Minimum Spanning Tree (MST)

By

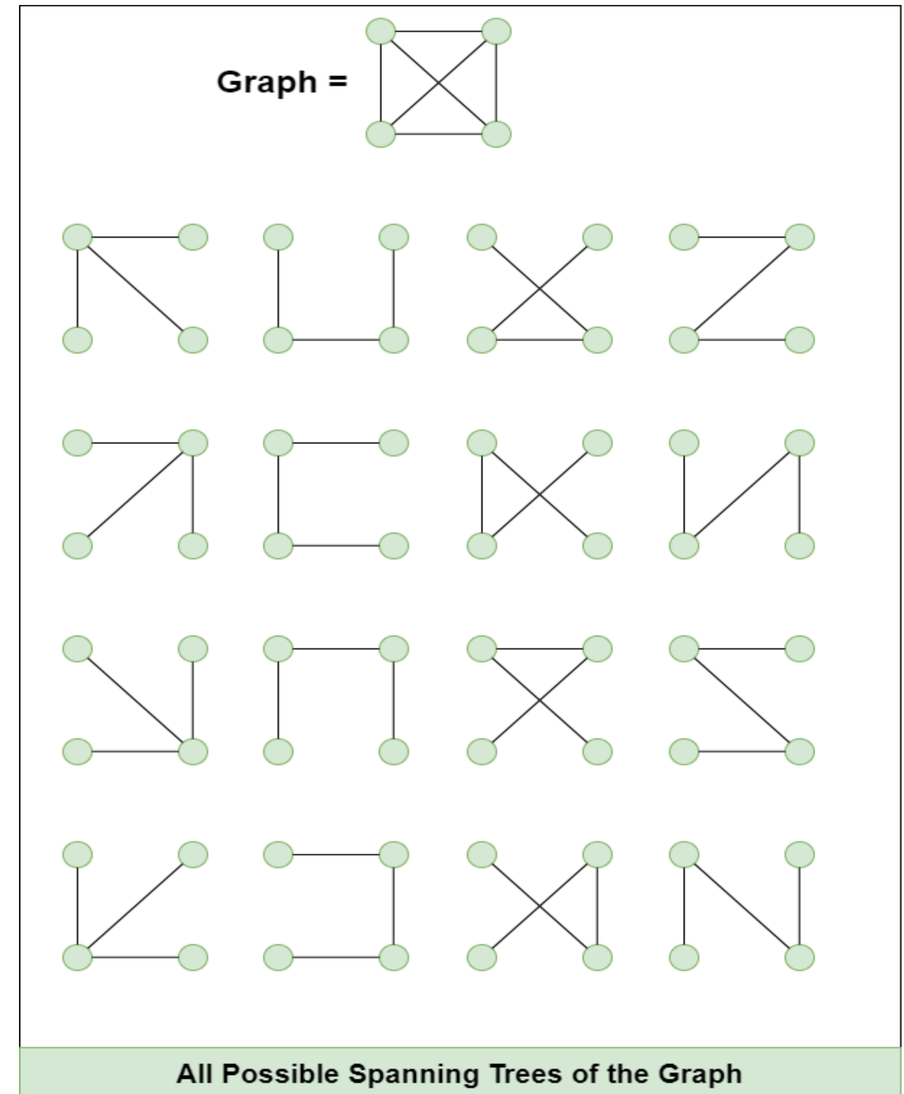
Dr. GC Jana

Minimum Spanning Tree (MST)

What is a Spanning Tree?

A spanning tree is a subset of Graph G, such that **all the vertices are connected using minimum possible number of edges**.

Hence, a spanning tree does not have cycles and a graph may have more than one spanning tree.



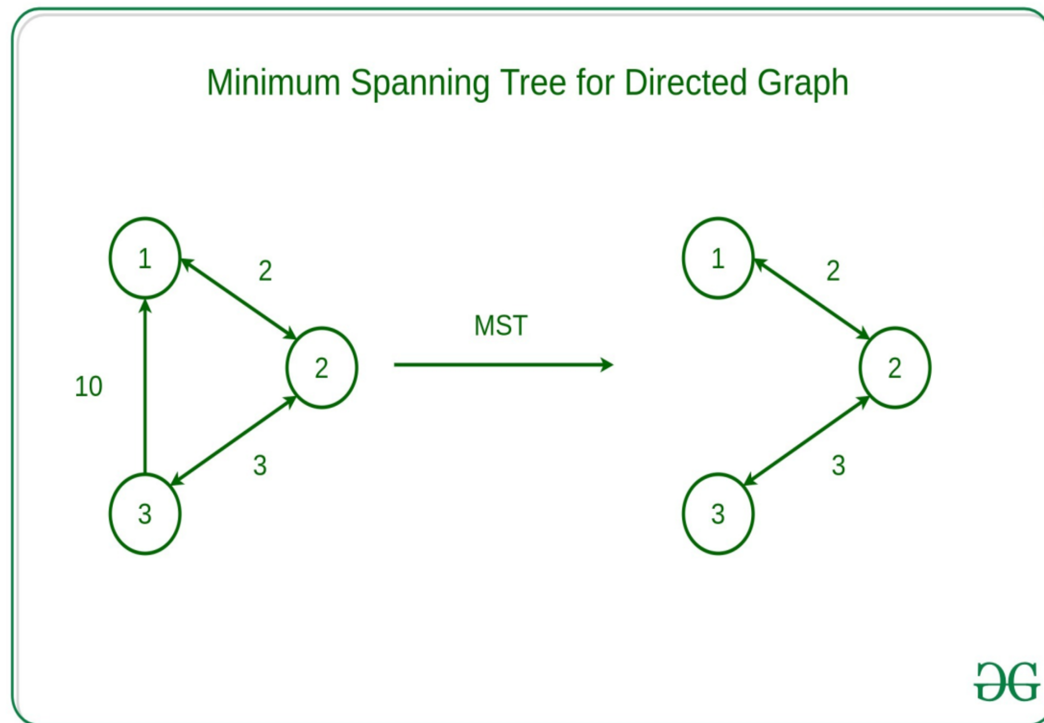
All Possible Spanning Trees of the Graph

Properties of a Spanning Tree

- A Spanning tree does not exist for a disconnected graph.
- For a connected graph having **N** vertices then the number of edges in the spanning tree for that graph will be **N-1**.
- A Spanning tree does not have any cycle.
- We can construct a spanning tree for a complete graph by removing **E-N+1** edges, where **E** is the number of Edges and **N** is the number of vertices.
- **Cayley's Formula:** It states that the number of spanning trees in a complete graph with N vertices is N^N-2
    - For example: N=4, then maximum number of spanning tree possible = 4^4-2 = 16 (shown in the above image).

What is Minimum Spanning Tree (MST)

A **minimum spanning tree (MST)** is defined as a **spanning tree that has the minimum weight among all the possible spanning trees**.



Necessary conditions for Minimum Spanning Tree:

1.It must not form a cycle i.e, no edge is traversed twice.

2.There must be no other spanning tree with lesser weight.

Properties of Minimum Spanning Tree (MST)

- A minimum spanning tree connects all the vertices in the graph, ensuring that there is a path between any pair of nodes.

- An **MST** is **acyclic**, meaning it contains no cycles. This property ensures that it remains a tree and not a graph with loops.

- An **MST** with **V** vertices (where **V** is the number of vertices in the original graph) will have exactly **V – 1** edges, where **V** is the number of vertices.

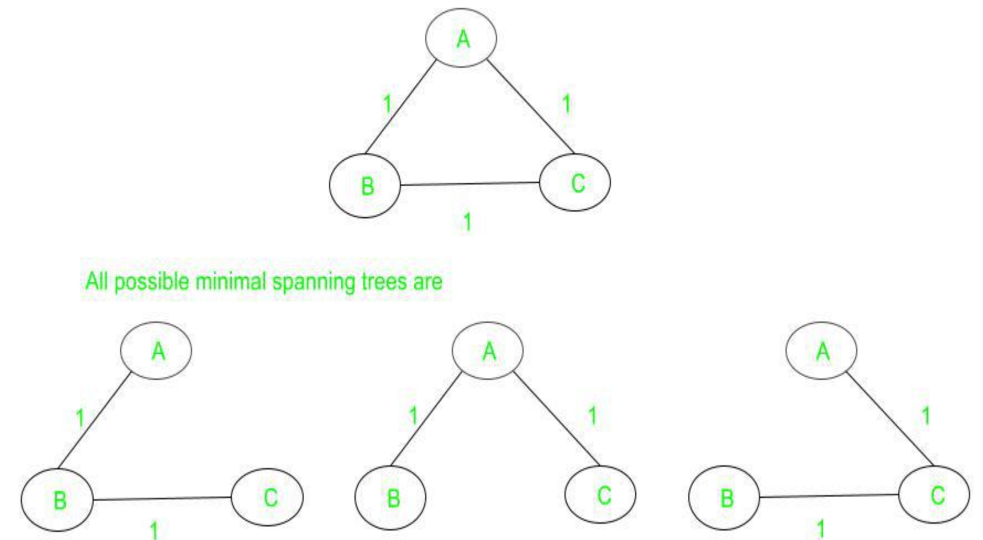
- An **MST** is optimal for minimizing the total edge weight, but it may not necessarily be unique.

- The cut property states that if you take any cut (a partition of the vertices into two sets) in the original graph and consider the minimum-weight edge that crosses the cut, that edge is part of the **MST**.

Possible Multiplicity:

If **G(V, E)** is a graph then every spanning tree of graph **G** consists of **(V – 1)** edges, where **V** is the number of vertices in the graph and **E** is the number of edges in the graph. So, **(E – V + 1)** edges are not a part of the spanning tree.

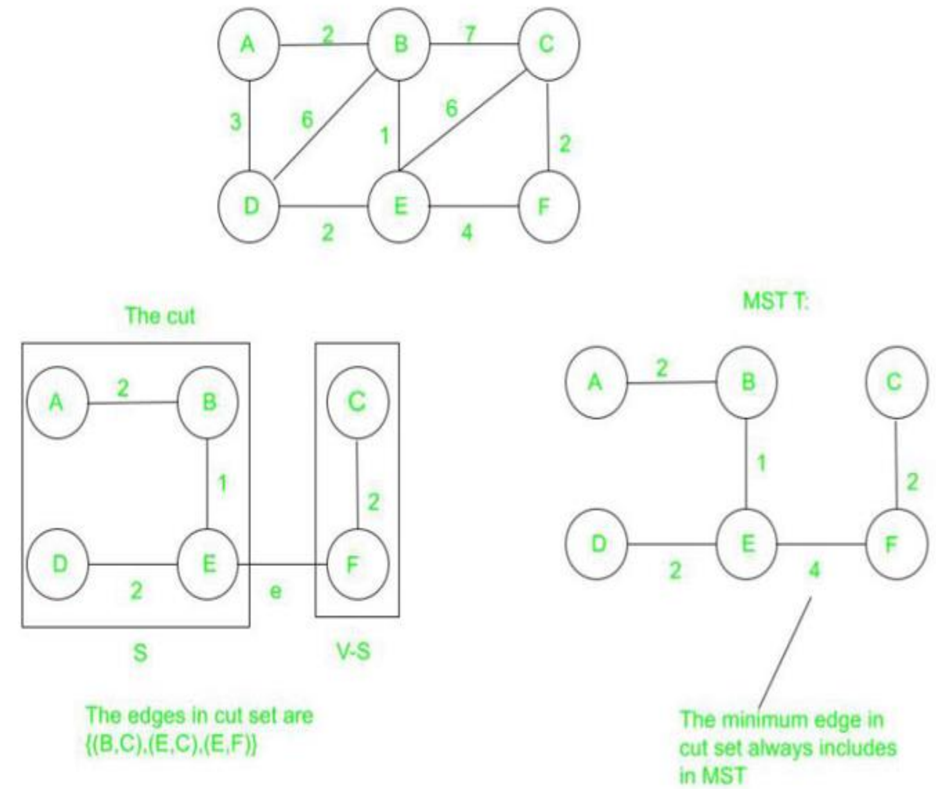
There may be several minimum spanning trees of the same weight. If all the edge weights of a graph are the same, then every spanning tree of that graph is minimum.



Each of the spanning trees has the same weight equal to 2.

Cut property:

For any cut **C** of the graph, if the weight of an edge **E** in the cut-set of **C** is strictly smaller than the weights of all other edges of the cut-set of **C**, then this edge belongs to all the MSTs of the graph. Below is the image to illustrate the same:

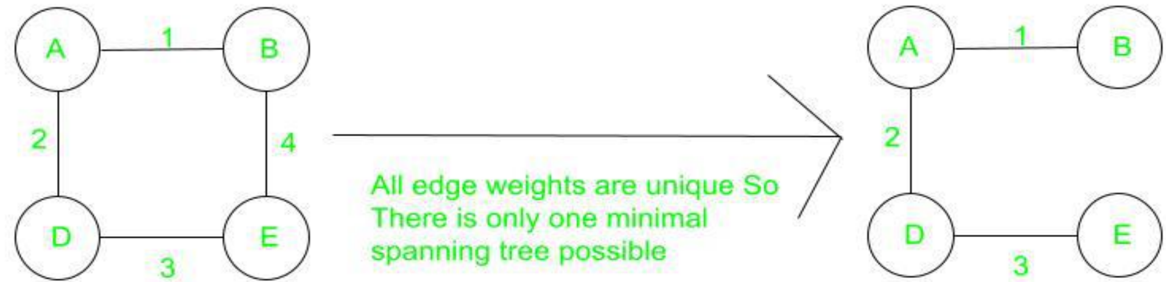


Cycle property:

For any cycle **C** in the graph, if the weight of an edge **E** of **C** is larger than the individual weights of all other edges of **C**, then this edge cannot belong to an **MST**. In the above figure, in cycle **ABD**, edge **BD** can not be present in any minimal spanning tree because it has the largest weight among all the edges in the cycle.

Uniqueness:

If each edge has a distinct weight then there will be only one, i.e., a unique minimum spanning tree.



Minimum Cost Subgraph

For all the possible spanning trees, the minimum spanning tree must have the minimum weight possible. However, there may exist some more spanning with the same weight that of minimum spanning tree, and those all may also be considered as Minimum Spanning tree.

•**Minimum Cost Edge:** If the minimum cost edge of a graph is unique, then this edge is included in any MST. For example, in the above figure, the edge **AB** (of the least weight) is always included in MST.

•If a new edge is added to the spanning tree then it will become cyclic because every spanning tree is minimally acyclic. In the above figure, if edge **AD** or **BC** is added to the resultant MST, then it will form a cycle.

•The spanning tree is minimally connected, i.e., if any edge is removed from the spanning tree it will disconnect the graph. In the above figure, if any edge is removed from the resultant MST, then it will disconnect the graph.

Algorithms for finding Minimum Spanning Tree(MST):

1. Prim’s MST Algorithm
2. Krushkal’s MST Algorithm
3. Boruvka’s Algorithm
4. Reverse-Delete Algorithm

Prim's Algorithm for Minimum Spanning Tree (MST)

https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

How does Prim's Algorithm Work?

The working of Prim's algorithm can be described by using the following steps:

Step 1: *Determine an arbitrary vertex as the starting vertex of the MST.*

Step 2: *Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).*

Step 3: *Find edges connecting any tree vertex with the fringe vertices.*

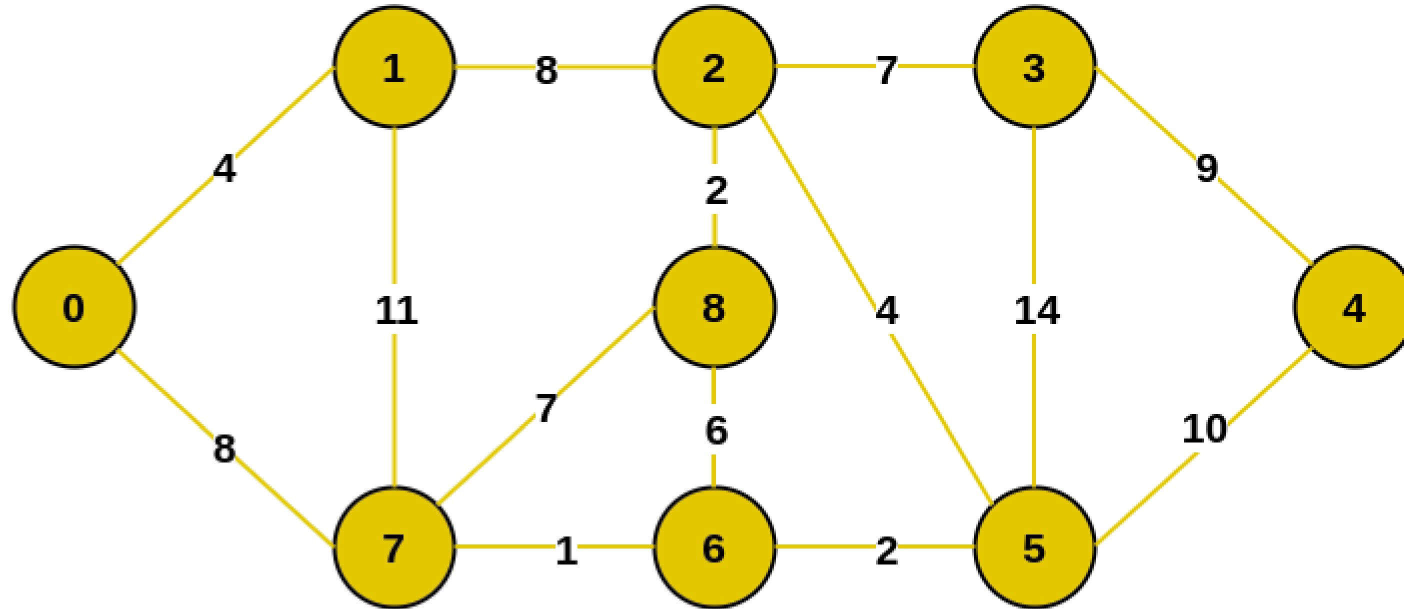
Step 4: *Find the minimum among these edges.*

Step 5: *Add the chosen edge to the MST if it does not form any cycle.*

Step 6: *Return the MST and exit*

Illustration of Prim’s Algorithm:

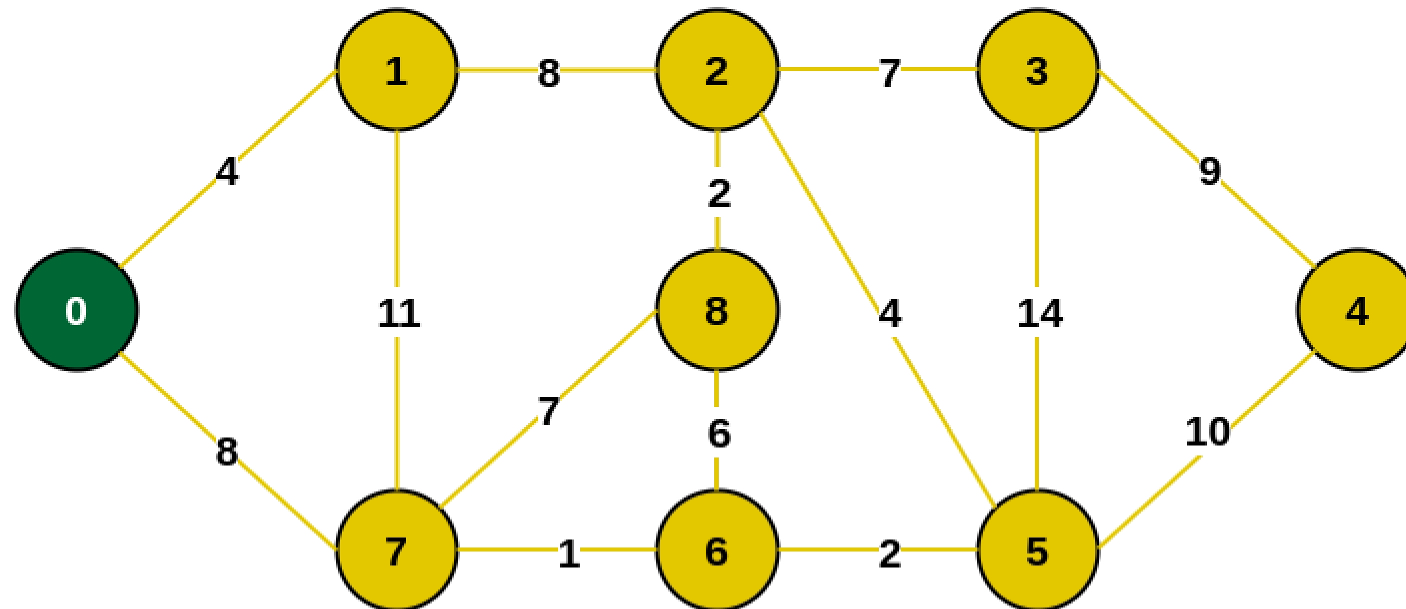
Consider the following graph as an example for which we need to find the Minimum Spanning Tree (MST).



Example of a Graph

Illustration of Prim’s Algorithm:

Step 1: *Firstly, we select an arbitrary vertex that acts as the starting vertex of the Minimum Spanning Tree. Here we have selected vertex* ***0*** *as the starting vertex.*



Select an arbitrary starting vertex. Here we have selected 0

Illustration of Prim’s Algorithm:

Step 2: *All the edges connecting the incomplete MST and other vertices are the edges {0, 1} and {0, 7}. Between these two the edge with minimum weight is {0, 1}. So include the edge and vertex 1 in the MST.*

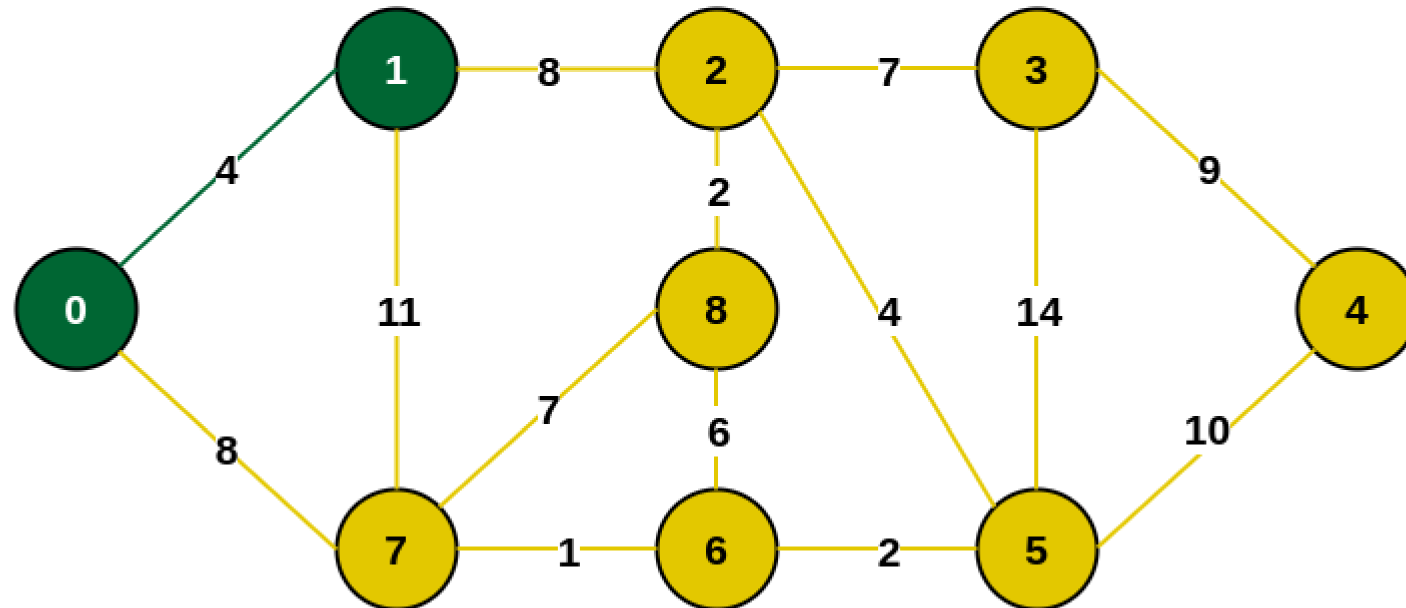


Illustration of Prim's Algorithm:

Step 3: *The edges connecting the incomplete MST to other vertices are {0, 7}, {1, 7} and {1, 2}. Among these edges the minimum weight is 8 which is of the edges {0, 7} and {1, 2}. Let us here include the edge {0, 7} and the vertex 7 in the MST. [We could have also included edge {1, 2} and vertex 2 in the MST].*

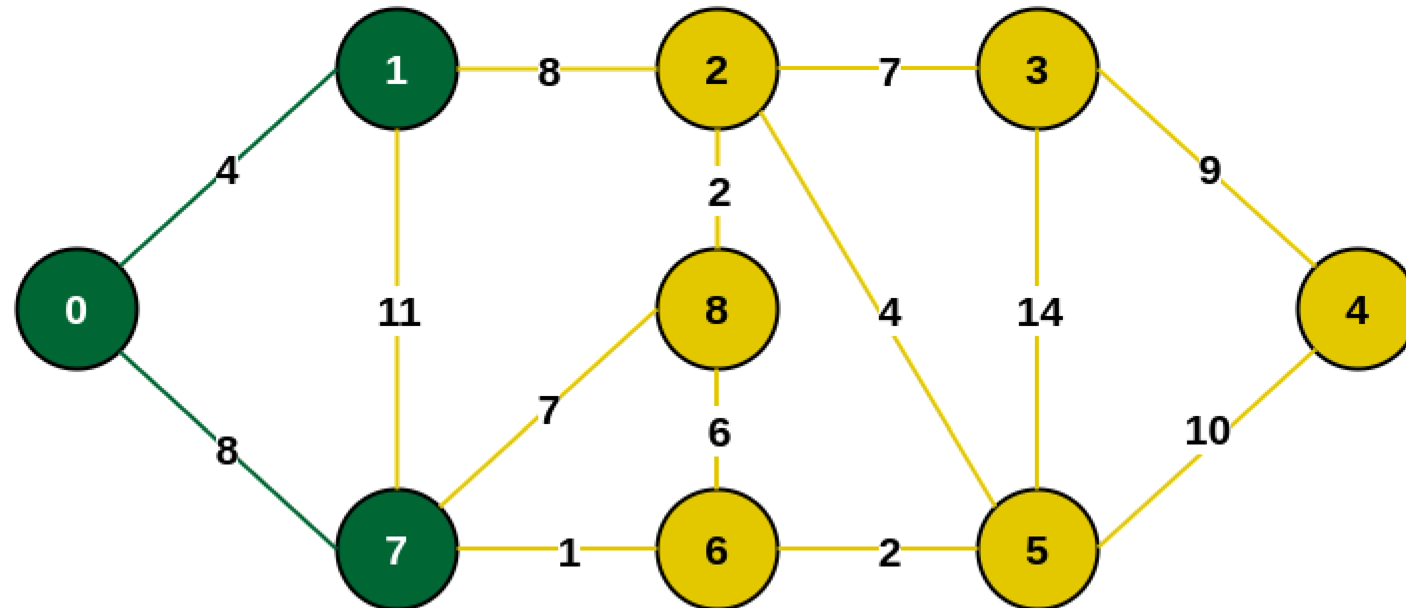
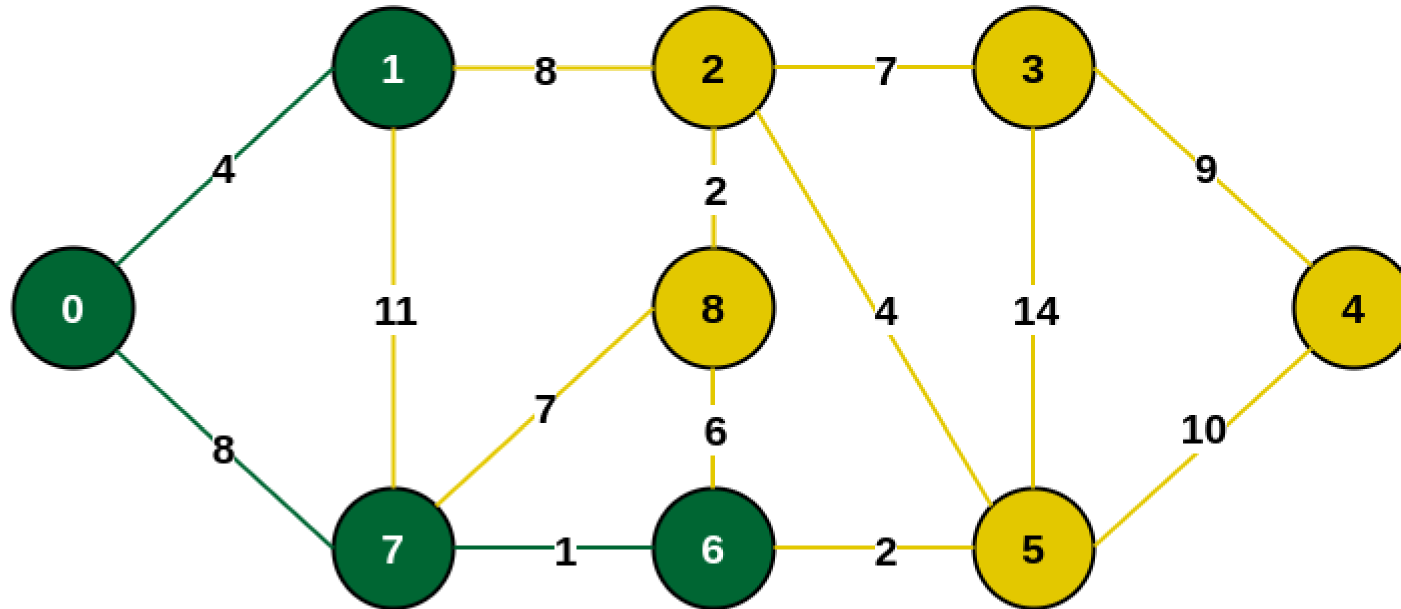


Illustration of Prim’s Algorithm:

Step 4: *The edges that connect the incomplete MST with the fringe vertices are {1, 2}, {7, 6} and {7, 8}. Add the edge {7, 6} and the vertex 6 in the MST as it has the least weight (i.e., 1).*



Minimum weighted edge from MST to other vertices is 7-6 with weight 1

Illustration of Prim's Algorithm:

Step 5: *The connecting edges now are {7, 8}, {1, 2}, {6, 8} and {6, 5}. Include edge {6, 5} and vertex 5 in the MST as the edge has the minimum weight (i.e., 2) among them.*

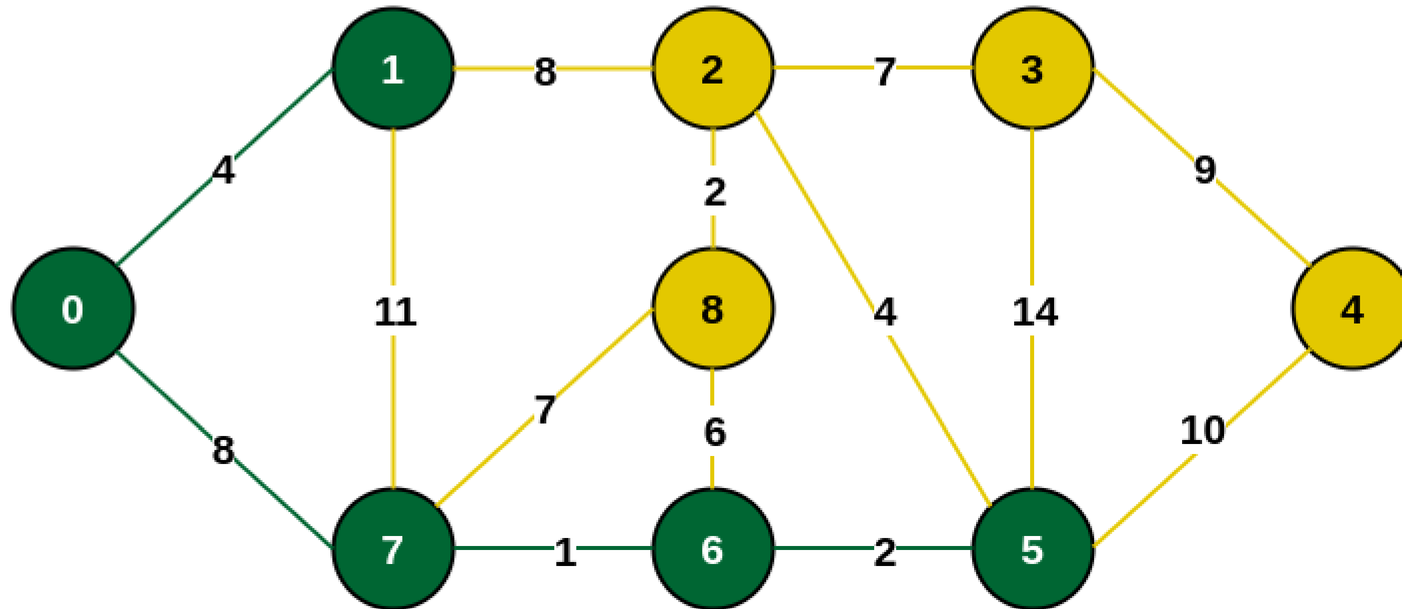


Illustration of Prim's Algorithm:

Step 6: *Among the current connecting edges, the edge {5, 2} has the minimum weight. So include that edge and the vertex 2 in the MST.*

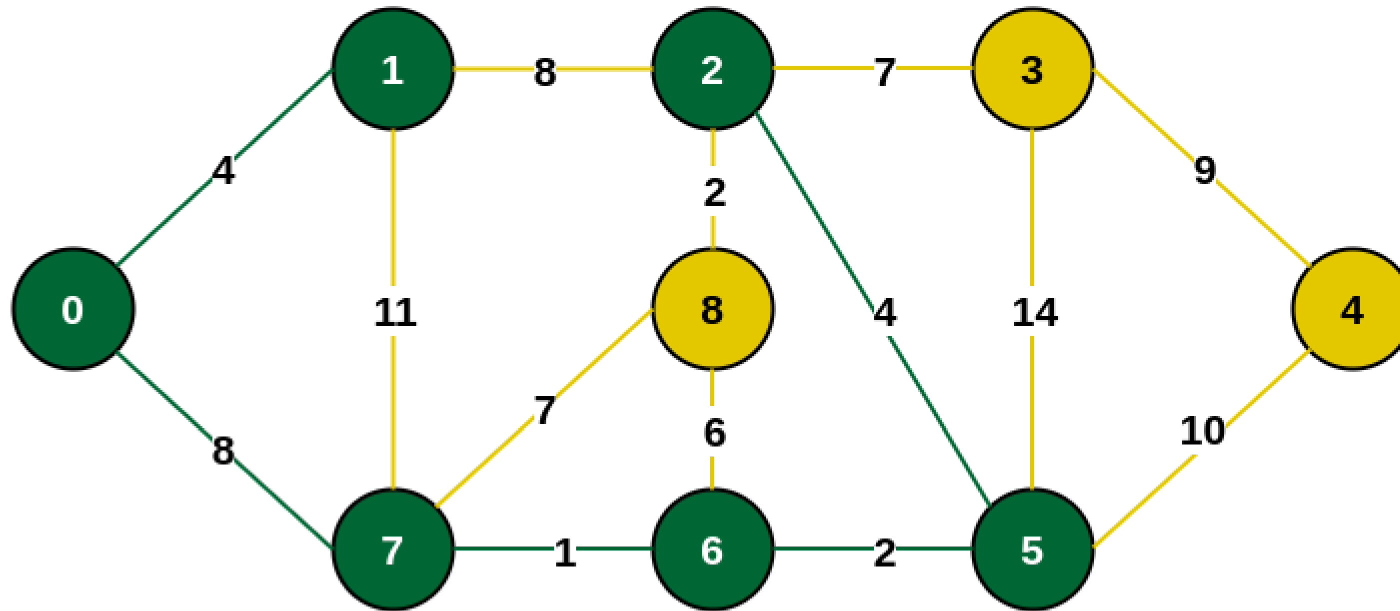
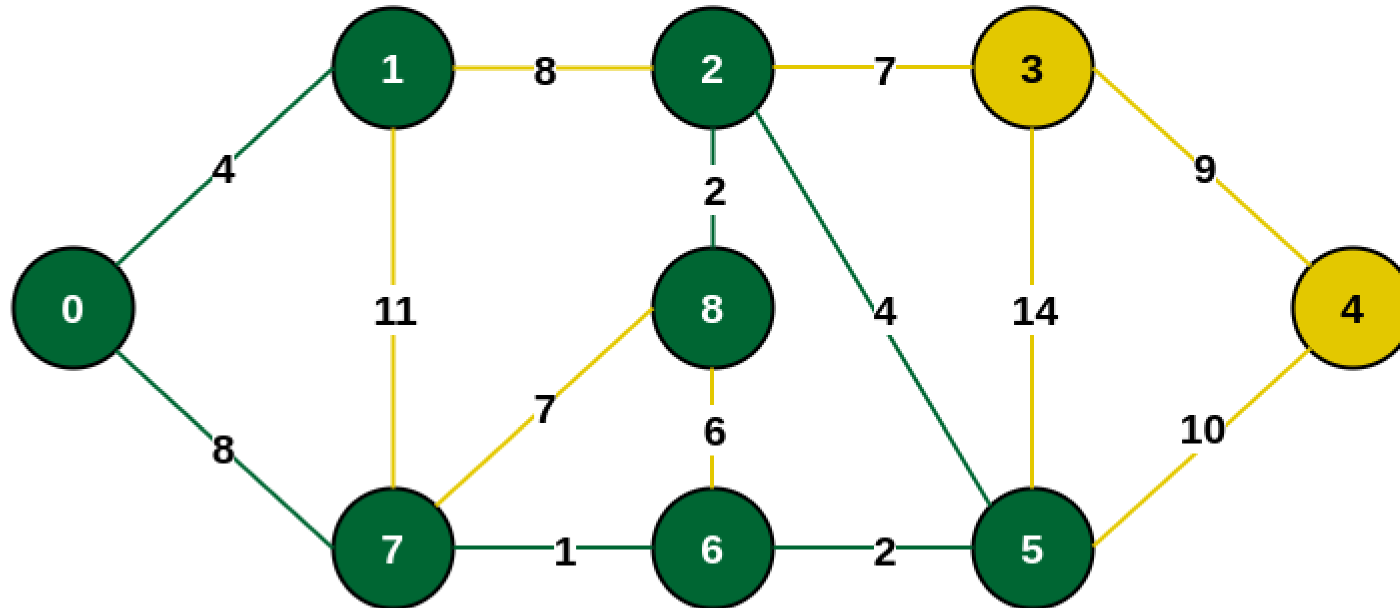


Illustration of Prim’s Algorithm:

Step 7: *The connecting edges between the incomplete MST and the other edges are {2, 8}, {2, 3}, {5, 3} and {5, 4}. The edge with minimum weight is edge {2, 8} which has weight 2. So include this edge and the vertex 8 in the MST.*



Minimum weighted edge from MST to other vertices is 2-8 with weight 2

Illustration of Prim's Algorithm:

Step 8: *See here that the edges {7, 8} and {2, 3} both have same weight which are minimum. But 7 is already part of MST. So we will consider the edge {2, 3} and include that edge and vertex 3 in the MST.*

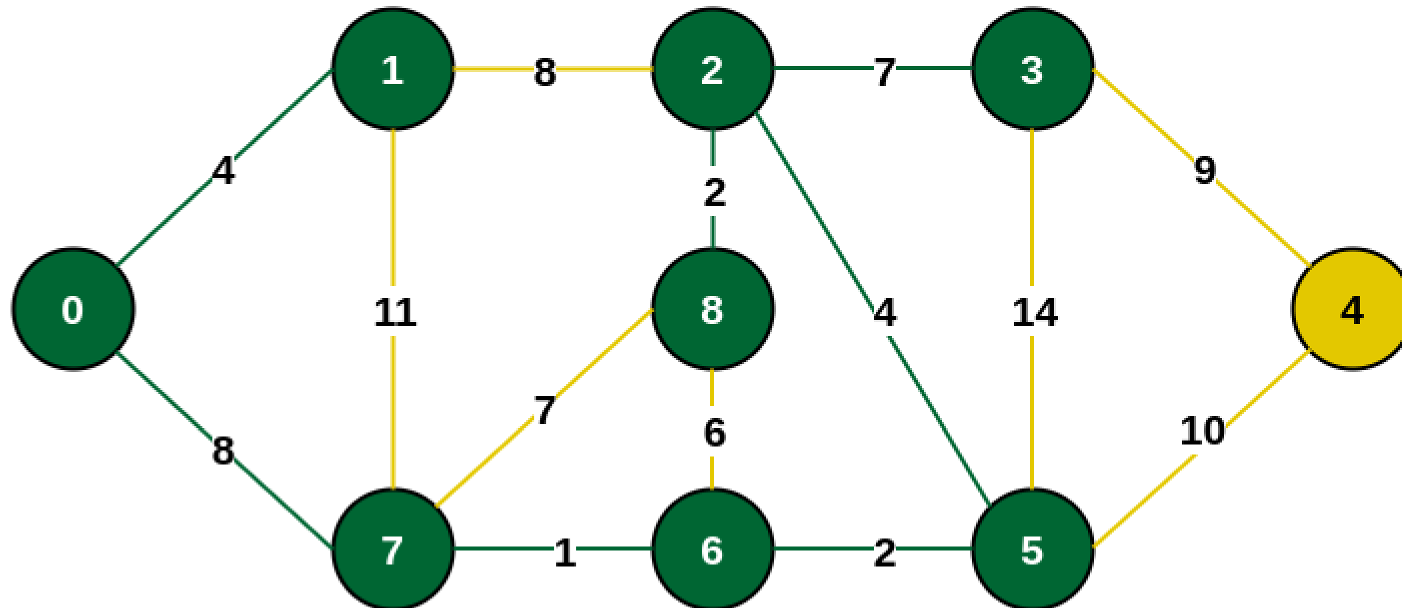


Illustration of Prim's Algorithm:

Step 9: *Only the vertex 4 remains to be included. The minimum weighted edge from the incomplete MST to 4 is {3, 4}.*

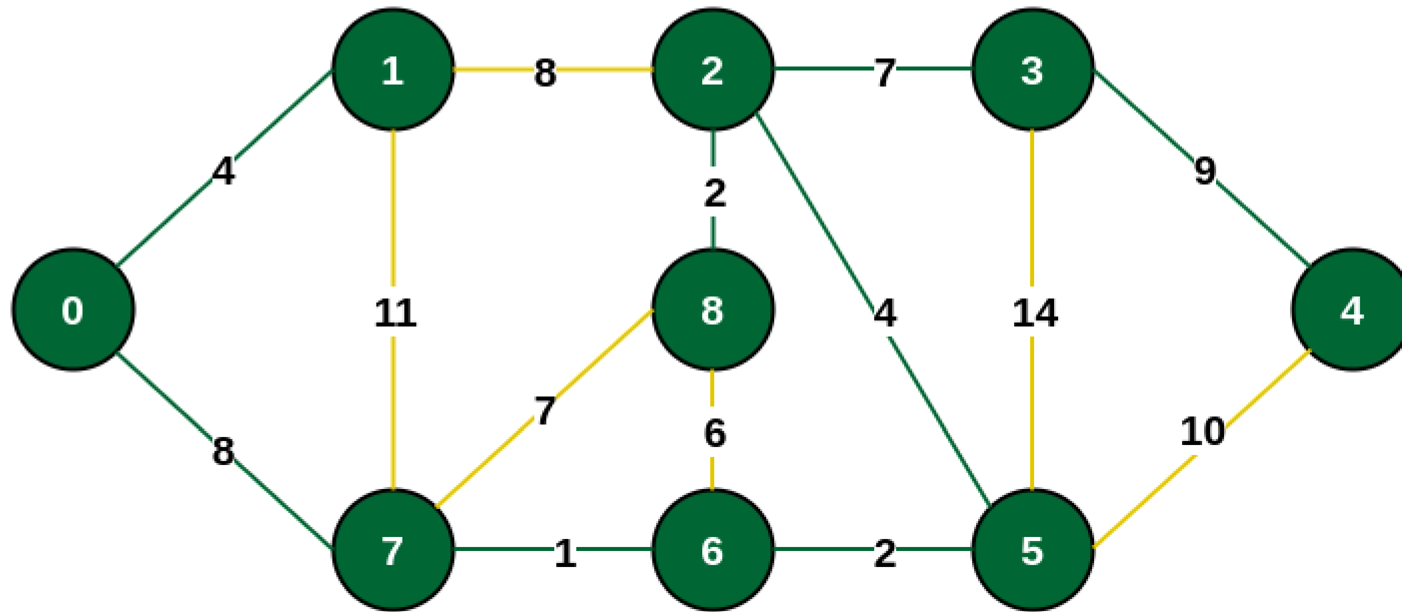
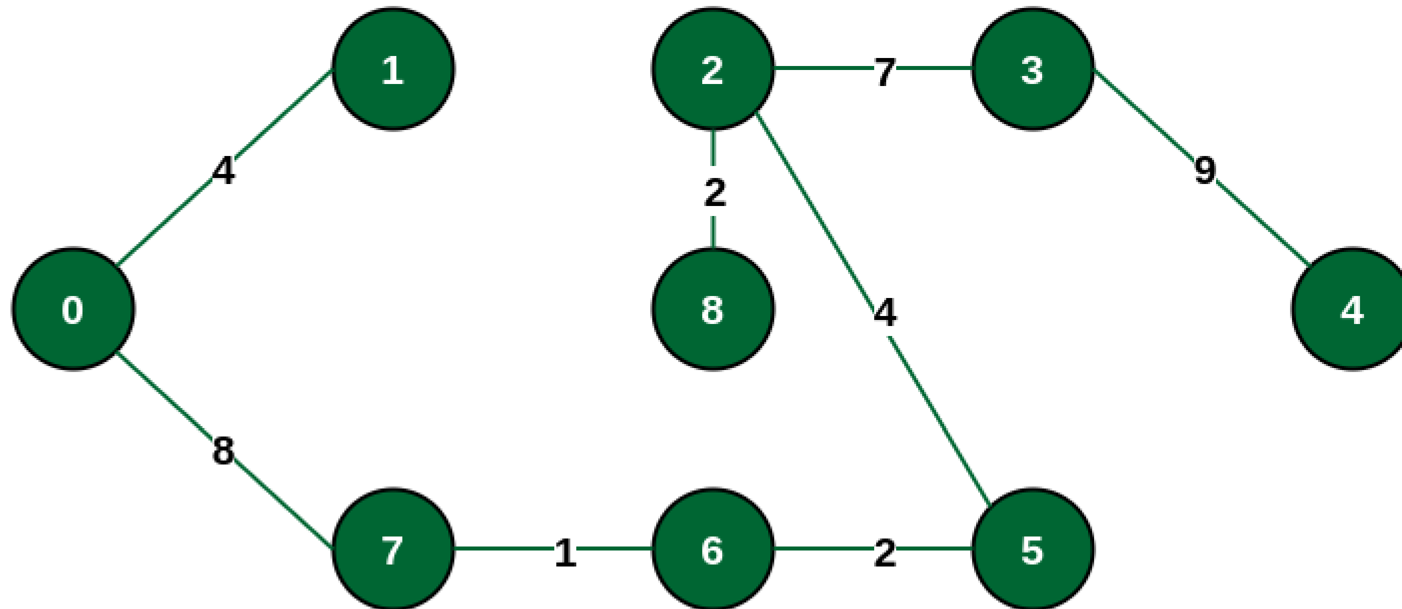


Illustration of Prim’s Algorithm:

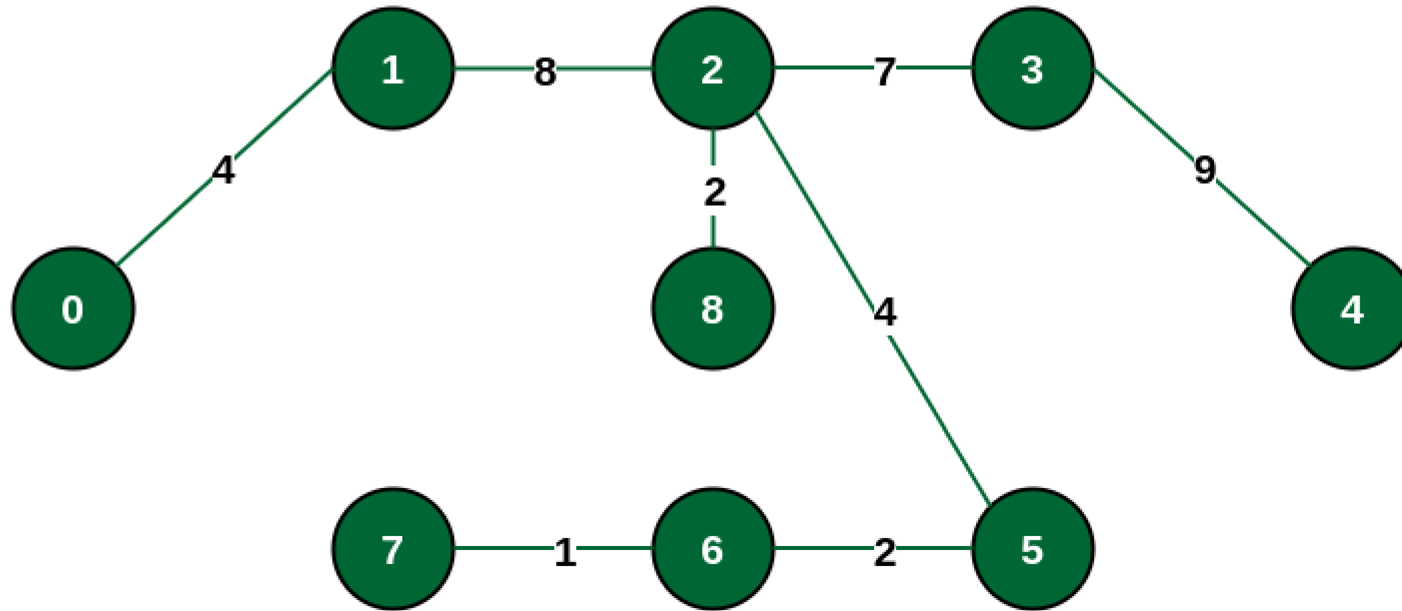
The final structure of the MST is as follows and the weight of the edges of the MST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = ***37****.*



The final structure of MST

Illustration of Prim's Algorithm:

Note: *If we had selected the edge {1, 2} in the third step then the MST would look like the following.*



Alternative MST structure

How to implement Prim's Algorithm?

Follow the given steps to utilize the **Prim's Algorithm** mentioned above for finding MST of a graph:

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

Complexity Analysis of Prim's Algorithm:

Time Complexity: $O(V^2)$, If the input graph is represented using an adjacency list, then the time complexity of Prim's algorithm can be reduced to O(E * logV) with the help of a binary heap. In this implementation, we are always considering the spanning tree to start from the root of the graph

Auxiliary Space: O(V)

Implementations of Prim's Algorithm:

Given below are some other implementations of Prim's Algorithm

- Prim's Algorithm for Adjacency Matrix Representation – In this article we have discussed the method of implementing Prim's Algorithm if the graph is represented by an adjacency matrix.
- Prim's Algorithm for Adjacency List Representation – In this article Prim's Algorithm implementation is described for graphs represented by an adjacency list.
- Prim's Algorithm using Priority Queue: In this article, we have discussed a time-efficient approach to implement Prim's algorithm.

Java Implementation (Not Optimized approach)

```java
// A Java program for Prim's Minimum Spanning Tree (MST)
// algorithm. The program is for adjacency matrix
// representation of the graph

import java.io.*;
import java.lang.*;
import java.util.*;

class MST {

   // Number of vertices in the graph
   private static final int V = 5;

   // A utility function to find the vertex with minimum
   // key value, from the set of vertices not yet included
   // in MST
   int minKey(int key[], Boolean mstSet[])
   {
      // Initialize min value
      int min = Integer.MAX_VALUE, min_index = -1;

      for (int v = 0; v < V; v++)
         if (mstSet[v] == false && key[v] < min) {
            min = key[v];
            min_index = v;
         }

      return min_index;
   }

// A utility function to print the constructed MST
   // stored in parent[]
   void printMST(int parent[], int graph[][])
   {
      System.out.println("Edge \tWeight");
      for (int i = 1; i < V; i++)
         System.out.println(parent[i] + " - " + i + "\t"
                       + graph[i][parent[i]]);
   }

   // Function to construct and print MST for a graph
   // represented using adjacency matrix representation
   void primMST(int graph[][])
   {
      // Array to store constructed MST
      int parent[] = new int[V];

      // Key values used to pick minimum weight edge in
      // cut
      int key[] = new int[V];

      // To represent set of vertices included in MST
      Boolean mstSet[] = new Boolean[V];

      // Initialize all keys as INFINITE
      for (int i = 0; i < V; i++) {
         key[i] = Integer.MAX_VALUE;
         mstSet[i] = false;
      }

// Always include first 1st vertex in MST.
   // Make key 0 so that this vertex is
   // picked as first vertex
   key[0] = 0;

   // First node is always root of MST
   parent[0] = -1;

   // The MST will have V vertices
   for (int count = 0; count < V - 1; count++) {

      // Pick the minimum key vertex from the set of
      // vertices not yet included in MST
      int u = minKey(key, mstSet);

      // Add the picked vertex to the MST Set
      mstSet[u] = true;

      // Update key value and parent index of the
      // adjacent vertices of the picked vertex.
      // Consider only those vertices which are not
      // yet included in MST
      for (int v = 0; v < V; v++)

         // graph[u][v] is non zero only for adjacent
         // vertices of m mstSet[v] is false for
         // vertices not yet included in MST Update
         // the key only if graph[u][v] is smaller
         // than key[v]
         if (graph[u][v] != 0 && mstSet[v] == false
            && graph[u][v] < key[v]) {
            parent[v] = u;
            key[v] = graph[u][v];
         }
   }

   // Print the constructed MST
   printMST(parent, graph);
 }
```

Java Implementation (Not Optimized approach)

```
public static void main(String[] args)
    {
        MST t = new MST();
        int graph[][] = new int[][] { { 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 } };

        // Print the solution
        t.primMST(graph);
    }
}
// This co
```

Output:

```
Edge    Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

Optimized Implementation using Adjacency List Representation (of Graph) and Priority Queue

Intuition

*1.We transform the adjacency matrix into adjacency list using **ArrayList<ArrayList<Integer>>**. in Java, list of list in Python*
and array of vectors in C++.

2.Then we create a Pair class to store the vertex and its weight .

3.We sort the list on the basis of lowest weight.

4.We create priority queue and push the first vertex and its weight in the queue

*5.Then we just traverse through its edges and store the least weight in a variable called **ans.***

6.At last after all the vertex we return the ans.

```
// A Java program for Prim's Minimum Spanning Tree
(MST)
// algorithm. The program is for adjacency list
// representation of the graph

import java.io.*;
import java.util.*;

// Class to form pair
class Pair implements Comparable<Pair>
{
   int v;
   int wt;
   Pair(int v,int wt)
   {
      this.v=v;
      this.wt=wt;
   }
   public int compareTo(Pair that)
   {
      return this.wt-that.wt;
   }
}
```

Optimized Implementation using Adjacency List Representation (of Graph) and Priority Queue

```
class GFG {

// Function of spanning tree
static int spanningTree(int V, int E, int edges[][])
   {
       ArrayList<ArrayList<Pair>> adj=new ArrayList<>();
       for(int i=0;i<V;i++)
       {
          adj.add(new ArrayList<Pair>());
       }
       for(int i=0;i<edges.length;i++)
       {
          int u=edges[i][0];
          int v=edges[i][1];
          int wt=edges[i][2];
          adj.get(u).add(new Pair(v,wt));
          adj.get(v).add(new Pair(u,wt));
       }
```

```
PriorityQueue<Pair> pq = new PriorityQueue<Pair>();
       pq.add(new Pair(0,0));
       int[] vis=new int[V];
       int s=0;
       while(!pq.isEmpty())
       {
          Pair node=pq.poll();
          int v=node.v;
          int wt=node.wt;
          if(vis[v]==1)
          continue;

          s+=wt;
          vis[v]=1;
          for(Pair it:adj.get(v))
          {
             if(vis[it.v]==0)
             {
                pq.add(new Pair(it.v,it.wt));
             }
          }
       }
       return s;
   }
```

Optimized Implementation using Adjacency List Representation (of Graph) and Priority Queue

```
// Driver code
  public static void main (String[] args) {
     int graph[][] = new int[][] {{0,1,5},
                                  {1,2,3},
                                  {0,2,1}};

     // Function call
     System.out.println(spanningTree(3,3,graph));
  }
}
```

Output

4

Complexity Analysis of Prim's Algorithm:

Time Complexity: O(E*log(E)) where E is the number of edges

Auxiliary Space: O(V^2) where V is the number of vertex

Kruskal’s Minimum Spanning Tree (MST) Algorithm

https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

How to find MST using Kruskal’s algorithm?

Below are the steps for finding MST using Kruskal’s algorithm:

1. Sort all the edges in a non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.

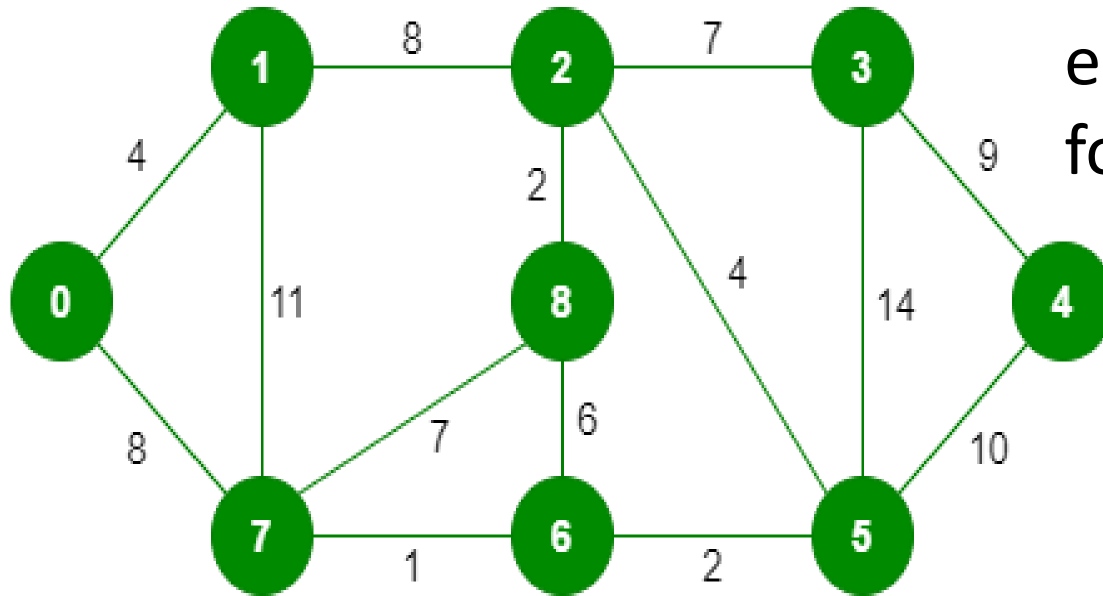
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Kruskal's Minimum Spanning Tree (MST) Algorithm

Note: *uses the* ***Union-Find algorithm*** *to detect cycles. So it is recommended reading the following algo as a prerequisite.*

- *Union-Find Algorithm | Set 1 (Detect Cycle in a Graph)*
- *Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)*

Kruskal's Minimum Spanning Tree (MST) Algorithm - Illustration



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

After sorting:

Weight	Source	Destination
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

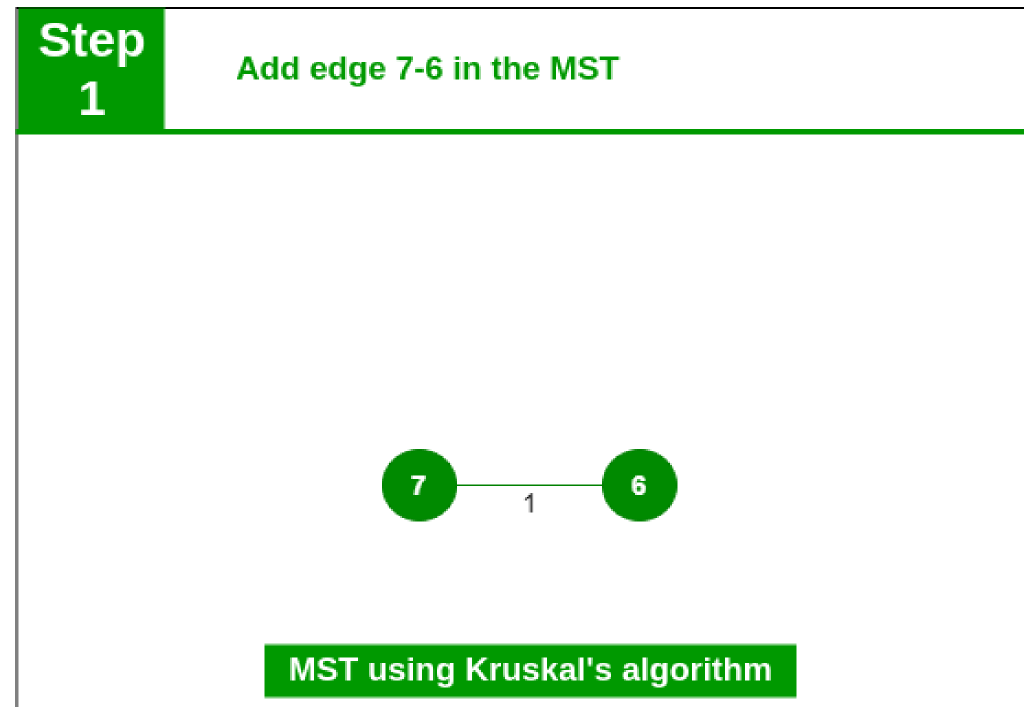
Kruskal's Minimum Spanning Tree (MST) Algorithm - Illustration

After sorting:

Weight	Source	Destination
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

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



Add edge 7-6 in the MST

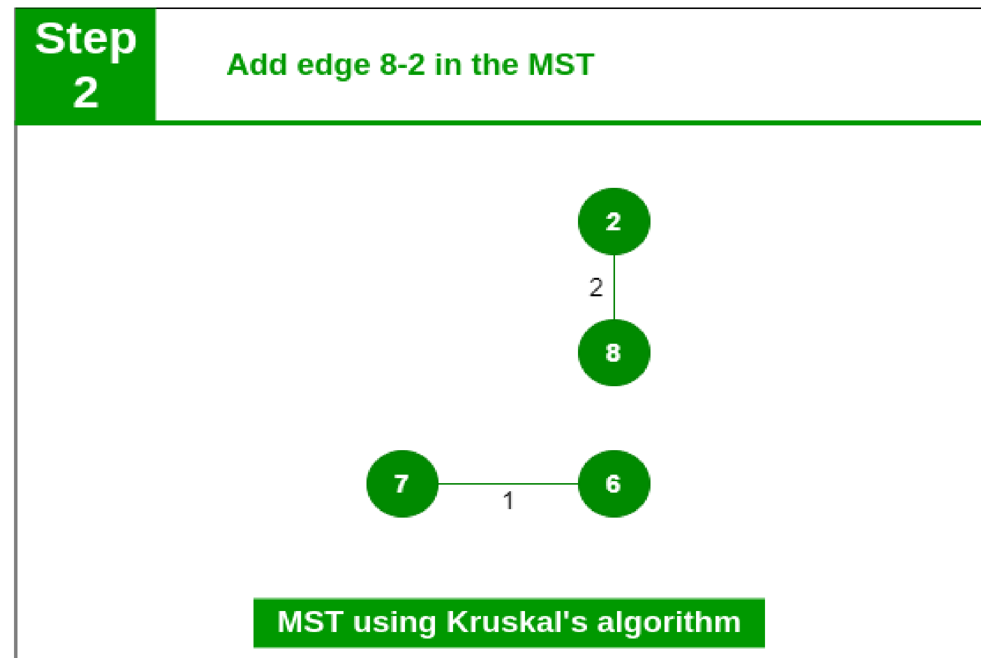
Kruskal’s Minimum Spanning Tree (MST) Algorithm - Illustration

After sorting:

Weight	Source	Destination
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

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



Add edge 8-2 in the MST

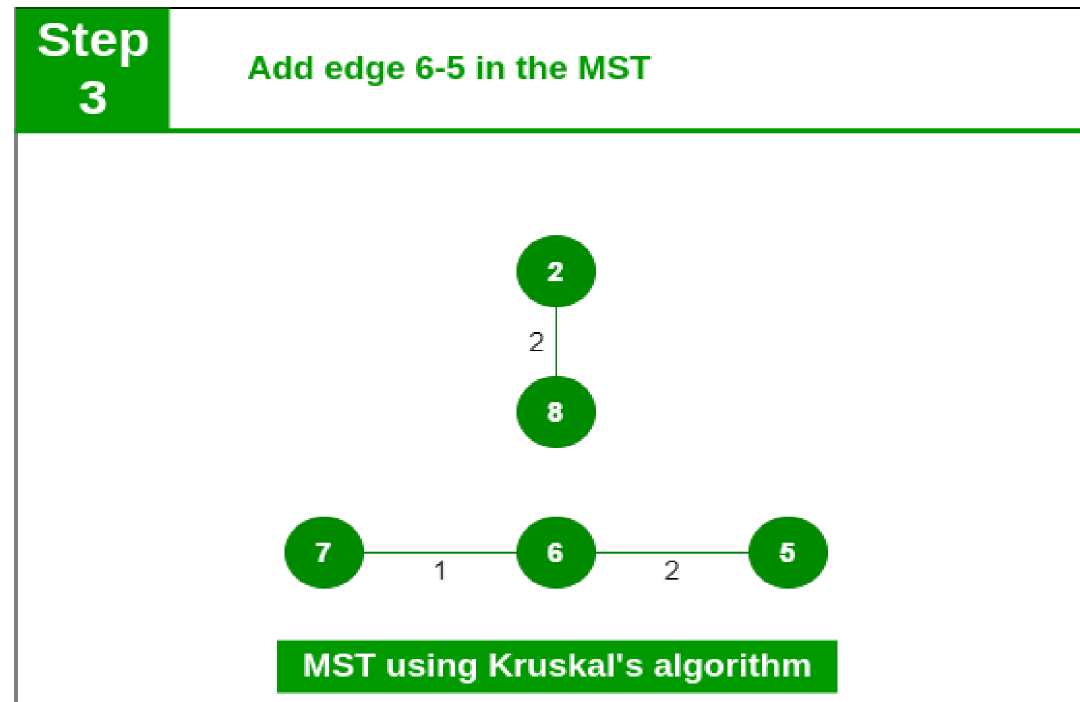
Kruskal’s Minimum Spanning Tree (MST) Algorithm - Illustration

After sorting:

Weight	Source	Destination
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

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



Add edge 6-5 in the MST

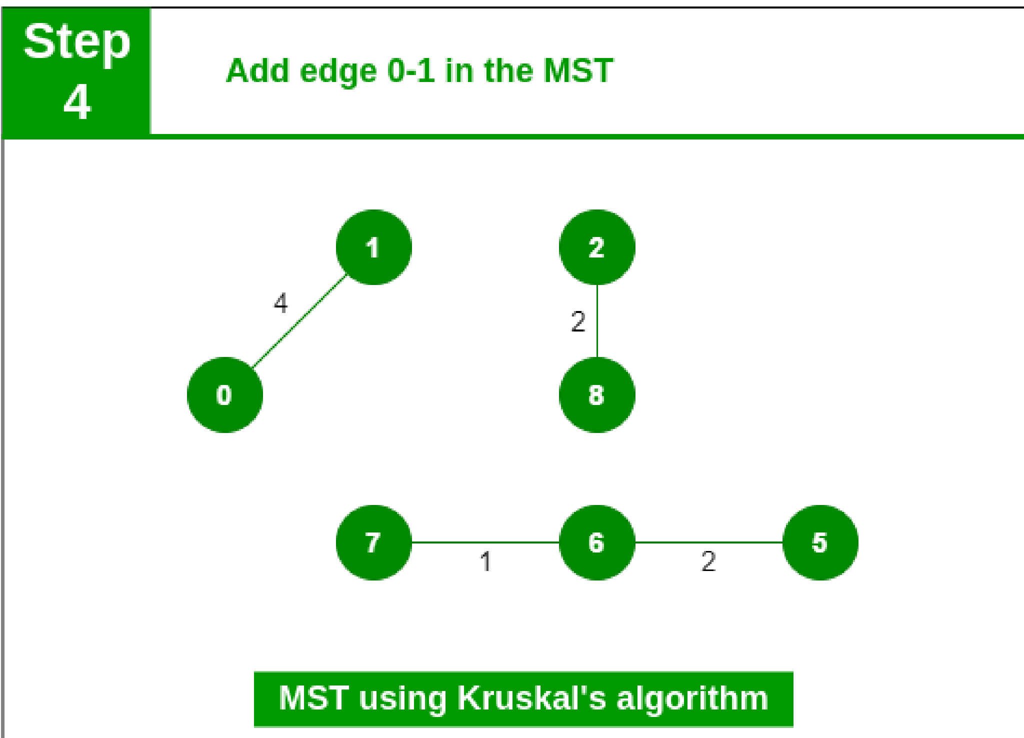
Kruskal’s Minimum Spanning Tree (MST) Algorithm - Illustration

After sorting:

Weight	Source	Destination
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

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



Add edge 0-1 in the MST

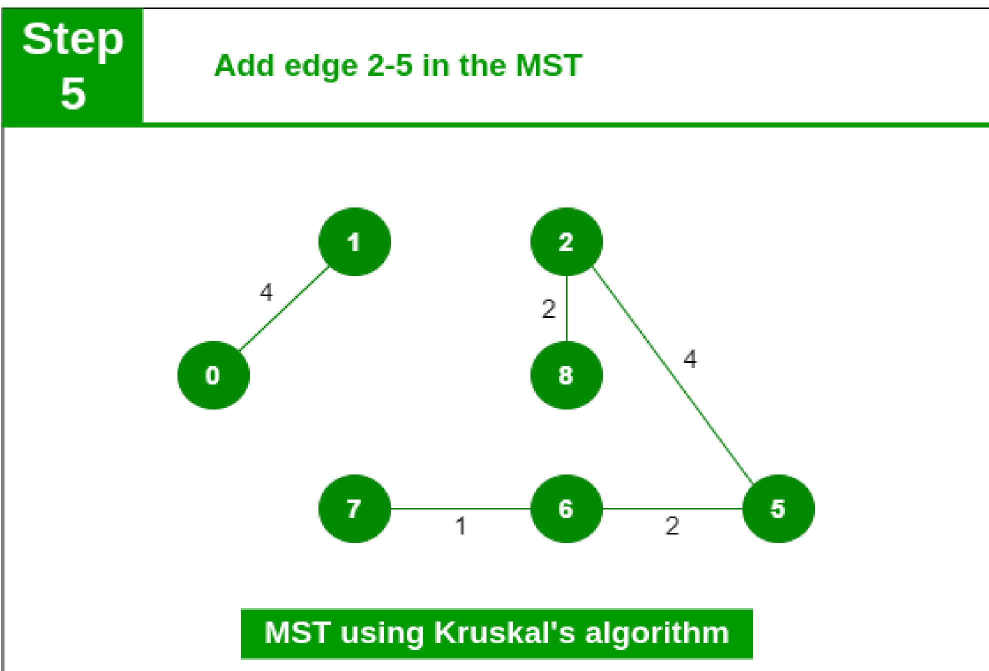
Kruskal's Minimum Spanning Tree (MST) Algorithm - Illustration

After sorting:

Weight	Source	Destination
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

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



Add edge 2-5 in the MST

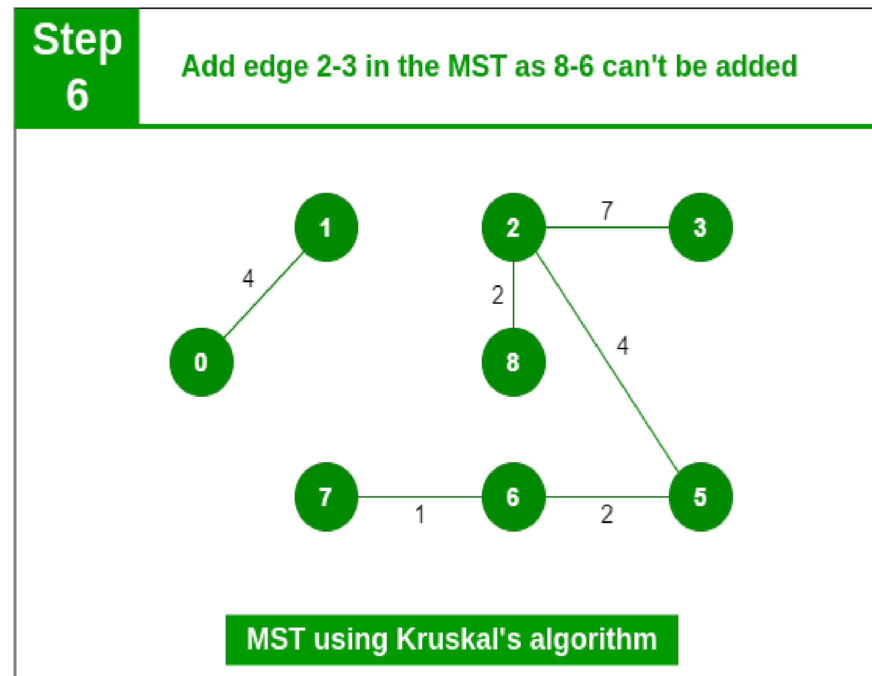
Kruskal's Minimum Spanning Tree (MST) Algorithm - Illustration

After sorting:

Weight	Source	Destination
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

Step 6: *Pick edge 8-6. Since including this edge results in the cycle, discard it. Pick edge 2-3: No cycle is formed, include it.*



Add edge 2-3 in the MST

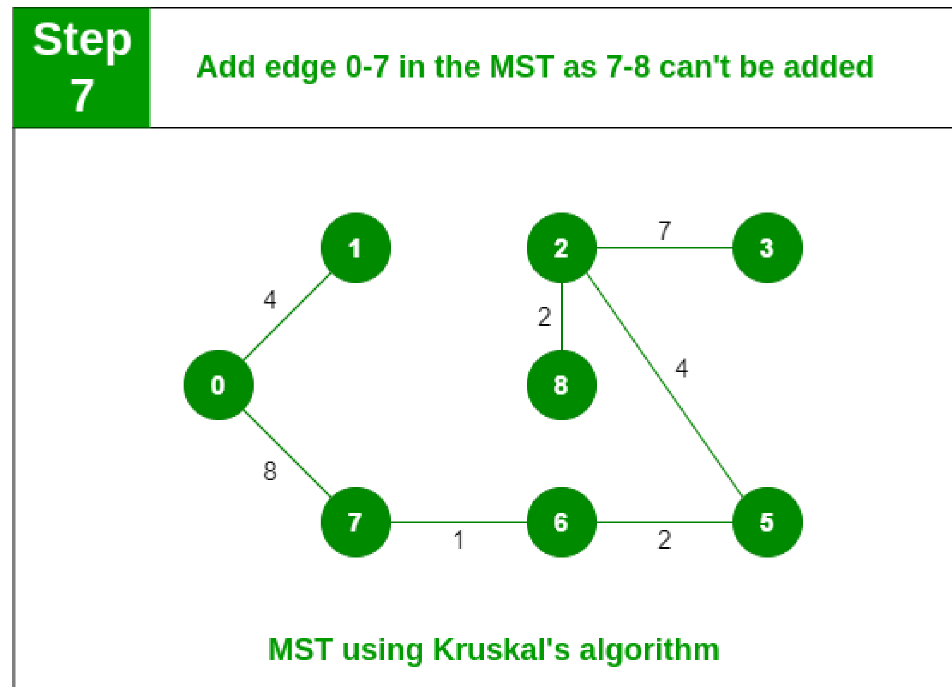
Kruskal's Minimum Spanning Tree (MST) Algorithm - Illustration

After sorting:

Weight	Source	Destination
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

Step 7: *Pick edge 7-8. Since including this edge results in the cycle, discard it. Pick edge 0-7. No cycle is formed, include it.*



Add edge 0-7 in MST

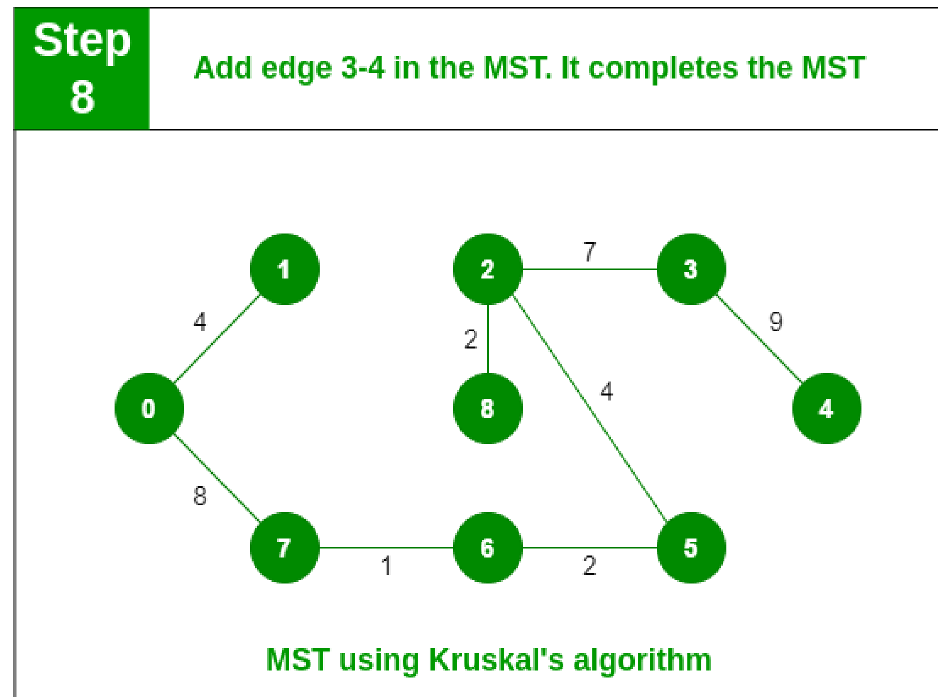
Kruskal's Minimum Spanning Tree (MST) Algorithm - Illustration

After sorting:

Weight	Source	Destination
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

Step 8: *Pick edge 1-2. Since including this edge results in the cycle, discard it. Pick edge 3-4. No cycle is formed, include it.*



Add edge 3-4 in the MST

Note: Since the number of edges included in the MST equals to (V – 1), so the algorithm stops here

Kruskal’s Minimum Spanning Tree (MST) Algorithm - Illustration

Time Complexity: O(E * logE) or O(E * logV)

- Sorting of edges takes O(E * logE) time.

- After sorting, we iterate through all edges and apply the find-union algorithm. The find and union operations can take at most O(logV) time.

- So overall complexity is O(E * logE + E * logV) time.

- The value of E can be at most $O(V^2)$, so O(logV) and O(logE) are the same. Therefore, the overall time complexity is O(E * logE) or O(E*logV)

Auxiliary Space: O(V + E), where V is the number of vertices and E is the number of edges in the graph.

Kruskal’s Minimum Spanning Tree (MST) Algorithm –Java Implementation

```java
// Java program for Kruskal's algorithm

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class KruskalsMST {

    // Defines edge structure
    static class Edge {
        int src, dest, weight;

        public Edge(int src, int dest, int
weight)
        {
            this.src = src;
            this.dest = dest;
            this.weight = weight;
        }
    }
```

```java
// Defines subset element structure
    static class Subset {
        int parent, rank;

        public Subset(int parent, int
rank)
        {
            this.parent = parent;
            this.rank = rank;
        }
    }
```

Kruskal's Minimum Spanning Tree (MST) Algorithm –Java Implementation

```java
// Starting point of program execution
    public static void main(String[] args)
    {
        int V = 4;
        List<Edge> graphEdges = new ArrayList<Edge>(
            List.of(new Edge(0, 1, 10), new Edge(0, 2, 6),
                    new Edge(0, 3, 5), new Edge(1, 3, 15),
                    new Edge(2, 3, 4)));

        // Sort the edges in non-decreasing order
        // (increasing with repetition allowed)
        graphEdges.sort(new Comparator<Edge>() {
            @Override public int compare(Edge o1, Edge o2)
            {
                return o1.weight - o2.weight;
            }
        });

        kruskals(V, graphEdges);
    }
```

```java
// Function to find the MST
    private static void kruskals(int V,
List<Edge> edges)
    {
        int j = 0;
        int noOfEdges = 0;

        // Allocate memory for creating V
subsets
        Subset subsets[] = new Subset[V];

        // Allocate memory for results
        Edge results[] = new Edge[V];

        // Create V subsets with single
elements
        for (int i = 0; i < V; i++) {
            subsets[i] = new Subset(i, 0);
        }
```

Kruskal’s Minimum Spanning Tree (MST) Algorithm –Java Implementation

```
// Number of edges to be taken is equal to V-1
        while (noOfEdges < V - 1) {

            // Pick the smallest edge. And increment
            // the index for next iteration
            Edge nextEdge = edges.get(j);
            int x = findRoot(subsets, nextEdge.src);
            int y = findRoot(subsets, nextEdge.dest);

            // If including this edge doesn't cause cycle,
            // include it in result and increment the index
            // of result for next edge
            if (x != y) {
                results[noOfEdges] = nextEdge;
                union(subsets, x, y);
                noOfEdges++;
            }

            j++;
        }
```

```
// Print the contents of result[] to display the
        // built MST
        System.out.println(
            "Following are the edges of the constructed MST:");
        int minCost = 0;
        for (int i = 0; i < noOfEdges; i++) {
            System.out.println(results[i].src + " -- "
                               + results[i].dest + " == "
                               + results[i].weight);
            minCost += results[i].weight;
        }
        System.out.println("Total cost of MST: " + minCost);
    }
```

Kruskal’s Minimum Spanning Tree (MST) Algorithm –Java Implementation

```java
// Function to unite two disjoint sets
    private static void union(Subset[] subsets, int x,
                                    int y)
    {
        int rootX = findRoot(subsets, x);
        int rootY = findRoot(subsets, y);

        if (subsets[rootY].rank < subsets[rootX].rank)
{
            subsets[rootY].parent = rootX;
        }
        else if (subsets[rootX].rank
                 < subsets[rootY].rank) {
            subsets[rootX].parent = rootY;
        }
        else {
            subsets[rootY].parent = rootX;
            subsets[rootX].rank++;
        }
    }
```

```java
// Function to find parent of a set
    private static int findRoot(Subset[] subsets, int i)
    {
        if (subsets[i].parent == i)
            return subsets[i].parent;

        subsets[i].parent
            = findRoot(subsets, subsets[i].parent);
        return subsets[i].parent;
    }
}
```

Output

Following are the edges in the constructed MST

2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Cost Spanning Tree: 19

Acknowledge

PPT has been prepared from the content available on GeekforGeek