ICS 171
Fall 2005

Project
(due November 10)

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).

ICS 171
Fall 2005

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 (but could take very long to find it), i.e. it must be optimal. 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

- Source code must be submitted through CHECKMATE (checkmate.ics.uci.edu).
- You will hand in a report containing the description of your method as well as examples demonstrating that your program works correctly. Limit your discussion to 2 pages.

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 should have two arguments. The first one is the name of the file containing the Sudoku puzzle. The second argument is the name of the output file and it should contain the solution. We suggest the following format: **sudoku puzzle_file solution_file.** We want to be able to run your program without changes to your source code.
- We recommend the following file format: the puzzle is represented by a 9x9 matrix, where a "0" means that the respective cell is unassigned. For example, the input file corresponding to the Sudoku puzzle in Figure 1 is given below. The solution file should respect a similar format.

```
060104050
008305600
200000001
800407006
006000300
700901004
500000002
007206900
040508070
```

ICS 171
Fall 2005

- You are **required** to use C/C++, Java or MATLAB programming languages.
- Project descriptions will be discussed in sections. Feel free to contact your TA if you have any uncertainties.
- Additional info at: www.sudoku.com