Assignment 4

Goal

Consolidate two course topics in one practical case (“Smart City / Smart Campus Scheduling”):

1. Strongly Connected Components (SCC) & Topological Ordering
2. Shortest Paths in DAGs

Scenario

You receive datasets for city-service tasks (street cleaning, repairs, camera/sensor maintenance) and internal analytics subtasks. Some dependencies are cyclic (detect & compress them), others are acyclic (plan optimally). Separate subtasks use standard DP patterns.

What to implement

1. Graph tasks

1.1 SCC (Tarjan or Kosaraju)

- **Input:** directed dependency graph tasks.json (format uploaded).
- **Output:** the list of strongly connected components (each as a list of vertices) and their sizes.
- Build the **condensation graph** (a DAG of components).

1.2 Topological Sort

- Compute a topological order of the condensation DAG (Kahn or DFS variant).
- Output a valid order of components and a derived order of original tasks after SCC compression.

1.3 Shortest Paths in a DAG

- The DAG uses either **edge weights** or **node durations** (choose one; document your choice in the README).
- Implement:
    - Single-source shortest paths on the DAG.
    - Longest path on the DAG (via sign inversion or max-DP over topo order).
- **Output:** critical path (longest), its length; for shortest distances from a given source; reconstruct one optimal path.

2. Dataset Generation (Graph Data)

Each student must **generate multiple datasets** to test the algorithms on different graph structures and sizes.
All datasets should be stored under **/data/.**

Category	Nodes (n)	Description	Variants
Small	6–10	Simple cases, 1–2 cycles or pure DAG	3
Medium	10–20	Mixed structures, several SCCs	3
Large	20–50	Performance and timing tests	3

Total: 9 datasets per student.

- Use different density levels (sparse vs dense) and include both cyclic and acyclic examples.
- Include at least one graph with multiple SCCs.
- Document each dataset briefly in the report (number of vertices, edges, whether cyclic or DAG).

Instrumentation

- A common Metrics interface (operation counters + time).
- Timing via System.nanoTime().
- Counters: DFS visits/edges (SCC), pops/pushes (Kahn), relaxations (DAG-SP)

Code quality

- Packages: graph.scc, graph.topo, graph.dagsp.
- Comment key steps; Javadoc for public classes.
- JUnit tests: small deterministic cases + edge cases.

Code (GitHub):

- Builds from a clean clone.
- Run instructions in README.md.
- Tests under src/test/java.

Data (/data) + optional test generators.

Report (PDF or README.md):

- **Data summary:** sizes, weight model.

- **Results:** per-task tables (metrics, time, n).
- **Analysis:**
    - SCC/Topo/DAG-SP: bottlenecks; effect of structure (density, SCC sizes).
- **Conclusions:** when to use each method/pattern; practical recommendations.

Grading (100%)

Algorithmic correctness (SCC, Topo, DAG-SP) – **55%**

- SCC + Condensation + Topo – 35%
- DAG Shortest + Longest – 20%

Report & analysis – **25%**

Code quality & tests – **15%** (readability, modularity, JUnit, reproducibility)

Repo/Git hygiene – **5%** (README, clear structure)