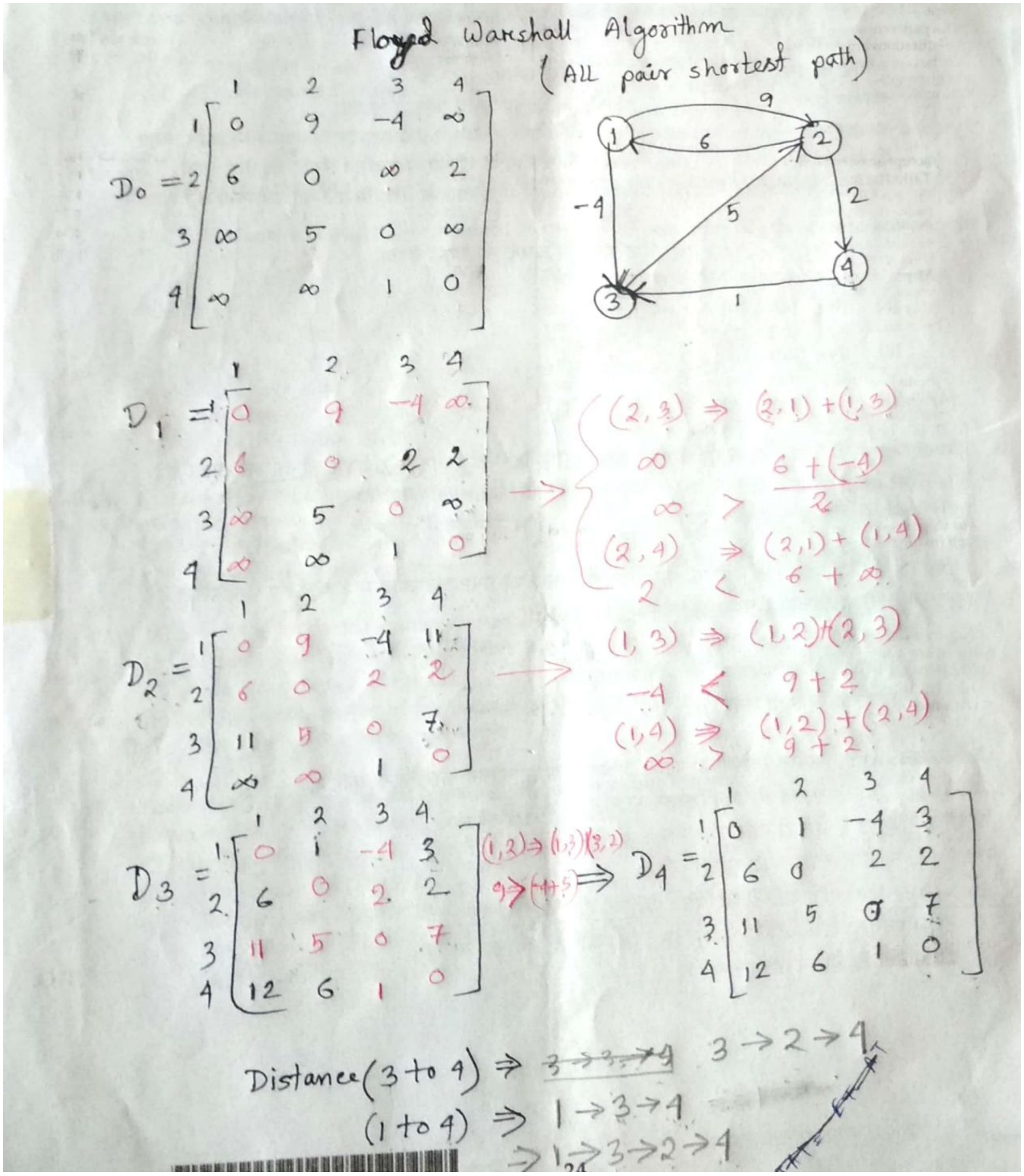
Floyd-Warshall Algorithm

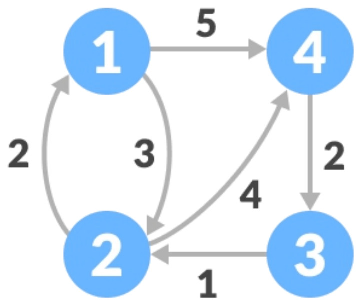


The **Floyd-Warshall Algorithm** is an efficient algorithm to find all-pairs shortest paths on a graph. That is, it is guaranteed to find the shortest path between every pair of vertices in a graph. The graph may have negative weight edges, but no negative weight cycles (where the sum of the edges in a cycle is negative, then the shortest path is undefined).

This algorithm can also be used to detect the presence of negative cycles—the graph has one if at the end of the algorithm, the distance from a vertex ‘v’ to itself is negative.

Floyd-Warhshall algorithm is also called as Floyd's algorithm, Roy-Floyd algorithm, Roy-Warshall algorithm, or WFI algorithm.

This algorithm follows the dynamic programming approach to find the shortest paths.



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

1. Create a matrix A^0 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.

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

Fill each cell with the distance between ith and jth vertex

2. Now, create a matrix A^1 using matrix A^0. 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.

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

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

For example: For $A^1[2, 4]$, 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 $4 < 7$, $A^0[2, 4]$ is filled with 4.

3. Similarly, A^2 is created using A^1. 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**.

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

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

4. Similarly, A^3 and A^4 is also created.

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

Calculate the distance from the source vertex to destination vertex through this vertex 3

$$A^4 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & & & 5 \\ & 0 & & 4 \\ & & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array} \rightarrow \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & 7 & 5 \\ 2 & 0 & 6 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

Calculate the distance from the source vertex to destination vertex through this vertex 4

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

Floyd-Warshall Algorithm

The pseudocode below assumes an input graph of N vertices.

```
for i = 1 to N
  for j = 1 to N
    if there is an edge from i to j
      dist[0][i][j] = the length of the edge from i to j
    else
      dist[0][i][j] = INFINITY

for k = 1 to N
  for i = 1 to N
    for j = 1 to N
      dist[k][i][j] = min(dist[k-1][i][j], dist[k-1][i][k] + dist[k-1][k][j])
```

This will give the shortest distances between any two nodes, from which shortest paths may be constructed.

Floyd Warshall Algorithm Complexity

Time Complexity:

There are three loop.Each loop has constant complexities.So, the time complexity of Floyd -Warshall algorithm is $O(n^3)$.

Space Complexity:

The space complexity of the Floyd-Warshall algorithm is $O(n^2)$.

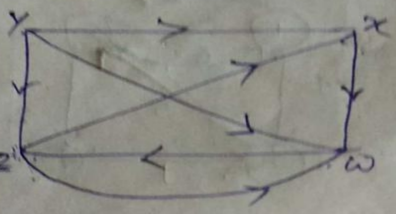
Floyd Warshall Algorithm Applications

- To find the shortest path is a directed graph
- To find the transitive closure of directed graphs
- To find the Inversion of real matrices
- For testing whether an undirected graph is bipartite

Warshall's Algorithm:

Warshall's Algorithm to find path matrix.

Path matrix



$$A = P = \begin{array}{c|cccc} & x & y & z & w \\ \hline x & 0 & 0 & 0 & 1 \\ y & 1 & 0 & 1 & 1 \\ z & 1 & 0 & 0 & 1 \\ w & 0 & 0 & 1 & 0 \end{array}$$

OR

$x \xrightarrow{0} x$

$0 \to x$ 0

And

$$P_x = \begin{array}{c|cccc} & x & y & z & w \\ \hline x & 0 & 0 & 0 & 1 \\ y & 1 & 0 & 1 & 1 \\ z & 1 & 0 & 0 & 1 \\ w & 0 & 0 & 1 & 0 \end{array}$$

OR

$2 \xrightarrow{1} w$

1 And

OR

$w \xrightarrow{1} z$

0 0

AND

0

$$P_y = \begin{array}{c|cccc} & x & y & z & w \\ \hline x & 0 & 0 & 0 & 1 \\ y & 1 & 0 & 1 & 1 \\ z & 1 & 0 & 0 & 1 \\ w & 0 & 0 & 1 & 0 \end{array}$$

$y \xrightarrow{1} x$

$0 \to y$ 1

$$P_z = \begin{array}{c|cccc} & x & y & z & w \\ \hline x & 0 & 0 & 0 & 1 \\ y & 1 & 0 & 1 & 1 \\ z & 1 & 0 & 0 & 1 \\ w & 1 & 0 & 1 & 1 \end{array}$$

$$P_w = \begin{array}{c|cccc} & x & y & z & w \\ \hline x & 1 & 0 & 1 & 1 \\ y & 1 & 0 & 1 & 1 \\ z & 1 & 0 & 1 & 1 \\ w & 1 & 0 & 1 & 1 \end{array}$$

OR

$z \xrightarrow{0} z$

$1 \to w$

AND

A directed graph G is maintained in memory by its adjacency matrix A. The algo finds the (Boolean) path matrix P of the graph G

Repeat for I, J = 1, 2, ... n

if (A[i, j] = 0 then P[i, j] = 0

else P[i, j] = 1

Repeat for K = 1, 2 ... n

Repeat for I = 1, 2, ... n

Repeat for j = 1, 2, ... n

P[i, j] = P[i, j] or (P[i, K] and P[K, j])