**KEYWORDS A**

* **Smart cities** - A smart city is a place where traditional networks and services are made more efficient with the use of digital solutions for the benefit of its inhabitants and business.
* **Itinerary (on a trial basis)** - a travel document recording a route or journey.
* **Optimal route** - 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, as well as time windows for deliveries.
* **Algorithmic approach** - a high-performance method in Computer Science that involves calculation, data processing, and automated prediction tasks following specific rules and problem-solving operations.
* **Flipchart** - Flip charts are large sheets of paper, usually positioned on a tripod, to be used with thick and differently coloured 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 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 together by describing their relationship to one another. With many nodes connected by many edges, a graph emerges.
* **Bridge riddle** - Seven Bridges of Königsberg - to devise a walk through the city that would cross each of those bridges once and only once.
* **Bewilderment** - a feeling of being perplexed and confused.
* **Computational time** - the time required for a computer system to perform a specific task or calculation
* **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 amount 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).
* **Hamiltonian path** - a Hamiltonian path (or traceable path) is a path in an undirected or directed graph that visits each vertex exactly once.

**CONTEXT A**

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 A**

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.

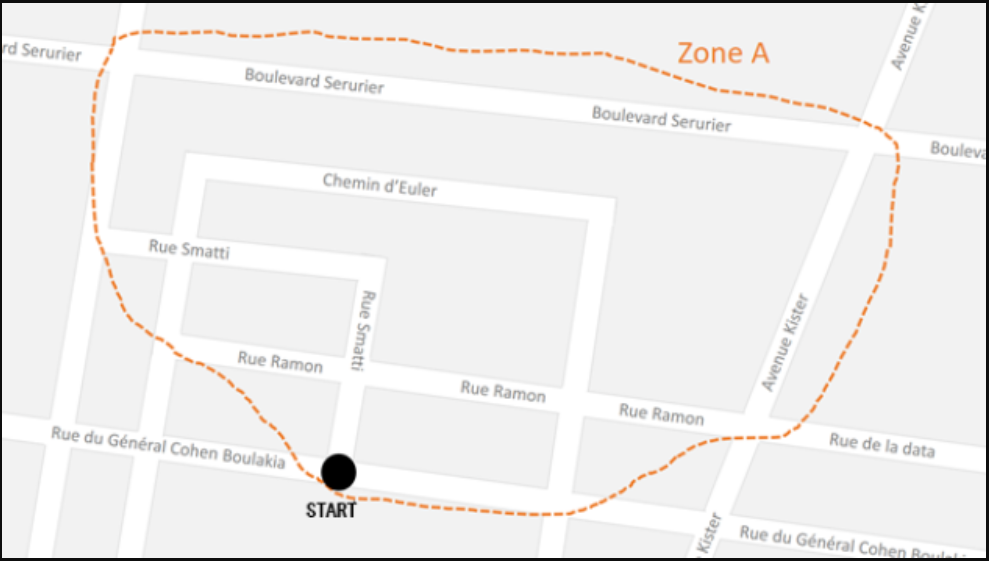
**SOLUTION APPROACHES A**

* 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 a 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

**MY SOLUTION / REPORT / APPROACH A**



D

L

10

C

M

K

J

I

H

G

F

E

9

B

A

8

7

6

5

1

2

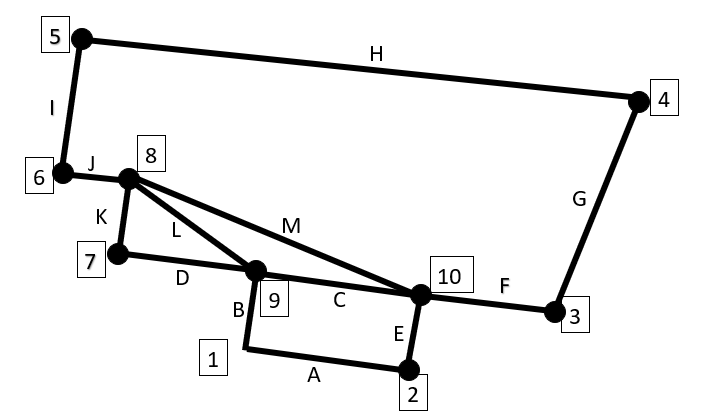
3

4

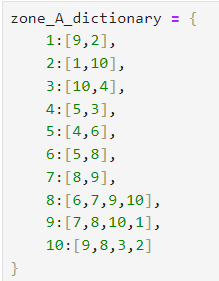
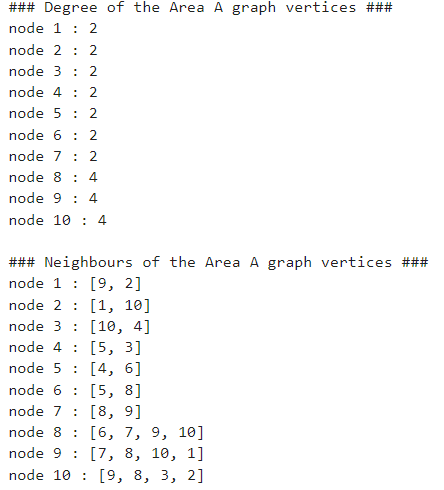
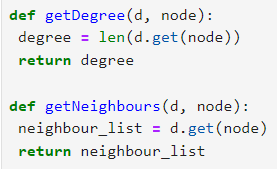
Set of nodes – crossroads

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

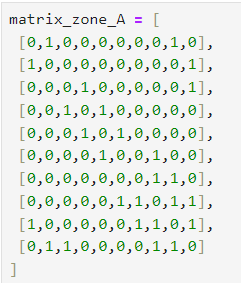
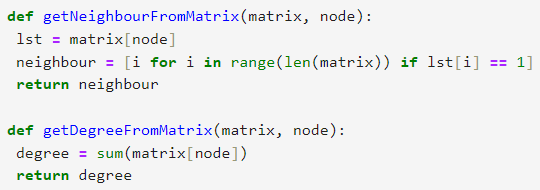
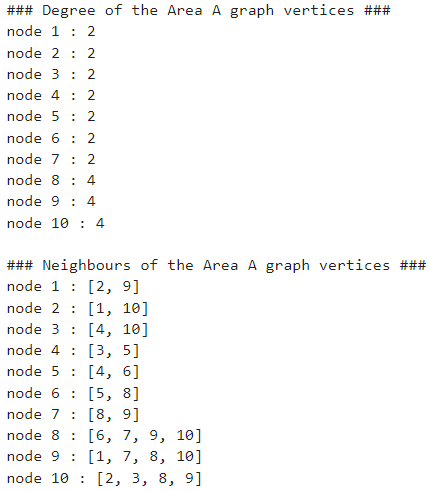
MODELLING THE PROBLEM USING GRAPH



Representation of the graph as an adjacency list:

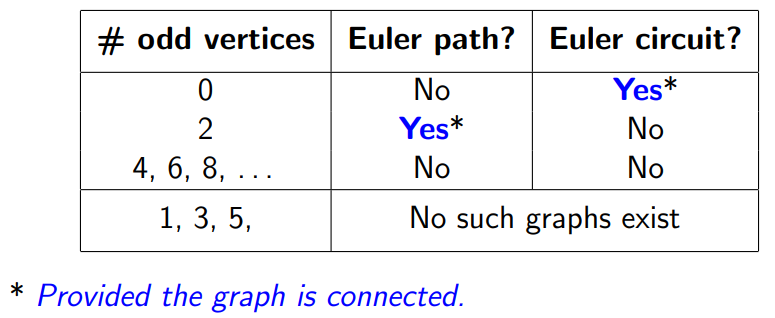
  

Representation of the graph as an adjacency matrix:

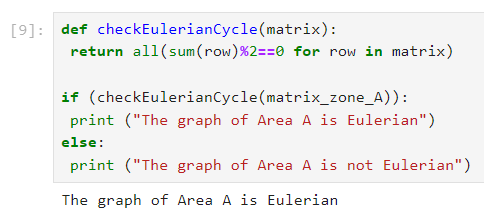
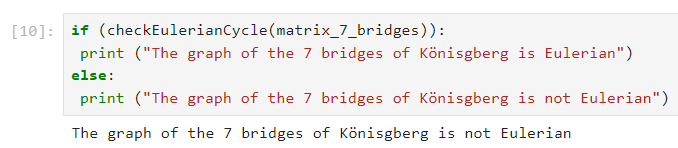
  

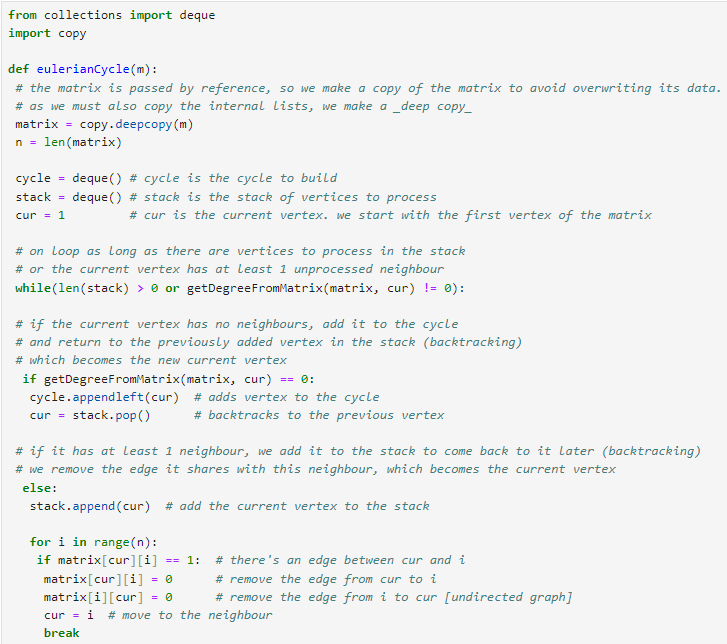
EULERIAN CYCLE

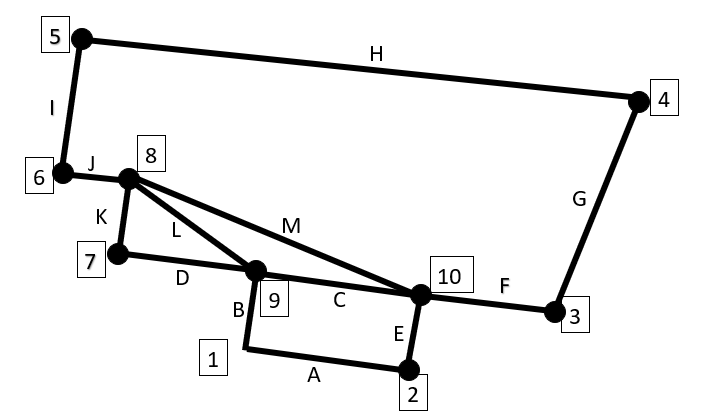
* Euler path - a path that uses every edge of a graph exactly once. It starts and ends at different vertices.
  + If a graph G has an Euler path, then it must have exactly **two odd vertices**.
* Euler cycle/circuit - a circuit that uses every edge of a graph exactly once. It starts and ends at the same 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

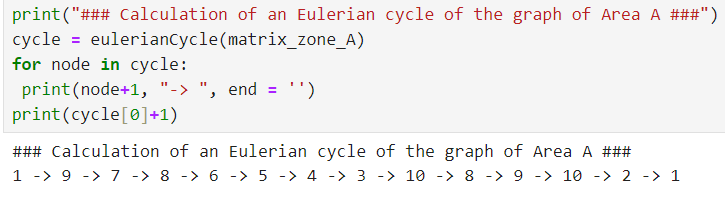
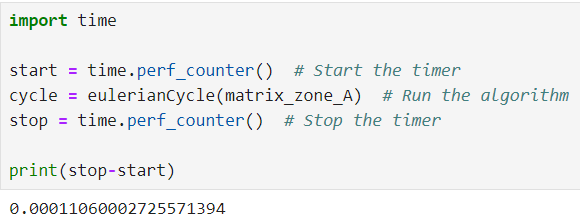


* Bridge - Removing a single edge from a connected graph can make it disconnected. Such an edge is called a bridge.
* Fleury’s Algorithm
  + Make sure the graph has either 0 or 2 odd vertices.
  + If there are 0 odd vertices, start anywhere. If there are 2 odd vertices, start at one of them.
  + Follow 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.

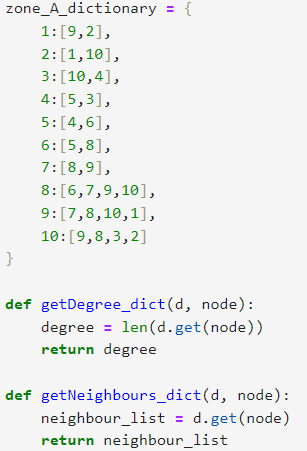
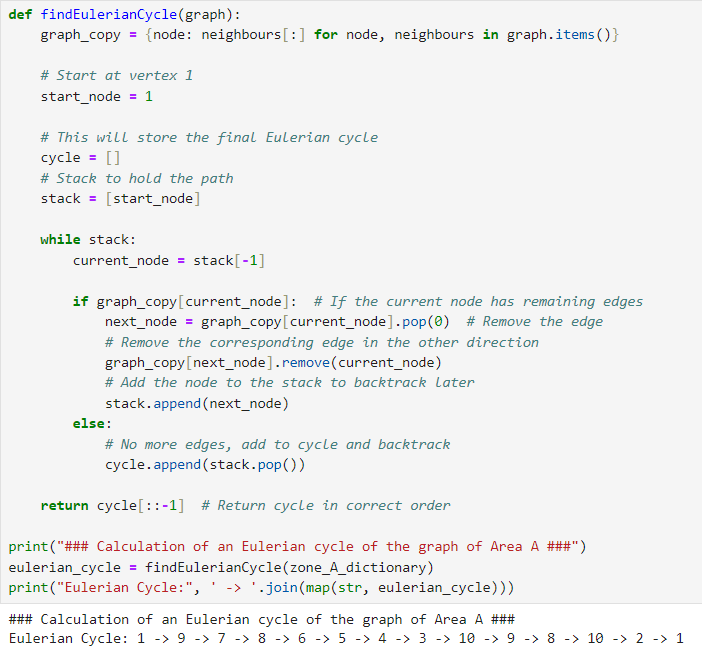
 Constraint - ∀ v ∈ V, deg(v) ≡ 0 (mod 2)



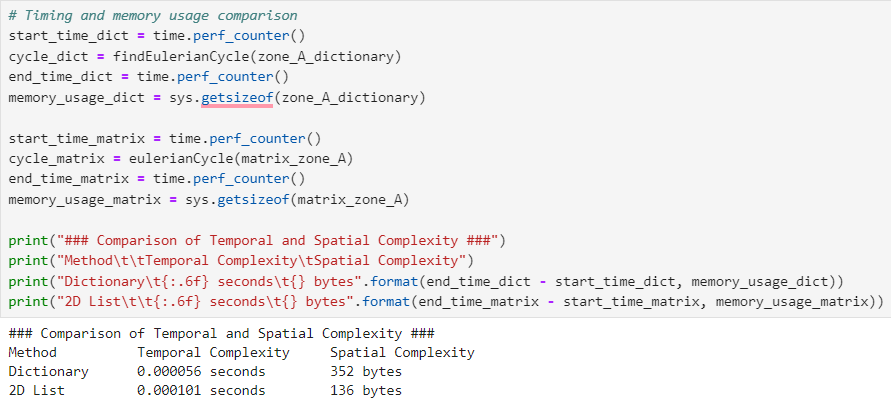
 

Python libraries required – collections and copy

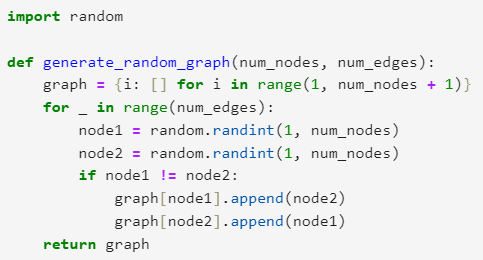
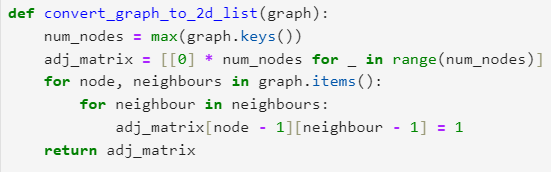
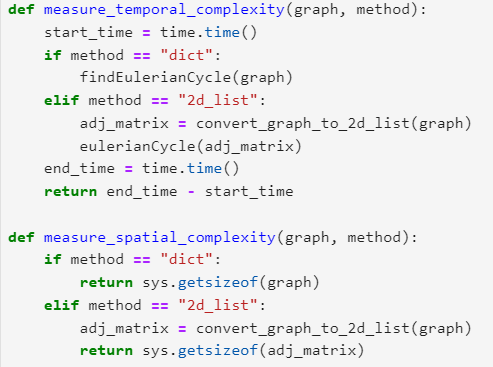
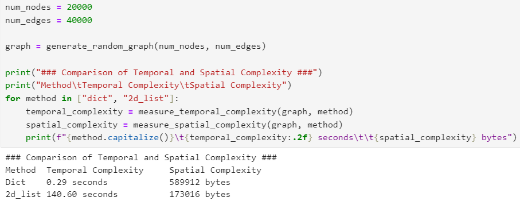
By using dictionary (adjacency list):

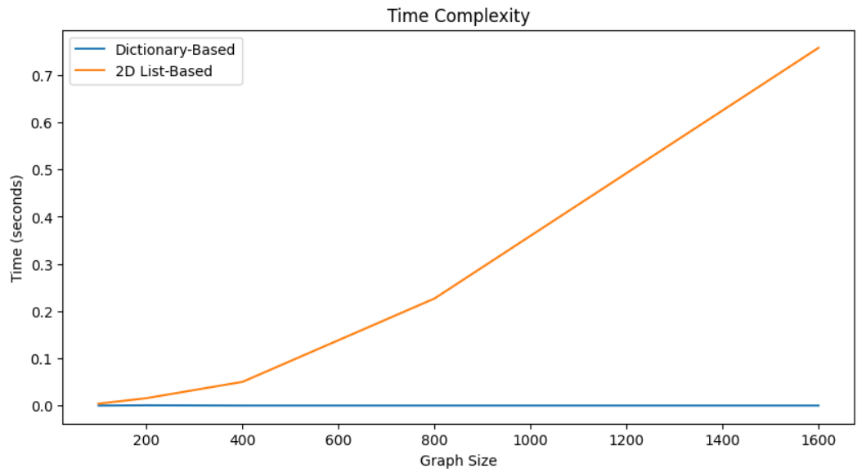
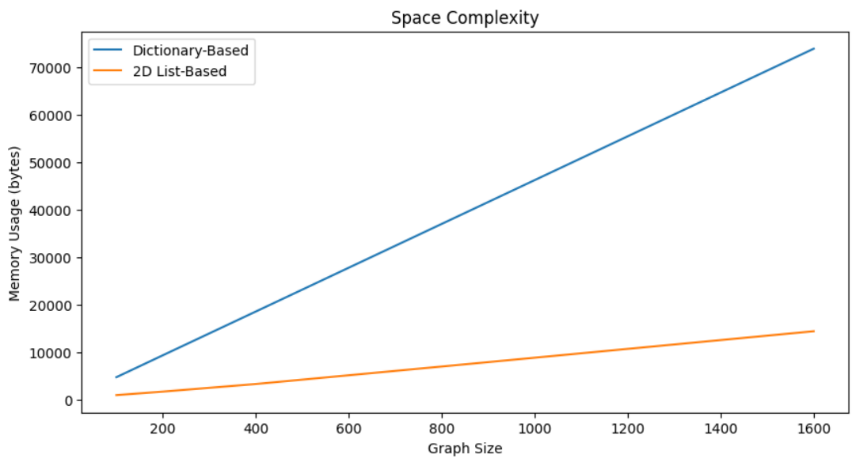


Time and Space comparison:



Algorithm performance on random graphs:



Dictionary-Based:

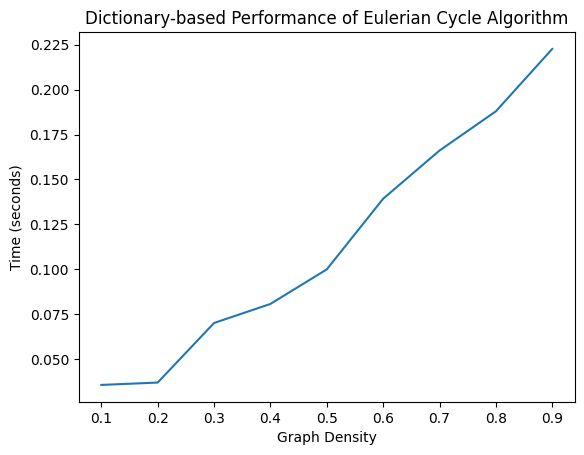
Time Complexity: O(1)

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]:

