Imperial College
London

Software Engineering 1: Introduction to Computing

Sudoku solving

Assignment 3, Spring 2014

Max Cattafi (m.cattafi@imperial.ac.uk)

Sudoku is a puzzle: given a 9x9 grid with some preassigned cells (containing a digit in the range from 1 to 9), the task is assigning a digit in the range from 1 to 9 to all the remaining cells avoiding repetitions in each row, column and 3x3 subgrid. This assignment deals with designing, implementing (in C or C++) and testing sudoku solvers.

Program interface

The program should read from the standard input the name of a text file containing an (incomplete) sudoku, read the input from the file, solve the sudoku and print the solution on the screen.

For instance if `sudoku1.txt` contains:

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

(with zeroes representing empty cells) and if the name of the executable for

your program is `sudokusolver`, then execution from the terminal would look like this:

```
$ ./sudokusolver
sudoku1.txt
4 8 3 9 2 1 6 5 7
9 6 7 3 4 5 8 2 1
2 5 1 8 7 6 4 9 3
5 4 8 1 3 2 9 7 6
7 2 9 5 6 4 1 3 8
1 3 6 7 9 8 2 4 5
3 7 2 6 8 9 5 1 4
8 1 4 2 5 3 7 6 9
6 9 5 4 1 7 3 8 2
```

It's important that the program doesn't print anything else and that the format of input from the standard input and from the file follows the example. Consider that the program will actually be run for testing as follows (thus test it also like this before submitting):

```
$ echo sudoku1.txt | ./sudokusolver
4 8 3 9 2 1 6 5 7
9 6 7 3 4 5 8 2 1
2 5 1 8 7 6 4 9 3
5 4 8 1 3 2 9 7 6
7 2 9 5 6 4 1 3 8
1 3 6 7 9 8 2 4 5
3 7 2 6 8 9 5 1 4
8 1 4 2 5 3 7 6 9
6 9 5 4 1 7 3 8 2
```

(if you don't understand the command format have a look again at the first few lab instructions from Autumn term).

Algorithms

Several algorithms (and related variations) can be implemented for a sudoku solver. Reading about the subject you will probably mostly come across "brute force" and "backtracking" approaches. These algorithms perform a search in the space of the candidate solutions.

The search can be more or less "intelligent", for instance a "brute force"

approach will try for the example mentioned above a full solution beginning with `1 1 3 ...`, thus placing two 1s in the first two free cells, even if it is clearly not a promising attempt.

A more refined "standard backtracking" algorithm will instead try to build a candidate solution incrementally. Thus, for instance, it would first try 1 for the first cell, would notice that the solution cannot have 1 in the first cell because there is already a 1 in the same subgrid, then it would try a 2, notice that there is already a 2 in the same row etc.

Notice also that one might start from another cell, not necessarily the top-left one. Other cells might be more promising and could be selected applying some variable selection heuristics (e.g. tentatively assign a value to the most constrained cell first).

Requirements

Up to 75% will be awarded for submissions including a sudoku solver (correctly and effectively) implementing an algorithm at least as sophisticated as the "standard backtracking" described above and a report explaining the problem and the algorithm.

A submission aiming at full marks should include several versions of the solver implementing different algorithms (and related variations, e.g. heuristics) and a report explaining the problem and the algorithms and containing an experimental evaluation section.

The experimental evaluation should compare the behaviour of several algorithms (and variations) on several instances in terms of time and/or steps required to find the solution in order to assess, for instance, the (respective and general) effectiveness of the algorithms (and of your implementation), how the behaviour changes on different instances, what features characterize a puzzle as difficult etc.

In any case it's very important that your code is readable and that it shows your design and implementation skills (familiarity with language-specific features, use of data structures, organization in functions, etc).