**Project 2**

**Analyzing the Floyd - Warshall algorithm**

CSE 5211: Analysis of Algorithms

Dr. William Shoaff

**Zubin Kadva**

Table of Contents

[1. Problem Description 1](#_Toc477378579)

[2. Known algorithms 1](#_Toc477378580)

[3. The shortest path problem 1](#_Toc477378581)

[4. A historical perspective 2](#_Toc477378582)

[5. Algorithm description 2](#_Toc477378583)

[5.1. A recursive solution 3](#_Toc477378584)

[5.2. Matrix representation 5](#_Toc477378585)

[6. Implementation 6](#_Toc477378586)

[7. Generating random data 7](#_Toc477378587)

[8. Analysis 9](#_Toc477378588)

[9. Graphs 10](#_Toc477378589)

[9.1. Execution Time 10](#_Toc477378590)

[9.2. Memory consumption 12](#_Toc477378591)

[9.3. Average performance 13](#_Toc477378592)

[10. References and tools 14](#_Toc477378593)

# 1. Problem Description

The main purpose of this project is to analyse the Floyd-Warshall algorithm used to find the shortest path in a weighted graph with positive or negative edges, but no negative cycles. A shortest path problem is the problem of finding the path between two nodes in a graph such that the sum of the weights of its edges is minimized.

# 2. Known algorithms

The most important algorithms for solving this problem are:

* Dijkstra's algorithm
* Bellman – Ford algorithm
* A\* search algorithm
* Floyd – Warshall algorithm
* Johnson's algorithm
* Viterbi algorithm

# 3. The shortest path problem

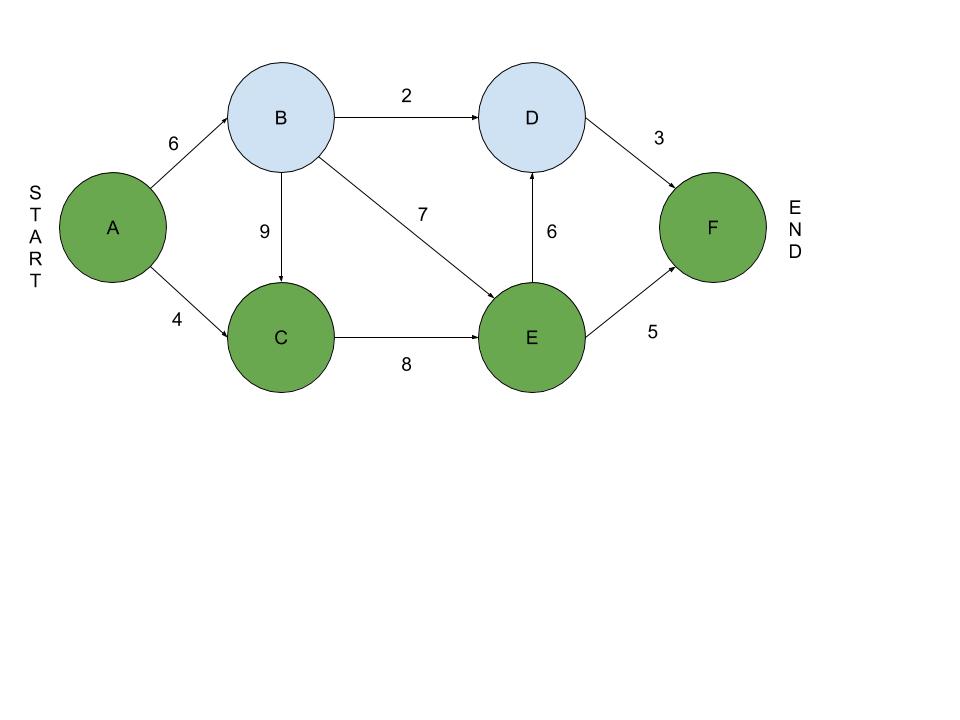
Two vertices are adjacent when they are both incident to a common edge. A path in an undirected graph is a sequence of vertices

such that is adjacent to for. Such a path is called a path of length from to.

Let be the edge incident to both and. Given a real-valued weight function , and an undirected (simple) graph, the shortest path from to is the path (where and that over all possible minimizes the sum

.

When each edge in the graph has unit weight or, this is equivalent to finding the path with fewest edges. [1]



*Figure: Example of shortest path in a graph*

In the above figure, the shortest path is given by with a total cost of

# 4. A historical perspective

The Floyd-Warshall algorithm is an example of a dynamic programming problem. It was first published by **Robert Floyd** in 1962. However, the working of this algorithm is similar to the algorithms previously published by Bernard Roy in 1959 and **Stephen Warshall** in 1962 for finding the transitive closure of a graph. The current formulation of the algorithm as three nested for loops was first described by Peter Ingerman in 1962. Hence, the algorithm is also known as the **Floyds algorithm**, the **Roy-Warshall algorithm** or the **Roy-Floyd algorithm.**

# 5. Algorithm description

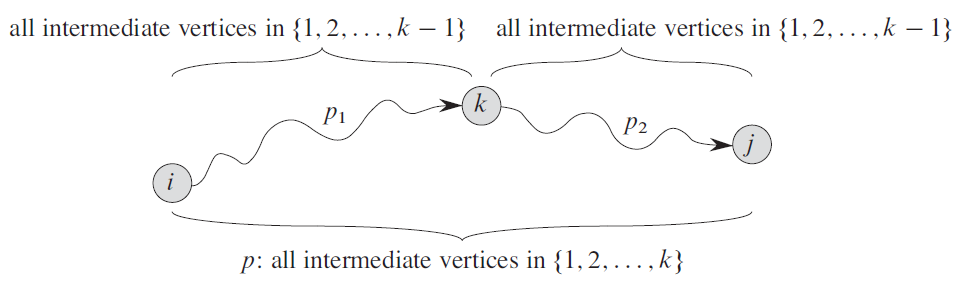
The Floyd-Warshall algorithm considers the intermediate vertices of a shortest path, where an *intermediate vertex* of a simple path is any vertex of other thanthat is, any vertex in the set

The algorithm relies on the observation that the vertices of are , consider a subset of vertices for some. For any pair of vertices, consider all paths from to whose intermediate vertices are all drawn from , and let be a minimum-weight path from among them.

The algorithm exploits a relationship between path and shortest paths from to with all intermediate vertices in the set. The relationship depends on whether or not k is an intermediate vertex of path.

This gives rise to the following cases:

1. *If is not an intermediate vertex of path*, then all intermediate vertices of path are in the set. Thus, a shortest path from vertex to vertex with all intermediate vertices in the set is also a shortest path from to with all intermediate vertices in the set.
2. *If is an intermediate vertex of path*, then we decompose into. is a shortest path from to with all intermediate vertices in the set. Similarly, is a shortest path from vertex to vertex with all intermediate vertices in the set.



*Figure: Path is a shortest path from vertex to vertex, and is the highest-numbered intermediate vertex of [2]*

## 5.1. A recursive solution

Let be the weight of a shortest path from vertex to vertex for which all *intermediate vertices* are in the set. When, a path from vertex to vertex with no intermediate vertex numbered higher than has no intermediate vertices at all. Such a path has at most one edge, and hence. [2]

This can be represented as:

*Figure: The Floyd-Warshall algorithmic steps*

**function** Floyd-Warshall(V)

dist[v][v] = ∞

**for each** vertex *v*

dist[*v*][*v*] := 0

**for each** edge (*u*,*v*)

dist[*u*][*v*] := w(*u*,*v*)

**for** *k* = 1 **to** V

**for** *i* = 1 **to** V

**for** *j* = 1 **to** V

**if** dist[*i*][*j*] > dist[*i*][*k*] + dist[*k*][*j*]

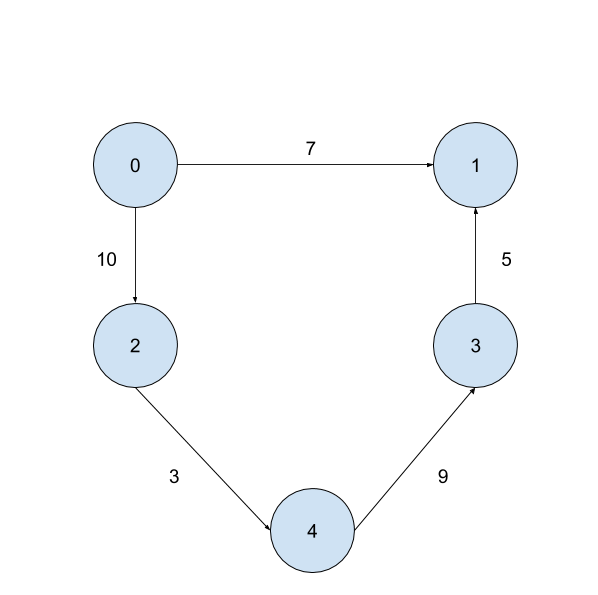
dist[*i*][*j*] := dist[*i*][*k*] + dist[*k*][*j*]

**end if**

*Figure: The Floyd-Warshall algorithm pseudocode*

## 5.2. Matrix representation

Given the following graph:



*Figure: Example directed graph*

Suppose the above graph is denoted by. Then, the corresponding matrix is given as:

Programmatically, this can be represented as:

g[][] = { {0 , 7 , 10 , INF, INF},

{INF, 0 , INF, INF, INF},

{INF, INF, 0 , INF, 3 },

{INF, 5 , INF, 0 , INF},

{INF, INF, INF, 9 , 0 } }

# 6. Implementation

/\* Author: Zubin Kadva

\* Class: Analysis of Algorithms, Spring 2017

\* Project: Floyd-Warshall algorithm

\*/

public class FloydWarshall {

final static int INF = 999;

static void shortestPath(int graph[][]) {

int v = graph.length;

int dist[][] = new int[v][v];

**// Matrix initialization is done here**

for (int i = 0; i < v; i++) {

System.arraycopy(graph[i], 0, dist[i], 0, v);

}

for (int k = 0; k < v; k++) {

**// Start from a vertex as source**

for (int i = 0; i < v; i++) {

**// End at a vertex at the destination staring from the source**

for (int j = 0; j < v; j++) {

**// If vertex k is on the shortest path from i to j, then update the value of dist[i][j]**

if (dist[i][k] + dist[k][j] < dist[i][j]) {

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

}

}

}

}

**// Print the shortest distance matrix**

print(dist);

}

static void print(int dist[][]) {

int v = dist.length;

for (int i = 0; i < v; i++) {

for (int j = 0; j < v; j++) {

if (dist[i][j] == INF) {

System.out.print("INF\t");

} else {

System.out.print(dist[i][j] + "\t");

}

}

}

}

}

# 7. Generating random data

/\* Author: Zubin Kadva

\* Class: Analysis of Algorithms, Spring 2017

\* Project: Floyd-Warshall algorithm

\*/

static int[][] generate() {

final int DIAGONAL = 9999;

int V = 225;

int[][] graph = new int[V][V];

**// Initalize diagonals first**

for (int i = 0; i < V; i++) {

graph[i][i] = DIAGONAL;

}

**// Pick a random distance from 1 to 10**

for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

if (i == j) {

continue;

}

graph[i][j] = new Random().nextInt(10);

}

}

**// Make sure there are INF distances also**

for (int i = 0; i < V; i++) {

int a = new Random().nextInt(V);

int b = new Random().nextInt(V);

if (graph[a][b] != DIAGONAL) {

graph[a][b] = INF;

}

}

**// Reset diagonals to 0**

for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

if (graph[i][j] == DIAGONAL) {

graph[i][j] = 0;

}

}

}

return graph;

}

shortestPath(generate());

# 8. Analysis

Time Complexity

The print subroutine is expressed as:

Thus, complexity is**.**

The shortestPath routine is expressed as:

Thus, the worst case complexity is**.**

The best, average case complexity of the algorithm is also and follows a similar proof. Hence, it follows that the **best case complexity** is and the **average case complexity** is **.**

Space complexity

We store the distances in a 2-D array with rows and columns. Therefore, the space complexity is given by

Thus, the complexity is**.**

In summary,

|  |  |  |
| --- | --- | --- |
| Performance | Best |  |
| **Average** |  |
| **Worst** |  |
| Space | |  |

# 9. Graphs

## 9.1. Execution Time

|  |  |
| --- | --- |
| V | Time (in ms) |
| 5 | 1.13 |
| 10 | 5.48 |
| 15 | 7.86 |
| 20 | 10.8 |
| 25 | 18.5 |
| 30 | 21.3 |
| 35 | 27.5 |
| 40 | 30.7 |
| 45 | 36.8 |
| 50 | 43.1 |
| 55 | 48 |
| 60 | 49.7 |
| 65 | 54.1 |
| 70 | 60.6 |
| 75 | 73.6 |
| 80 | 87.6 |
| 85 | 101.5 |
| 90 | 115 |
| 95 | 129 |
| 100 | 147 |

*Table: Floyd-Warshall execution time from v = 5 to 100*

*Graph: Scatter plot of the above data with a cubic trend line*

## 9.2. Memory consumption

|  |  |
| --- | --- |
| V | Memory (in MB) |
| 5 | 7.24 |
| 25 | 7.58 |
| 45 | 7.92 |
| 65 | 8.25 |
| 85 | 8.92 |
| 105 | 9.93 |
| 125 | 10.93 |
| 145 | 12.28 |
| 165 | 13.62 |
| 185 | 15.3 |

*Table: Floyd-Warshall execution time from v = 5 to 185*

*Graph: Scatter plot of the above data with a quadratic trend line*

## 9.3. Average performance

|  |  |
| --- | --- |
| V | Average |
| 10 | 6.36 |
| 20 | 9.02 |
| 30 | 14.44 |
| 40 | 19.14 |
| 50 | 25.51 |
| 60 | 28.81 |
| 70 | 34.425 |
| 80 | 47.925 |
| 90 | 61.96 |
| 100 | 77.96 |

*Table: Average performance for Floyd-Warshall*

*Graph: Scatter plot of the above data*

# 10. References and tools

[1] Shortest path from <https://en.wikipedia.org/wiki/Shortest_path_problem>

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, “*Introduction to Algorithms”*, MIT Press, 2009

[3] Algorithm pseudocode from [https://en.wikipedia.org/wiki/Floyd%E2%80% 93Warshall\_ algorithm](https://en.wikipedia.org/wiki/Floyd%E2%80%25%2093Warshall_%20algorithm)

The NetBeans profiler for profiling the performance of the algorithm.

Microsoft Excel for plotting graphs of the gathered data.

Google Drawings for diagrammatic representation of data.