Floyd-Warshall Algorithm

CS313E

What is Floyd-Warshall Algorithm ?

- **The Floyd-Warshall algorithm is a dynamic programming algorithm formulation to solve the all-pairs shortest-paths problem on a directed graph G = (V, E).**

- **The resulting algorithm, known as the *Floyd-Warshall algorithm*.**

- **In Floyds Algorithm, negative-weight edges may be present, but we assume that there are no negative-weight cycles.**

The structure of a shortest path

- Consider a graph G(V, E) with vertices V numbered 1 through N.
- Further consider a function shortest Path(i , j , k) that returns the shortest possible path from i to j using vertices only from the set { 1 , 2 , … , k } as intermediate points along the way.
- Now, given this function, our goal is to find the shortest path from each i to each j using *any* vertex in { 1 , 2 , … , N }.

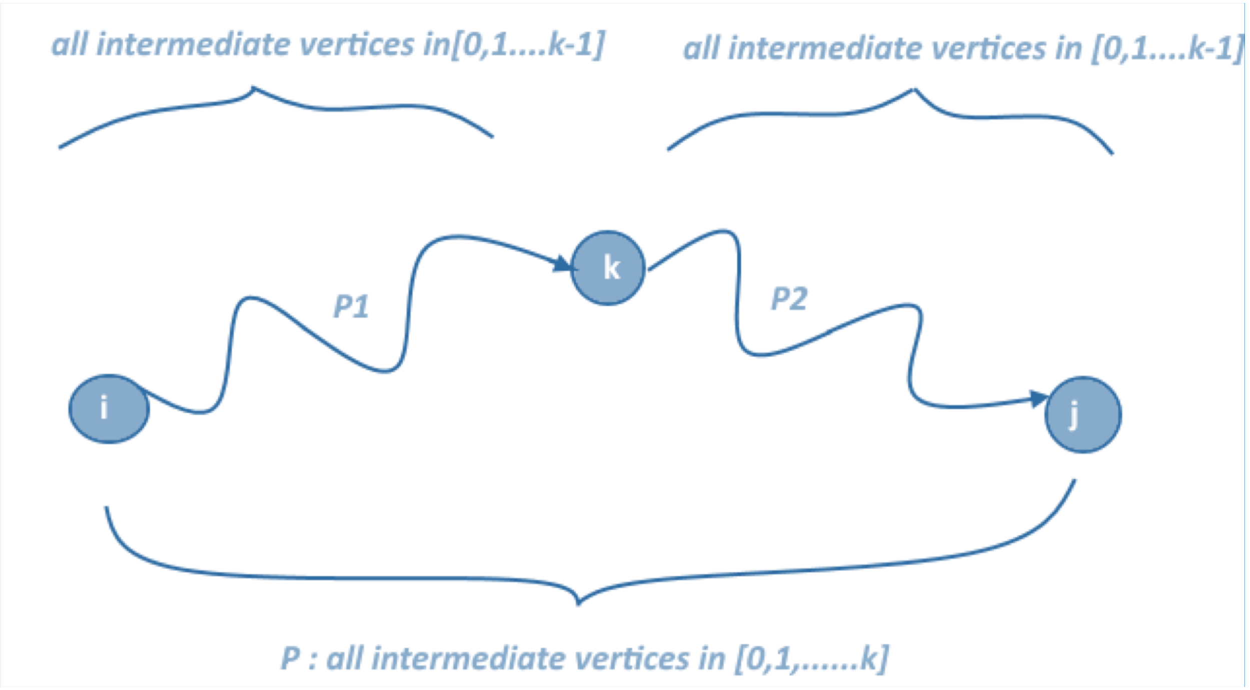
Two possible cases:

1) k is not an intermediate vertex in shortest path from i to j. We keep the value of

dist[i][j] as it is.

2) 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]



- First take the shortest path from I to k using intermediate vertices from the set {1,2.......,k-1}
- Then take the shortest path from k to j using intermediate vertices from the set {1,2.........., k-1}

$$d_{ij}^{k} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$$

A recursive solution to the all-pairs shortest-paths problem

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 , \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 . \end{cases}$$

- This is a recursive formulation of shortest path to find the weight from vertex I to vertex j. for which all intermediate vertices are in the set {1,2..........k}.
- When k = 0, a path from vertex i to vertex j with no intermediate vertex numbered higher than 0 has no intermediate vertices at all.

Floyd-Warshall Algorithm

FLOYD-WARSHALL(W)

$n = W.rows$
$D^{(0)} = W$
for $k = 1$ **to** n
    let $D^{(k)} = \left(d_{ij}^{(k)}\right)$ be a new $n \times n$ matrix
    for $i = 1$ **to** n
        for $j = 1$ **to** n
            $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$

- **Time Complexity**

  There are three loops. Each loop has constant complexities.

  So, the time complexity of the Floyd-Warshall algorithm is **O(n^3).**

Constructing a shortest path

- There are a variety of different methods for constructing shortest paths in the Floyd-Warshall algorithm.

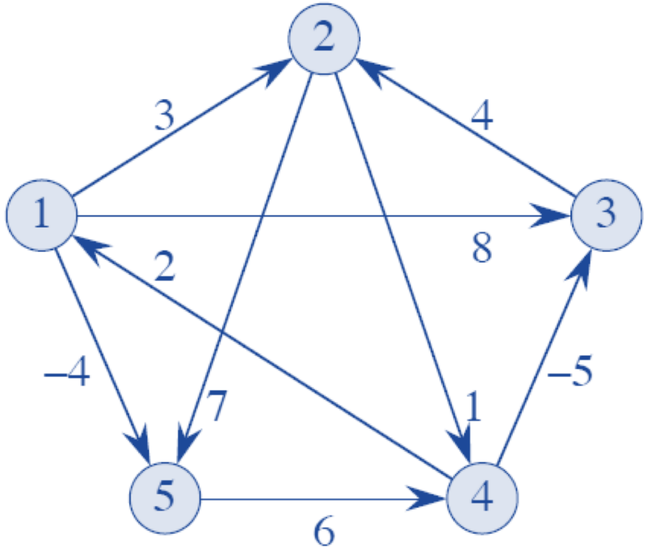
- First One way is to compute the matrix D of shortest-path weights and then construct the predecessor matrix π from the D matrix.

- second one we can compute the predecessor matrix π while the algorithm computes the matrices D^k.

$$\pi_{ij}^{(k)} = \begin{cases} NIL & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

How Floyd-Warshall Algorithm Works?

- Let's take an example:
- Create a matrix D^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 *D[i][j]* is filled with the distance from the i^{th} vertex to the j^{th} vertex. If there is no path from i^{th} vertex to j^{th} vertex, the cell is left as infinity.



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \mathrm{NIL} & 1 & 1 & \mathrm{NIL} & 1 \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 2 & 2 \\ \mathrm{NIL} & 3 & \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} \\ 4 & \mathrm{NIL} & 4 & \mathrm{NIL} & \mathrm{NIL} \\ \mathrm{NIL} & \mathrm{NIL} & \mathrm{NIL} & 5 & \mathrm{NIL} \end{pmatrix}$$

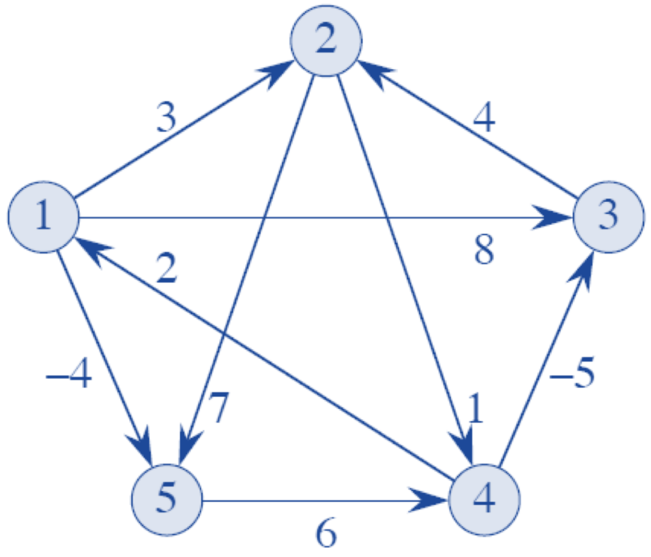
Now, create a matrix D^1 using matrix D^0. The elements in the first column and the first row are left as they are.

Let *k* be the intermediate vertex in the shortest path from source to destination. In this step, *k* is the first vertex. D[i][j] is filled with

(D[i][k] + D[k][j]) if (D[i][j] > D[i][k] + D[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

D[i][k] + D[k][j].



For example:

D^0 [4,2] = [4,1] + [1,2]

∞ = 2+3

∞ > 5

So, we are going to take 5 instead of ∞

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

Similarly, D^2 is also created.

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

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

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

D^5 gives the shortest path between each pair of vertices.

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Conclusion

- ***Floyd-Warshall algorithm is a good way to solve all shortest path problems on a directed graph by using an intermediate vertex.***

- ***Floyd-Warshall algorithm use a dynamic programing to solve all shortest path problems by take a sequence of decision.***

Reading and Visualization

- Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms. MIT press, 2009.
- Floyd-Warshall All-Pairs Shortest Path https://www.cs.usfca.edu/~galles/visualization/Floyd.html