Practical-9

Name:-Vishwajit Kate

Reg. No:-2020BIT011

# 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

# Code:-

// 2020BIT011

#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 },

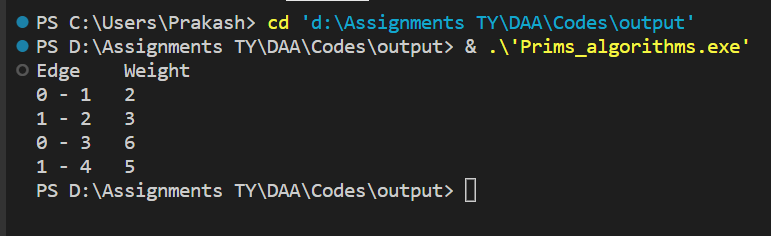
{ 6, 8, 0, 0, 9 },

{ 0, 5, 7, 9, 0 } };

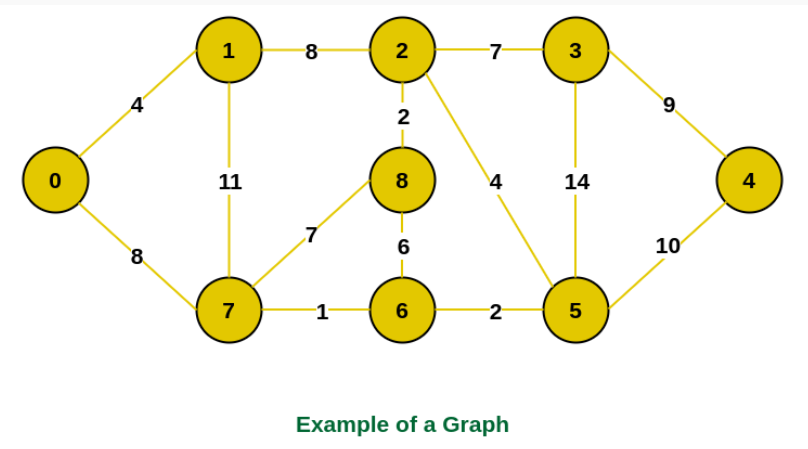
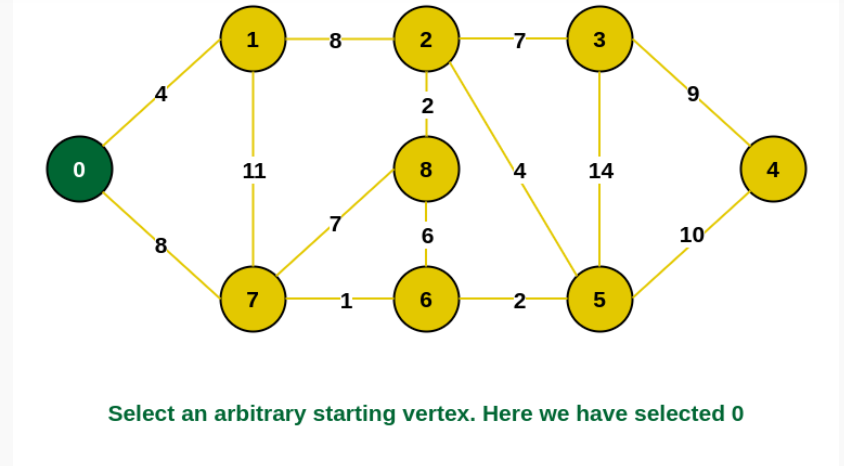
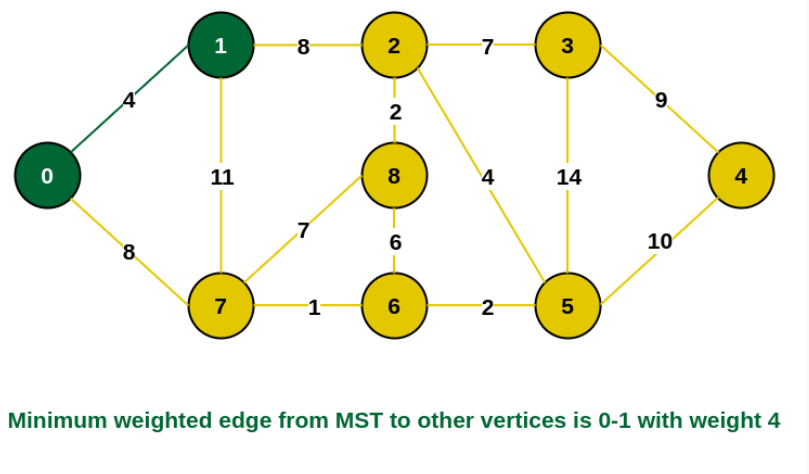
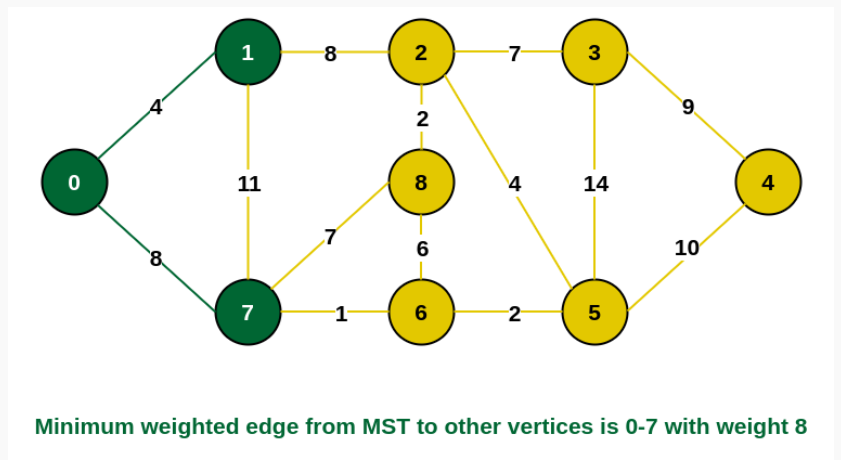
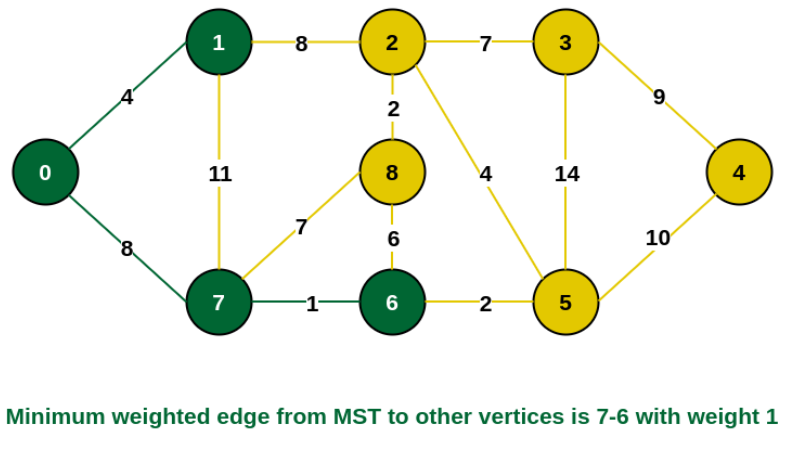
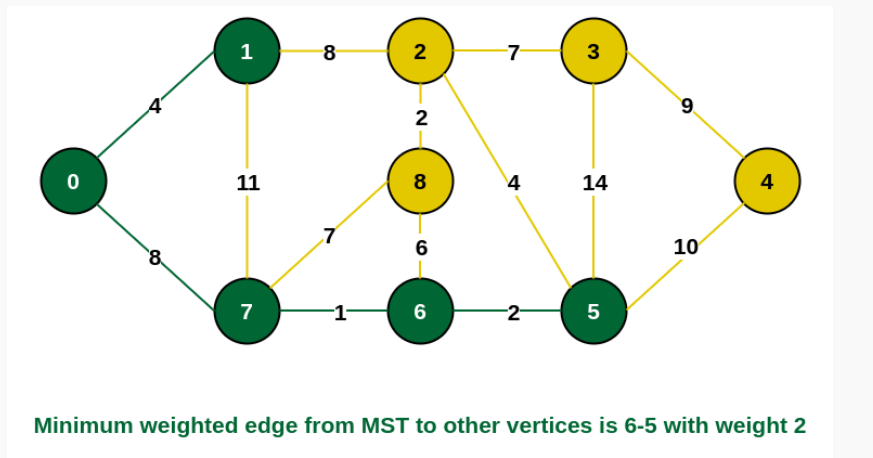
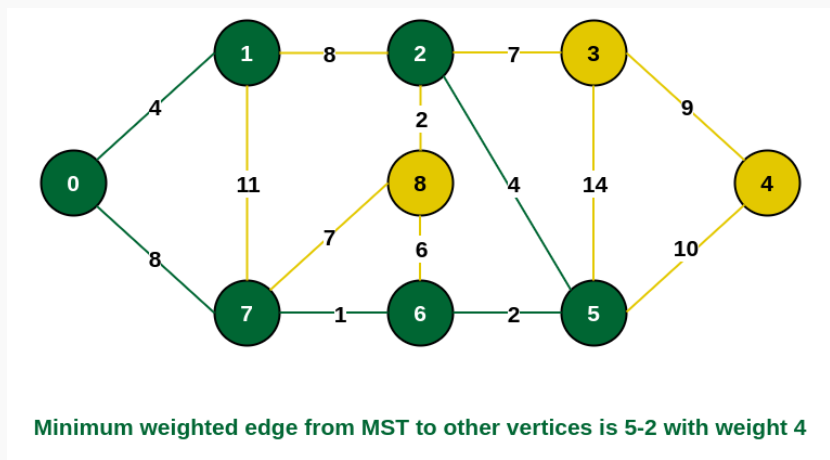
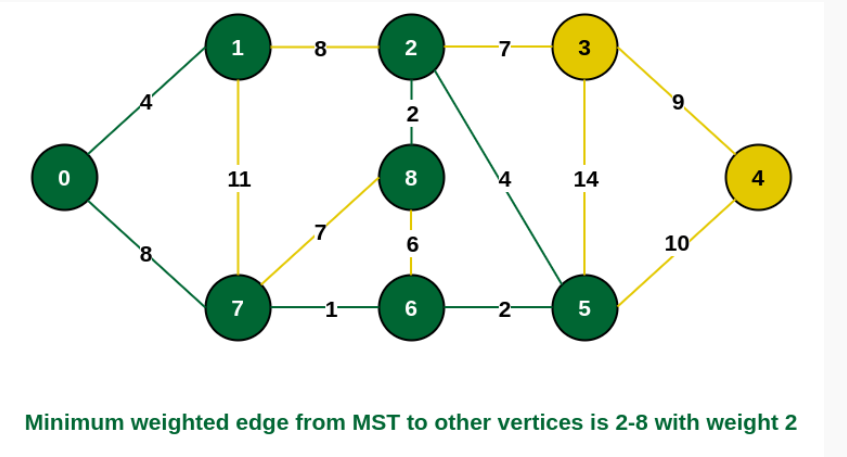
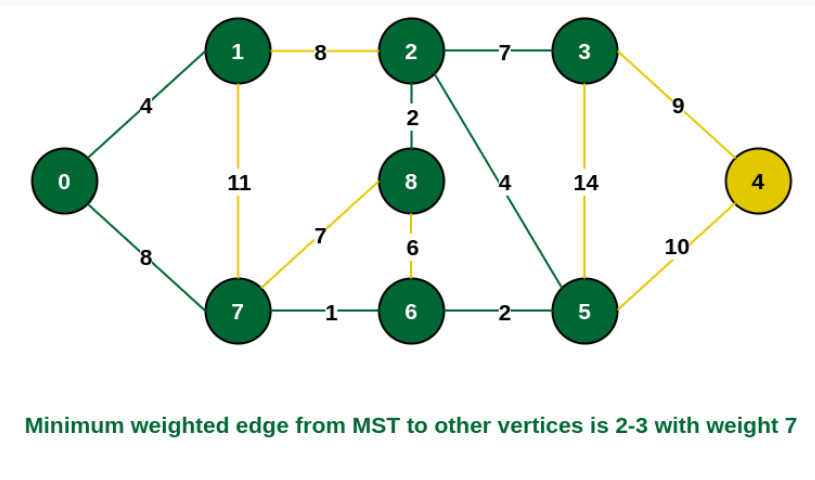
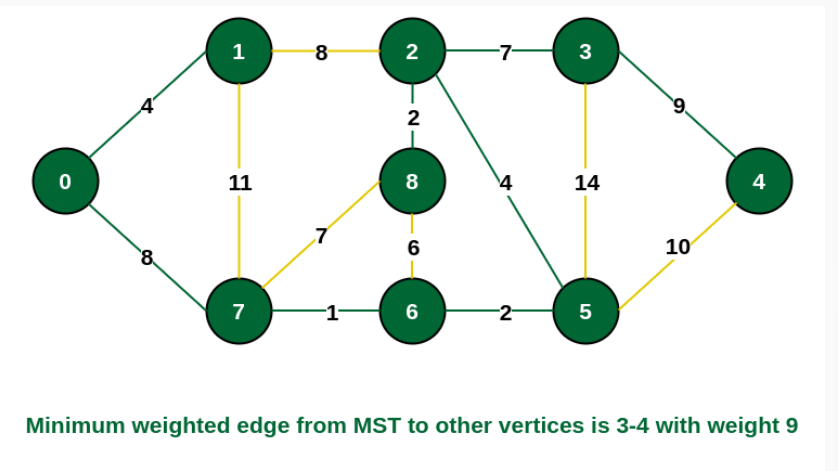
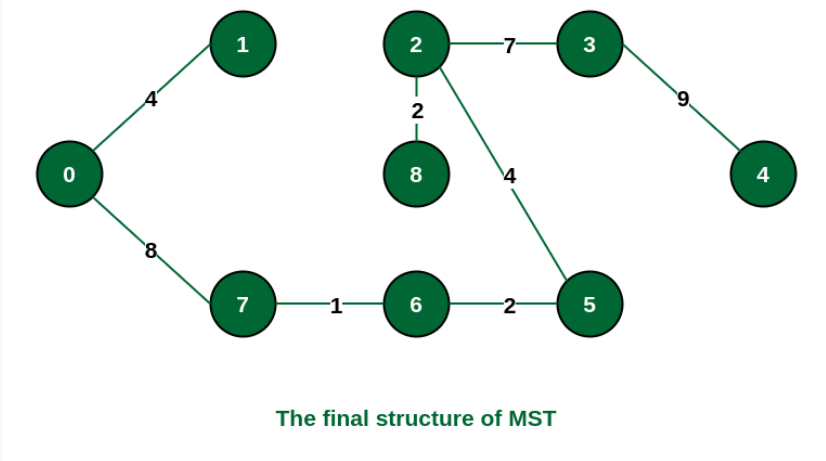
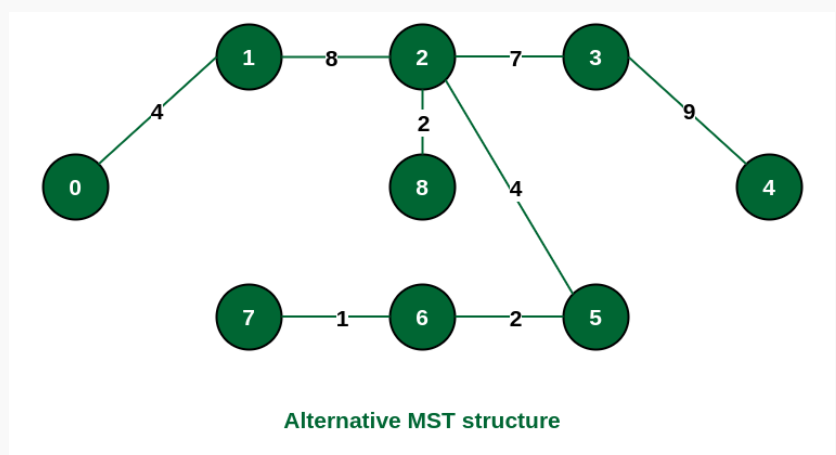
primMST(graph);

return 0;

}**Output:**

****

**Simulation:**

****           

# 2) Kruskal's algorithm using Min Heap

# Code:-

# // 2020BIT011

# #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 },

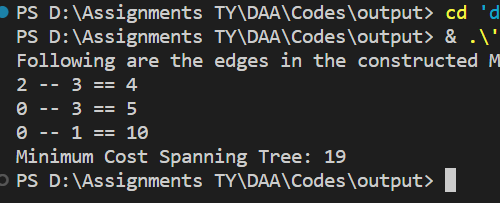
# { 1, 3, 15 },

# { 2, 3, 4 } };

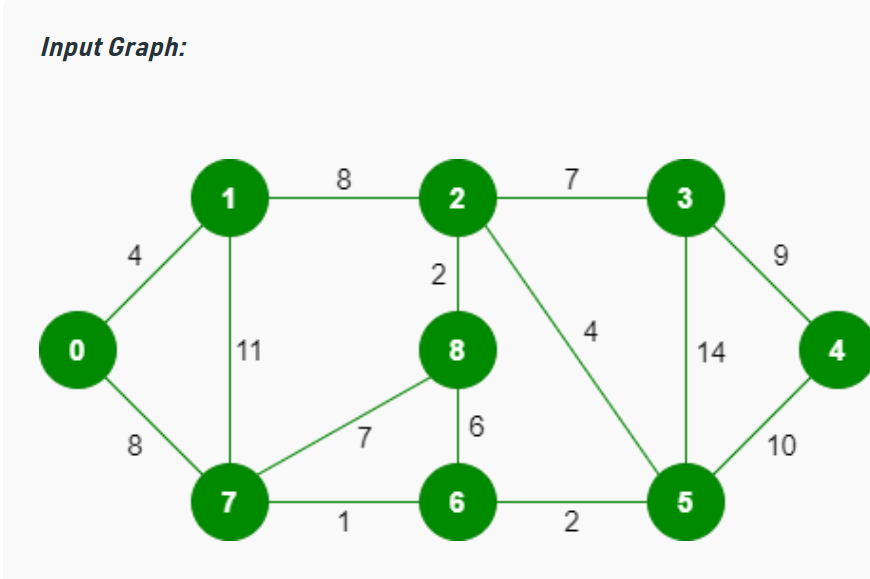
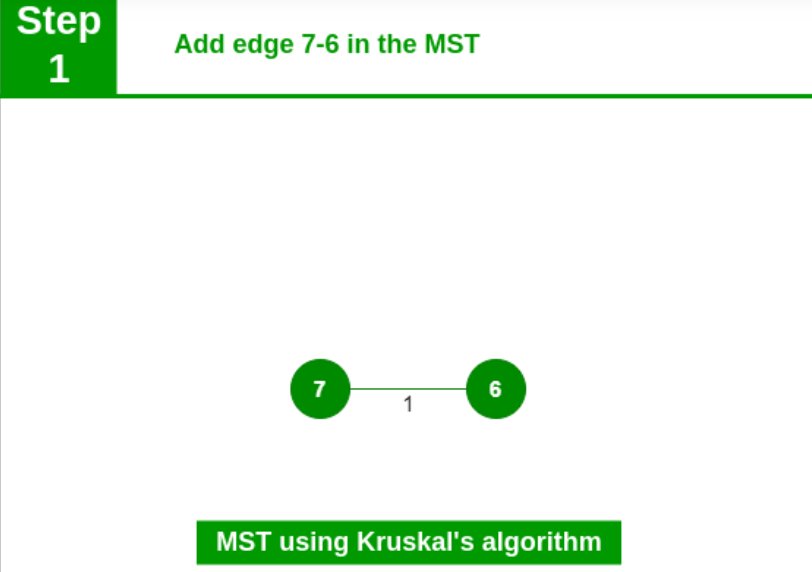
# kruskalAlgo(5, edge);

# return 0;

# }**Output:**

****

**Simulation:**

**** **** 