Assignment 1: Crisis on Infinite Logics

SAT Encodings, Smart Constraints, and Optimization

Course: COL-750, Sem-II 2025-26, Prof. Priyanka Golia. **Total Marks:** 126

Due Date: 31st January 2026, 11:59 pm Indian Standard Time

- **Please read:** These problems are simple to solve with careful reading and algorithmic thinking. Boilerplate code here with function signatures is provided for all parts. your task is to translate the algorithmic logic into working Python implementations.
- **Academic Integrity:** All parts of assignment have to be done individually. All code submissions will be cross-referenced using MOSS (Measure of Software Similarity). Any similarity found with peer submissions, online repositories, or AI-generated code will trigger severe penalties (minimum penalty being zero in the assignment).

Submission Instructions

Directory Structure

Your submission must be a **single ZIP file** named `<RollNumber>_Assignment1.zip` containing exactly five subdirectories:

```
<RollNumber>_Assignment1/
 part1_joker/
    encoder.py
 part2_riddler_lattice/
    encoder.py
 part3_mirror_master/
    encoder.py
 part4_brainiac/
    solver.py
 part5_justice_league/
    optimizer.py
 reflections.pdf
```

Files to Submit

- **Code Files:** Only the Python files listed above. Do NOT include:
  - Test cases or input files
  - The provided `hero_db.json` (we will use our own)
  - Checker scripts or any other files
  - `__pycache__`, `.pyc`, or virtual environment folders
- **reflections.pdf:** A document (NOT generated by AI) containing:
  1. **Approach:** Brief description of your solution approach for each part (2-3 sentences per part)
  2. **Learnings:** What you learned from each part (1-2 key takeaways per part)
  3. **Challenges:** Any difficulties you faced and how you overcame them

Submission Portal

Submit your ZIP file via the course gradescope page before the deadline. Refer to course website for late submission policy.

Required Libraries and Resources

Python SAT Solver (PySAT):

- Installation: `pip install python-sat`
- Documentation: `https://pysathq.github.io/`
- Usage: `from pysat.solvers import Glucose3`

Z3 Solver:

- Installation: `pip install z3-solver`
- Documentation: `https://github.com/Z3Prover/z3`
- Python API: `https://ericpony.github.io/z3py-tutorial/guide-examples.htm`

DIMACS CNF Format:

- Official Specification: `https://www.satcompetition.org/2009/format-benchmarks2009.html`
- Tutorial: `https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html`
- Format: Each clause ends with 0, variables are integers, negation is represented by negative sign

Testing Your Code

Before submission, ensure:

- Your code runs with Python 3.8+ without errors
- Required libraries (`python-sat`, `z3-solver`) are installed and imported correctly
- Output format exactly matches the DIMACS CNF specifications
- File names and directory structure are correct

---

1 The Joker's Game

Problem Description

You are provided with a 5×5 grid that is partially populated with the characters **X** and **O**, and empty cells marked with a dot. Your objective is to determine if it is possible to populate the remaining empty cells with the character **O** such that the opponent, Player **X**, possesses no winning configuration. A winning configuration for Player **X** is defined as four consecutive **X**s appearing in any row, column, or diagonal. The program must generate a boolean formula that is satisfiable if and only if such a defensive configuration exists. Refer to the `part1_joker` directory in the boilerplate for the starter script.

Additionally you should try exploring the set of clauses needed if only a single **O** would be allowed to populate (**ungraded**)

Input and Output Specifications

Input: The program reads from standard input. The input consists of exactly 5 lines where each line contains a string of 5 characters representing the board rows. Allowed characters are 'X', 'O', and '.'.
Output: The program must print a valid DIMACS CNF formula to standard output. The SAT solver used by the autograder must return SAT if a valid placement of 'O's exists that prevents 'X' from winning, and UNSAT otherwise.

2 The Riddler's Lattice

Problem Description

You are given a prime number P. You must construct a grid of size $N \times N$ where $N = P - 1$. The cells of the grid must be filled with integers from 1 to N subject to two strict constraints. First, the grid must satisfy the Latin Square property, meaning every row and every column must contain a unique permutation of the integers 1 to N. Second, the grid must satisfy the Generator Constraint. This constraint dictates that every row in the grid must correspond to a cyclic sequence generated by a Primitive Root modulo P. Formally, for every row index r, there must exist a primitive root g of P and a shift parameter s such that the value at column c is congruent to $g^{(s+c)} \pmod{P}$.

Additionally, you may be provided with **hints** - pre-assigned values for specific cells. These hints act as fixed constraints that your solution must satisfy. A hint (r, c, v) means cell at row r and column c must contain value v. Your CNF encoding must ensure these cells are assigned their specified values while still satisfying both the Latin Square and Generator constraints. If the hints create a conflict (e.g., two different values for the same cell, or a hint that violates the primitive root structure), your CNF should be unsatisfiable.

Refer to the `part2_riddler_lattice` directory in the boilerplate code.

Input and Output Specifications

Input: The program reads from standard input. The first line contains the prime P. The second line contains an integer K representing the number of hints. The subsequent K lines contain three integers r, c, and v specifying that the cell at row r and column c must contain the value v.
Output: The program must print a valid DIMACS CNF formula to standard output. The formula must be satisfiable if and only if a grid satisfying both the Latin Square and Generator constraints exists given the hints.
Variable Encoding Requirement: Your CNF encoding must use the following standardized variable mapping to enable solution verification. For each possible cell-value assignment, define a Boolean variable with the following numbering scheme:

$$\text{var}(r, c, v) = r \times N \times N + c \times N + v$$

where $N = P - 1$. The function returns a positive integer that serves as a unique identifier (variable index) for the corresponding Boolean variable in DIMACS CNF format. When the SAT solver assigns this variable TRUE in a satisfying model, it means cell (r, c) holds value v.
Indexing conventions:

- Rows r and columns c are 0-indexed: $r, c \in \{0, 1, \ldots, N-1\}$
- Values v are 1-indexed: $v \in \{1, 2, \ldots, N\}$ (representing actual grid values)
- The $(v-1)$ term converts value indexing to 0-based for the formula

Example

Sample Input (No Hints):

```
5
0
```

Sample Output (first few lines):

```
p cnf <num_vars> <num_clauses>
<clause_1>
<clause_2>
...
```

Explanation: For prime $P = 5$, we need a 4×4 Latin Square where each row is generated by a primitive root modulo 5. The primitive roots of 5 are $\{2, 3\}$. With no hints ($K = 0$), all cells are free variables, and the encoder must generate clauses for:

- Latin Square constraints (each row/column contains 1,2,3,4 exactly once)
- Generator constraints (each row follows pattern $g^s, g^{s+1}, g^{s+2}, g^{s+3} \pmod 5$ for some primitive root g and shift s)

The formula is satisfiable.
Sample Input (With Hints):

```
5
2
0 0 2
1 1 3
```

Explanation: Same as above, but now cell $(0,0)$ must be 2 and cell $(1,1)$ must be 3. Your encoder must add unit clauses to fix these values. The SAT solver will find a solution respecting these constraints, or determine it's impossible.

3 Mirror Master's Maze

Problem Description

You are provided with a grid representing a room layout containing empty spaces, opaque walls, and mirrors. Mirrors are oriented at either 45 degrees or 135 degrees. You are also given an integer K representing the number of available light emitters. Light travels in straight lines along the cardinal directions and reflects 90 degrees upon hitting a mirror according to standard optical laws. Light is absorbed by walls and passes through empty space. Your task is to select exactly K coordinates in the empty space to place emitters such that every empty cell in the grid is illuminated by at least one light beam. Refer to the `part3_mirror_master` directory in the boilerplate code.

Input and Output Specifications

Input: The first line contains three integers H, W, and K representing Height, Width, and Emitters respectively. The following H lines contain strings of length W representing the grid layout using the characters '.', '#', '/', and '\'.
Output: The program must print a valid DIMACS CNF formula to standard output. The formula represents the Set Cover problem instance derived from the geometric visibility constraints.

Encoding Requirements:

- Your CNF must encode two constraints: (1) every empty cell is illuminated by at least one light beam, and (2) exactly K emitters are placed in empty cells.
- Light can illuminate both empty cells (.) and mirror cells (/ or \). Only empty cells can contain emitters.
- Each emitter emits light in all four cardinal directions (North, East, South, West) simultaneously.

Example

Sample Input:

```
5 5 2
.....
.#.#.
.....
.#.#.
.....
```

Sample Output (first few lines):

```
p cnf 105 126
8 12 16 20 21 33 53 65 0
2 12 16 20 0
2 6 16 20 25 41 57 73 0
...
```

4 Brainiac's Firewall

Problem Description

The Justice League Watchtower has been infected by Brainiac's logic virus. The alien code creates a firewall that rejects access unless a specific boolean formula is satisfied. However, Brainiac has neutralized all standard terrestrial solving algorithms; the system detects and blocks binaries like MiniSAT or Z3. To regain control of the Watchtower, you must manually implement a clean-room logic solver from scratch that can traverse Brainiac's decision tree and find the breaking key.

You are required to implement a complete SAT solver based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. The solver must be implemented in Python without the use of external SAT libraries. To meet the grading performance benchmarks, your implementation must include efficient Unit Propagation and a deterministic branching heuristic such as MOMs or DLIS. Refer to the `part4_brainiac` directory in the boilerplate code.

Input and Output Specifications

Input: The program reads a standard DIMACS CNF formatted string from standard input.
Output: The program must print the satisfiability verdict to standard output. If satisfiable, print `s SATISFIABLE` followed by a line starting with `v` listing the satisfying assignment. If unsatisfiable, print `s UNSATISFIABLE`. The solver must complete execution within 30 seconds for standard benchmarks.

5 The Justice League Protocol

Problem Description

You are provided with a JSON database of candidate heroes where each candidate has an associated recruitment cost, a power level, and a list of attribute tags. You must implement an optimizer using Z3 to select exactly six candidates to form a team. The selection is constrained by two factors. First, the sum of the recruitment costs must not exceed a specified budget. Second, the team must form a compatibility clique which is defined as a group where every member shares at least one attribute tag with every other member. Refer to the `part5_justice_league` directory in the boilerplate code.

Input and Output Specifications

Input: The program accepts two command-line arguments. The first is the path to the JSON database file and the second is an integer representing the Budget.
Output: The program must print a single line containing the space-separated indices of the selected heroes. If no valid team exists, print `None`.

Grading Scheme

Parts 1-4: Test Case Based Grading (70 marks total)

Your grade for each part is determined by the **percentage of hidden test cases passed**:

$$\text{Marks Earned} = \text{Total Marks for Part} \times \frac{\text{Test Cases Passed}}{\text{Total Test Cases}}$$

Example: If Part 2 (23 marks) has 20 hidden test cases and you pass 15 of them:

$$\text{Marks} = 23 \times \frac{15}{20} = 17.25 \text{ marks}$$

Part	Total Marks	Grading Method
Part 1: Joker's Game	7	% of test cases passed
Part 2: Riddler's Lattice	23	% of test cases passed
Part 3: Mirror Master's Maze	16	% of test cases passed
Part 4: Brainiac's Firewall	24	% of test cases passed

Test Case Requirements:

- **Parts 1-3:** Your CNF formula must be correct and efficient(SAT/UNSAT verdict matches expected). Focus deeply on the number of clauses you generate in these parts as naive implementation might time-out for larger testcases
- **Part 4:** Your DPLL solver must output correct verdict and valid model (if SAT)
- **Time Limit:** There will be a very tight time limit for part 4 that bounds it to mimic or improve the DPLL algorithm. In any case, it will be manually screened.
- **Hidden Tests:** Test cases will not be released; ensure your code handles edge cases.

Part 5: Competitive Grading (30 marks)

Part 5 uses a hybrid grading scheme:

- **10 marks (Baseline):** Awarded if your solution produces a **valid team** that satisfies all constraints:
  - Team size = exactly 6 heroes
  - Total cost $\leq$ budget
  - All pairs share at least one tag (compatibility clique)
- **20 marks (Competitive):** To be eligible for this, you must produce a valid team. You will be scored based on your solution's power score relative to the class as a decaying function of rank determined by valid max power, and ties broken by time taken.

Reflections Document (26 marks)

You must submit a professionally formatted `reflections.pdf` document containing your insights and learnings from the assignment. This document will be graded on completeness, attention to detail and novelty.

Enfin

- Start early because debugging CNF encodings and DPLL implementations takes time.
- Ask questions to the TA (cs5221643 - Raj) on MS teams if you're stuck on concepts.
- Ensure you follow the Input Output format strictly, as most of the assignment will be auto-graded.

Good luck!