```
/*Name: Yash Jaware

Roll No.: 142

Batch: C2*/

#include <iostream>

#include<bits/stdc++.h>


using namespace std;


class disjointset

{

public:

   int djset[20];


   disjointset(int v)

   {

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

      {

         djset[i] = i;

      }

   }

            //to find the root

   int find_root(int v)

   {

      while(v != djset[v])

      {

         v = djset[v];

      }

      return v;

   }

            //to take union

   void take_union(int v1, int v2)
```

```
    {
        int r1 = find_root(v1);
        int r2 = find_root(v2);

                        //both v1 and v2 are pointing themselves
        if(v1 == r1 && v2 == r2)
        {
            djset[v1] = v2;
        }
        else if(v1 != r1 && v2 == r2)
        {
            djset[v2] = v1;
        }
        else if(v1 == r1 && v2!= r2)
        {
            djset[v1] = v2;
        }
        else if(v1 != r1 && v2 != r2)
        {
            djset[r1] = r2;
        }
    }
};

class edge
{
public:
    int v1;
    int v2;
    int wt;
};
```

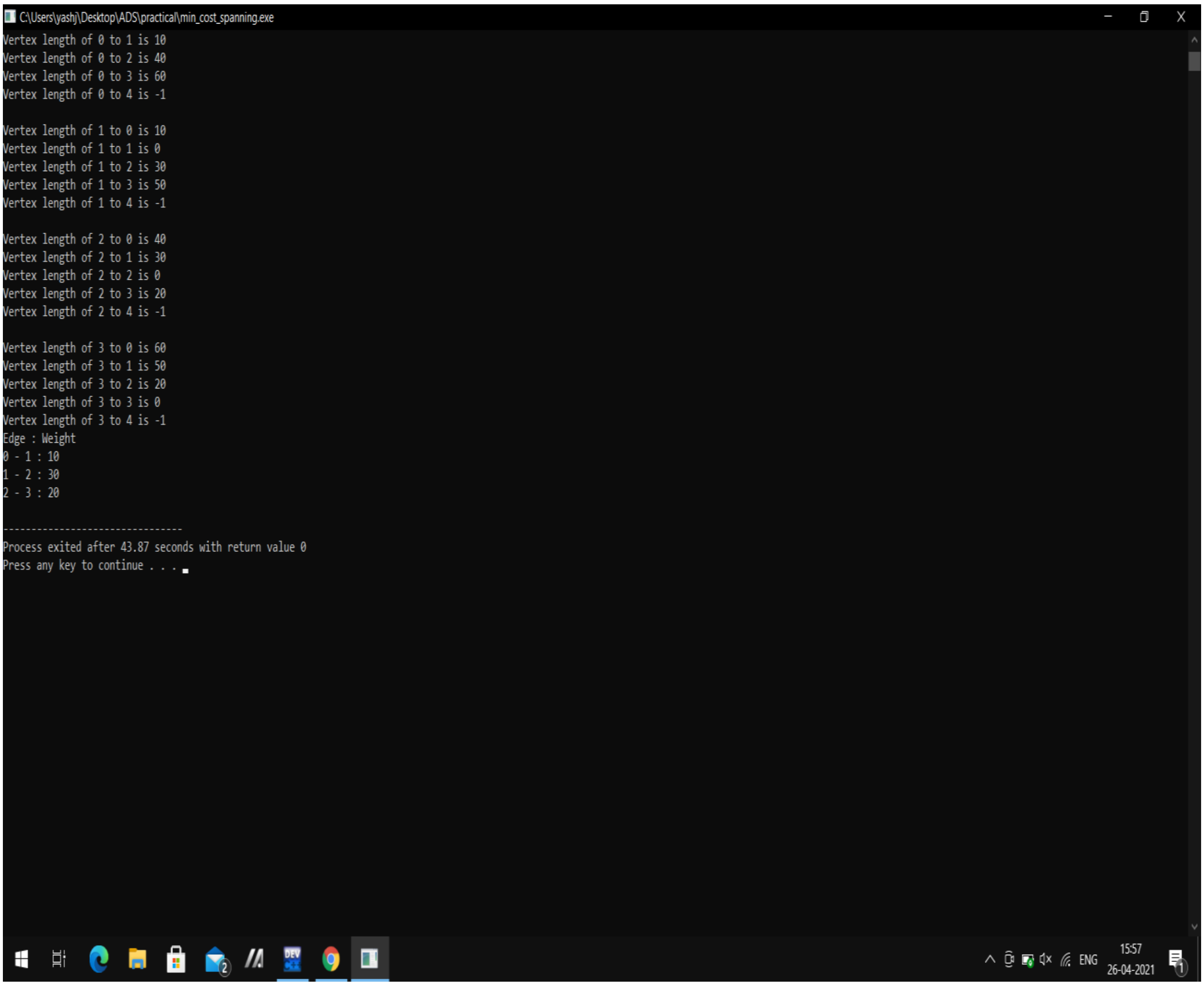
```
class graph
{
public:
   int v;
   int e;
   edge ed[20];

   graph(int vertices, int edges)
   {
      v = vertices;
      e = edges;
   }

   void accept_graph();
   void display_graph();
   void kruskal_mst();
   void sort_edges();

};

//Bubble sort to sort in kruskals
void graph::sort_edges()
{
   edge temp;
   for(int i=0;i<e;i++)
   {
      for(int j=0;j<e-i-1;j++)
      {
         if(ed[j].wt > ed[j+1].wt)
         {
            temp.v1 = ed[j].v1;
            temp.v2 = ed[j].v2;
            temp.wt = ed[j].wt;

            ed[j].v1 = ed[j+1].v1;
            ed[j].v2 = ed[j+1].v2;
            ed[j].wt = ed[j+1].wt;

            ed[j+1].v1 = temp.v1;
            ed[j+1].v2 = temp.v2;
            ed[j+1].wt = temp.wt;
          }
       }
   }
}

//function for kruskals algorithm
void graph::kruskal_mst()
{
   edge mst[20];
   int mst_ctr = 0;
   int mst_cost = 0;
   disjointset dj(v);
   sort_edges();
   cout<<"\n Edges after sorting: ";
   display_graph();
   cout<<"\n";

   for(int i=0;i<e;i++)
   {
      int r1 = dj.find_root(ed[i].v1);
```

```
        int r2 = dj.find_root(ed[i].v2);
        if(r1 != r2)
        {
            mst[mst_ctr].v1 = ed[i].v1;
            mst[mst_ctr].v2 = ed[i].v2;
            mst[mst_ctr].wt = ed[i].wt;
            mst_ctr++;
            mst_cost = mst_cost + ed[i].wt;
            dj.take_union(ed[i].v1,ed[i].v2);
        }
    }

    cout<<"\n MST is : ";
    for(int i=0;i<mst_ctr;i++)
    {
        cout<<"\n   "<<mst[i].v1<<"     "<<mst[i].v2<<"     "<<mst[i].wt;
    }
    cout<<"\n Total cost of MST is: "<<mst_cost;

}

//to accept the value of graph
void graph::accept_graph()
{
    for(int i=0;i<e;i++)
    {
        cout<<"\n Enter vertice 1 :";
        cin>>ed[i].v1;
        cout<<"\n Enter vertice 2 :";
        cin>>ed[i].v2;
        cout<<"\n Enter weight :";
        cin>>ed[i].wt;
    }
}

//to display the graph
void graph::display_graph()
{
    for(int i=0;i<e;i++)
    {
        cout<<"\n   "<<ed[i].v1<<"     "<<ed[i].v2<<"     "<<ed[i].wt;
    }
}

//function for prims algorithm
void prims(){
            cout<<"Enter The number of nodes of Graph-> ";
            int nodes;
            cin>>nodes;
            int graph[nodes+1][nodes+1];
            memset(graph,-1,sizeof(graph));
            for(int i=0;i<=nodes;i++)
            {
                        graph[i][i] = 0;
            }
            int i = 0 ;
    for (int i = 0; i < nodes; i++)
            {
                        int j =0;
                        for (int j = 0; j < nodes; j++)
                        {
                                    if(graph[i][j]==-1)
        {
            int data;
            cout<<"\nEnter the Vertex length for "<<i<<" to "<<j<<" -> ";
            cin>>data;
            graph[i][j] = graph[j][i] = data;
        }
    }
}
cout<<"\n\nGraph is shown below as";
i = 0;
for (int i = 0; i < nodes; i++)
{
    int j =0;
    for (int j = 0; j <= nodes; j++)
    {
        cout<<"\nVertex length of "<<i<<" to "<<j<<" is "<<graph[i][j]<<" ";

    }

    cout<<endl;
}

int selected[nodes];
for(int i=0;i<nodes;i++){
    selected[i]=false;
}
int no_edge=0;
selected[0]=true;

int x; // vertex 1
int y; // vertex 2
```

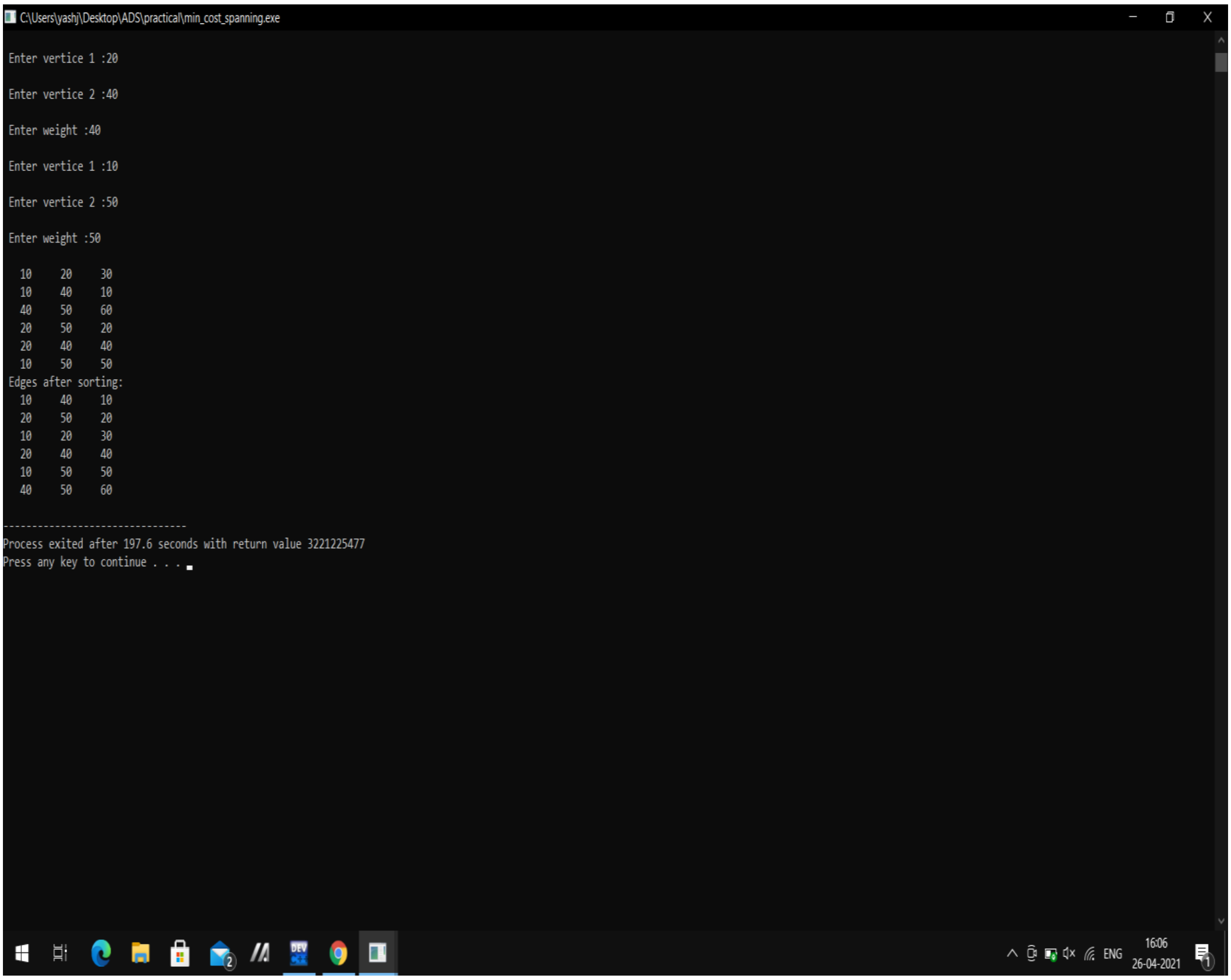
```
    cout << "Edge"<< " : " "Weight";
    cout << endl;

    while (no_edge < nodes - 1){
        int min=INT_MAX;
        x = 0;
        y = 0;

        for(int i=0;i<nodes;i++){
            if(selected[i]==true){
                for(int j=0;j<nodes;j++){
                    if(selected[j]==false && graph[i][j]!=0){
                        if(min>graph[i][j]){
                            min=graph[i][j];
                            x=i;
                            y=j;
                        }
                    }

                }
            }
        }
        cout << x << " - " << y << " : "<<graph[x][y]<<endl;
        selected[y]=true;
        no_edge++;



    }
}
int main(){
```

```
int choice;
cout<<"1.Prims Algorithm\n";
cout<<"2.Kruskals Algorithm";
cout<<"\n\nEnter choice : ";
cin>>choice;

switch(choice){
        case 1:
                cout<<"----------------------------------";
                cout<<"\nkruskal's algorithm\n";
                cout<<"----------------------------------\n";
                prims();
                break;
        case 2:
                cout<<"----------------------------------";
                cout<<"\nkruskal's algorithm\n";
                cout<<"----------------------------------\n";
                int v, e;
                cout<<"\n Enter the number of vertics : ";
                cin>>v;
                cout<<"\n Enter the number of edges : ";
                cin>>e;

                graph g(v, e);
                g.accept_graph();
                g.display_graph();
                g.kruskal_mst();

                break;

}
```

```
}
```



```
Vertex length of 0 to 1 is 10
Vertex length of 0 to 2 is 40
Vertex length of 0 to 3 is 60
Vertex length of 0 to 4 is -1

Vertex length of 1 to 0 is 10
Vertex length of 1 to 1 is 0
Vertex length of 1 to 2 is 30
Vertex length of 1 to 3 is 50
Vertex length of 1 to 4 is -1

Vertex length of 2 to 0 is 40
Vertex length of 2 to 1 is 30
Vertex length of 2 to 2 is 0
Vertex length of 2 to 3 is 20
Vertex length of 2 to 4 is -1

Vertex length of 3 to 0 is 60
Vertex length of 3 to 1 is 50
Vertex length of 3 to 2 is 20
Vertex length of 3 to 3 is 0
Vertex length of 3 to 4 is -1
Edge : Weight
0 - 1 : 10
1 - 2 : 30
2 - 3 : 20

--------------------------------
Process exited after 43.87 seconds with return value 0
Press any key to continue . . .
```



```
 Enter vertice 1 :20

 Enter vertice 2 :40

 Enter weight :40

 Enter vertice 1 :10

 Enter vertice 2 :50

 Enter weight :50

   10     20     30
   10     40     10
   40     50     60
   20     50     20
   20     40     40
   10     50     50
 Edges after sorting:
   10     40     10
   20     50     20
   10     20     30
   20     40     40
   10     50     50
   40     50     60

--------------------------------
Process exited after 197.6 seconds with return value 3221225477
Press any key to continue . . .
```