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.

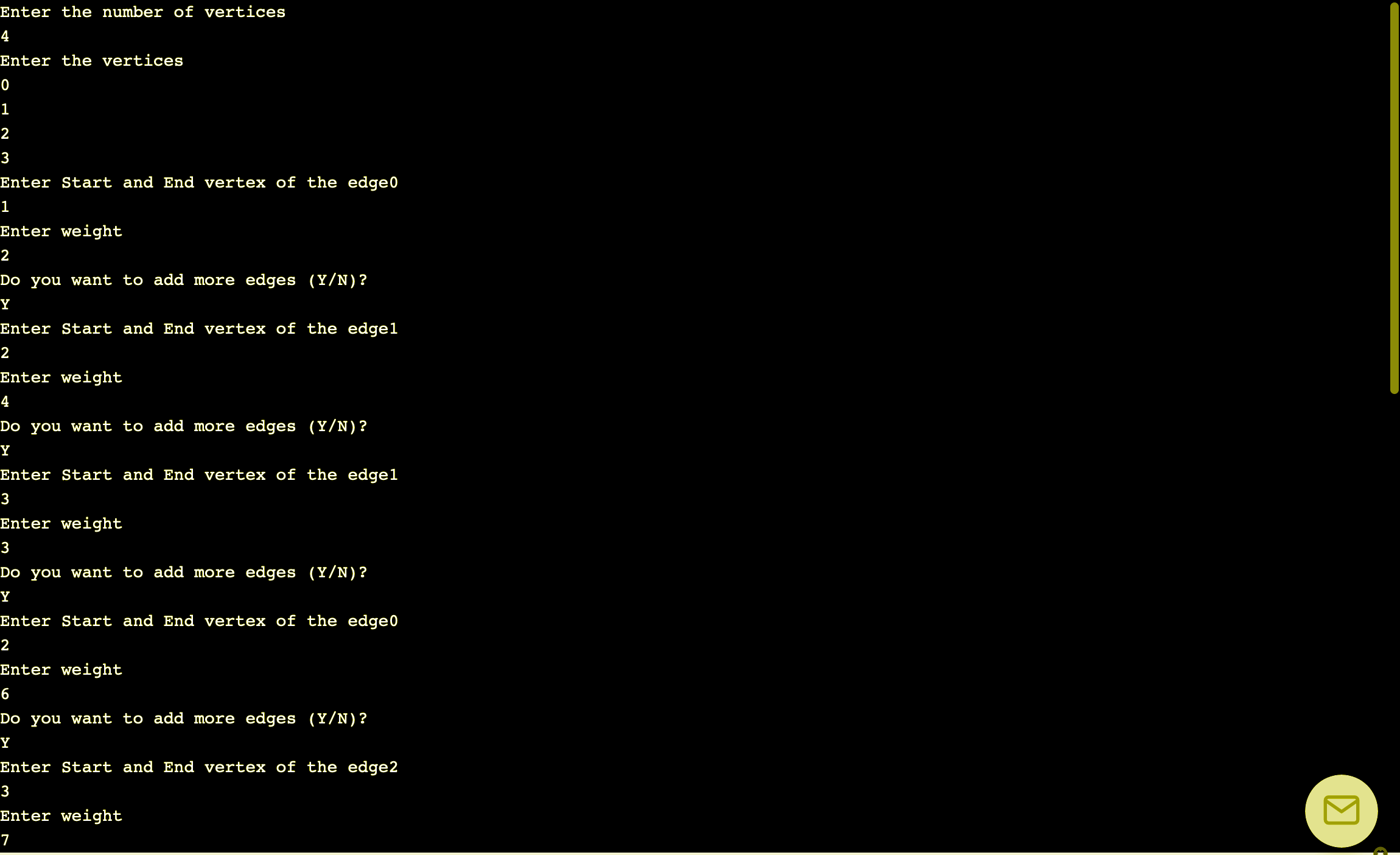
## Algorithm

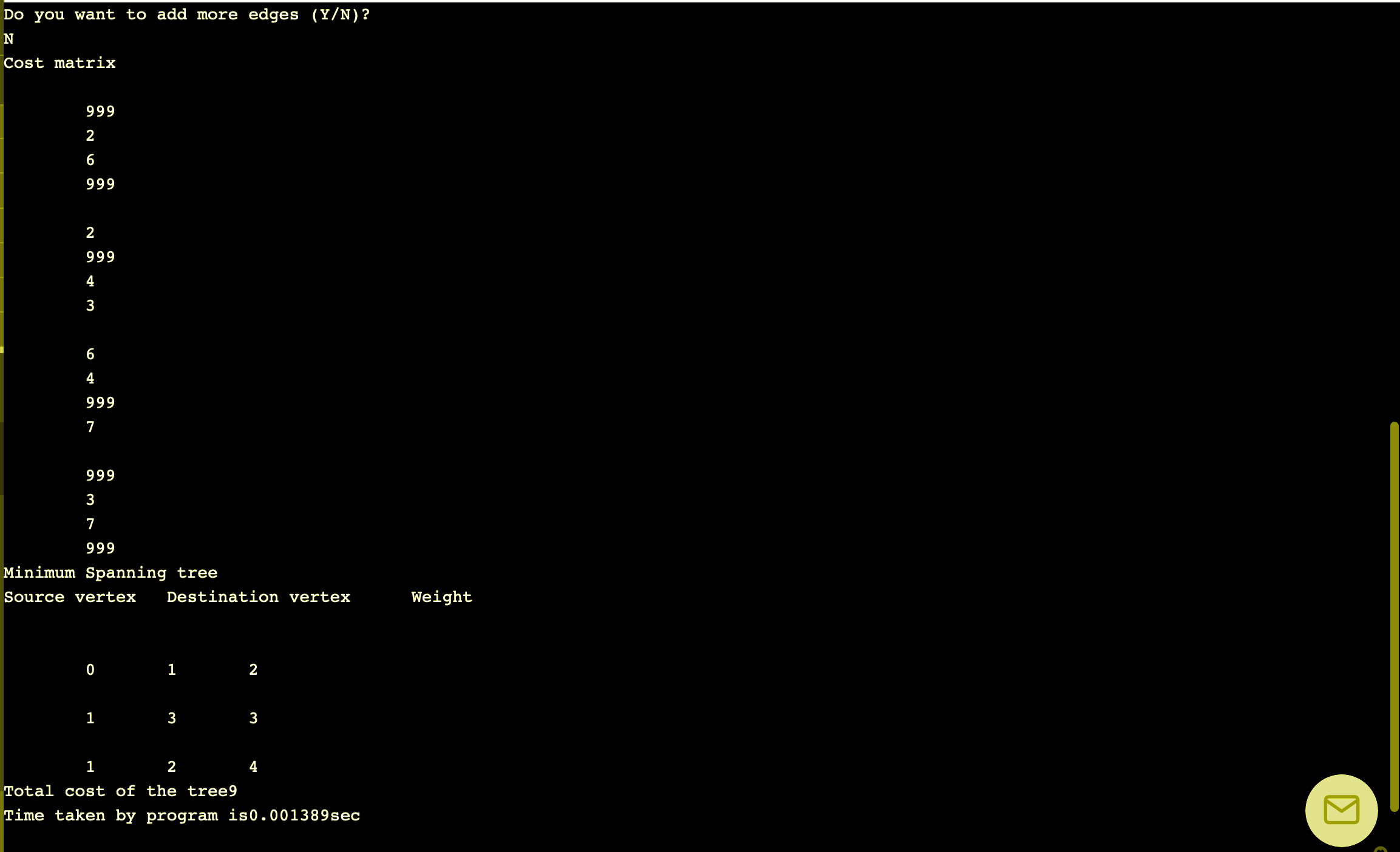
* Step 1: Create a forest in such a way that each graph is a separate tree.
* Step 2: Create a priority queue Q that contains all the edges of the graph.
* Step 3: Repeat Steps 4 and 5 while Q is NOT EMPTY
* Step 4: Remove an edge from Q
* Step 5: IF the edge obtained in Step 4 connects two different 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**

****

****