**Assignment-7**

**Problem Statement**:

A customer wants to travel from source A to destination B, he books a cab from source A to reach destination B calculate a shortest path by avoiding real time traffic to reach destination B.

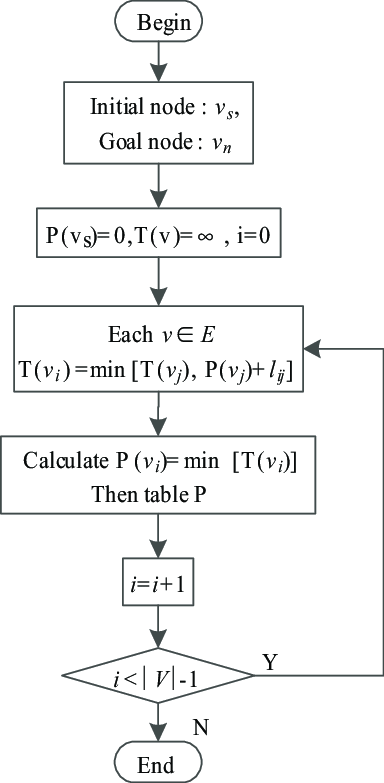
**Aim**: To solve the given problem using Dijkstra’s Algorithm.

**Objective**: To understand the problem statement and use suitable algorithm to solve the problem.To design and implement solution using Dijkstra’s algorithm.

**Theory**: Dijkstra's algorithm (or Dijkstra's Shortest Path First algorithm, SPF algorithm) is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

**Algorithm:**

1. Set the distance to the source to 0 and the distance to the remaining vertices to infinity.
2. Set the **current** vertex to the source.
3. Flag the **current** vertex as visited.
4. For all vertices adjacent to the **current** vertex, set the distance from the source to the **adjacent** vertex equal to the minimum of its present distance and the **sum** of the **weight of the edge** from the current vertex to the adjacent vertex and the distance from the source to the **current** vertex.
5. From the set of **unvisited vertices**, arbitrarily set one as the new **current** vertex, provided that there exists an edge to it such that it is the minimum of all edges from a vertex in the set of **visited vertices** to a vertex in the set of **unvisited vertices**. To reiterate: The new current vertex must be unvisited and have a minimum weight edges from a visited vertex to it. This can be done trivially by looping through all visited vertices and all adjacent unvisited vertices to those visited vertices, keeping the vertex with the minimum weight edge connecting it.
6. Repeat steps 3-5 until all vertices are flagged as visited.

****

**Code:**

**#**include<iostream>

#include<vector>

#include<stdlib.h>

using namespace std;

int min(int dis[],int n,int vis[])

{

int m,i,j;

m=32767;

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

{

if(dis[i]<=m && vis[i]==0)

{

m=dis[i];

j=i;

}

}

return j;

}

int main()

{

vector <int>v;

vector<int>::iterator it;

int n,i,j,u;

char ch;

cout<<"Enter the number of vertices\n";

cin>>n;

int arr[n][n],dest,source=0;

cout<<"Enter the destination\n";

cin>>dest;

do

{

cout<<"Source:"<<source<<"\n";

v.clear();

cout<<"Enter the adjecancy matrix w.r.t time\n";

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

{

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

{

cin>>arr[i][j];

}

}

cout<<"Entered matrix is:\n";

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

{

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

{

cout<<""<<arr[i][j]<<"\t";

}

cout<<"\n";

}

int count=1;

int vis[n],dis[n],parent[n];

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

{

vis[i]=0;

dis[i]=32767;

parent[i]=0;

}

dis[source]=0;

parent[source]=-1;

cout<<"Parent of source:"<<parent[source]<<"\n";

cout<<"\n";

while(count<=n)

{

u=min(dis,n,vis);

vis[u]=1;

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

{

if(arr[u][j]!=0 && dis[j]> dis[u]+arr[u][j])

{

dis[j]=dis[u]+arr[u][j];

parent[j]=u;

}

}

/\*for(i=0;i<n;i++)

{

cout<<""<<dis[i]<<"\t";

}

cout<<"\n";\*/

//cout<<"u:"<<u<<"\n";

if(u==dest)

{

while(parent[u]!=-1)

{

v.push\_back(u);

u=parent[u];

}

v.push\_back(source);

it=v.end();

it--;

while(it!=(v.begin()--))

{

cout<<""<<\*it<<" -\t";

it--;

}

cout<<""<<\*it<<"\n";

}

count++;

}

//cout<<"After using Dijkstra's algorithm:\n";

if(u==dest)

{

cout<<""<<u<<" : "<<dis[u]<<"\n";

}

it=v.end();

it--;

while(it!=(v.begin()-1))

{

cout<<"Has the traffic conditions changed?\n";

cin>>ch;

if(ch=='y' || ch=='Y')

{

/\*if(source==0)

{

it=v.end();

it--;

}\*/

source=\*it;

//cout<<"Source:"<<source<<"\n";

break;

}

else

{

if(\*it==dest)

{

cout<<"You have reached your destintion\n";

exit(1);

}

it--;

cout<<""<<\*it<<"\n";

continue;

}

}

}while(1);

return 0;

}

/\*

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 14 0 0 0

0 0 0 9 0 10 0 0 0

0 0 4 14 10 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

Changed traffic conditions:

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 1 0 0 0

0 0 0 9 0 11 0 0 0

0 0 4 1 11 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

\*/

/\*

OUTPUT:

Enter the number of vertices

9

Enter the destination

4

Source:0

Enter the adjecancy matrix w.r.t time

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 14 0 0 0

0 0 0 9 0 10 0 0 0

0 0 4 14 10 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

Entered matrix is:

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 14 0 0 0

0 0 0 9 0 10 0 0 0

0 0 4 14 10 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

Parent of source:-1

0 - 7 - 6 - 5 - 4

Has the traffic conditions changed?

n

7

Has the traffic conditions changed?

n

6

Has the traffic conditions changed?

n

5

Has the traffic conditions changed?

Y

Source:5

Enter the adjecancy matrix w.r.t time

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 1 0 0 0

0 0 0 9 0 11 0 0 0

0 0 4 1 11 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

Entered matrix is:

0 4 0 0 0 0 0 8 0

4 0 8 0 0 0 0 11 0

0 8 0 7 0 4 0 0 2

0 0 7 0 9 1 0 0 0

0 0 0 9 0 11 0 0 0

0 0 4 1 11 0 2 0 0

0 0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0 7

0 0 2 0 0 0 6 7 0

Parent of source:-1

5 - 3 - 4

Has the traffic conditions changed?

n

3

Has the traffic conditions changed?

4

Has the traffic conditions changed?

n

You have reached your destintion

\*/

**Conclusion:** In this assignment we learnt the implementation of dijkatra’s algorithm.