**AIM: FIND THE MINIMUM SPANNING TREE(MST) USING**

1. **KRUSKAL’S ALGORITHM**
2. **PRIM’S ALGORITHM**

**THEORY:**

A minimum spanning tree (MST) or minimum weight spanning tree 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. That is, it is a spanning tree whose sum of edge weights is as small as possible. More generally, any edge-weighted undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of the minimum spanning trees for its connected components.

There are many use cases for minimum spanning trees. One example is a telecommunications company trying to lay cable in a new neighborhood. If it is constrained to bury the cable only along certain paths (e.g. roads), then there would be a graph containing the points (e.g. houses) connected by those paths. Some of the paths might be more expensive, because they are longer, or require the cable to be buried deeper; these paths would be represented by edges with larger weights. Currency is an acceptable unit for edge weight – there is no requirement for edge lengths to obey normal rules of geometry such as the triangle inequality. A spanning tree for that graph would be a subset of those paths that has no cycles but still connects every house; there might be several spanning trees possible. A minimum spanning tree would be one with the lowest total cost, representing the least expensive path for laying the cable.

1. **Kruskal's Algorithm**

Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

* form a tree that includes every vertex
* has the minimum sum of weights among all the trees that can be formed from the graph

**How Kruskal's algorithm works**

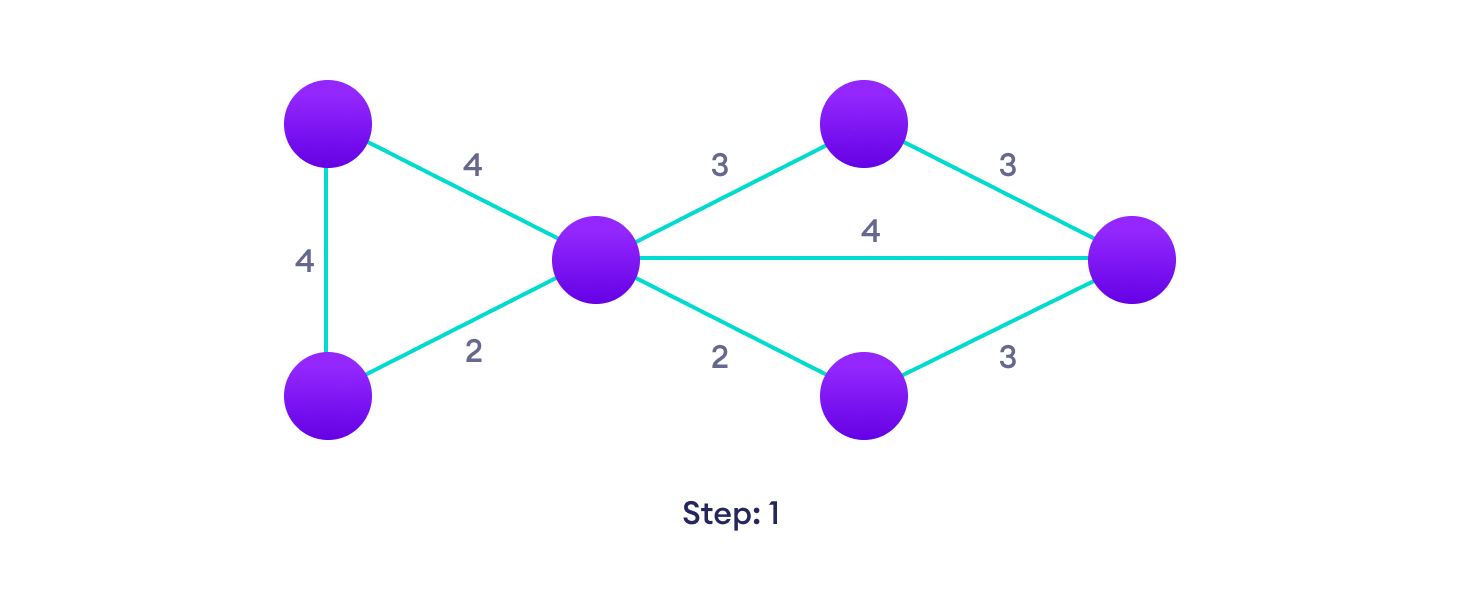
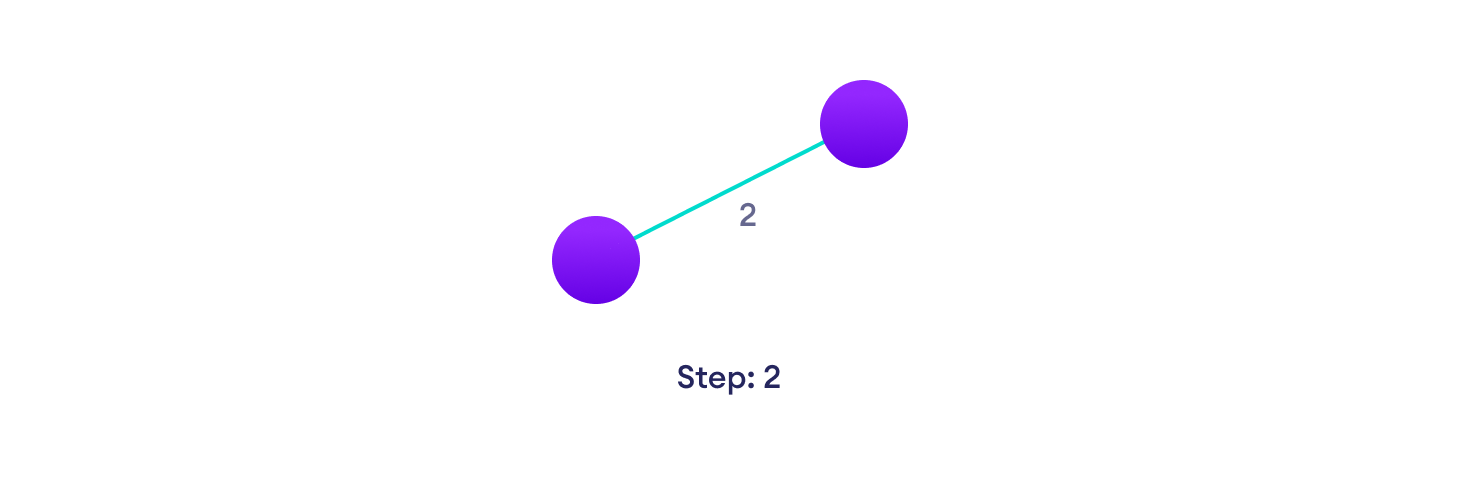
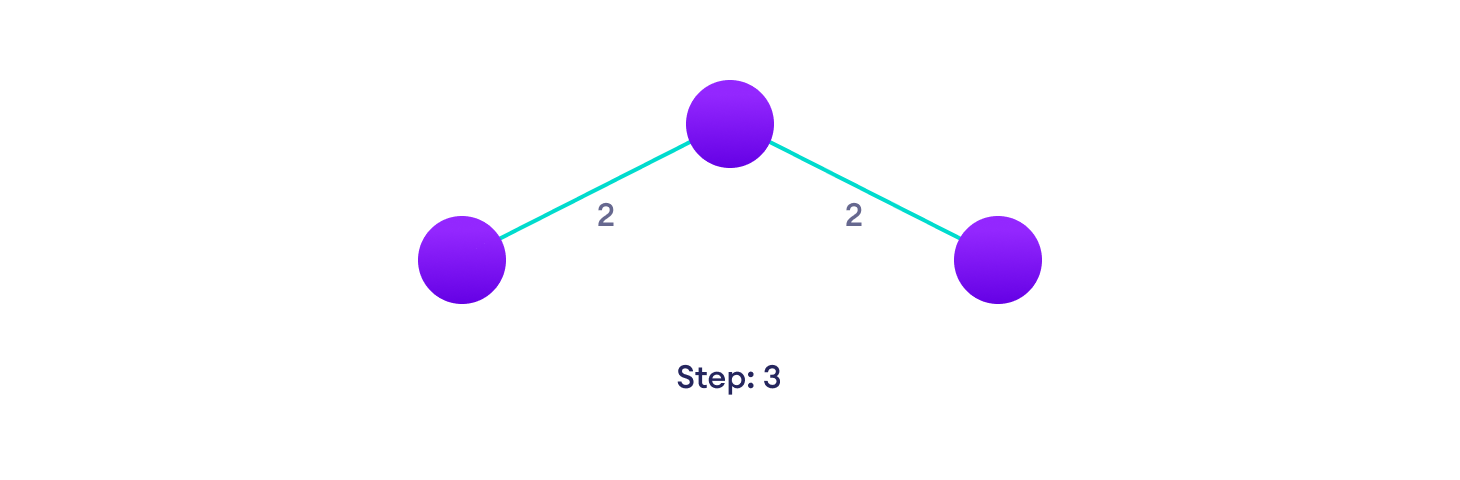
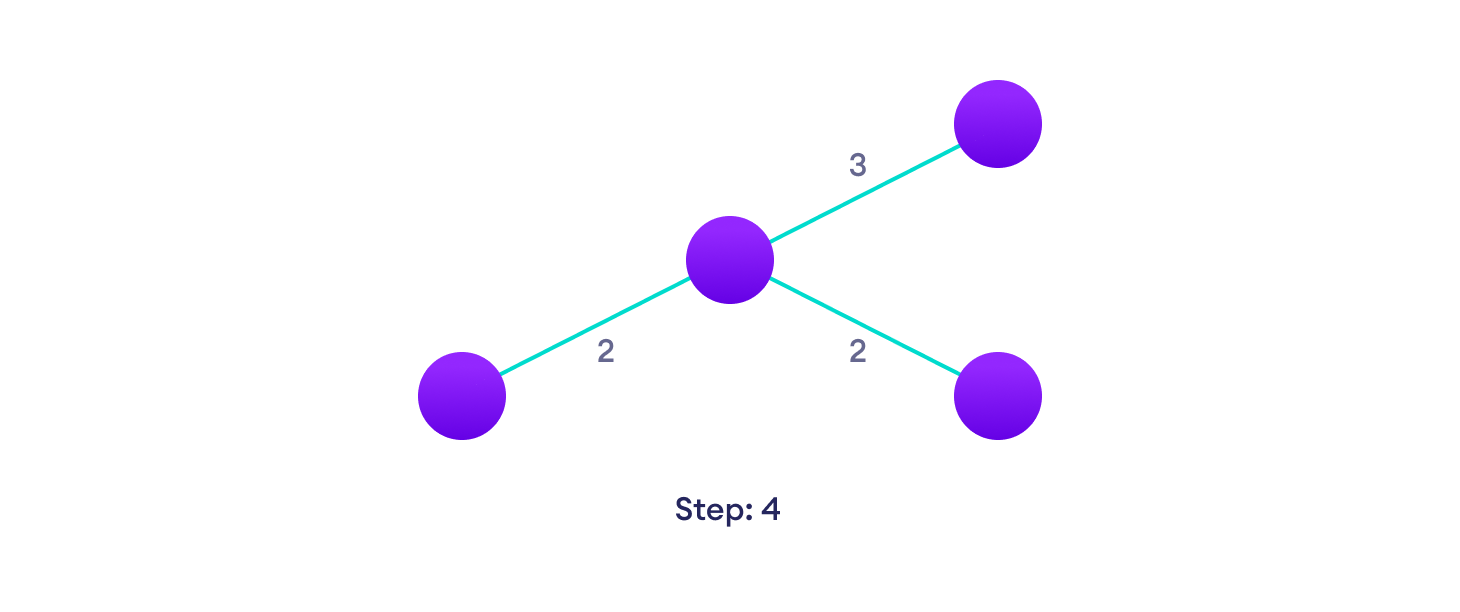
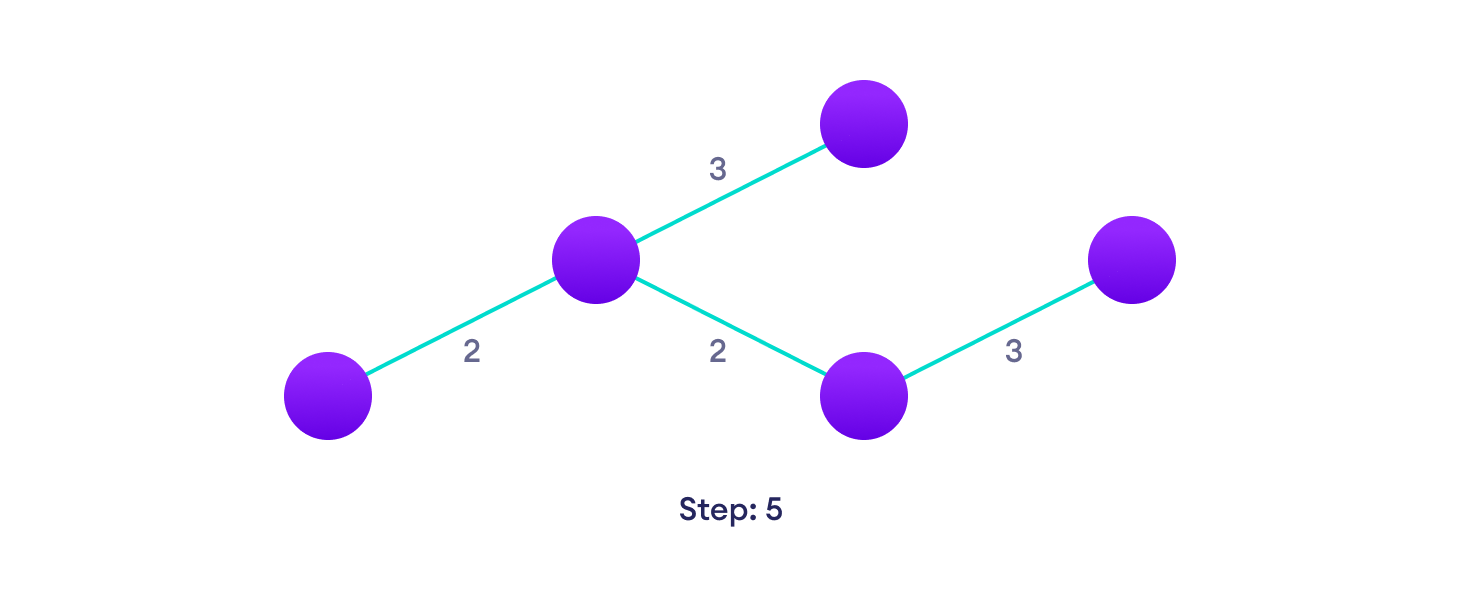
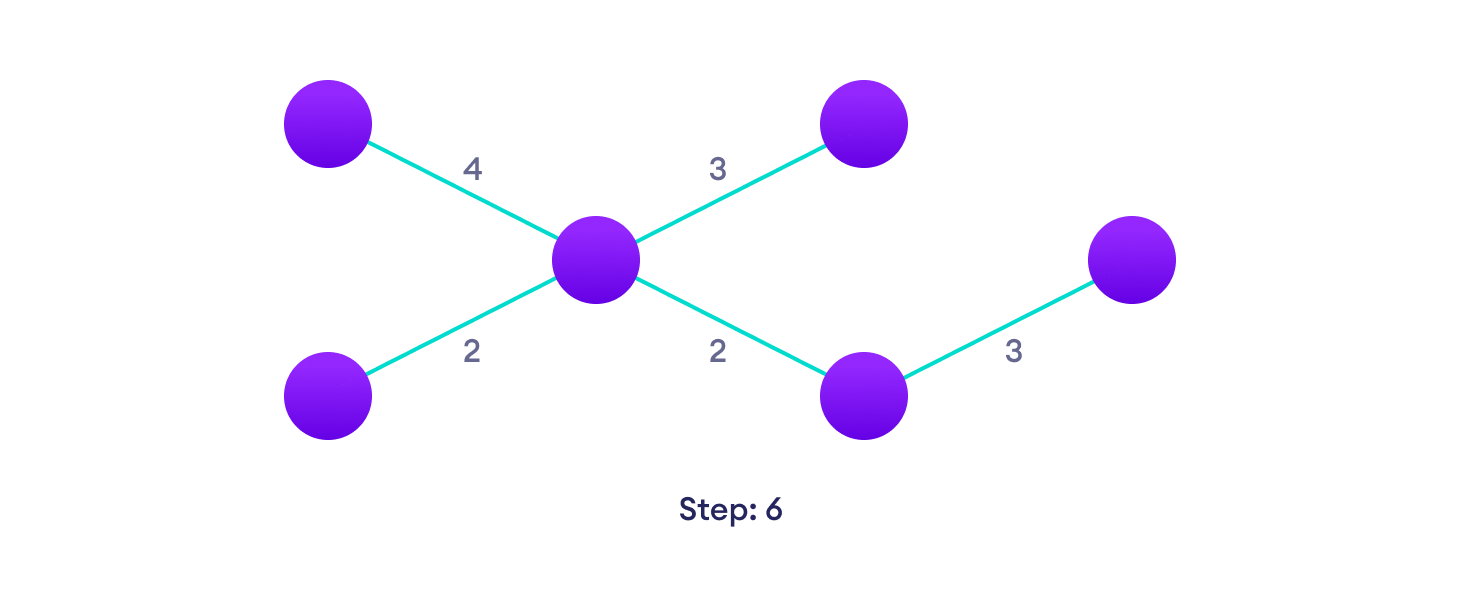
It falls under a class of algorithms called [greedy algorithms](http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Greedy/greedyIntro.htm) that find the local optimum in the hopes of finding a global optimum.

We start from the edges with the lowest weight and keep adding edges until we reach our goal.

The steps for implementing Kruskal's algorithm are as follows:

1. Sort all the edges from low weight to high
2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

**Example of Kruskal's algorithm**

Start with a weighted graphChoose the edge with the least weight, if there are more than 1, choose anyoneChoose the next shortest edge and add itChoose the next shortest edge that doesn't create a cycle and add itChoose the next shortest edge that doesn't create a cycle and add itRepeat until you have a spanning tree

**Kruskal Algorithm Pseudocode**

Any minimum spanning tree algorithm revolves around checking if adding an edge creates a loop or not.

The most common way to find this out is an algorithm called Union FInd. The Union-Find algorithm divides the vertices into clusters and allows us to check if two vertices belong to the same cluster or not and hence decide whether adding an edge creates a cycle.

KRUSKAL(G):

A = ∅

For each vertex v ∈ G.V:

MAKE-SET(v)

For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):

if FIND-SET(u) ≠ FIND-SET(v):

A = A ∪ {(u, v)}

UNION(u, v)

return A

## Kruskal's vs Prim's Algorithm

Prim's algorithm is another popular minimum spanning tree algorithm that uses a different logic to find the MST of a graph. Instead of starting from an edge, Prim's algorithm starts from a vertex and keeps adding lowest-weight edges which aren't in the tree, until all vertices have been covered.

## Kruskal's Algorithm Complexity

The time complexity Of Kruskal's Algorithm is: O(E log E).

## Kruskal's Algorithm Applications

* In order to layout electrical wiring
* In computer network (LAN connection)

# Prim's Algorithm

In this tutorial, you will learn how Prim's Algorithm works. Also, you will find working examples of Prim's Algorithm in C, C++, Java and Python.

Prim's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

* form a tree that includes every vertex
* has the minimum sum of weights among all the trees that can be formed from the graph

## How Prim's algorithm works

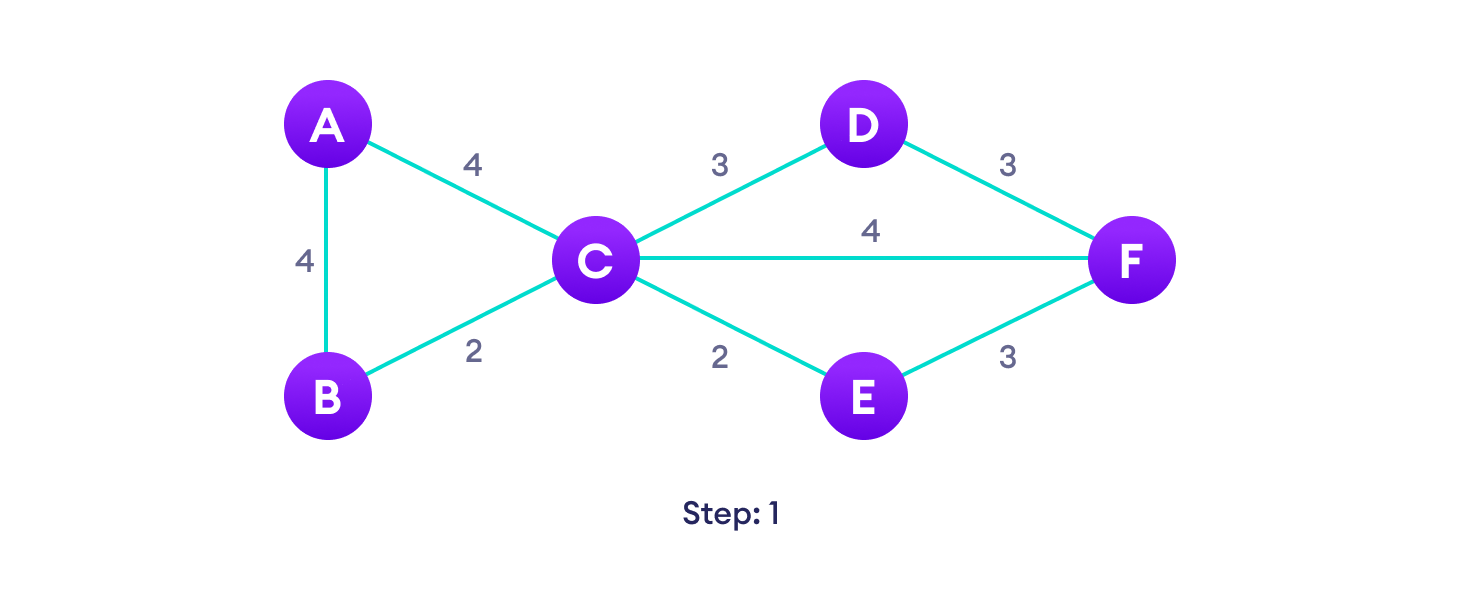
It falls under a class of algorithms called [greedy algorithms](https://www.programiz.com/dsa/greedy-algorithm) that find the local optimum in the hopes of finding a global optimum.

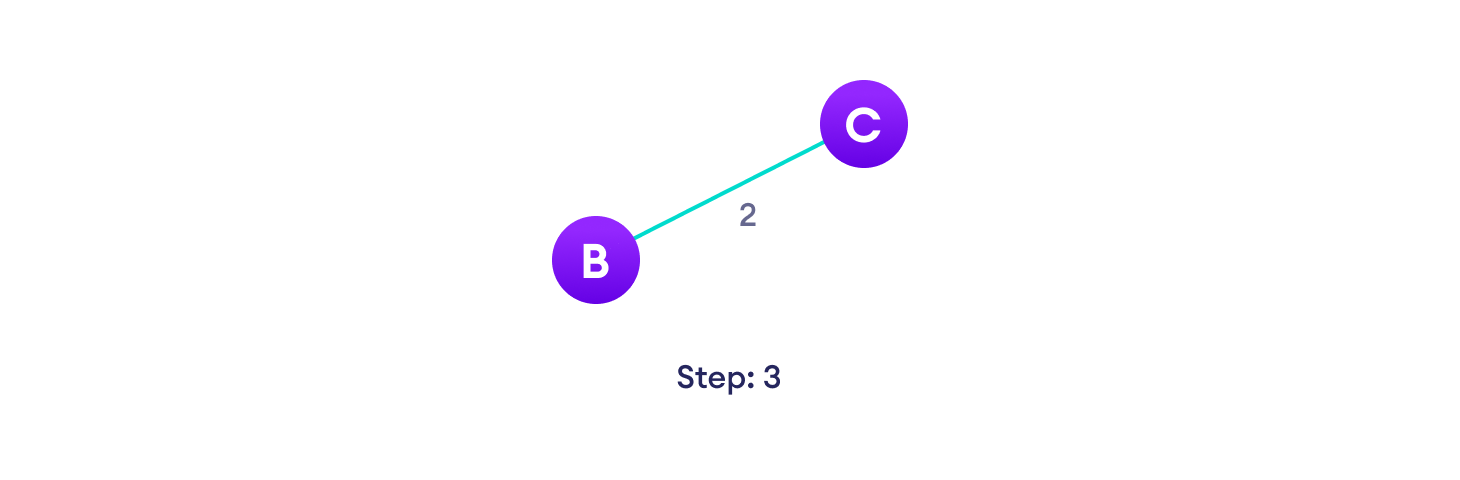
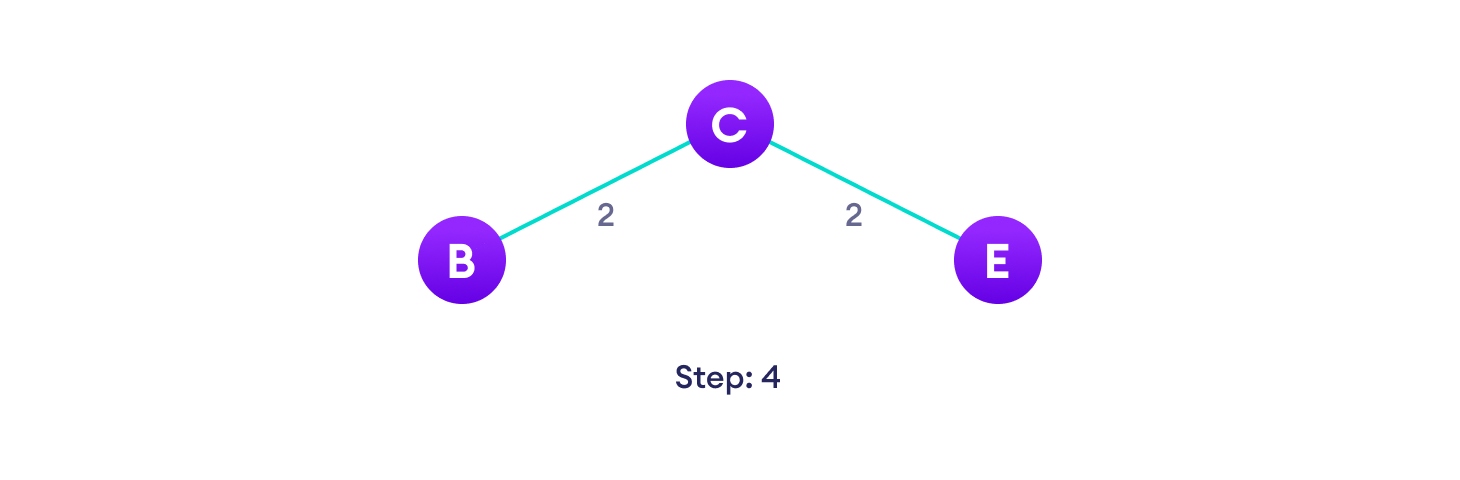
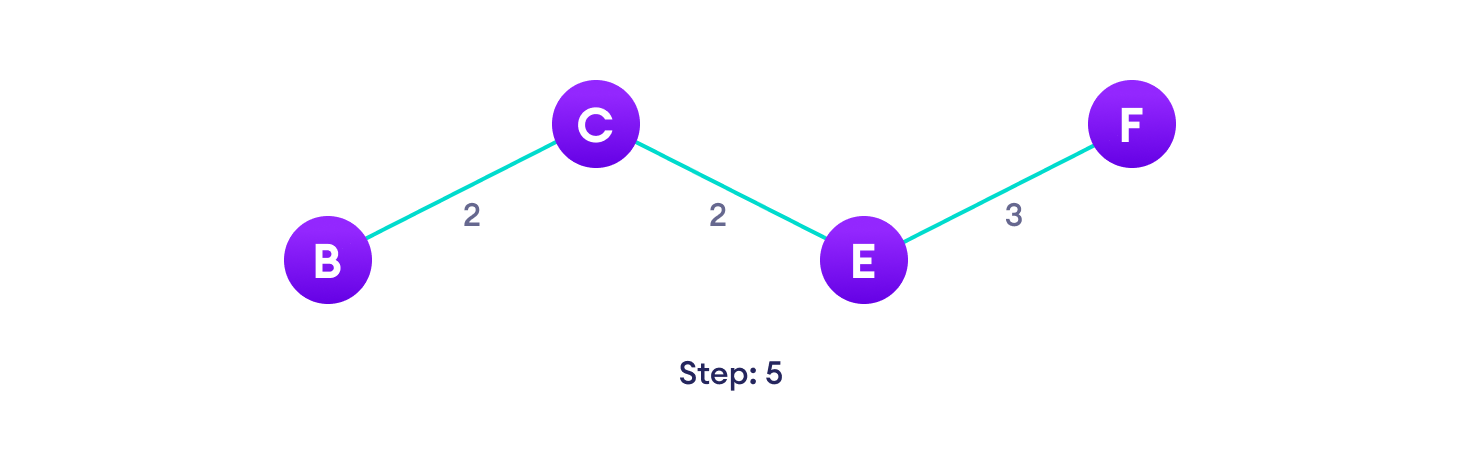
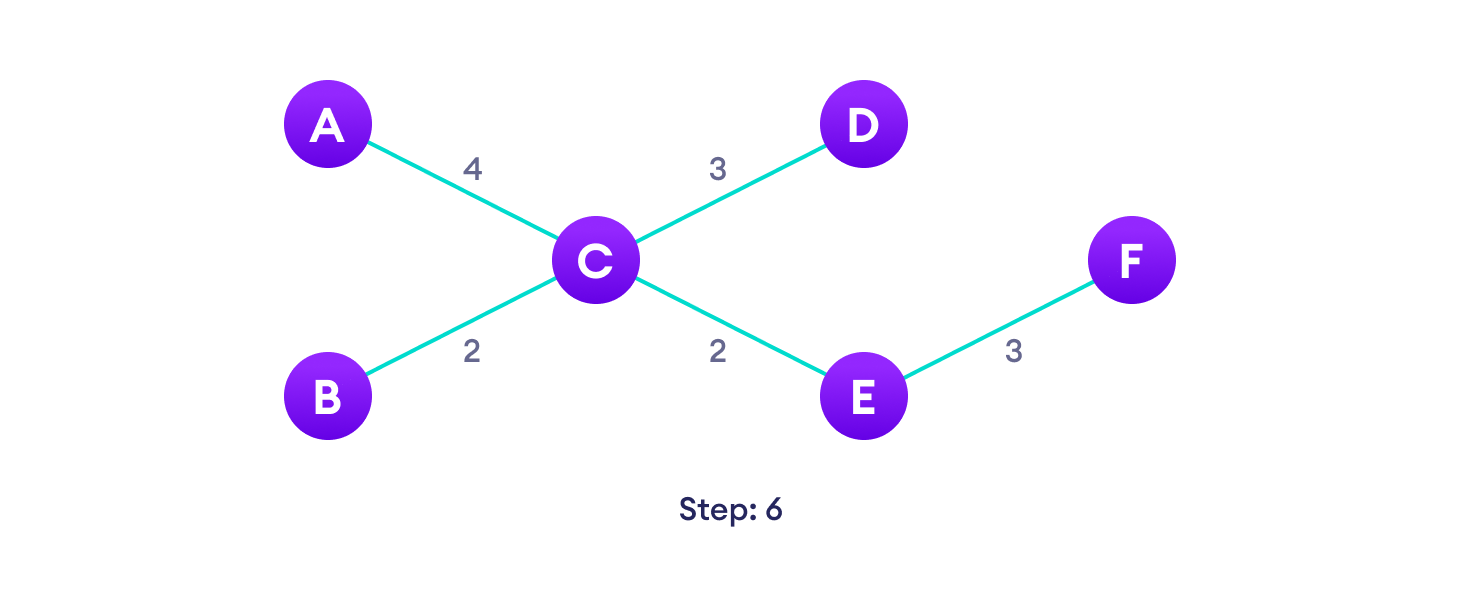
We start from one vertex and keep adding edges with the lowest weight until we reach our goal.

The steps for implementing Prim's algorithm are as follows:

1. Initialize the minimum spanning tree with a vertex chosen at random.
2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
3. Keep repeating step 2 until we get a minimum spanning tree

## Example of Prim's algorithm



Start with a weighted graphChoose a vertexChoose the shortest edge from this vertex and add itChoose the nearest vertex not yet in the solutionChoose the nearest edge not yet in the solution, if there are multiple choices, choose one at randomRepeat until you have a spanning tree

## Prim's Algorithm pseudocode

The pseudocode for prim's algorithm shows how we create two sets of vertices U and V-U. U contains the list of vertices that have been visited and V-U the list of vertices that haven't. One by one, we move vertices from set V-U to set U by connecting the least weight edge.

T = ∅;

U = { 1 };

while (U ≠ V)

let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;

T = T ∪ {(u, v)}

U = U ∪ {v}

## Prim's vs Kruskal's Algorithm

Kruskal's algorithm is another popular minimum spanning tree algorithm that uses a different logic to find the MST of a graph. Instead of starting from a vertex, Kruskal's algorithm sorts all the edges from low weight to high and keeps adding the lowest edges, ignoring those edges that create a cycle.

## Prim's Algorithm Complexity

The time complexity of Prim's algorithm is O(E log V).

## Prim's Algorithm Application

* Laying cables of electrical wiring
* In network designed
* To make protocols in network cycles
  1. **KRUSKAL'S ALGORITHM**

**SOURCE CODE:**

#include<iostream>

#include <algorithm>

#include<vector>

using namespace std;

class edge

{

public:

int s;

int d;

int w;

edge()

{

}

edge(int src,int des,int wei)

{

s=src;

d=des;

w=wei;

}

};

bool compare(edge e1,edge e2)

{

return e1.w<e2.w;

}

int findparent(int i,int\* parent )

{

if(parent[i]==i)

{

return i;

}

return findparent(parent[i],parent);

}

class graph

{

public:

int e,n;

edge\* v;

graph(int n,int e)

{

this->n=n;

this->e=e;

v=new edge[e];

for(int i=0; i<e; i++)

{

int x,y,w;

cout<<"ENTER VERTICES AND WEIGHT OF EDGE "<<i+1<<" : ";

cin>>x>>y>>w;

edge e(x,y,w);

v[i]=e;

}

}

edge\* unionfind()

{

int\* parent=new int[n];

for(int i=0; i<n; i++)

{

parent[i]=i;

}

sort(v,v+e,compare);

edge\* output;

output=new edge[n-1];

int count=0,i=0;

while(count!=n-1)

{

edge c=v[i];

int sourceparent=findparent(v[i].s,parent);

int desparent=findparent(v[i].d,parent);

if(sourceparent!=desparent)

{

output[count]=c;

parent[sourceparent]=desparent;

count++;

}

i++;

}

int sum=0;

cout<<endl<<"-------MST-------\n";

for(int i=0; i<n-1; i++)

{

cout<<output[i].s<<" "<<output[i].d<<" "<<output[i].w<<endl;

sum+=output[i].w;

}

cout<<"\nWEIGHT OF MST IS "<<sum;

return output;

}

};

int main()

{

int n,e;

cout<<"KRUSKAL'S ALGORITHM\nENTER NUMBER OF VERTICES : ";

cin>>n;

cout<<"ENTER NUMBER OF EDGEES : ";

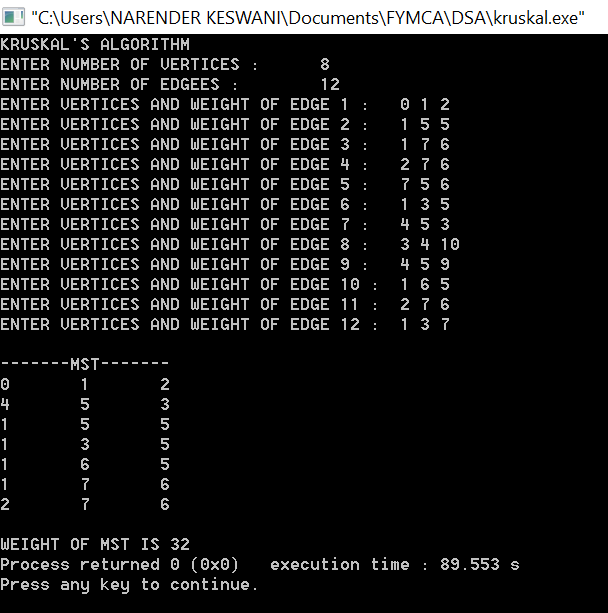
cin>>e;

graph g(n,e);

edge\* mst=g.unionfind();

}

**OUTPUT:**



* 1. **PRIM’S ALGORITHM:**

**SOURCE CODE:**

#include <bits/stdc++.h>

using namespace std;

#define V 5

int minKey(int key[], bool mstSet[])

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

{

if (mstSet[v] == false && key[v] < min)

{

min = key[v], min\_index = v;

}

}

return min\_index;

}

void printMST(int parent[], int graph[V][V])

{

int a = 0;

cout<<"Edge \tWeight\n";

for (int i = 1; i < V; i++)

{

cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<" \n";

a = a + graph[i][parent[i]];

}

cout<<"The weight of graph is: "<<a<<endl;

}

void primMST(int graph[V][V])

{

int parent[V];

int key[V];

bool mstSet[V];

for (int i = 0; i < V; i++)

{

key[i] = INT\_MAX, mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++)

{

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++)

{

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

{

parent[v] = u, key[v] = graph[u][v];

}

}

}

printMST(parent, graph);

}

int main()

{

/\* Let us create the following graph

2 3

(0)--(1)--(2)

| / \ |

6| 8/ \5 |7

| / \ |

(3)-------(4)

9 \*/

int graph[V][V] =

{ { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

{ 0, 5, 7, 9, 0 }

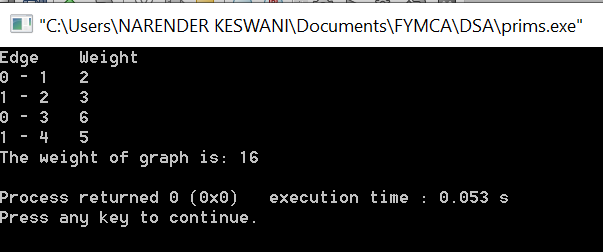
};

primMST(graph);

return 0;

}

**OUTPUT:**



**CONCLUSION:**

From this practical, I have learned how to implement minimum spanning tree using Kruskal & prims algorithms.