



## Symbiosis Institute of Technology

Department of Computer Science and Engineering

Academic Year 2025-26

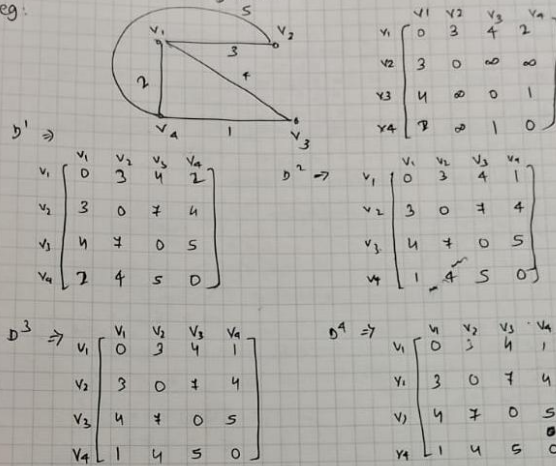
Design Analysis of Algorithm– Lab

Batch 2023-27 - Sem V

Lab Assignment No:- 6	
<b>Name of Student</b>	Deepti Pal
<b>PRN No.</b>	23070122081
<b>Batch</b>	2023-27
<b>Class</b>	CSE – A3
<b>Academic Year &amp; Semester</b>	2025-26 , TY, 5 <sup>th</sup> sem
<b>Date of Submission</b>	22 sept 2025
<b>Title of Assignment:</b>	<p>Assume that you are a software engineer working for a logistics company that operates a fleet of delivery trucks across multiple cities. The company wants to optimize delivery routes to minimize fuel consumption and delivery time. The road network between the cities can be represented as a weighted graph, where:</p> <ul style="list-style-type: none"><li>• Each <b>vertex</b> represents a city.</li><li>• Each <b>edge</b> represents a direct road between two cities.</li><li>• Each <b>weight</b> on the edge represents the <b>distance in kilometers</b> or <b>average travel time</b> between cities.</li></ul>

	<p><b>Problem Statement:</b></p> <p>Implement <b>Floyd's Algorithm</b> to compute the shortest paths between <b>every pair of cities</b>. Then, use the result to answer the following:</p> <ol style="list-style-type: none"> <li>1. What is the <b>shortest distance</b> between City A and City D?</li> <li>2. Which <b>intermediate cities</b> (if any) should the delivery truck pass through to achieve this shortest distance?</li> </ol> <p>Write a program to:</p> <ul style="list-style-type: none"> <li>• Read the distance matrix representing the weighted graph from a file.</li> <li>• Apply <b>Floyd's Algorithm</b> to compute all-pairs shortest paths.</li> <li>• Output the updated shortest distance matrix.</li> <li>• Allow querying of shortest paths between any two cities.</li> </ul>
<p><b>Theory: (Handwritten)</b></p>	<ol style="list-style-type: none"> <li>1. Explain Floyd's algorithm with an example. Show all steps</li> <li>2. Discuss Complexity (Best, Worst, and Average Case) &amp; Space Complexity</li> </ol>

# ① Floyd - Warshall Algorithm eg.



Floyd-Warshall algorithm is used to find shortest paths between all pairs of vertices in a weighted graph (directed/undirected).

- matrix  $dist[i][j]$  store shortest distance from vertex  $i$  to  $j$ .
- Initially, if edge exists:  
 $dist[i][j] = \text{weight } i \text{ to } j$   
 else:  
 $dist[i][j] = \infty$
- update the vertex by keeping vertex  $k$  as intermediate.  
 $dist[i][j] = \min(dist[i][j], dist[i][k] + dist[k][j])$

- ② Floyd-Warshall uses 3 nested loops over vertices:
- So, Best case:  $O(n^3)$   
 Avg case:  $O(n^3)$   
 Worst case:  $O(n^3)$
- Space complexity:  $O(n^2)$

## Source code

```
#include <iostream>
#include <vector>
#include <algorithm>

const int INF = 1e9;

vector<vector<int>> readMatrix(const string &filename)
{
    ifstream file(filename);
    vector<vector<int>> matrix;
    string token;

    while (true)
    {
        vector<int> row;
        string line;
```

```

        if (!getline(file, line))
            break;
        stringstream ss(line);

        while (ss >> token)
        {
            if (token == "INF")
                row.push_back(INF);
            else
                row.push_back(stoi(token));
        }
        if (!row.empty())
            matrix.push_back(row);
    }
    return matrix;
}

void floydWarshall(vector<vector<int>> &dist,
vector<vector<int>> &next)
{
    int n = dist.size();
    next.assign(n, vector<int>(n, -1));

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (dist[i][j] != INF)
                next[i][j] = j;

    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (dist[i][k] + dist[k][j] < dist[i][j])
                {
                    dist[i][j] = dist[i][k] + dist[k][j];
                    next[i][j] = next[i][k];
                }
    }

vector<int> getPath(int u, int v, const
vector<vector<int>> &next)
{
    if (next[u][v] == -1)
        return {};
    vector<int> path = {u};
    while (u != v)

```

```

    {
        u = next[u][v];
        path.push_back(u);
    }
    return path;
}

void printMatrix(const vector<vector<int>> &matrix)
{
    for (auto &row : matrix)
    {
        for (int val : row)
            cout << (val == INF ? "INF" : to_string(val))
<< " ";
        cout << "\n";
    }
}

int main()
{
    vector<vector<int>> dist =
readMatrix("distances.txt");

    if (dist.empty())
    {
        cout << "Error: distance matrix is empty. Check
file path/format!\n";
        return 0;
    }

    cout << "Initial Distance Matrix:\n";
    printMatrix(dist);

    vector<vector<int>> next;
    floydWarshall(dist, next);

    cout << "\nShortest Distance Matrix:\n";
    printMatrix(dist);

    // example
    int start = 0, end = 3;
    string cityNames = "ABCD";

    vector<int> path = getPath(start, end, next);

```

```
    if (!path.empty())
    {
        cout << "\nShortest distance from City " <<
cityNames[start]
            << " to City " << cityNames[end] << ": " <<
dist[start][end] << "\n";
        cout << "Path: ";
        for (int i = 0; i < path.size(); i++)
        {
            cout << cityNames[path[i]];
            if (i < path.size() - 1)
                cout << " -> ";
        }
        cout << "\n";
    }
    else
    {
        cout << "No path exists!\n";
    }

    return 0;
}
```

**Output Screenshots (if applicable)**

The screenshot shows a C++ IDE with the following components:

- File Explorer:** Shows files: Dijkstra.cpp, dijkstra\_2.cpp, Floyd\_Warshall.cpp (active), distances.txt, and job\_scheduling.cpp.
- Editor:** Displays the code for `Floyd_Warshall.cpp` with line numbers 76 to 122. The code implements the Floyd-Warshall algorithm to find the shortest path between four cities (A, B, C, D).
- Terminal:** Shows the execution of the program. It compiles `Floyd_Warshall.cpp` and runs `./Floyd_Warshall.exe`. The output displays the initial distance matrix, the shortest distance matrix, and the shortest path from City A to City D.

```
76  int main()
102  if (!path.empty())
107      for (int i = 0; i < path.size(); i++)
111          cout << " -> ";
112      }
113      cout << "\n";
114  }
115  else
116  {
117      cout << "No path exists!\n";
118  }
119
120  return 0;
121  }
122
```

PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> g++ Floyd\_Warshall.cpp -o Floyd\_Warsh

PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> ./Floyd\_Warshall.exe

Initial Distance Matrix:

```
0 3 INF 7
8 0 2 INF
5 INF 0 1
2 INF INF 0
```

Shortest Distance Matrix:

```
0 3 5 6
5 0 2 3
3 6 0 1
2 5 7 0
```

Shortest distance from City A to City D: 6  
Path: A -> B -> C -> D

PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs>

**Problems Solved from Hacker Rank**

1. <https://www.hackerrank.com/challenges/floyd-city-of-blinding-lights/problem>

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int INF = 1000000000;
5
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9
10    int n, m;
11    cin >> n >> m;
12
13    // Initialize distance matrix
14    vector<vector<int>> dist(n + 1, vector<int>(n + 1, INF));
15    for (int i = 1; i <= n; i++) dist[i][i] = 0;
16
17    // Read edges
18    for (int i = 0; i < m; i++) {
19        int u, v, w;
20        cin >> u >> v >> w;
21        dist[u][v] = w; // overwrite last edge if multiple
22    }
23
24    // Floyd-Warshall algorithm
25    for (int k = 1; k <= n; k++) {
26        for (int i = 1; i <= n; i++) {
27            for (int j = 1; j <= n; j++) {
28                if (dist[i][k] < INF && dist[k][j] < INF) {
29                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
30                }
31            }
32        }
33    }
34
35    // Process queries
36    int q;
37    cin >> q;
38    while (q--) {
39        int u, v;
40        cin >> u >> v;
41        if (dist[u][v] >= INF) cout << -1 << "\n";
42        else cout << dist[u][v] << "\n";
43    }
44
45    return 0;
46 }
47

```

Test case 0

Compiler Message

Success

Test case 1

Test case 2

Test case 3

Test case 4

Test case 5

Test case 6

Input (stdin)

```

1 5 7
2 1 2 20
3 1 3 50
4 1 4 70
5 1 5 90
6 2 3 30
7 3 4 40
8 4 5 60
9

```

2. <https://www.hackerrank.com/challenges/red-knights-shortest-path/problem>



	<pre>1 #include &lt;bits/stdc++.h&gt; 2 using namespace std; 3 4 struct Cell { 5     int r, c; 6 }; 7 8 int main() { 9     ios::sync_with_stdio(false); 10    cin.tie(nullptr); 11 12    int n; 13    cin &gt;&gt; n; 14    int i_start, j_start, i_end, j_end; 15    cin &gt;&gt; i_start &gt;&gt; j_start &gt;&gt; i_end &gt;&gt; j_end; 16 17    const int dr[6] = { -2, -2, 0, 2, 2, 0 }; 18    const int dc[6] = { -1, 1, 2, 1, -1, -2 }; 19    const string moveName[6] = { "UL", "UR", "R", "LR", "LL", "L" }; 20 21    vector&lt;vector&lt;int&gt;&gt; dist(n, vector&lt;int&gt;(n, -1)); 22 23    vector&lt;vector&lt;pair&lt;Cell, int&gt;&gt;&gt; parent(n, vector&lt;pair&lt;Cell, int&gt;&gt;(n, { {-1, -1}, -1 })); 24 25    queue&lt;Cell&gt; q; 26    dist[i_start][j_start] = 0; 27    q.push({ i_start, j_start }); 28 29    while (!q.empty()) { 30        Cell cur = q.front(); 31        q.pop(); 32 33        if (cur.r == i_end &amp;&amp; cur.c == j_end) { 34            break; 35        } 36 37        for (int m = 0; m &lt; 6; m++) { 38            int nr = cur.r + dr[m]; 39            int nc = cur.c + dc[m]; 40            if (nr &lt; 0    nr &gt;= n    nc &lt; 0    nc &gt;= n) continue; 41            if (dist[nr][nc] != -1) continue; // already visited 42 43            dist[nr][nc] = dist[cur.r][cur.c] + 1; 44            parent[nr][nc] = { { cur.r, cur.c }, m }; 45            q.push({ nr, nc }); 46        } 47    } 48 49 }</pre> <div><div><div>Test case 0</div><div>Test case 1</div><div>Test case 2</div><div>Test case 3</div><div>Test case 4</div><div>Test case 5</div><div>Test case 6</div></div><div><div>Compiler Message</div><div>Success</div><div>Input (stdin)</div><div><div>17</div><div>26601</div></div><div>Expected Output</div><div><div>14</div><div>2UL UL UL L</div></div></div></div>
Conclusion	Thus, we have studied Floyd’s algorithm using a dynamic method.

3. <https://www.hackerrank.com/challenges/clues-on-a-binary-path/problem>
4. <https://www.hackerrank.com/challenges/subtrees-and-paths/problem>