Experiment No. : 6

Title: Floyd-Warshall Algorithm using Dynamic programming approach

Batch: B2 Roll No.: 16010421119
Experiment No.: 6

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,

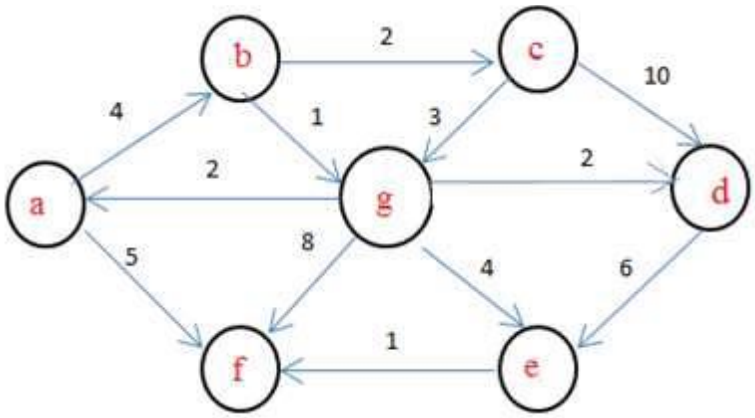
$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases} \tag{25.6}$$

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




```
PS C:\Users\Aarya\OneDrive\Desktop\SEM 4\Analysis of Algorithms (Lab)\EXP 6> cd 'c:\Users\Aarya\OneDrive\Desktop\SEM 4\Analysis of Algorithms (Lab)\EXP 6\output'
PS C:\Users\Aarya\OneDrive\Desktop\SEM 4\Analysis of Algorithms (Lab)\EXP 6\output> & .\'floyd.exe'
 Enter the weight of the edge between 1 and 1: 0
 Enter the weight of the edge between 1 and 2: 4
 Enter the weight of the edge between 1 and 3: 999
 Enter the weight of the edge between 1 and 4: 999
 Enter the weight of the edge between 1 and 5: 999
 Enter the weight of the edge between 1 and 6: 5
 Enter the weight of the edge between 1 and 7: 999
 Enter the weight of the edge between 2 and 1: 999
 Enter the weight of the edge between 2 and 2: 0
 Enter the weight of the edge between 2 and 3: 2
 Enter the weight of the edge between 2 and 4: 999
 Enter the weight of the edge between 2 and 5: 999
 Enter the weight of the edge between 2 and 6: 999
 Enter the weight of the edge between 6 and 3: 999
 Enter the weight of the edge between 6 and 4: 999
 Enter the weight of the edge between 6 and 5: 999
 Enter the weight of the edge between 6 and 6: 0
 Enter the weight of the edge between 6 and 7: 999
 Enter the weight of the edge between 7 and 1: 2
 Enter the weight of the edge between 7 and 2: 999
 Enter the weight of the edge between 7 and 3: 999
 Enter the weight of the edge between 7 and 4: 2
 Enter the weight of the edge between 7 and 5: 4
 Enter the weight of the edge between 7 and 6: 8
 Enter the weight of the edge between 7 and 7: 0
 0 4 6 7 9 5 5

 3 0 2 3 5 6 1

 5 9 0 5 7 8 3

 999 999 999 0 6 7 999

 999 999 999 999 0 1 999

 999 999 999 999 999 0 999

 2 6 8 2 4 5 0

PS C:\Users\Aarya\OneDrive\Desktop\SEM 4\Analysis of Algorithms (Lab)\EXP 6\output>
```

Derivation of Floyd-Warshall Algorithm:

- Time complexity Analysis

There are three loops. Each loop has constant complexities. So, the time complexity of the Floyd-Warshall algorithm is $O(n^3)$.

- Space complexity Analysis

The space complexity of the Floyd-Warshall algorithm is $O(n^2)$.

Program(s) of Floyd-Warshall Algorithm:

```
#include <iostream>
using namespace std;
#define size 4
// #define INF 999

void printMatrix(int matrix[][size]);

void floydWarshall(int graph[][size])
{
    int matrix[size][size], i, j, k;
    // for (i = 0; i < size; i++)
    //     for (j = 0; j < size; j++)
    //         matrix[i][j] = graph[i][j];
    for (k = 0; k < size; k++)
    {
        for (i = 0; i < size; i++)
        {
            for (j = 0; j < size; j++)
            {
                if (matrix[i][k] + matrix[k][j] <
matrix[i][j])
                    matrix[i][j] = matrix[i][k] +
matrix[k][j];
            }
        }
    }
    printMatrix(matrix);
}

void printMatrix(int matrix[][size])
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            // if (matrix[i][j] == INF)
            //     printf("%4s", "INF");
            // else
            cout<< matrix[i][j]<<" ";
        }
        cout<<endl;
        printf("\n");
    }
```

```
}

int main()
{
    int graph[size][size];
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            cout << "Enter the weight of the edge
between " << i+1 << " and " << j+1 << ": ";
            cin >> graph[i][j];
        }
    }

    floydWarshall(graph);
}
```

Output(o) of Floyd-Warshall Algorithm:

Post Lab Questions:-

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

Ans: Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).

Applications of the Floyd Warshall Algorithm

- Fast computation of Pathfinder networks.
- Inversion of real matrices.
- The transitive closure of directed graphs.
- Checking if an undirected graph is bipartite or not.
- Shortest path in a graph.

Conclusion: (Based on the observations):

We can conclude that we have learnt about Floyd-Warshall Algorithm..

Outcome:

CO3 :- Implement Backtracking and Branch-and-bound 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.