Problem set 1: SAT Solving

Exercises are meant to be solved collaboratively during the tutorial. Projects have to be handed in.

Exercise 1 :

Setting Up and Running a SAT Solver:

(a) **Step 1**: Install MiniSat using your package manager (e.g., apt on Debian/Ubuntu, brew on macOS, or download and compile from the official MiniSat website http://minisat.se/).

(b) **Step 2**: Create a text file (e.g., example.cnf) with a SAT problem in DIMACS CNF format. DIMACS CNF format specifies the number of variables, number of clauses, and the clauses themselves. Here's an example problem:

```
c Sample SAT Problem
p cnf 3 3
1 -2 3 0
-1 2 0
-1 -2 -3 0
```

(c) **Step 3**: Run MiniSat on the problem with the following command:

```
minisat example.cnf output.txt
```

(d) **Step 4**: Interpret the Results

1. If the problem is satisfiable, MiniSat will create an output.txt file with a line like:

   ```
   s SATISFIABLE
   ```

2. If the problem is unsatisfiable, MiniSat will output:

   ```
   s UNSATISFIABLE
   ```

3. If satisfiable, you can find the assignment for the variables in the solution. They will be listed under the v line in the *output.txt* file, like:

   ```
   v 1 -2 -3
   ```

(e) **Step 5**: Run MiniSat on different problems to get a better understanding of its usage.

(f) **Step 6 (Optional)**: You can repeat the above steps using CaDiCaL instead of MiniSat by installing CaDiCaL and running it in a similar manner.

This exercise should help you get started with setting up and running a SAT solver like MiniSat or CaDiCaL. Experiment with different SAT problems to gain more experience with these solvers.

Exercise 2 :

Encoding the N-Queens Puzzle into SAT Formulas. In this exercise, you will learn how to encode the N-Queens puzzle into SAT formulas. The N-Queens problem involves placing N chess queens on an NxN chessboard, so that no two queens threaten each other. This exercise will focus on encoding the 8-Queens problem into SAT formulas. In chess, queens can move horizontally, vertically, and diagonally.

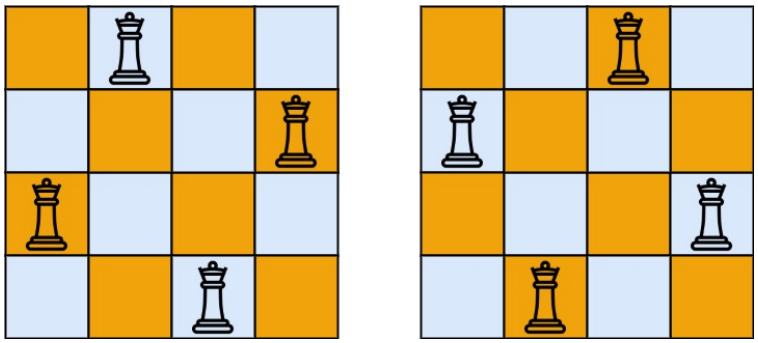

Figure 1: 4-queens

(a) Define a set of Boolean variables to represent the placement of queens on the chessboard. For an 8x8 board, you'll need 64 variables. You can name them, for example, Qxy, where x and y represent the row and column, respectively.

(b) Formulate constraints to ensure that no two queens can threaten each other. These constraints can include:

1. One queen in each row: For each row, there must be exactly one queen placed.
2. One queen in each column: For each column, there must be exactly one queen placed.
3. Diagonal constraints: No two queens should threaten each other diagonally.

(c) Encode Constraints into SAT Formulas

1. For each constraint, create SAT formulas that represent it using the Boolean variables defined before. For example, the constraint *One queen in each row* can be encoded as a formula:
   - For each row i, you should have a clause like $(Qi1 \vee Qi2 \vee ... \vee Qi8)$, which ensures at least one queen is in that row.
   - You should also have clauses ensuring that no two queens are in the same row, like $(\neg Qi1 \vee \neg Qi2) \wedge (\neg Qi1 \vee \neg Qi3)$, and so on.
2. Similarly, create formulas for "One queen in each column" and "Diagonal constraints." Diagonal constraints are a bit more complex and might require extra variables and clauses.

(d) Install a SAT solver like MiniSat or CaDiCaL (as in the previous exercise). Then, Create a DIMACS CNF file containing the SAT formulas generated before.

(e) Run the SAT solver on the DIMACS CNF file. If the solver returns *SATISFIABLE*, it means a valid solution has been found. You can extract the queen placements from the solver's output.

(f) You can experiment with different board sizes (e.g., 4-Queens, 5-Queens) by modifying the number of variables and constraints in your SAT encoding.

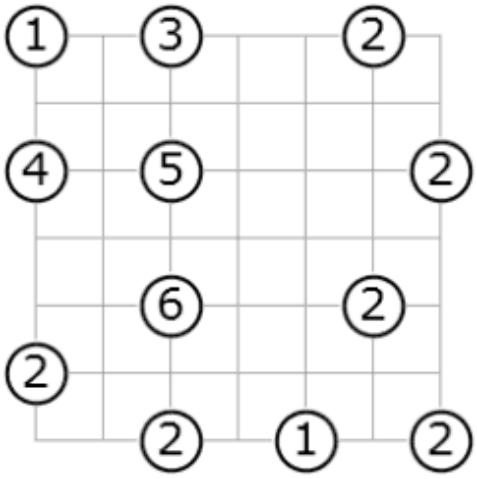

Figure 2: Puzzle 1

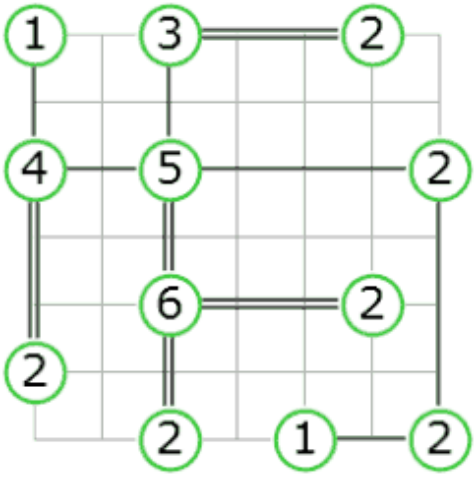

Figure 3: Puzzle 1 solved

Project 1: The goal of this project is to encode the Bridges puzzle into SAT formulas. Each puzzle is based on a rectangular arrangement of islands where the number in each island tells how many bridges are connected to it. The object is to connect all islands according to the number of bridges so:

1. There are no more than two bridges in the same direction.
2. Bridges can only be vertical or horizontal and are not allowed to cross islands or other bridges.
3. When completed, all bridges are interconnected enabling passage from any island to another.

You can find more information and play Bridges here: Bridges Puzzle.

We will use the following format to represent puzzles as inputs to the program: The first line contains n, the number of rows and columns of the puzzle, respectively. The following grid represents an example puzzle input. A dot . indicates an empty cell, while a number marks an island with the number of bridges that can leave the island cell. The puzzle in figures 2 and 3 correspond to the following input:

```
7
1.3..2.
.......
4.5...2
.......
..6..2.
2......
..2.1.2
```

This project consists of four parts. The first part is necessary to pass, while the other parts will get you a better grade. Keep in mind that quality is more important than quantity, though.

(a) Compulsory part: Implement a program that reads puzzles in the above format, converts them into a SAT instance, runs a SAT solver to find a solution, and displays the result graphically. A set of sample inputs are attached to the problem set, more can be generated from the template puzzles on the website linked above. We will verify your program against different inputs as well and the provided ones are only meant to help you test your program.

(b) Bonus part: How well does you encoding scale with increasing grid sizes? Measure its performance and devise a more efficient encoding.

(c) Bonus part: Add a feature to enter new puzzles manually via the GUI.

(d) Bonus part: Add a feature to generate new puzzles automatically and store them in the above format. Ensure that the puzzles generated have a unique solution.

Solutions can be submitted via Moodle until Thursday, 23. November 2023, 14:00 in groups of 1-3 students. The groups can be specified on Moodle. On the courses website, under section "Groups" you can find a link leading to the group selection. There you can form groups of 1-3 students.
The programming languages C/C++, Rust, Java, Python and Haskell may be used. Other languages might be permitted as well, just ask us beforehand. We recommend using C++, because particularly for the later projects, a competitive performance is difficult to attain in other languages.
Projects will be stored in GitLab. Make sure that you have access with your cip account and if not follow the instructions listed here to gain access. In a new GitLab repository, named after your group number, upload your project files and also add a short README file, explaining how to build/run your program. You are also instructed to upload your generated DIMACS files for the solved instances of the puzzle to the repository. When you are ready to submit your project, please fork your repository and share the forked version with us.