Programming Assignment 2
Mystic Sudoku

COP3503 Computer Science 2 Spring 2026

Due Date

The assignment is due on **Sunday, February 15 at 11:59 PM EST** via Webcourses. **Do not email the professor or TAs. Submissions by email will not be accepted.**

Assignments may be submitted up to 24 hours late with a penalty. Please refer to the syllabus for details. To receive full credit, submit on time and allow extra time for uploading. Last minute submissions often fail due to slow Internet, resulting in late penalties.

The **Webcourses timestamp** determines submission time. If your submission is even one second past the deadline, it will be marked late. Internet issues are not an excuse. Please plan accordingly.

Assignment Description

The purpose of this assignment is to help you understand how a real-world problem can be solve using backtracking algorithm that we have learned in class as part of our CS2 course. The solution to this assignment has to be written in Java and tested on the Eustis server before submission.

Problem Description

Knightro, the UCF mascot, has discovered a magical Sudoku puzzle in the Pegasus Archives. Each puzzle is a 9x9 grid partially filled with digits from 1 to 9. Additionally, the puzzle board is divided into nine fixed 3×3 subgrids: rows (0–2), (3–5), (6–8) × columns (0–2), (3–5), (6–8). Your task is to help Knightro solve these puzzles using backtracking, under both the classic Sudoku rules and the additional magical rules described below.

Rules

- Each row must contain the digits 1 through 9 exactly once.
- Each column must contain the digits 1 through 9 exactly once.
- Each 3x3 subgrid must contain the digits 1 through 9 exactly once. Subgrids do not wrap around the edges.

- No two identical digits may be placed in cells that are a chess knight's move apart (i.e., offsets $(\pm 2, \pm 1)$ and $(\pm 1, \pm 2)$).
- The board wraps at the edges. When checking neighbors (including knight moves and orthogonal adjacency), row/column indices must be wrapped around the edges. For example, from (0,0), a knight move of (-2,-1) refers to cell (7,8). Wrap-around applies only to neighborhood checks (knight moves and orthogonal adjacency). Row, column, and 3×3 subgrid checks do not wrap.
- After each 9×9 grid, K pairs of digits (unordered) will be given. For any pair (a,b), no cell containing a can be orthogonally adjacent (up, down, left, right) to a cell containing b, and vice versa. Orthogonal adjacency checks also wrap around the edges. Treat each forbidden pair as unordered: if (a,b) is listed, then a cannot be orthogonally adjacent to b, and b cannot be orthogonally adjacent to a. **Note:** Orthogonal adjacency refers only to the four neighbors: up, down, left, and right (with wrap-around). Diagonal neighbors do not count.

Sample Sudoku Puzzle with rules and restrictions

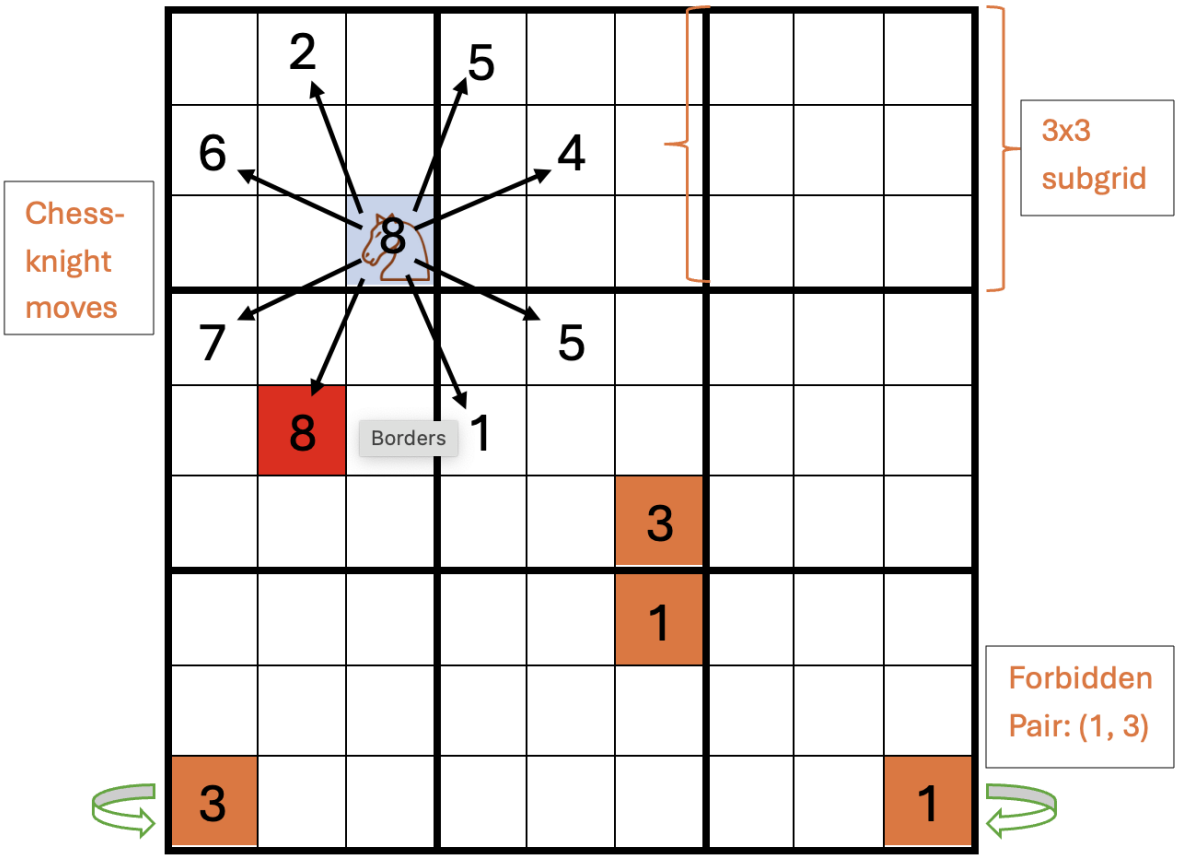

Figure 1: Sudoku board with restrictions

The Provided Driver File

You have been provided with a driver file called `SudokuSolverDriver.java`. This driver file is responsible for running a set of predefined test cases for the methods you will implement in `SudokuSolver.java`. A driver file is a simple program, usually a class with only a main method, whose job is to run, test, or demonstrate other classes in separate files (the ones you will create). If you open the driver file, you will notice a main method containing some code that will open the case specific input files and produce the Sudoku board with the data provided in those input files. Each case represents a test scenario to execute.

Please do not modify the provided driver file in any way. This includes adding or removing any content. Changes to the driver file can prevent your code from being evaluated correctly, which may result in lost points that cannot be recovered. In some cases, modifications may cause your code to fail compilation, which will have a serious impact on your grade and will not be reassessed. Graders will use a driver file with modified test cases but will rely on the same setup provided in this assignment.

Inputs

The input files provided with this assignment contain the inputs. Each input file contains the following information:

- The first line contains a positive integer n, the number of puzzles.
- Each puzzle consists of:
  - 9 lines, each with 9 space-separated integers in [0-9], **0 represents an empty cell**.
  - 1 line describing forbidden pairs: the first integer is K ($0 \leq K \leq 20$), followed by $2 \cdot K$ integers representing the unordered pairs $(a_1, b_1), \ldots, (a_K, b_K)$, where each digit is in [1–9].

The input from the files will be handled by the provided driver file, and only a two-dimensional integer array `board[][]` of size 9×9, representing the Sudoku puzzle, and a two-dimensional integer array `forbiddenPairs[][]` of size $K \times 2$, will be provided to your methods to solve.

Output

For this assignment, the Driver file will handle the output process. Your task is to return the number of solution exists through the “solve” function. For each puzzle, “Puzzle k:” (without the quotes) will be printed where k is the puzzle number. Then,

- If a **UNIQUE** solution is found, the completed grid in 9 lines will be printed. Each line will contain space-separated 9 numbers.
- If more than one solution is found, the number of solutions found will be printed.
- If no solution exists, “No solution possible.” (without the quotes) will be printed.

Sample Input 1

```
1
9 8 0 2 0 3 7 0 0
5 0 0 0 0 9 8 0 0
0 0 0 0 0 8 0 0 4
0 0 0 0 0 0 5 0 0
0 5 0 0 9 0 3 0 0
0 0 0 0 5 0 0 7 0
0 1 9 8 3 0 2 0 0
0 0 0 0 4 0 0 3 8
0 0 0 1 2 7 6 5 9
2 2 6 2 3
```

Sample Output 1

```
Puzzle 1:
9 8 4 2 1 3 7 6 5
5 7 2 4 6 9 8 1 3
3 6 1 5 7 8 9 2 4
4 3 6 7 8 1 5 9 2
2 5 7 6 9 4 3 8 1
1 9 8 3 5 2 4 7 6
6 1 9 8 3 5 2 4 7
7 2 5 9 4 6 1 3 8
8 4 3 1 2 7 6 5 9
```

Sample Input 2

```
1
0 0 8 0 4 0 6 0 0
0 0 0 0 0 8 0 0 0
0 0 0 0 6 0 1 0 8
8 9 0 0 0 0 2 0 0
0 0 0 0 1 0 0 0 0
4 0 0 9 0 3 0 0 0
7 0 0 0 0 0 0 2 0
6 0 0 7 0 1 4 0 0
0 4 0 2 0 0 0 0 0
2 5 7 9 3
```

Sample Output 2

```
Puzzle 1:
2 solutions found.
```

Sample Input 3

```
1
3 0 0 0 2 0 0 1 0
0 0 0 0 0 0 7 0 2
0 0 0 5 0 1 3 0 0
9 0 0 0 0 8 0 0 0
0 0 0 1 0 6 0 2 0
0 0 1 7 0 0 6 3 5
0 0 0 0 0 9 4 0 1
0 0 4 2 0 0 0 0 6
6 7 9 0 0 0 0 0 0
1 2 7
```

Sample Output 3

```
Puzzle 1:
No solution possible.
```

Implementation Directions

You are provided with a driver file `SudokuSolverDriver.java`. You must implement the class `SudokuSolver` in `SudokuSolver.java` with the following solver method:

```
public int solve(int[][] board, int[][] forbiddenPairs);
```

The solve method must:

- Modify the board in-place.
- Return an integer that represents `the number of solutions found` with the input puzzle.

Important: Your solution must enforce all constraints: classic row/column/subgrid rules, knight's move with wrap-around, and forbidden pairs on orthogonally adjacent cells with wrap-around. You must use Backtracking to solve the problem. Do not write a main method. Your code will be executed using the provided driver script.

Note: You are welcome to add any other methods as long as you do not modify the driver script.

Important Guidelines:

- You **must use Backtracking with recursion** to solve this problem.
- **Do not write a** `main` **function.** Your code will be executed using the provided driver script.
- Carefully review how the driver script calls your class and ensure your implementation works as expected.
- Test your code with additional test cases that you create.
- The grader will use different test cases, but they will follow the same format as the provided examples.
- Run and verify your code on **Eustis** before submitting to ensure compatibility with the grading environment.
- Follow the provided coding style guide to receive full credit.
- Add a header to your program with your name, course name, and assignment name.

Driver Contract The driver will:

- Read n.
- For each puzzle, read 9 lines of the grid.
- Read 1 line containing K followed by $2 \cdot K$ digits for forbidden pairs.
- Construct the solver as new SudokuSolver() and then call solve(board, forbiddenPairs) function.
- The driver will print `Puzzle k:` followed by the solved 9×9 grid (9 lines) if there is a unique solution. `Puzzle k:` followed by "s solutions found." will be printed if there exist s $(s > 1)$ solutions. If no solution exist for the input puzzle, then `Puzzle k:` followed by "No solution possible." will be printed.

Note: The driver does not alter your board or pairs; your $solve$ implementation must enforce all constraints.

Code Style

It is expected that students follow the code style guide requirements provided for this course. A style guide has been made available in the assignment page and students are expected to adhere to it starting with this programming assignment.

Testing my Code in Eustis

Make sure to test your code on `Eustis` before submitting your assignment, as this is the environment the TAs will use to compile and run your code. Failure to successfully compile or run on Eustis will result in a score of **zero**, even if the code runs correctly on your local machine.

Manual Run through `javac` and `java`

After moving your files into the same directory on Eustis, make sure they are compiled before testing your assignment. To compile your code, use the following command:

```
javac SudokuSolverDriver.java
```

If no messages are displayed, your code has compiled successfully with no errors or warnings. After compiling, you can execute the program by selecting the test case you wish to run. Use the following command format:

```
java SudokuSolverDriver <test_case_number>
```

For example, to run test case 2, use:

```
java SudokuSolverDriver 2
```

How graders will test your code through `python3`

Once you have completed manual testing of all test cases individually and verified that your output is correctly formatted, you may perform a final test using the Python script provided for grading purposes. This script will execute all test cases and generate a report indicating whether all tests passed or if any cases are still incorrect.

The command to run the Python script is:

```
python3 SudokuSolverTester.py
```

For more details on how this script works, please refer to the section titled *How Graders Will Test Your Code Using Python3* from the Eustis Setup assignment.

Warning About Test Case 5

This test case takes a fair amount of time to run. Please plan accordingly when testing on Eustis, and don't wait until the last minute before the deadline to make sure everything works properly.

Comment Header

Please make sure to provide the appropriate comment header as the very first content at the top of your file (including before any import statements). If your comment header is incorrect or not placed literally at the top of the file, points will be deducted according to the rubric.

The header must be formatted exactly as shown below and placed on the first line of the file with no whitespace above it. For example, if your name is Steve Harrington, the comment header should look like:

```
1 /*  Steve Harrington
2     Mystic Sudoku
3     COP3503 Computer Science 2
4     SudokuSolver.java
5 */
```

The format is as follows:

- First line: Your first and last name.
- Second line: The assignment name.
- Third line: The course code and course name.
- Fourth line: The name of the Java file.

This format must be followed exactly for the Python script to properly capture your name. Failure to follow this format will result in point deductions. The main reason for this requirement is that graders will use a special Python script to record your name during test runs.

Grading Rubric

Please refer to Webcourses for the official grading rubric. It is located on the assignment page.

A Very Important Note for Running Eustis and Packages! Read Carefully!

Many of you are probably using IDEs like Netbeans and Eclipse to build your Java Solutions. Please note that some of these IDEs will automatically place your Java file in some sort of package. Please make sure your Java file is not defined in some package as this can result package private errors. Any such error that occurs during the grading will not be fixed and points will be deducted as such in accordance with the respective categories in the rubric. Also, DO NOT create a main method in your solution file!! This will result in your code not running properly with the runner file which will result in points being deducted from the respective categories.

What to submit to Webcourse?

For this assignment, please submit your `SudokuSolver.java` file only.

Do not submit the driver file. The graders will use their own driver file with slightly modified test cases to evaluate your submission.

What happens if my submitted code doesn't compile or is stuck in an infinite loop? How will my code be graded?

If your code does not compile at all, the graders will not award any credit (a score of 0) for categories related to compilation or any functionality that depends on successful compilation.

If your code compiles but enters an infinite loop or throws an exception, points will be deducted in the categories directly affected by these issues.

AI Policy

Use of Generative Artificial Intelligence (GenAI) is prohibited. Use of GenAI tools via website, app, or any other access, is not permitted in this class. All components of assignments in this course must be independently and originally completed by the student. Representing work created by GenAI as your own will be treated as plagiarism.