NAME- Aniket Kalbhor

PRN- 12210601

ROLL- CSB-48

CODE- 9a**)Implement Kruskals Algorithm to generate MST using Adjacency Matrix.**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 10

int findParent(int i, int parent[MAX\_VERTICES])

{

while (i != parent[i])

{

i = parent[i];

}

return i;

}

void getUnion(int i, int j, int parent[MAX\_VERTICES])

{

int a = findParent(i, parent);

int b = findParent(j, parent);

parent[b] = a;

}

void Kruskals(int G[MAX\_VERTICES][MAX\_VERTICES], int numVertices)

{

int minWeight = 0;

int edges = 0;

int parent[MAX\_VERTICES];

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

{

parent[i] = i;

}

while (edges < numVertices - 1)

{

int min = 999;

int a = -1;

int b = -1;

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

{

for (int j = 0; j < numVertices; j++)

{

if (G[i][j] < min && findParent(i, parent) != findParent(j, parent))

{

a = i;

b = j;

min = G[i][j];

}

}

}

if (a != -1 && b != -1)

{

getUnion(a, b, parent);

printf("%d edge: %d - %d = %d\n", edges, a, b, min);

minWeight += min;

edges++;

}

}

printf("Weight of MST is %d\n", minWeight);

}

int main()

{

int G[MAX\_VERTICES][MAX\_VERTICES];

int numVertices, numEdges, u, v, weight;

printf("Enter number of vertices: ");

scanf("%d", &numVertices);

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

{

for (int j = 0; j < numVertices; j++)

{

G[i][j] = 999;

}

}

printf("Enter the number of edges: ");

scanf("%d", &numEdges);

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

{

printf("Enter edge: ");

scanf("%d %d", &u, &v);

printf("Enter weight: ");

scanf("%d", &weight);

G[u][v] = G[v][u] = weight;

}

for (int k = 0; k < numVertices; k++)

{

for (int l = 0; l < numVertices; l++)

{

printf("%d ", G[k][l]);

}

printf("\n");

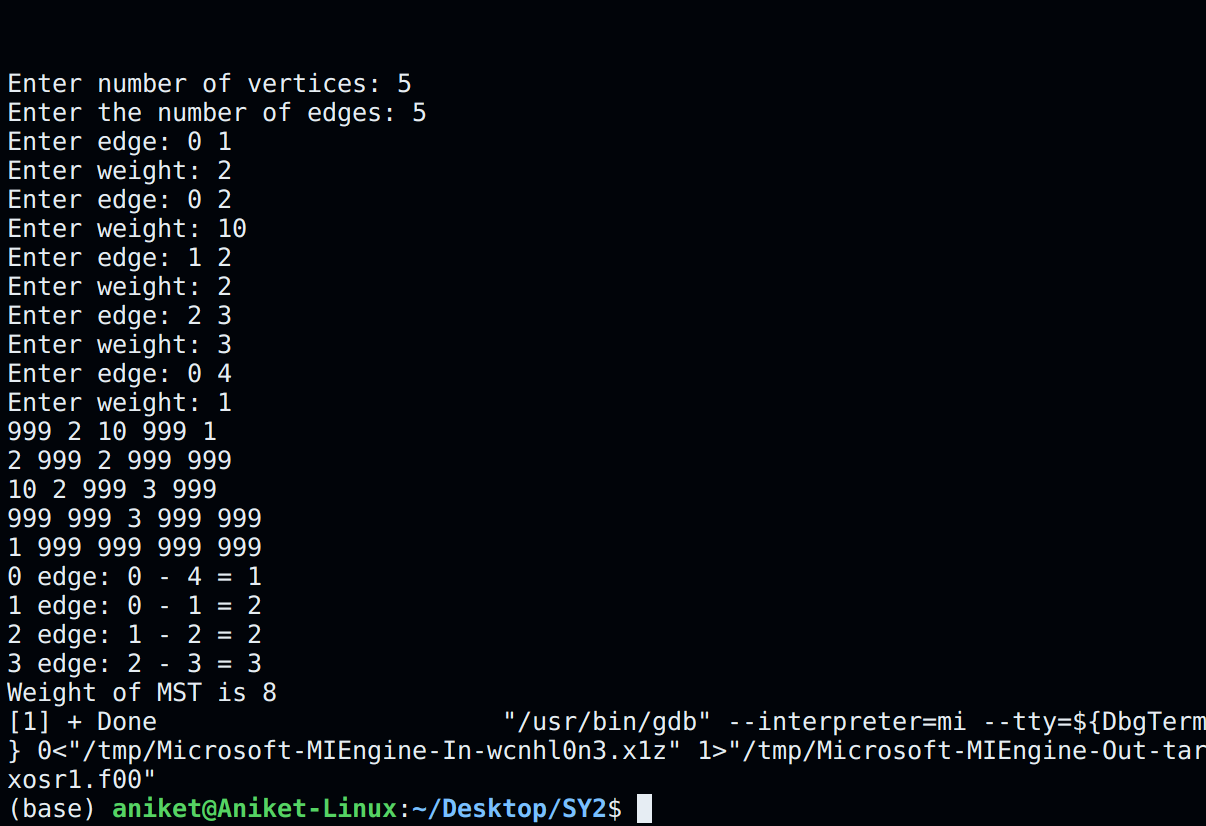
}

Kruskals(G, numVertices);

return 0;

}

**OUTPUT**



9a**)Implement Kruskals Algorithm to generate MST using Adjacency List.**

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int vertex;

int weight;

struct Node\* next;

};

struct Node\* a[10];

int parent[10];

int findParent(int i, int parent[10])

{

while (i != parent[i])

i = parent[i];

return i;

}

void getUnion(int i, int j, int parent[10])

{

int a, b;

a = findParent(i, parent);

b = findParent(j, parent);

parent[b] = a;

}

void Kruskals(struct Node\* a[10], int v)

{

int min = 999;

int edges = 0;

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

parent[i] = i;

while (edges < v - 1)

{

min = 999;

int min\_u = -1, min\_v = -1; // variables to store vertices of the edge with minimum weight

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

{

struct Node\* p = a[i];

while (p != NULL)

{

int u = i;

int v = p->vertex;

if (p->weight < min && findParent(u, parent) != findParent(v, parent))

{

min = p->weight;

min\_u = u;

min\_v = v;

}

p = p->next;

}

}

if (min\_u != -1 && min\_v != -1)

{

printf("Edge %d-%d Weight: %d\n", min\_u, min\_v, min);

getUnion(min\_u, min\_v, parent);

edges++;

}

}

}

void addEdge(int u, int v, int weight)

{

struct Node\* newV = (struct Node\*)malloc(sizeof(struct Node));

newV->vertex = v;

newV->weight = weight;

newV->next = NULL;

struct Node\* p = a[u];

if (p == NULL)

a[u] = newV;

else

{

while(p->next != NULL)

p = p->next;

p->next = newV;

}

}

void printAdjacencyList(int vnum)

{

struct Node\* p;

printf("Adjacency List\n");

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

{

printf("Vertex %d-> ", i);

p = a[i];

while (p != NULL)

{

printf("%d,%d\t", p->vertex, p->weight);

p = p->next;

}

printf("\n");

}

}

int main()

{

struct Node\* p;

int u, v, vnum, weight;

int count = 0;

printf("Enter the number of vertices:");

scanf("%d", &vnum);

printf("Enter the number of edges:");

int edges;

scanf("%d", &edges);

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

a[i] = NULL;

do {

count++;

printf("Enter edge: ");

scanf("%d",&u);

scanf("%d",&v);

printf("Enter weight: ");

scanf("%d",&weight);

addEdge(u, v, weight);

addEdge(v, u, weight);

} while(count < edges);

printAdjacencyList(vnum);

printf("Minimum Spanning Tree\n");

Kruskals(a, vnum);

return 0;

}

**OUTPUT**

