CS 171: Introduction to Computer Science II
Assignment #3: N-Queens

(Due: February 22 at 11:59 PM)

Goals

This assignment practices the use of Stacks and using the backtracking algorithm to solve the N-Queens problem.

Problem

The N-Queens problem is a popular puzzle. Assume that you have an $N \times N$ chessboard, and N queens. The problem asks you to place all N queens on the board so that no queen can attack another queen. Two queens are attacking each other if they are on the same row, column or the same diagonal. The diagrams below show examples of invalid positions of two queens that conflict with each other:

```
* Q * * *          * * * * *          * * * * Q          * * * * *
* * * * *          * Q * * Q          * * * * *          * Q * * *
* * * * *          * * * * *          * * * * *          * * * * *
* Q * * *          * * * * *          * * * * *          * * * Q *
* * * * *          * * * * *          Q * * * *          * * * * *
(same column)     (same row)         (same diagonal)       (same diagonal)
```

On the other hand, the diagrams below show valid positions of two queens, which do not conflict with each other:

```
* Q * * *          * * * * *          * * * * Q          * * * * *
* * * * *          * Q * * *          * * * * *          * Q * * *
* * * * *          * * * * Q          * * * * *          * * * * *
* * Q * *          * * * * *          Q * * * *          * * Q * *
* * * * *          * * * * *          * * * * *          * * * * *
```

For a given N, there are usually more than one valid solution. For example, the following solutions are valid for N = 5:

```
Q * * * *          * * Q * *          * Q * * *
* * Q * *          * * * * Q          * * * Q *
* * * * Q          * Q * * *          Q * * * *
* Q * * *          * * * Q *          * * Q * *
* * * Q *          Q * * * *          * * * * Q
```

Algorithm

1. *Backtracking:* There are many algorithms for finding solutions to the N-Queens problem. One of them is a simple backtracking algorithm, which you will implement in this assignment using a stack. The basic idea of this algorithm is to place one queen on each row, one at a time, but you need to solve for the position of the queen on each row so that no two of them have a column conflict or a diagonal conflict.

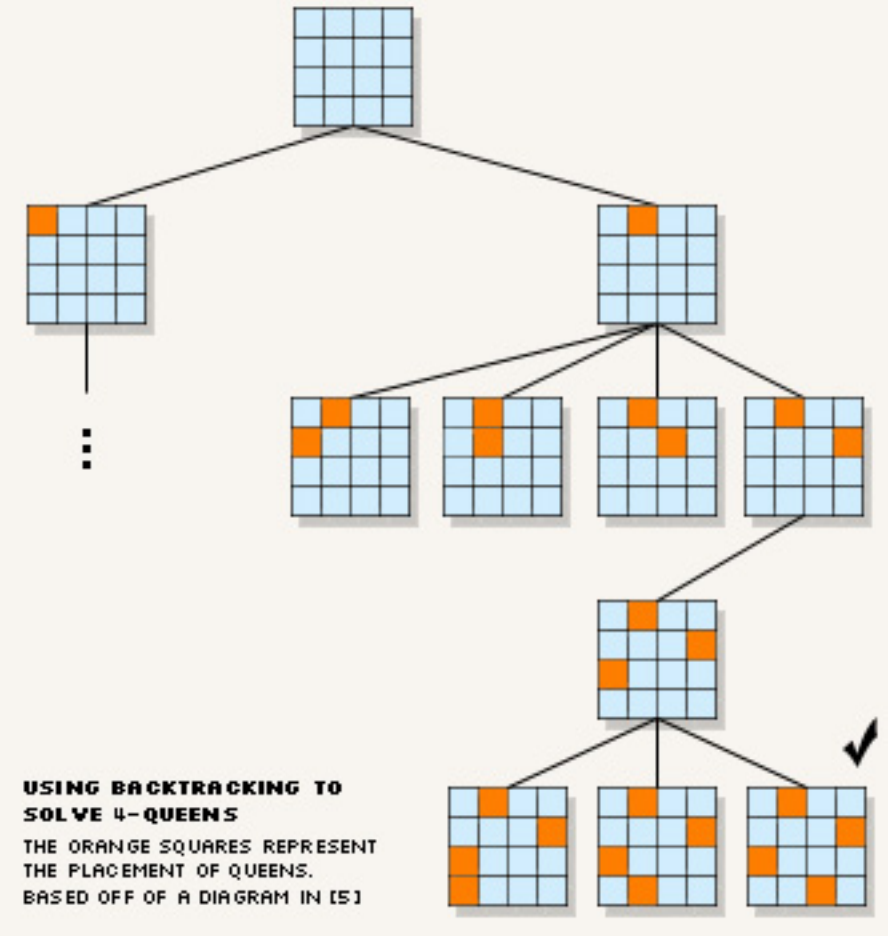

Figure 1: Example of backtracking for 4-queens

To do so, you start from the first row, and try to place the queen at each of the N available spots (columns) of that row. However, before placing it, you need to check whether it conflicts with the queens already on the board. Note that when you are working on row k, there should be k - 1 queens already placed on the board. If a spot is invalid (i.e. has conflicts with an existing queen in a column or diagonal), it will try the next slot (column); otherwise, if the slot is valid, the algorithm will then record the position of the queen, and continue to work on the next row. What happens if you reach the end of a row but cannot find a valid spot? Well, you backtrack – moving back to the previous row and trying to place the queen further down in that row.

In a way, this is quite similar to how you solve a maze problem: you start walking in a maze, and when you are at a crossing faced with several turns, you pick one turn and keep walking. At some point you may find that the path you picked is a dead end, so you need to backtrack to the last turn and pick another one, then keep trying until you are out of the maze! See Figure 1 for an example illustration on the 4-Queens problem.

2. *Stack:*

   The algorithm can be implemented using recursion (which you may have learned in CS170 and we will learn more about in this course). In this assignment, you **must use an explicit**

stack to keep track of the positions of queens placed so far (i.e., no recursion). Each time the algorithm successfully places a queen in a row, it pushes the column position of that queen onto the stack and starts working on the next row. If it can't find a valid spot, the algorithm backtracks: it returns to the previous row by popping the top element of the stack. This popped element indicates where you were in that previous row. You should then continue searching for valid spots further down in that row. Again, as soon as you find a valid spot, push it to the stack and work on the next row.

When you find a valid spot on the last row, then congratulations, you've got a solution. At this moment, the positions of queen stored in the stack represent the solution. As an example, look at the three example solutions given above for the 5-Queens problem. The stack elements corresponding to the first solution are: 0, 2, 4, 1, 3. Note that the position in each row starts at 0, similar to an array index. Likewise, the second solution corresponds to stack elements 2, 4, 1, 3, 0; and the third solution is: 1, 3, 0, 2, 4.

Your algorithm should find and print out not just one, but all solutions to the N-Queens problem, given some N as input. To find more solutions beyond one, simply continue the searching from the last row.

Starter code

Starter code is provided. Note that static methods are called directly from main(). If you write any new methods, be sure to make them static as well. The methods of interest are

- `public static int solve(int n)`

  This method is the primary method called by main() that will search for all solutions to the N-Queens problem given N as a parameter. It returns the number of solutions found. It should also print out each solution by calling the printSolution() method each time a solution is found.

- `private static void printSolution(Stack<Integer> s)`

  This method prints out a solution in the same manner as the examples provided above (using *'s for empty spaces and Q's for queens). Each element of the stack represents the column position of a queen. This method has been written and should not need to be changed.

The given code uses the Stack class in the java.util package to maintain the search stack. Please check out its API and familiarize yourself with the class.

The only parameter to this program is the number N. By default, N=8, corresponding to the 8-Queens problem. You can pass in a different N by providing it as a command line parameter. For example, the command java NQueens 11 tells the program that N=11. Alternatively, if you work in Eclipse, you can set the command line parameter by clicking on menu item 'Run', then 'Arguments' tab, and input 11 in the 'Program arguments' edit box.

Requirements

Your assignment mainly consists of writing the `solve()` method, though you will probably want to write one or two helper methods as needed. Be sure to make them static. The default size of the problem is 8 (a standard chessboard), but you can specify an alternate problem size as a command line parameter. Your program should run correctly for a problem of any size. When

testing your program, start with a small N (e.g. 4), so that it's easy to do step-by-step inspection of your program.

Experiment and measure the runtime of your program by trying different problem sizes. Be mindful though that large numbers will be infeasible to run to completion, so start with small increments. Additionally, printing out the solutions slows down the search significantly. When experimenting, you can comment out the call to printSolution() to see how long your program takes to find all solutions. Be sure to leave that line uncommented when you submit your work.

Getting started

- Download the starter code on Canvas and understand it.
- Fill in your implementation. Remember that when a solution is found, you should call `printSolution()` to print it out. Also, at the end of the `solve()` function, return the total number of solutions you've found.
- Run and test your program with various inputs.

Submission Instructions

The programming assignment should be submitted electronically via the turnin program. From a terminal, you should type the commands:

```
>cd cs171
>home/cs171003/turnin NQueens.java hw3
```

. Make sure it can be compiled and run against Java Development Kit 8, the version which will be used to compile and run the program.

Grading

- If your program does not compile, you will get 0 points
- Note that you must use the stack to solve this problem. If your solution uses recursion, you will get 0 points.
- Correctness
  - Your program outputs at least one correct solution for the default N = 8 (15 pts).
  - Your program outputs all correct solutions (there are 92 distinct solutions) for the default N = 8 (40 pts)
  - Your program outputs all correct solutions for any N (testing from 1 to 12) (15 pts)
- Robustness (your program never crashes for any N (from 1 to 12)) (10 pts)
- Understanding (since this is a popular puzzle and there are many solutions available online, your code should contain detailed comments that reflect your understanding of the solution) (15 pts)
- Code clarity and style (5 pts)

Honor Code

The assignment is governed by the College Honor Code and Departmental Policy. Please remember to have the following comment included at the top of the file.

```
/*
THIS CODE WAS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING
CODE WRITTEN BY OTHER STUDENTS. _Your_Name_Here_
*/
```