PART A

(PART A : TO BE REFFERED BY STUDENTS)

Experiment No.06

A.1 Aim:

Write a program to implement all pair shortest path using Dynamic Programming Approach.

A.2 Prerequisite: -

A.3 Outcome:

After successful completion of this experiment students will be able to solve a problem by applying dynamic programming approach.

A.4 Theory:

The Floyd–Warshall algorithm is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles). A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices, though it does not return details of the paths themselves. Versions of the algorithm can also be used for finding the transitive closure of a relation R, or (in connection with the Schulze voting system) widest paths between all pairs of vertices in a weighted graph.

The Floyd–Warshall algorithm is an example of dynamic programming, The algorithm is also known as Floyd's algorithm, the Roy–Warshall algorithm, the Roy–Floyd algorithm, or the WFI algorithm.

Algorithm:

let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)

for each vertex v

dist[v][v] ← 0

for each edge (u,v)

dist[u][v] ← w(u,v) // the weight of the edge (u,v)

for k from 1 to |V|

for i from 1 to |V|

for j from 1 to |V|

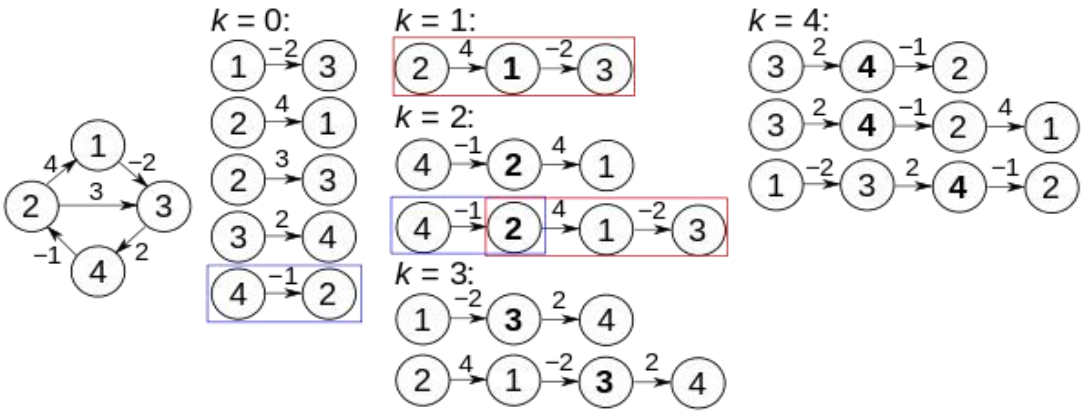
if dist[i][j] >dist[i][k] + dist[k][j]

dist[i][j] ← dist[i][k] + dist[k][j]

end if

Example:

The algorithm is executed on the graph on the below:



Time Complexity:

First double for loop = O(n^2)

Nested 3 for loop = O(n^3)

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

Roll No.:A35	Name: Sanika Gulabrao Nalawade
Class:AI&DS	Batch:A2
Date of Experiment: 20-3-2024	Date of Submission: 26-3-2024
Grade:	

B.1 Software Code written by student:

```
import java.util.*;


public class FloydWarshall {


   public static int INF = 99999, V = 0;


   public static void floydWarshall(int graph[][]) {
      int dist[][] = new int[V][V];
      int i, j, k;


      for (i = 0; i < V; i++)
         for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];


      for (k = 0; k < V; k++) {
         for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
```

```
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    printSolution(dist);
}

public static void printSolution(int dist[][]) {
    System.out.println("The following matrix shows the shortest distances between every pair of vertices:");
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            if (dist[i][j] == INF)
                System.out.print("INF ");
            else
                System.out.print(dist[i][j] + "   ");
        }
        System.out.println();
    }
}

public static void main(String[] args) {
    Scanner sn = new Scanner(System.in);
    System.out.print("Enter the number of vertices: ");
```

```
        V = sn.nextInt(); // Read number of vertices
        sn.nextLine(); // Consume newline left-over

        int graph[][] = new int[V][V];

        System.out.println("Enter the matrix (use 'INF' for infinity):");
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                String input = sn.next();
                if (input.equalsIgnoreCase("INF")) {
                    graph[i][j] = INF;
                } else {
                    graph[i][j] = Integer.parseInt(input);
                }
            }
            if (sn.hasNextLine()) {
                sn.nextLine(); // Consume newline left-over
            }
        }

        sn.close();
        floydWarshall(graph);
    }
}
```

B.2 Input and Output:

```
Enter the number of vertices: 4
Enter the matrix (use 'INF' for infinity):
0 5 INF 10
INF 0 3 NF
Exception in thread "main" java.lang.NumberFormatException: For input string:
 "NF"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at FloydWarshall.main(FloydWarshall.java:55)
PS C:\Users\admin\Documents\A@(> javac FloydWarshall.java
PS C:\Users\admin\Documents\A@(> java FloydWarshall
Enter the number of vertices: 4
Enter the matrix (use 'INF' for infinity):
0 5 INF 10
INF 0 3 INF INF
INF INF 0 1
INF INF INF 0
The following matrix shows the shortest distances between every pair of verti
ces:
0   5   8   9
INF 0   3   4
INF INF 0   1
INF INF INF 0
```

B.3 Observations and learning:

- The Floyd-Warshall algorithm is a dynamic programming approach that solves the all-pairs shortest path problem.
- It systematically checks all possible paths in the graph to find the shortest paths between every pair of vertices.
- The algorithm is versatile as it can handle graphs with negative weights, provided there are no negative cycles.
- It is particularly useful for dense graphs where every node is likely to be connected to many other nodes.

B.4 Conclusion:

- The Floyd-Warshall algorithm is an efficient method to find the shortest paths between all pairs of vertices in a weighted graph.
- It is a robust algorithm that can accommodate various types of graphs, including those with negative weights.
- The algorithm's simplicity and ability to handle negative weights make it a valuable tool in the field of graph theory.

B.5 Question of Curiosity

Q1: What are different algorithms available to find shortest path?

Different Algorithms available for finding shortest path include :

- **Depth-First Search (DFS):** Primarily used for unweighted graphs, it explores as far as possible along each branch before backtracking. It's not guaranteed to find the shortest path in a weighted graph.
- **Breadth-First Search (BFS):** Effective for unweighted graphs, it finds the shortest path from a single source to all other vertices by exploring neighbor nodes level by level.
- **Dijkstra's Algorithm:** A single-source shortest path algorithm for graphs with non-negative weights. It uses a greedy approach and is efficient for sparse graphs.
- **Bellman-Ford Algorithm:** Can handle graphs with negative weights and detects negative cycles. It's slower than Dijkstra's but more versatile.
- *A Search Algorithm:** Uses heuristics to find the shortest path faster by prioritizing paths that seem to be leading closer to the goal.
- **Floyd-Warshall Algorithm:** An all-pairs shortest path algorithm that calculates the shortest paths between every pair of vertices in a weighted graph.
- **Johnson's Algorithm:** Useful for sparse graphs, it combines both Dijkstra's and Bellman-Ford algorithms to find shortest paths between all pairs of vertices.

Q2: Derive time complexity of Floyd's algorithm?

The time complexity of Floyd's algorithm is (O(V^3)), where (V) is the number of vertices in the graph. This cubic time complexity arises from the three nested loops that the algorithm uses to update the shortest path matrix for all pairs of vertices.

- **Best Case Scenario (O(V^3)):** Even in the best case, where no path relaxations are needed because the shortest paths are already determined, the algorithm still needs to perform all iterations of the nested loops. Each loop iterates (V) times, leading to the cubic time complexity.
- **Average Case Scenario (O(V^3)):** The average case also has a time complexity of (O(V^3)). This holds true across various graph structures and densities since the performance primarily depends on the number of vertices and the iterations needed to compute the shortest paths.
- **Worst Case Scenario (O(V^3)):** In the worst case, the algorithm performs relaxation steps for each pair of vertices during all iterations of the nested loops. The time complexity remains (O(V^3)) due to the need to update distances between vertices multiple times until the shortest paths are determined.

The space complexity is (O(V^2)), as the algorithm requires a 2D array to store the shortest distances between all pairs of vertices. Additional variables used for iteration and calculation are minimal compared to the distance matrix,

Q3: Compare Floyd's algorithm with other algorithms?

Floyd's algorithm is an all-pairs shortest path algorithm, unlike Dijkstra's or Bellman-Ford, which are single-source shortest path algorithms. Here's how Floyd's algorithm compares to others:

Dijkstra's Algorithm:

- It is a single-source shortest path algorithm that uses a greedy approach.
- It maintains a set of vertices with known minimum distances from the source and selects the vertex with the shortest known distance for exploration.
- The time complexity can vary from (O(V^2)) for dense graphs to (O(E + V \log V)) for sparse graphs using priority queues or min-heaps.

Bellman-Ford Algorithm:

- It can handle graphs with negative weights and detect negative cycles.
- It relaxes edges repeatedly to find the shortest path from a single source to all other vertices.
- The time complexity is (O(VE)), where (E) is the number of edges.

Floyd's Algorithm:

- It uses a dynamic programming approach to compute the shortest distance between all pairs of vertices.
- It iteratively considers all possible intermediate vertices to optimize the paths between vertex pairs.
- The time complexity is (O(V^3)), which is independent of the graph's edge density.

Q4: Explain application areas of all pair shortest path algorithm.

All-pair shortest path algorithms like Floyd-Warshall have a wide range of applications:

- **Network Routing:** Used in computer networking to determine the shortest path for data packets across a network.
- **Flight Connectivity Analysis:** In the aviation industry, it helps in finding the shortest routes between airports.
- **Geographic Information Systems (GIS):** Utilized for analyzing spatial data, such as road networks, to find the shortest paths between locations.
- **Urban Planning:** Helps in optimizing the layout of infrastructure like roads and utilities.
- **Social Network Analysis:** Can be used to find the shortest connection paths between individuals in a social network.

****