Analytical Report

Assignment 4 – Scheduling and Optimization of City Service Tasks (SCC + DAG Paths)
Author: ***Kassymova Tomiris***

1. Introduction

Modern cities rely on interdependent digital services—street cleaning, road repairs, camera and sensor maintenance, and internal analytics pipelines. These dependencies often form directed graphs where each task depends on others.
Such networks may contain cycles (mutual dependencies) that must be detected and collapsed, after which an acyclic structure can be optimized for time-critical execution.

This assignment integrates two major algorithmic topics:

1. **Strongly Connected Components (SCC) and Topological Ordering, and**
2. **Shortest and Longest Paths in Directed Acyclic Graphs (DAGs).**

The objective is to:

- **Identify all cyclic task clusters (SCCs) using Tarjan's algorithm.**
- **Build the condensation DAG of components.**
- **Compute a valid topological order of that DAG.**
- **Using this order, compute single-source shortest and longest (critical) paths to plan optimal execution times.**

All implementations were written in Java 17 within a modular Maven project using packages graph, model, and util.
JSON input files were parsed with Jackson, timing measured with System.nanoTime(), and operation metrics collected via a custom Metrics class.

2. Input Data and Experimental Setup

Nine datasets were generated under /data/ to model city-task dependencies of varying size and density.

Category	Nodes (V)	Edges (E)	Structure	Purpose
Small	6 – 10	10 – 15	1–2 cycles or pure DAG	Verify correctness
Medium	10 – 20	25 – 40	Mixed, several SCCs	Observe scaling
Large	20 – 50	80 – 120	Dense with multiple clusters	Performance benchmark

Each file (small_1.json, medium_2.json, etc.) contained:

```
{"nodes": ["A","B","C", ...],
 "edges":[{"from":"A","to":"B","weight":2.0}, {"from":"B","to":"C","weight":1.5}]}
```

All algorithms operated on edge weights as duration or cost values.
When multiple original edges connected the same SCCs, the minimum weight was taken in the condensation graph.

The metrics recorded:

- **dfsVisits, edgesExplored → SCC stage**
- **pushes, pops → Topological Sort**
- **relaxations → DAG Shortest Paths**
  Execution time was measured separately for each stage.

3. Summary of Algorithm Results

Dataset	#SCC	SCC Time (ns)	Topo Time (ns)	DAG-SP Time (ns)	DFS Visits	Pushes	Relaxations
small_1	2	12 300	6 200	4 000	8	5	12
small_2	1	11 800	5 700	3 900	9	4	10
medium_1	4	47 000	11 000	8 200	15	9	27
medium_2	6	61 500	14 200	9 800	18	11	32
large_1	9	210 000	65 000	38 000	45	26	117

All stages executed successfully across datasets of increasing size.
SCC runtime grew roughly linearly with E (≈ O(V + E)).
Topological sorting remained negligible in cost.
DAG shortest-path relaxations dominated only in the largest dense graphs.

4. Theoretical Background

4.1 Strongly Connected Components (Tarjan)

Tarjan's algorithm performs a single DFS traversal, assigning each vertex a discovery index and low-link value.
When a node's low-link = its discovery index, an SCC is formed.
The algorithm runs in O(V + E) time and is efficient for detecting cyclic dependencies in directed networks (Cormen et al., 2022).

4.2 Topological Ordering (Kahn)

Kahn's algorithm repeatedly removes nodes with zero in-degree, appending them to the ordering queue.
It ensures that every edge (u → v) has u before v.
Complexity = O(V + E), highly suitable for DAGs generated after SCC compression (Sedgewick & Wayne, 2011).

4.3 Shortest and Longest Paths in DAGs

After obtaining a topological order, dynamic programming computes shortest and longest paths efficiently in O(V + E) time by relaxing edges in topological sequence.
This is analogous to the single-source shortest path (DP) formulation described in Kleinberg & Tardos (2006).
The longest path reveals the critical chain of dependent tasks (maximum cumulative weight).

5. Comparison: Theory vs Practice

Observed Performance

- **SCC: linear scalability confirmed; dense graphs increased DFS edge scans.**
- **Topological Sort: minimal overhead (< 10 μs for small graphs).**
- **Shortest Path DP: proportional to number of edges in the condensation DAG.**

Memory Usage

All algorithms store adjacency lists and small auxiliary arrays, yielding O(V + E) space.
Condensation significantly reduces graph size when many cycles exist.

Implementation Complexity

Tarjan's algorithm required careful stack management but proved stable.
Kahn's and DP modules were straightforward to implement once the condensation structure was available.

Scalability and Bottlenecks

Graph Condition	Dominant Stage	Reason
Sparse DAG (no cycles)	Topo + DP	Few DFS calls
Dense cyclic graph	SCC	Multiple DFS visits per cycle
Many components ($k \gg V$)	DP Stage	Many relaxations in condensed graph

6. Conclusions

This project demonstrated a complete workflow for cyclic task detection, scheduling, and optimization in smart-city operations:

Phase	Algorithm	Use Case
Cycle Detection	Tarjan SCC	Collapse mutual dependencies → simpler DAG
Execution Order	Kahn Topo	Enforce dependency precedence
Optimization	DAG DP	Find fastest and critical execution chains

Key Insights:

- **Cyclic interdependencies can be safely compressed without losing task reachability.**
- **Topological ordering enables deterministic scheduling for all city-service modules.**

- **Dynamic programming on DAGs is far more efficient than generic shortest-path algorithms (no need for Dijkstra/Bellman-Ford).**
- **Runtime and operation counts grow linearly with graph size, confirming theoretical complexity.**

7. References

1. **Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022).** ***Introduction to Algorithms*** **(4th ed.). MIT Press.**
2. **Sedgewick, R., & Wayne, K. (2011).** ***Algorithms*** **(4th ed.). Addison-Wesley.**
3. **Kleinberg, J., & Tardos, É. (2006).** ***Algorithm Design.*** **Pearson Education.**
4. **Weiss, M. A. (2013).** ***Data Structures and Algorithm Analysis in Java*** **(4th ed.). Pearson.**
5. **Orlin, J. B., Ahuja, R. K., & Magnanti, T. L. (1993).** ***Network Flows: Theory, Algorithms, and Applications.*** **Prentice-Hall.**
6. **Oracle (2025).** ***Java Platform, Standard Edition 17 API Specification.***