AM.SC.P2CSC19034

SRUTHY P R

**COMPITATIVE PROGRAMMING**

DIJKSTRA’S ALGORITHM

#include<iostream>

using namespace std;

int N;

int graph[10][10];

int dist[10];

bool visited[10];

int parent[10];

void createGraph()

{

int i,j,max,u,v,w;

cout<<"Enter the number of nodes : ";

cin>>N;

for(i=0;i<=N;i++)

for(j=0;j<=N;j++)

graph[i][j]=0;

max=N\*(N+1);

for(i=0;i<max;i++)

{

cout<<"Enter Edge and Weight : ";

cin>>u>>v>>w;

if(u==-1) break;

else

{

graph[u][v]=w;

graph[v][u]=w;

}

}

}

int minDistance()

{

int min = 10000, minDist;

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

if (visited[v] == false && dist[v] <= min)

{

min = dist[v];

minDist = v;

}

return minDist;

}

void printPath(int j)

{

if (parent[j]==-1)

return;

printPath(parent[j]);

cout<<j<<" ";

}

void dijkstra()

{

int src;

cout<<"Enter the Source Node : ";

cin>>src;

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

{

parent[0] = -1;

dist[i] = 10000;

visited[i] = false;

}

dist[src] = 0;

for (int count = 0; count < N-1; count++)

{

int u = minDistance();

visited[u] = true;

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

if (!visited[v] && graph[u][v] &&

dist[u] + graph[u][v] < dist[v])

{

parent[v] = u;

dist[v] = dist[u] + graph[u][v];

}

}

cout<<"Src->Dest\tDistance\tPath"<<endl;

for (int i = 1; i < N; i++)

{

cout<<src<<"->"<<i<<"\t\t"<<dist[i]<<"\t\t"<<src<<" ";

printPath(i);

cout<<endl;

}

}

int main()

{

createGraph();

dijkstra();

return 0;

}

**OUTPUT:**

