COMS 4701 Artificial Intelligence

Homework 2: Coding - CSPs

Due Date: October 23, 2024

Please read all the sections carefully:

I. Introduction
II. Executing your program
III. Backtracking Algorithm
IV. Important Information
V. What You Need To Submit
VI. Before You Submit

I. Introduction

Futoshiki is a CSP puzzle which is similar to Sudoku. The objective of Futoshiki is to fill an nxn grid with the numbers 1 to n so that each column and row contains one of each digit. In addition, each board may or may not have inequality signs between two boxes that are adjacent horizontally or vertically (no diagonal inequalities are allowed). You may try out the game here: **futoshiki.com**. Futoshiki has n^2 **variables**, i.e. n^2 tiles. The variables are named by **row** and **column**, and are **valued** from 1 to n subject to the constraints that no two cells in the same row or column may be the same. Additionally, the constraints described by the inequalities must also be satisfied. Note that all inequalities present will be strict, since two adjacent boxes may not have the same value (this would break the row or column constraint).

For this homework we only expect your code to function for $1 \leq n \leq 9$. In our testing, it seems unreasonable to expect to run boards with $n > 7$ due to time constraints. So, our test cases will be limited to $1 \leq n \leq 7$. Please make sure that boards of size $n \leq 5$ run in under a second, boards with $n = 6$ run in at most 5 seconds and boards with $n = 7$ run in at most 5 minutes. It may be a fun challenge for you to implement an algorithm that runs boards with $n = 8$ or even $n = 9$ in a reasonable amount of time, however these cases will not be tested.

Frame your problem in terms of **variables**, **domains**, and **constraints**. We suggest representing a Futoshiki board with a Python dictionary, where each key is a variable name based on location, and value of the tile placed there. Using variable names **Al**... **A9**... **I1**... **I9**, the board above has:

- my_board["B1"] = **2**, and
- my_board["E2"] = **9**.

We give value **zero** to a tile that has not yet been filled.

Additionally, we suggest using a '*' in the key of the dictionary to represent possible inequalities between boards. An '*' after a letter indicates the inequality between the row represented by the letter and the next row.

e.g. my_board["A*1"] = '<' means the value at A1 must be less than the value at B1.

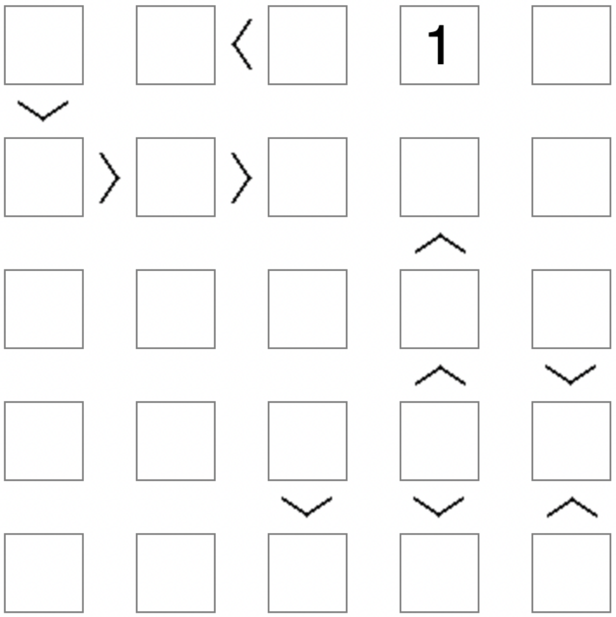
Similarly, an "*" after the number indicates the inequality between the column represented by the number and the next column.

e.g. my_board["A1*"] = '>' means the value at A1 is greater than the value at A2.

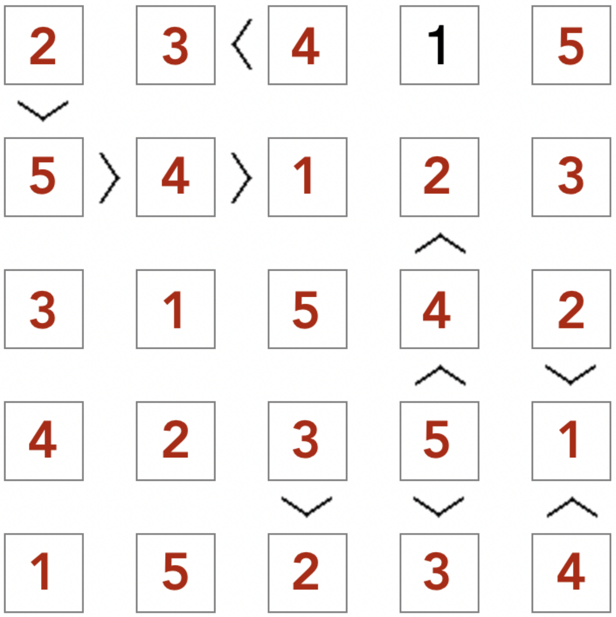
If there are no inequalities between adjacent tiles, the corresponding entry in the dictionary is '-' (You do not have to replace these with inequality signs. The only variables in this problem are the number tiles). The structure of a futoshiki board has been encoded for you in the skeleton code.

The following is an example of an empty futoshiki board followed by a filled futoshiki board:

Unsolved Board:



Solved Board



II. Executing your program

Your program will be executed in one of two ways:

```
$ python3 futoshiki.py <input_string>
```

```
$ python3 futoshiki.py
```

The first method will solve the Futoshiki board represented as a string, printing the solution to standard output and saving the string to a file called `output.txt`. The second method will look for and run a file called `futoshiki_start.txt` in the same directory as `futoshiki.py`. It will run each line in the file as a seperate board, print all the solutions to stdout and generate a `output.txt` file with all the solutions to the futoshiki. The skeleton code implements all of this functionality. You may refer to it for more details.

In the starter zip, `futoshiki_start.txt`, contains 10 sample unsolved Futoshiki boards, and `futoshiki_finish.txt` the corresponding solutions. Each board is represented as a single line of text, starting from the top-left corner of the board, and listed left-to-right, top-to-bottom. Horizontal inequalities are placed between the numbers and vertical inequalities are placed between each row.

The first board in `sudokus_start.txt` is represented as the string:

```
0-0<0---0<2-0<--0-0-0
```

Which is equivalent to:

```
0     0  <  0

0  <  2     0
∧
0     0     0
```

Your program will generate `output.txt`, containing a single line of text representing the finished Futoshiki board. E.g.:

```
3-1<2---1<2-3<--2-3-1
```

Test your program using `futoshiki_finish.txt`, which contains the solved versions of all of the same puzzles. All puzzles we provided contain exactly one unique solution each. All of our test cases will also only include examples that have exactly one solution per futoshiki puzzle.

III. Backtracking Algorithm

Implement **backtracking** search using the minimum remaining value heuristic. Pick your own order of values to try for each variable, and apply forward checking to reduce variables domains.

- Test your program on `futoshiki_start.txt`.
- Report the number of puzzles you can solve and the mean, standard deviation, min, and max of the runtime over all puzzles in `futoshiki_start.txt`.

IV. Important Information

1. Test-Run Your Code

Test, test, test. Make sure you produce an output file with the **exact format** of the example given.

2. Grading Submissions

We test your final program on **~ 200 boards**. Each board is worth **0.5 points** if solved, and zero otherwise. These boards are similar to those in your starter zip.

3. Time Limit

No brute-force! Make sure that boards of size $n \leq 5$ run in under a second, boards with $n = 6$ run in at most 5 seconds and boards with $n = 7$ run in at most 5 minutes (some boards with $n = 7$ should run in a few seconds, but there are a few that take much longer).

V. What You Need To Submit

You must submit a `futoshiki.py` file. Ensure that this is the exact name of the file and that it functions exactly to the specifications mentioned in this document. While not required, we highly recommend using the provided skeleton file since you will find that a lot of the non-backtracking related functionality has already been implemented for you. Note that if you choose not to use the template, we may not regrade your submission in the event of an autograder error.

VI. Before You Submit

- Ensure that your file is named `futoshiki.py`
- Ensure that your file compiles and runs.
- The Gradescope submission box will be configured with an autograder that will tell you whether your file compiles and runs on our system. After submitting your assignment on gradescope, wait until you see this message before considering yourself done!