From Iterative to Recursive: Floyd-Warshall Algorithm

By:

John H. Gray

Submitted to:

The University of Liverpool

CSCK541

Software Development in Practice October 2022 B

Word Count: 1,119

11/28/2022

Table of Contents

[Introduction 2](#_Toc120570577)

[Floyd-Warshall Algorithm 2](#_Toc120570578)

[Recursive Relation 3](#_Toc120570579)

[Hypothesis Updated 4](#_Toc120570580)

[Integrated Developer Environment 4](#_Toc120570581)

[Programming Language 4](#_Toc120570582)

[Language Style Standard: PEP 8 5](#_Toc120570583)

[Source Control 5](#_Toc120570584)

[Unit Tests 5](#_Toc120570585)

[Performance Tests 5](#_Toc120570586)

[Supporting Documentation 6](#_Toc120570587)

[Conclusion 6](#_Toc120570588)

[References 7](#_Toc120570589)

# Introduction

This paper works through the development of the Floyd-Warshall (FW) algorithm in order to solve so called ‘All Pairs Shortest Paths’ (APSP) problems. And we do thisboth from an iterative example given (Geeks, 2022) to recursive approach prepared. We enter the files into source control including scripts for unit and performance tests (GitHub, 2022).

*Expectation*

We hypothesize that both iterative and recursive approaches produce correct solution matrices and what separates them comes down to questions of problem size and complexity. It is also shown that the iterative approach is not always faster and the reasons for this can be various.

## Floyd-Warshall Algorithm

Like many dynamic programming algorithms, FW is generally useful when the problem is worked by solving similar or in this case same subproblems such as the perennial APSP problem. It is said that FW is a simple as it is elegant – sometimes preferred for relatively small problems and dense graphs compared to counterparts such as Dijkstra’s or Johnson’s (Toroslu, 2021).

As a first step of development, we introduce a matrix which reflects all pairs and directed distances (Figure 1).

*Figure 1 – Square Adjacency Matrix*

This structured set of lists forms the input basis for the FW algorithm to consume as it continuously updates e*n route* to computing shortest distances in the solution matrix.

Values of infinity “ are set variables meant to give indication that no path or no distance exists (either there is intermediate path or no path whatsoever). Values not equal to “ and “0” indicate there is some directed distance between them.

# Recursive Relation

For every node or pair, we present input distances for the algorithm to compute. In the broadest sense, for each distance there are two possible scenarios: 1) where start and destination node are equal then distance is zero (hence for the above square matrix the negative forty-five-degree diagonal of zeros); or 2) start and destination are not equal, and yet a path exists, and values reflect valid distances. This recursive relation is at the heart of the FW algorithm.

(1)

The following pseudo code describes systematic solving of same subproblems to find the cheapest paths of all pairs of vertices (Figure 2) and this works provided there are no negative distances given. Gross and Yellen (2003) in their graph theory textbook describe the code as follows.

|  |  |
| --- | --- |
| Floyd Warshall ‘FW’ Pseudo Code | |
| 1: | *N*: count of vertices |
| 2: | *A*[1.*.N*, 1..*N*]: matrix () |
| 3: | **program** FW(*N*, A[1..*N*, 1..*N*]) |
| 4: | **for** k: 1 → *N* **do** |
| 5: | **for** i: 1 → *N* **do** |
| 6: | **for** j: 1 → *N* **do** |
| 7: | **if** A[i, j] > A[i, k] + A[k, j] **then** |
| 8: | A[i, j] ← A[i, k] + A[k, j] |
| 9: | **end if** |
| 10: | **end for** |
| 11: | **end for** |
| 12: | **end for** |
| 13: | **end program** |

*Figure 2 – FW Pseudo Code*

Complexity for time and space are O(V^3) and O(V^2) respectively because we are re-using the given matrix and updating it to arrive at the solution matrix (Gross, 2003).

The calculated solution of cheapest or shortest paths is described below (Figure 3).

*Figure 3 – Solution Matrix*

## Hypothesis Updated

The simple hypothesis is that 1) both iterative and recursive approaches will generate the correct solution matrix; and 2) the iterative approach will always perform faster with fewer lines of code.

*Results*

Unit testing and performance trials demonstrate that both approaches arrive at the same solution matrix however the iterative approach is not always faster.

## Integrated Developer Environment

Visual Studio Code was selected for it is a complete and ideal integrated development environment (IDE) either for creating small or large, complex and scalable applications (Garcia, 2022).

## Programming Language

Leading imperative programming languages like Python, Java, JavaScript and C++ are consistently represented as most popular languages used in practice (Gunnarsson, 2022). Python was selected as it consistently gets top marks for data science because of its strengths in terms of statistics, readability and data modelling (GeeksforGeeks, 2020).

## Language Style Standard: PEP 8

PEP 8 is the authoritative and evolving style guide for writing Python code. Its original insight is that readability matters (PEP 257, 2001) and that code is read far more often than it is written, therefore a consistent framework of style and convention is necessary (PEP 8, 2001).

## Source Control

All files including the given iterative example provided, the recursive counterpart, the unit tests, the requirements are put under source control (GitHub, 2022).

## Unit Tests

Three tests were prepared and conducted of the recursive function. In the first two, the simple test of two targeted value updates in the solution matrix comes back true. In the third, the entire solution matrix is tested to confirm all pairs have solved shortest paths.

This overlapping coverage only generates unit test value if very specific values need investigation. Taken together the unit tests fall short of handling automatically any changes to input matrix (as both graph input and result in test are hard-coded). In future these results under test could be parameterized. Further, additional tests could be designed for every block of code or very specific parts to the functions.

## Performance Tests

Ten trials were conducted to measure run time in milliseconds where the iterative approach was faster 90 percent of the time (Table 1). Recursive and iterative programs ran on average for 7.35 and 6.82 milliseconds respectively.

|  |  |  |
| --- | --- | --- |
| **Trial** | **Recursive** | **Iterative** |
| 1 | 7.36 | 6.88 |
| 2 | 7.27 | 6.72 |
| 3 | 7.28 | 6.90 |
| 4 | 7.50 | 6.63 |
| 5 | 7.35 | 6.74 |
| 6 | 7.75 | 6.83 |
| 7 | 6.80 | 6.99 |
| 8 | 7.17 | 6.82 |
| 9 | 7.66 | 6.90 |
| 10 | 7.51 | 6.60 |
| **average:** | **7.35** | **6.82** |

*Table 1 – Average Duration in Milliseconds*

The interesting seventh trial shows iterative performance was slower so it cannot be concluded by this test if iterative approaches are always faster than recursive. While recursion can have larger amounts of overhead it can also be favourable for shorter code (GeeksforGeeks, 2022). The latter point might have been the reason for the very similar durations.

# Supporting Documentation

A simple markup document was created to provide guidance on the Python version used, downloading the requirements and their versions, understanding at a high-level the problem being worked and finally running the programs and tests.

# Conclusion

This paper focused on the FW algorithm to solve APSP problems. We overviewed some of the maths and pseudo code behind the recursive relation which is fundamental to working the APSP problem. We contrasted iterative versus recursive coding concepts – and in addition, an overview of the selected technology and controls in source and tests.

The expectation or hypothesis proved partially true: iterative and recursive approaches produce correct solution matrices however the iterative approach is not always faster and the reasons for this can be various.

# References

Garcia, M.A.T. (2022), *‘Hands-On Visual Studio 2022: A developer’s guide to exploring new features and best practices in VS2022 for maximum productivity’*. Available at: https://search-ebscohost-com.liverpool.idm.oclc.org/login.aspx?direct=true&db=edsoai&AN=edsoai.on1337932572&site=eds-live&scope=site (Accessed: November 28, 2022).

GeeksforGeeks. (2020), *‘Top Programming Languages for Data Science in 2020’*. Available at: <https://www.geeksforgeeks.org/top-programming-languages-for-data-science-in-2020/> (Accessed: November 26, 2022).

GeeksforGeeks. (2022), *‘Difference between Recursion and Iteration’*. Available at: https://www.geeksforgeeks.org/difference-between-recursion-and-iteration/ (Accessed: November 26, 2022).

GitHub (2022), JohnHGray/floyd. Available at: https://github.com/JohnHGray/floyd (Accessed: November 28, 2022).

Gross, J.L. and Yellen, J. (2003). *‘Handbook of Graph Theory’*. Google Books. CRC Press. Available at: https://books.google.de/books?id=mKkIGIea\_BkC&pg=PA65&redir\_esc=y#v=onepage&q=floyd&f=false (Accessed: November 28, 2022).

Gunnarsson, K. et al., (2020), *‘The Most Popular Programming Languages of GitHub’s Trending Repositories’*. Available at: https://search-ebscohost-com.liverpool.idm.oclc.org/login.aspx?direct=true&db=edsoai&AN=edsoai.on1235326346&site=eds-live&scope=site (Accessed: November 28, 2022).

Toroslu, Ismail H., (2021), *‘Improving the Floyd-Warshall All Pairs Shortest Paths Algorithm’*. Available at: https://search-ebscohost-com.liverpool.idm.oclc.org/login.aspx?direct=true&db=edsoai&AN=edsoai.on1269574234&site=eds-live&scope=site (Accessed: November 27, 2022).

PEP 8, (2001), *‘Style Guide for Python Code’*. Available at: https://peps.python.org/pep-0008/ (Accessed: November 26, 2022).

PEP 257, (2001), *‘Docstring Conventions’*. Available at: https://peps.python.org/pep-0257/ (Accessed: November 26, 2022).

Viva (2019). *‘14 Difference Between Recursion and Iteration with Example’*. Viva Differences. Available at: https://vivadifferences.com/difference-between-recursion-and-iteration-with-example/ (Accessed: November 26, 2022).