**2.1.4 Floyd Warshall & Bellman Ford**

## Floyd-Warshall Algorithm

Let the vertices of G be V = {1, 2........n} and consider a subset {1, 2........k} of vertices for some k. For any pair of vertices i, j ∈ V, considered all paths from i to j whose intermediate vertices are all drawn from {1, 2.......k}, and let p be a minimum weight path from amongst them. The Floyd-Warshall algorithm exploits a link between path p and shortest paths from i to j with all intermediate vertices in the set {1, 2.......k-1}. The link depends on whether or not k is an intermediate vertex of path p.

If k is not an intermediate vertex of path p, then all intermediate vertices of path p are in the set {1, 2........k-1}. Thus, the shortest path from vertex i to vertex j with all intermediate vertices in the set {1, 2.......k-1} is also the shortest path i to j with all intermediate vertices in the set {1, 2.......k}.

If k is an intermediate vertex of path p, then we break p down into i → k → j.

Let dij(k) be the weight of the shortest path from vertex i to vertex j with all intermediate vertices in the set {1, 2.......k}.

A recursive definition is given by

Floyd-Warshall Algorithm

**FLOYD - WARSHALL (W)**

1. n ← rows [W].

2. D0 ← W

3. for k ← 1 to n

4. do for i ← 1 to n

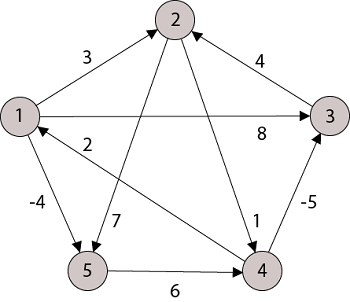
5. do for j ← 1 to n

6. do dij(k) ← min (dij(k-1),dik(k-1)+dkj(k-1) )

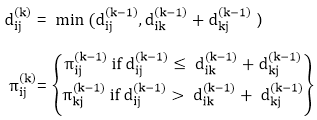
7. return D(n)

The strategy adopted by the Floyd-Warshall algorithm is **Dynamic Programming**. The running time of the Floyd-Warshall algorithm is determined by the triply nested for loops of lines 3-6. Each execution of line 6 takes O (1) time. The algorithm thus runs in time θ(n3 ).

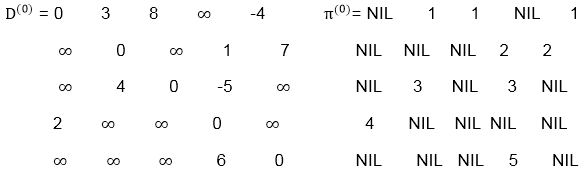
**Example:** Apply Floyd-Warshall algorithm for constructing the shortest path. Show that matrices D(k) and π(k) computed by the Floyd-Warshall algorithm for the graph.



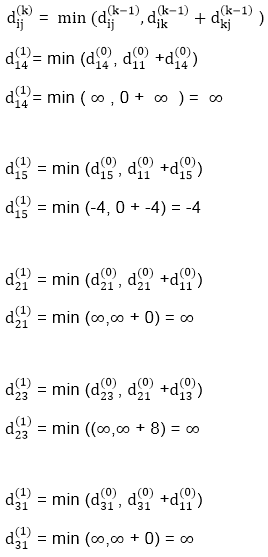
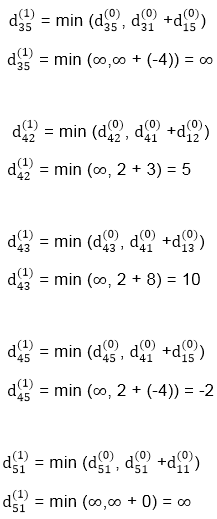
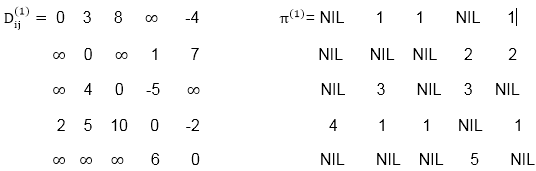
**Solution:**



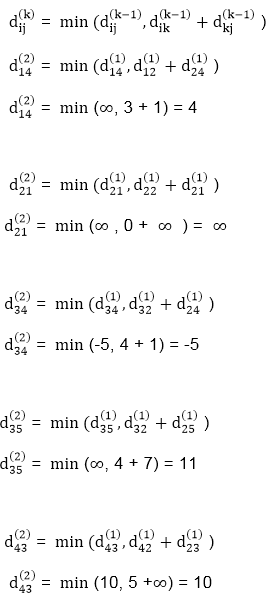
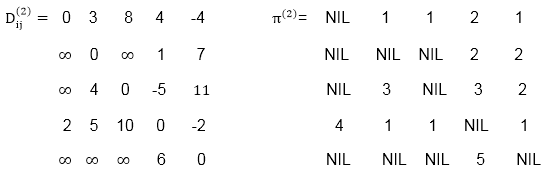
**Step (i)** When k = 0



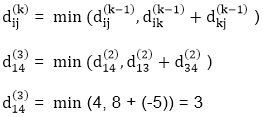
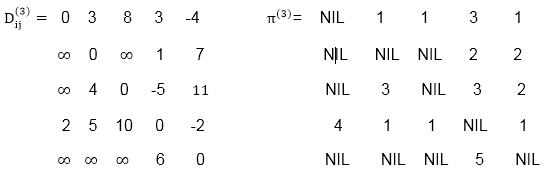
**Step (ii)** When k =1

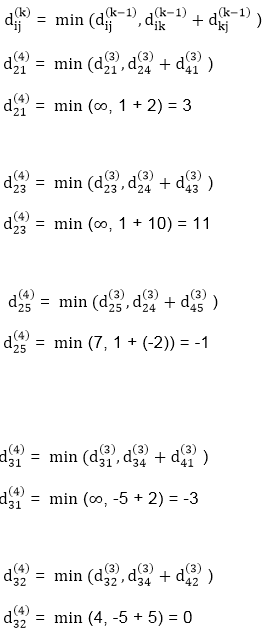
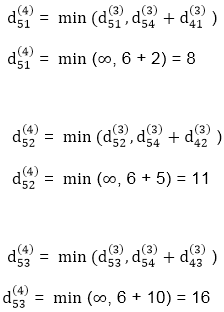
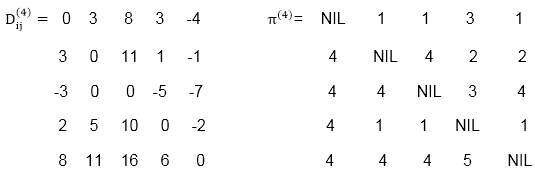
**Step (iii)** When k = 2

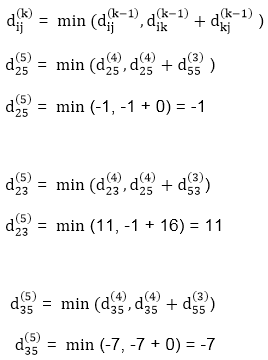
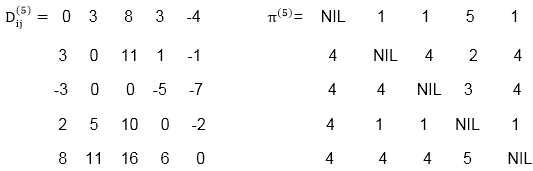
**Step (iv)** When k = 3

**Step (v)** When k = 4

**Step (vi)** When k = 5

**TRANSITIVE- CLOSURE (G)**

1. n ← |V[G]|

2. for i ← 1 to n

3. do for j ← 1 to n

4. do if i = j or (i, j) ∈ E [G]

5. the Floyd-Warshall Algorithm← 1

6. else Floyd-Warshall Algorithm← 0

7. for k ← 1 to n

8. do for i ← 1 to n

9. do for j ← 1 to n

10. dod ij(k) ← Floyd-Warshall Algorithm

11. Return T(n).

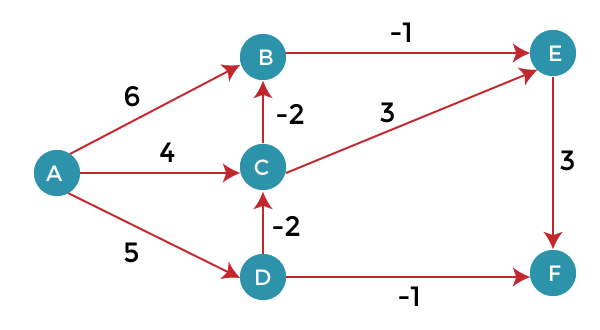
# Bellman Ford Algorithm

Bellman ford algorithm is a single-source shortest path algorithm. This algorithm is used to find the shortest distance from the single vertex to all the other vertices of a weighted graph. There are various other algorithms used to find the shortest path like Dijkstra algorithm, etc. If the weighted graph contains the negative weight values, then the Dijkstra algorithm does not confirm whether it produces the correct answer or not. In contrast to Dijkstra algorithm, bellman ford algorithm guarantees the correct answer even if the weighted graph contains the negative weight values.

**Rule of this algorithm**

1. We will go on relaxing all the edges (n - 1) times where,
2. n = number of vertices

**Consider the below graph:**



As we can observe in the above graph that some of the weights are negative. The above graph contains 6 vertices so we will go on relaxing till the 5 vertices. Here, we will relax all the edges 5 times. The loop will iterate 5 times to get the correct answer. If the loop is iterated more than 5 times then also the answer will be the same, i.e., there would be no change in the distance between the vertices.

Competitive questions on Structures in HindiKeep Watching

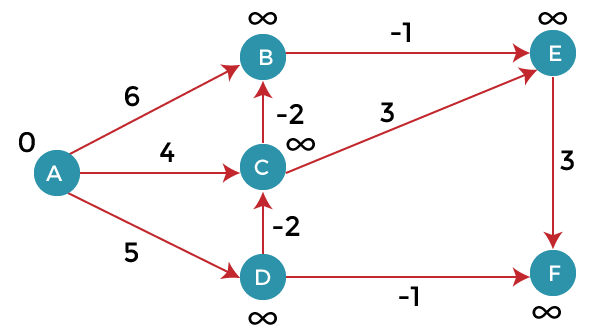
**Relaxing means:**

1. If (d(u) + c(u , v) < d(v))
2. d(v) = d(u) + c(u , v)

To find the shortest path of the above graph, the first step is note down all the edges which are given below:

(A, B), (A, C), (A, D), (B, E), (C, E), (D, C), (D, F), (E, F), (C, B)

Let's consider the source vertex as 'A'; therefore, the distance value at vertex A is 0 and the distance value at all the other vertices as infinity shown as below:



Since the graph has six vertices so it will have five iterations.

**First iteration**

Consider the edge (A, B). Denote vertex 'A' as 'u' and vertex 'B' as 'v'. Now use the relaxing formula:

d(u) = 0

d(v) = ∞

c(u , v) = 6

Since (0 + 6) is less than ∞, so update

1. d(v) = d(u) + c(u , v)

d(v) = 0 + 6 = 6

Therefore, the distance of vertex B is 6.

Consider the edge (A, C). Denote vertex 'A' as 'u' and vertex 'C' as 'v'. Now use the relaxing formula:

d(u) = 0

d(v) = ∞

c(u , v) = 4

Since (0 + 4) is less than ∞, so update

1. d(v) = d(u) + c(u , v)

d(v) = 0 + 4 = 4

Therefore, the distance of vertex C is 4.

Consider the edge (A, D). Denote vertex 'A' as 'u' and vertex 'D' as 'v'. Now use the relaxing formula:

d(u) = 0

d(v) = ∞

c(u , v) = 5

Since (0 + 5) is less than ∞, so update

1. d(v) = d(u) + c(u , v)

d(v) = 0 + 5 = 5

Therefore, the distance of vertex D is 5.

Consider the edge (B, E). Denote vertex 'B' as 'u' and vertex 'E' as 'v'. Now use the relaxing formula:

d(u) = 6

d(v) = ∞

c(u , v) = -1

Since (6 - 1) is less than ∞, so update

1. d(v) = d(u) + c(u , v)

d(v) = 6 - 1= 5

Therefore, the distance of vertex E is 5.

Consider the edge (C, E). Denote vertex 'C' as 'u' and vertex 'E' as 'v'. Now use the relaxing formula:

d(u) = 4

d(v) = 5

c(u , v) = 3

Since (4 + 3) is greater than 5, so there will be no updation. The value at vertex E is 5.

Consider the edge (D, C). Denote vertex 'D' as 'u' and vertex 'C' as 'v'. Now use the relaxing formula:

d(u) = 5

d(v) = 4

c(u , v) = -2

Since (5 -2) is less than 4, so update

1. d(v) = d(u) + c(u , v)

d(v) = 5 - 2 = 3

Therefore, the distance of vertex C is 3.

Consider the edge (D, F). Denote vertex 'D' as 'u' and vertex 'F' as 'v'. Now use the relaxing formula:

d(u) = 5

d(v) = ∞

c(u , v) = -1

Since (5 -1) is less than ∞, so update

1. d(v) = d(u) + c(u , v)

d(v) = 5 - 1 = 4

Therefore, the distance of vertex F is 4.

Consider the edge (E, F). Denote vertex 'E' as 'u' and vertex 'F' as 'v'. Now use the relaxing formula:

d(u) = 5

d(v) = ∞

c(u , v) = 3

Since (5 + 3) is greater than 4, so there would be no updation on the distance value of vertex F.

Consider the edge (C, B). Denote vertex 'C' as 'u' and vertex 'B' as 'v'. Now use the relaxing formula:

d(u) = 3

d(v) = 6

c(u , v) = -2

Since (3 - 2) is less than 6, so update

1. d(v) = d(u) + c(u , v)

d(v) = 3 - 2 = 1

Therefore, the distance of vertex B is 1.

Now the first iteration is completed. We move to the second iteration.

**Second iteration:**

In the second iteration, we again check all the edges. The first edge is (A, B). Since (0 + 6) is greater than 1 so there would be no updation in the vertex B.

The next edge is (A, C). Since (0 + 4) is greater than 3 so there would be no updation in the vertex C.

The next edge is (A, D). Since (0 + 5) equals to 5 so there would be no updation in the vertex D.

The next edge is (B, E). Since (1 - 1) equals to 0 which is less than 5 so update:

d(v) = d(u) + c(u, v)

d(E) = d(B) +c(B , E)

= 1 - 1 = 0

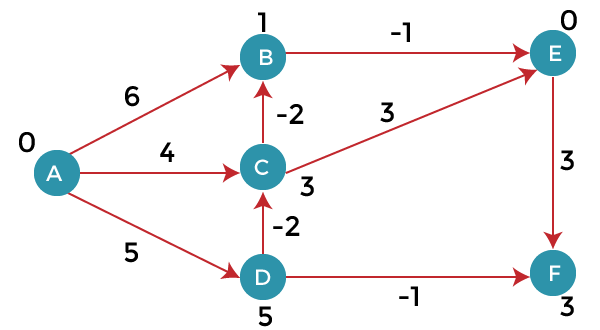
The next edge is (C, E). Since (3 + 3) equals to 6 which is greater than 5 so there would be no updation in the vertex E.

The next edge is (D, C). Since (5 - 2) equals to 3 so there would be no updation in the vertex C.

The next edge is (D, F). Since (5 - 1) equals to 4 so there would be no updation in the vertex F.

The next edge is (E, F). Since (5 + 3) equals to 8 which is greater than 4 so there would be no updation in the vertex F.

The next edge is (C, B). Since (3 - 2) equals to 1` so there would be no updation in the vertex B.



**Third iteration**

We will perform the same steps as we did in the previous iterations. We will observe that there will be no updation in the distance of vertices.

1. The following are the distances of vertices:
2. A: 0
3. B: 1
4. C: 3
5. D: 5
6. E: 0
7. F: 3

**Time Complexity**

The time complexity of Bellman ford algorithm would be O(E|V| - 1).

1. function bellmanFord(G, S)
2. for each vertex V in G
3. distance[V] <- infinite
4. previous[V] <- NULL
5. distance[S] <- 0
7. for each vertex V in G
8. for each edge (U,V) in G
9. tempDistance <- distance[U] + edge\_weight(U, V)
10. if tempDistance < distance[V]
11. distance[V] <- tempDistance
12. previous[V] <- U
14. for each edge (U,V) in G
15. If distance[U] + edge\_weight(U, V) < distance[V}
16. Error: Negative Cycle Exists
18. return distance[], previous[]