Data Structures and Algorithms

Lecture 37: Floyd-Warshall Algorithm

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).

A weighted graph is a graph in which each edge has a numerical value associated with it.

Floyd-Warshall Algorithm

- Floyd Warshall Algorithm is a famous algorithm.
- It is used to solve All Pairs Shortest Path Problem.
- It computes the shortest path between every pair of vertices of the given graph.
- Floyd Warshall Algorithm is an example of dynamic programming approach.

Advantages-

Floyd Warshall Algorithm has the following main advantages-

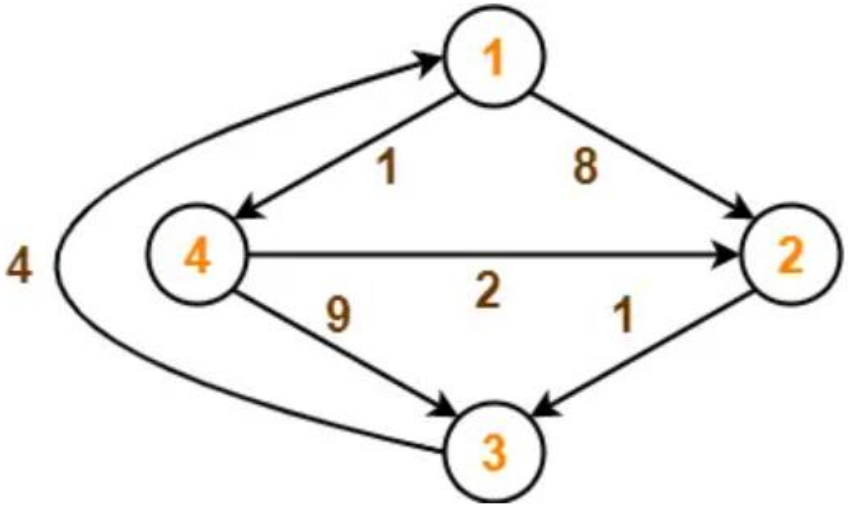
- It is extremely simple.
- It is easy to implement.

When Floyd Warshall Algorithm Is Used?

- Floyd Warshall Algorithm is best suited for dense graphs.
- This is because its complexity depends only on the number of vertices in the given graph.
- For sparse graphs, Johnson's Algorithm is more suitable.

Floyd-Warshall Algorithm

Consider the following directed weighted graph-



Using Floyd Warshall Algorithm, find the shortest path distance between every pair of vertices.

Floyd-Warshall Algorithm

Step-01:

- Remove all the self loops and parallel edges (keeping the lowest weight edge) from the graph.
- In the given graph, there are neither self edges nor parallel edges.

Floyd-Warshall Algorithm

Step-02:

- Write the initial distance matrix.
- It represents the distance between every pair of vertices in the form of given weights.
- For diagonal elements (representing self-loops), distance value = 0.
- For vertices having a direct edge between them, distance value = weight of that edge.
- For vertices having no direct edge between them, distance value = ∞.

Initial distance matrix for the given graph is-

$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

Step-03:

Using Floyd Warshall Algorithm, write the following 4 matrices-

$D_1 =$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	9	0

$D_2 =$

	1	2	3	4
1	0	8	9	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	3	0

$D_3 =$

	1	2	3	4
1	0	8	9	1
2	5	0	1	6
3	4	12	0	5
4	7	2	3	0

$D_4 =$

	1	2	3	4
1	0	3	4	1
2	5	0	1	6
3	4	7	0	5
4	7	2	3	0

The last matrix D4 represents the shortest path distance between every pair of vertices.

- In the above problem, there are 4 vertices in the given graph.
- So, there will be total 4 matrices of order 4 x 4 in the solution excluding the initial distance matrix.
- Diagonal elements of each matrix will always be 0.

Floyd's Algorithm (pseudocode and analysis)

ALGORITHM *Floyd*$(W[1..n, 1..n])$

```
//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths
D ← W //is not necessary if W can be overwritten
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
return D
```

Time efficiency: $\Theta(n^3)$

Space efficiency: Matrices can be written over their predecessors

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

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

Modified Warshall's Algorithm

- Suppose we have a Graph G with n vertices as $v_1, v_2, v_3, \ldots\ldots\ldots, v_n$.
- First we will take the matrices $Q_0, Q_1, Q_3, \ldots\ldots\ldots, Q_k$ where Q_k[i][j] is defined as

Q_k[i][j]= length of the shortest path from v_i, v_j using nodes $v_1, v_2, v_3, \ldots\ldots\ldots, v_k$.

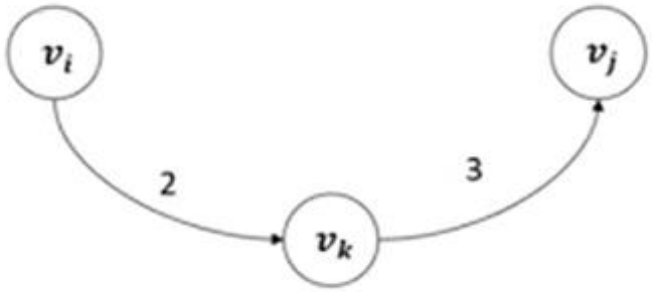
$=\infty$ otherwise

- Q_0[i][j]= length of the shortest path from v_i to v_j.
- Q_1[i][j]= length of the shortest path from v_i to v_j using v_i
- Q_2[i][j]= length of the shortest path from v_i to v_j using v_1, v_2

.....

.....

- Q_k[i][j]= length of the shortest path from v_i to v_j using $v_1, v_2, v_3, \ldots\ldots\ldots, v_k$



W= $\begin{matrix} \infty & \infty & 2 \\ \infty & \infty & \infty \\ \infty & 3 & \infty \end{matrix}$

If $Q_{k-1}[i][j] = \infty$ then

$Q_k[i][j]$ = Minimum($Q_{k-1}[i][j]$, $Q_{k-1}[i][k] + Q_{k-1}[k][j]$)

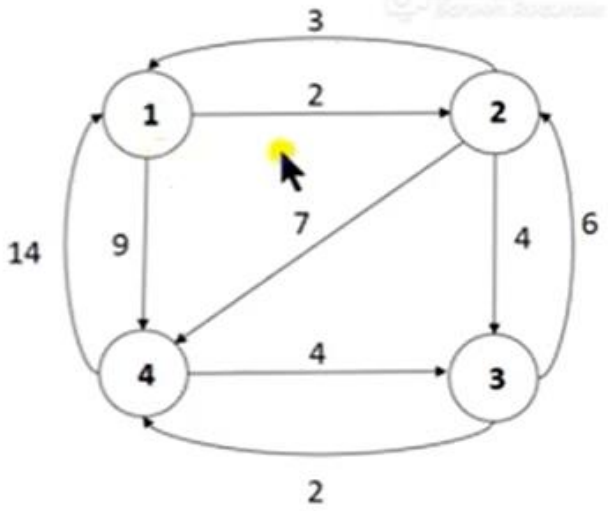
Example

$$W=\begin{bmatrix}0 & 2 & 0 & 9\\ 3 & 0 & 4 & 7\\ 0 & 6 & 0 & 2\\ 14 & 0 & 4 & 0\end{bmatrix}$$

$$Q_0=\begin{bmatrix}\infty & 2 & \infty & 9\\ 3 & \infty & 4 & 7\\ \infty & 6 & \infty & 2\\ 14 & \infty & 4 & \infty\end{bmatrix}$$

After including node 1

$$Q_1=\begin{bmatrix}\infty & 2 & \infty & 9\\ 3 & \mathbf{5} & 4 & 7\\ \infty & 6 & \infty & 2\\ 14 & \mathbf{16} & 4 & \mathbf{23}\end{bmatrix}$$



$Q_1(2,2)=\min(Q_0(2,2), Q_0(2,1)+Q_0(1,2))$