

Project (due Feb.23)

Sudoku Puzzle

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

Figure 1: Sudoku puzzle

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

Figure 2: Solution

Consider the classic 9-by-9 Sudoku puzzle in Figure 1. The goal is to fill in the empty cells such that every row, every column and every 3-by-3 box contains the digits 1 through 9. The solution to the puzzle is given in Figure 2 and it satisfies the following constraints:

- The digits to be entered are 1, 2, 3, 4, 5, 6, 7, 8, 9.
- A row is 9 cells wide. A filled-in row must have one of each digit. That means that each digit appears only once in the row. There are 9 rows in the grid, and the same applies to each of them.
- A column is 9 cells tall. A filled-in column must have one of each digit. That means that each digit appears only once in the column. There are 9 columns in the grid, and the same applies to each of them.
- A box contains 9 cells in a 3-by-3 layout. A filled-in box must have one of each digit. That means that each digit appears only once in the box. There are 9 boxes in the grid, and the same applies to each of them.

You are to write a program for solving the classic 9x9 Sudoku puzzle. Note that every puzzle has only **one** correct solution (see Figure 2).

Your program must be based on the search algorithms presented in class (e.g. backtracking, forward-checking etc.). Your algorithm should be *guaranteed* to find the answer within a reasonable amount of time (say less than 5 minutes). We have learned that this is easy to achieve, in fact the fastest programs last year only took milliseconds to solve even the hardest problems.

Note that *local* search algorithms can get stuck in local minima and are typically not guaranteed to find the correct answer (unless you *prove* otherwise). However, one could imagine that local search is so much faster that is worth it giving it a try and only switch to exhaustive search when it fails.

Besides the correct solution, your program must also output the running time (in seconds and milliseconds) it took to solve the puzzle. The programs will compete on some benchmarks to select the fastest solver. Fame will be bestowed upon the winner!

What to hand in

- All your source code (no class files) are to be zipped in a file called `<last name>_<student id>.zip` (e.g. smith_12345678.zip). You will upload this file into the EEE drop-box called Project 1.
- You will hand in (through the ICS distribution center) a report containing the description of your method as well as examples demonstrating that your program works correctly. Limit your discussion to 2-5 pages (Include the electronic version of the report in the zip file).

We require that you test your program on the puzzle from Figure 1. Additional puzzles will be posted in EEE. We will also organize a competition to determine the fastest puzzle solver.

General Instructions

- You may discuss the problem with other people but must develop your program independently.
- Your program **must** take as input a file containing the Sudoku puzzle (i.e. the program has only one argument which is the input file).
- We **require** the following file format: the puzzle is represented by a 9x9 matrix, where a "0" means that the respective cell is unassigned (the cells are separated by space characters). For example, the input file corresponding to the Sudoku puzzle in Figure 1 is given below.

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

ICS 171
Winter 2006

0 0 7 2 0 6 9 0 0
0 4 0 5 0 8 0 7 0

- Your program **must** output at the standard console the solution in a similar manner (the 0-s should be replaced by the correct digits). Your **must** also output the running time in seconds and milliseconds (or nanoseconds if it is really fast and it takes less than one millisecond!)
- You are **required** to use the Java programming language.
- The main class (i.e. containing the **main** function) **must** be named **SudokuSolver**. Here is an example of execution: `java SudokuSolver example1.txt`
We want to be able to execute your program without changing the code.
- Project descriptions will be discussed in sections. Feel free to contact your TA if you have any uncertainties.
- Additional info at: www.sudoku.com