Kruskal’s Algorithm

Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph. Kruskal's algorithm follows a greedy approach which finds an optimum solution at every stage instead of focusing on a global optimum.

The Kruskal's algorithm is given as follows.

Algorithm

- Step 1: Create a forest in such a way that each graph is a separate tree.
- Step 2: Create a priority queue Q that contains all the edges of the graph.
- Step 3: Repeat Steps 4 and 5 while Q is NOT EMPTY
- Step 4: Remove an edge from Q
- Step 5: IF the edge obtained in Step 4 connects two different trees, then Add it to the forest (for combining two trees into one tree).
  ELSE
  Discard the edge
- Step 6: END

Code -

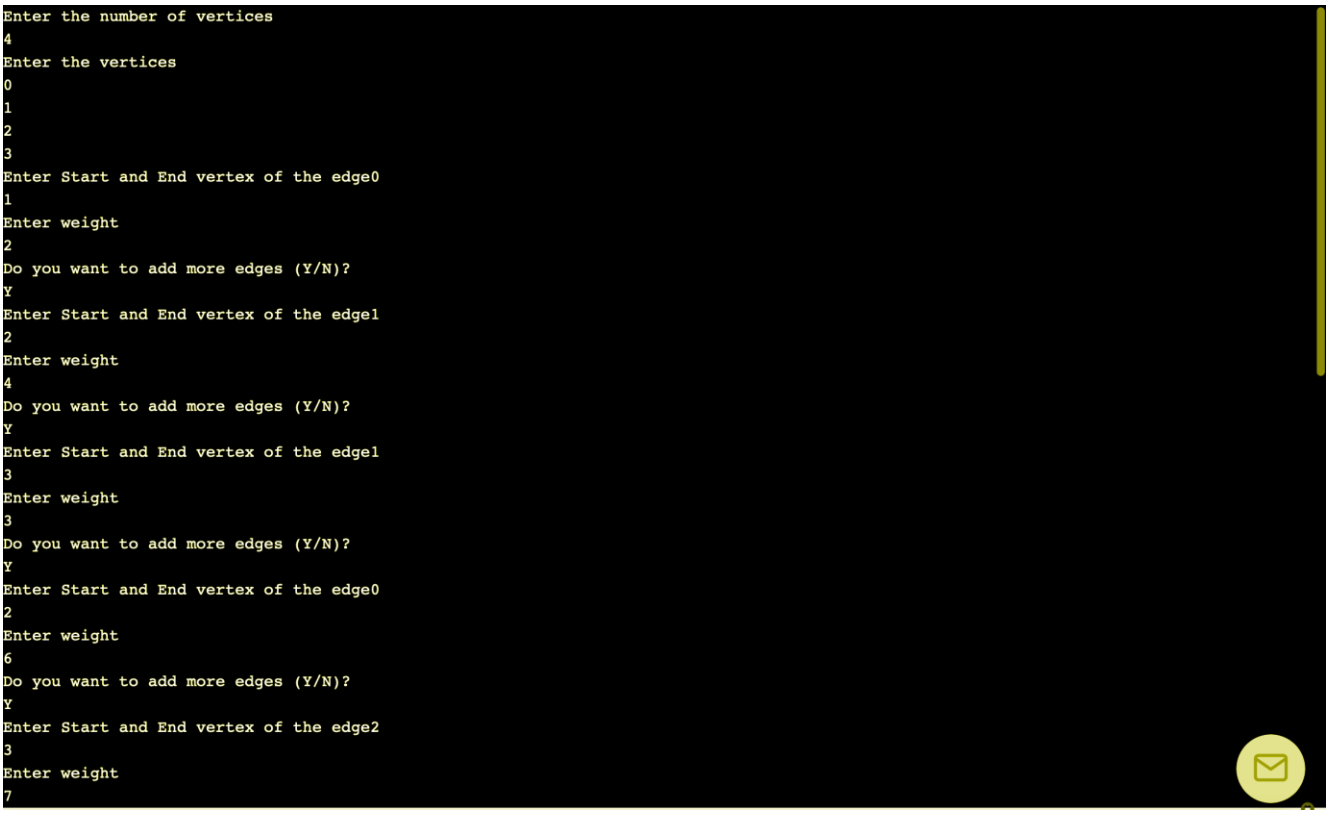
```
#include<iostream>
#include<string.h>
#include<bits/stdc++.h>
using namespace std;
class Graph
{
      char vertices[10][10];
      int cost[10][10],no;
      public:
      Graph();
      void creat_graph();
      void display();
      int Position(char[]);
      void kruskal_algo();
```

```
};
Graph::Graph()
{
no=0;
for(int i=0;i<10;i++)
for(int j=0;j<10;j++)
{
cost[i][j]=999;
}
}

void Graph::creat_graph()
{
char ans,Start[10],End[10];
int wt,i,j;
cout<<"Enter the number of vertices"<<endl;
cin>>no;
cout<<"Enter the vertices"<<endl;
for(i=0;i<no;i++)
      cin>>vertices[i];
do
{
cout<<"Enter Start and End vertex of the edge";
cin>>Start>>End;
cout<<"Enter weight"<<endl;
cin>>wt;
i=Position(Start);
j=Position(End);
cost[i][j]=cost[j][i]=wt;
cout<<"Do you want to add more edges (Y/N)?"<<endl;
cin>>ans;
}while(ans=='y' || ans=='Y');
}
void Graph::display()
{
int i,j;
cout<<"Cost matrix"<<endl;
for(i=0;i<no;i++)
{
cout<<endl;
for(j=0;j<no;j++)
cout<<"\t"<<cost[i][j]<<endl;
}
```

```
}

int Graph::Position(char key[10])
{
int i;
for(i=0;i<10;i++)
if(strcmp(vertices[i],key)==0)
return i;
return -1;
}
void Graph::kruskal_algo()
{
int i,j,v[10]={0},x,y,Total_cost=0,min,gr=1,flag=0,temp,d;
while(flag==0)
{
min=999;
for(i=0;i<no;i++)
{
      for(j=0;j<no;j++)
      {
      if(cost[i][j]<min)
            {
      min=cost[i][j];
      x=i;
      y=j;
            }
      }
}

if(v[x]==0 && v[y]==0)
{
      v[x]=v[y]=gr;
      gr++;
}
else if(v[x]!=0 && v[y]==0)
      v[y]=v[x];
else if(v[x]==0 && v[y]!=0)
      v[x]=v[y];
else
{
      if(v[x]!=v[y])
      {
      d=v[x];
```

```
            for(i=0;i<no;i++)
            {
            if(v[i]==d)
            v[i]=v[y];
            }
            }
      }

      cost[x][y]=cost[y][x]=999;
      Total_cost=Total_cost+min;
      cout<<"\n\t"<<vertices[x]<<"\t"<<vertices[y]<<"\t"<<min<<endl;

            temp=v[0]; flag=1;
            for(i=0;i<no;i++)
            {
            if(temp!=v[i])
            {
            flag=0;
            break;
            }
            }
      }
      cout<<"Total cost of the tree"<<Total_cost<<endl;
      }
      int main()
      {
      clock_t start, end;
      start = clock();
      Graph g;
      g.creat_graph();
      g.display();
      cout<<"Minimum Spanning tree"<<endl;
      cout<<"Source vertex\tDestination vertex\tWeight\n"<<endl;;
      g.kruskal_algo();
      end = clock();
      double time_taken = double(end - start) / double(CLOCKS_PER_SEC);
      cout << "Time taken by program is"<<fixed<< time_taken
<<setprecision(5);
      cout << "sec" << endl;
      return 0;
      }
```

OUTPUT


Enter the number of vertices
4
Enter the vertices
0
1
2
3
Enter Start and End vertex of the edge0
1
Enter weight
2
Do you want to add more edges (Y/N)?
Y
Enter Start and End vertex of the edge1
2
Enter weight
4
Do you want to add more edges (Y/N)?
Y
Enter Start and End vertex of the edge1
3
Enter weight
3
Do you want to add more edges (Y/N)?
Y
Enter Start and End vertex of the edge0
2
Enter weight
6
Do you want to add more edges (Y/N)?
Y
Enter Start and End vertex of the edge2
3
Enter weight
7

```
Do you want to add more edges (Y/N)?
N
Cost matrix

        999
        2
        6
        999

        2
        999
        4
        3

        6
        4
        999
        7

        999
        3
        7
        999
Minimum Spanning tree
Source vertex   Destination vertex      Weight


        0       1       2

        1       3       3

        1       2       4
Total cost of the tree9
Time taken by program is0.001389sec
```