**Recursive version of Floyd’s algorithm**

By

Pawan Thatal

Submitted to

The University of Liverpool

Master of Computer Science

*Software Development in Practice*

Word Count: 1323

18/09/2023

**Recursive version of Floyd’s algorithm**

Submitted to

The University of Liverpool

Word Count: 1323

18/09/2023

TABLE OF CONTENTS

Page

[Chapter 1. Introduction 2](#_Toc145977027)

[Chapter 2. How does it work 3](#_Toc145977028)

[2.1 Step 1 3](#_Toc145977029)

[2.2 Step 2 4](#_Toc145977030)

[2.3 Step 3 4](#_Toc145977031)

[2.4 Final output 5](#_Toc145977032)

[Chapter 3. Importance of Floyd’s Algorithm 6](#_Toc145977033)

[Chapter 4. Unit Testing 6](#_Toc145977034)

[Chapter 5. Performance Testing 6](#_Toc145977035)

[Chapter 6. Conclusion 7](#_Toc145977036)

[REFERENCES 8](#_Toc145977037)

# Introduction

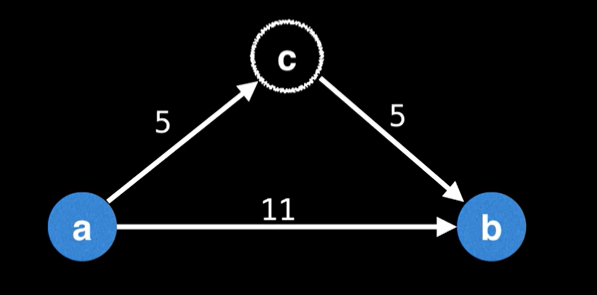
The Floyd Warshall Algorithm is employed to solve the shortest-path problem for all pairs. This algorithm is employed to identify the shortest path between two nodes in a directed graph with weighted edges (geeksforgeeks, not dated)

Finding the shortest path between all the pairs of vertices in a weighted graph programmatically, dynamic programming approach is followed. Dynamic programming is a technique where a complex problem is broken down into smaller sub-problems, solved and the solutions are combined to solve the original problem. (C. BasuMalik, 2023).

In dynamic programming, there is generally a sequence of decision that results in an output.

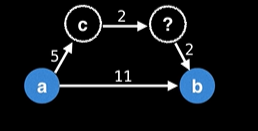
The main idea behind the Floyd Warshall algorithm is to gradually build up all the intermediate routes between the nodes to find the optimal path.

As an example, if you have a distance of 11 between node a and b, that doesn’t necessarily mean it is the shortest path. There could be a 3rd node “C” and the distance between a to c and c to b could be shorter as shown in **Fig 1.1**



**Fig 1.1**

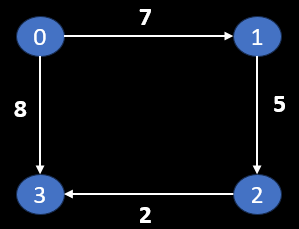
The goal of Floyd Warshall is to eventually consider going through all possible intermediate nodes on paths of different lengths shown in **Fig 1.2**



**Fig 1.2**

# How does it work

The matrix provided in the assigned can be made in a 4 x 4 matrix as below figure 2.1. This can be used to calculate the shortest path following Floyds Algorithm.



In the python program submitted along with this report the input graph is show as follows

graph = [

    [0, 7, INF, 8],

    [INF, 0, 5, INF],

    [INF, INF, 0, 2],

    [INF, INF, INF, 0]

## Step 1

Create matrix A0 of dimension n\*n where n is the number of vertices, in this case it is 4 x 4. The row and the column are indexed as “i” and “j” are the vertices of the graph. As shown **in Fig 2.2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** |
| **0** | 0 | 7 | NP | 8 |
| **1** | NP | 0 | 5 | NP |
| **2** | NP | NP | 0 | 2 |
| **3** | NP | NP | NP | 0 |

**Fig 2.2**

In the python program base case is set by the following code -

    # base case - source node = destinate node path distance is zero therefore return to matrix without update

    if k <0:

        return graph

    if i == j:

        return graph

## Step 2

Create matrix A1 using Matrix A0. Entries in the first row and column are left as they are, and rest of the cells are filled using the formula as below

A[i][k] + A[k][j] < A[i][j] than   A[i][j] = A[i][k] + A[k][j]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** |
| **0** | 0 | 7 | 12 | NP |
| **1** | NP | 0 | 5 | NP |
| **2** | NP | NP | 0 | 2 |
| **3** | NP | NP | NP | 0 |

**Fig 2.2**

In the python program this is represented as -

    if graph[i][k] + graph[k][j] < graph[i][j]:

        graph[i][j] = graph[i][k] + graph[k][j]

## Step 3

Create matrix A2 using Matrix A1. Entries in the first row and column are left as they are, and rest of the cells are filled using the formula.

A[i][k] + A[k][j] < A[i][j] than   A[i][j] = A[i][k] + A[k][j]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** |
| **0** | 0 | 7 | 12 | NP |
| **1** | NP | 0 | 5 | NP |
| **2** | NP | NP | 0 | NP |
| **3** | NP | NP | NP | 0 |

**Fig 2.4**

The recursive calling of function to complete all rows and columns is done as follows:

    if graph[i][k] + graph[k][j] < graph[i][j]:

        graph[i][j] = graph[i][k] + graph[k][j]

    if i < len(graph) - 1: # check to identify end of the row

        if j < len(graph) - 1: # check to identify end of the column

            return floyd\_recursive(graph, i, j + 1, k) # if it is not end of the column, it increments the column "j" by 1 to find the next shortest path

        else:

            return floyd\_recursive(graph, i + 1, 0, k) # if it is end of the column, it resets the column to zero and increments "i" by 1 to move to next row

    else:

        if j < len(graph) - 1: # check to see if there are more column to process

            return floyd\_recursive(graph, 0, j + 1, k) #if true "j" column is incremented by 1 tp proces remaing column

        else:

            return floyd\_recursive(graph, 0, 0, k - 1) # once all rows and columns are processed, intermediate is decremented by 1 to move to the next intermediate node.

## Final output

Create matrix A3 using Matrix A2. Entries in the first row and column are left as they are, and rest of the cells are filled using the formula.

A[i][k] + A[k][j] < A[i][j] than   A[i][j] = A[i][k] + A[k][j]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** |
| **0** | 0 | 7 | 12 | 8 |
| **1** | NP | 0 | 5 | 7 |
| **2** | NP | NP | 0 | 2 |
| **3** | NP | NP | NP | 0 |

**Fig 2.5**

This final matrix is the version that outputs the shortest path in the given problem.

Following code prints the result.

# Call the recursive function to compute shortest distances

result = floyd\_recursive(graph, 0, 0, 0)

# Print the result to show the shortest path for all vertices.

for row in result:

    print(row)

# Importance of Floyd’s Algorithm

Floyd's Algorithm plays a significant role in various fields of decision mathematics, a branch of applied mathematics that deals with constructing and analysing methods to make informed decisions. Some of these applications include network routing, operations research, game theory and computer graphics (studysmarter, not dated)

# Unit Testing

The test script is for unit testing of function “floyd\_recursive”, function designed to recursively calculated shortest path with a given matrix.

The script uses Python unittest module. setUp method is used to define individual test cases.

Test case is to call “floyd\_recursive” with a test graph and starting parameters and compare the output with expected output which is defined in “expected\_result”

In summary, this unit testing code sets up a test case for the floyd\_recursive function and checks if the function produces the expected results for a given input graph. If the function behaves as expected, the test case will pass; otherwise, it will fail.

# Performance Testing

The test script is for performance testing of function “floyd\_recursive”, function designed to recursively calculated shortest path with a given matrix.

Performance test is designed to take user input graph as a parameter and used “random” module to create a random matrix based on initial user input. Once the graph size is defined, performance test function “performance\_test” is designed to track the start time when the “floyd\_recursive” function is called and end time when the function completes the calculation. The difference between them is used to calculate “performance\_time”. Adding the option of user input allows for customization and comparative analysis on performance on different graph sizes.

# Conclusion

The reason iterative solutions are preferred and more efficient for implementing Floyd's algorithm is due to several factors –

Simplicity and readability - iterative solutions are simpler to understand and update.

The recursive function can cause overhead and high memory usage as each function call will remain in use until the base case is reached. For the same reason, it requires greater time (educative.io, not dated)

Overall, while it is possible to implement Floyd's algorithm recursively, an iterative approach is generally preferred for its simplicity, efficiency, and better suitability for real-world applications involving large or dense graphs.

REFERENCES

geeksforgeeks, not dated. Available at: <https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/> (Accessed 18 September 2023).

C BasuMalik, 2023. Available at: <https://www.spiceworks.com/tech/devops/articles/what-is-dynamic-programming/> Accessed 18 September 2023).

studysmarter, 2023. Available at: <https://www.studysmarter.co.uk/explanations/math/decision-maths/floyds-algorithm/> Accessed 18 September 2023).

educative.io, notdated. Available at:https://www.educative.io/courses/recursion-for-coding-interviews-in-cpp/advantages-and-disadvantages. Accessed (18 September 2023)