# 22c:31 Project 4: Due 12/11/2013

---

For this assignment, you will improve the performance of a soduko puzzle solver based on the backtrack search.

In the code sudokuSolver.java, a simple backtrack search procedure is implemented that will read a sudoku puzzle from a file (some examples are sudoku1.txt, sudoku2.txt, sudoku3.txt, and sudoku4.txt) and print out the solution. For modern computers, its performance is okay for most sudoku puzzles. However its performance has a lot of space to be improved.

Suppose we measure the performance of the sudoku solver by the number of recursive calls to `backtrack`, which shows approximately how many times we filled an empty cell by a digit in the search process. In this project, you are asked to reduce the number of recursive calls to `backtrack`, by implementing the following two ideas:

- Idea 1: Use a different `int next(int pos)` that will return a position in the puzzle with the least number of feasible values.
- Idea 2: Use the arc-consistency algorithm AC3 to remove some feasible values from the domain of a cell. The AC3 is called after a cell c is signed a value. The initiation of `worklist` in the AC3 is changed to

  ```
  worklist := { (c, x) | x is an open cell in the same row, column, or region as c }.
  ```

  You need design data structures to store the domain of each open cell and updates to them during the backtrack search.

Please submit all the codes along with a transcript of three executions (of the original code, using idea 1 alone, and using both ideas) on each the four sudoku puzzles sudoku1.txt, sudoku2.txt, sudoku3.txt, and sudoku4.txt. Submit everything, including a table summerizing all the numbers of recursive calls to backtrack, in the ICON dropbox for Project4 before or on 12/11/2013.

Thank you!

---