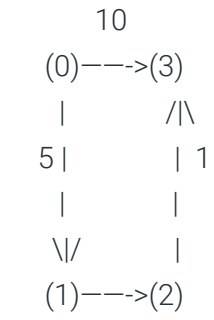
Floyd Warshall Algorithm

Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).

Input: graph[][] = { {0, 5, INF, 10},
{INF, 0, 3, INF},
{INF, INF, 0, 1},
{INF, INF, INF, 0} }
which represents the following graph

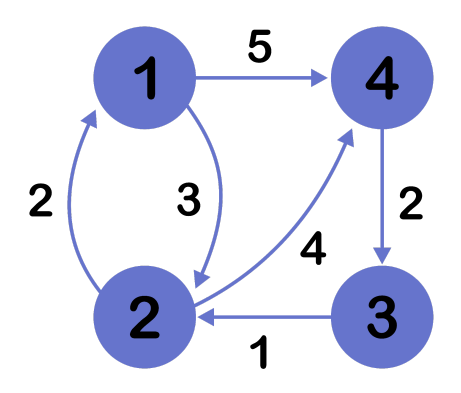


Output: Shortest distance matrix

0	5	8	9
INF	0	3	4
INF	INF	0	1
INF	INF	INF	0

How Floyd-Warshall Algorithm Works?

Let the graph be:



Follow the steps below to find the shortest path between all the pairs of vertices.

- Create a matrix A0 of dimension n*n where n is the number of vertices. The row and the column are indexed as i and j respectively. i and j are the vertices of the graph. Each cell A[i][j] is filled with the distance from the ith vertex to the jth vertex. If there is no path from ith vertex to jth vertex, the cell is left as infinity.

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}_4$$

Fill each cell with the distance between ith and jth vertex

- Now, create a matrix A1 using matrix A0. The elements in the first column and the first row are left as they are. The remaining cells are filled in the following way. Let k be the intermediate vertex in the shortest path from source to destination. In this step, k is the first vertex. A[i][j] is filled with (A[i][k] + A[k][j]) if (A[i][j] > A[i][k] + A[k][j]).

  That is, if the direct distance from the source to the destination is greater than the path through the vertex k, then the cell is filled with A[i][k] + A[k][j].

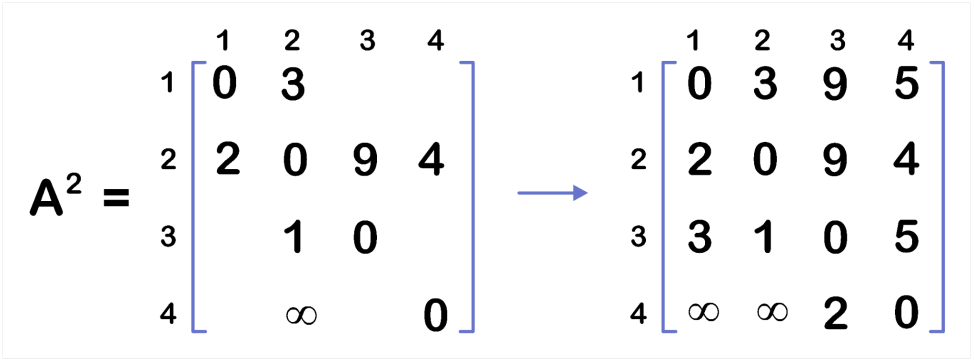
  In this step, k is vertex 1. We calculate the distance from source vertex to destination vertex through this vertex k.

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & & \\ \infty & & 0 & \\ \infty & & & 0 \end{bmatrix} \end{array} \rightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & 9 & 4 \\ \infty & 1 & 0 & 8 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

Calculate the distance from the source vertex to destination vertex through this vertex k

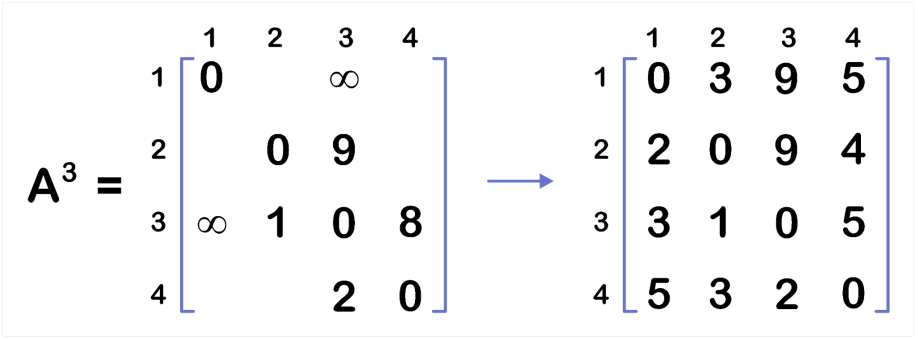
For example: For A1[2, 4], the direct distance from vertex 2 to 4 is 4 and the sum of the distance from vertex 2 to 4 through vertex (ie. from vertex 2 to 1 and from vertex 1 to 4) is 7. Since 4 < 7, A0[2, 4] is filled with 4.

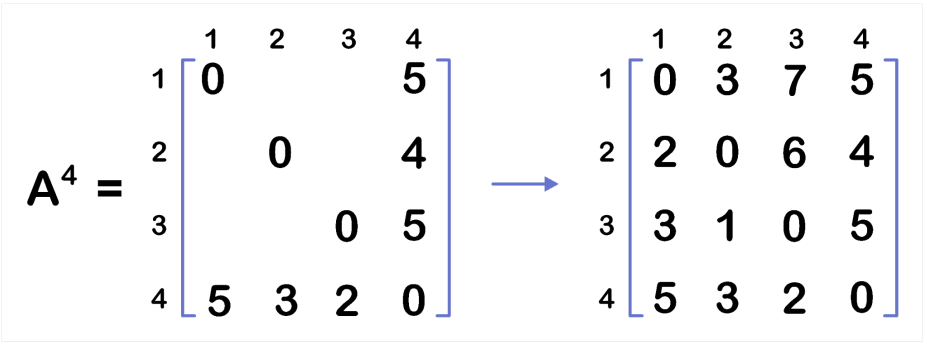
- Similarly, A2 is created using A1. The elements in the second column and the second row are left as they are. In this step, k is the second vertex (i.e. vertex 2). The remaining steps are the same as in step 2.



Calculate the distance from the source vertex to destination vertex through this vertex 2

- Similarly, A3 and A4 is also created





Calculate the distance from the source vertex to destination vertex through this vertex 3 first and then 4.

- A4 gives the shortest path between each pair of vertices.

Floyd Warshall Algorithm:

- Initialize the solution matrix same as the input graph matrix as a first step.
- Then update the solution matrix by considering all vertices as an intermediate vertex.
- The idea is to one by one pick all vertices and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.
- When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices.

- For every pair (i, j) of the source and destination vertices respectively, there are two possible cases.
    - k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.
    - k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j]

Code:

```
import java.io.*;
import java.lang.*;
import java.util.*;

class Solution {
    final static int INF = 99999, V = 4;

    void floydWarshall(int graph[][]){
        int dist[][] = new int[V][V];
        int i, j, k;

        for (i = 0; i < V; i++)
            for (j = 0; j < V; j++)
                dist[i][j] = graph[i][j];

        for (k = 0; k < V; k++) {
            for (i = 0; i < V; i++) {
                for (j = 0; j < V; j++) {
                    if (dist[i][k] + dist[k][j]
                        < dist[i][j])
                        dist[i][j]
                            = dist[i][k] + dist[k][j];
                }
            }
        }
        printSolution(dist);
    }

    void printSolution(int dist[][]){
        System.out.println(
            "The following matrix shows the shortest "
            + "distances between every pair of vertices");
```

```
                for (int i = 0; i < V; ++i) {
                        for (int j = 0; j < V; ++j) {
                                if (dist[i][j] == INF)
                                        System.out.print("INF ");
                                else
                                        System.out.print(dist[i][j] + " ");
                        }
                        System.out.println();
                }
        }

        public static void main(String[] args){
                int graph[][] = { { 0, 5, INF, 10 },
                                                { INF, 0, 3, INF },
                                                { INF, INF, 0, 1 },
                                                { INF, INF, INF, 0 } };
                Solution a = new Solution();

                a.floydWarshall(graph);
        }
}
```

Thanks for reading the article, hope it helps :)