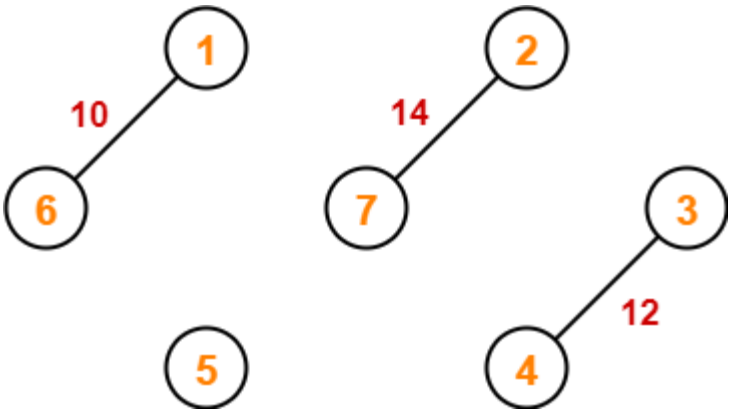
Step-02:



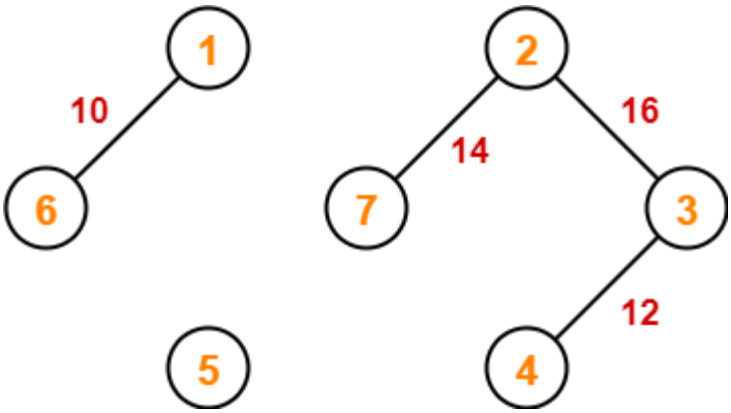
Step-03:



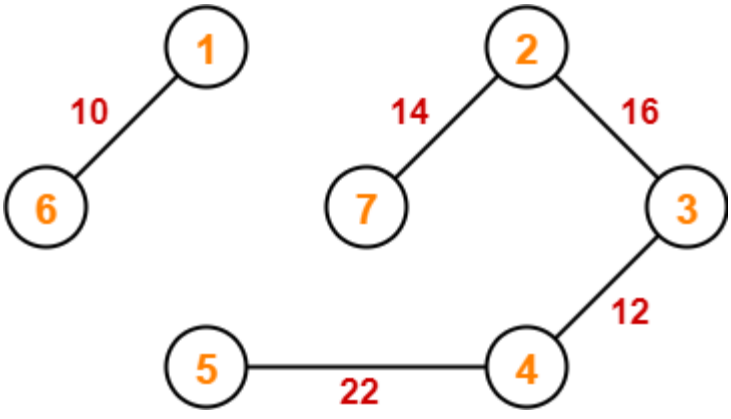
Step-04:



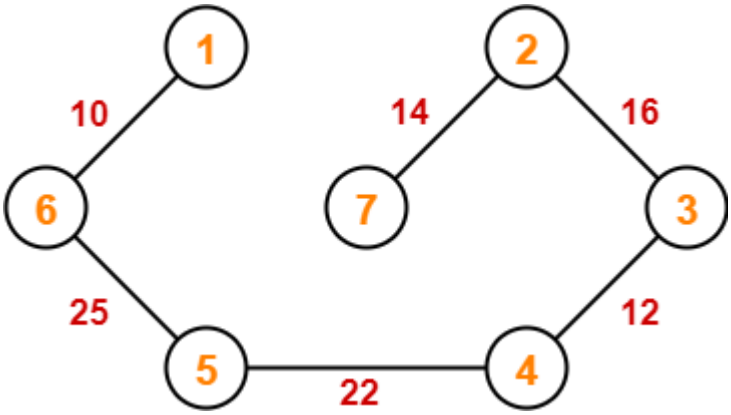
Step-05:



Step-06:



Step-07:



Since all the vertices have been connected / included in the MST, so we stop.

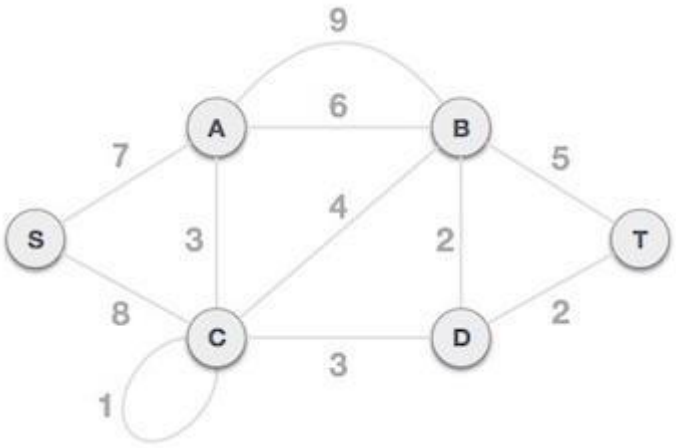
Weight of the MST

= Sum of all edge weights

= 10 + 25 + 22 + 12 + 16 + 14

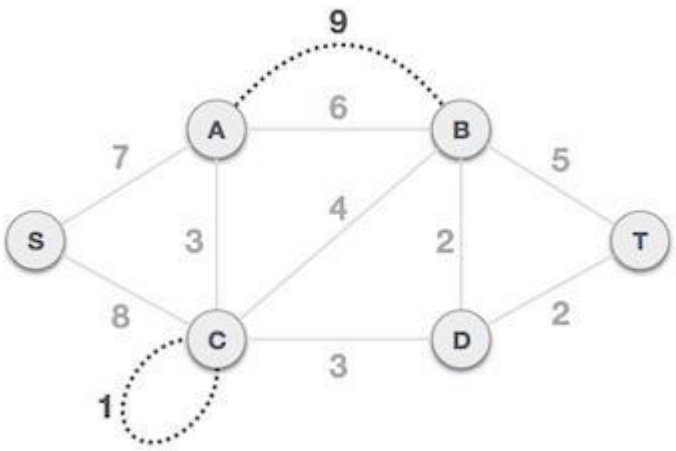
= 99 units

Example 3:

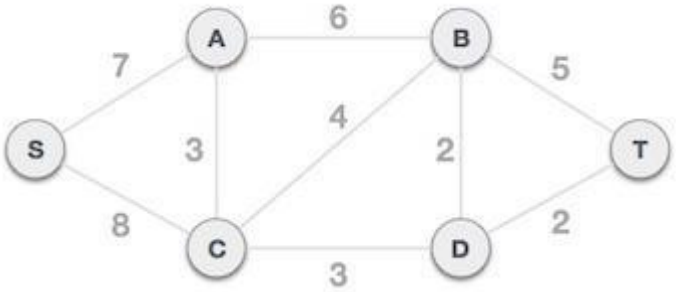


Step 1 - Remove all loops and Parallel Edges

Remove all loops and parallel edges from the given graph.



In case of parallel edges, keep the one which has the least cost associated and remove all others.



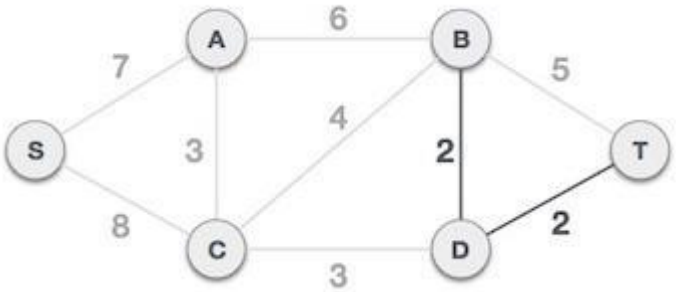
Step 2 - Arrange all edges in their increasing order of weight

The next step is to create a set of edges and weight, and arrange them in an ascending order of weightage (cost).

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

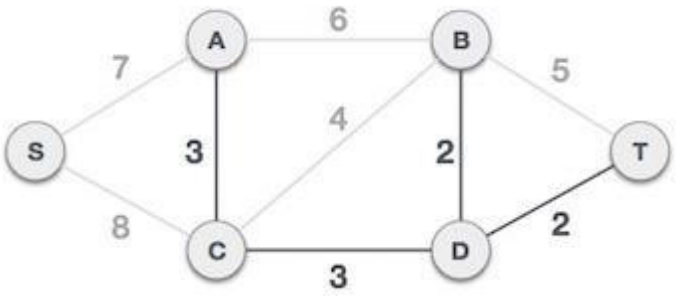
Step 3 - Add the edge which has the least weightage

Now we start adding edges to the graph beginning from the one which has the least weight. Throughout, we shall keep checking that the spanning properties remain intact. In case, by adding one edge, the spanning tree property does not hold then we shall consider not to include the edge in the graph.

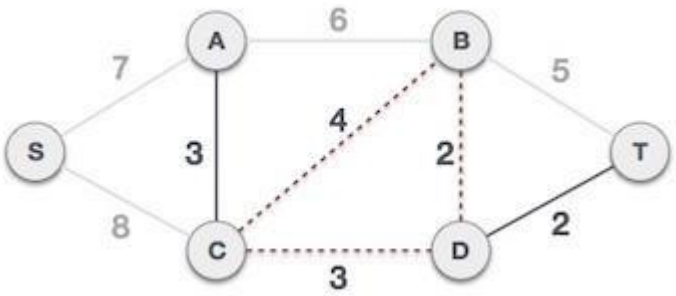


The least cost is 2 and edges involved are B,D and D,T. We add them. Adding them does not violate spanning tree properties, so we continue to our next edge selection.

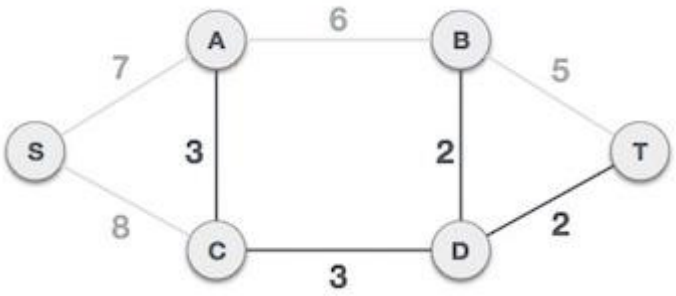
Next cost is 3, and associated edges are A,C and C,D. We add them again −



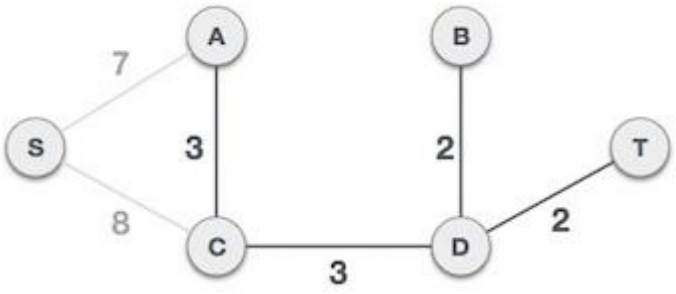
Next cost in the table is 4, and we observe that adding it will create a circuit in the graph. −



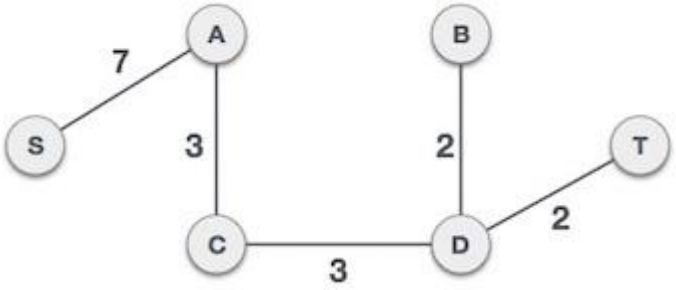
We ignore it. In the process we shall ignore/avoid all edges that create a circuit.



We observe that edges with cost 5 and 6 also create circuits. We ignore them and move on.



Now we are left with only one node to be added. Between the two least cost edges available 7 and 8, we shall add the edge with cost 7.



By adding edge S,A we have included all the nodes of the graph and we now have minimum cost spanning tree.

Prim's Algorithm (Algorithm with suitable DS)

- It starts with an empty spanning tree.
- The idea is to maintain two sets of vertices.
    - The first set contains the vertices already included in the MST,
    - the other set contains the vertices not yet included.

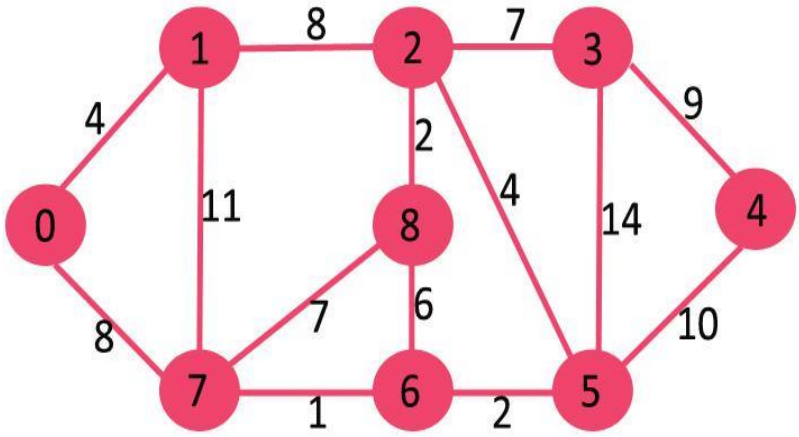
- At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. (If including that edge creates a cycle, then reject that edge and look for the next least weight edge)
- After picking the edge, it moves the other endpoint of the edge to the set containing MST.
- A group of edges that connects two set of vertices in a graph is called cut in graph theory.
    - *So, at every step of Prim's algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices),*
    - *pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).*

Algorithm

1. Create a set *mstSet* that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph.
3. Initialize all key values as INFINITE.
4. Assign key value as 0 for the first vertex so that it is picked first.
5. While mstSet doesn't include all vertices
   a) Pick a vertex *u* which is not there in *mstSet* and has minimum key value.
   b) Include *u* to mstSet.
   c) Update key value of all adjacent vertices of *u*.
    - To update the key values, iterate through all adjacent vertices.
    - For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

The idea of using key values is to pick the minimum weight edge from cut.

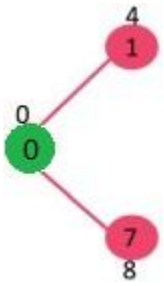
The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.



- Mstset = { }
- Key_Value = { 0, ∞,∞,∞,∞,∞,∞, ∞,∞ }
- Pick the vertex with the minimum key value.
    - The vertex 0 is picked, include it in *mstSet*.
    - So *mstSet* becomes {0}.
- Update key values of adjacent vertices.
    - Adjacent vertices of 0 are 1 and 7.
    - The key values of 1 and 7 are updated as 4 and 8.

Mstset={0} Key_value = {0,4, ∞,∞,∞,∞,∞,8, ∞}

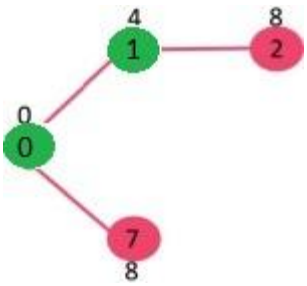
Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color

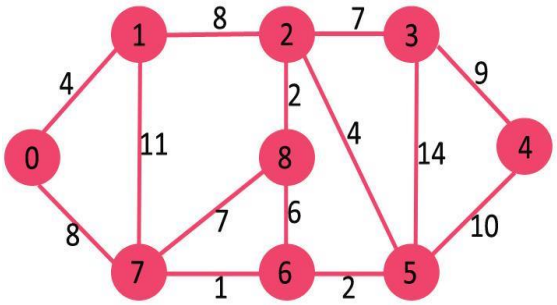


- Pick the vertex with minimum key value and if not in mstSet include(So in MST)
- The vertex 1 is picked and added to mstSet.
- mstSet= {0, 1}

- Update the key values of adjacent vertices of 1.
- The key value of vertex 2 becomes 8.

• Mstset={0,1} • Key_value = {0,4, 8,∞,∞,∞,∞,8, ∞}





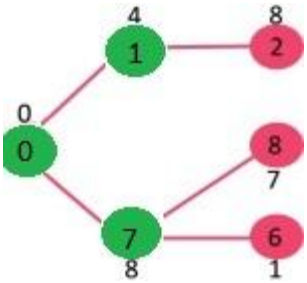
We can either pick vertex 7 or vertex 2,

let vertex 7 is picked.

mstSet= {0, 1, 7}.

Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (1 and 7 respectively).

Mstset={0,1,7} Key_value = {0,4, 8,∞,∞,∞,1,8,7}



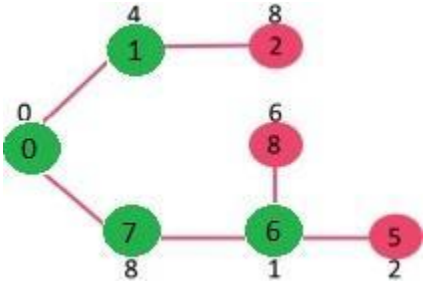
Pick the vertex with minimum key value and not already included in MST (not in mstSET).

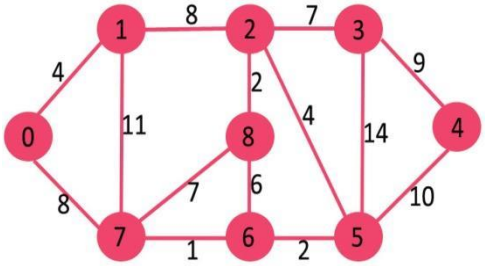
Vertex 6 is picked.

So mstSet now becomes {0, 1, 7, 6}.

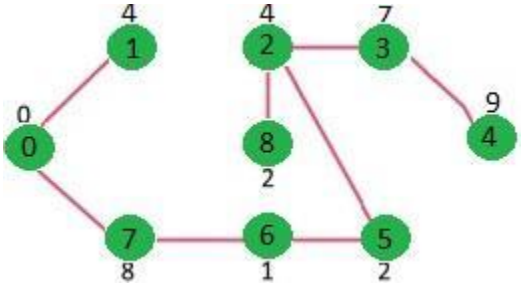
Update the key values of adjacent vertices of 6. The key value of vertex 5 and 8 are updated.

Mstset={0,1,7,6} Key_value = {0,4, 8,∞,∞,2,1,8,6} Edge_list = {01,07,76….}

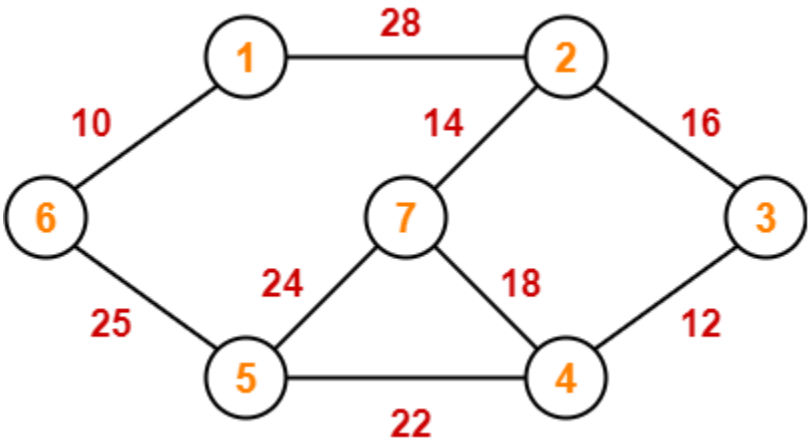




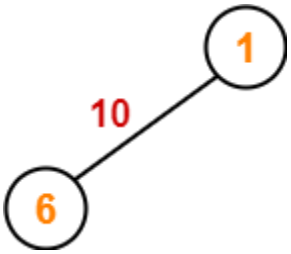
We repeat the above steps until *mstSet* includes all vertices of given graph. Finally, we get the following graph.



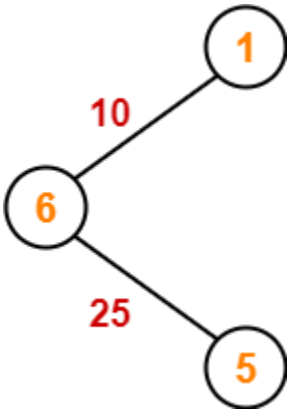
Example 2:



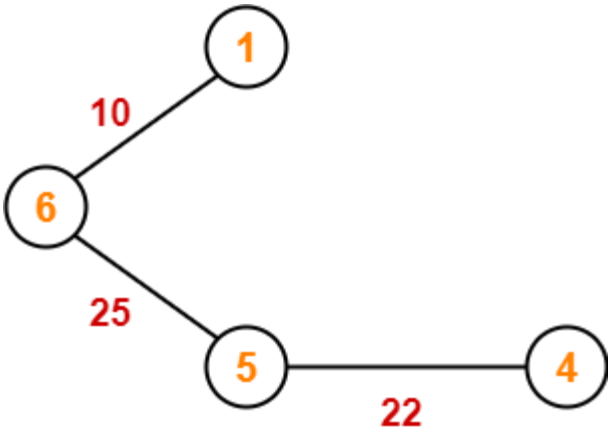
Step-01:



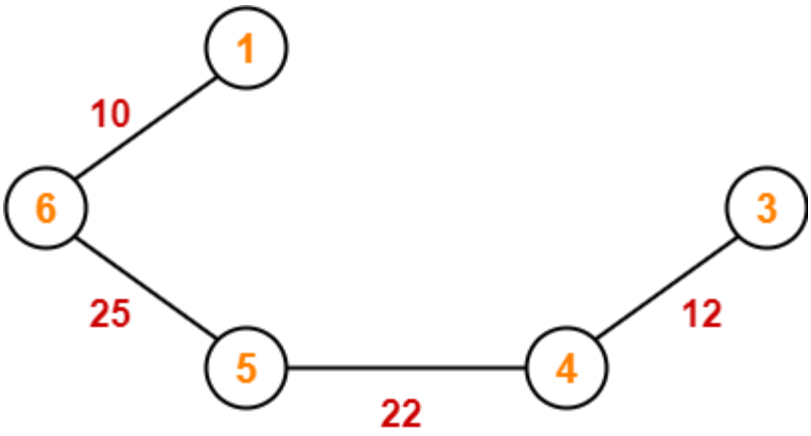
Step-02:



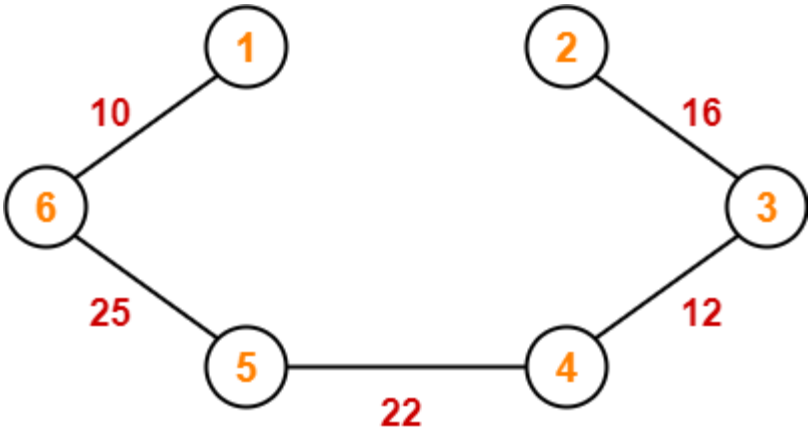
Step-03:



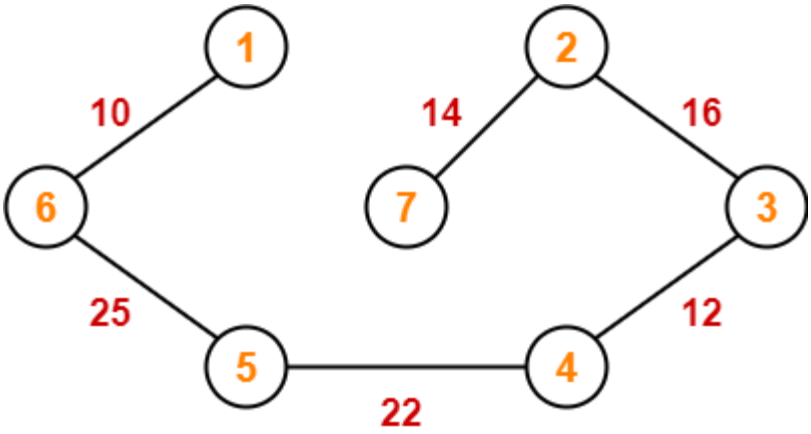
Step-04:



Step-05:



Step-06:

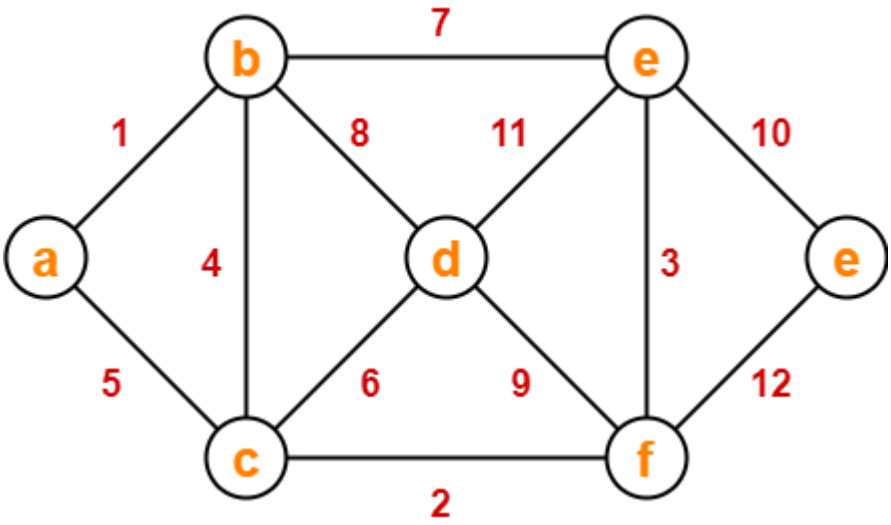


Since all the vertices have been included in the MST, so we stop.

Now, Cost of Minimum Spanning Tree
= Sum of all edge weights
= 10 + 25 + 22 + 12 + 16 + 14
= 99 units

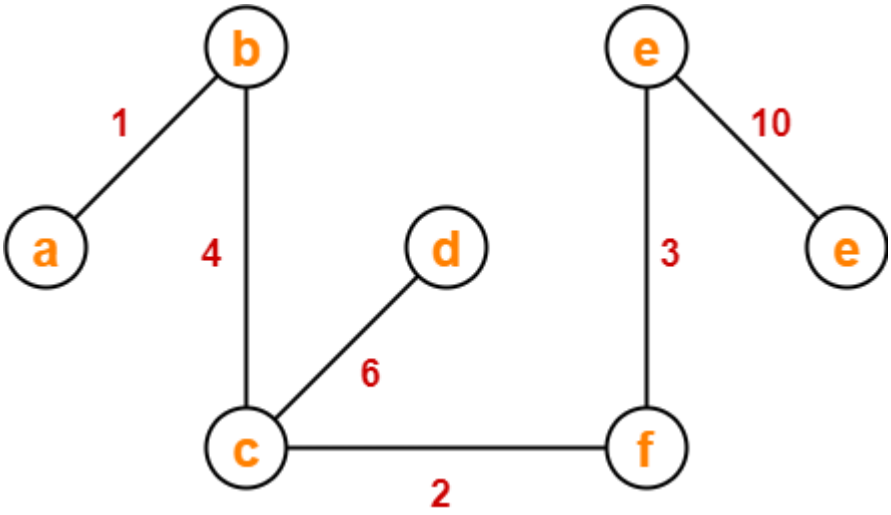
Example 3:

Using Prim's Algorithm, find the cost of minimum spanning tree (MST) of the given graph-



Solution-

The minimum spanning tree obtained by the application of Prim's Algorithm on the given graph is as shown below-



Now, Cost of Minimum Spanning Tree

= Sum of all edge weights

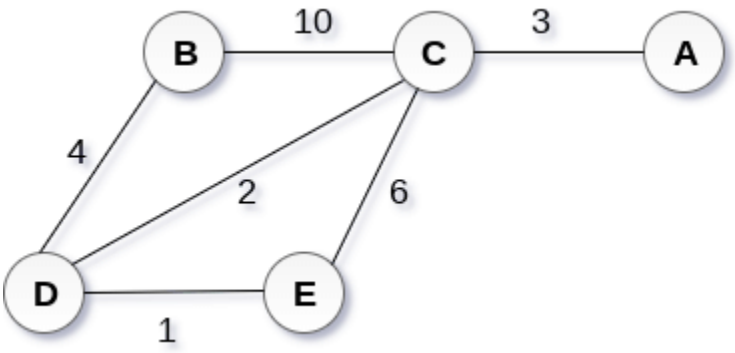
= 1 + 4 + 2 + 6 + 3 + 10

= 26 units

Example 4:

Example :

Construct a minimum spanning tree of the graph given in the following figure by using prim's algorithm.



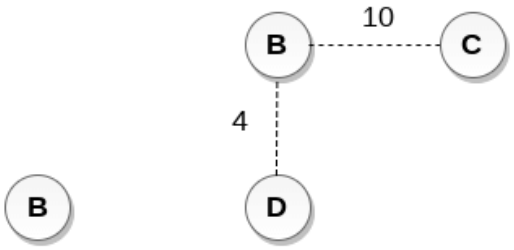
Solution

- **Step 1 :** Choose a starting vertex B.
- **Step 2:** Add the vertices that are adjacent to A. the edges that connecting the vertices are shown by dotted lines.
- **Step 3:** Choose the edge with the minimum weight among all. i.e. BD and add it to MST. Add the adjacent vertices of D i.e. C and E.
- **Step 3:** Choose the edge with the minimum weight among all. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C i.e. E and A.
- **Step 4:** Choose the edge with the minimum weight i.e. CA. We can't choose CE as it would cause cycle in the graph.

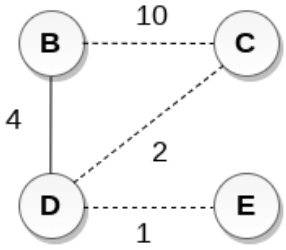
The graph produces in the step 4 is the minimum spanning tree of the graph shown in the above figure.

The cost of MST will be calculated as;

cost(MST) = 4 + 2 + 1 + 3 = 10 units.

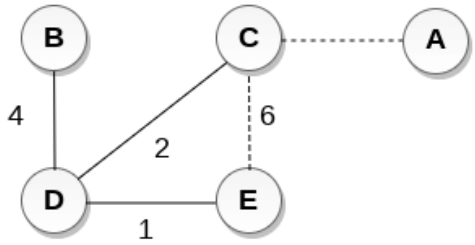


Step 1

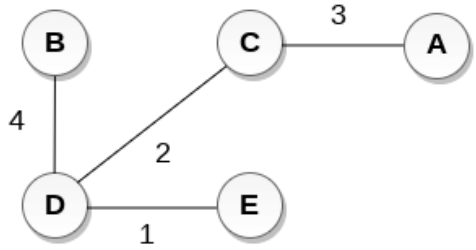


Step 2

Step 3



Step 4



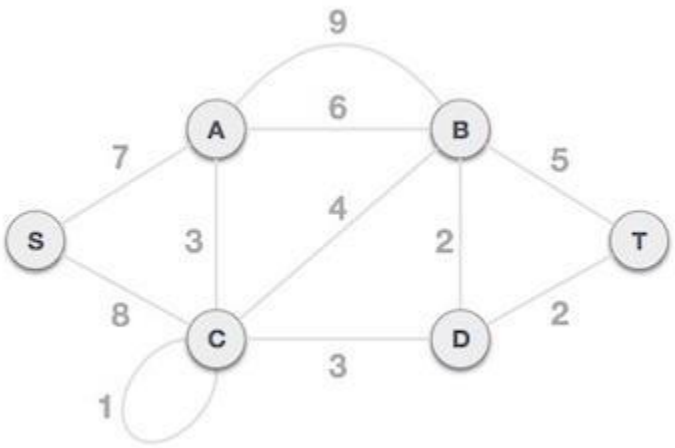
Step 5

Example 5:

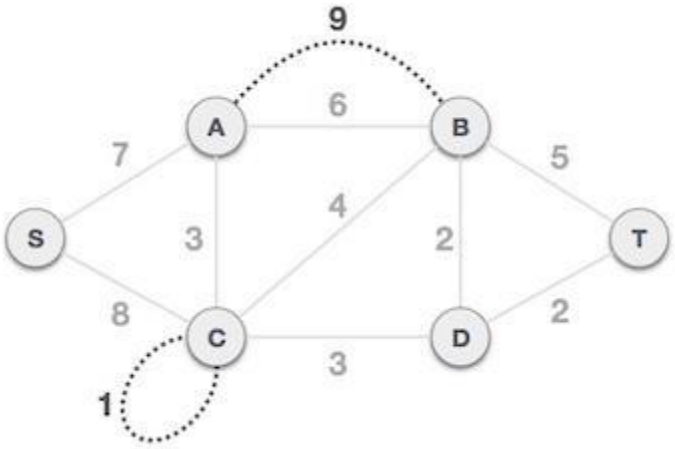
Prim's algorithm to find minimum cost spanning tree (as Kruskal's algorithm) uses the greedy approach. Prim's algorithm shares a similarity with the **shortest path first** algorithms.

Prim's algorithm, in contrast with Kruskal's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.

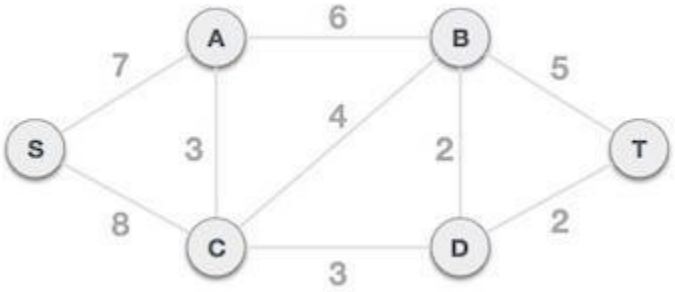
To contrast with Kruskal's algorithm and to understand Prim's algorithm better, we shall use the same example −



Step 1 - Remove all loops and parallel edges



Remove all loops and parallel edges from the given graph. In case of parallel edges, keep the one which has the least cost associated and remove all others.

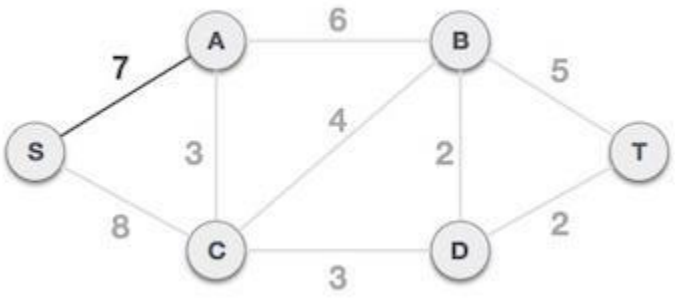


Step 2 - Choose any arbitrary node as root node

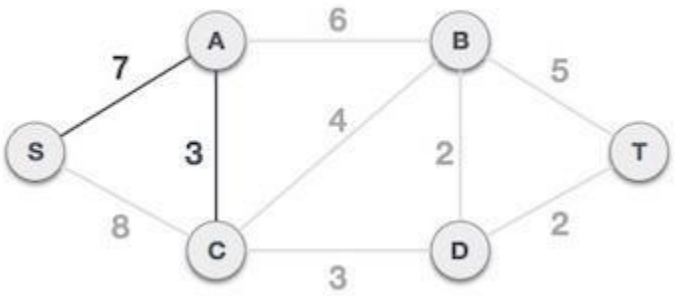
In this case, we choose **S** node as the root node of Prim's spanning tree. This node is arbitrarily chosen, so any node can be the root node. One may wonder why any video can be a root node. So the answer is, in the spanning tree all the nodes of a graph are included and because it is connected then there must be at least one edge, which will join it to the rest of the tree.

Step 3 - Check outgoing edges and select the one with less cost

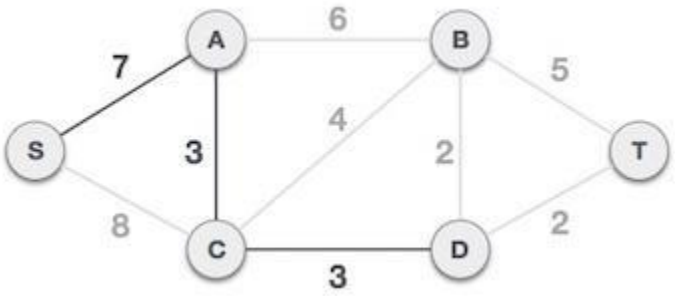
sAfter choosing the root node **S**, we see that S,A and S,C are two edges with weight 7 and 8, respectively. We choose the edge S,A as it is lesser than the other.



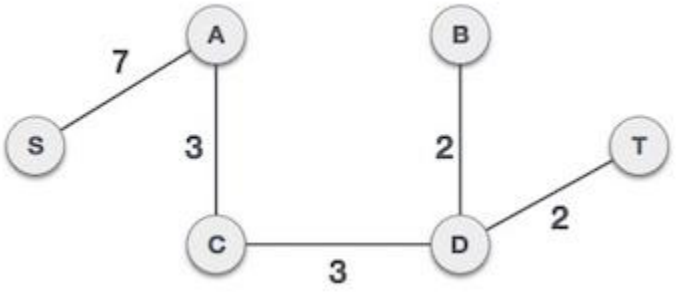
Now, the tree S-7-A is treated as one node and we check for all edges going out from it. We select the one which has the lowest cost and include it in the tree.



After this step, S-7-A-3-C tree is formed. Now we'll again treat it as a node and will check all the edges again. However, we will choose only the least cost edge. In this case, C-3-D is the new edge, which is less than other edges' cost 8, 6, 4, etc.



After adding node **D** to the spanning tree, we now have two edges going out of it having the same cost, i.e. D-2-T and D-2-B. Thus, we can add either one. But the next step will again yield edge 2 as the least cost. Hence, we are showing a spanning tree with both edges included.


A
B
7
S
3
2
T
2
C
D
3