PRACTICAL - 9

Name:shubham shivraj Suryawanshi
Reg. No:2020BIT004

Implement the following algorithm for minimum cost spanning tree

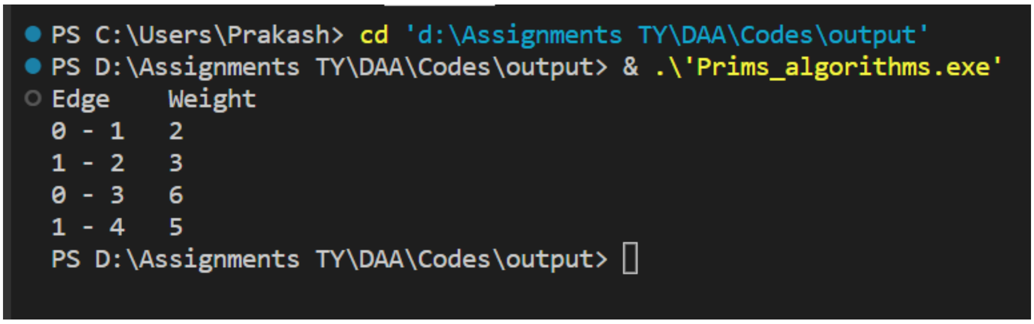
Prims algoritham using Binary Heap

Kruskal's algorithm using Min Heap

Write a Algorithm with complete Simulation

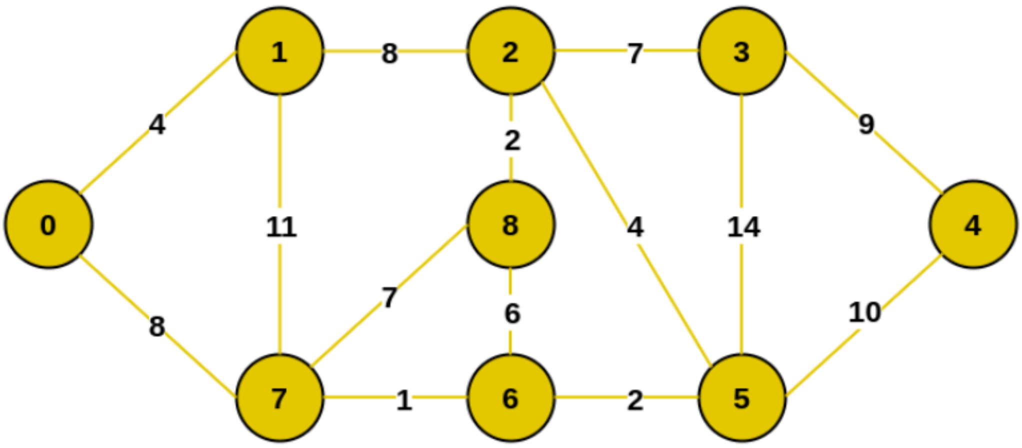
1) Prims algoritham using Binary Heap

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#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;
}
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]]);
}
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()
{
 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:

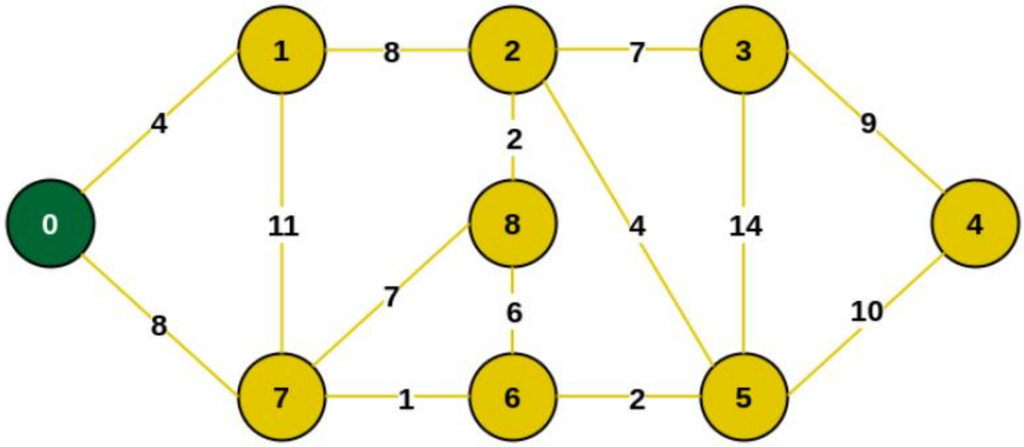


```
PS C:\Users\Prakash> cd 'd:\Assignments TY\DAA\Codes\output'
PS D:\Assignments TY\DAA\Codes\output> & .\'Prims_algorithms.exe'
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
PS D:\Assignments TY\DAA\Codes\output> 
```

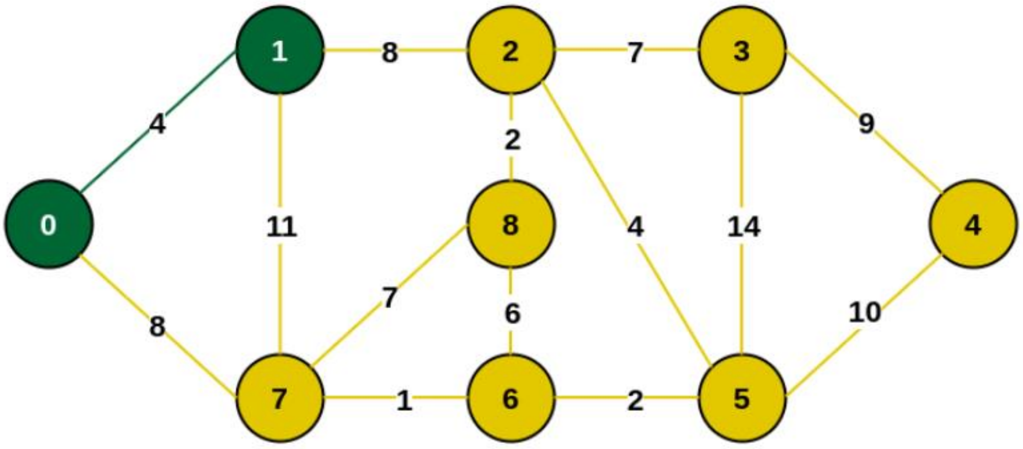
Simulation:



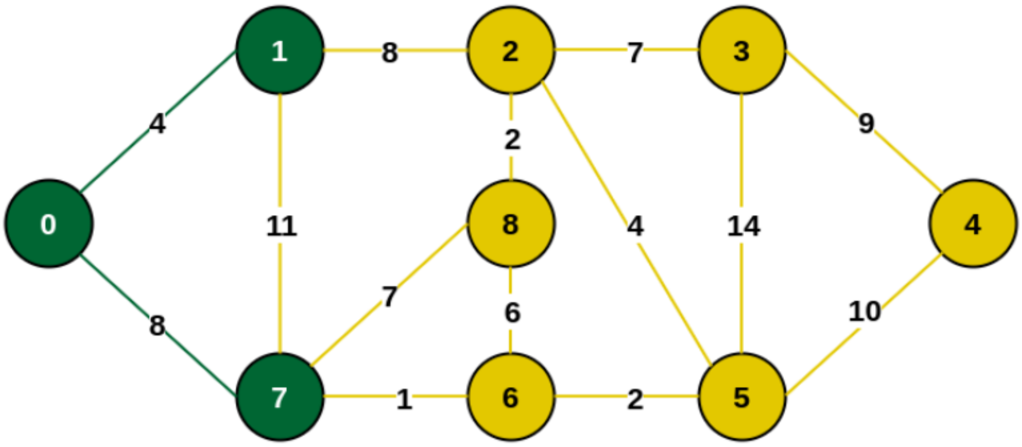
Example of a Graph



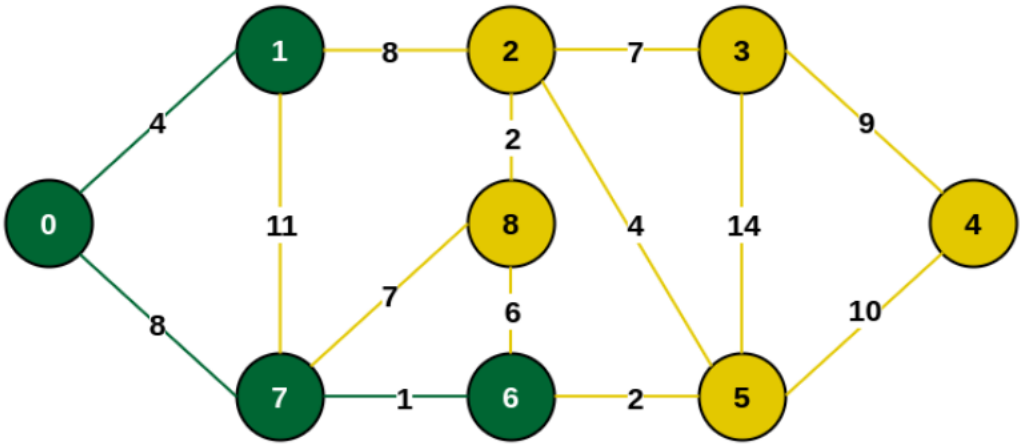
Select an arbitrary starting vertex. Here we have selected 0



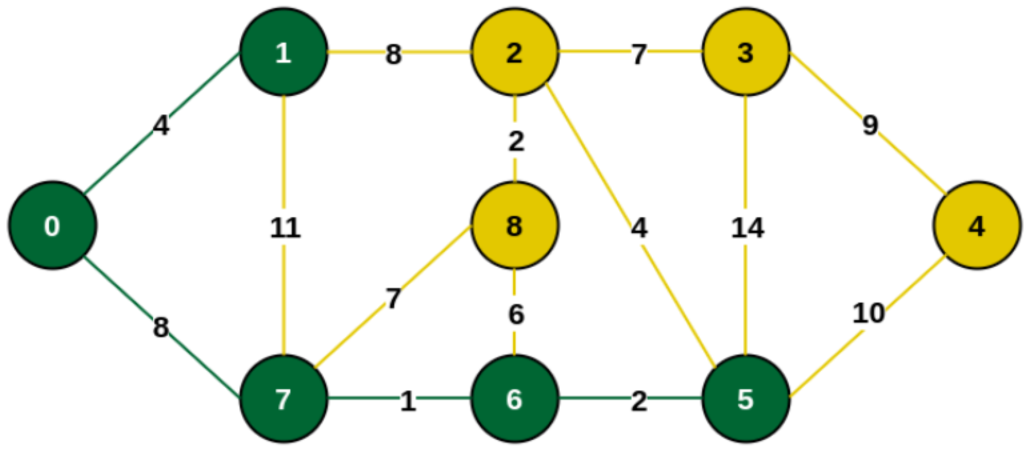
Minimum weighted edge from MST to other vertices is 0-1 with weight 4



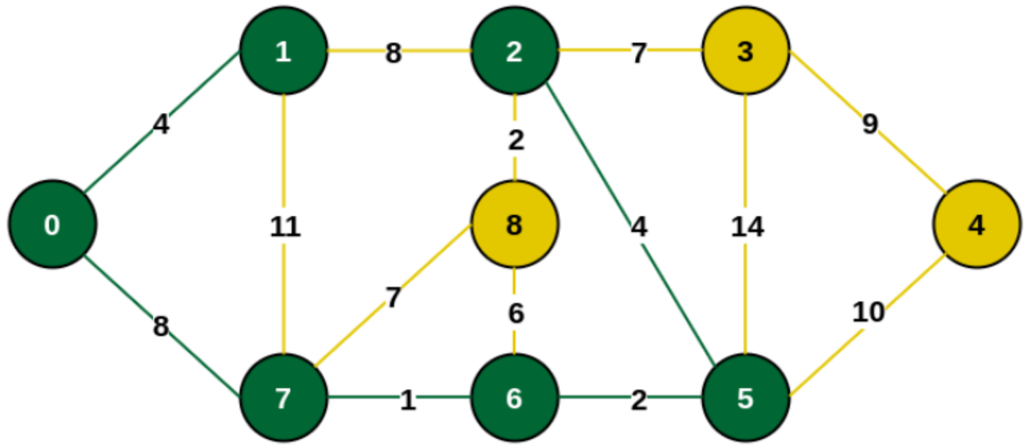
Minimum weighted edge from MST to other vertices is 0-7 with weight 8



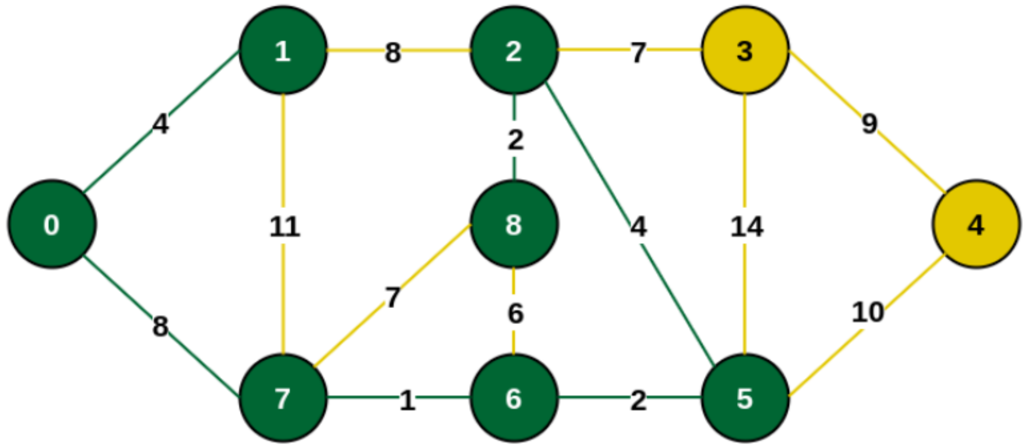
Minimum weighted edge from MST to other vertices is 7-6 with weight 1



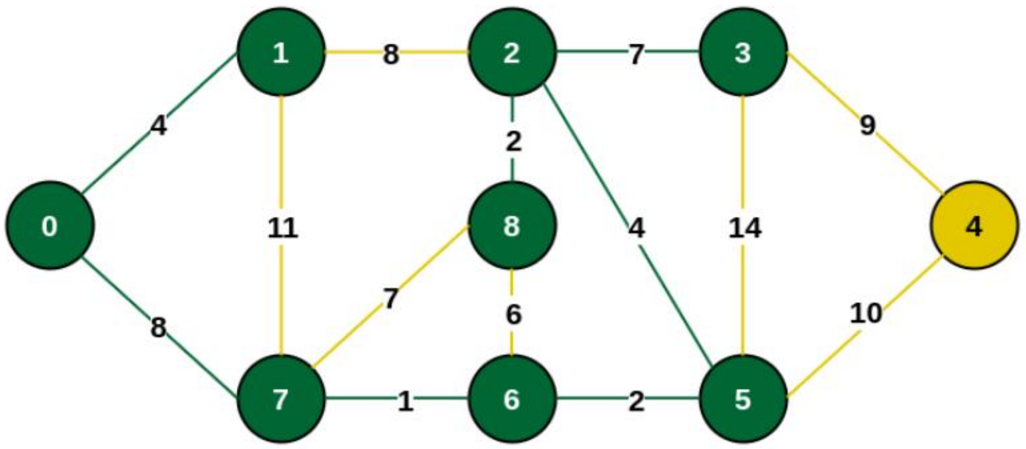
Minimum weighted edge from MST to other vertices is 6-5 with weight 2



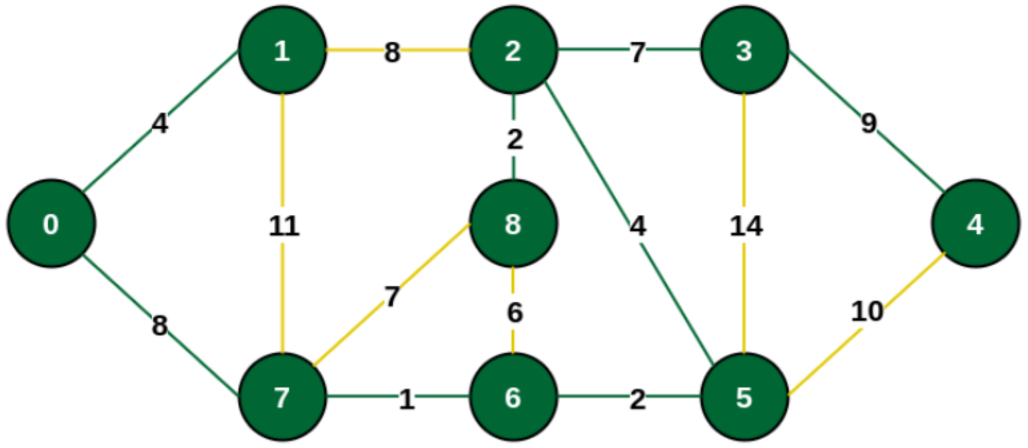
Minimum weighted edge from MST to other vertices is 5-2 with weight 4



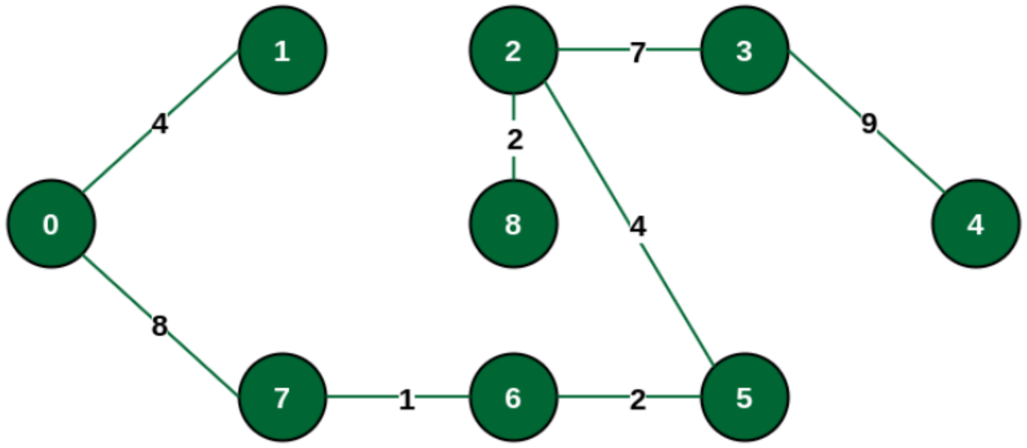
Minimum weighted edge from MST to other vertices is 2-8 with weight 2



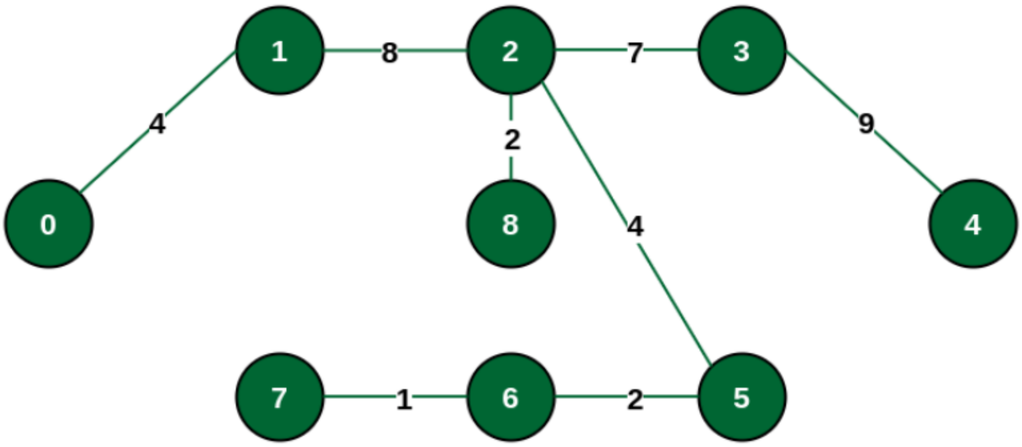
Minimum weighted edge from MST to other vertices is 2-3 with weight 7



Minimum weighted edge from MST to other vertices is 3-4 with weight 9



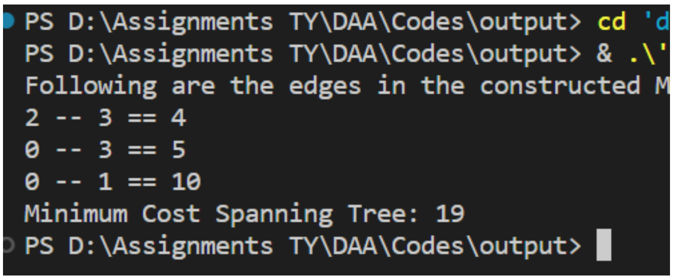
The final structure of MST



Alternative MST structure

2) Kruskal's algorithm using Min Heap

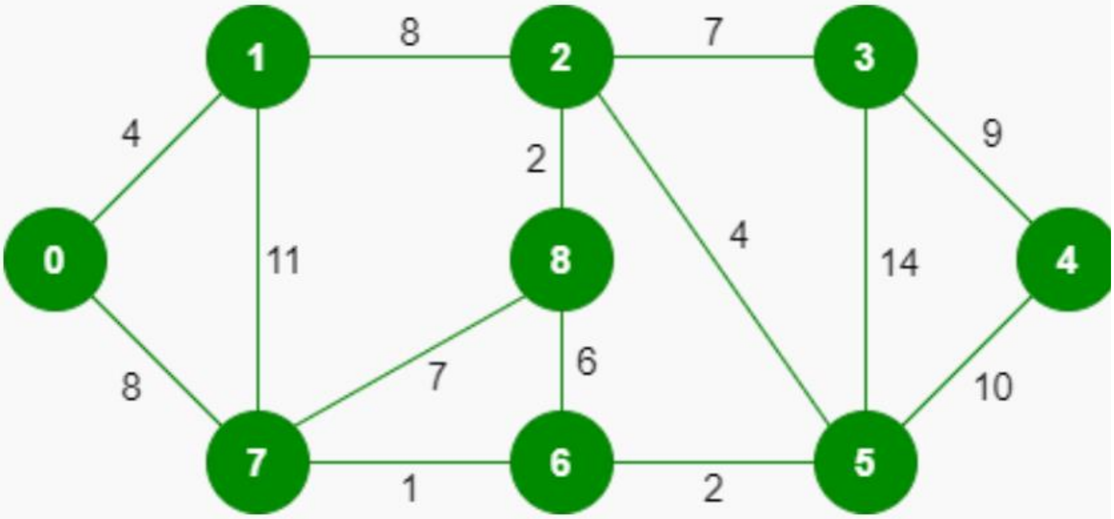
```
#include <stdio.h>
#include <stdlib.h>
int comparator(const void* p1, const void* p2)
{
    const int(*x)[3] = p1;
    const int(*y)[3] = p2;

    return (*x)[2] - (*y)[2];
}
void makeSet(int parent[], int rank[], int n)
{
    for (int i = 0; i < n; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}
int findParent(int parent[], int component)
{
    if (parent[component] == component)
        return component;

    return parent[component]
        = findParent(parent, parent[component]);
}
void unionSet(int u, int v, int parent[], int rank[], int n)
{
    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;
        rank[u]++;
    }
}
void kruskalAlgo(int n, int edge[n][3])
{
    qsort(edge, n, sizeof(edge[0]), comparator);

    int parent[n];
    int rank[n];
    makeSet(parent, rank, n);
    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 (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:
```



```
PS D:\Assignments TY\DAA\Codes\output> cd 'd
PS D:\Assignments TY\DAA\Codes\output> & .\'
Following are the edges in the constructed M
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19
PS D:\Assignments TY\DAA\Codes\output> 
```

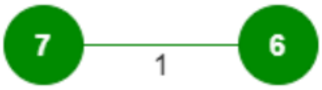
Simulation:

Input Graph:

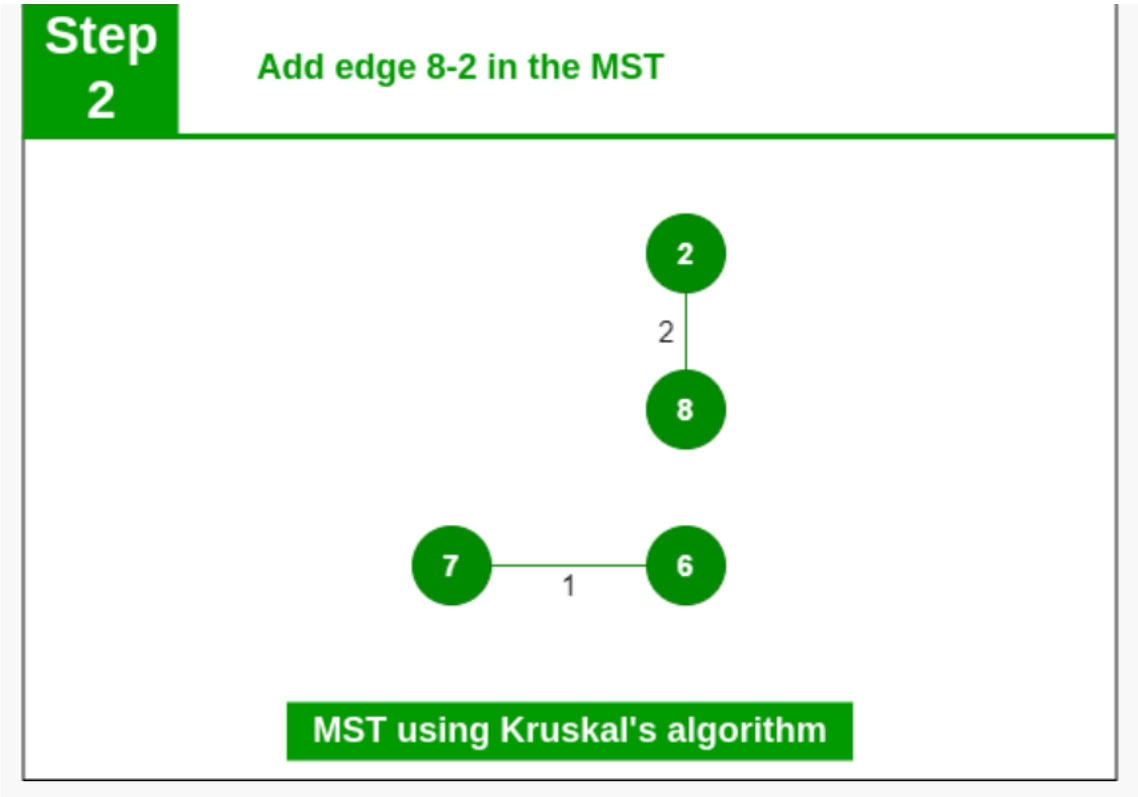


Step 1

Add edge 7-6 in the MST



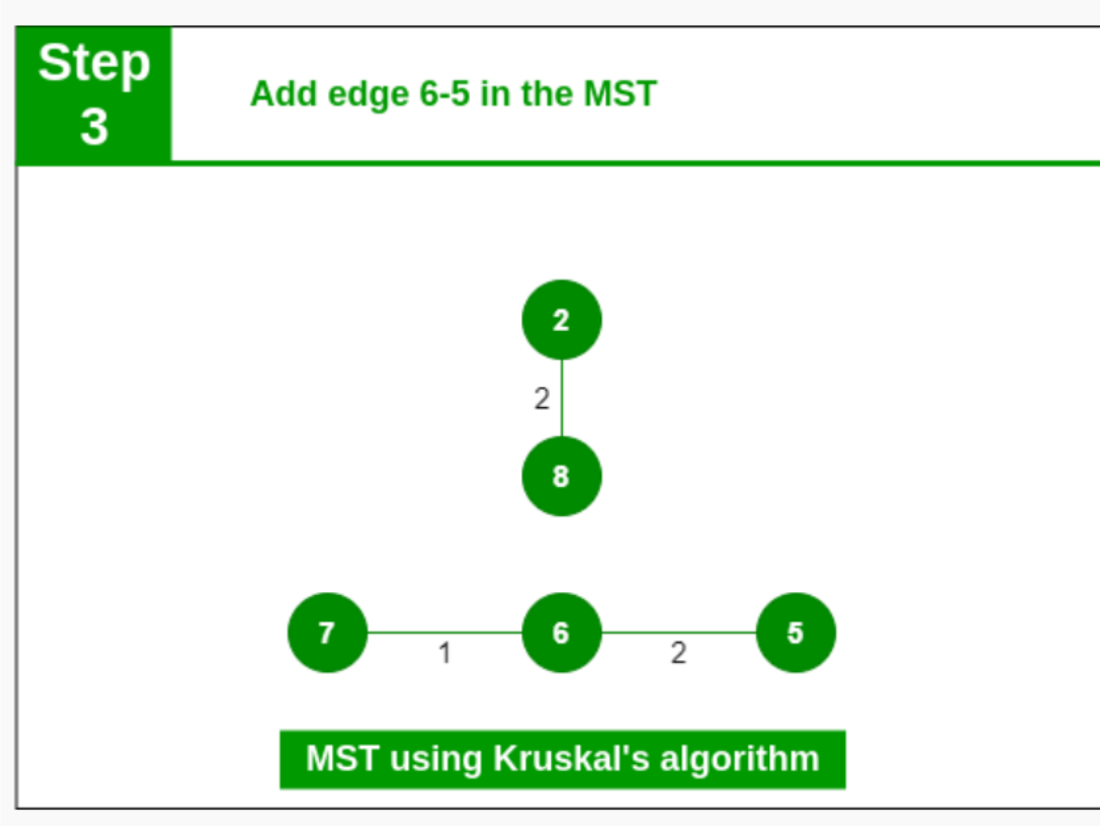
MST using Kruskal's algorithm



Step 2

Add edge 8-2 in the MST

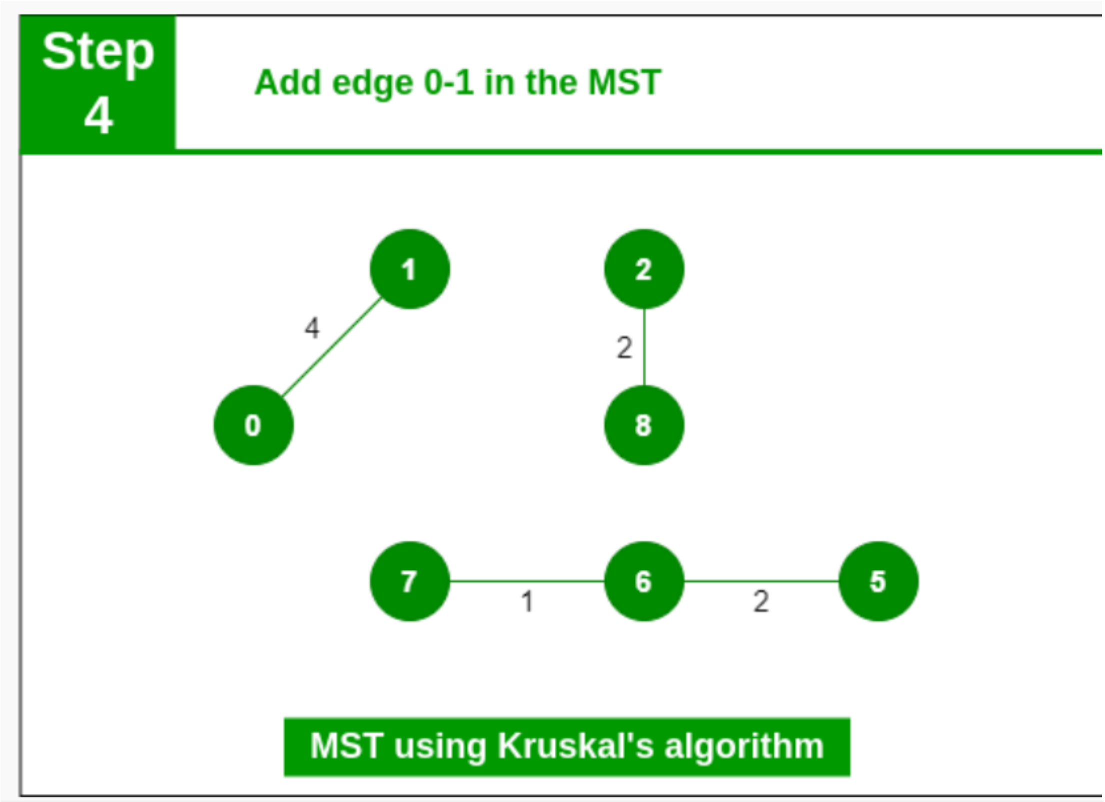
MST using Kruskal's algorithm



Step 3

Add edge 6-5 in the MST

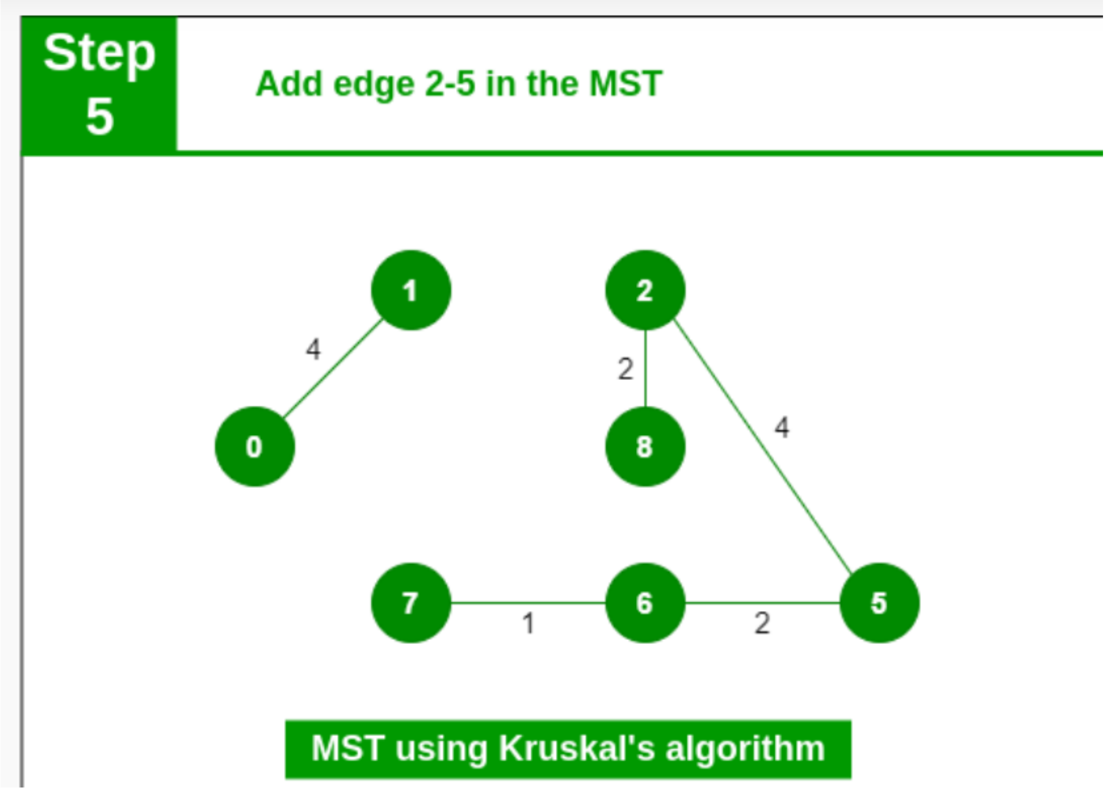
MST using Kruskal's algorithm



Step 4

Add edge 0-1 in the MST

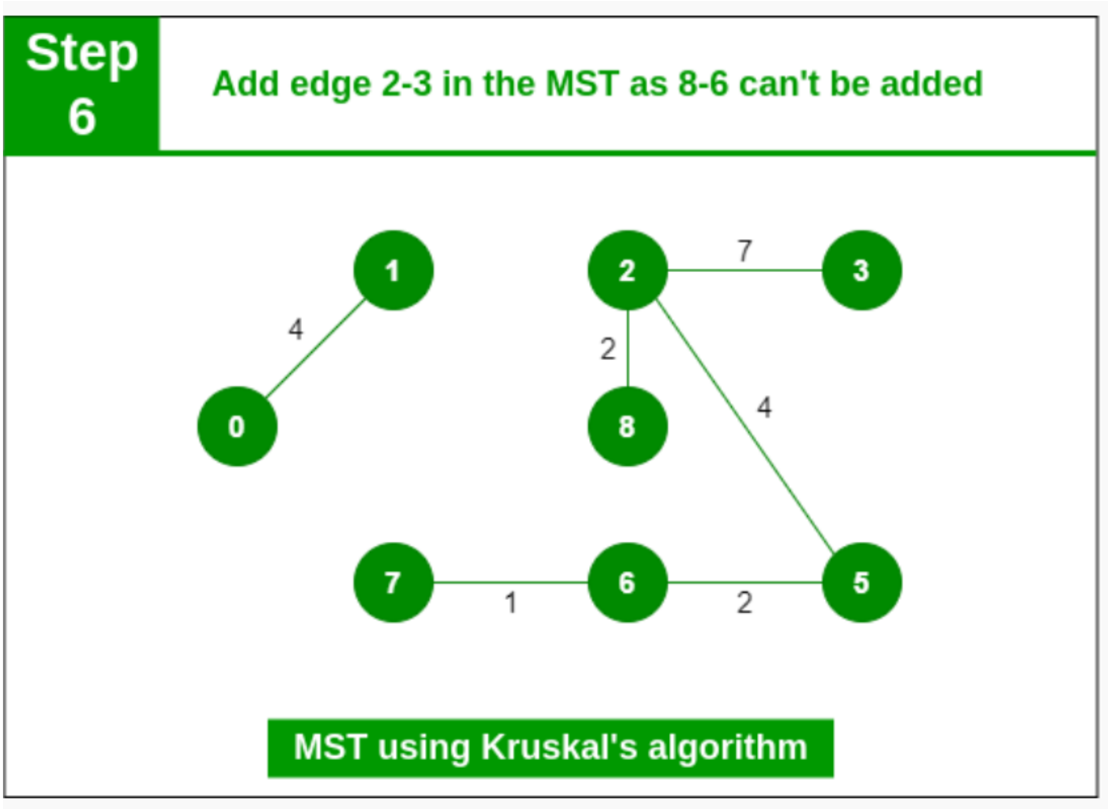
MST using Kruskal's algorithm



Step 5

Add edge 2-5 in the MST

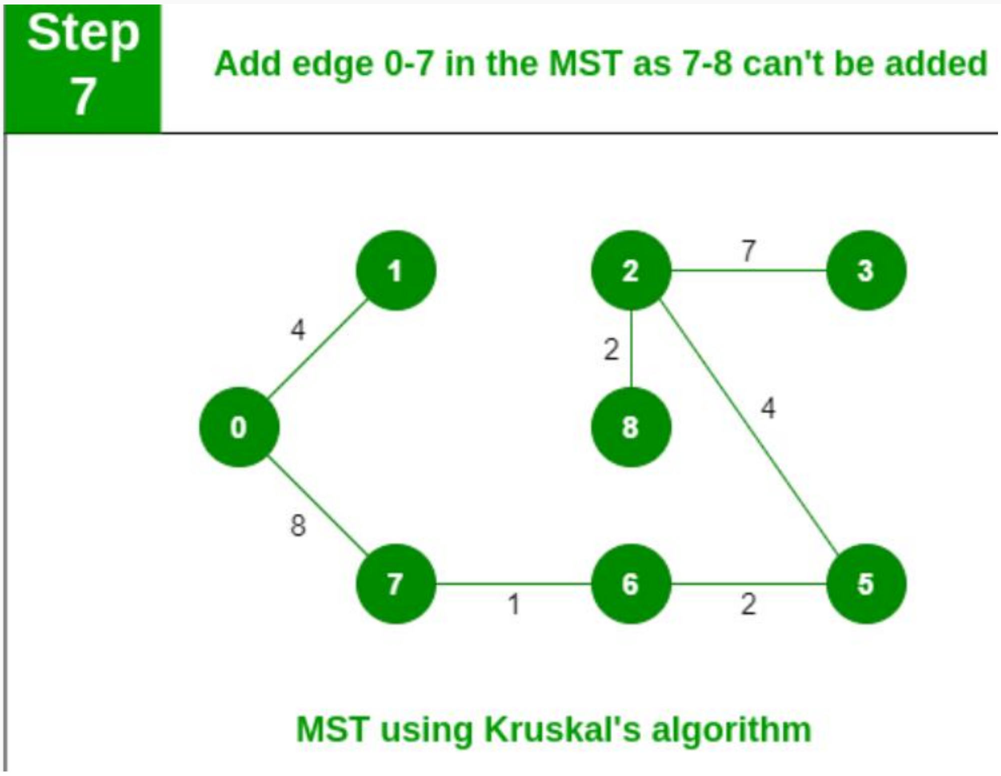
MST using Kruskal's algorithm



Step 6

Add edge 2-3 in the MST as 8-6 can't be added

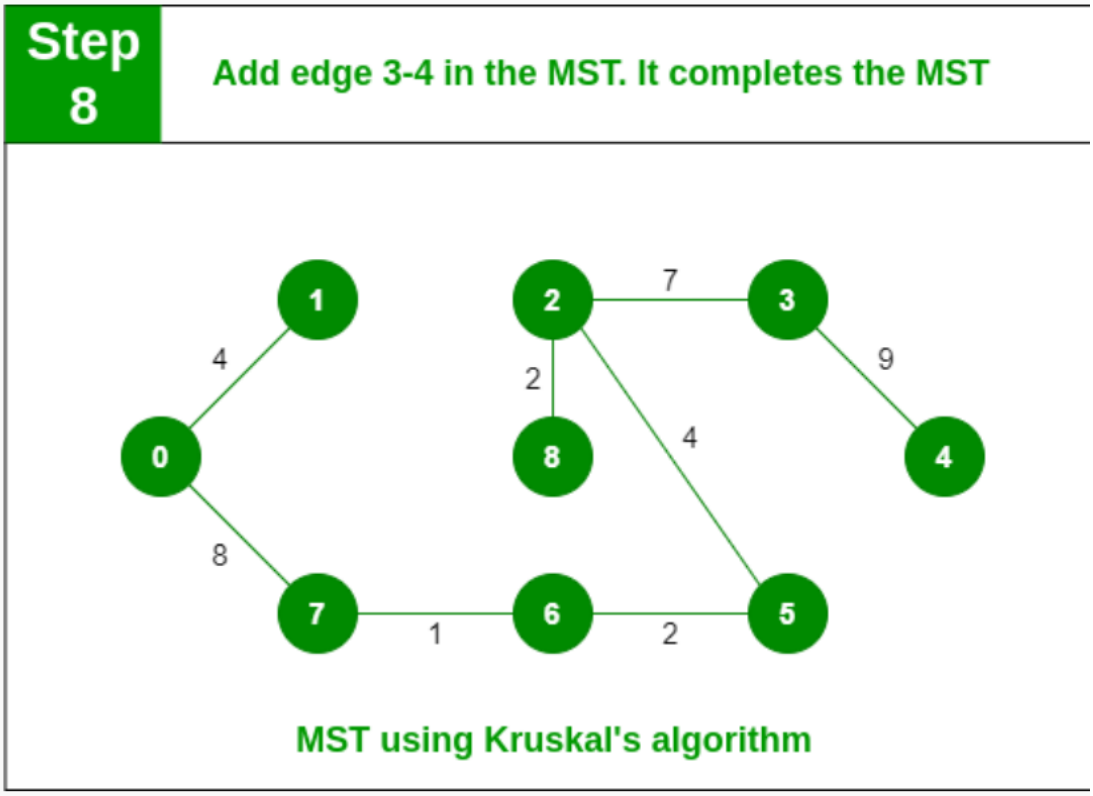
MST using Kruskal's algorithm



Step 7

Add edge 0-7 in the MST as 7-8 can't be added

MST using Kruskal's algorithm



Step 8

Add edge 3-4 in the MST. It completes the MST

MST using Kruskal's algorithm