FLOYD'S ALGORITHM RECURSIVE IMPLEMENTATION

By

FOUSEKIS EFTHYMIOS

Submitted to

The University of Liverpool

MSc Data Science and Artificial Intelligence

Software Development in Practice

Word Count: 1.080

26/02/2024

FLOYD'S ALGORITHM RECURSIVE IMPLEMENTATION

Submitted to

The University of Liverpool

Word Count: 1.080

26/02/2024

TABLE OF CONTENTS

Chapter 1. INTRODUCTION

Floyd's Algorithm, also known as the Floyd-Warshall Algorithm, is a dynamic programming algorithm designed to find the shortest paths between all pairs of vertices in a weighted graph. It addresses the problem of finding the most efficient way to traverse from one vertex to another while considering the weights of the edges. This algorithm, named after the eminent computer scientist Robert Floyd, plays a pivotal role in diverse fields such as network routing, transportation systems, and data analysis.

The primary objective of Floyd's algorithm is to unravel the mystery of optimal paths within a graph. Whether it's about mapping the most efficient route in a network or understanding the connectivity of data points, Floyd's algorithm delivers a comprehensive solution by computing the shortest paths between every pair of vertices.

Chapter 2. RECURSIVE IMPLEMENTATION

Implementing Floyd's algorithm recursively introduces several advantages that contribute to the elegance and simplicity of the code (Thomas, Recursion: Recursion: An assessment of its pros and cons 2023):

1. **Readability and Simplicity**: Recursive implementations often result in code that mirrors the underlying mathematical definition of the algorithm. This inherent simplicity makes the code more readable and easier to understand, especially for those well-versed in graph theory.

2. **Intuitive Approach**: The recursive approach aligns with the natural definition of the problem, making it intuitive and closely tied to the conceptual understanding of graph traversal and path optimization.

3. **Conciseness**: Recursive code tends to be more concise compared to its iterative counterpart. This conciseness not only aids in understanding but also reduces the amount of boilerplate code, leading to cleaner and more elegant solutions.

2.1 Performance Differences with Recursive Implementation

While the recursive implementation brings conceptual clarity and elegance, it may come at the cost of performance. Recursive algorithms, including Floyd's, often exhibit differences in terms of memory usage and execution time when compared to their iterative counterparts.

1. **Stack Overhead**: Recursive calls contribute to the call stack, and deep recursion can lead to a stack overflow. The depth of recursion becomes a crucial factor, especially for larger graphs.

2. **Memory Utilization**: Recursive functions tend to consume more memory due to the overhead of maintaining the call stack. This can result in higher memory usage, impacting the efficiency of the algorithm, particularly for large datasets.

3. **Execution Time**: Recursive implementations may suffer from increased function call overhead, potentially leading to slower execution times. This becomes more noticeable when dealing with computationally intensive tasks or large datasets.

2.2 Performance and Unit Test Considerations

To ensure the reliability and efficiency of Floyd's algorithm, a comprehensive testing strategy is essential:

1. **Unit Tests**: Unit tests should cover a spectrum of scenarios, including positive and negative edge weights, graphs with cycles, disconnected graphs, and other corner cases. These tests validate the correctness of the algorithm under various conditions.

2. **Performance Tests**: Evaluate the algorithm's performance on graphs of varying sizes. Measure execution time and memory usage for both the recursive and iterative implementations to gauge the efficiency of each.

In the process of building the application, a comprehensive suite of tests was created to ensure the correctness and efficiency of the Floyd's Algorithm implementation. Unit tests focused on verifying the correctness of individual components and functions. These tests covered scenarios such as graphs with varying sizes, different edge weights, and special cases like graphs with only one vertex.

Performance tests were designed to evaluate the algorithm's efficiency and scalability. This involved creating graphs of increasing sizes and measuring the execution time of the algorithm. The profiling tool cProfile is utilized to gather detailed performance metrics.

2.3 Potential Issues with Recursive Implementation

Despite its elegance, the recursive implementation of Floyd's algorithm may encounter challenges:

1. **Stack Overflow**: Deep recursion can lead to a stack overflow, resulting in runtime errors. This is a common issue in recursive algorithms, and careful consideration of recursion depth is crucial.

2. **Memory Intensive**: Recursive algorithms can be more memory-intensive due to the overhead of maintaining the call stack. This can lead to higher memory usage, particularly for large datasets.

3. **Suboptimal Performance**: Recursive implementations may exhibit suboptimal performance, especially when compared to their iterative counterparts. The overhead of function calls and memory allocation can contribute to slower execution times.

2.4 Building the Application and Test Suite

The application was built using a modular approach, separating concerns into different modules such as the recursive implementation of Floyd's algorithm (floyd_recursive), utility functions, and testing modules. The unittest framework was employed to create a suite of unit tests, covering various aspects of the algorithm's functionality.

The unit tests encompass scenarios ranging from simple cases to more complex situations. They include tests for valid graphs, invalid graph dimensions, negative edge weights, and memoization behavior. Additionally, performance tests measure execution time and memory usage for both recursive and iterative implementations.

2.5 Hypothesis for Performance Differences

The recursive implementation is expected to exhibit differences in performance compared to the iterative version due to the inherent characteristics of recursive algorithms. The primary hypothesis revolves around the potential stack overflow issues and increased memory usage associated with recursive function calls.

The iterative version, is likely to demonstrate better performance in terms of both execution time and memory utilization. The iterative approach typically involves a more optimized memory management strategy, reducing the risk of stack overflows and optimizing overall resource consumption.

Chapter 3. CONCLUSIONS

In conclusion, Floyd's algorithm provides an elegant solution to the shortest path problem in weighted graphs. The recursive implementation offers simplicity and readability, but potential issues such as stack overflow and increased memory usage may impact its performance.

The testing strategy, involving comprehensive unit and performance tests, is crucial for validating the correctness and efficiency of the algorithm. Consideration of algorithmic complexity, optimization techniques, and careful analysis of memory usage are essential for building robust and efficient recursive implementations.

As we navigate the intricacies of Floyd's algorithm, a balance between elegance and performance becomes crucial. The recursive approach, while conceptually appealing, demands careful scrutiny and optimization to meet the demands of efficiency and scalability in real-world applications.

The recursive implementation might exhibit slower performance for larger graphs, and this could be attributed to the accumulation of function call overhead. Additionally, the

lack of tail call optimization in some Python interpreters could contribute to a noticeable performance gap between the two implementations.

When choosing between recursive and iterative implementations, developers must carefully weigh the trade-offs based on the specific requirements of their application and the characteristics of the graphs they expect to handle. The insights gained from performance testing and analysis contribute to informed decision-making and the continuous improvement of the algorithm's implementation.

REFERENCES

GfG (2024) *Floyd Warshall algorithm, GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/ (Accessed: 25 February 2024).

Thomas, C. (2023) *Recursion: Recursion: An assessment of its pros and cons*, *CopyProgramming*. Available at: https://copyprogram-ming.com/howto/what-are-the-advantages-and-disadvantages-of-recursion (Accessed: 26 February 2024).