Project 2 Report: CSP Graph Coloring

Nijat Jafarov Fatulla Bashirov

1 Problem Summary

This assignment solves graph coloring as a Constraint Satisfaction Problem (CSP). Each vertex must get one color. Two connected vertices cannot share the same color.

The input file contains:

- comment lines that start with `#`,
- one line for the number of colors, for example `colors = 4`,
- undirected edges written as `u,v`.

2 Algorithm Choice

The implementation uses backtracking search with CSP heuristics and constraint propagation. The main components are:

- **MRV (Minimum Remaining Values):** choose the unassigned vertex with the smallest remaining domain.
- **Tie break with degree:** if MRV ties, choose the vertex with higher degree.
- **LCV (Least Constraining Value):** try colors that reduce neighbor options the least.
- **AC-3:** enforce arc consistency after each assignment.

3 How the Solver Works

The solver follows these steps:

1. Parse the input file and build the graph.
2. Create one domain per vertex: `{0,1,...,k-1}`.
3. Run AC-3 once to remove values that already violate constraints.
4. Start backtracking:
   - select a vertex with MRV and degree tie break,
   - order candidate colors with LCV,
   - assign one color and run AC-3 again,
   - recurse if domains stay non-empty,
   - backtrack on failure.
5. Return the full assignment if found; otherwise return `None`.

4 Why These Heuristics Help

MRV picks hard vertices early, so failures appear sooner. The degree tie break picks vertices that constrain many neighbors. LCV keeps more choices open for future steps. AC-3 removes impossible values before deep recursion. Together, these reduce unnecessary search compared with plain backtracking.

5 Implementation Notes

The code is in:

- `main.py`: solver implementation,
- `test.py`: unit tests for parser, graph build, AC-3, heuristics, and full solve cases.

The parser removes duplicate undirected edges by sorting edge endpoints. It also ignores self-loops.

6 How to Run

From the `Project2` directory:

Run solver

```
python main.py
```

Run tests

```
python -m unittest test.py
```

7 Limits and Scale

This is still an exact CSP search method, so worst-case runtime is exponential. For very large graphs, runtime and memory can become high. To scale further, practical next steps are:

- avoid full domain copies and use in-place updates with undo logs,
- use incremental AC-3 queues instead of full queue rebuilds,
- add forward checking before full propagation,
- use compact domain structures such as bitmasks.