Program:

/\*primes\*/

#include <iostream>

using namespace std;

class graph

{

int G[20][20], n;

public:

void accept()

{

int i, j, e;

int src, dest, cost;

cout << "\nEnter the no. of vertices: ";

cin >> n;

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++){

G[i][j] = 0;

}

}

cout << "\nEnter the no. of Edges: ";

cin >> e;

for (i = 0; i < e; i++){

cout << "\nEnter Source: ";

cin >> src;

cout << "\nDestination: ";

cin >> dest;

cout << "\nCost: ";

cin >> cost;78

G[src][dest] = cost;

G[dest][src] = cost;

}

}

void display(){

int i, j;

for (i = 0; i < n; i++){

cout << "\n";

for (j = 0; j < n; j++){

cout << "\t" << G[i][j];

}

}

}

void prims()

{

int i, j, R[20][20];

int src, dest, cost, count, min;

int total = 0;

int visited[20];

for (i = 0; i < n; i++){

for (j = 0; j < n; j++){

if (G[i][j] == 0){

R[i][j] = 999;

}

else

R[i][j] = G[i][j];

}

}

for (i = 0; i < n; i++) {

visited[i] = 0;

}

cout << "\nEnter start vertex: ";79

cin >> src;

visited[src] = 1;

count = 0;

while (count < n - 1) {

min = 999;

for (i = 0; i < n; i++){

if (visited[i] == 1)

for (j = 0; j < n; j++){

if (visited[j] != 1){

if (min > R[i][j]){

min = R[i][j];

src = i;

dest = j;

}

}

}

}

cout << "\nEdge from " << src << " to " << dest << " \twith cost: " <<

min;

total = total + min;

visited[dest] = 1;

count++;

}

cout << "\nTotal Cost: " << total << "\n";

}

};

int main()

{

graph g;

g.accept();

g.display();

g.prims();

}80

**OUTPUT OF PRIMS ALGORITHM:**

Enter the no. of vertices: 4

Enter the no. of Edges: 5

Enter Source: 0

Destination: 1

Cost: 15

Enter Source: 1

Destination: 2

Cost: 15

Enter Source: 0

Destination: 2

Cost: 5

Enter Source: 3

Destination: 1

Cost: 2

Enter Source: 3

Destination: 2

Cost: 4081

0 15 5 0

15 0 15 2

5 15 0 40

0 2 40 0

Enter start vertex: 0

Edge from 0 to 2 with cost: 5

Edge from 0 to 1 with cost: 15

Edge from 1 to 3 with cost: 2

Total Cost: 22

**B] Kruskal’s Algorithm:**

#include <iostream>

#define INFINITY 999

using namespace std;

class kruskal{

typedef struct graph{

int v1, v2, cost;

}GR;

GR G[20];

public:

int tot\_edges , tot\_nodes;

void create();

void Spanning\_tree();

void get\_input();

int minimum(int);

};

void kruskal::get\_input(){

cout<<"\nEnter total number of nodes: ";

cin>>tot\_nodes;

cout<<"\nEnter total number of edges: ";

cin>>tot\_edges;

}

void kruskal::create(){82

for(int k=0; k<tot\_edges; k++){

cout<<"\nEnter edge v1: ";

cin>>G[k].v1;

cout<<"\nEnter edge v2: ";

cin>>G[k].v2;

cout<<"\nEnter corresponding cost: ";

cin>>G[k].cost;

}

}

int kruskal::minimum(int n){

int i,small, pos;

small=INFINITY;

pos=-1;

for(i=0; i<n; i++){

if(G[i].cost<small){

small=G[i].cost;

pos=i;

}

}

return pos;

}

void un(int i, int j, int parent[]){

if(i<j)

parent[j]=i;

else

parent[i]=j;

}

int find(int v2, int parent[]){

while(parent[v2]!=v2){

v2=parent[v2];

}

return v2;

}

void kruskal:: Spanning\_tree(){

int count, k, v1, v2, i, j, tree[10][10], pos, parent[10];

int sum;83

count=0;

k=0;

sum=0;

for(i=0; i<tot\_nodes; i++)

parent[i]=i;

while(count!=tot\_nodes-1){

pos=minimum(tot\_edges);

if(pos==-1)

break;

v1=G[pos].v1;

v2=G[pos].v2;

i=find(v1,parent);

j=find(v2,parent);

if(i!=j){

tree[k][0]=v1;

tree[k][1]=v2;

k++;

count++;

sum+=G[pos].cost;

un(i,j,parent);

}

G[pos].cost=INFINITY;

}

if(count==tot\_nodes-1){

cout<<"\nSpanning tree is: "<<endl;

for(i=0; i<tot\_nodes-1; i++){

cout<<"|"<<tree[i][0];

cout<<" ";

cout<<tree[i][1]<<"|"<<endl;

}

cout<<"cost of spanning tree is: "<<sum<<endl;

}

else{

cout<<"There is no spannin tree"<<endl;

}

}84

int main()

{

kruskal obj;

obj.get\_input();

obj.create();

obj.Spanning\_tree();

return 0;

}

**OUTPUT OF KRUSKAL’S ALGORITHM:**

Enter total number of nodes: 8

Enter total number of edges: 9

Enter edge v1: 1

Enter edge v2: 2

Enter corresponding cost: 5

Enter edge v1: 2

Enter edge v2: 3

Enter corresponding cost: 6

Enter edge v1: 4

Enter edge v2: 3

Enter corresponding cost: 2

Enter edge v1: 185

Enter edge v2: 4

Enter corresponding cost: 9

Enter edge v1: 3

Enter edge v2: 5

Enter corresponding cost: 5

Enter edge v1: 5

Enter edge v2: 6

Enter corresponding cost: 10

Enter edge v1: 6

Enter edge v2: 7

Enter corresponding cost: 7

Enter edge v1: 7

Enter edge v2: 8

Enter corresponding cost: 1

Enter edge v1: 8

Enter edge v2: 5

Enter corresponding cost: 1

Spanning tree is:

|7 8|86

|8 5|

|4 3|

|1 2|

|3 5|

|2 3|

|6 7|

cost of spanning tree is: 27