Experiment-8

**Date**-May 27, 2021.

**AIM-** Find shortest path between two nodes in a computer network using Dijkstra’s shortest path algorithm.

**Dijkstra Shortest path algorithm**

This is a single-source shortest path algorithm and aims to find solution to the given problem statement. This algorithm works for both directed and undirected graphs. It works only for connected graphs. The graph should not contain negative edge weights. The algorithm predominantly follows Greedy approach for finding locally optimal solution. But, it also uses Dynamic Programming approach for building globally optimal solution, since the previous solutions are stored and further added to get final distances from the source vertex.

The main logic of this algorithm is based on the following formula- **dist[r]=min(dist[r], dist[q]+cost[q][r])**

This formula states that distance vertex r, which is adjacent to vertex q, will be updated if and only if the value of dist[q]+cost[q][r] is less than dist[r]. Here-

* Dist is a 1-D array which, at every step, keeps track of the shortest distance from source vertex to all other vertices.
* Cost is a 2-D array, representing the cost adjacency matrix for the graph.
* This formula uses both Greedy and Dynamic approaches. The Greedy approach is used for finding the minimum distance value, whereas the Dynamic approach is used for combining the previous solutions (**dist[q]** is already calculated and is used to calculate **dist[r]**).

**Algorithm-**

**Step 1**; Set dist[s]=0, S=ϕ.

**Step 2**: For all nodes v except s, set dist[v]= ∞

**Step 3**: find q not in S such that dist[q] is minimum.

**Step 4**: add q to S.

**Step 5**: update dist[r] for all r adjacent to q such that r is not in S //vertex r should not be visited dist[r]=min(dist[r], dist[q]+cost[q][r]).

**Step 6**: Repeat Steps 3 to 5 until all the nodes are in S. Repeat till all the vertices have been visited.

**Step 7**: Print array dist having shortest path from the source vertex u to all other vertices.

**Step 8**: Exit.



**PROGRAM (JAVA)-**

import java.util.\*;

import java.lang.\*;

import java.io.\*;

public class Dijkstra {

static final int V = 9;

static int minDistance(int dist[], Boolean sptSet[]) {

int min = Integer.MAX\_VALUE, min\_index = -1;

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

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

min = dist[v];

min\_index = v;

}

return min\_index;

}

static void dijkstra(int graph[][], int src , int dest) {

int dist[] = new int[V];

Boolean sptSet[] = new Boolean[V];

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

dist[i] = Integer.MAX\_VALUE;

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] != 0 && dist[u] != Integer.MAX\_VALUE && dist[u] + graph[u][v] < dist[v])

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

}

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

if(i == dest)

System.out.println("\nDistance from " + src + " to " + i + " : " + dist[i]);

}

public static void main(String[] args) {

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

Scanner scn = new Scanner(System.in);

int src = 0, dest = 1;

System.out.print("Enter the source node : ");

src = scn.nextInt();

System.out.print("Enter the destination node : ");

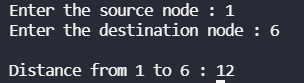
dest = scn.nextInt();

dijkstra(graph, src , dest);

}

}

**OUTPUT-**

****