**Aim-** To write a code and run it for minimum spanning tree suing Prim’s and Kruskal’s Algorithm

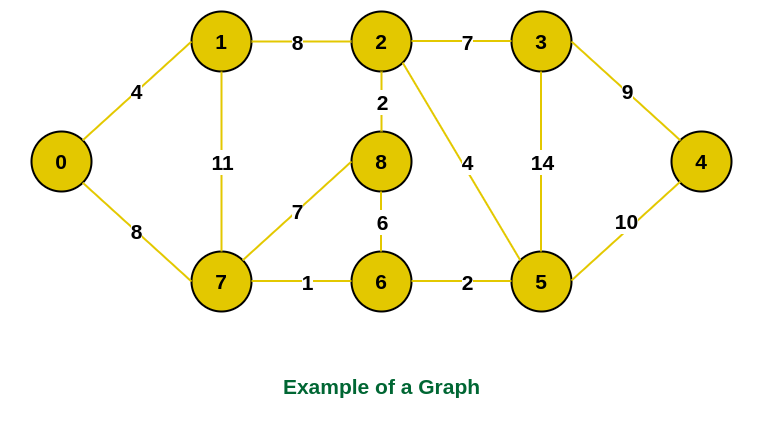
**Theory-** In Kruskal’s algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first at the maximum weighted edge at last. Thus we can say that it makes a locally optimal choice in each step in order to find the optimal solution. Hence this is a Greedy Algorithm.

Like Kruskal’s algorithm, Prim’s algorithm is also a Greedy algorithm. This algorithm always starts with a single node and moves through several adjacent nodes, in order to explore all of the connected edges along the way. A group of edges that connects two sets of vertices in a graph is called cut in graph theory. So, at every step of Prim’s algorithm, find a cut, pick the minimum weight edge from the cut, and include this vertex in MST Set (the set that contains already included vertices).

**Example-**

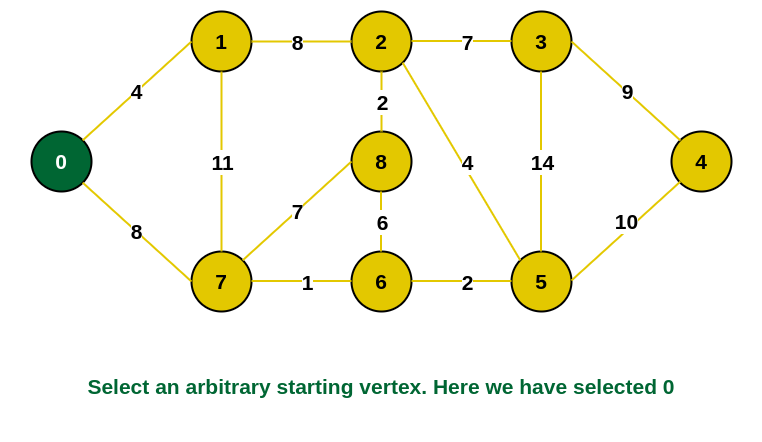
**Prim’s-**

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



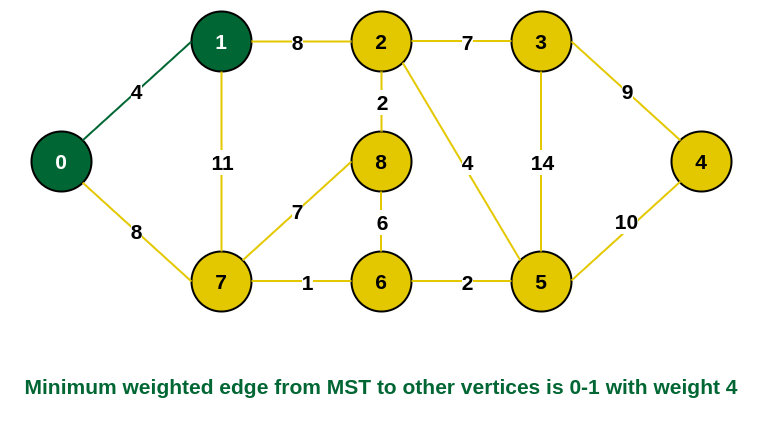
Example of a graph

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.



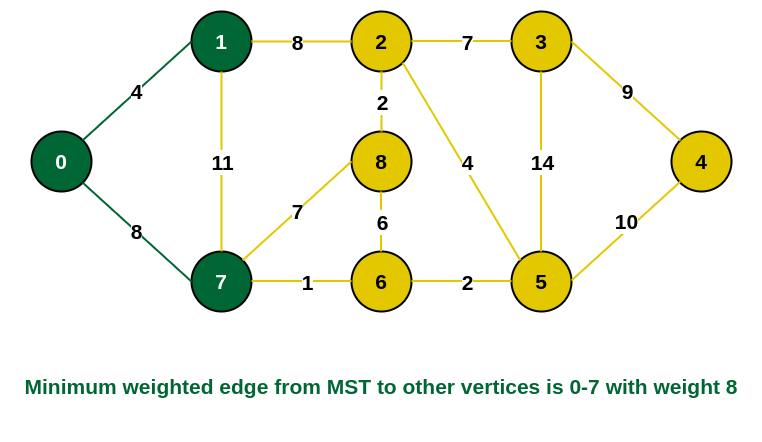
0 is selected as starting vertex

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.



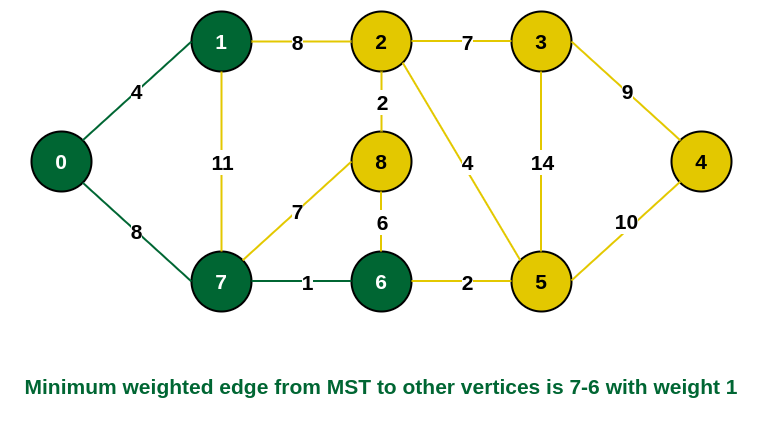
1 is added to the MST

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].



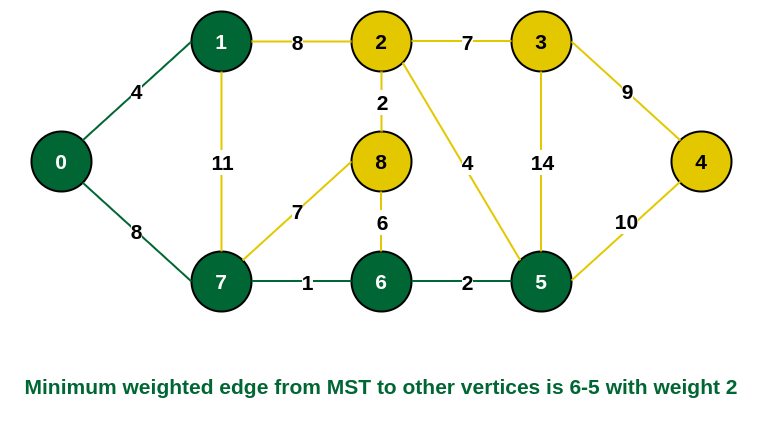
7 is added in the MST

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).



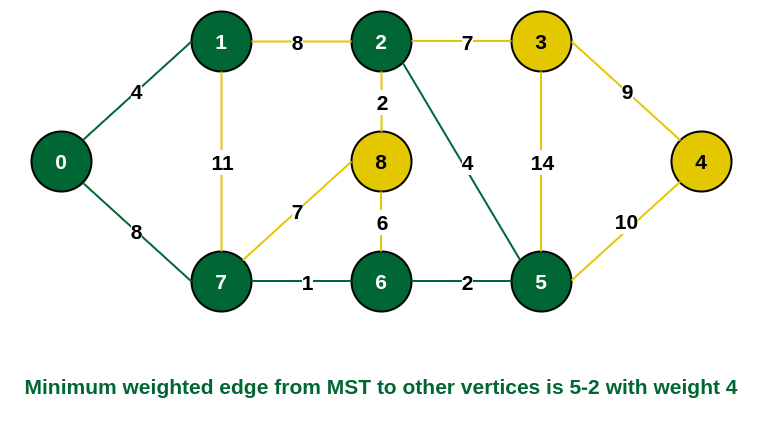
6 is added in the MST

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.



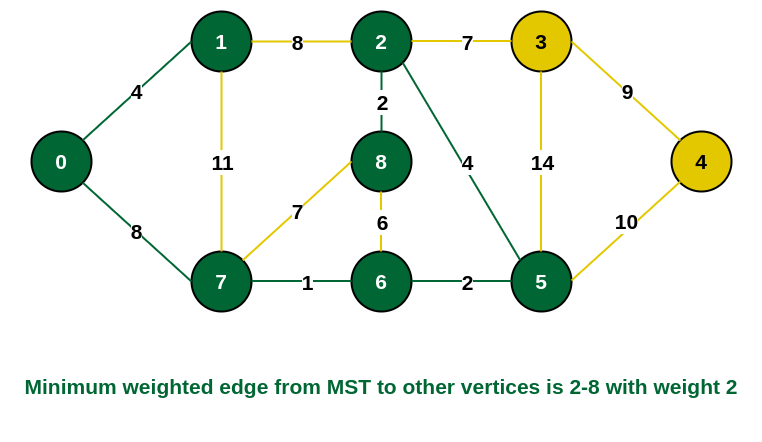
Include vertex 5 in the MST

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.



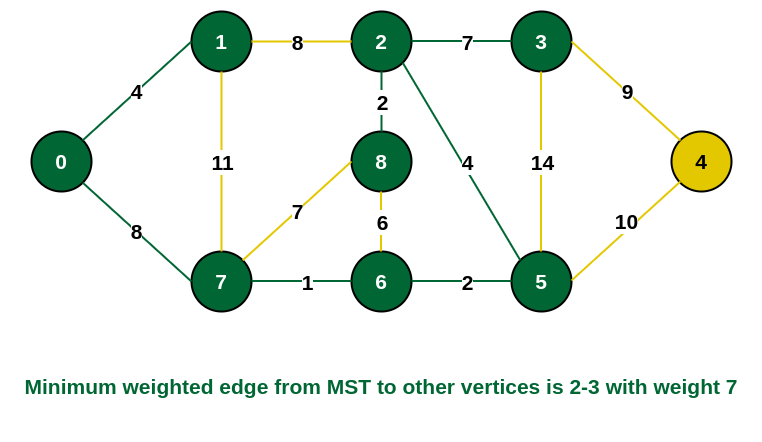
Include vertex 2 in the MST

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.

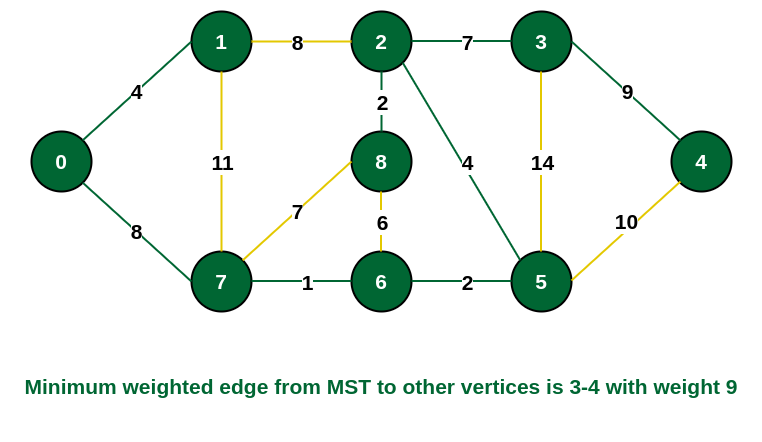


Add vertex 8 in the MST

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.

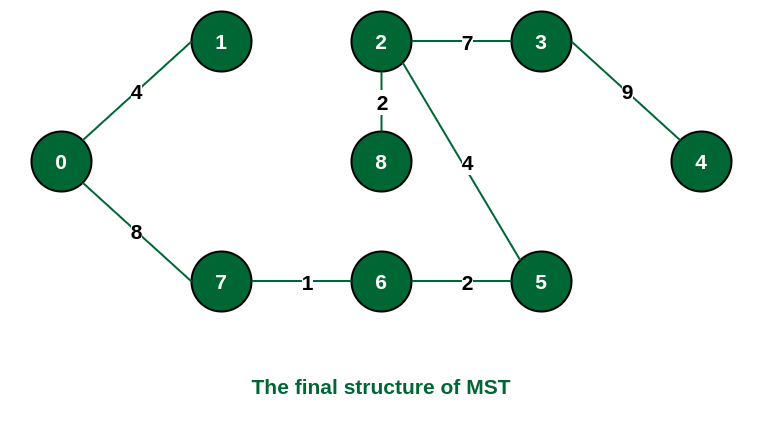


Include vertex 3 in MST

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

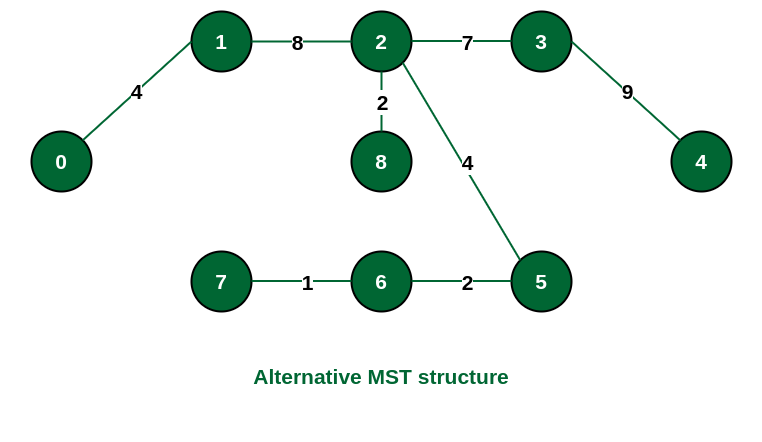
Include vertex 4 in the MST

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 structure of the MST formed using the above method

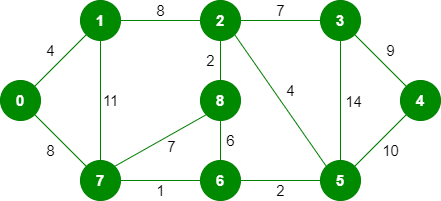
Note: If we had selected the edge {1, 2} in the third step then the MST would look like the following.



Structure of the alternate MST if we had selected edge {1, 2} in the MST

**Kruskal’s-**

Input Graph:

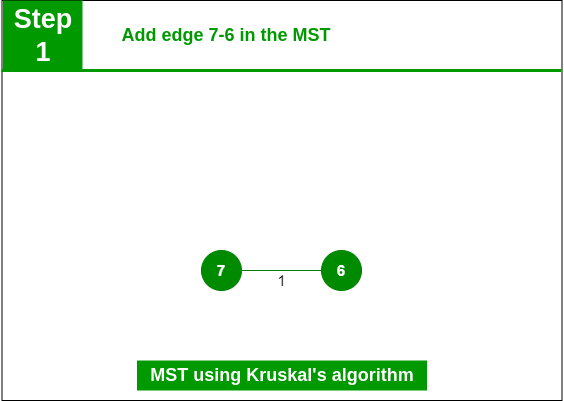


The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.   
After sorting:

|  |  |  |
| --- | --- | --- |
| 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 |

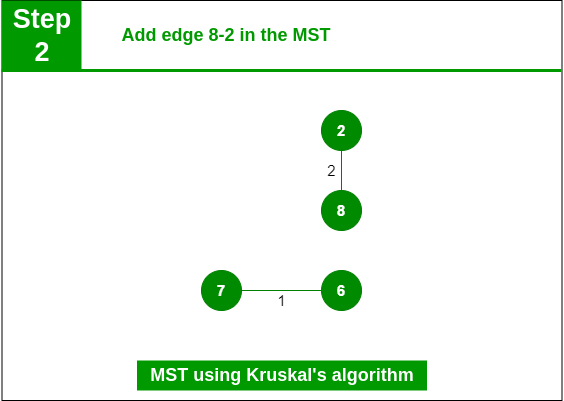
Now pick all edges one by one from the sorted list of edges

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



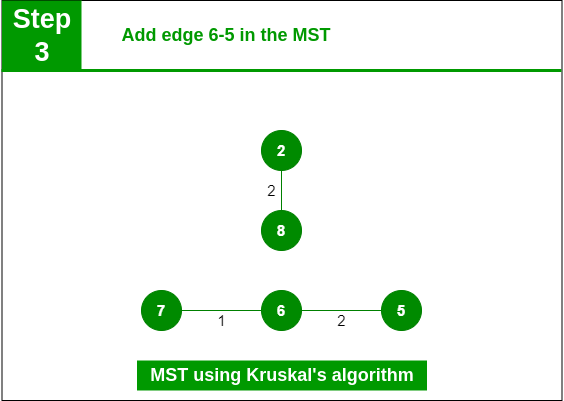
Add edge 7-6 in the MST

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



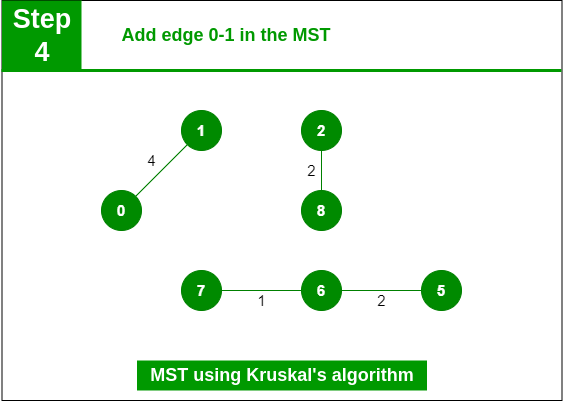
Add edge 8-2 in the MST

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



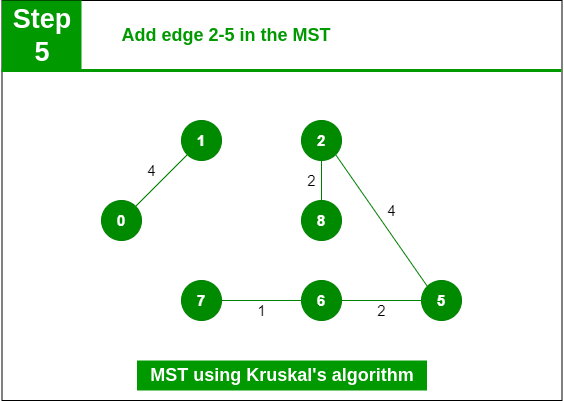
Add edge 6-5 in the MST

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



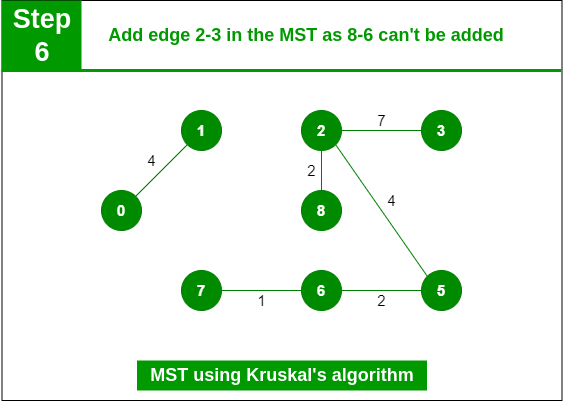
Add edge 0-1 in the MST

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



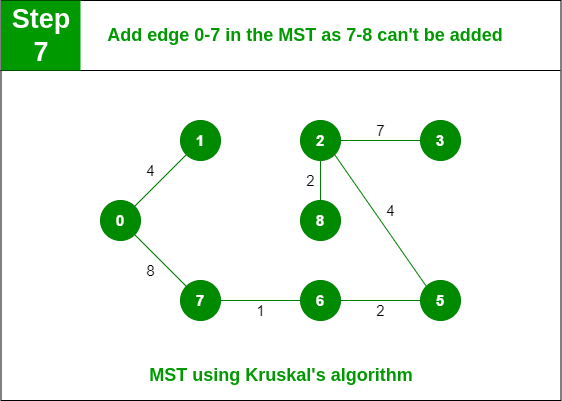
Add edge 2-5 in the MST

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.

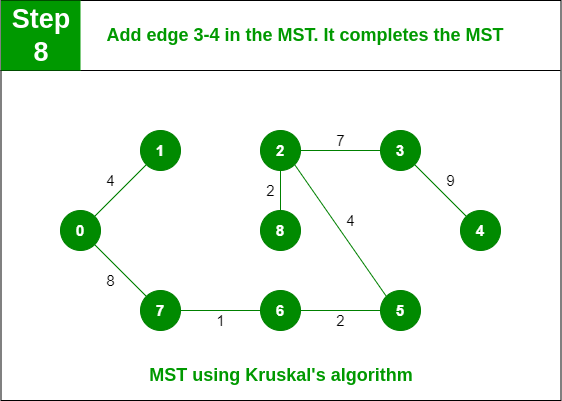


Add edge 2-3 in the MST

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

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

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

**Algorithm-**

Prim’s-

**MST-PRIM (G, w, r)**

1. for each u ∈ V [G]

2. do key [u] ← ∞

3. π [u] ← NIL

4. key [r] ← 0

5. Q ← V [G]

6. While Q? ∅

7. do u ← EXTRACT - MIN (Q)

8. for each v ∈ Adj [u]

9. do if v ∈ Q and w (u, v) < key [v]

10. then π [v] ← u

11. key [v] ← w (u, v)

Kruskal’s-

**MST- KRUSKAL (G, w)**

1. A ← ∅

2. for each vertex v ∈ V [G]

3. do MAKE - SET (v)

4. sort the edges of E into non decreasing order by weight w

5. for each edge (u, v) ∈ E, taken in non-decreasing order by weight

6. do if FIND-SET (μ) ≠ if FIND-SET (v)

7. then A ← A ∪ {(u, v)}

8. UNION (u, v)

9. return A

**Code-**

**Prim’s**

#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

// Number of vertices in the graph

#define 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[], bool mstSet[])

{

    // Initialize min value

    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;

}

// A utility function to print the

// constructed MST stored in parent[]

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

{

    printf("Edge \tWeight\n");

    for (int i = 1; i < V; i++)

        printf("%d - %d \t%d \n", parent[i], i,

               graph[i][parent[i]]);

}

// Function to construct and print MST for

// a graph represented using adjacency

// matrix representation

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

{

    // Array to store constructed MST

    int parent[V];

    // Key values used to pick minimum weight edge in cut

    int key[V];

    // To represent set of vertices included in MST

    bool mstSet[V];

    // Initialize all keys as INFINITE

    for (int i = 0; i < V; i++)

        key[i] = INT\_MAX, 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] && mstSet[v] == false

                && graph[u][v] < key[v])

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

    }

    // print the constructed MST

    printMST(parent, graph);

}

// Driver's code

int main()

{

    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 } };

    // Print the solution

    primMST(graph);

    return 0;

}

**Kruskal’s**

#include <stdio.h>

#include <stdlib.h>

// Comparator function to use in sorting

int comparator(const void\* p1, const void\* p2)

{

    const int(\*x)[3] = p1;

    const int(\*y)[3] = p2;

    return (\*x)[2] - (\*y)[2];

}

// Initialization of parent[] and rank[] arrays

void makeSet(int parent[], int rank[], int n)

{

    for (int i = 0; i < n; i++) {

        parent[i] = i;

        rank[i] = 0;

    }

}

// Function to find the parent of a node

int findParent(int parent[], int component)

{

    if (parent[component] == component)

        return component;

    return parent[component]

           = findParent(parent, parent[component]);

}

// Function to unite two sets

void unionSet(int u, int v, int parent[], int rank[], int n)

{

    // Finding the parents

    u = findParent(parent, u);

    v = findParent(parent, v);

    if (rank[u] < rank[v]) {

        parent[u] = v;

    }

    else if (rank[u] > rank[v]) {

        parent[v] = u;

    }

    else {

        parent[v] = u;

        // Since the rank increases if the ranks of two sets are same

        rank[u]++;

    }

}

// Function to find the MST

void kruskalAlgo(int n, int edge[n][3])

{

    // First we sort the edge array in ascending order

    // so that we can access minimum distances/cost

    qsort(edge, n, sizeof(edge[0]), comparator);

    int parent[n];

    int rank[n];

    // Function to initialize parent[] and rank[]

    makeSet(parent, rank, n);

    // To store the minimun cost

    int minCost = 0;

    printf("Following are the edges in the constructed MST\n");

    for (int i = 0; i < n; i++) {

        int v1 = findParent(parent, edge[i][0]);

        int v2 = findParent(parent, edge[i][1]);

        int wt = edge[i][2];

        // If the parents are different that

        // means they are in different sets so

        // union them

        if (v1 != v2) {

            unionSet(v1, v2, parent, rank, n);

            minCost += wt;

            printf("%d -- %d == %d\n", edge[i][0],

                   edge[i][1], wt);

        }

    }

    printf("Minimum Cost Spanning Tree: %d\n", minCost);

}

int main()

{

    int edge[5][3] = { { 0, 1, 10 },

                       { 0, 2, 6 },

                       { 0, 3, 5 },

                       { 1, 3, 15 },

                       { 2, 3, 4 } };

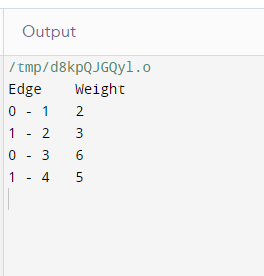
    kruskalAlgo(5, edge);

    return 0;

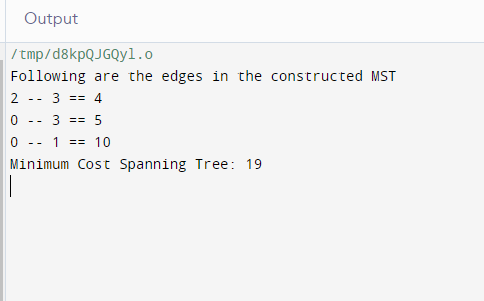
}

**Output-**

**Prim’s**



**Kruskal’s**



**Conclusion-**Thus we have used an example to illustrate Minimum spanning tree using Prim’s and Kruskal’s algorithm