Aim-Store a graph using adjacency matrix or list representation or adjacency matrix representation and implement Dijkstra’s algorithm to find shortest path.

Theory- Dijkstra's Algorithm allows you to calculate the shortest path between one node (you pick which one) and every other node in the graph.  The graph can either be directed or undirected. One stipulation to using the algorithm is that the graph needs to have a nonnegative weight on every edge.

1. Mark your selected initial node with a current distance of 0 and the rest with infinity.
2. Set the non-visited node with the smallest current distance as the current node C.
3. For each neighbour N of your current node C: add the current distance of C with the weight of the edge connecting C-N. If it's smaller than the current distance of N, set it as the new current distance of N.
4. Mark the current node as visited.
5. If there are non-visited nodes, go to step 2.

This is done by initializing three values:

* Dist*-* an array of distances from the source node s*s* to each node in the graph, initialized the following way: dist*dist*(s*s*) = 0; and for all other nodes v*v*, dist*dist*(v*v*) =\infty∞. This is done at the beginning because as the algorithm proceeds, the dist*dist* from the source to each node v*v* in the graph will be recalculated and finalized when the shortest distance to v*v* is found
* Q*Q*, a [queue](https://brilliant.org/wiki/queues-basic/) of all nodes in the graph. At the end of the algorithm's progress, Q*Q* will be empty.
* S*S*, an empty [set](https://brilliant.org/wiki/sets/), to indicate which nodes the algorithm has visited. At the end of the algorithm's run, S*S* will contain all the nodes of the graph.

Source Code:

#include<stdio.h>

#include<conio.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 no. of vertices:");

scanf("%d",&n);

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

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

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

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

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

scanf("%d",&u);

dijkstra(G,n,u);

return 0;

}

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

{

int cost[MAX][MAX],distance[MAX],pred[MAX];

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

//pred[] stores the predecessor of each node

//count gives the number of nodes seen so far

//create the cost matrix

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

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

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

cost[i][j]=INFINITY;

else

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

//initialize pred[],distance[] and visited[]

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

{

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;

//nextnode gives the node at minimum distance

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

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

{

mindistance=distance[i];

nextnode=i;

}

//check if a better path exists through nextnode

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++;

}

//print the path and distance of each node

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

if(i!=startnode)

{

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

printf("\nPath=%d",i);

j=i;

do

{

j=pred[j];

printf("<-%d",j);

}while(j!=startnode);

}

}

Conclusion:

In this assignment we stored a graph using adjacency matrix or list representation or adjacency matrix representation and implemented Dijkstra’s algorithm to find shortest path.