Experiment No. : 6

Title: Floyd-Warshall Algorithm using Dynamic programming approach

Batch: B1 **Roll No.:16010420133** **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 $D^{(k)} = \left(d_{ij}^{(k)}\right)$ be a new $n \times n$ matrix
5      for $i = 1$ to $n$
6          for $j = 1$ to $n$
7              $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
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,

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

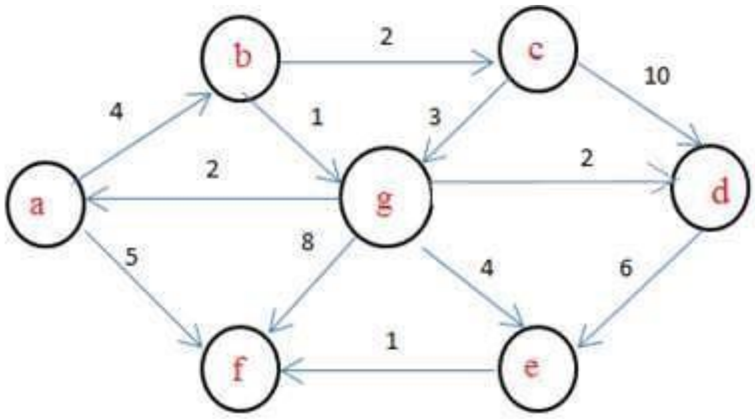
For $k \geq 1$, if we take the path $i \rightsquigarrow k \rightsquigarrow j$, where $k \neq j$, then the predecessor of j we choose is the same as the predecessor of j we chose on a shortest path from k with all intermediate vertices in the set $\{1, 2, \ldots, k-1\}$. Otherwise, we choose the same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

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

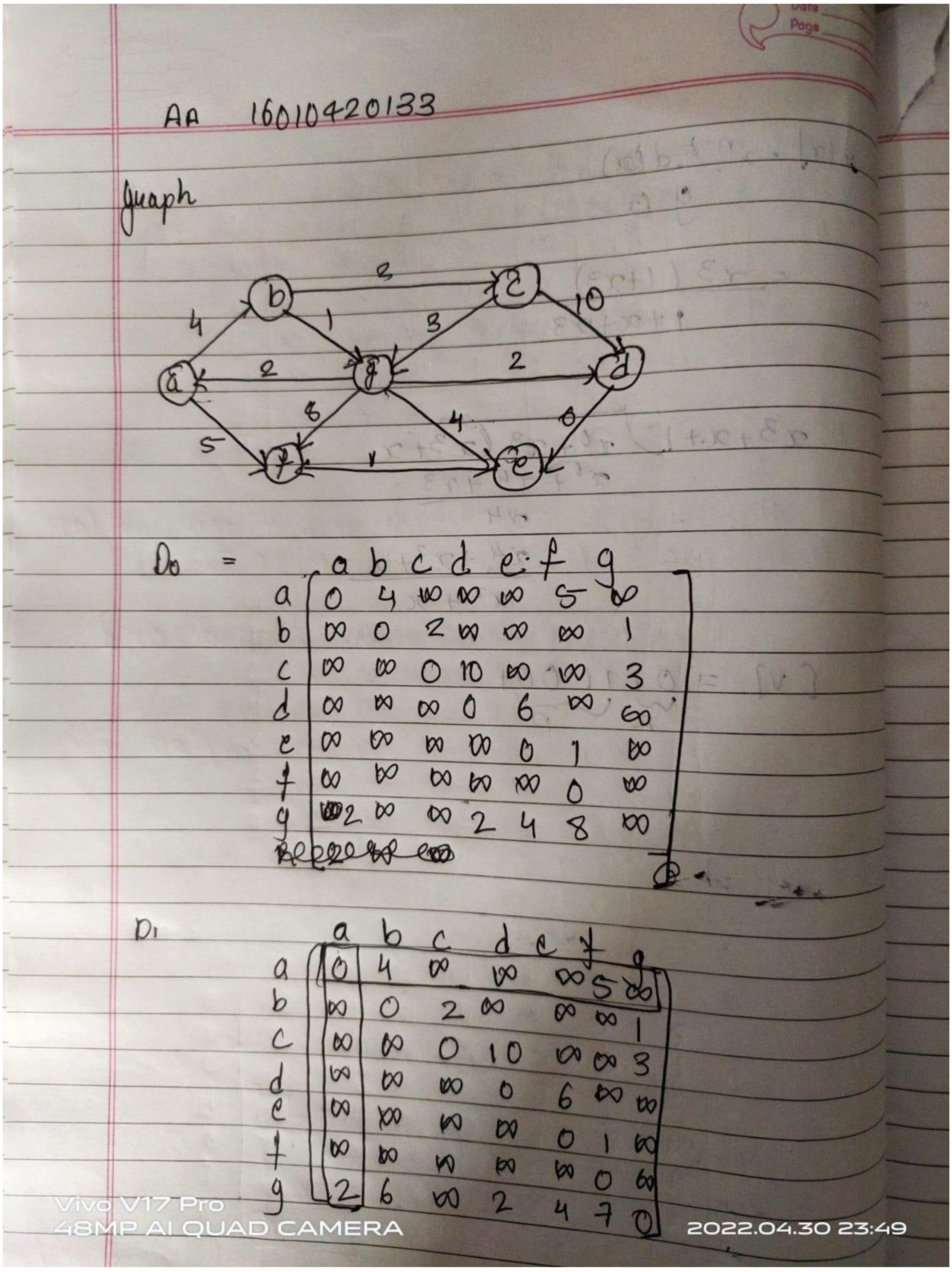
Working of Floyd-Warshall Algorithm:

Problem Statement

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

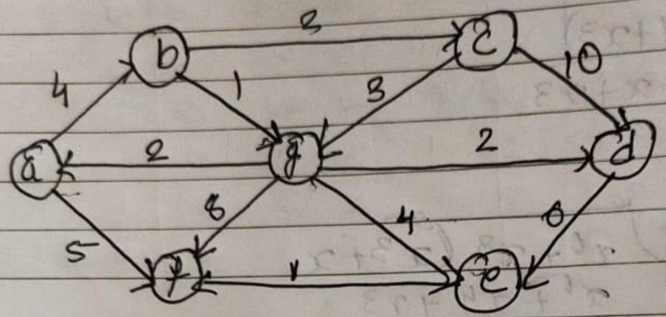


Solution



AA 16010420133

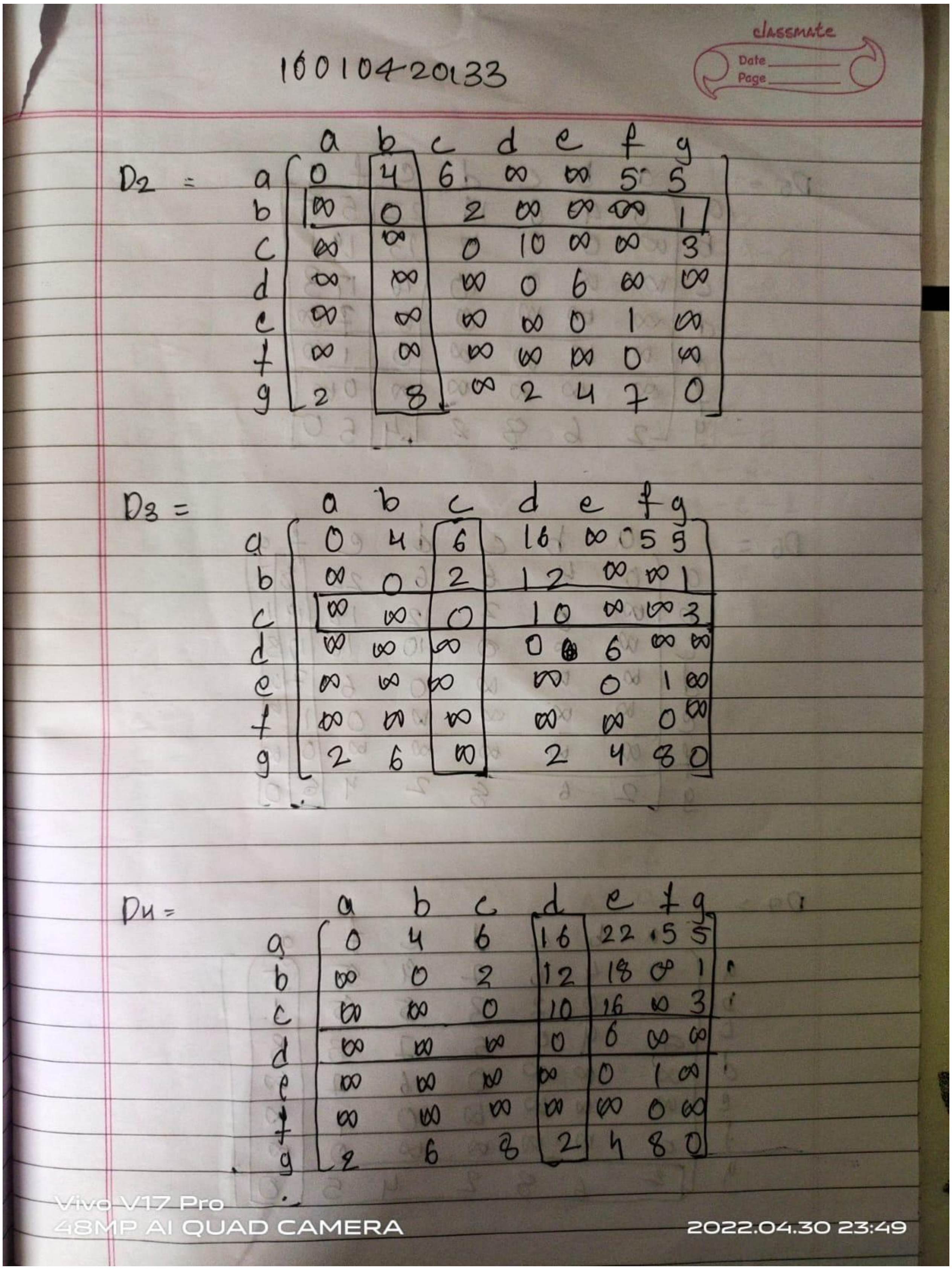
Graph



$$D_0 = \begin{array}{c|ccccccc} & a & b & c & d & e & f & g \\ \hline a & 0 & 4 & \infty & \infty & \infty & 5 & \infty \\ b & \infty & 0 & 2 & \infty & \infty & \infty & 1 \\ c & \infty & \infty & 0 & 10 & \infty & \infty & 3 \\ d & \infty & \infty & \infty & 0 & 6 & \infty & \infty \\ e & \infty & \infty & \infty & \infty & 0 & 1 & \infty \\ f & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ g & 2 & \infty & \infty & 2 & 4 & 8 & \infty \end{array}$$

$$D_1 \quad \begin{array}{c|ccccccc} & a & b & c & d & e & f & g \\ \hline a & 0 & 4 & \infty & \infty & \infty & 5 & \infty \\ b & \infty & 0 & 2 & \infty & \infty & \infty & 1 \\ c & \infty & \infty & 0 & 10 & \infty & \infty & 3 \\ d & \infty & \infty & \infty & 0 & 6 & \infty & \infty \\ e & \infty & \infty & \infty & \infty & 0 & 1 & \infty \\ f & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ g & 2 & 6 & \infty & 2 & 4 & 7 & 0 \end{array}$$

Vivo V17 Pro
48MP AI QUAD CAMERA 2022.04.30 23:49

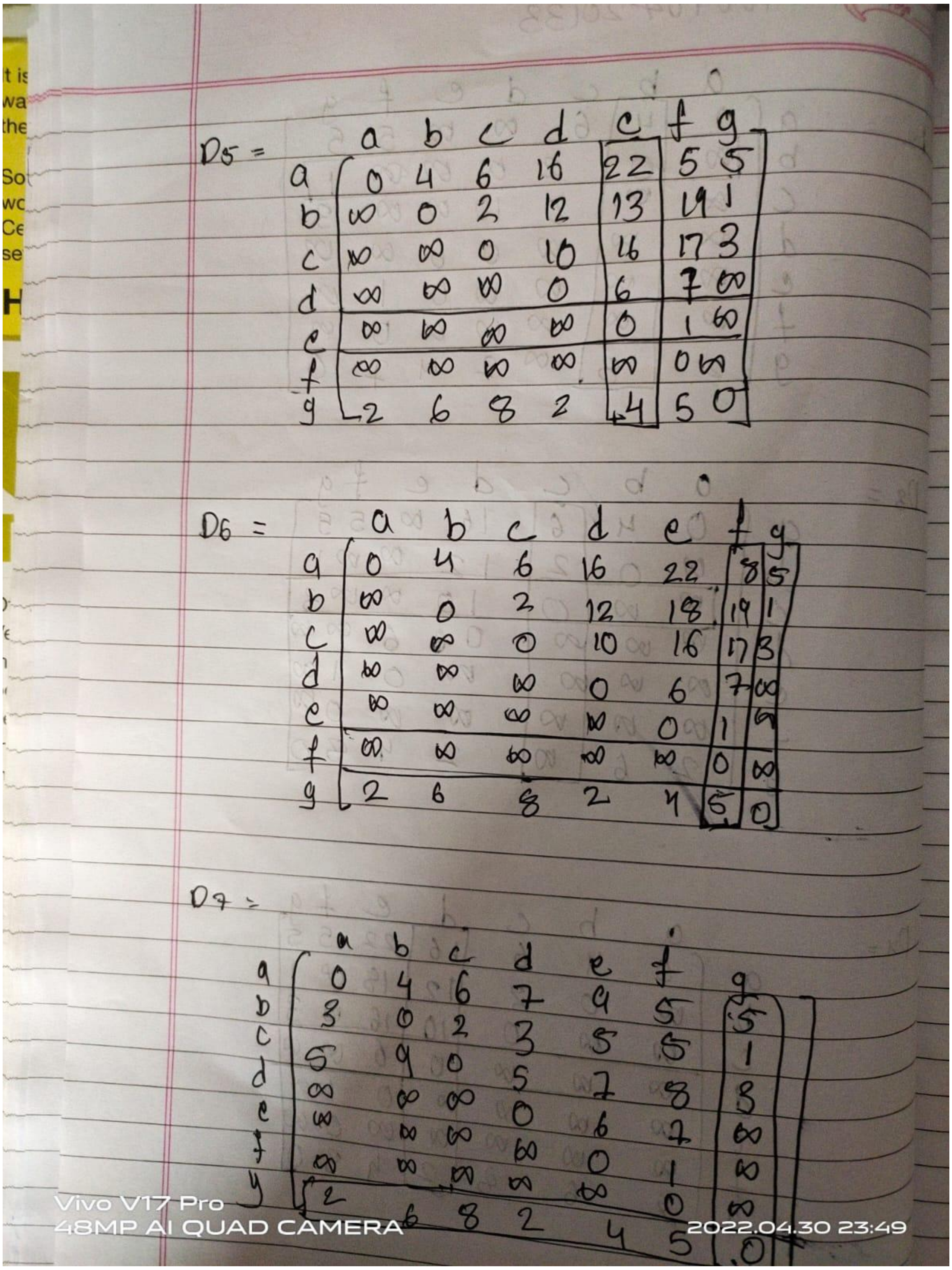


16010420133

$$D_2 = \begin{array}{c|ccccccc} & a & b & c & d & e & f & g \\ \hline a & 0 & 4 & 6 & \infty & \infty & 5 & 5 \\ b & \infty & 0 & 2 & \infty & \infty & \infty & 1 \\ c & \infty & \infty & 0 & 10 & \infty & \infty & 3 \\ d & \infty & \infty & \infty & 0 & 6 & \infty & \infty \\ e & \infty & \infty & \infty & \infty & 0 & 1 & \infty \\ f & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ g & 2 & 8 & \infty & 2 & 4 & 7 & 0 \end{array}$$

$$D_3 = \begin{array}{c|ccccccc} & a & b & c & d & e & f & g \\ \hline a & 0 & 4 & 6 & 16 & \infty & 5 & 5 \\ b & \infty & 0 & 2 & 12 & \infty & \infty & 1 \\ c & \infty & \infty & 0 & 10 & \infty & \infty & 3 \\ d & \infty & \infty & \infty & 0 & 6 & \infty & \infty \\ e & \infty & \infty & \infty & \infty & 0 & 1 & \infty \\ f & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ g & 2 & 6 & \infty & 2 & 4 & 8 & 0 \end{array}$$

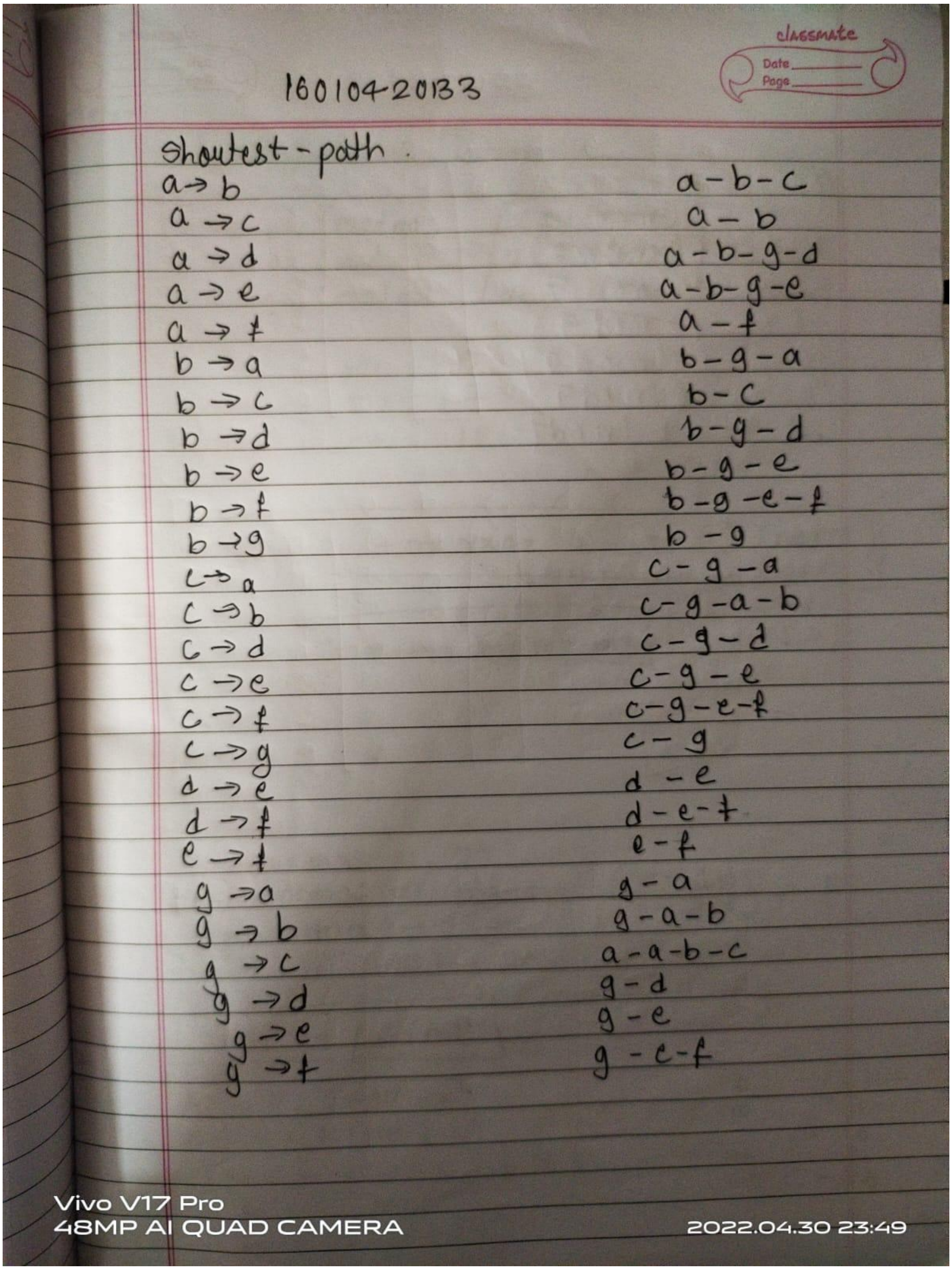
$$D_4 = \begin{array}{c|ccccccc} & a & b & c & d & e & f & g \\ \hline a & 0 & 4 & 6 & 16 & 22 & 5 & 5 \\ b & \infty & 0 & 2 & 12 & 18 & \infty & 1 \\ c & \infty & \infty & 0 & 10 & 16 & \infty & 3 \\ d & \infty & \infty & \infty & 0 & 6 & \infty & \infty \\ e & \infty & \infty & \infty & \infty & 0 & 1 & \infty \\ f & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ g & 2 & 6 & 8 & 2 & 4 & 8 & 0 \end{array}$$



$$D_5 = \begin{array}{c|ccccccc} & a & b & c & d & e & f & g \\ \hline a & 0 & 4 & 6 & 16 & 22 & 5 & 5 \\ b & \infty & 0 & 2 & 12 & 13 & 19 & 1 \\ c & \infty & \infty & 0 & 10 & 16 & 17 & 3 \\ d & \infty & \infty & \infty & 0 & 6 & 7 & \infty \\ e & \infty & \infty & \infty & \infty & 0 & 1 & \infty \\ f & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ g & 2 & 6 & 8 & 2 & 4 & 5 & 0 \end{array}$$

$$D_6 = \begin{array}{c|ccccccc} & a & b & c & d & e & f & g \\ \hline a & 0 & 4 & 6 & 16 & 22 & 8 & 5 \\ b & \infty & 0 & 2 & 12 & 18 & 19 & 1 \\ c & \infty & \infty & 0 & 10 & 16 & 17 & 3 \\ d & \infty & \infty & \infty & 0 & 6 & 7 & \infty \\ e & \infty & \infty & \infty & \infty & 0 & 1 & 9 \\ f & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ g & 2 & 6 & 8 & 2 & 4 & 5 & 0 \end{array}$$

$$D_7 = \begin{array}{c|ccccccc} & a & b & c & d & e & f & g \\ \hline a & 0 & 4 & 6 & 7 & 9 & 5 & 5 \\ b & 3 & 0 & 2 & 3 & 5 & 5 & 1 \\ c & 5 & 9 & 0 & 5 & 7 & 8 & 3 \\ d & \infty & \infty & \infty & 0 & 6 & 7 & \infty \\ e & \infty & \infty & \infty & \infty & 0 & 1 & \infty \\ f & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ g & 2 & 6 & 8 & 2 & 4 & 5 & 0 \end{array}$$


classmate
Date
Page
Vivo V17 Pro
48MP AI QUAD CAMERA
2022.04.30 23:49

16010420133

Shortest-path.

$a \to b$	a-b-c
$a \to c$	a-b
$a \to d$	a-b-g-d
$a \to e$	a-b-g-e
$a \to f$	a-f
$b \to a$	b-g-a
$b \to c$	b-c
$b \to d$	b-g-d
$b \to e$	b-g-e
$b \to f$	b-g-e-f
$b \to g$	b-g
$c \to a$	c-g-a
$c \to b$	c-g-a-b
$c \to d$	c-g-d
$c \to e$	c-g-e
$c \to f$	c-g-e-f
$c \to g$	c-g
$d \to e$	d-e
$d \to f$	d-e-f
$e \to f$	e-f
$g \to a$	g-a
$g \to b$	g-a-b
$g \to c$	a-a-b-c
$g \to d$	g-d
$g \to e$	g-e
$g \to f$	g-e-f

Derivation of Floyd-Warshall Algorithm:

Time complexity Analysis

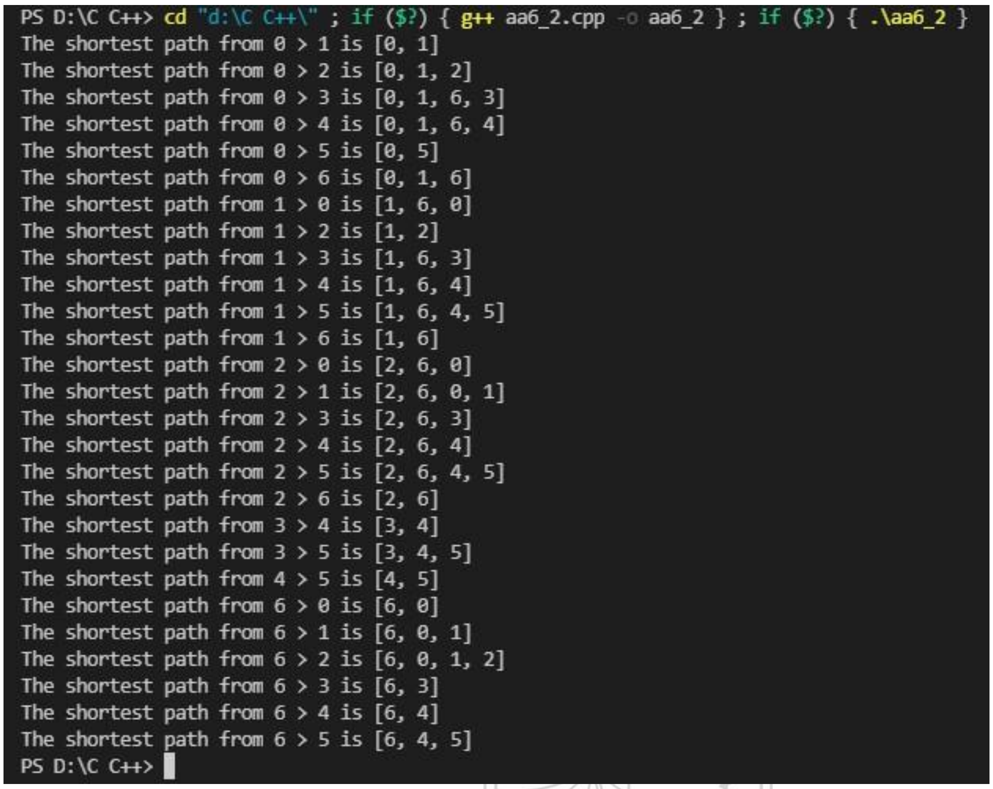
The running time of the Floyd Warshall algorithm is determined by the triply nested for loops. Since each execution of the for loop takes O(1) time, the algorithm runs in time Theta(V3). It is a dynamic programming algorithm with O(|V|3) time complexity and O(|V|2) space complexity.

Program(s) of Floyd-Warshall Algorithm:

```
#include <iostream>
#include <vector>
#include   <climits>
#include   <iomanip>
using namespace std;
   void printPath(vector<vector<int>> const &path, int v, int
u) {     if (path[v][u] == v) {         return;
    }     printPath(path, v,
path[v][u]);     cout << path[v][u]
<< ", ";
}    void printSolution(vector<vector<int>> const &cost, vector<vector<int>>
const &path) {     int n = cost.size();     for (int v = 0; v < n; v++)
    {         for (int u = 0; u <
n; u++)
        {             if (u != v &&
path[v][u] != -1)
            {                 cout << "The shortest path from " << v << " > "
<< u << " is ["
                    << v << ", ";
printPath(path, v, u);                 cout
<< u << "]" << endl;
            }
        }
    }
}   void floydWarshall(vector<vector<int>> const
&adjMatrix)
{     int n =
adjMatrix.size();    if (n ==
0) {         return;
    }      vector<vector<int>> cost(n,
vector<int>(n));     vector<vector<int>> path(n,
vector<int>(n));      for (int v = 0; v < n;
v++)
    {         for (int u = 0; u <
n; u++)
```

```
        {            cost[v][u] =
adjMatrix[v][u];
              if  (v  ==  u)  {
path[v][u] = 0;
            }            else if (cost[v][u]
!= INT_MAX) {                path[v][u] = v;
            }
            else {
                path[v][u] = -1;
            }
        }     }     for (int k
= 0; k < n; k++)
    {         for (int v = 0; v <
n; v++)
        {             for (int u = 0; u
< n; u++)
            {                if (cost[v][k] != INT_MAX &&
cost[k][u] != INT_MAX
                    && cost[v][k] + cost[k][u] < cost[v][u])
                {                   cost[v][u] =
cost[v][k] + cost[k][u];               path[v][u]
= path[k][u];
                }             }
if (cost[v][v] < 0)
            {                cout << "Negative-weight
cycle found!!";                 return;
            }
        }     }
printSolution(cost, path);
}   int main() {
int I = INT_MAX;
    vector<vector<int>> adjMatrix =
    {
        { 0, 4, I, I , I, 5, I },
{ I, 0, 2, I, I, I, 1},
        { I, I, 0, 10, I, I, 3 },
        { I, I, I, 0, 6, I, I},
        { I, I, I, I, 0, 1, I},
        {I, I, I, I, I, 0, I},
        {2, I, I, 2, 4 , 8, 0}
    };
floydWarshall(adjMatrix);
return 0;
}
```

Output(o) of Floyd-Warshall Algorithm:



```
PS D:\C C++> cd "d:\C C++\" ; if ($?) { g++ aa6_2.cpp -o aa6_2 } ; if ($?) { .\aa6_2 }
The shortest path from 0 > 1 is [0, 1]
The shortest path from 0 > 2 is [0, 1, 2]
The shortest path from 0 > 3 is [0, 1, 6, 3]
The shortest path from 0 > 4 is [0, 1, 6, 4]
The shortest path from 0 > 5 is [0, 5]
The shortest path from 0 > 6 is [0, 1, 6]
The shortest path from 1 > 0 is [1, 6, 0]
The shortest path from 1 > 2 is [1, 2]
The shortest path from 1 > 3 is [1, 6, 3]
The shortest path from 1 > 4 is [1, 6, 4]
The shortest path from 1 > 5 is [1, 6, 4, 5]
The shortest path from 1 > 6 is [1, 6]
The shortest path from 2 > 0 is [2, 6, 0]
The shortest path from 2 > 1 is [2, 6, 0, 1]
The shortest path from 2 > 3 is [2, 6, 3]
The shortest path from 2 > 4 is [2, 6, 4]
The shortest path from 2 > 5 is [2, 6, 4, 5]
The shortest path from 2 > 6 is [2, 6]
The shortest path from 3 > 4 is [3, 4]
The shortest path from 3 > 5 is [3, 4, 5]
The shortest path from 4 > 5 is [4, 5]
The shortest path from 6 > 0 is [6, 0]
The shortest path from 6 > 1 is [6, 0, 1]
The shortest path from 6 > 2 is [6, 0, 1, 2]
The shortest path from 6 > 3 is [6, 3]
The shortest path from 6 > 4 is [6, 4]
The shortest path from 6 > 5 is [6, 4, 5]
PS D:\C C++>
```

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

Answer:

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).
Dynamic Programming is a technique in computer programming that helps to efficiently solve a class of problems that have overlapping sub problems and optimal structure property. Such problems involve repeatedly calculating the value of the same sub problems to find the optimum solution.

Floyd Warshall Algorithm Applications:

- To find the shortest path is a directed graph
- To find the transitive closure of directed graphs
- To find the Inversion of real matrices
- For testing whether an undirected graph is bipartite

Conclusion: (Based on the observations):

Thus, we have studied and understood the Floyd-Warshall algorithm using Dynamic programming, found time complexity and implemented it.

Outcome:

CO3: 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.