



Worksheet 5

Student Name: Ramesh Kumar

Branch: MCA (AI & ML)

Semester: 1st

Subject Name: Design and Analysis of Algorithm

UID: 25MCI10304

Section/Group: 25MAM-5A

Date of Performance: 06-10-2025

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:

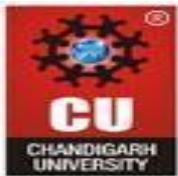
- Start the program.
- Input the number of vertices n .
- Read the adjacency matrix of the graph, where each element (i, j) represents the weight of the edge from vertex i to vertex j .
- Use 'inf' if there is no direct edge between two vertices.
- Initialize a distance matrix $dist$ with the same values 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:



- $\text{graph}[i][j]$ represents the weight of the edge from vertex i to vertex j .
- if there is no directed edge between i and j , set $\text{graph}[i][j] = \text{infinity}$

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

$$\text{dist}[i][j] = \text{graph}[i][j]$$

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

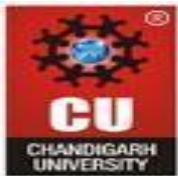
- For each vertex j from 0 to $n-1$:
- If both $\text{dist}[i][k]$ and $\text{dist}[k][j]$ are not infinity: $\text{dist}[i][j] = \min(\text{dist}[i][j], \text{dist}[i][k] + \text{dist}[k][j])$

Step 6: End of loops

- → Now, $\text{dist}[i][j]$ contains the shortest distance from vertex i to vertex j .

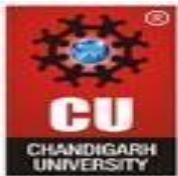
Step 7: Display the final shortest path matrix.

Step 8: Stop



4.Code for experiment/practical:

```
exp5.py > floyd_marshall_algo
1     inf = float('inf')
2
3     def floyd_marshall_algo(matrix):
4         n = len(matrix)
5         dist = [row[:] for row in matrix]
6         for k in range(n):
7             for i in range(n):
8                 for j in range(n):
9                     if dist[i][k] + dist[k][j] < dist[i][j]:
10                         dist[i][j] = dist[i][k] + dist[k][j]
11
12         print("\nFinal Shortest Path Matrix:")
13         for i in range(n):
14             for j in range(n):
15                 print(0 if dist[i][j] == inf else dist[i][j], end="\t")
16             print()
17
18 v = int(input("Enter total number of vertices: "))
19 print("Enter adjacency matrix (use 'inf' for no direct connection):")
20
21 graph = []
22 for i in range(v):
23     row_input = input().split()
24     if len(row_input) != v:
25         exit()
26     row = [inf if val.lower() == 'inf' else int(val) for val in row_input]
27     graph.append(row)
28
29 floyd_marshall_algo(graph)
30
```

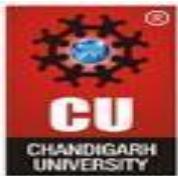


5.Output:

▼ TERMINAL

```
PS R:\Desktop\DAAs & C:/Python313/python.exe r:/Desktop/DAAs/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\DAAs █
```



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**.

-