Batch: A1	**Roll No.: 16010123012**
Experiment No.: 7	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation and Analysis of All Pair Shortest Path.

Objective To learn the All-Pair Shortest Path using Floyd-Warshall's algorithm

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Saratej Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://users.cecs.anu.edu.au/~Alistair.Rendell/Teaching/apac_comp3600/module4/all_pairs_shortest_paths.xhtml**
4. **https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/**
5. **http://www.cs.bilkent.edu.tr/~atat/502/AllPairsSP.ppt**

Theory:

It aims to figure out the shortest path from each vertex v to every other u.

1. In all pair shortest path, when a weighted graph is represented by its weight matrix W then objective is to find the distance between every pair of nodes.
2. Apply dynamic programming to solve the all pairs shortest path.
3. In all pair shortest path algorithm, we first decomposed the given problem into sub problems.
4. In this principle of optimally is used for solving the problem.
5. It means any sub path of shortest path is a shortest path between the end nodes.

Algorithm:

```
Algorithm All_pair(W, A)
{
For i = 1 to n do
For j = 1 to n do
A [i , j] = W [i , j]
For k = 1 to n do
    {
    For i = 1 to n do
        {
        For j = 1 to n do
        {
        A [i , j] = min(A [i, j], A [i, k] + A [k, j])
        }
        }}}
```

Code:

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

void allPair(vector<vector<int>> &A)
{
  int n = A.size();

  for (int k = 0; k < n; k++)
  {
    for (int i = 0; i < n; i++)
    {
      for (int j = 0; j < n; j++)
      {
        if (A[i][k] == -1 || A[k][j] == -1)
        {
          continue;
        }

        if (A[i][j] == -1 || A[i][j] > A[i][k] + A[k][j])
        {
          A[i][j] = A[i][k] + A[k][j];
        }
      }
    }
  }
```

```
}

void printMatrix(const vector<vector<int>> &A)
{
  int n = A.size();
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      if (A[i][j] == -1)
      {
        cout << "INF ";
      }
      else
      {
        cout << A[i][j] << "    ";
      }
    }
    cout << endl;
  }
}

int main()
{
  vector<vector<int>> A = {
      {0, 3, -1, 7},
      {8, 0, 2, -1},
      {5, -1, 0, 1},
      {2, -1, -1, 0}};

  cout << "Initial Adjacency Matrix:" << endl;
  printMatrix(A);

  allPair(A);

  cout << "\nAll-Pairs Shortest Path Matrix:" << endl;
  printMatrix(A);

  return 0;
}
```

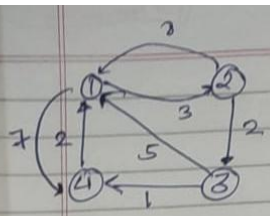
Output:

```
Initial Adjacency Matrix:
0    3    INF 7
8    0    2    INF
5    INF 0    1
2    INF INF 0

All-Pairs Shortest Path Matrix:
0    3    5    6
5    0    2    3
3    6    0    1
2    5    7    0
```

Analysis of algorithm:
Time Complexity: $O(n^3)$
Space Complexity: $O(n^2)$

Example / Solution for the example:



	1	2	3	4
1	0	3	∞	7
2	8	0	2	∞
3	5	∞	0	1
4	2	∞	∞	0

K=1

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	5	8	0	1
4	2	5	∞	0

$(A^0[2,3], A^0[2,1] + A^0[1,3])$
$2 < 8+\infty$
$(A^0[2,4], A^0[2,1] + A^0[1,4])$
$\infty > 8+7$

K=2

	1	2	3	4
1	0	3	5	7
2	8	0	2	15
3	5	8	0	1
4	2	5	7	0

$(A'[1,3], A'[1,2] + A'[2,3])$
$\infty > 3+2$
$(A'[1,4], A'[1,2] + A'[2,4])$
$7 < 3+15$

K=3

	1	2	3	4
1	0	3	5	6
2	7	0	2	3
3	5	8	0	1
4	2	5	7	0

$(A^2[1,2], A^2[1,3] + A^2[3,2])$
$3 < 5+8$
$(A^2[1,4], A^2[1,3] + A^2[3,4])$
$7 > 5+1$

K=4

	1	2	3	4
1	0	3	5	6
2	5	0	2	3
3	3	6	0	1
4	2	5	7	0

$(A^3[3,1], A^3[3,4] + A^3[4,1])$
$5 > 1+2$
$(A^3[3,2], A^3[3,4] + A^3[2,1])$
$8 > 1+6$

CONCLUSION:

I have completed the study, implementation, and analysis of the All-Pairs Shortest Path problem using the Floyd-Warshall algorithm. Through this experiment, I learned how dynamic programming can be applied to solve shortest path problems efficiently. The time complexity of the Floyd-Warshall algorithm is $O(n^3)$, making it suitable for small to medium-sized graphs. The space complexity is $O(n^2)$ due to the adjacency matrix representation. By applying the principle of optimal substructure, I observed that each shortest path consists of optimal subpaths, ensuring accurate results. This experiment enhanced my understanding of graph algorithms, dynamic programming and real-world applications such as network routing and navigation systems.