

$$a^0 = 1 [a^0]$$

Welcome to Algorithm Presentation

*Topic:
Floyd-Warshall Algorithm*

$$\arcsin(z)$$

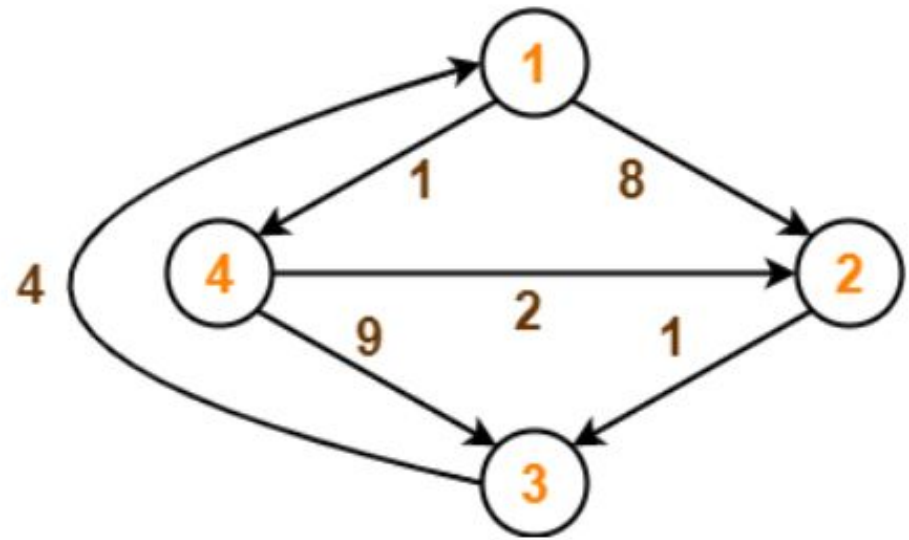
$$x_{n+1} =$$

What is Floyd Warshall Algorithm?

The Floyd Warshall algorithm is a fundamental algorithm in computer science used for finding shortest paths between all pairs of vertices in a weighted graph.

- *Example of dynamic Programming*
- *Useful for dense graphs*
- *Can handle both positive and negative edge weights*
- *No negative cycle*

*For example, we can solve
Shortest path problem of the
following weighted graph by
using Floyd-Warshall
Algorithm*



Step:01

We will create matrix according to vertices given in weighted graph

For example, for this example we will create D_0, D_1, D_2, D_3, D_4

Create a matrix D_0 of dimension where n is the number of vertices.

The row and the column are indexed as i and j respectively

For D_0 we insert direct distance given in weighted graph

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

Step:02

We will leave 1st row and column as same as before

For D₁, we will find shortest distance by going through '1' vertices

We will compare newfound value with previous matrix

We will fill that values which is shortest of two

$D_1 =$

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

Step:03

We will leave 2nd row and column as same as before

For D2, we will find shortest distance by going through '2' vertices

We will compare newfound value with previous matrix

If we get minimum shortest path then we replace otherwise not

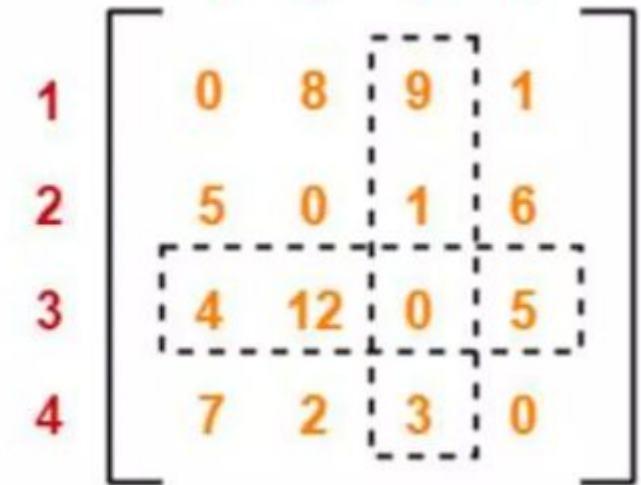
$$D_2 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix} \end{matrix}$$

Step:04

We will leave 3rd row and column as same as before

For D_3 , we will find shortest distance by going through '3' vertices

We will compare newfound value with previous matrix

$$D_3 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{matrix}$$
The diagram shows a 4x4 distance matrix D3. The rows and columns are indexed 1 to 4. A dashed box highlights the 3rd row and 3rd column, indicating that these values are preserved from the previous step. The matrix values are: Row 1: [0, 8, 9, 1], Row 2: [5, 0, 1, 6], Row 3: [4, 12, 0, 5], Row 4: [7, 2, 3, 0].

Step:05

We will leave 4th row and column as same as before

For D4, we will find shortest distance by going through '4' vertices

We will compare newfound value with previous matrix

We will fill that values which is shortest of two otherwise not

$D_4 =$

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

Applications of Floyd-Warshall Algorithm

- *Network Routing Protocol*
- *Traffic Management Systems*
- *Distance Vector Routing Protocols*

Comparative Analysis: Floyd-Warshall Algorithm

Floyd-Warshall vs. Dijkstra's:

Floyd-Warshall handles negative edge weights and finds shortest paths between all pairs of vertices, while Dijkstra's algorithm works for single-source shortest path.

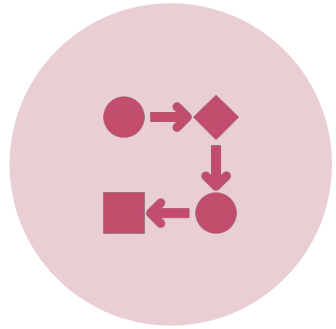


Floyd-Warshall vs. Bellman-Ford:

Floyd-Warshall is more efficient for finding shortest paths between all pairs of vertices, whereas Bellman-Ford works well for graphs with negative cycles.



Advantages of floyd warshall algorithm



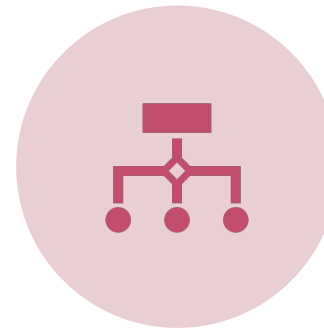
All-pairs shortest paths: Unlike Dijkstra's algorithm that finds the shortest path from a single source, Floyd-Warshall calculates the shortest distances between every pair of vertices in the graph. This is incredibly useful for route planning and network analysis.



Relatively easy to understand and implement: Compared to some other graph algorithms, Floyd-Warshall is considered conceptually straightforward. This makes it a good choice for beginners to learn about dynamic programming.



Efficient for dense graphs: With a time complexity of $O(V^3)$, Floyd-Warshall can handle dense graphs (where most vertices are connected) efficiently.



Parallelizable: The algorithm can be broken down into independent subproblems, allowing it to be run on multiple processors for faster execution.

Disadvantages of floyd warshall algorithm



Slow for sparse graphs: For sparse graphs (where there are few connections between vertices), the $O(V^3)$ complexity becomes a bottleneck. There are more efficient algorithms like Dijkstra's for sparse graphs.



Memory usage: The algorithm uses a distance matrix to store all pair-wise shortest distances. This can consume significant memory for large graphs.



Doesn't provide actual paths: While it calculates shortest distances, Floyd-Warshall doesn't explicitly tell you the exact sequence of vertices for the shortest path between two points. You might need additional modifications to find the actual paths.



Not suitable for dynamic graphs: If the edge weights in your graph change frequently, Floyd-Warshall becomes inefficient as it requires recalculating the entire matrix for each change.

Complexity Analysis

1. Time

2. Space



Time Complexity Analysis

Triple Nested Loops:

Outer Loop (k): Runs V times

Middle Loop (i): Runs V times

Inner Loop (j): Runs V times

Total Iterations:

$$V \times V \times V = V^3$$

Thus, the time complexity is $O(V^3)$.

Graphical Representation:

| Number of Vertices (V) | Time Complexity |
|------------------------|--------------------------|
| ----- | ----- |
| 10 | 1,000 operations |
| 100 | 1,000,000 operations |
| 1,000 | 1,000,000,000 operations |



Best Case, Average Case, and Worst Case

Best Case

In the best case, if the initial input distance matrix already represents the shortest paths (i.e., no updates are needed), the algorithm still needs to iterate through all possible triples of vertices to confirm this.

Time Complexity: $O(V^3)$.

Average Case

On average, the graph may have a typical distribution of edge weights and distances. The algorithm will generally perform a mix of updates and checks.

Time Complexity: $O(V^3)$.

Worst Case

In the worst case, the algorithm must update the shortest path estimate for every pair of vertices through every possible intermediate vertex.

Time Complexity: $O(V^3)$.

Space Complexity Examination

Distance Matrix (dist):

The algorithm maintains a $V \times V$ matrix (dist) where $\text{dist}[i][j]$ represents the shortest distance from vertex (i) to vertex (j).

This matrix requires: (V^2)

Thank you

Presented By :

Riyadul Islam Riyadh

Rokaiya Sultana Bonhy

Habiba Sultana Othy

Md. Abu Talab Anik

Presented to :

Sumona Yeasmin

Lecturer

Dept. of CSE , BUBT