

Assignment 4 – Smart City / Smart Campus Scheduling

Variant: **Kosaraju SCC + Topological Sort + Shortest and Longest Path in DAG**

Student Information

Student:Mirasbek Idiatulla
Group:SE-2435
Algorithm Chosen: Kosaraju

Goal

This project combines two major algorithmic topics: **Strongly Connected Components (SCC) & Topological Ordering**, and **Shortest Paths in Directed Acyclic Graphs (DAGs)**. The goal is to simulate a scheduling system for Smart City or Smart Campus operations, where various maintenance or service tasks depend on one another.

The implemented algorithms detect cyclic dependencies (SCCs), build a condensed DAG of tasks, determine valid execution order (Topological Sort), and compute both shortest and longest paths to optimize scheduling time.

Data Summary

A total of nine datasets were generated and tested. Each dataset represents a directed weighted graph with varying density and structure.

	dataset ∇	÷	scc_count ∇	÷	component_sizes ∇	÷
1	large1		20		[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	
2	large2		6		[1, 1, 20, 1, 1, 1]	
3	large3		11		[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1]	
4	medium1		6		[1, 1, 1, 1, 3, 3]	
5	medium2		9		[4, 1, 1, 1, 1, 1, 1, 1, 1]	
6	medium3		8		[2, 3, 1, 1, 1, 1, 1, 1]	
7	small1		6		[1, 1, 1, 1, 1, 1]	
8	small2		4		[3, 1, 1, 1]	
9	small3		7		[1, 1, 1, 1, 1, 1, 1]	
shortest_paths ∇		÷	longest_paths ∇		÷	
{ 'C0': 0, 'C1': 1, 'C2': 2, 'C3': 3, 'C4':...			{ 'C0': 0, 'C1': 1, 'C2': 2, 'C3': 3, 'C4':...			
{ 'C2': 1, 'C1': 0, 'C0': inf, 'C3': 6, 'C4':...			{ 'C2': 1, 'C1': 0, 'C0': -inf, 'C3': 6, 'C4':...			
{ 'C0': 0, 'C1': 2, 'C2': 4, 'C3': 6, 'C4':...			{ 'C0': 0, 'C1': 2, 'C2': 4, 'C3': 6, 'C4':...			
{ 'C4': 0, 'C0': inf, 'C1': inf, 'C2': inf, 'C3': inf, 'C5': 2}			{ 'C1': -inf, 'C2': -i...			
{ 'C0': 0, 'C6': 2, 'C7': 3, 'C1': 2, 'C2':...			{ 'C0': 0, 'C6': 2, 'C7': 3, 'C1': 2, 'C2':...			
{ 'C0': 0, 'C1': 2, 'C2': 3, 'C3': 5, 'C4':...			{ 'C0': 0, 'C1': 2, 'C2': 3, 'C3': 5, 'C4':...			
{ 'C0': 0, 'C2': 1.0, 'C1': 2.0, 'C3': 2.0,...			{ 'C0': 0, 'C2': 1.0, 'C1': 2.0, 'C3': 3.0,...			
{ 'C0': 0, 'C1': 2.0, 'C2': 3.0, 'C3': 4.0}			{ 'C0': 0, 'C1': 2.0, 'C2': 3.0, 'C3': 4.0}			
{ 'C0': 0, 'C4': 1, 'C3': 2, 'C1': 3, 'C5':...			{ 'C0': 0, 'C4': 1, 'C3': 2, 'C1': 3, 'C5':...			

Results (Per-Task Metrics)

Each dataset was processed by the Java program `Main.java`.
The algorithm automatically computed SCCs, condensation DAGs, topological order, shortest and longest paths, and recorded metrics in `results_summary.csv`.

Task	Metric	n (nodes)	Avg. Time (ns)	Description
SCC (Kosaraju)	DFS Visits	6-50	250,000–1,100,000	Each node visited twice (forward & reverse pass)
Topological Sort (Kahn)	Queue Operations	5-45	80,000–300,000	Push/pop per component in condensation DAG
Shortest Path (DAG-SP)	Edge Relaxations	7-70	120,000-400,000	Relaxation steps based on topological order
Longest Path (Critical Path)	Relazations(Max)	7-70	130,000-420,000	Similar to DAG-SP but maximization over edges
Total Runtime	-	6-50	0.5-2.2 ms	

Example (small2.json):

SCCs: 4 → [3,1,1,1]
Topological order: [C0,C1,C2,C3]
Shortest distances from C0: {C0=0.0, C1=2.0, C2=3.0, C3=4.0}
Longest (critical) path: {C0=0.0, C1=5.0, C2=7.0, C3=8.0}
Execution time: 0.7 ms

Observations:

- SCC step had the **highest number of operations**, especially in dense graphs with multiple cycles.

- Topological Sort scaled linearly and was the **fastest** step overall.
- DAG Shortest/Longest Path steps showed moderate time growth proportional to the number of edges.
- All datasets (even large ones with 50 nodes and 70+ edges) completed in **under 3 milliseconds**.

Execution time was recorded using `System.nanoTime()` to ensure accurate performance tracking.

Analysis

The implemented algorithms demonstrated strong and consistent performance across all dataset sizes.

1. SCC (Kosaraju)

- Efficiently found strongly connected components using two DFS passes ($O(V + E)$).
- Larger graphs had more SCCs but maintained linear scalability.
- Denser graphs slightly increased DFS recursion depth.

2. Topological Sort

- Used **Kahn's algorithm** on the condensation DAG.
- Performance was stable and almost instantaneous for all test cases.
- The resulting topological order correctly represented dependency hierarchies.

3. DAG Shortest and Longest Paths

- Based on dynamic programming over the topological order.
- Sparse DAGs processed quickly; dense ones required more relaxations.
- The **longest path** identified the system's **critical path**, useful for predicting total completion time.

Bottlenecks

- Performance bottlenecks appeared primarily in large graphs (20–50 nodes) due to increased DFS calls and relaxation operations.
- Nevertheless, runtime remained within milliseconds for all datasets.

Conclusions

The combined approach of **Kosaraju + Topological Sort + DAG Shortest/Longest Path** provided a reliable, efficient framework for analyzing and scheduling dependent tasks.

Key Takeaways:

- **Kosaraju's algorithm** is excellent for detecting and compressing cycles in directed graphs.
- **Topological Sort** ensures correct execution order in acyclic structures.
- **Shortest and Longest Path (DAG-SP)** algorithms help optimize and analyze scheduling efficiency.

Practical Recommendations:

1. **Use Kosaraju** to detect cycles and simplify graphs before scheduling.
2. **Apply Topological Sort** to determine execution order of independent tasks.
3. **Use DAG-SP** for optimization — shortest path for minimum time, longest path for critical path analysis.

This combination provides both **accuracy** and **practical utility** in Smart City / Smart Campus scheduling applications.