Experiment No. : 5

Title: Floyd-Warshall Algorithm using Dynamic programming approach

Batch: A1 **Roll No.: 16010422013** **Experiment No.: 5**

Aim: To Implement All pair shortest path Floyd-Warshall Algorithm using Dynamic programming approach and analyse its time Complexity.

Algorithm of Floyd-Warshall Algorithm:

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4       let D^(k) = (d_ij^(k)) be a new n × n matrix
5       for i = 1 to n
6            for j = 1 to n
7                 d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

Constructing Shortest Path:

We can give a recursive formulation of $\pi_{ij}^{(k)}$. When $k = 0$, a shortest path from i to j has no intermediate vertices at all. Thus,

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases} \tag{25.6}$$

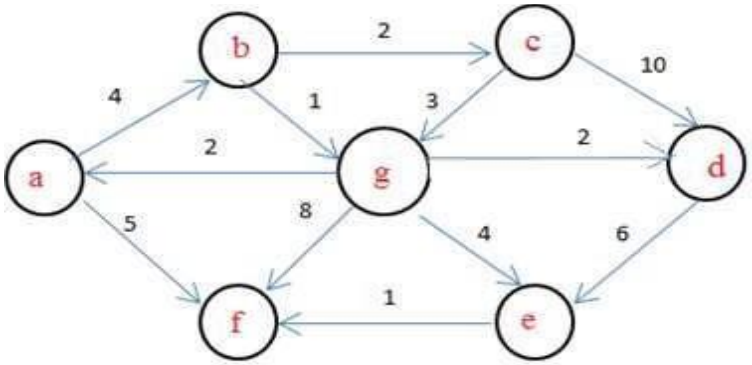
For $k \geq 1$, if we take the path $i \rightsquigarrow k \rightsquigarrow j$, where $k \neq j$, then the predecessor of j we choose is the same as the predecessor of j we chose on a shortest path from k with all intermediate vertices in the set $\{1, 2, \ldots, k-1\}$. Otherwise, we choose the same predecessor of j that we chose on a shortest path from i with all intermediate vertices in the set $\{1, 2, \ldots, k-1\}$. Formally, for $k \geq 1$,

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases} \tag{25.7}$$

Working of Floyd-Warshall Algorithm:

Problem Statement

Find Shortest Path for each source to all destinations using Floyd-Warshall Algorithm for the following graph



Solution:

```
Enter the number of vertices: 7
Enter the adjacency matrix (enter 'inf' for infinity):
0 4 inf inf inf 5 inf
inf 0 2 inf inf inf 1
inf inf 0 10 inf inf 3
inf inf inf 0 6 inf inf
inf inf inf inf 0 1 inf
inf inf inf inf inf 0 inf
2 inf inf 2 4 8 0
Shortest distances between every pair of vertices:
      0   1   2   3   4   5   6
 0    0 4.0 6.0 7.0 9.0 5.0 5.0
 1  3.0   0 2.0 3.0 5.0 6.0 1.0
 2  5.0 9.0   0 5.0 7.0 8.0 3.0
 3    ∞   ∞   ∞   0 6.0 7.0   ∞
 4    ∞   ∞   ∞   ∞   0 1.0   ∞
 5    ∞   ∞   ∞   ∞   ∞   0   ∞
 6  2.0 6.0 8.0 2.0 4.0 5.0   0
```

Derivation of Floyd-Warshall Algorithm:

Let dist[i][j] be the shortest distance from vertex i to vertex j. We initialize the dist matrix with the edge weights of the graph, setting dist[i][j] to the weight of the edge between i and j if there is an edge, or to infinity otherwise.

The algorithm then considers all vertices as intermediate vertices one by one and updates the shortest path if a shorter path is found through the current intermediate vertex. The key idea is that for each pair of vertices i and j, if there is an intermediate vertex k such that the path from i to j through k is shorter than the current shortest path, then update the dist[i][j] to the new shorter path dist[i][k] + dist[k][j].

Time complexity Analysis :

Initialization: Initializing the dist matrix takes O(V^2) time, where V is the number of vertices in the graph.

Main Algorithm:

The main algorithm consists of three nested loops, each iterating over all vertices in the graph. For each pair of vertices i and j, we check if going through each intermediate vertex k leads to a shorter path.

Since there are V vertices and we consider each pair of vertices, the total number of iterations in the nested loops is O(V^3).

Overall Complexity:
Thus, the overall time complexity of the Floyd-Warshall algorithm is O(V^2) (for initialization) + O(V^3) (for the main algorithm) = O(V^3), where V is the number of vertices in the graph.
Space Complexity:

The space complexity of the algorithm is O(V^2) for storing the dist matrix.

Program(s) of Floyd-Warshall Algorithm:

```
INF = float('inf')

def floyd_warshall(graph, V):
    dist = [[0 if i == j else graph[i][j] if graph[i][j] else INF for j in range(V)]
for i in range(V)]

    for k in range(V):
        for i in range(V):
            for j in range(V):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist

def print_table(dist, V):
    print("Shortest distances between every pair of vertices:")
    print("   ", end="")
    for i in range(V):
        print(f" {i}", end="")
    print()
    for i in range(V):
        print(f"{i:2} ", end="")
        for j in range(V):
            if dist[i][j] == INF:
                print("  ∞", end="")
            else:
                print(f"{dist[i][j]:3}", end="")
        print()

def main():
    V = int(input("Enter the number of vertices: "))
    graph = []
    print("Enter the adjacency matrix (enter 'inf' for infinity):")
    for i in range(V):
        row = input().split()
        row = [float('inf') if x == 'inf' else float(x) for x in row]
        graph.append(row)

    dist = floyd_warshall(graph, V)
    print_table(dist, V)

if __name__ == "__main__":
    main()
```

Output(o) of Floyd-Warshall Algorithm:

```
Enter the number of vertices: 5
Enter the adjacency matrix (enter 'inf' for infinity):
0 4 inf 5 inf
inf 0 1 inf 6
2 inf 0 3 inf
inf inf 1 0 2
1 inf inf 4 0
Shortest distances between every pair of vertices:
      0   1   2   3   4
 0    0 4.0 5.0 5.0 7.0
 1  3.0   0 1.0 4.0 6.0
 2  2.0 6.0   0 3.0 5.0
 3  3.0 7.0 1.0   0 2.0
 4  1.0 5.0 5.0 4.0   0
```

Post Lab Questions:-

Explain dynamic programming approach for Floyd-Warshall algorithm and write the various applications of it.

Ans)

The Floyd-Warshall algorithm is a dynamic programming approach used to find the shortest paths between all pairs of vertices in a weighted graph. The algorithm works for both directed and undirected graphs with positive or negative edge weights, but it does not work with graphs containing negative cycles.

Here's how the dynamic programming approach works for the Floyd-Warshall algorithm:

1. Initialize: Given an input graph represented by an adjacency matrix W, where W[i][j] represents the weight of the edge from vertex i to vertex j. Let n be the number of vertices in the graph.
2. Initialize the distance matrix D^(0) with the same values as the adjacency matrix W. This represents the shortest distances between pairs of vertices considering only direct edges.
3. For each vertex k from 1 to n:
    - Create a new distance matrix D^(k) of size nxn.
    - For each pair of vertices i and j:
        - Update the distance D[i][j] by comparing the previous distance D[i][j] with the distance obtained by going from vertex i to vertex j through vertex k. The update rule is: D[i][j] = min(D[i][j], D[i][k] + D[k][j]).
4. After iterating through all vertices as intermediates, the final distance matrix D^(n) will contain the shortest distances between all pairs of vertices in the graph.
5. Return the final distance matrix D^(n).

Algorithm FLOYD-WARSHALL (W) :

```
n = W.rows
D^(0)=W
For k= 1 to n
      let D^(k) = (dij^(k)) be a new nxn matrix
      for i= 1 to n
           for j=1 to n
                dij^(k)= = min(dij^(k-1), dik^(k-1) + dkj^(k-1))
return D^(n)
```

6. The time complexity of the Floyd-Warshall algorithm is O(n^3), where n is the number of vertices in the graph, making it efficient for small to medium-sized graphs.

Applications :

The Floyd-Warshall algorithm is a classic dynamic programming approach used to find the shortest paths between all pairs of vertices in a weighted graph. It's commonly used in real-world applications where you need to find the shortest path between all pairs of nodes, such as network routing algorithms, transportation networks, and even in some optimization problems.

1. Network Routing: In computer networking, routers use the Floyd-Warshall algorithm to compute the shortest paths between all pairs of nodes in a network. This helps in efficiently routing packets through the network to their destination.

2. Transportation Networks: In transportation planning, the algorithm can be used to find the shortest paths between all pairs of locations in a road network. This can help in optimizing traffic flow, minimizing travel time, and improving overall transportation efficiency.

3. Flight Routing: Airlines use the Floyd-Warshall algorithm to determine the shortest paths between all pairs of airports in their flight network. This helps in scheduling flights, optimizing routes, and minimizing fuel consumption.

4. Urban Planning: City planners use the algorithm to analyze transportation networks, public transit systems, and road infrastructure. By finding the shortest paths between all pairs of locations, they can identify areas with high traffic congestion, plan new roadways or public transportation routes, and improve overall urban mobility.

5. Robotics and Autonomous Vehicles: The algorithm can be used in robotics and autonomous vehicle navigation systems to compute the shortest paths between all pairs of locations in a given environment. This helps in planning collision-free paths and navigating efficiently through complex terrain.

These are just a few examples of real-world applications where the Floyd-Warshall algorithm can be applied using a dynamic programming approach. Its ability to efficiently find the shortest paths between all pairs of vertices in a graph makes it a powerful tool in various fields requiring optimization and pathfinding.

Conclusion: (Based on the observations):

Through this experiment, we learnt about Floyd Warshall algorithm to find the shortest path. We also learnt about its algorithm, working, derivation on time complexity and space complexity. After that we wrote a python program on the algorithm to sing the shortest path. We also solved a problem manually to verify the result. For post lab we learnt about Floyd Warshall in dynamic programming approach and its applications.

Outcome:

CO2: Implement Greedy and Dynamic Programming algorithms

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.