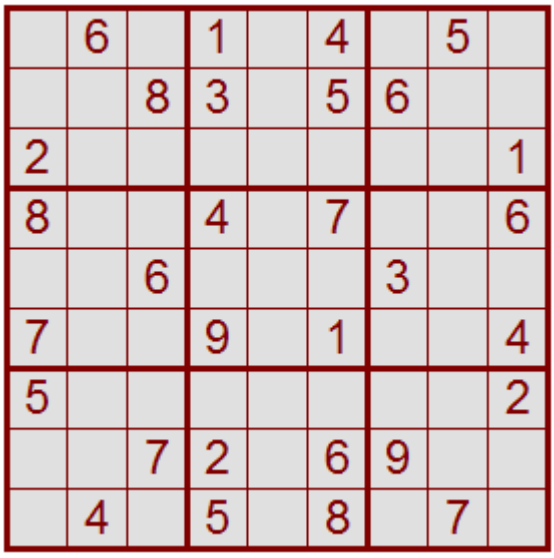
CS 171, Fall 2006
Prof. Rina Dechter

Project: Sudoku Solver

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 can find additional info on Sudoku at: www.sudoku.com

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).

Part 1 (due Tuesday, November 7 before 2:00pm)

Implement a backtracking search with the minimum remaining value (MRV) heuristic for the Sudoku puzzle.

In a Word document, provide a table that compares, for each of the test cases listed below, the performance (the number of nodes generated and their time) of the following two algorithms: 1) naïve backtracking (BT) without MRV, where the variables have a fixed ordering of your choice. 2) backtracking with MRV (BT-MRV) heuristic. Discuss your results.

Part 2 (due Tuesday, November 21 before 2:00pm)

BT-MRV which you implemented in part 1 of the project performs FC to propagate value assignments. In the second part of the project you are asked to augment backtracking with arc-consistency after each value assignment and use the MRV to select the variable ordering during search. Thus BT-MRV will be denoted BT-FC-MRV.

Extend the table in the Word document from part 1 to include, for each test case, backtracking with MRV and arc consistency during search (BT-AC-MRV). Compare the performance of the 3 algorithms (a. BT, b. BT-FC-MRV, c. BT-AC-MRV) and discuss your results.

d) In the same document, for each test case, provide the values remaining for each variable after running the arc consistency algorithm before search starts.

General Instructions

- You may discuss the problem with other people but must develop your program independently.
- The main class (i.e. containing the **main** function) **must** be named **SudokuSolver**. It is **strongly** recommended that you test your system on the command line as we will use the command line to grade each part. Projects that work “on your machine” or “only in Eclipse” will not receive full credit. Projects that do not compile will receive very little credit.
- Your program **must** take as command line arguments a file containing the Sudoku puzzle and the techniques to be used by the algorithm: MRV for minimum remaining value, AC for arc consistency to be used during search, ACP for arc consistency to be used before the search. The file containing the puzzle is

required, while the techniques are optional. If no techniques are specified, naïve backtracking is performed. The first command line argument should be the file, while the order of the others does not matter.

Examples:
java SudokuSolver puzzle1.txt MRV AC //backtracking with MRV and AC
java SudokuSolver puzzle1.txt AC MRV //same as above
java SudokuSolver puzzle1.txt //naive backtracking

- 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
  0 0 7 2 0 6 9 0 0
  0 4 0 5 0 8 0 7 0
- Your program **must** output to the screen 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!) and the number of nodes generated.
- You are **required** to use the Java programming language.
- Your source code **must** have comments.

Test Cases

We require that you run your experiments on the following three test cases. However, make sure that your system works for any legal 9x9 Sudoku puzzle. In addition to these three test cases, we will use other test cases to grade your system.

Test Case 1

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

Test Case 2

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

Test Case 3

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

What to hand in

When submitting each part, zip all of your source code and the Word document in a file called **<last name>_<student id>.zip** (e.g. smith_12345678.zip). Do not include any *.class files. You will upload the zip file for each part into the EEE dropbox called "Project Part 1" and "Project Part 2" respectively before 2:00pm on the due dates. Also, you **must** hand in a hardcopy of your source code and Word document stapled together in class on the due date for each part. Late projects will receive 0 credit. For a project to be on time, **both** your zip file and the hardcopy **must** be turned in on time.