Experiment No. : 6

Title: Floyd-Warshall Algorithm using Dynamic programming approach

Batch: A2 **Roll No.: 1601421063** **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 $n \times n$ 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,

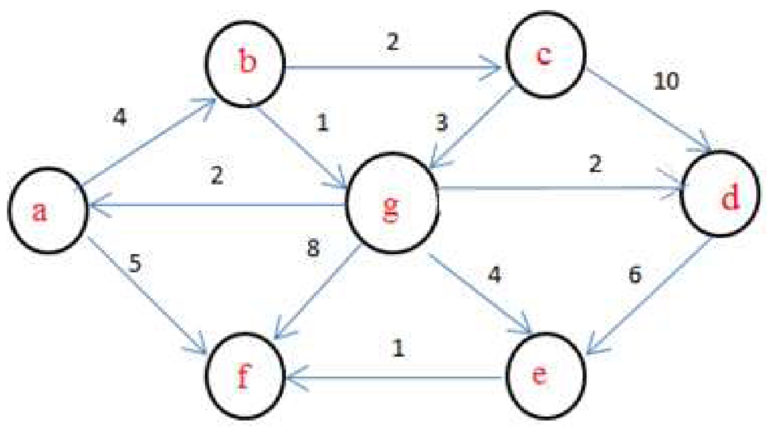
$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty , \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty . \end{cases} \qquad (25.6)$$

For $k \geq 1$, if we take the path $i \leadsto k \leadsto j$, 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 $k \geq 1$,

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

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
Consider the vertices of G as V = 1, 2........n and a subset of vertices as 1, 2........k for some k. Consider all pathways from I to j whose intermediate vertices are all drawn from 1, 2.......k, and let p be the lowest weight path among them for every pair of vertices i, j ∈ V,. The Floyd-Warshall algorithm uses a link between path p and all intermediate vertices in the set 1, 2.......k-1 to find the shortest path from i to j. Whether or not k is an intermediary vertex of path p determines the link.
If k is not an intermediate vertex of path p, then all intermediate vertices of path p are in the set {1, 2........k-1}. Thus, the shortest path from vertex i to vertex j with all intermediate vertices in the set {1, 2.......k-1} is also the shortest path i to j with all intermediate vertices in the set {1, 2.......k}.
If k is an intermediate vertex of path p, then we break p down into i → k → j.
Let dij(k) be the weight of the shortest path from vertex i to vertex j with all intermediate vertices in the set {1, 2.......k}.
A recursive definition is given by

$$d_{ij}^{(k)} = \begin{cases} W_{ij} & \text{if} \quad k=0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} & \text{if} \quad k \geq 1) \end{cases}$$

K. J. SOMAIYA COLLEGE OF ENGG.

Derivation of Floyd-Warshall Algorithm:

Time complexity Analysis
There are three loops in this game. Each loop has a different level of complexity. As a result, the Floyd-Warshall method has an O time complexity (n3).

Program(s) of Floyd-Warshall Algorithm:

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
```

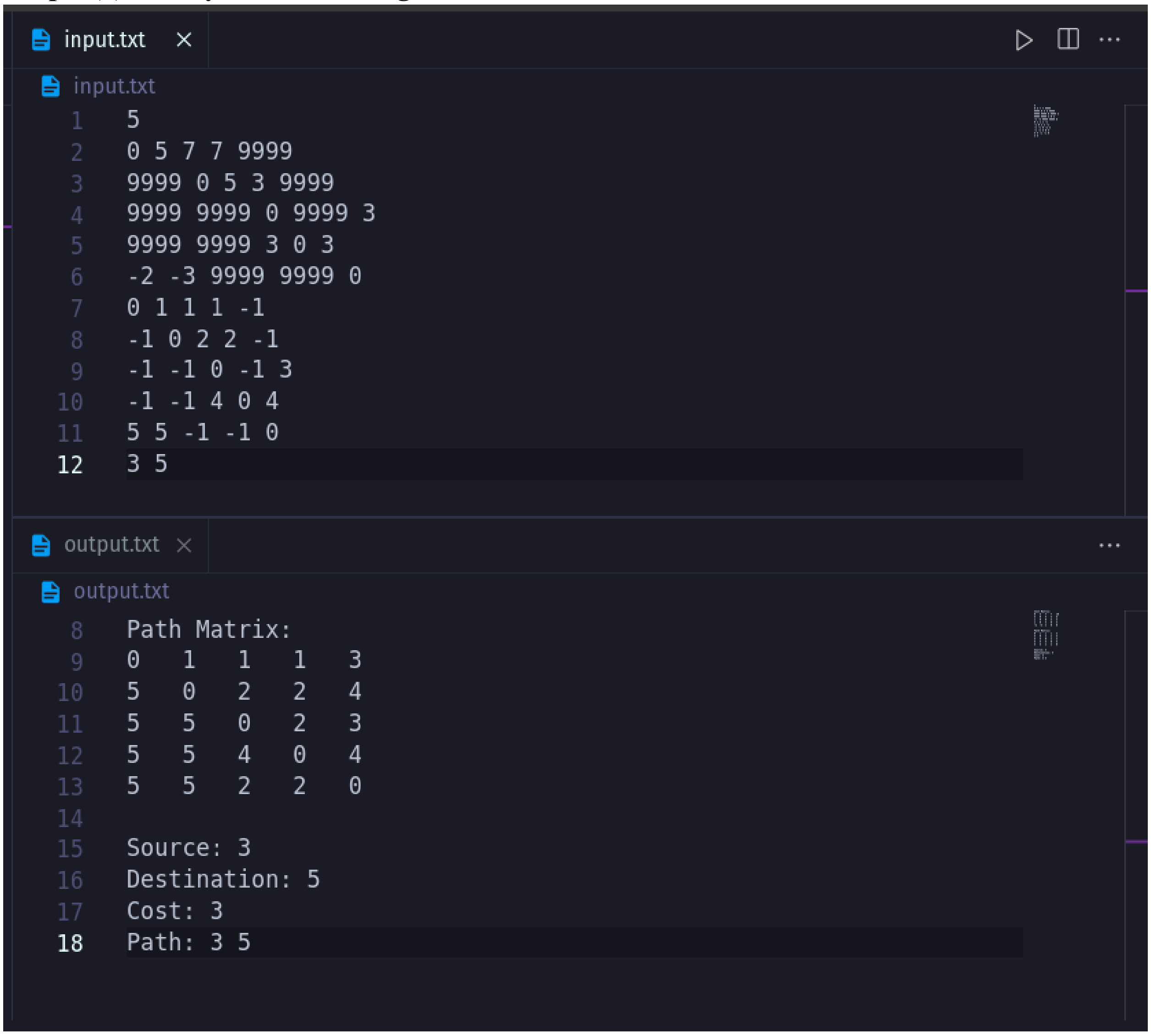
```
void floydWarshall(vector<vector<int>> &w, vector<vector<int>> &pi)
{
   int n = w.size();
   vector<vector<int>> matrix = w;
   for (int k = 0; k < n; k++)
   {
       vector<vector<int>> D = matrix;
       for (int i = 0; i < n; i++)
       {
           for (int j = 0; j < n; j++)
           {
               if (matrix[i][j] > matrix[i][k] + matrix[k][j])
               {
                   D[i][j] = min(matrix[i][j], matrix[i][k] +
matrix[k][j]);
                   pi[i][j] = pi[k][j];
               }
           }
       }
       matrix = D;
   }
   w = matrix;
}

int32_t main()
{
   ios_base::sync_with_stdio(false);
   cin.tie(NULL);
#ifndef ONLINE_JUDGE
   freopen("input.txt", "r", stdin);
   freopen("output.txt", "w", stdout);
#endif
   int n;
   cin >> n;
   vector<vector<int>> cost(n, vector<int>(n, LONG_LONG_MAX));
   vector<vector<int>> path(n, vector<int>(n, -1));
   for (int i = 0; i < n; i++)
   {
       for (int j = 0; j < n; j++)
       {
           cin >> cost[i][j];
       }
   }
```

```
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> path[i][j];
        }
    }
    floydWarshall(cost, path);
    cout << "Cost Matrix:\n";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << cost[i][j] << "\t";
        }
        cout << "\n";
    }
    cout << "\nPath Matrix:\n";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << path[i][j] << "\t";
        }
        cout << "\n";
    }
    int source = 0, destination = 0;
    cin >> source >> destination;
    cout << "\nSource: " << source;
    cout << "\nDestination: " << destination;
    source -= 1;
    destination -= 1;

    cout << "\nCost: " << cost[source][destination];
    vector<int> v;
    int i = destination;
    v.push_back(i + 1);
    while (i != source+1)
    {
        // cout<<path[source][i]<<"\n";
        v.push_back(path[source][i]);
        // s.push(path[source][i]);
        i = path[source][i];
    }
    cout << "\nPath: ";
    for (int k = v.size() - 1; k >= 0; k--)
    {
        cout << v[k] << " ";
    }
}
```

Output(o) of Floyd-Warshall Algorithm:



```
5
0 5 7 7 9999
9999 0 5 3 9999
9999 9999 0 9999 3
9999 9999 3 0 3
-2 -3 9999 9999 0
0 1 1 1 -1
-1 0 2 2 -1
-1 -1 0 -1 3
-1 -1 4 0 4
5 5 -1 -1 0
3 5
```

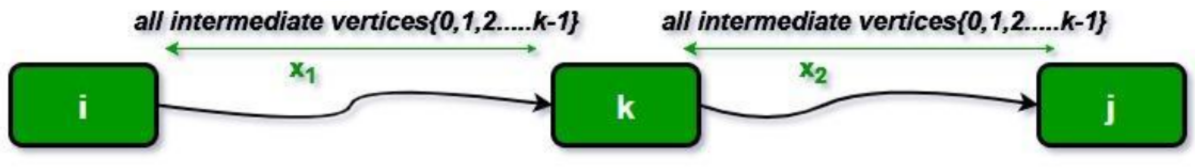
```
Path Matrix:
0   1   1   1   3
5   0   2   2   4
5   5   0   2   3
5   5   4   0   4
5   5   2   2   0

Source: 3
Destination: 5
Cost: 3
Path: 3 5
```

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

As a preliminary step, we initialise the solution matrix to the same values as the input graph

matrix. The solution matrix is then updated by treating all vertices as intermediate vertex. The goal is to pick all vertices one by one and update all shortest paths that involve the picked vertex as an intermediate vertex. We already treated vertices 0, 1, 2,... k-1 as intermediate vertices when we choose vertex number k as an intermediate vertex. There are two alternative scenarios for each pair of source and destination vertices (i, j).
1) In the shortest path from I to j, k is not an intermediate vertex. We don't change the value of dist[i][j].
2) In the shortest path from I to j, k is an intermediate vertex. The value of dist[i][j] is updated to dist[i][k] + dist[k][j]. if dist[i][j] is greater than dist[i][k] + dist[k][j],
In the all-pairs shortest path issue, the above optimal substructure property is shown in the diagram below.


all intermediate vertices{0,1,2.....k-1}
all intermediate vertices{0,1,2.....k-1}
x1
x2
i
k
j

Conclusion: (Based on the observations):
Floyd-Warshall algorithm was implemented and its time complexity was analysed.

Outcome:
CO-2: Implement Greedy and Dynamic Programming algorithms

K. J. SOMAIYA COLLEGE OF ENGG.

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.