# FLOYD's ALGORITHM FOR ALL-PAIR SHORTEST PATH PROBLEM

## Problem Definition:

Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.

CSE@GIT

## Objectives of the Experiment:

1. To introduce the concept of Dynamic Programming.
2. Present the working of Floyd's Algorithm.
3. To find the shortest path form all nodes to all other nodes in a given graph.
4. Analyze the Algorithm Complexity.

# Dynamic Programming:

## Definition:

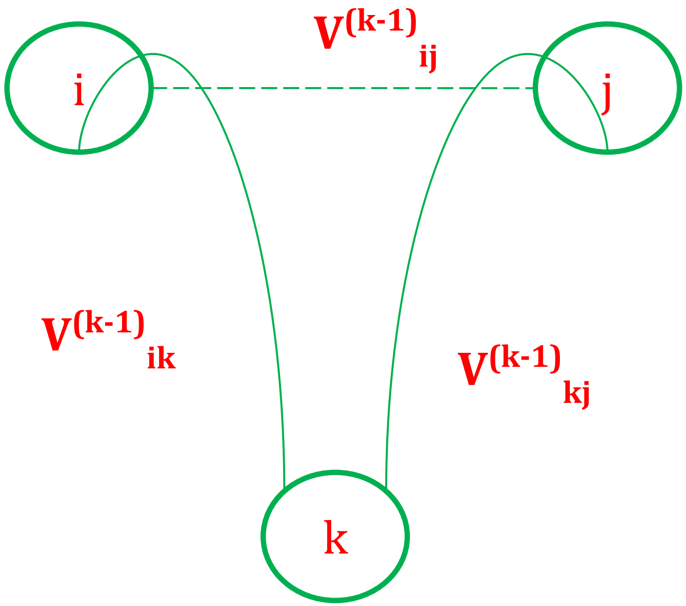
- Dynamic programming is a technique for solving problems with overlapping sub problems.

Dynamic programming suggests solving each of the smaller sub-problems only once and recording the results in a table from which a solution to the original problem can then be obtained.

# Theoretical Background:

# Underlying Idea of Floyd's Algorithm

The idea of Floyd's algorithm is similar to Warshall's Transitive Closure




Transitive Closure:

If there exists an edge between $V_i$ and $V_k$ and also between $V_k$ to $V_j$ then there is a path between $V_i$ and $V_j$

Vij may be larger than Vik and Vkj

Then

Vij is replaced by Vik+Vkj

# Theoretical Background of All Pair shortest path

*The problem:* find the shortest path between every pair of vertices of a graph

*The graph*: may contain negative edges but no negative cycles

*A representation*: a weight matrix where

W( i, j)=0 if i=j.

W( i, j)=∞ if there is no edge between i and j.

W( i, j)="weight of edge"

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

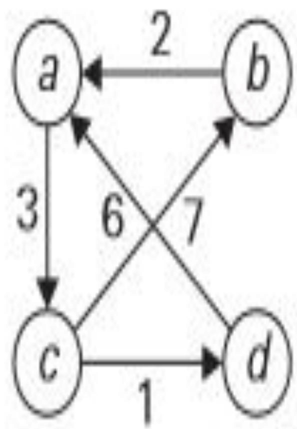
//Implements Floyd's algorithm for the all-pairs shortest-paths problem

//Input: The weight matrix *W of a graph with no negative-length cycle*

//Output: The distance matrix of the shortest paths' lengths

```
D ←W    ,   P [i, j] <- 0                    // Distance matrix <- Weight matrix
for k←1 to n do                  //k is intermediate node
    for i ←1 to n do             // i is source node
        for j ←1 to n do         // j is destination node
            { D[i, j ]←min{D[i, j ], D[i, k]+ D[k, j]} ,  P[i,j] <- k }
return D
```

**Initial Call: Floyd(W[ ][ ], n)**




(a)

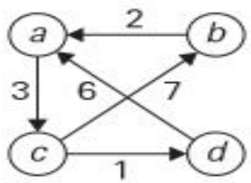
$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

(b)

$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

(c)

**FIGURE 8.14** (a) Digraph. (b) Its weight matrix. (c) Its distance matrix.




$$D^{(0)} = \begin{array}{cc} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix).

$$D^{(1)} = \begin{array}{cc} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \mathbf{5} & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \mathbf{9} & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 1, i.e., just *a* (note two new shortest paths from *b* to *c* and from *d* to *c*).

$$D^{(2)} = \begin{array}{cc} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \mathbf{9} & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

$$D^{(3)} = \begin{array}{cc} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & \mathbf{10} & 3 & \mathbf{4} \\ 2 & 0 & 5 & \mathbf{6} \\ 9 & 7 & 0 & 1 \\ 6 & \mathbf{16} & 9 & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 3, i.e., a, *b*, and *c* (note four new shortest paths from *a* to *b*, from a to *d*, from *b* to *d*, and from *d* to *b*).

$$D^{(4)} = \begin{array}{cc} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ \mathbf{7} & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 4, i.e., a, *b*, *c*, and *d* (note a new shortest path from *c* to *a*).

**FIGURE 8.16** Application of Floyd's algorithm to the digraph shown. Updated elements are shown in bold.

# Efficiency of Floyd's Algorithm

Worst-case performance $\Theta(|V|^3)$

Best-case performance $\Theta(|V|^3)$

Average performance $\Theta(|V|^3)$

# Learning Outcome of the Experiment and Conclusion

At the end of the session, students should be able to :

1. Explain the working of Dynamic Programming.
2. Demonstrate the working of Floyd's algorithm.
3. Write the program in C to implement Floyd's Algorithm.
4. Estimate the shortest path between All pairs of vertices in a weighted connected graph.

# THANK YOU…..