| Autonomous Institute of Government of Maharashtra) | |
|---|---|
| **Name :** Shubham Pramod Badgujar | **PRN :** 2241004 |
| **Class :** T. Y. B.Tech Computer | **Academic Year :** 2024-25 |
| **Subject :** DAA Lab | **Course Teacher** : Mr. Vinit Kakde |
| **Date of Performance :** | **Date of Completion :** |

# PRACTICAL NO.

**AIM:**
Develop simulator for all pair shortest paths problem using Floyd's algorithm.

**Theory:**

**Floyd-Warshall Algorithm**

The Floyd-Warshall algorithm is a dynamic programming algorithm used to discover the shortest paths in a weighted graph, including graphs with negative edge weights (but no negative cycles). It computes the shortest paths between every pair of vertices in a graph by iteratively improving the distance estimates using intermediate vertices.

**History of Floyd-Warshall Algorithm:**

**The Floyd-Warshall algorithm was independently developed by Robert Floyd and Stephen Warshall in 1962.**

- Robert Floyd, a mathematician and computer scientist at IBM's Thomas J. Watson Research Center, initially presented the algorithm in a technical report titled *"Algorithm 97: Shortest Path"*.
- Stephen Warshall, from the University of California, Berkeley, independently discovered and published it in his technical paper *"A Theorem on Boolean Matrices"*.

The algorithm was initially applied to operations research, solving the all-pairs shortest path problem in directed graphs with both positive and negative weights.
Its efficiency and capability to handle a wide range of graphs have made it a cornerstone in areas such as transportation, logistics, networking, and optimization.

**Working of Floyd-Warshall Algorithm:**

**The algorithm proceeds with the following steps:**

1. Initialize a distance matrix D where D[i][j] represents the shortest distance from vertex i to vertex j.
2. Set the diagonal entries D[i][i] = 0 (distance from a node to itself).
3. For every edge (u, v) in the graph, update D[u][v] = weight(u, v).
4. For each intermediate vertex k, update the distances between all pairs (i, j) as:

**D[i][j] = min(D[i][j], D[i][k] + D[k][j])**

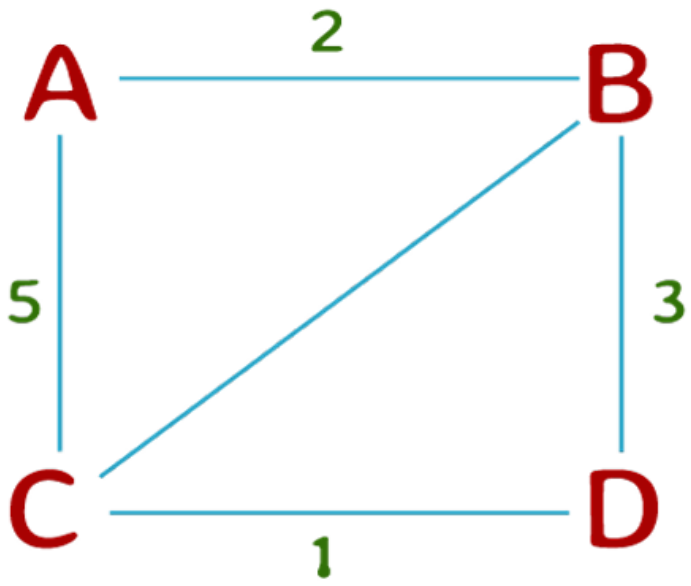
5. After all iterations, the matrix D contains the shortest distances between all pairs of vertices.

**Example:**

Floyd-Warshall is an algorithm used to locate the shortest course between all pairs of vertices in a weighted graph. It works by means of keeping a matrix of distances between each pair of vertices and updating this matrix iteratively till the shortest paths are discovered.

**Let's see at an example to illustrate how the Floyd-Warshall algorithm works:**

**Consider the following weighted graph:**




In this graph, the vertices are represented by letters (A, B, C, D), and the numbers on the edges represent the weights of those edges.

To follow the Floyd-Warshall algorithm to this graph, we start by way of initializing a matrix of distances among every pair of vertices. If two vertices are immediately related by using a side, their distance is the load of that edge. If there may be no direct edge among vertices, their distance is infinite.

| | A | B | C | D |
|---|---|---|---|---|
| **A** | 0 | 2 | inf | 5 |
| **B** | inf | 0 | 3 | 2 |
| **C** | 5 | inf | 0 | 1 |
| **D** | inf | inf | inf | 0 |

In the first iteration of the set of rules, we keep in mind the possibility of the usage of vertex 1 (A) as an intermediate vertex in paths among all pairs of vertices. If the space from vertex 1 to vertex 2 plus the space from vertex 2 to vertex three is much less than the present-day distance from vertex 1 to vertex three, then we replace the matrix with this new distance. We try this for each possible pair of vertices.

| | A | B | C | D |
|---|---|---|---|---|
| **A** | 0 | 2 | 5 | 4 |
| **B** | inf | 0 | 3 | 2 |
| **C** | 5 | 7 | 0 | 1 |
| **D** | inf | inf | inf | 0 |

In the second iteration, we recollect the possibility to use of vertex 2 (B) as an intermediate vertex in paths among all pairs of vertices. We replace the matrix in the same manner as earlier before.

| | A | B | C | D |
|---|---|---|---|---|
| **A** | 0 | 2 | 5 | 4 |
| **B** | inf | 0 | 3 | 2 |
| **C** | 5 | 7 | 0 | 1 |
| **D** | inf | 2 | 5 | 0 |

In the third iteration, we consider the possibility of using vertex 3 (C) as an intermediate vertex in paths between all pairs of vertices.

| | A | B | C | D |
|---|---|---|---|---|
| **A** | 0 | 2 | 5 | 4 |
| **B** | inf | 0 | 3 | 2 |
| **C** | 5 | 7 | 0 | 1 |
| **D** | 6 | 2 | 5 | 0 |

Finally, in the fourth and final iteration, we consider the possibility of using vertex 4 (D) as an intermediate vertex in paths between all pairs of vertices.

| | A | B | C | D |
|---|---|---|---|---|
| **A** | 0 | 2 | 5 | 4 |
| **B** | 7 | 0 | 3 | 2 |
| **C** | 5 | 7 | 0 | 1 |
| **D** | 6 | 2 | 5 | 0 |

After the fourth iteration, we have got the shortest path between every pair of vertices in the graph. For example, the shortest path from vertex A to vertex D is 4, which is the value in the matrix at row A and column D.

**Pseudocode:**

```
Floyd-Warshall(w, n){ // w: weights, n: number of vertices
  for i = 1 to n do // initialize, D (0) = [wij]
    for j = 1 to n do{
      d[i, j] = w[i, j];
    }
    for k = 1 to n do // Compute D (k) from D (k-1)
      for i = 1 to n do
        for j = 1 to n do
          if (d[i, k] + d[k, j] < d[i, j]){
            d[i, j] = d[i, k] + d[k, j];
          }
    return d[1..n, 1..n];}
```

**Advantages of the Floyd-Warshall Algorithm:**

1. It can discover the shortest path between all pairs of vertices in a weighted graph, including graphs with negative edge weights.
2. It is a simple and easy-to-implement algorithm, making it accessible to developers of all skill levels.
3. It is suitable for both dense and sparse graphs.
4. It has a time complexity of **$O(N^3)$**, which is relatively efficient for most real-world applications.
5. It can be used to detect negative weight cycles in a graph.

---

**Disadvantages of the Floyd-Warshall Algorithm:**

1. It requires a matrix of size **$N^2$** to store the intermediate results, which may be prohibitively large for very large graphs.
2. It is not the most efficient algorithm for solving the all-pairs shortest path problem in certain types of graphs, such as sparse graphs or graphs with only non-negative edge weights.
3. It may not be suitable for real-time applications or applications with strict memory constraints, as it can take a long time to compute the shortest paths in very large graphs.
4. It may be less intuitive than other algorithms, such as **Dijkstra's Algorithm** or the **Bellman-Ford Algorithm**, making it harder to understand for some developers.

---

**Applications of the Floyd-Warshall Algorithm:**

1. **Routing Algorithms:**
   Used in routing algorithms such as the **OSPF (Open Shortest Path First)** protocol in Internet routing. It helps determine the shortest path between two nodes and is useful for identifying the least congested route.
2. **Airline Networks:**
   Helps find the shortest and most cost-effective path between cities, aiding airlines in planning routes and minimizing fuel costs.
3. **Traffic Networks:**
   Used in urban traffic networks to compute the shortest routes between intersections or regions, helping reduce congestion and improve traffic flow.

4. **Computer Networks:**
   Helps determine the shortest path between hosts in a computer network, minimizing latency and improving overall performance.

5. **Game Development:**
   Useful in navigating characters or objects within game worlds, especially in pathfinding scenarios like mazes, cities, or obstacle-laden environments.

---

**Time Complexity:**

The time complexity of the Floyd-Warshall algorithm is **$O(n^3)$** due to three nested loops that iterate over the nodes of the graph. This means the runtime increases rapidly with graph size. While the algorithm is simple to implement, it is best suited for smaller graphs.

---

**Space Complexity:**

The space complexity is **$O(n^2)$** for storing the distance matrix D, where n is the number of vertices. It does not require additional data structures based on the number of edges, making it memory-efficient for moderately sized graphs.

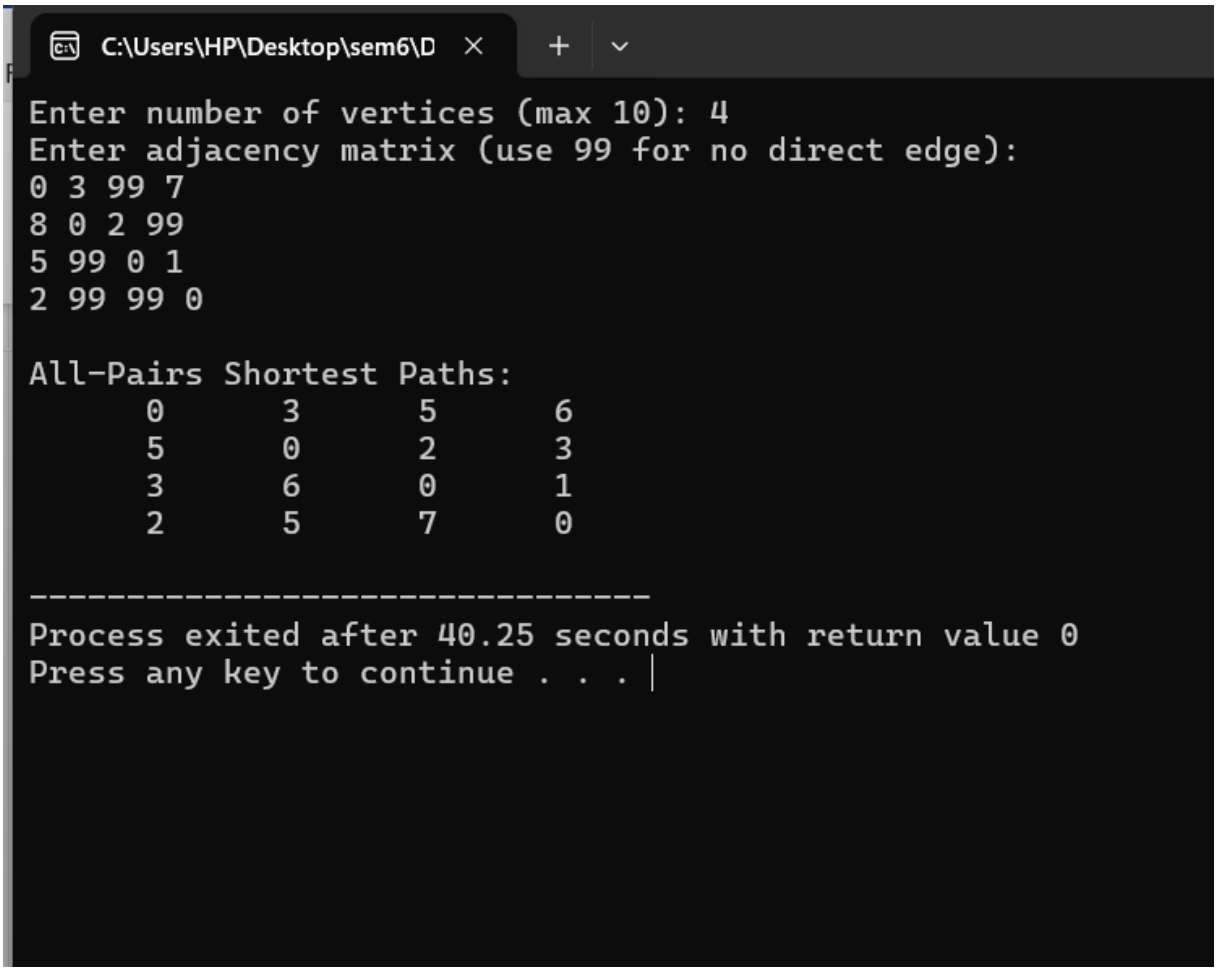
**PROGRAM:**

```
#include <stdio.h>

#define V 10     // Maximum number of vertices
#define INF 99   // Represents infinity for no direct path

// Function to perform Floyd-Warshall Algorithm
void floydWarshall(int graph[V][V], int n) {
   int dist[V][V];
   int i, j, k;

   // Initialize the distance matrix with input graph values
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         dist[i][j] = graph[i][j];

   // Floyd-Warshall algorithm: update shortest distances
```

```
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    // Display the final distance matrix
    printf("\nAll-Pairs Shortest Paths:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int n, i, j;
    int graph[V][V];

    // Input number of vertices
    printf("Enter number of vertices (max %d): ", V);
    scanf("%d", &n);

    // Input adjacency matrix
    printf("Enter adjacency matrix (use 99 for no direct edge):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    // Run Floyd-Warshall Algorithm
    floydWarshall(graph, n);
```

```
    return 0;
}
```

**#OUTPUT:**




**QUESTIONS:**

1. What is the purpose of the Floyd-Warshall algorithm?
   → To find the shortest paths between all pairs of vertices in a weighted graph.
2. Can Floyd-Warshall handle negative edge weights?
   → Yes, it can handle negative edge weights but not negative weight cycles.
3. What is the time complexity of Floyd-Warshall algorithm?
   → The time complexity is $O(n^3)$, where n is the number of vertices.
4. What data structure is used to store distances in Floyd-Warshall?
   → A 2D matrix (distance matrix) is used to store and update shortest paths.

5. **How does the algorithm detect a negative weight cycle?**
   → If dist[i][i] < 0 for any vertex i after completion, the graph contains a negative cycle.

---

## 📃 Conclusion:

In this practical, we successfully implemented the Floyd-Warshall algorithm to compute the shortest paths between all pairs of vertices in a graph. The algorithm is efficient and versatile for both dense and sparse graphs. It simplifies solving real-world problems involving routing, logistics, and networking by using dynamic programming. This practical helped us understand graph traversal techniques and matrix manipulation for optimization problems.

.

**Sign of course Teacher**

**Mr. Vinit Kakde**