Vidyavardhini's College of Engineering & Technology

Department of Computer Science and Engineering (Data Science)

ACADEMIC YEAR: 2024-25

Course: Analysis of Algorithm Lab

Course code: CSL401

Year/Sem: SE/IV

Experiment No.: 06
Aim: To implement All pair shortest path – Floyd Warshall algorithm using Dynamic programming approach
Name: SOHAM HEMENDRA RAUT
Roll Number: 24
Date of Performance: 06/03/2025
Date of Submission: 13/03/2025

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission.	10	
Total	**20**	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	5	3	2
Understanding	5	3	2
Journal work and timely submission.	10	8	4

Checked by

Name of Faculty : Mrs. Komal Champanerkar

Signature : **Date** :

- **Aim: To implement All-pairs shortest path – Floyd Warshall algorithm using Dynamic programming approach.**
- **Theory:**

- *All pair shortest path:* The All Pair Shortest Path problem aims to find the shortest paths between every pair of vertices in a graph. Given a graph with weighted edges, the problem finds the shortest path between all pairs of nodes. This problem is often solved using algorithms like Floyd-Warshall.

- *Floyd warshall:* The Floyd-Warshall algorithm is a well-known algorithm used to find the shortest paths between all pairs of vertices in a weighted graph. It works by progressively improving an estimate of the shortest paths between two vertices, considering intermediate vertices.
  The key idea of the Floyd-Warshall algorithm is that it uses a dynamic programming approach to gradually refine the shortest paths by adding intermediate vertices one by one.

- **Floyd Warshall Algorithm:**
  Step 1: Initialize the distance matrix:
  Set `dist[i][j]` to the weight of the edge between vertices i and j (or infinity if no edge exists).
  Set `dist[i][i] = 0` for all vertices i.

  Step 2: Iterate through each intermediate vertex kkk:
  For each kkk, check all pairs of vertices i, ji, ji,j.

  Step 3: Update distances:
  For each pair of vertices i,j, update the distance if a shorter path through vertex k exists:
  dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j])

Step 4: Repeat for all intermediate vertices k from 0 to n−1.

Step 5: Result

After completing the iterations, `dist[i][j]` will hold the shortest distance between vertices i and j.

- **Complexity:**

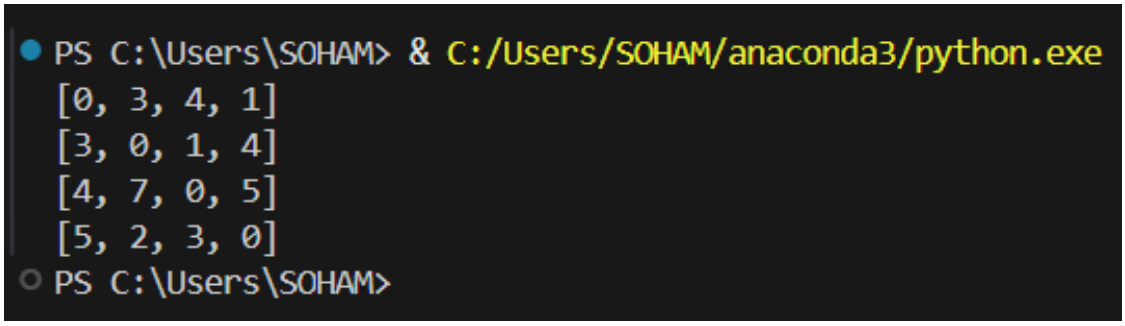
  The Floyd-Warshall algorithm consists of three nested loops over the vertices. The outer loop iterates over all intermediate vertices, and the two inner loops iterate over all pairs of vertices. Therefore, the time complexity of the algorithm is O(n^3).

❖ **Program:**

```
def floyd_warshall(graph):
        n = len(graph)              # Number of vertices in the graph
        # dist[][] will be the shortest distance matrix
        dist = [[float('inf')] * n for _ in range(n)]
        # Initialize the distance matrix with the graph's weights
        for i in range(n):
        for j in range(n):
        if i == j:
          dist[i][j] = 0  # self loop
        elif graph[i][j] != 0:
          dist[i][j] = graph[i][j]  # Direct edge weight
```

```
        for k in range(n):
        for i in range(n):              # Start node
        for j in range(n):              # End node
        # If the path from i to j through k is shorter, update dist[i][j]
        if dist[i][j] > dist[i][k] + dist[k][j]:
            dist[i][j] = dist[i][k] + dist[k][j]
        return dist
graph = [
        [0, 8, 0, 1],
        [3, 0, 1, 0],
        [4, 0, 1, 0],
        [0, 2, 0, 9],
]
 # Call the Floyd-Warshall function
shortest_paths = floyd_warshall(graph)
 # Print the shortest distance matrix
for row in shortest_paths:
        print(row)
```

❖ **Output:**



❖ **Conclusion:** The Floyd-Warshall algorithm efficiently solves the All Pair Shortest Path problem with a time complexity of O(n3)O(n^3) and space complexity of O(n2)O(n^2). While simple and reliable, it becomes impractical for large graphs due to its cubic time complexity, making it better suited for smaller or dense graphs.