CMPS 12B-01, Fall 2017: HW 4
N-queens renewed, redux without recursion

- All assignments must be submitted through git. Please look at the Piazza guide on submitting assignments.
- Please follow the naming conventions properly. Please look at the Piazza guide on the checking script. Run the checking script to make sure your files are named correctly. You will get no credit if the checking script fails!
- Follow instructions, and carefully read through the input/output formats.
- Clearly acknowledge sources, and mention if you discussed the problems with other students or groups. In all cases, the course policy on collaboration applies, and you should refrain from getting direct answers from anybody or any source. If in doubt, please ask the instructors or TAs.

1 Problem description

Main objective: Solve the n queens problems *without recursion.* You have to place n queens on an $n \times n$ chessboard such that no two attack each other. Important: the chessboard should be indexed starting from 1, in standard (x, y) coordinates. Thus, $(4, 3)$ refers to the square in the 4th column and 3rd row.

We will take as input the position of many queens, and have to generate a solution with queens in this positions. (Remember, in HW1, there was only one input queen. Now, there are many.)

You cannot make any recursive calls. If you make a recursive call, you will get no credit.

Suggestions for coding: Use a stack. You can use any in-built stack that java provides, and can store the input chess pieces however you wish. To stress, you do not need to write your own stack.

Think about the box-trace (which is the same as the stack frames) for the recursive solution for NQueens. See how your stack solution will perform the same backtracking. It's not a bad idea to start from your recursive solution for HW1, and adapt it using stacks.

Format: You should provide a Makefile. On running `make`, it should create "NQueens.jar". You should run the jar with *two* command line arguments: the first is an input file, the second is the output file. For example, we would call the command `java -jar NQueens.jar in.txt solution.txt`. The file `in.txt` will have inputs to the main program (as described below), and the file `solution.txt` will have the desired solution.

Each line of the input file corresponds to a different instance. Each line of the input file will have three integers: the chessboard size, the column where the input queen is placed, and the row where the input queen is placed. For example, the file may look like:

8 4 4 6 3
11 4 4 6 3

The first line means we have a 8×8 chessboard, with two queens placed at $(4, 4), (6, 3)$. We wish to place 6 more queens without any attacking the other. The second line means we have a 11×11 chessboard, with two queens placed at $(4, 4), (6, 3)$. We wish to place 9 more queens without any attacks. So on and so forth.

Output: On running the command, the following should be printed in the output file. For each line of the input file, there is a line in the output file with the placement of queens.

- If there is no solution to the problem, print "No solution" (with a newline at the end)
- If there is a solution, print the position of each queen as `<column> <space> <row> <space>` followed by the position for the next queen. The positions should be in increasing order of column, so first column first, second column second, etc. Please follow this format exactly, so that the checking script works for you.

For example, the output for the input describe above could be

No solution
1 2 2 8 3 10 4 4 5 9 6 3 7 5 8 7 9 11 10 1 11 6

Interestingly, when you place queens as $(4, 4)$ and $(6, 3)$, there is no solution when the board has size 8. But there is a solution for a board of size 11. Note that when a solution exists, it may not be unique. So you may get solutions different from the examples provided.

Helper code: On the HW website, you will find a java file that prints out your solution on chessboard (with queen positions) on to your console. This is extremely helpful in checking if your solution is correct. Compile and execute the java code on terminal directly using the commands below:

- `javac PrintSolutionHelper.java`
- `java PrintSolutionHelper <your output file>`

For each line in your output file that has a list of queens positions, the console will print the chessboard with queens on then. (The code is actually pretty interesting!)

Example commands and outputs: On the HW website, you will find a file `Examples.zip`. Download and unzip it. There are a number of input and corresponding output files.

- `test-input.txt, test-output.txt`. If you run your program with input file `test-input.txt`, the output file should be exactly the same as `test-output.txt`. This will be used by the checking script.

2 Grading

You code should terminate within 3 minutes for all runs (maximum chessboard size will be 13). You get no credit for a run that does not provide a proper output.

1. (10 points) For a full solution as described above.
2. (7 points) Only solves the problem when there is a single input queen.
3. (5 points) Only solves the problem when there is a single input queen in the first column.