Assignment 16

AIM:-

To store a graph using adjacency matrix/list representation or adjacency matrix representation and implement Dijkstra's algorithm to find shortest path.

OBJECTIVE:-

Using adjacency matrix representation stores a graph to find shortest path using Dijkstra's algorithm.

THEORY:-

Dijkstra’s algorithm is very similar to Prim’s algorithm for minimum s panning tree. 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 a minimum distance from the source.

ALGORITHM:-

1. Make the source node a “Permanent” Node. The source node is the first working node.

2. Examine each non-permanent node adjacent to the working node. If it is not labelled, label it with the distance from the source and the name of the working node. If it is labelled, see if the cost computed using the working node is cheaper than the cost in the label; if so re-label the node as above.

3. Find the non-permanent node with the smallest label, and make it permanent. If this is the destination, then finished. Otherwise, this node is the next working node, continue from step 2.

SOURCE CODE:

#include <limits.h>

#include <iostream>

#define V 9

using namespace std;

int minDistance(int dist[], bool sptSet[])

{

int min = INT\_MAX, min\_index;

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

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

min = dist[v], min\_index = v;

return min\_index;

}

int printSolution(int dist[])

{

cout<<"Vertex \t\t Distance from Source\n";

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

cout<<i<<"\t\t"<<dist[i]<<endl;

}

void dijkstra(int graph[V][V], int src)

{

int dist[V];

bool sptSet[V];

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

dist[i] = INT\_MAX, sptSet[i] = false;

dist[src] = 0;

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

int u = minDistance(dist, sptSet);

sptSet[u] = true;

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

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

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

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

}

printSolution(dist);

}

int main()

{

int graph[V][V] = { { 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 } };

dijkstra(graph, 0);

return 0;

}

OUTPUT:-

Vertex Distance from Source

0 0

1 4

2 12

3 19

4 21

5 11

6 9

7 8

8 14

--------------------------------

Process exited after 7.286 seconds with return value 0

Press any key to continue . . .

CONCLUSION:-

Dijkstra’s algorithm has some similarity to both breadth first search (BFS) and Prim’s Algorithm for computing minimum spanning trees.

BFS: Similar in that the set S corresponds to the set of black vertices in the BFS just as the vertices in S have their final shortest-path weights, so do black vertices in a BFS for their correct vertices.

Prim’s: Similar in that both algorithms use a minpriority queue to find the “lightest” vertex outside a given set, add this vertex into the set, and adjust the weights of the remaining vertices outside the set according

Time Complexity:-Dijkstra’s algorithm’s Time complexity is *O*(*n2*) run time.