/*Represent any real world graph using adjacency list /adjacency matrix find minimum spanning tree using 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);
};

int find(int v2,int parent[])
{
while(parent[v2]!=v2)
{
v2=parent[v2];
}
return v2;
}

void un(int i,int j,int parent[])
{
if(i<j)
parent[j]=i;
else
parent[i]=j;
}

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()
{
for(int k=0;k<tot_edges;k++)
{
cout<<"\nEnter edges in (v1,v2) form: ";
```

```
cin>>G[k].v1>>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 kruskal::spanning_tree()
{
int count,k,v1,v2,i,j,tree[10][10],pos,parent[10];
int sum;
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: \n";
cout<<"\n---------------------------- \n";
for(i=0;i<tot_nodes-1;i++)
{
cout<<"|"<<tree[i][0];
cout<<" ";
cout<<tree[i][1]<<"|"<<endl;
}
cout<<"\n---------------------------- \n";
cout<<"cost of spanning tree is: "<<sum<<endl;
}
else
{
cout<<"there is no spanning tree "<<endl;
}
}

int main()
{
kruskal obj;
cout<<"\n\t Graph creation ";
obj.get_input();
obj.create();
obj.spanning_tree();
return 0;
}
```

Graph creation
Enter total number of nodes 5

Enter total number of edges 6

Enter edges in (v1,v2) form: 1 4

Enter corresponding cost 1

Enter edges in (v1,v2) form: 4 3

Enter corresponding cost 1

Enter edges in (v1,v2) form: 0 1

Enter corresponding cost 2

Enter edges in (v1,v2) form: 4 2

Enter corresponding cost 2

Enter edges in (v1,v2) form: 1 2

Enter corresponding cost 3

Enter edges in (v1,v2) form: 2 3

Enter corresponding cost 3

Spanning tree is:

|1 4|
|4 3|
|0 1|
|4 2|

---
cost of spanning tree is: 6

...Program finished with exit code 0
Press ENTER to exit console.