Worksheet 5

Student Name: *Ramesh Kumar* **UID:** *25MCI10304*
Branch: *MCA (AI & ML)* **Section/Group:** *25MAM-5A*
Semester: *1st* **Date of Performance:** *06-10-2025*
Subject Name: *Design and Analysis of Algorithm* **Subject Code:** *25CAP-612*

1. ***Aim/Overview of the practical:***

   *To determine the **transitive closure** of a directed graph by applying **Warshall's algorithm**.*

2. ***Task to be done:***

- *Start the program.*
- *Input the number of vertices n.*
- *Read the adjacency matrix of the graph, where each element (i, j) represents the weight of the edge from vertex i to vertex j.*
- *Use 'inf' if there is no direct edge between two vertices.*
- *Initialize a distance matrix dist with the same values as the adjacency matrix.*
- *Repeat for each vertex k (as an intermediate node):*
- *For each vertex i (source):*
- *For each vertex j (destination):*
- *Update dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]).*
- *Continue this process until all vertices have been used as intermediate nodes.*
- *Display the final Shortest Path Matrix, showing the minimum distance between every pair of vertices.*
- *Print 'inf' where no path exists.*
- *Verify the output with a sample input.*
- *Stop the program.*

3. ***Algorithm/Flowchart:***

   Step 1: Start
   Step 2: Input the number of vertices n in the graph.
   Step 3: Read the adjacency matrix graph[n] where:

- *graph[i][j] represents the weight of the edge from vertex i to vertex j.*
- *if there is no directed edge between I and j,set graph [i][j] = infinity*

Step 4: Initialize a distance matrix dist as a copy of the adjacency

matrix:

dist[i][j] = graph[i][j]

Step 5: Repeat for each intermediate vertex k from 0 to n-1: For each vertex i from 0 to n-1:

- *For each vertex j from 0 to n-1:*
- *If both dist[i][k] and dist[k][j] are not infinity: dist[i][j] =*

  min(dist[i][j], dist[i][k] + dist[k][j])

Step 6: End of loops

- *→ Now, dist[i][j] contains the shortest distance from vertex i to vertex j.*

Step 7: Display the final shortest path matrix.

Step 8: Stop

4. Code for experiment/practical:

exp5.py > floyd_warshall_algo

```
inf = float('inf')

def floyd_warshall_algo(matrix):
    n = len(matrix)
    dist = [row[:] for row in matrix]
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    print("\nFinal Shortest Path Matrix:")
    for i in range(n):
        for j in range(n):
            print(0 if dist[i][j] == inf else dist[i][j], end="\t")
        print()

v = int(input("Enter total number of vertices: "))
print("Enter adjacency matrix (use 'inf' for no direct connection):")

graph = []
for i in range(v):
    row_input = input().split()
    if len(row_input) != v:
        exit()
    row = [inf if val.lower() == 'inf' else int(val) for val in row_input]
    graph.append(row)

floyd_warshall_algo(graph)

```

5.Output:

TERMINAL

```
PS R:\Desktop\DAA> & C:/Python313/python.exe r:/Desktop/DAA/exp5.py
Enter total number of vertices: 4
Enter adjacency matrix (use 'inf' for no direct connection):
0 5 inf 10
inf 0 3 inf
inf inf 0 1
inf inf inf 0

Final Shortest Path Matrix:
0       5       8       9
0       0       3       4
0       0       0       1
0       0       0       0
PS R:\Desktop\DAA>
```

6. Learning outcomes:

- Understand the **concept and working principle** of the **Floyd–Warshall algorithm** used to find the **shortest paths between all pairs of vertices** in a weighted graph.
- **Represent graphs** using an **adjacency matrix** and correctly handle **infinite or no-edge conditions** during implementation.
- Apply the **dynamic programming approach** to efficiently solve **all-pairs shortest path problems**.
- Evaluate the **time complexity ($O(V^3)$)** and **space complexity ($O(V^2)$)** of the Floyd–Warshall algorithm.
- Design and **implement a Python program** to compute and display the **shortest path matrix**.
- **Interpret and analyze** the resulting shortest path matrix to understand how **intermediate vertices contribute to optimal path selection**.

-