 <b>Marwadi University</b>	<b>Marwari University</b> <b>Faculty of Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Design and Analysis of Algorithms (01CT0512)</b>	<b>Aim: Implementing Floyd Warshall All Pair Shortest Path Algorithm</b>	
<b>Experiment No: 13</b>	<b>Date:</b>	<b>Enrollment No: 92200133030</b>

**Aim:** Implementing Floyd Warshall All Pair Shortest Path Algorithm

**IDE:** Visual Studio Code

### **Implementing Floyd Warshall All Pair Shortest Path Algorithm**

#### **Theory: -**


- The Floyd-Warshall algorithm, named after its creators Robert Floyd and Stephen Warshall, is a fundamental algorithm in computer science and graph theory. It is used to find the shortest paths between all pairs of nodes in a weighted graph. This algorithm is highly efficient and can handle graphs with both positive and negative edge weights, making it a versatile tool for solving a wide range of network and connectivity problems.

### **Floyd Warshall Algorithm:**

- The Floyd Warshall Algorithm is an all pair shortest path algorithm unlike Dijkstra and Bellman Ford which are single source shortest path algorithms. 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). It follows Dynamic Programming approach to check every possible path going via every possible node in order to calculate shortest distance between every pair of nodes.

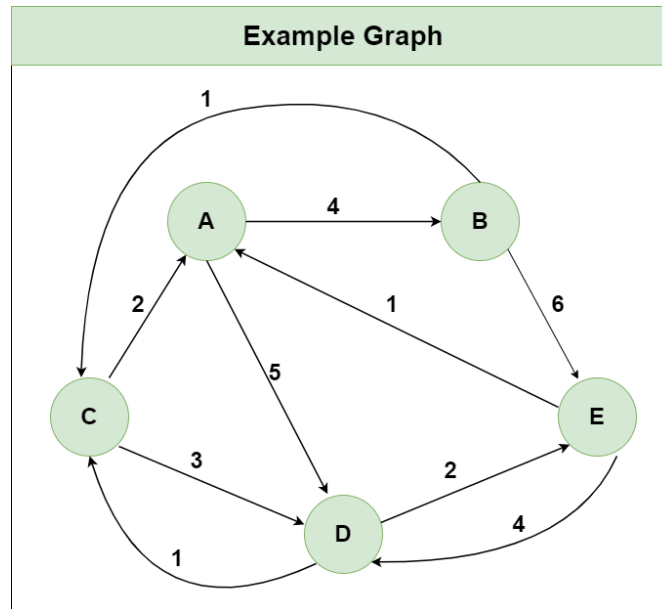
### **Idea Behind Floyd Warshall Algorithm:**

- Initialize the solution matrix same as the input graph matrix as a first step.
- Then update the solution matrix by considering all vertices as an intermediate vertex.
- The idea is to pick all vertices one by one and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.
- When we pick vertex number **k** as an intermediate vertex, we already have considered vertices **{0, 1, 2, .. k-1}** as intermediate vertices.
- For every pair **(i, j)** of the source and destination vertices respectively, there are two possible cases.
  - **k** is not an intermediate vertex in shortest path from **i** to **j**. We keep the value of **dist[i][j]** as it is.
  - **k** is an intermediate vertex in shortest path from **i** to **j**. We update the value of **dist[i][j]** as **dist[i][k] + dist[k][j]**, if **dist[i][j] > dist[i][k] + dist[k][j]**

 <b>Marwadi University</b>	<b>Marwari University</b> <b>Faculty of Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Design and Analysis of Algorithms (01CT0512)</b>	<b>Aim: Implementing Floyd Warshall All Pair Shortest Path Algorithm</b>	
<b>Experiment No: 13</b>	<b>Date:</b>	<b>Enrollment No: 92200133030</b>


### Illustration of Floyd Warshall Algorithm :

Suppose we have a graph as shown in the image:



**Step 1:** Initialize the Distance[][] matrix using the input graph such that Distance[i][j]= weight of edge from i to j, also Distance[i][j] = Infinity if there is no edge from i to j.

Step1: Initializing Distance[ ][ ] using the Input Graph					
	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	∞	0	3	∞
D	∞	∞	1	0	2
E	1	∞	∞	4	0

 <b>Marwadi University</b>	<b>Marwari University</b> <b>Faculty of Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Design and Analysis of Algorithms (01CT0512)</b>	<b>Aim: Implementing Floyd Warshall All Pair Shortest Path Algorithm</b>	
<b>Experiment No: 13</b>	<b>Date:</b>	<b>Enrollment No: 92200133030</b>

Step 2: Treat node A as an intermediate node and calculate the Distance[i][j] for every {i,j} node pair using the formula:

= Distance[i][j] = minimum (Distance[i][j], (Distance from i to A) + (Distance from A to j))

= Distance[i][j] = minimum (Distance[i][j], Distance[i][A] + Distance[A][j])

Step 2: Using Node A as the Intermediate node					
Distance[i][j] = min (Distance[i][j], Distance[i][A] + Distance[A][j])					
	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	?	?	?	?
C	2	?	?	?	?
D	∞	?	?	?	?
E	1	?	?	?	?

→

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	6	0	3	12
D	∞	∞	1	0	2
E	1	5	∞	4	0

Step 3: Treat node B as an intermediate node and calculate the Distance[i][j] for every {i,j} node pair using the formula:


= Distance[i][j] = minimum (Distance[i][j], (Distance from i to B) + (Distance from B to j))

= Distance[i][j] = minimum (Distance[i][j], Distance[i][B] + Distance[B][j])

Step 3: Using Node B as the Intermediate node					
Distance[i][j] = min (Distance[i][j], Distance[i][B] + Distance[B][j])					
	A	B	C	D	E
A	?	4	?	?	?
B	∞	0	1	∞	6
C	?	6	?	?	?
D	?	∞	?	?	?
E	?	5	?	?	?

→

	A	B	C	D	E
A	0	4	5	5	10
B	∞	0	1	∞	6
C	2	6	0	3	12
D	∞	∞	1	0	2
E	1	5	6	4	0

 <b>Marwadi University</b>	<b>Marwari University</b> <b>Faculty of Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Design and Analysis of Algorithms (01CT0512)</b>	<b>Aim: Implementing Floyd Warshall All Pair Shortest Path Algorithm</b>	
<b>Experiment No: 13</b>	<b>Date:</b>	<b>Enrollment No: 92200133030</b>

Step 4: Treat node C as an intermediate node and calculate the Distance[i][j] for every {i,j} node pair using the formula:

= Distance[i][j] = minimum (Distance[i][j], (Distance from i to C) + (Distance from C to j))

= Distance[i][j] = minimum (Distance[i][j], Distance[i][C] + Distance[C][j])

Step 4: Using Node C as the Intermediate node					
Distance[i][j] = min (Distance[i][j], Distance[i][C] + Distance[C][j])					
	A	B	C	D	E
A	?	?	5	?	?
B	?	?	1	?	?
C	2	6	0	3	12
D	?	?	1	?	?
E	?	?	6	?	?

	A	B	C	D	E
A	0	4	5	5	10
B	3	0	1	4	6
C	2	6	0	3	12
D	3	7	1	0	2
E	1	5	6	4	0


Step 5: Treat node D as an intermediate node and calculate the Distance[i][j] for every {i,j} node pair using the formula:

= Distance[i][j] = minimum (Distance[i][j], (Distance from i to D) + (Distance from D to j))

= Distance[i][j] = minimum (Distance[i][j], Distance[i][D] + Distance[D][j])

Step 5: Using Node D as the Intermediate node					
Distance[i][j] = min (Distance[i][j], Distance[i][D] + Distance[D][j])					
	A	B	C	D	E
A	?	?	?	5	?
B	?	?	?	4	?
C	?	?	?	3	?
D	3	7	1	0	2
E	?	?	?	4	?

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

 <b>Marwadi University</b>	<b>Marwari University</b> <b>Faculty of Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Design and Analysis of Algorithms (01CT0512)</b>	<b>Aim: Implementing Floyd Warshall All Pair Shortest Path Algorithm</b>	
<b>Experiment No: 13</b>	<b>Date:</b>	<b>Enrollment No: 92200133030</b>

Step 6: Treat node E as an intermediate node and calculate the Distance[i][j] for every {i,j} node pair using the formula:

= Distance[i][j] = minimum (Distance[i][j], (Distance from i to E) + (Distance from E to j))

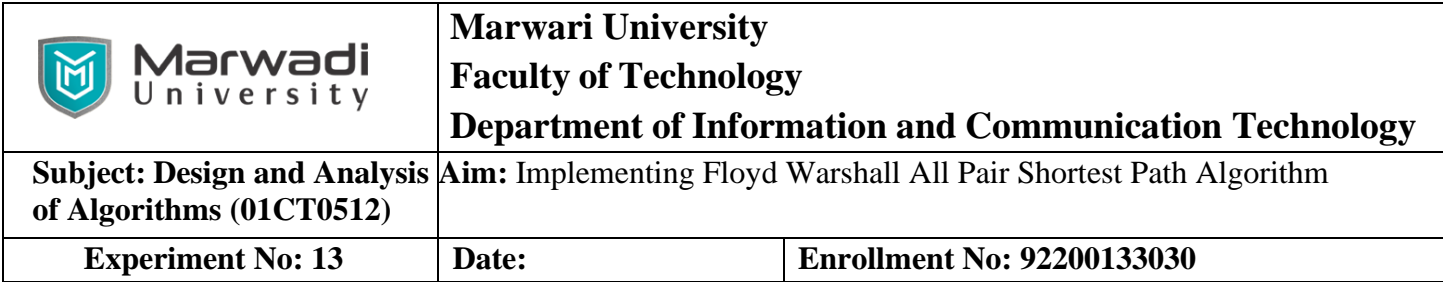
= Distance[i][j] = minimum (Distance[i][j], Distance[i][E] + Distance[E][j])

Step 6: Using Node E as the Intermediate node					
Distance[i][j] = min (Distance[i][j], Distance[i][E] + Distance[E][j])					
	A	B	C	D	E
A	?	?	?	?	7
B	?	?	?	?	6
C	?	?	?	?	5
D	?	?	?	?	2
E	1	5	5	4	0

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Step 7: Since all the nodes have been treated as an intermediate node, we can now return the updated Distance[i][j] matrix as our answer matrix.

Step 7: Return Distance[ ][ ] matrix as the result					
	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0



**Marwari University**  
**Faculty of Technology**  
**Department of Information and Communication Technology**

### **Aim:** Implementing Floyd Warshall All Pair Shortest Path Algorithm

Date:


**Enrollment No: 92200133030**[illegible]

**Code :-**

```
#include <bits/stdc++.h>
using namespace std;
#define V 5
#define INF 99999

void printSolution(int dist[][V]);

void floydWarshall(int dist[][V]){
    int i, j, k;
    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (dist[i][j] > (dist[i][k] + dist[k][j]) && (dist[k][j] != INF &&
dist[i][k] != INF))
```

 <b>Marwadi University</b>	<b>Marwari University</b> <b>Faculty of Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Design and Analysis of Algorithms (01CT0512)</b>	<b>Aim: Implementing Floyd Warshall All Pair Shortest Path Algorithm</b>	
<b>Experiment No: 13</b>	<b>Date:</b>	<b>Enrollment No: 92200133030</b>

```

        dist[i][j] = dist[i][k] + dist[k][j];
    }
}

printSolution(dist);
}

void printSolution(int dist[][V]) {
    cout << "The following matrix shows the shortest distances between every pair of
vertices \n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                cout << "INF" << " ";
            else
                cout << dist[i][j] << " ";
        }
        cout << endl;
    }
}

int main(){
    int graph[V][V] = { { 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 } };

    floydWarshall(graph);
    return 0;
}


```

### Output :-

```

PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 13> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design
-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 13\" ; if ($?) { g++ Flyod_Warshall_Algorithm.cpp -o Flyod_Warshall_Algorithm } ; if ($?) { .\Flyod_War
shall_Algorithm }
The following matrix shows the shortest distances between every pair of vertices
0 4 5 5 7
3 0 1 4 6
2 6 0 3 5
3 7 1 0 2
1 5 5 4 0
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 13>

```

 <b>Marwadi</b> University	<b>Marwari University</b> <b>Faculty of Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Design and Analysis of Algorithms (01CT0512)</b>	<b>Aim: Implementing Floyd Warshall All Pair Shortest Path Algorithm</b>	
<b>Experiment No: 13</b>	<b>Date:</b>	<b>Enrollment No: 92200133030</b>

**Space Complexity:-** \_\_\_\_\_

**Justification: -**

---

---

---

---

---

**Time Complexity:**

**Best Case Time Complexity:** \_\_\_\_\_

**Justification: -**

---

---

---

---

---

**Worst Case Time Complexity:-** \_\_\_\_\_

**Justification: -**

---

---

---

---

---

**Conclusion:-**

---

---

---

---

---