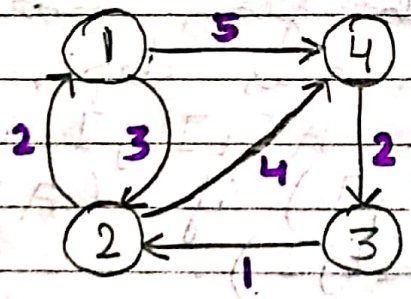
FLOYD - WARSHALL ALGORITHM

Floyd - Warshall Algorithm is an algorithm for finding the shortest path b/w 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)

HOW IT WORKS?

Let the graph be:



STEP-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 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.

$$A_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

STEP: 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 ~~distance~~ direct distance from the source to the destination is greater that 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.

$A_1 =$

	1	2	3	4
1	0	3	∞	5
2	2	0		
3	∞		0	
4	∞			0

⟶

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

For ex: For $A_1[2,4]$, direct distance from vertex 2 to 4 is 4 and the sum of the distance from vertex 2 to 4 through vertex (i.e from vertex 2 to 1 and from vertex 1 to 4) is 7. Since 4 < 7, $A_0[2,4]$ is filled with 4.

STEP 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.

$$A_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & & \\ 2 & 0 & 9 & 4 \\ & 1 & 0 & \\ & \infty & & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 9 & 5 \\ 2 & 0 & 9 & 4 \\ 3 & 1 & 0 & 5 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

STEP:4 Similarly, A_3 and A_4 is also created.

$$A_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & & \infty & \\ & 0 & 9 & \\ \infty & 1 & 0 & 8 \\ & & 2 & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 9 & 5 \\ 2 & 0 & 9 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

$$A_4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & & & 5 \\ & 0 & & 4 \\ & & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 7 & 5 \\ 2 & 0 & 6 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

∴ A^4 gives the shortest path b/w each pair of vertices.

CODE

→ N is small because T.C of this algorithm is $O(n^3)$

```
CONST INT N = 510;
CONST INT INF = 1e9+10;
```

→ Distance matrix to store corresponding distance b/w i, j

```
INT DIST[N][N];

INT MAIN()
{
    → Initial setup of distance matrix
    FOR (INT i=0; i<N; i++)
    {
        FOR (INT j=0; j<N; j++)
        {
            IF (i==j) DIST[i][j] = 0; // If
            i and j are same then distance
            b/w them will be zero, ex:
            distance b/w (1,1) will be 0.
            ELSE DIST[i][j] = INF; // Else
            set all distances to zero. INF
            indicates no path exist b/w these
            pairs of i,j for now
        }
    }

    → Taking inputs and setting their
    weights
    INT n, m;
    CIN >> n >> m;
    FOR (INT i=0; i<m; i++)
    {
        INT x, y, WT;
        CIN >> x >> y >> WT;
        DIST[x][y] = WT;
    }
```

→ Floyd Warshall Algo.

```
FOR (INT K=1; K≤n; K++)
{
    FOR (INT i=1; i≤n; i++)
    {
        FOR (INT j=1; j≤n; j++)
        {
            D3[i][j] = min (D3[i][j]
            D3[i][K] + D3[K][j]);
        }
    }
}
```

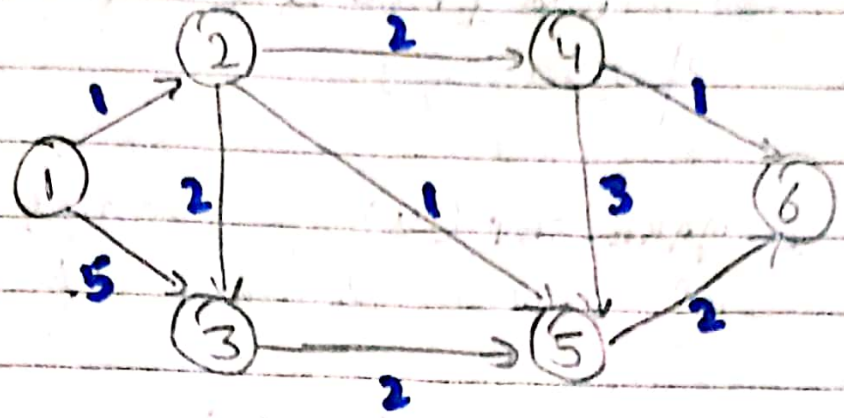
→ Printing Distance Matrix

```
FOR (INT i=1; i≤n; i++)
{
    FOR (INT j=1; j≤n; j++)
    {
        IF (D3[i][j] == INF) COUT << "I";
        // If there is no path b/w (i,j)
        ELSE COUT << D3[i][j] << " ";
    }
    COUT << "\n";
}
RETURN 0;
}
```

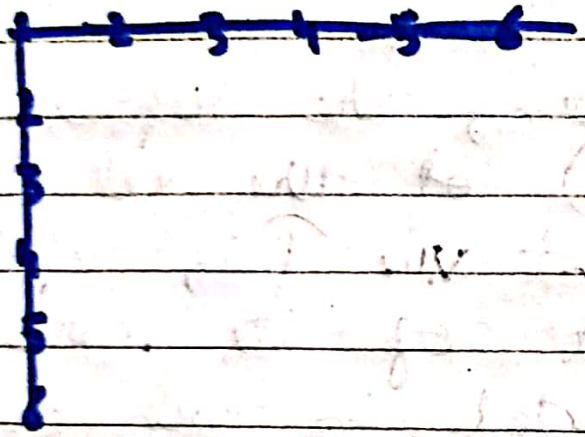
INPUT :

```
6  9
1  2  1
1  3  5
```

2	3	2
2	4	2
2	5	1
3	5	2
4	5	3
4	6	1
5	6	2



OUTPUT:



	1	2	3	4	5	6
1	0	1	3	3	2	4
2	I	0	2	2	1	3
3	I	I	0	I	2	4
4	I	I	I	0	3	1
5	I	I	I	I	0	2
6	I	I	I	I	I	0