COMPSCI 2XC3: WINTER 2024
FINAL PROJECT
(INDIVIDUAL PART 6)

Jenil Maru
MAC ID: maruj
STUDENT ID: 400431337

TABLE OF CONTENTS

Overview:

It appears that the unknown() function calculates the shortest paths between each pair of nodes in a directed weighted graph, which the DirectedWeightedGraph class represents. The Floyd-Warshall algorithm is a well-known graph theory algorithm that looks to be implemented by the unknown() function based on the code that is provided.This algorithm is used for finding the shortest paths in a weighted graph with positive or negative edge weights (but no negative cycles).

Function Analysis:

Graph Representation: The graph is represented using an adjacency list ('self.adj') and a dictionary to store weights of edges ('self.weights'). The nodes are assumed to be numbered as 0, 1, ..., n-1.

Initialization: The 'init_d(G)' function initializes a matrix 'd', where 'd[i][j]' represents the shortest path distance from node 'i' to node 'j'. It sets each 'd[i][j]' to infinity initially unless there is a direct edge between 'i' and 'j', in which case it sets it to the edge weight. The distance from a node to itself is set to 0, which is why the diagonal elements ‘d[I][I]’ are set to 0.

Core of the Algorithm: The 'unknown()' function then iteratively updates the matrix 'd' to find the shortest paths. This function uses three nested loops to iterate over all possible paths between node pairs. For each pair of nodes '(i, j)', it checks if the path from 'i' to 'j' through another node 'k' is shorter than the currently known shortest path. If it is, it updates the distance.

The function efficiently calculates the shortest path distances between every pair of nodes in the graph by repeating this process for every possible intermediate node, 'k'.

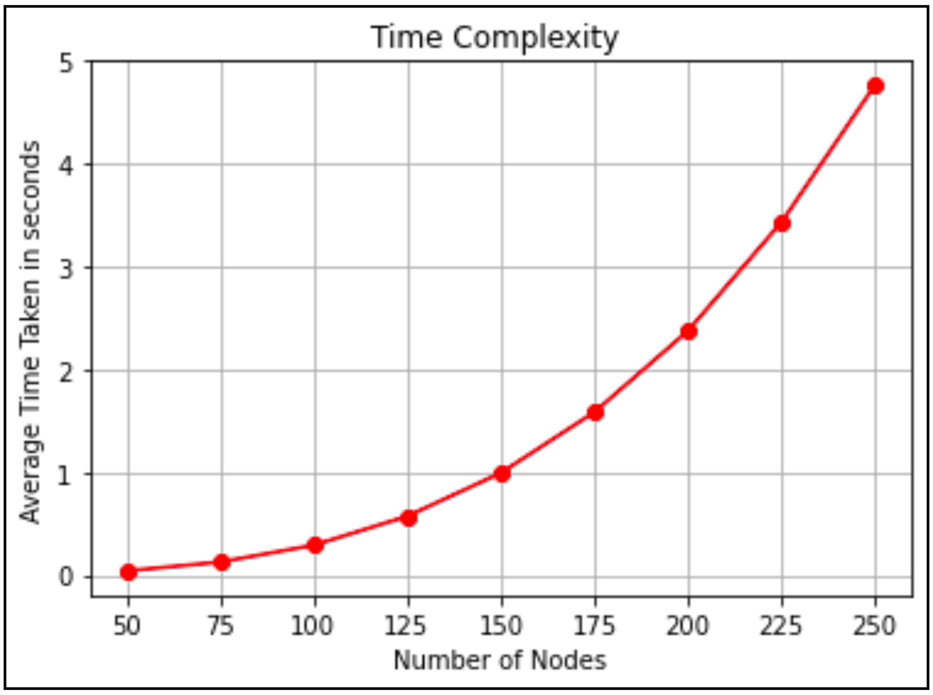
Time Complexity:

The time complexity of the initialization step (init_d), where n is the number of nodes, is O(n^2).

The complexity of the unknown algorithm is (O(n^3)), where (n) is the number of nodes. This comes from the three nested loops iterating over all nodes.

The O(n^3) time complexity is not unexpected given the nested loop structure and the need to calculate the shortest path distances between every pair of nodes. The function executes a cubic number of operations, which is typical of algorithms like Floyd-Warshall's algorithm that use dynamic programming techniques to solve all-pairs shortest paths problems. The size, density, and effectiveness of the underlying operations in the DirectedWeightedGraph class, as well as other factors, may all affect the actual performance.

We can run some experiments and show a graph to verify this time complexity.
We use graph sizes: graph_sizes = [50, 75, 100, 125, 150, 175, 200, 225, 250].

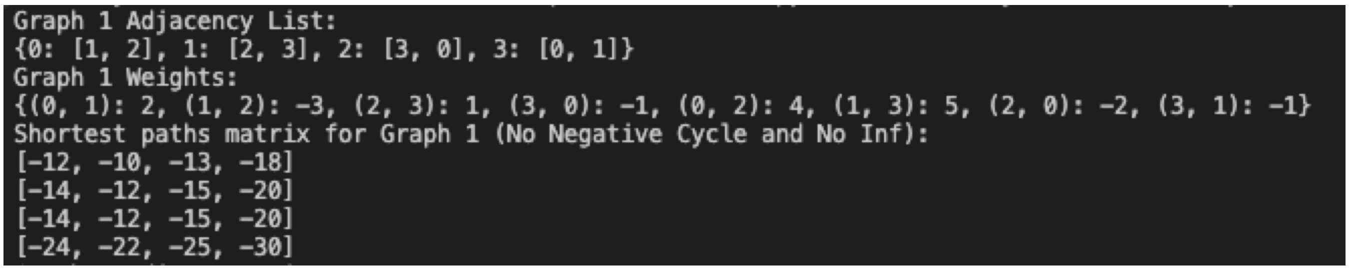


Experimentation and Testing:

We can test this function on graphs with negative weights but without negative cycles to see if it correctly calculates the shortest paths.

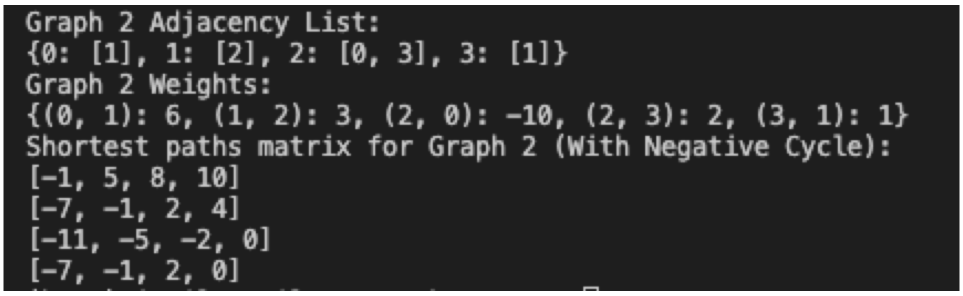
In the first case, given the algorithm appears to be Floyd-Warshall algorithm, it will compute the shortest paths correctly as there are no negative cycles. In the second test case, the algorithm will still run, but its output may not be valid due to the presence of a negative cycle, which the Floyd-Warshall algorithm is not designed to handle correctly without modification.

Negative edge weights without cycle:



```
Graph 1 Adjacency List:
{0: [1, 2], 1: [2, 3], 2: [3, 0], 3: [0, 1]}
Graph 1 Weights:
{(0, 1): 2, (1, 2): -3, (2, 3): 1, (3, 0): -1, (0, 2): 4, (1, 3): 5, (2, 0): -2, (3, 1): -1}
Shortest paths matrix for Graph 1 (No Negative Cycle and No Inf):
[-12, -10, -13, -18]
[-14, -12, -15, -20]
[-14, -12, -15, -20]
[-24, -22, -25, -30]
```

Negative edge weights with negative cycle:



```
Graph 2 Adjacency List:
{0: [1], 1: [2], 2: [0, 3], 3: [1]}
Graph 2 Weights:
{(0, 1): 6, (1, 2): 3, (2, 0): -10, (2, 3): 2, (3, 1): 1}
Shortest paths matrix for Graph 2 (With Negative Cycle):
[-1, 5, 8, 10]
[-7, -1, 2, 4]
[-11, -5, -2, 0]
[-7, -1, 2, 0]
```

The output clearly demonstrates the typical behavior of the Floyd-Warshall algorithm when faced with negative cycles: it continues to reduce the path cost as it includes paths passing through the cycle, leading to increasingly negative path costs without a mechanism to detect or stop at a negative cycle. This behavior highlights the need for algorithms like Bellman-Ford for environments where negative cycles may exist, as it can detect such cycles and prevent these pathologically low path costs.

The algorithm generates a false result and is unable to identify a cycle, as is evident from the above. On the other hand, it produces an accurate result in the absence of a negative cycle.