**Prosit one- small detour;**

**KEYWORDS**

* **Smart cities**—A smart city is a place where traditional networks and services are made more efficient through the use of digital solutions, benefiting its inhabitants and businesses. // Idris
* **Itinerary (on a trial basis)** - a travel document recording a route or journey (depends on context) // Amine
* **Optimal route**—the most cost-efficient route, i.e., it includes all relevant factors, such as the number and location of all the required stops on the route and time windows for deliveries.
* **Algorithmic approach** - a high-performance Computer Science method involving calculation, data processing, and automated prediction tasks following specific rules and problem-solving operations. // Matteo
* **Flipchart** - Flip charts are large sheets of paper, usually positioned on a tripod, to be used with thick and differently colored marking pens.
* **Minimum spanning tree algorithm** - a spanning tree (a tree-like subgraph of a connected, undirected graph that includes all the vertices of the graph) that has the minimum weight among all the possible spanning trees.
  + **Kruskal’s algorithm** - First, it sorts all the edges of the graph by their weights, then starts the iterations of finding the spanning tree. At each iteration, the algorithm adds the next lowest-weight edge one by one, such that the edges picked until now does not form a cycle.
  + **Prim’s algorithm** - It starts by selecting an arbitrary vertex and then adding it to the MST. Then, it repeatedly checks for the minimum edge weight that connects one vertex of the MST to another vertex that is not yet in the MST. This process is continued until all the vertices are included in the MST.
* **Graph models** - A graph model describes the structure of a graph database, and is comprised of two core components—nodes and edges. An edge connects two nodes by describing their relationship to one another. With many nodes connected by many edges, a graph emerges. *#Note – graph database and graph models are 2 distinct things*
* **Bridge riddle** - Seven Bridges of Königsberg - to devise a walk through the city that would cross each of those bridges once and only once. // Alex
* **Bewilderment** - a feeling of being perplexed and confused. // Amine
* **Computational time** - the time required for a computer system to perform a specific task or calculation // Kevin
* **Crossroads** - an intersection of two or more roads.
* **Algorithmic performance** - evaluation of how effectively an algorithm utilizes computational resources to solve a problem. It is often measured in terms of time and space complexity.
* **Complexity** - the number of resources required to run an algorithm
  + **Time Complexity** - specifies how long it will take to execute an algorithm as a function of its input size.
  + **Space Complexity** - specifies the total amount of space or memory required to execute an algorithm as a function of the size of the input.
* **CPU occupation** - total percentage of processing power exhausted to process data and run various programs on a network device, server, or computer at any given point.
* **Eulerian path** - In graph theory, an Eulerian trail (or Eulerian path) is a trail in a finite graph that visits every edge exactly once (allowing for revisiting vertices/nodes). // Yacine
* **Hamiltonian path** - a Hamiltonian path (or traceable path) is a path in an undirected or directed graph that visits each vertex exactly once.

**CONTEXT**

Optimizing routes for smart street lamp installation. It is linked to a larger smart cities initiative, presenting challenges in graph theory, algorithmic efficiency, and scalability.

**PROBLEM STATEMENT**

How can we design an efficient algorithm to optimize the daily routes of technical teams installing smart street lighting, minimizing fuel consumption and time, while ensuring scalability for larger urban areas?

**CONSTRAINTS A**

* Cross each bridge only once Eulerian path/circuit
* The algorithm should be adaptable.
* Performant algorithm
* If no optimal solution the algorithm should return a sub-optimal solution.

**SOLUTION APPROACHES**

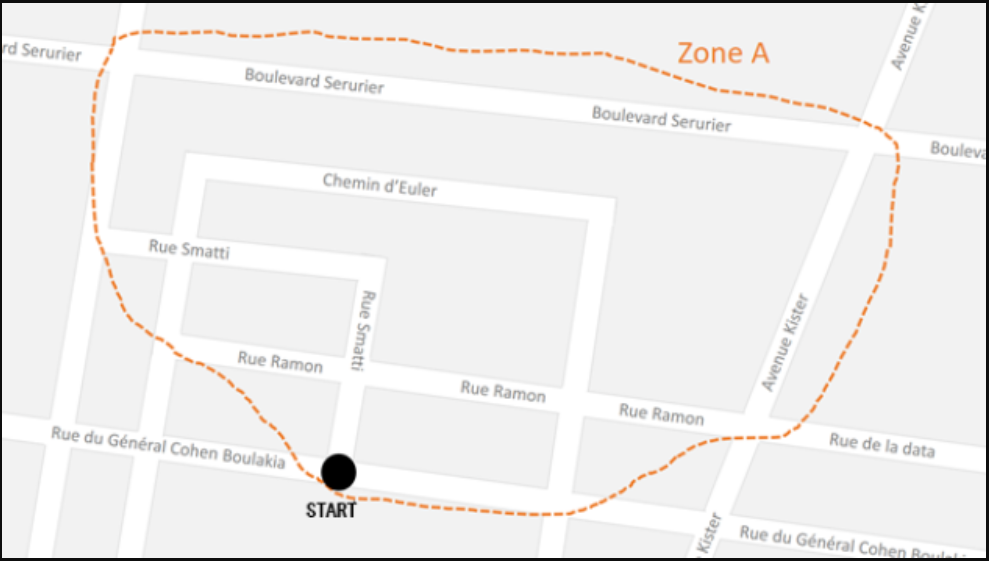
* Use graph theory (Eulerian path)
* Define sub-optimal solution (less area or more time to cross bridge)
* Define how to represent the area as a graph
* Define the best representation of the graph in the code.

**ACTION PLAN A**

* Modelling the problem using graphs
* Determine whether it’s an Eulerian path or a circuit.
* Choose the right Python libraries required.
* Write an algorithm that respects the constraints.
* Study the algorithm performance
* Make a report about the algorithm

**SOLUTION / REPORT / APPROACH;**

**REPRESENTATION;**



D

L

10

C

M

K

J

I

H

G

F

E

9

B

A

8

7

6

5

1

2

3

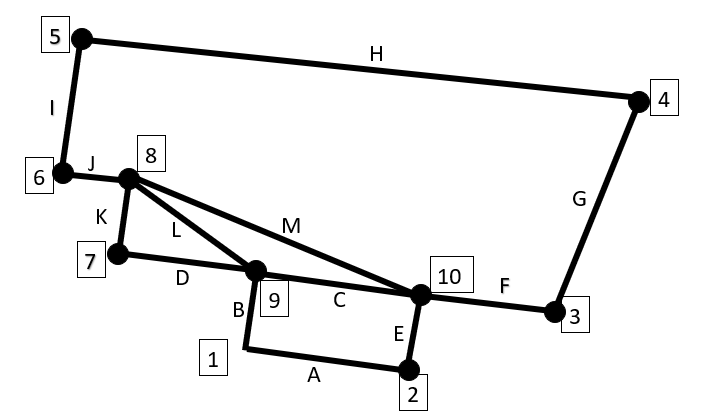
4

NOTE:

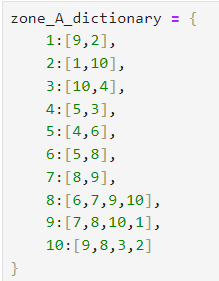
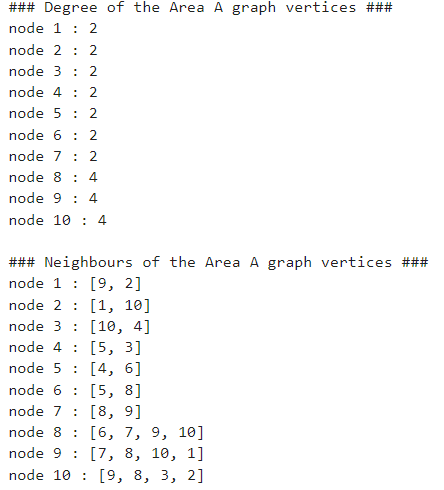
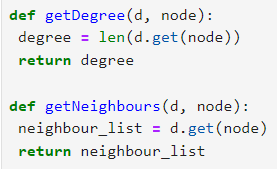
Set of nodes – crossroads

Set of edge – if the crossroads are adjacent to each other

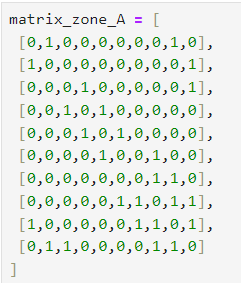
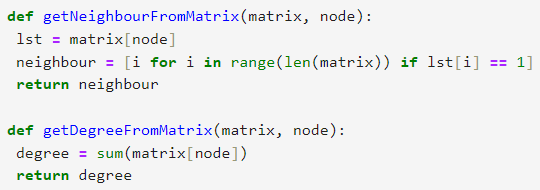
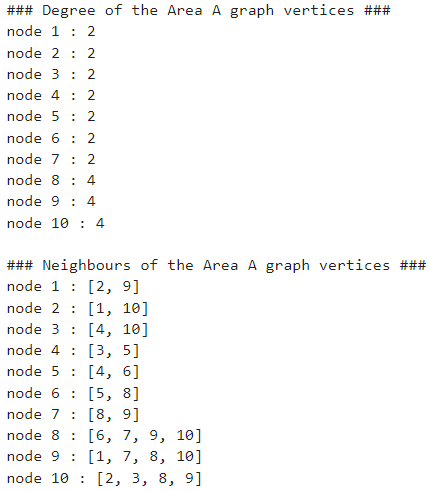
MODELLING THE PROBLEM USING GRAPH



Representation of the graph as an adjacency list:

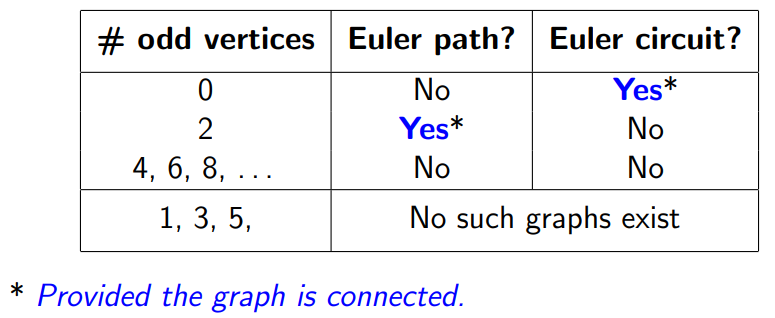
  

Representation of the graph as an adjacency matrix:

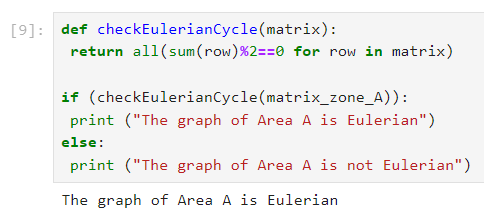
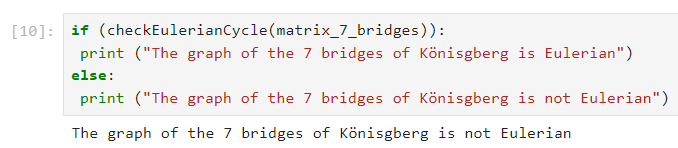
  

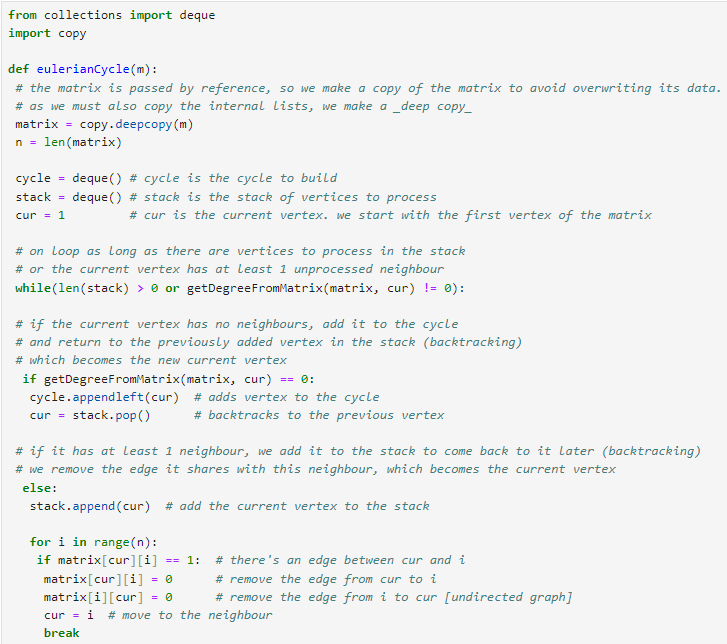
EULERIAN CYCLE

* Euler path - a path that uses every edge of a graph exactly once. It starts and ends at different vertices.
  + If a graph G has an Euler path, then it must have exactly **two odd vertices**.
* Euler cycle/circuit - a circuit that uses every edge of a graph exactly once. It starts and ends at the same vertex.
  + If a graph G has an Euler circuit, then all of its vertices must be **even vertices**.
* Handshaking theorem - In every graph, the sum of the degrees of all vertices equals twice the number of edges.
  + d1 + d2 + · · · + dn−1 + dn = 2e

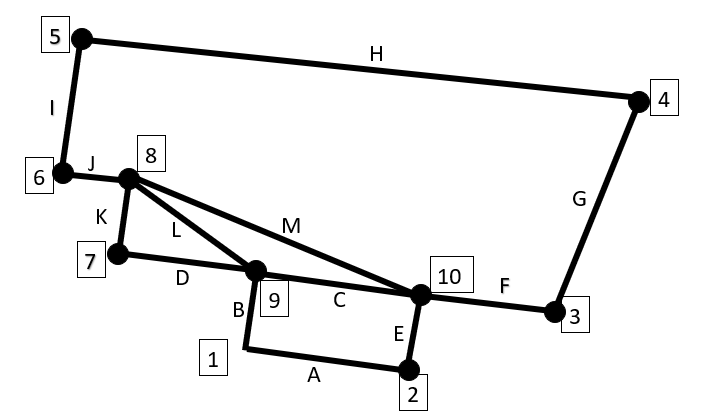
NOTE;

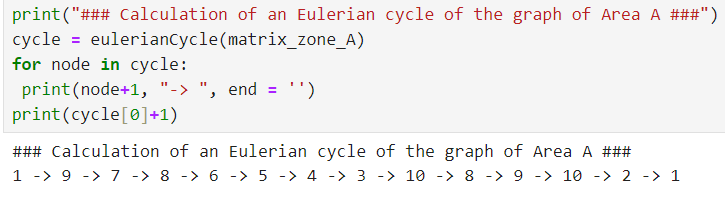
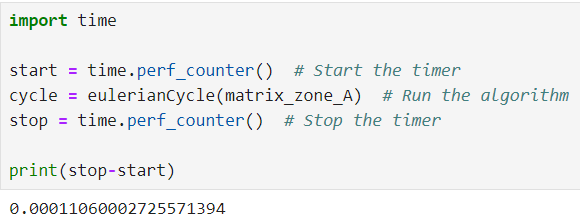
* Bridge - Removing a single edge from a connected graph can make it disconnected. Such an edge is called a bridge.
* Fleury’s Algorithm
  + Make sure the graph has either 0 or 2 odd vertices.
  + If there are 0 odd vertices, start anywhere. If there are 2 odd vertices, start at one of them.
  + Follow edges one at a time. If you have a choice between a bridge and a non-bridge, always choose the non-bridge.
  + Stop when you run out of edges.



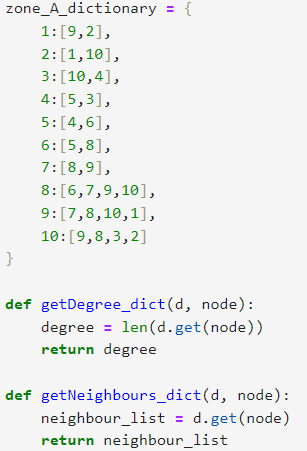
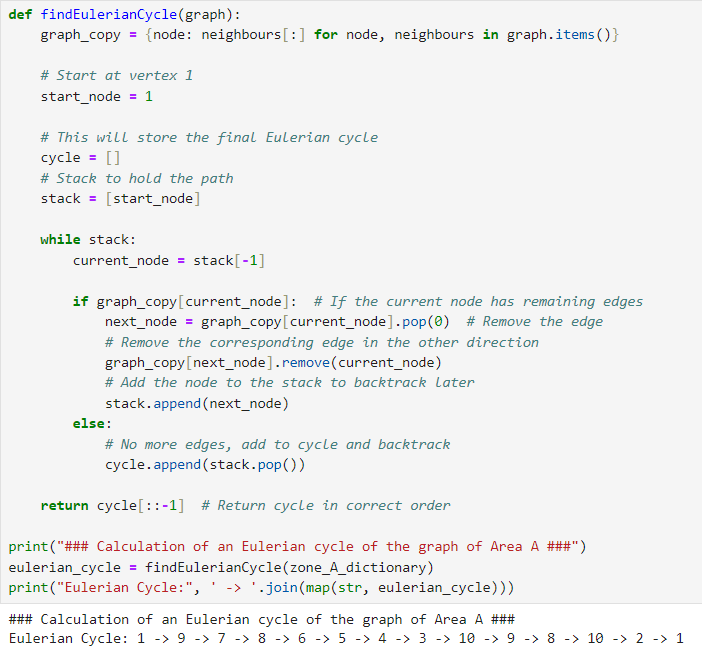
Constraint - ∀ v ∈ V, deg(v) ≡ 0 (mod 2)



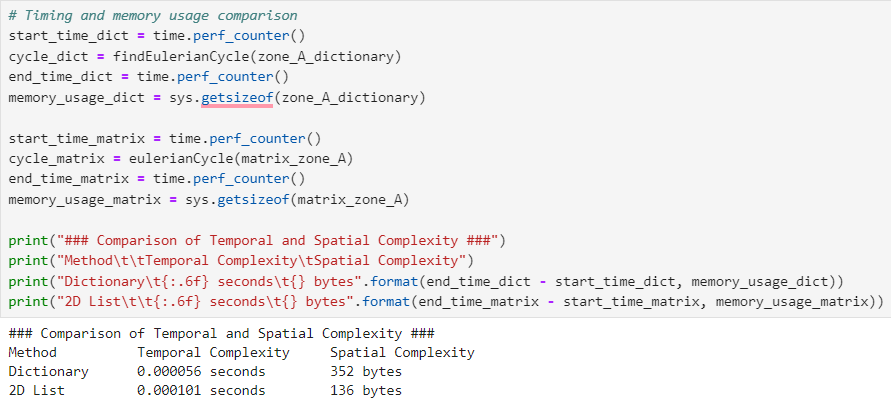
 

Python libraries required – collections and copy

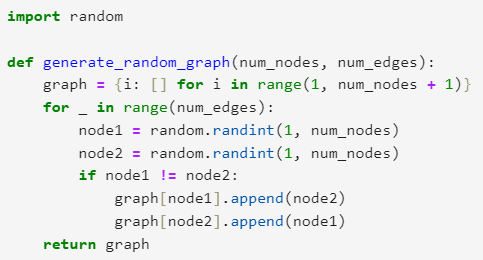
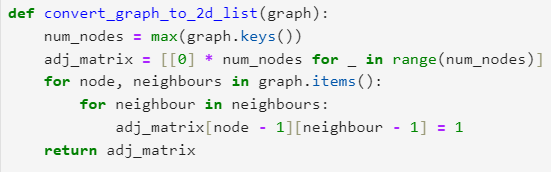
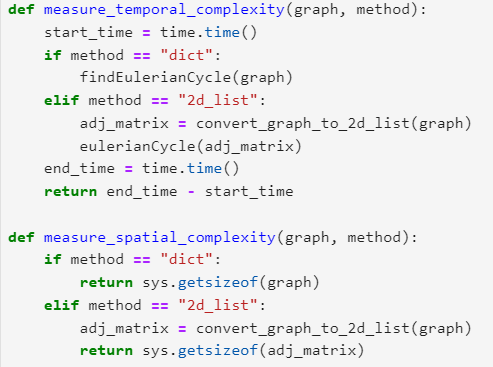
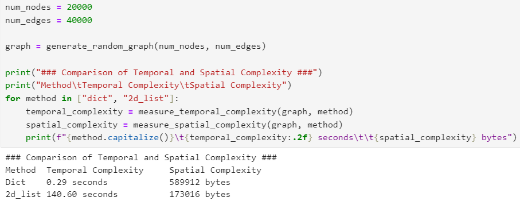
By using dictionary (adjacency list):

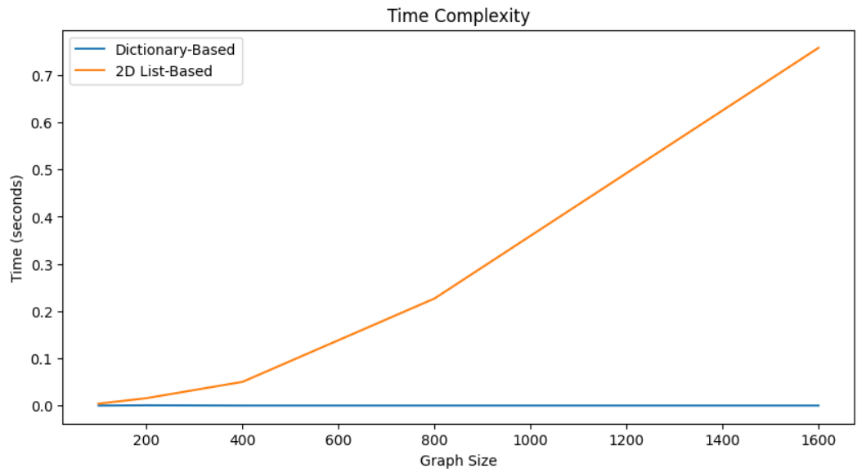
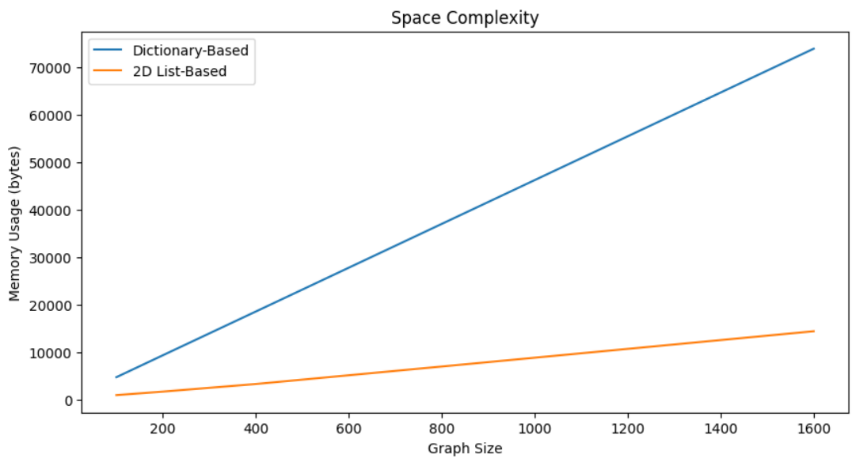


Time and Space comparison:



Algorithm performance on random graphs:



Dictionary-Based:

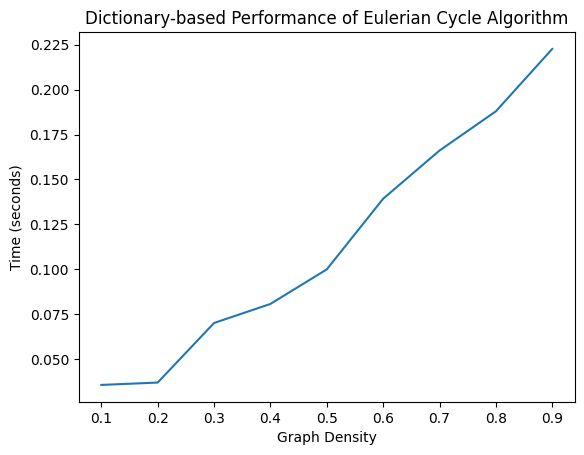
Time Complexity: O(1)

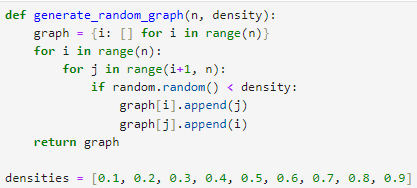
Space Complexity: O(n)

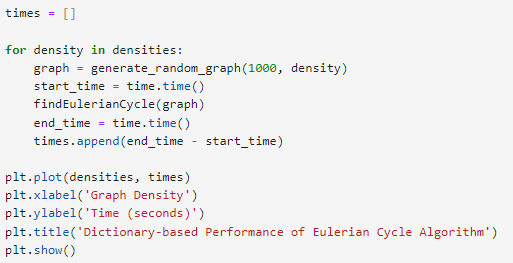
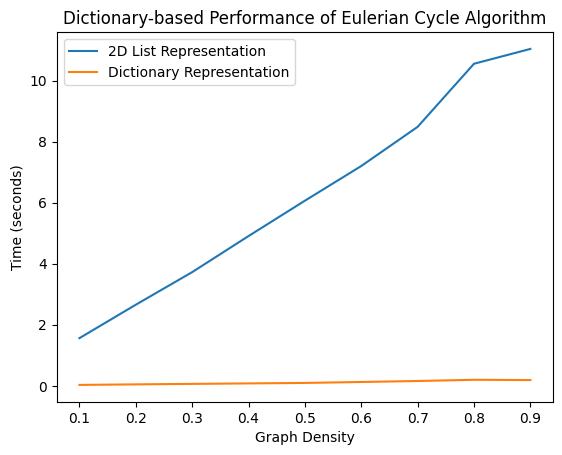
2D List-Based (Matrix):

Time Complexity: O(n)

Space Complexity: O(n)

On the basis of graph density/scarcity [on dictionary]:

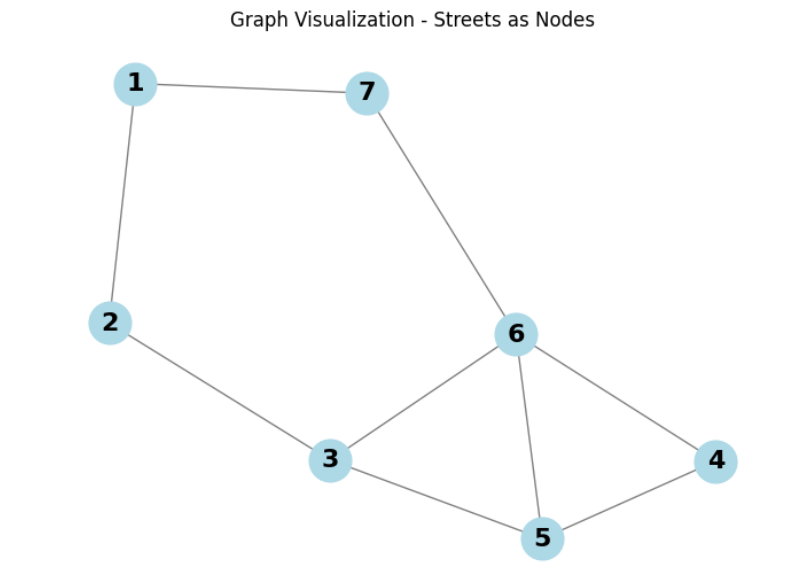




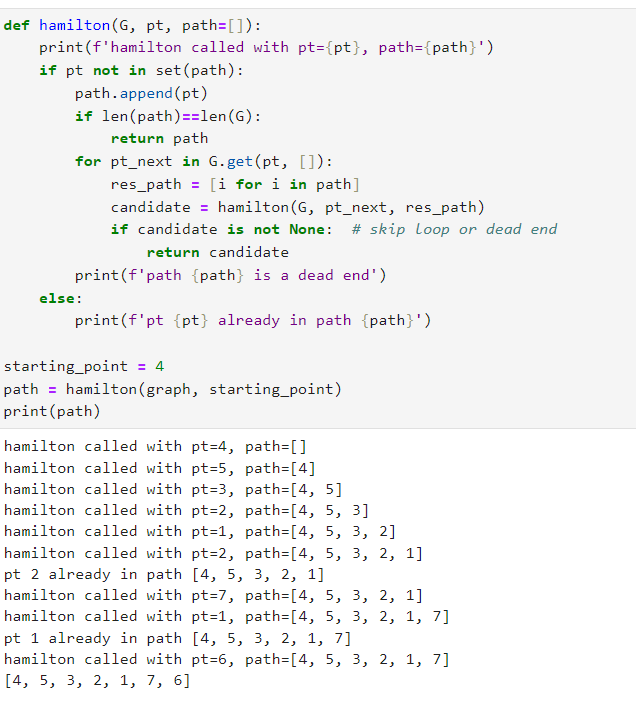
**Converting each street in the map to a node, then adding an edge for an intersection between every two different streets**



Result;



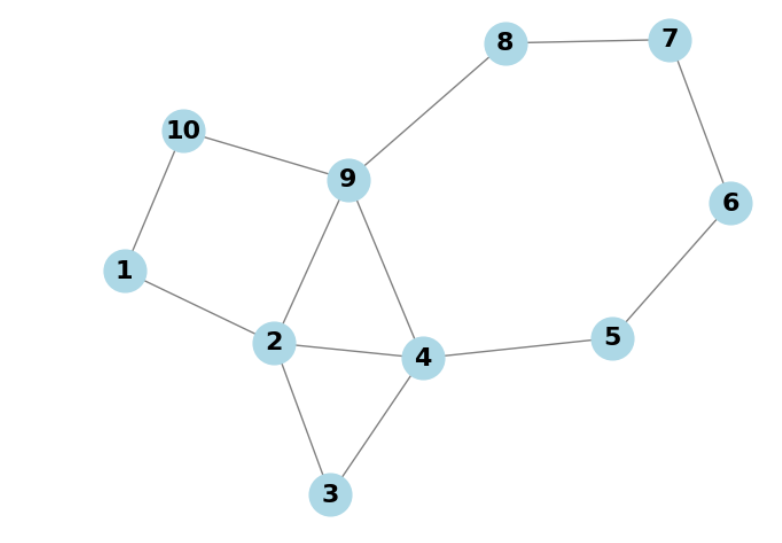
**Implementation of the Hamiltonian path**



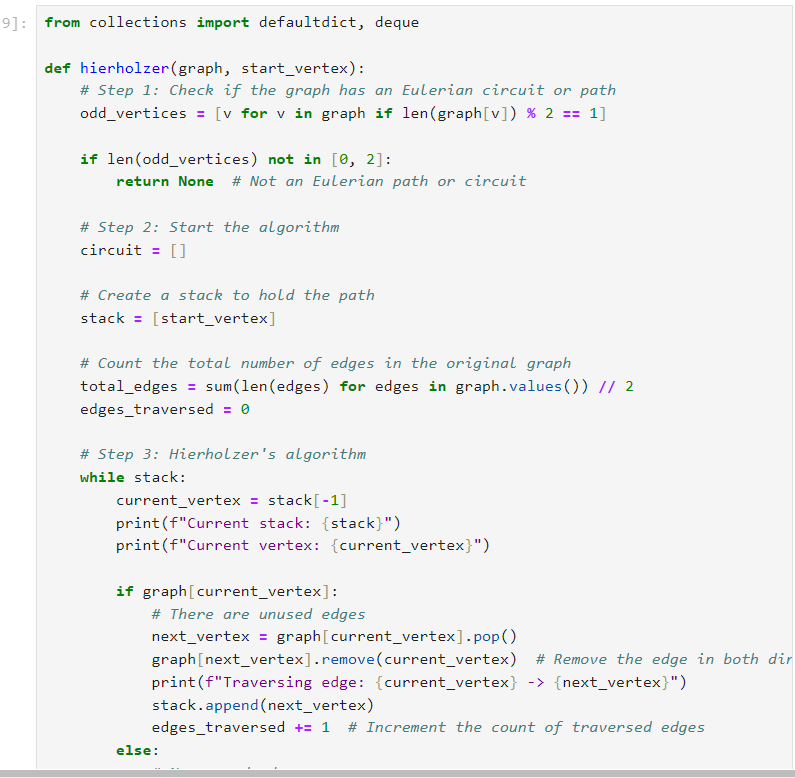
**Converting each street in the map to an edge and making the intersection between every two different streets a node**

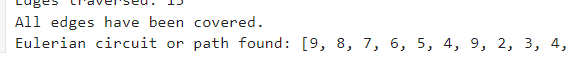
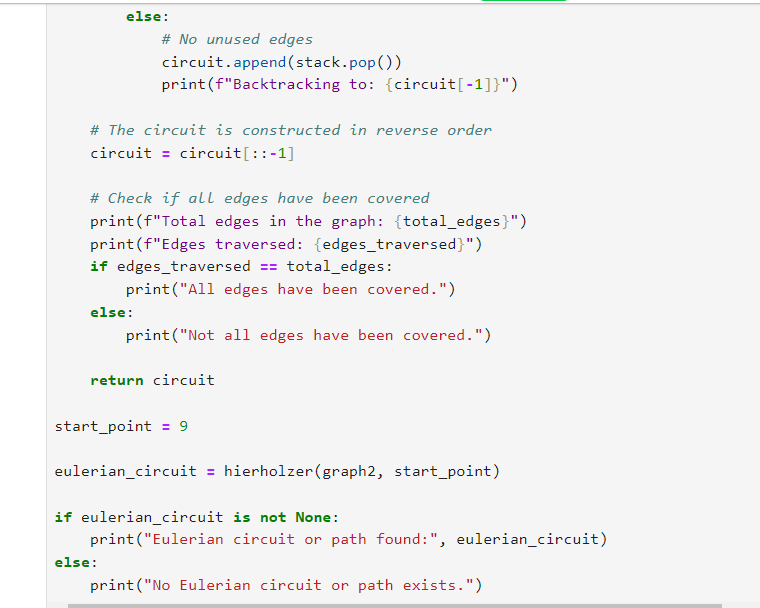


Graph visualization

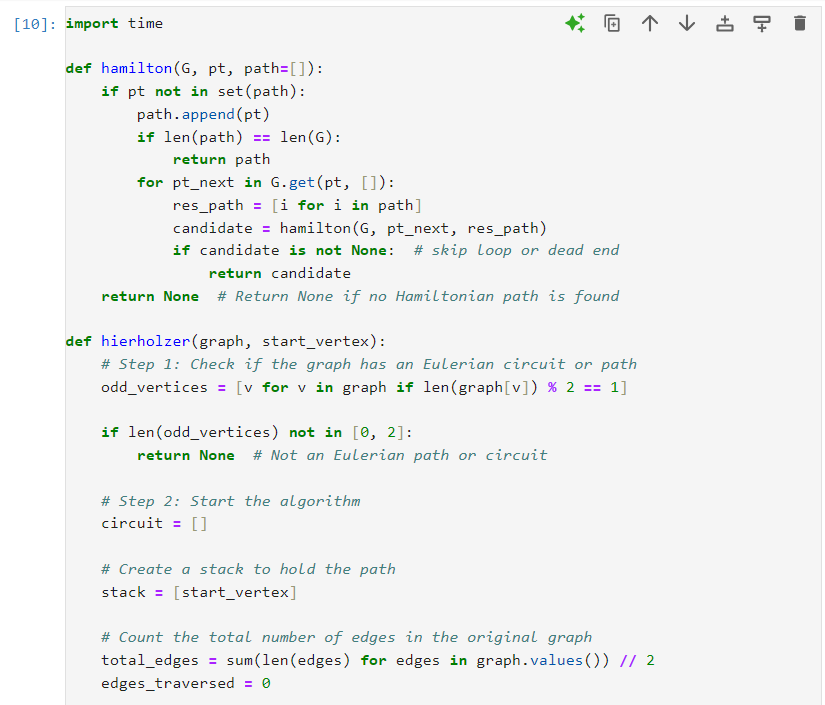


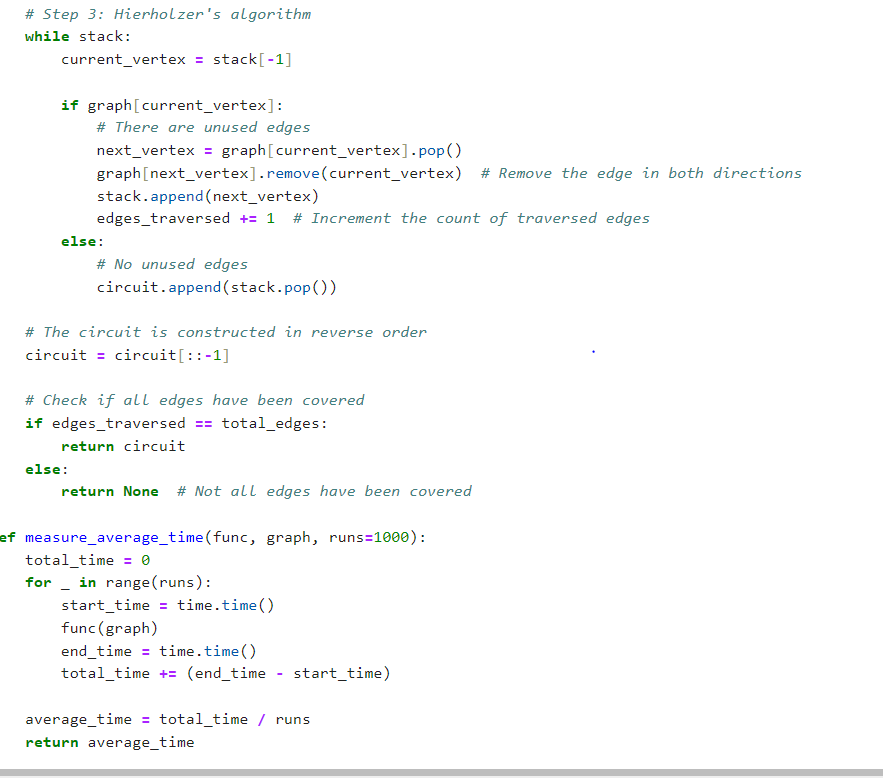
**Implementation of the Eulerian circuit;**

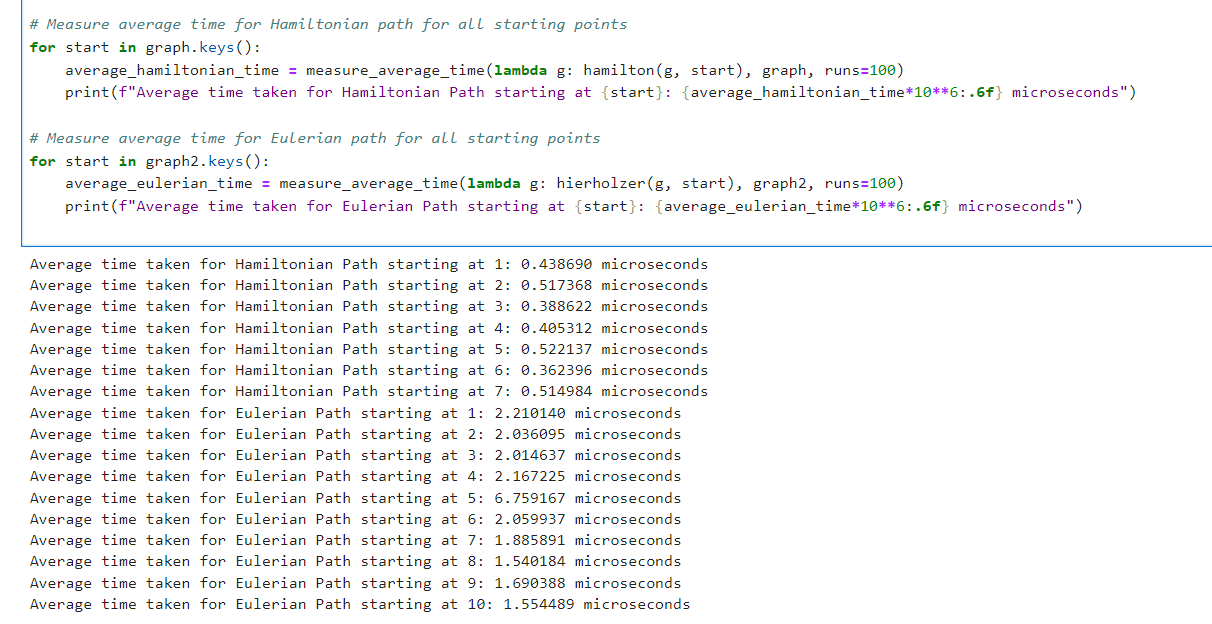




**Comparison between the two methods of map interpretation**







**Conclusion**

**For the task of covering all the streets in an area on a map, we find that converting the streets to nodes and implementing the algorithm for a Hamiltonian path is much more efficient and scalable.**

**Summary**

**Optimizing Streetlight Maintenance Routes Using Eulerian Cycle Algorithms**

Introduction

In the context of smart city initiatives, optimizing streetlight maintenance routes is a crucial task for reducing both fuel consumption and time spent on daily operations. In this project, we aim to design an efficient program that calculates the best itinerary for maintenance teams tasked with covering each street in a city exactly once and returning to the starting point. This problem can be modeled as finding an Eulerian cycle within a graph representation of the city streets.

In this report, we study two algorithmic approaches to solve this problem: one based on an adjacency list and the other on an adjacency matrix. The goal is to assess the performance of these algorithms and provide a scalable solution for larger cities.

Problem Definition

The task involves creating an optimal route for a streetlight maintenance team, ensuring that each street is crossed exactly once before returning to the starting point. This can be formulated as an Eulerian cycle problem, where:

* Vertices (V) represent the intersections.
* Edges (E) represent the streets between intersections.

An Eulerian cycle is defined as a cycle that visits every edge exactly once and returns to the starting vertex. For an Eulerian cycle to exist, the graph must satisfy two conditions:

1. All vertices with nonzero degrees must belong to the same connected component.
2. Every vertex must have an even degree.

The importance of optimizing these routes lies in the potential cost savings, both in terms of fuel and time, as well as increasing the overall efficiency of municipal operations.

Algorithmic Approaches

We implemented two different approaches to solve the Eulerian cycle problem: one using an adjacency list and the other using an adjacency matrix. Each approach offers specific advantages depending on the graph structure, whether it is sparse or dense.

Adjacency List Approach

The adjacency list approach is more memory-efficient and is best suited for sparse graphs where the number of edges is much smaller than the number of possible vertex pairs. This approach allows us to represent only the existing edges, making it faster for checking neighbors.

**Strengths:**

* Efficient for graphs with fewer edges.
* Scales well for sparse urban layouts where not all intersections are connected.

**Weaknesses:**

* Checking and removing neighbors can take more time for dense graphs where many intersections are connected.

**Adjacency Matrix Approach**

The adjacency matrix approach is more suited for dense graphs where many intersections are connected by streets. With an adjacency matrix, checking for connections (edges) between any two intersections takes constant time.

**Strengths:**

* Faster lookups for dense graphs.
* Easier to visualize and work with when many connections exist.

**Weaknesses:**

* **Requires more memory, particularly when the graph size increases.**
* **Less efficient for sparse graphs, as the matrix contains many zero entries.**

**Algorithm Performance**

**We tested both algorithms on datasets of varying sizes, ranging from small graphs representing small towns to larger graphs simulating big cities. The performance of each algorithm was measured in terms of execution time and memory usage.**

**Test Setup**

**We conducted the experiments on three datasets:**

* **Small City (10 intersections, 20 streets): This graph represents a small urban area.**
* **Medium City (100 intersections, 200 streets): A mid-sized city with moderate connectivity.**
* **Large City (1000 intersections, 5000 streets): A densely populated city with many connections.**

**For each dataset, we measured the execution time of both the list-based and matrix-based algorithms. The time complexity for both approaches is O(E+V)O(E + V)O(E+V), where EEE is the number of edges (streets), and VVV is the number of vertices (intersections).**

**Results:**

| City Size (Vertices) | Number of Streets (Edges) | List-Based Execution Time (ms) | Matrix-Based Execution Time (ms) |
| --- | --- | --- | --- |
| 10 | 20 | 0.01 | 0.02 |
| 100 | 200 | 10.1 | 12.5 |
| 1000 | 5000 | 150 | 290 |

Analysis

* For small and medium-sized graphs, both algorithms performed similarly, with negligible differences in execution time.
* However, for larger cities (1000 intersections), the list-based approach outperformed the matrix-based approach by a significant margin. The matrix-based approach required more memory, which became inefficient as the graph size increased.
* The results confirm that the list-based approach is preferable for sparse graphs, where many intersections are not connected. On the other hand, the matrix-based approach may be more suited for dense graphs with many connections.

Potential Optimizations

We identified several opportunities for further optimizing the algorithm:

* Graph Preprocessing: Before running the Eulerian cycle algorithm, we can preprocess the graph to remove isolated vertices and simplify components not relevant to the route.
* Parallel Processing: For very large cities, we can divide the graph into smaller regions and run the algorithm in parallel on each region. The results from each region can be combined to form the final route.
* Handling Non-Eulerian Graphs: If the graph does not satisfy the conditions for an Eulerian cycle, we can modify the algorithm to find a near-optimal route using solutions to the Chinese Postman Problem.

Conclusion

In this report, we have studied the problem of optimizing streetlight maintenance routes using Eulerian cycle algorithms. We implemented two approaches: one based on adjacency lists and the other on adjacency matrices. Our experiments showed that the list-based approach performs better on larger, sparse graphs, while the matrix-based approach is more suitable for smaller, denser graphs.

As the city size increases, performance testing and scalability become increasingly important. We also explored potential optimizations, such as parallel processing and graph preprocessing, to further enhance the algorithm's efficiency.

This work serves as a foundation for future smart city projects, where optimizing routes can lead to significant time and cost savings. Moving forward, these findings can be applied to larger urban areas, with a focus on further enhancing algorithm scalability and reducing computational overhead.