**Tutorial-3 :**

**Floyd-warshall algorithm**

Floyd-warshall algorithm is used for solving the All Pairs Shortest Path problem. The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph. It follow dynamic programming approach.

**Simple Example:**

**Input:**

graph[][] = { {0, 5, INF, 10},

{INF, 0, 3, INF},

{INF, INF, 0, 1},

{INF, INF, INF, 0} }

which represents the following graph

10

(0)------->(3)

| /|\

5| |

| | 1

\|/ |

(1)------->(2)

3

Note that the value of graph[i][j] is 0 if i is equal to j

And graph[i][j] is INF (infinite) if there is no edge from vertex i to j.

**Python Code:**

import sys

INF = sys.maxsize

def floydWarshall(graph):

    # number of vertices in the graph

    n = len(graph)

    # dist will be the output matrix that will have the shortest distances between every pair of vertex.

    dist = [[] for i in range(n)]

    # Initialize the dist matrix as same as the input graph matrix.

    for i in range(n):

        for j in range(n):

            dist[i].append(graph[i][j])

    # Taking all vertices one by one and setting them as intermediate vertices

    for k in range(n):

        # Pick all vertices as source one by one.

        for i in range(n):

            # Pick all vertices as the destination for the above choosen source vertex.

            for j in range(n):

                # Update the value of dist[i][j] if k provides a shortest path from i to j

                dist[i][j] = min(dist[i][j],dist[i][k]+dist[k][j])

    # Shortest distance for every pair of vertex.

    print('Shortest Distance between every pair of vertex:-')

    for i in range(n):

        for j in range(n):

            if dist[i][j]==INF:

                print ("%7s" % ("INF"),end=' ')

            else:

                print ("%7s" % (dist[i][j]),end=' ')

        print()

graph = [[0,5,INF,10],[INF,0,3,INF],[INF,INF,0,1],[INF,INF,INF,0]]

floydWarshall(graph)

**Output:**

Shortest distance matrix

0 5 8 9

INF 0 3 4

INF INF 0 1

INF INF INF 0

**Example 2:**

This formula is the heart of the Floyd–Warshall algorithm. The algorithm works by first computing {\displaystyle \mathrm {shortestPath} (i,j,k)} for all {\displaystyle (i,j)}pairs for{\displaystyle k=1}, then {\displaystyle k=2}, and so on. This process continues until, {\displaystyle k=N}and we have found the shortest path for all {\displaystyle (i,j)}pairs using any intermediate vertices.

Pseudo code for this basic version follows:

Algorithm:

let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)

for each edge (*u*, *v*) do

dist[*u*][*v*] ← w(*u*, *v*) *// The weight of the edge (*u*,* v*)*

for each vertex *v* do

dist[*v*][*v*] ← 0

for *k* from 1 to |V|

for *i* from 1 to |V|

for *j* from 1 to |V|

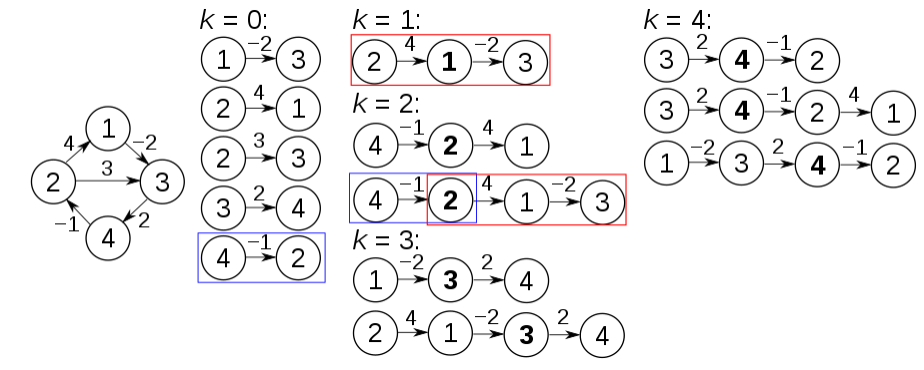
if dist[*i*][*j*] > dist[*i*][*k*] + dist[*k*][*j*]

dist[*i*][*j*] ← dist[*i*][*k*] + dist[*k*][*j*]

end if

Example:

The algorithm above is executed on the graph on the below:



**Explanation:**

Prior to the first recursion of the outer loop, labeled *k* = 0 above, the only known paths correspond to the single edges in the graph.

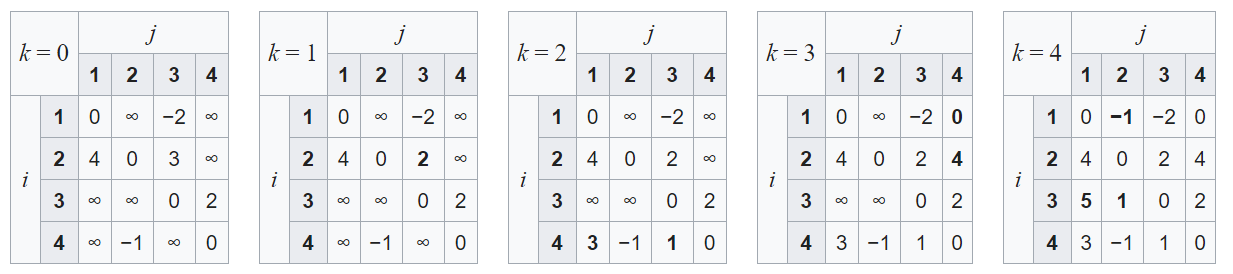
At *k* = 1, paths that go through the vertex 1 are found: in particular, the path [2,1,3] is found, replacing the path [2,3] which has fewer edges but is longer (in terms of weight).

At *k* = 2, paths going through the vertices {1,2} are found.

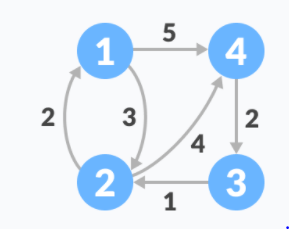
The red and blue boxes show how the path [4,2,1,3] is assembled from the two known paths [4,2] and [2,1,3] encountered in previous iterations, with 2 in the intersection. The path [4,2,3] is not considered, because [2,1,3] is the shortest path encountered so far from 2 to 3.

At *k* = 3, paths going through the vertices {1,2,3} are found. Finally, at *k* = 4, all shortest paths are found.

The distance matrix at each iteration of *k*, with the updated distances in **bold**, will be:

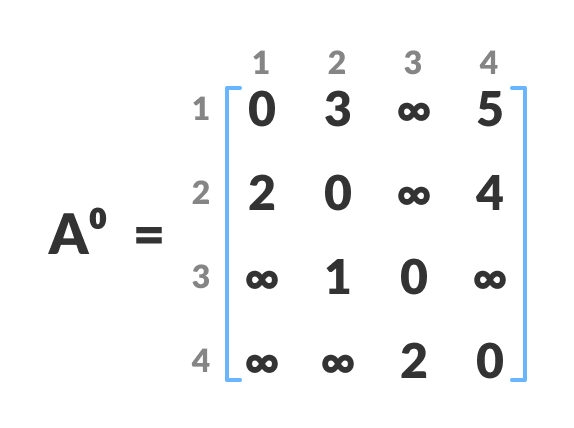


**Example 3:**

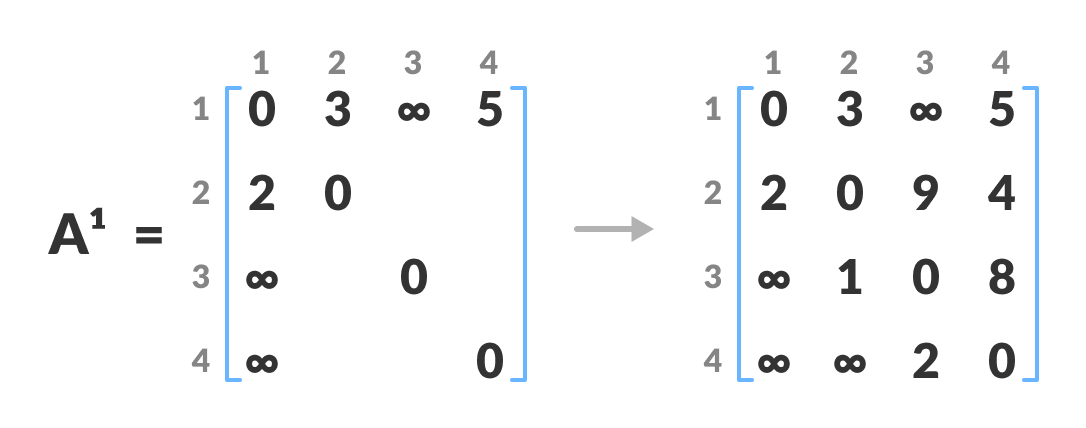


Follow the steps below to find the shortest path between all the pairs of vertices.

1. 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.

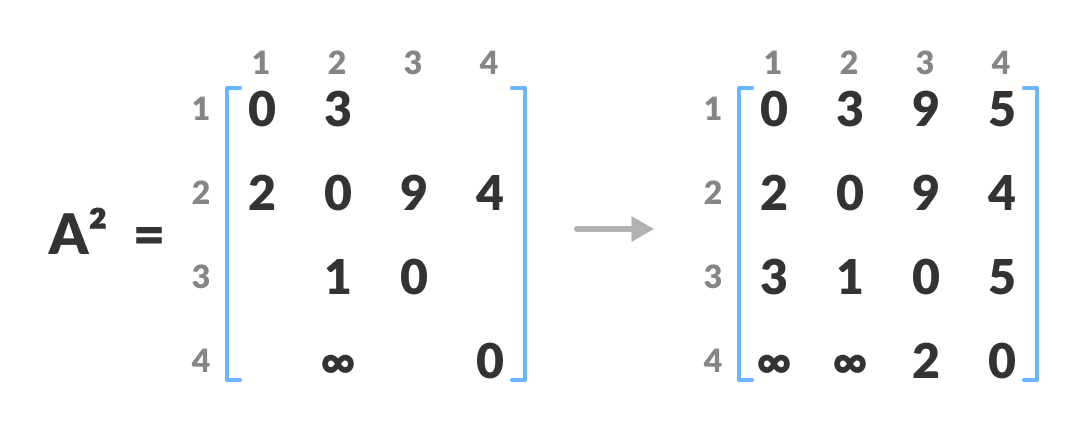
Fill each cell with the distance between ith and jth vertex

1. 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.

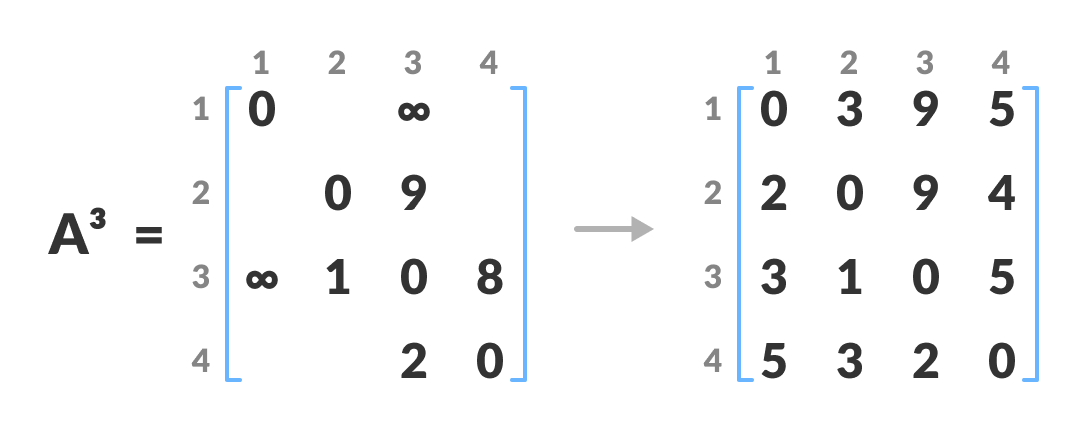
Calculate the distance from the source vertex to destination vertex through this vertex k.

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.

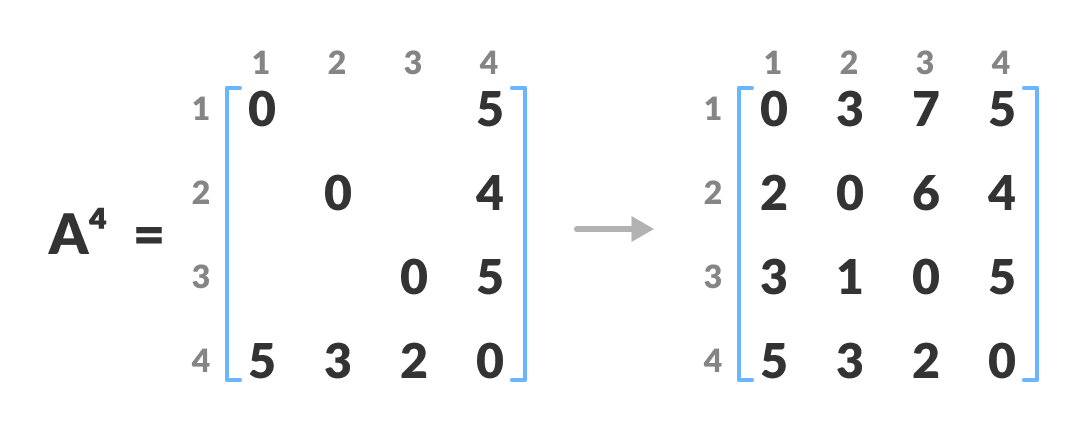
1. Similarly, A2 is created using A3. The elements in the second column and the second row are left as they are.  
   In this step, k is the second vertex (i.e. vertex 2). The remaining steps are the same as in step 2.



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

1. Similarly, A3 and A4 is also created. 

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



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

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

**Floyd-Warshall Algorithm**

n = no of vertices

A = matrix of dimension n\*n

for k = 1 to n

for i = 1 to n

for j = 1 to n

Ak[i, j] = min (Ak-1[i, j], Ak-1[i, k] + Ak-1[k, j])

return A

**Python code:**

# Floyd Warshall Algorithm in python

# The number of vertices

nV = 4

INF = 999

# Algorithm implementation

def floyd\_warshall(G):

distance = list(map(lambda i: list(map(lambda j: j, i)), G))

# Adding vertices individually

for k in range(nV):

for i in range(nV):

for j in range(nV):

distance[i][j] = min(distance[i][j], distance[i][k] + distance[k][j])

print\_solution(distance)

# Printing the solution

def print\_solution(distance):

for i in range(nV):

for j in range(nV):

if(distance[i][j] == INF):

print("INF", end=" ")

else:

print(distance[i][j], end=" ")

print(" ")

G = [[0, 3, INF, 5],

[2, 0, INF, 4],

[INF, 1, 0, INF],

[INF, INF, 2, 0]]

floyd\_warshall(G)

output:

0 3 7 5

2 0 6 4

3 1 0 5

5 3 2 0

Video lecture for Floyd’s Warshall Algorithm

<https://www.youtube.com/watch?v=eujc7_0FhCI>