Project Name: graph-coloring
Team Members: Anjolaoluwa Fajemisin (amf2320)

PROJECT OVERVIEW

For my project, I'll implement and parallelize a CSP solver for the graph coloring problem. The program will find valid colorings of graphs where no two adjacent vertices share the same color by using backtracking search. This project will include real-world instances such as map coloring, scheduling problems, and classic graph theory challenges.

ALGORITHM DESCRIPTION

- **The Graph Coloring Problem:**
  - Given an undirected graph G = (V, E) and k colors, assign a color to each vertex such that no two adjacent vertices have the same color.
- **Sequential Algorithm:**
  - I will use backtracking search with constraint propagation. Specifically, I will use:
    - The Most Constrained Variable (MCV) heuristic to select the next node
    - Forward checking to track remaining valid colors
    - Efficient adjacent list representation
- **Parallelization Strategy:**
  - Use Par monad to explore different color branches in parallel
  - Apply parMapM to test multiple colors simultaneously
  - Use depth threshold to help manage parallelization
- **Overall Key/Main Idea:**
  - When trying different colors for a node, test them in parallel instead of one-by-one. Only parallelize the first few levels to avoid creating too many small tasks that impede efficiency.

TEST CASES & INPUT DATA

- Instead of using random graphs, I wanted to use **real-world graph problems** to increase interaction.
  1. **US States Map (50 nodes, about 107 edges)** – Common 4-color theorem example
  2. **Course Scheduling (50-100 nodes)** – Minimize exam time slots
  3. **Petersen Graph (10 nodes)** – Known chromatic number: 3, for correctness testing
- **Input Format:**
  - Simple text files with adjacency lists, here's an example below:

- ```
  50 107        {--nodes, edges--}
  0 1           {--edge list--}
  0 4
  1 2
  ```
- **Data Sources:**
  - Manual creation for small graphs, random generation for benchmarks, public datasets for validation.

PERFORMANCE EVALUATION

- **Metrics:**
  - **Speedup:** Time (1 core) / Time (n cores) for $n \in \{2, 4, 8\}$
  - **Parallel Efficiency:** Speedup / n cores
  - I'll use +RTS -N8 -s -l for **ThreadScope analysis**

PROJECT SCOPE & TIMELINE

- Mainly, I'm focusing on the following **core deliverables**:
  - Sequential backtracking with MCV heuristic
  - Par monad parallelization with depth control
  - Comprehensive test suite (maps, scheduling, random graphs)
  - Performance benchmarks (1, 2, 4, 8 cores)
  - Documentation and reproducible build instructions
- If time permits, I'll work on the following **stretch goals**:
  - Graph difficulty analyzer (density, degree statistics)
  - Achievement system (speed milestones, optimal colorings)
  - Challenge selection menu
  - Strategies-based implementation for comparison
- **Out of scope**:
  - Real-time visualization
  - Interactive graph editor
  - Dynamic/weighted graph coloring
- **Rough Timeline:**
  - **Week 1:** Sequential implementation + basic tests
  - **Week 2:** Optimization (MCV, forward checking) + more test cases
  - **Week 3:** Par monad parallelization + benchmarking
  - **Week 4:** Final report, profiling, polish (+ stretch goals if ahead)
  ```