**Date-**

**Assignment No. :**

**Problem Statement:**

Program in C to find the shortest path between the nodes of a given graph by Dijkstra’s algorithm.

**Theory:**

Dijkstra’s algorithm is very similar to [Prim’s algorithm for minimum spanning tree](https://www.geeksforgeeks.org/archives/27455). Like Prim’s MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has minimum distance from source.

For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities.

Dijkstra's algorithm initially marks the distance (from the starting point) to every other intersection on the map with infinity. This is done not to imply there is an infinite distance, but to note that those intersections have not yet been visited; some variants of this method simply leave the intersections' distances unlabeled. Now, at each iteration, select the current intersection. For the first iteration, the current intersection will be the starting point, and the distance to it (the intersection's label) will be zero. For subsequent iterations (after the first), the current intersection will be a closest unvisited intersection to the starting point (this will be easy to find).

From the current intersection, update the distance to every unvisited intersection that is directly connected to it. This is done by determining the sum of the distance between an unvisited intersection and the value of the current intersection, and [relabeling](https://en.wikipedia.org/wiki/Graph_labeling) the unvisited intersection with this value (the sum), if it is less than its current value. In effect, the intersection is relabeled if the path to it through the current intersection is shorter than the previously known paths.

**Algorithm:**

**Input specification:**

1. I : The incidence matrix of dimension (n x n) of the given graph
2. vs : The source vertex to start the search from

**Output specification:**

1. A two dimensional array I[1..n][1..n] whose starting index is 1 and ending index is n, size of the array being (n x n).
2. A stack to store the intermediate vertices, say S.

**Steps:**

Algorithm for method main():

1. Print "Enter the no. of vertices: "
2. Input n
3. If(n < 2 || n>10) Then
4. Print "Number of vertices must be >= 2 and <=10"
5. return 1

[End of If structure]

1. Print "Enter the adjacency matrix:"
2. Repeat through step 8 to step 12 For i=0 to n-1
3. Repeat through step 8 to step 11 For j=0 to n-1
4. Print "Enter the element of row"(i+1)" and column "(j+1)
5. Input G[i][j])
6. Set j=j+1

[End of inner For loop]

1. Set i=i+1

[End of outer For loop]

1. Print "The entered matrix is: "
2. Print "Enter the starting node: "
3. Input u
4. If(u < 1 || u > n) Then
5. Print "Bad starting vertex : " u " Should be between 1 and " n
6. return 1

[End of If structure]

1. dijkstra(G,n,u-1) //Calling of the function dijkstra

[End of method main()]

Algorithm for method dijkstra(G[MAX][MAX], n, startnode):

1. Set k=0
2. Repeat through step 3 to step 7 For i=0 to n-1
3. Repeat through step 4 to step 7 For j=0 to n-1
4. If(G[i][j]==0) Then
5. cost[i][j]=INFINITY //INFINITY is a constant with an extreme value
6. Else
7. cost[i][j]=G[i][j]

[End of If-Else structure]

[End of inner For loop]

[End of outer For loop]

1. Repeat through step 9 to step 12 For i=0 to n-1
2. Set distance[i]=cost[startnode][i]
3. Set pred[i]=startnode
4. Set visited[i]=0
5. Set i=i+1

[End of For loop]

1. Set distance[startnode]=0
2. Set visited[startnode]=1
3. Set count=1
4. Repeat through step 16 to step 29 While(count < n-1)
5. Set mindistance=INFINITY
6. Repeat through step 18 to step 22 For i=0 to n-1
7. If(distance[i] < mindistance&&!visited[i]) Then
8. Set mindistance=distance[i]
9. Set nextnode=i

[End of If structure]

1. Set i=i+1

[End of For loop]

1. Set visited[nextnode]=1;
2. Repeat through step 25 to step 12 For i=0 to n-1
3. If(!visited[i]) Then
4. If(mindistance+cost[nextnode][i] < distance[i]) Then
5. Set distance[i]=mindistance+cost[nextnode][i]
6. Set pred[i]=nextnode

[End of inner If structure]

[End of outer If structure]

1. Set i=i+1

[End of For loop]

1. Count=count+1

[End of While loop]

1. Repeat through step 32 to step 45 For i=0 to n-1
2. If(i!=startnode) Then
3. Print "Distance of "i" = "distance[i])
4. Set j=i
5. Print "Path = "

[Starting of Do-While loop]

1. Set j=pred[j]
2. a[k]=j
3. k=k+1
4. While(j!=startnode)
5. s=k-1
6. Repeat through step 42 to step 43 For k=s to 0
7. Print a[k] "->"
8. Set k=k+1

[End of For loop]

1. Print i
2. Set k=0

[End of If structure]

1. Set i=i+1

[End of For loop]

[End of method dijkstra()]

**Source Code:**

#include<stdio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

int main()

{

int G[MAX][MAX], i, j, n, u;

printf("Enter the no. of vertices: ");

scanf("%d", &n);

if(n < 2 || n>10){//Checking for vertex no

printf("\nNumber of vertices must be >= 2 and <=10");

return 1;

}

printf("\nEnter the adjacency matrix:\n");

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

{

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

{

printf("Enter the element of row %d and column %d: ",(i+1),(j+1));

scanf("%d", &G[i][j]);

}

}

printf("The entered matrix is:\n");//Traversing the entered vertex

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

{

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

{

printf("%d ", G[i][j]);

}

printf("\n");

}

printf("\nEnter the starting node: ");

scanf("%d", &u);

if(u < 1 || u > n){

printf("\nBad starting vertex : %d\nShould be between 1 and %d!", u, n);

return 1;

}

dijkstra(G,n,u-1);//Calling the function

return 0;

}

void dijkstra(int G[MAX][MAX], int n, int startnode) //Main operation

{

int cost[MAX][MAX], distance[MAX], pred[MAX],k=0,a[MAX],s;

int visited[MAX], count, mindistance, nextnode, i,j;

for(i=0;i < n;i++)//Setting the matrix

{

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

{

if(G[i][j]==0)

cost[i][j]=INFINITY;

else

cost[i][j]=G[i][j];

}

}

for(i=0;i< n;i++)//Changing the matrix

{

distance[i]=cost[startnode][i];

pred[i]=startnode;

visited[i]=0;

}

distance[startnode]=0;

visited[startnode]=1;

count=1;

while(count < n-1){

mindistance=INFINITY;

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

{

if(distance[i] < mindistance&&!visited[i])

{

mindistance=distance[i];

nextnode=i;

}

}

visited[nextnode]=1;

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

{

if(!visited[i])

{

if(mindistance+cost[nextnode][i] < distance[i])

{

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode;

}

}

}

count++;

}

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

{

if(i!=startnode)//Finding the path

{

printf("\nDistance of %d = %d", i, distance[i]);

j=i;

printf("\nPath = ");

do

{

j=pred[j];

a[k++]=j;//Auxiliary storage to produce right sequence

}while(j!=startnode);

s=k-1;

for(k=s;k>=0;k--)

{

printf("%d->",a[k]);//Producing right sequence from the storage

}

printf("%d", i);

k=0;

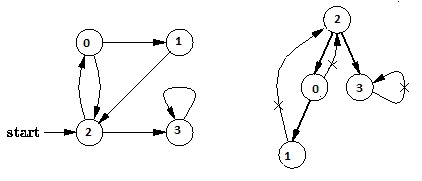
}

}

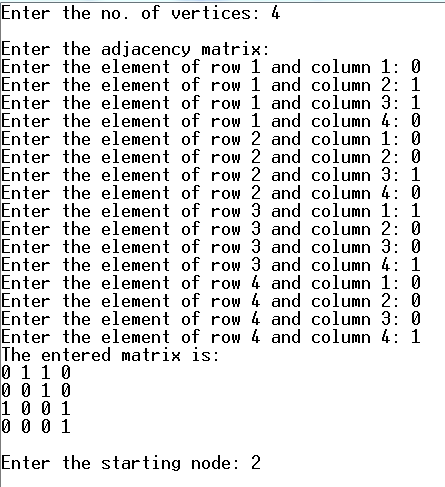
}

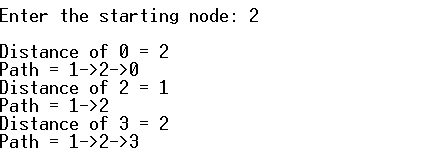
**Input & Output:**

Input graph:



Output of program:





**Discussion:**

Many more problems than you might at first think can be cast as shortest path problems, making Dijkstra’s algorithm a powerful and general tool. For example:

1. Dijkstra’s algorithm is applied to automatically find directions between physical locations, such as driving directions on websites like Mapquest or Google Maps.
2. In a networking or telecommunication applications, Dijkstra’s algorithm has been used for solving the min-delay path problem (which is the shortest path problem). For example in data network routing, the goal is to find the path for data packets to go through a switching network with minimal delay.
3. It is also used for solving a variety of shortest path problems arising in plant and facility layout, robotics, transportation, and VLSI∗ design.
4. Dijkstra’s algorithm solves such a problem:

• It finds the shortest path from a given node s to all other nodes in the

network.

• Node s is called a starting node or an initial node.

1. It is a Greedy algorithm.
2. It works by maintaining a set *S* of ``special'' vertices whose shortest distance from the source is already known. At each step, a ``non-special'' vertex is absorbed into *S*.
3. The absorption of an element of *V* - *S* into *S* is done by a greedy strategy.