Floyd-Warshall Algorithm

Floyd-Warshall algorithm is a dynamic programming formulation, to solve the all-pairs shortest path problem on directed graphs. It finds shortest path between all nodes in a graph. If finds only the lengths not the path. The algorithm considers the intermediate vertices of a simple path are any vertex present in that path other than the first and last vertex of that path.

Algorithm:

Input Format: Graph is directed and weighted. First two integers must be number of vertices and edges which must be followed by pairs of vertices which has an edge between them.

maxVertices represents maximum number of vertices that can be present in the graph.
vertices represent number of vertices and edges represent number of edges in the graph.
graph[i][j] represent the weight of edge joining i and j.
size[maxVertices] is initialed to{0}, represents the size of every vertex i.e. the number of edges corresponding to the vertex.
visited[maxVertices]={0} represents the vertex that have been visited.
distance[maxVertices][maxVertices] represents the weight of the edge between the two vertices or distance between two vertices.
Initialize the distance between two vertices using init() function.
init() function- It takes the distance matrix as an argument.

```
For iter=0 to maxVertices – 1
        For jter=0 to maxVertices – 1
                if(iter == jter)
                    distance[iter][jter] = 0 //Distance between two same vertices is 0
                else
                    distance[iter][jter] = INF//Distance between different vertices is INF
                jter + 1
        iter + 1
```

Where, INF is a very large integer value.
Initialize and input the graph.
Call FloydWarshall function.

- It takes the distance matrix (distance[maxVertices][maxVertices]) and number of vertices as argument (vertices).
- Initialize integer type from, to, via

```
For from=0 to vertices-1
        For to=0 to vertices-1
                For via=0 to vertices-1
                      distance[from][to] = min(distance[from][to],distance[from]
                      [via]+distance[via][to])
                via + 1
        to + 1
```

from + 1

This finds the minimum distance from from vertex to to vertex using the min function. It checks it there are intermediate vertices between the from and to vertex that form the shortest path between them

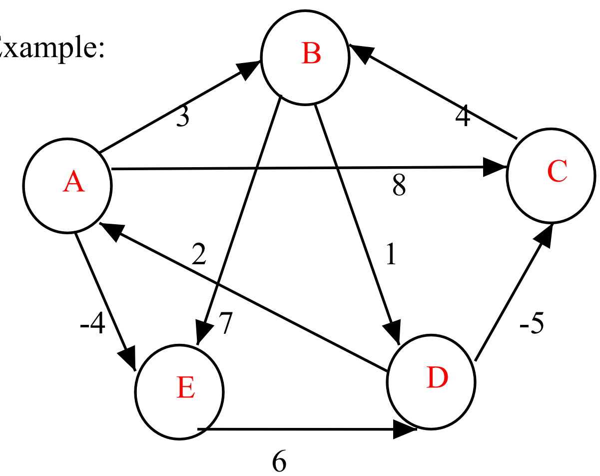
- min function returns the minimum of the two integers it takes as argument.

Output the distance between every two vertices.

Analysis:

The running time of Floyd-Warshall algorithm is $O(V^3)$, determined by the triply nested for loops.

Example:



Directed Graph with vertices A, B, C, D, E and weight of the edges connecting two vertices.

$D^{(0)}$: Initial distance matrix. It stores the distance between adjacent vertices. The distance is zero if the vertices are same and ∞ if they are not adjacent.

$$\begin{array}{cc} & \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} & \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \end{array}$$

$D^{(1)}$: Interspace A between any two nodes to find out a shorter path.

Interspacing A between D and B ‘s previous path changes the distance from ∞ to 5.
Interspacing A between D and E ‘s previous path changes the distance from ∞ to -2.

Resultant distance matrix

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}\begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \end{array}$$

$D^{(2)}$: Interspace B between any two nodes to find out a shorter path.

Interspacing B between A and D ‘s previous path changes the distance from ∞ to 4.
Interspacing B between C and D ‘s previous path changes the distance from ∞ to 5.
Interspacing B between C and E ‘s previous path changes the distance from ∞ to 11.

Resultant distance matrix

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}\begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \end{array}$$

$D^{(3)}$: Interspace C between any two nodes to find out a shorter path.

Interspacing C between D and B ‘s previouspath changes the distance from 5 to -2.

Resultant distance matrix

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \end{array}$$

$D^{(4)}$: Interspace D between any two nodes to find out a shorter path.

Interspacing D between A and C ‘s previous path changes the distance from 8 to -1.
Interspacing D between B and A ‘s previous path changes the distance from ∞ to 3.
Interspacing D between B and C ‘s previous path changes the distance from ∞ to -4.
Interspacing D between B and E ‘s previous path changes the distance from 7 to -1.
Interspacing D between C and A ‘s previous path changes the distance from ∞ to 7.
Interspacing D between C and E ‘s previous path changes the distance from 11 to 3.
Interspacing D between E and A ‘s previous path changes the distance from ∞ to 8.
Interspacing D between E and B ‘s previous path changes the distance from ∞ to 5.
Interspacing D between E and C ‘s previous path changes the distance from ∞ to 1.

Resultant distance matrix

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \end{array}$$

$D^{(5)}$: Interspace E between any two nodes to find out a shorter path.

Interspacing E between A and B ‘s previous path changes the distance from 3 to 1.
Interspacing E between A and C ‘s previous path changes the distance from -1 to -3.
Interspacing E between A and D ‘s previous path changes the distance from 4 to 2.

Resultant distance matrix

$$\begin{array}{c c} & \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} & \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \end{array}$$

Final distance matrix with shortest path between the two vertices

$$\begin{array}{c c} & \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} & \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \end{array}$$