CS 721: Advanced Algorithms

HW: Performance, Graph Modeling, and NetworkX

Due: ____________________

Learning Goals

- Understand how **algorithm performance depends on memory access patterns**.
- Measure real execution time using Python.
- Practice converting problems into graph models.
- Use **NetworkX** to solve graph problems efficiently.

Part A — Matrix Multiplication Performance Experiment (40 pts)

You will explore how **loop ordering affects runtime performance** due to CPU cache locality.

Step 1: Implement Matrix Multiplication

Write a Python program that multiplies two matrices using **three nested loops**.

```
def multiply(A, B):
    n = len(A)
    C = [[0]*n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

Step 2: Generate Large Random Matrices

- Create two random matrices of size **10,000** × **10,000**.
- Use Python's `random` or `numpy` library.

Step 3: Measure Execution Time

Use Python's `time` library:

```
import time
start = time.time()
multiply(A, B)
end = time.time()
print(end - start)
```

Step 4: Loop Order Experiments

Interchange the three loops and measure execution time for:

- ijk
- ikj
- jik
- jki
- kij
- kji

Questions to Answer

1. Record the execution time for each loop ordering.
2. Identify which ordering is fastest.
3. Explain **why** it is fastest using concepts of:
   - CPU cache locality
   - Memory access patterns
   - Row-major storage

Deliverable:

- Table of timings
- Short explanation (1 page max)

Part B — LeetCode-to-Graph Using NetworkX (60 pts)

In this part, you will solve graph problems using the **NetworkX library**.

IMPORTANT REQUIREMENTS

- Your program must accept **LeetCode-style input**.
- Convert the input into a graph representation.
- Use **NetworkX functions** to solve the problem.
- Follow examples provided in `assignment1.py`.

Workflow You Must Follow

For each problem:

1. Parse input exactly as LeetCode format.
2. Convert input into a NetworkX graph.
3. Use NetworkX algorithms.
4. Output result in LeetCode format.

Example Workflow

```
import networkx as nx

edges = [[0,1],[1,2],[2,3]]
G = nx.Graph()
G.add_edges_from(edges)

print(nx.has_path(G, 0, 3))
```

A) DFS-centric

1. **200. Number of Islands** — Grid cells as nodes; 4-adjacency edges between land cells. Count DFS components.
2. **547. Number of Provinces** — Adjacency matrix → list; count connected components in an undirected graph.
3. **695. Max Area of Island** — Same grid→graph; use DFS to compute component sizes.
4. **1971. Find if Path Exists in Graph** — Build adjacency from edges; run DFS reachability.

Alternates: **417. Pacific Atlantic Water Flow** (reverse-edges DFS from oceans), **841. Keys and Rooms** (reachability).

B) Topological Sort

5. **207. Course Schedule** — Nodes are courses; edges `pre` → `course`. Topo order exists iff DAG.
6. **210. Course Schedule II** — As above; return a topological order (or empty on cycle).
7. **1203. Sort Items by Groups Respecting Dependencies** — Two-level DAG: groups and items. Topo both and compose.

Alternate: **269. Alien Dictionary** (premium) — Characters as nodes; edges from first differing pair in adjacent words.

C) Cycle detection / SCC

8. **684. Redundant Connection** — Undirected cycle detection; find extra edge that forms a cycle (can adapt DFS with parent tracking).
9. **685. Redundant Connection II** — Directed variant; handle node with two parents and detect cycle.
10. **2360. Longest Cycle in a Graph** — Directed graph with outdegree ≤ 1; detect cycles (via discovery times or SCCs) and return longest length.

Alternate: **802. Find Eventual Safe States** — Nodes not in cycles (identify via SCC condensation).

For Each Problem Submit

- Input parsing code
- Graph construction code
- NetworkX function used
- Final output

Submission

Two required steps:

- Upload assignment documents to the Blackboard assignment link.
- Show & explain your work: you will explain 3–4 problems (chosen by instructor/TA). The TA will mark it as shown; the final grade is calculated after this.

Documents deliverable:

- Python file for part A and part B
- PDF report
  - Report with timing results for Part A
  - Explanation of all the functions used in networkx
- Solutions for all 10 problems in Part B