Assignment 1 - Specification

2. **`Forced min and max values`**

   Cells that are less than 2 must implicitly have the value 1 as it is the only admissible value on the board which respects that condition. Similarly, cells that are greater than the board size minus 1 must be equal to the board size. In the example below, the only possible value for the green cell (less than 2) is 1.

3. **`Exclusion of min and max values`**

   Cells that are greater than other cells cannot be 1, the lowest value allowed on the board, as there is no value smaller than 1. Similarly, cells that are less than than other cells cannot contain the max allowed value, as there would be nothing greater to be filled on the other side of the inequality. In the example below, 1 cannot be filled in the red squares as they are all greater than other board squares, so the only possible placement for 1 on the first row of the board is the green square.

   To find forced cell values, or excluded cell values, you need to inspect the inequality constraints and remove digits from your 'possibles' lists for certain cells, according to what you find.

4. (Harder to implement) **`Chains of inequalities`**

   If you notice a chain of inequalities, be it either < (all ascending) or > (all descending), equal in size with the board's size, then that chain must be a sequence from 1 up to the length of the board. The length of the chain guarantees that this sequence is the only possible solution that satisfies the monotone condition imposed by the inequality chain.

5. There are other intelligent things that can be done. You can 'cross-reference' between the 'possibles' list for a cell, and the 'missing' values in the relevant row and column. If there is only one value that is common to all sets, then this value must go in the cell.

Marking scheme

1. 10 marks for a correct program that uses a **`recursive backtracking`** strategy with basic pruning.
2. 5 marks for adding the facility to choose 'singletons' ahead of other cells, using an approach that inspects cell values. With this approach, your solution should be able to solve the trivial puzzles.
3. 5 marks for finding 'forced' and 'excluded' cell values, using an approach that inspects inequality constraints. With this approach, your solution should be able to solve the easy puzzles.
4. 5 marks for more sophisticated pruning strategies, either those described here or others, that allow for fast backtracking over hard puzzles.

We will test your program using puzzles similar to those provided, for trivial, easy and hard categories.

Submission

Submit your project in Stream as a single zipped file containing your completed code, contained in the folder **`Futoshiki_YourID`**.

Plagiarism

There are many Futoshiki solvers available on the internet (and many more Sudoku solvers). I am sure that many of these are recursive backtracking solutions implemented in Python. For this reason, and also so that the concepts