Programming Assignment #3: SneakyKnights

COP 3503, Fall 2016

Due: Tuesday, September 27, *before* 11:59 PM

Abstract

This problem is similar to SneakyQueens, but the solution requires a bit of a twist that will push you to employ your knowledge of data structures in clever ways. It has more restrictive runtime and space complexity requirements, as well: Your program needs to have an average-case / expected runtime complexity that does not exceed O(*n*) (where *n* is the number of knights), and a worst-case space complexity that does not exceed O(*n*).

The assignment also has a different board size restriction: The maximum board size is now `Integer.MAX_VALUE` × `Integer.MAX_VALUE`.

Please feel free to seek out help in office hours if you're lost, and remember that it's okay to have conceptual discussions with other students about this problem, as long as you're not sharing code (or pseudocode, which is practically the same thing). Just keep in mind that you'll benefit more from this problem if you struggle with it a bit before discussing it with anyone else.

Deliverables

SneakyKnights.java

(Note: Capitalization of your filename matters!)

1. Problem Statement

You will be given a list of coordinate strings for knights on an arbitrarily large square chess board, and you need to determine whether any of the knights can attack one another in the given configuration.

In the game of chess, knights can move two spaces vertically (up or down) and one space to the side (left or right), or they can move two spaces horizontally (left or right) and one space vertically (up or down). For example, the knight on the following board (denoted with a letter ‘N’, since the letter ‘K’ is traditionally reserved for the king in formal chess notation systems) can move to any position marked with an asterisk (‘*’), and no other positions:

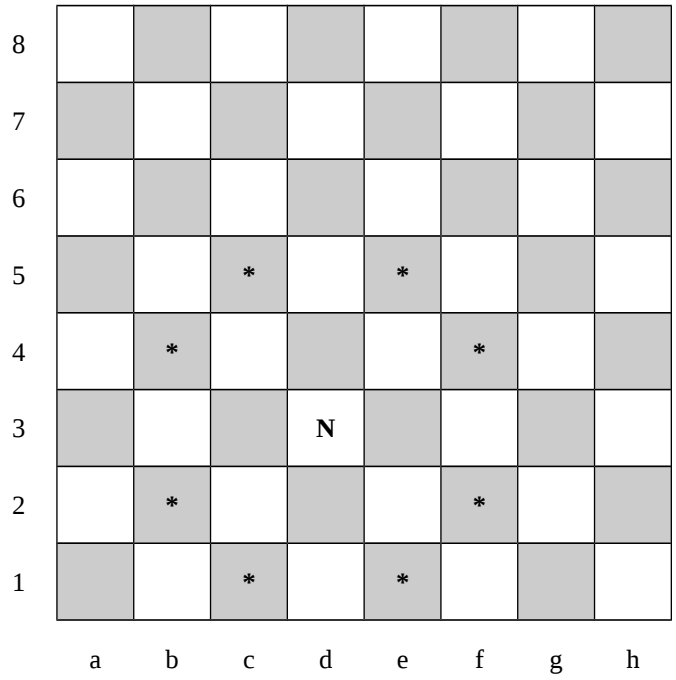

Figure 1: *The knight at position d3 can move to any square marked with an asterisk.*

Thus, on the following board, none of the knights (denoted with the letter ‘N’) can attack one another:

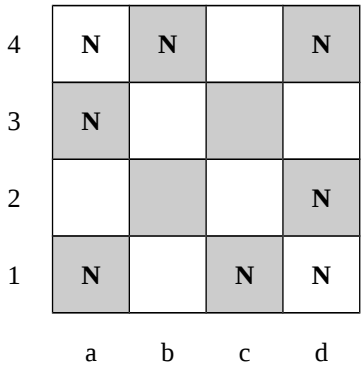

Figure 2: *A 4x4 board in which none of the knights can attack one another.*

In contrast, on the following board, the knights at *c6* and *d8* can attack one another, as can the knights at c*6* and *d4*:

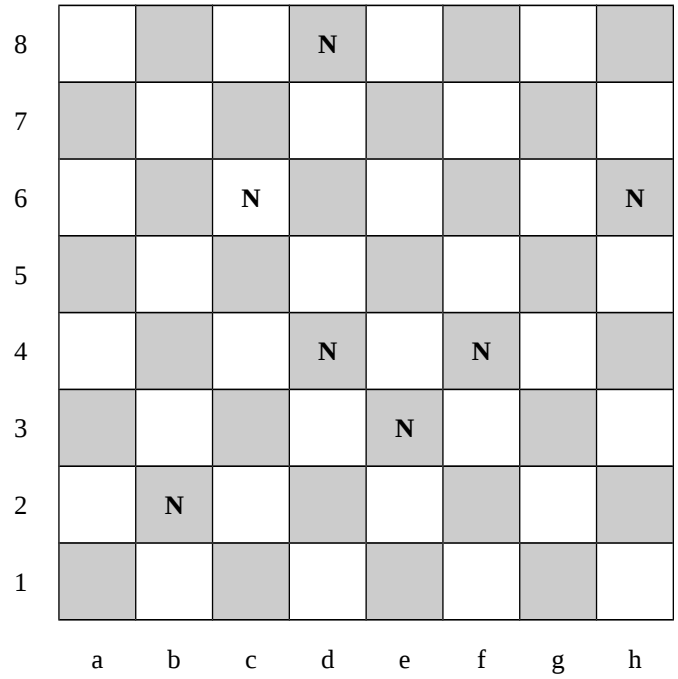

Figure 3: *An 8x8 board in which some of the knights can attack one another.*

2. Coordinate System

This program uses the same coordinate system given in the SneakyQueens assignment.

3. Runtime and Space Requirements

In order to pass all test cases, the average-case / expected runtime of your solution cannot exceed O(n), and the worst-case space complexity cannot exceed O(*n*) (where *n* is the number of coordinate strings to be processed).

As with SneakyQueens, this O(*n*) figure assumes that the length of each coordinate string is bounded by some constant, which means you needn't account for that length in your runtime analysis, provided that each string is processed or examined only some small, constant number of times (e.g., once or twice). Equivalently, you may conceive of all the string lengths as being less than or equal to *k*, in which case the average-case runtime complexity and worst-case space complexity that your solution cannot exceed would be expressed as O(*nk*).

Continued on the following page...

4. Method and Class Requirements

Implement the following methods in a class named `SneakyKnights`. Please note that they are all **`public`** and **`static`**. You may implement helper methods as you see fit.

```
public static boolean
allTheKnightsAreSafe(ArrayList<String> coordinateStrings, int boardSize)
```

> **Description:** Given an ArrayList of coordinate strings representing the locations of the knights on a *boardSize* × *boardSize* chess board, return `true` if none of the knights can attack one another. Otherwise, return `false`.
>
> **Parameter Restrictions:** *boardSize* will be a positive integer less than or equal to *Integer.MAX_VALUE*. *boardSize* describes both the length and width of the square board. (So, if *boardSize* = 8, then we have an 8 × 8 board.) *coordinateStrings* will be non-null, and any strings within that ArrayList will follow the format described above for valid coordinates on a *boardSize* × *boardSize* board. Each coordinate string in the ArrayList is guaranteed to be unique; the same coordinate string will never appear in the list more than once.
>
> **Output:** This method should **not** print anything to the screen. Printing stray characters to the screen (including newline characters) is a leading cause of test case failure.

```
public static double difficultyRating()
```

> Return a double indicating how difficult you found this assignment on a scale of 1.0 (ridiculously easy) through 5.0 (insanely difficult).

```
public static double hoursSpent()
```

> Return an estimate (greater than zero) of the number of hours you spent on this assignment.

5. Compiling and Testing SneakyKnights on Eustis

Recall that your code must compile, run, and produce precisely the correct output on Eustis in order to receive full credit. Here's how to make that happen:

1. To compile your program with one of my test cases:

```
javac SneakyKnights.java TestCase01.java
```

2. To run this test case and redirect your output to a text file:

```
java TestCase01 > myoutput01.txt
```

3. To compare your program's output against the sample output file I've provided for this test case:

```
diff myoutput01.txt sample_output/TestCase01-output.txt
```

If the contents of `myoutput01.txt` and `TestCase01-output.txt` are exactly the same, `diff` won't print anything to the screen. It will just look like this:

```
seansz@eustis:~$ diff myoutput01.txt sample_output/TestCase01-output.txt
seansz@eustis:~$ _
```

Otherwise, if the files differ, `diff` will spit out some information about the lines that aren't the same.

6. Using the run-all-test-cases.sh Script

I've also included a script called `run-all-test-cases.sh` that you can use to run all the test cases I released with this assignment. To run the script, you must go through the following steps:

1. Upload the script to Eustis, and then issue the following command in Eustis to make the script executable. You only have to run this command once. After that, the file will remain executable:

```
seansz@eustis:~$ chmod +x run-all-test-cases.sh
```

2. Once the script is executable, you can run it like so:

```
seansz@eustis:~$ ./run-all-test-cases.sh
```

Note! Because the last two test cases are very large, you might not be able to transfer them to Eustis without exceeding your disk quota there. You might have to run only the first seven tests on Eustis.

7. Grading Criteria and Miscellaneous Requirements

The *tentative* scoring breakdown (not set in stone) for this programming assignment is:

80%	passes test cases (in linear runtime)
10%	`difficultyRating()` and `hoursSpent()` are implemented correctly
10%	adequate comments and whitespace; source code includes your name

Please be sure to submit your `.java` file, not a `.class` file (and certainly not a `.doc` or `.pdf` file). Your best bet is to submit your program in advance of the deadline, then download the source code from Webcourses, re-compile, and re-test your code in order to ensure that you uploaded the correct version of your source code.

Important! Programs that do not compile will receive zero credit. When testing your code, you should ensure that you place SneakyKnights.java alone in a directory with the test case files (source files, input_files directory, and sample_output directory), and no other files. That will help ensure that your SneakyKnights.java is not relying on external support classes that you've written in separate .java files but won't be including with your program submission.

Important! You might want to remove `main()` and then double check that your program compiles without it before submitting. Including a `main()` method can cause compilation issues if it includes references to home-brewed classes that you are not submitting with the assignment. Please remove.

Important! Your program should not print anything to the screen. Extraneous output is disruptive to the TAs' grading process and will result in severe point deductions. Please do not print to the screen.

Important! Please do not create a java `package`. Articulating a `package` in your source code could prevent it from compiling with our test cases, resulting in severe point deductions.

Important! Name your source file, class(es), and method(s) correctly. Minor errors in spelling and/or capitalization could be hugely disruptive to the grading process and may result in severe point deductions. Similarly, failing to make any of the three required methods, or failing to make them `public` and `static`, may cause test case failure. Please double check your work!

Input specifications are a contract. We promise that we will work within the confines of the problem statement when creating the test cases that we'll use for grading. For example, the strings we pass to `allTheKnightsAreSafe` are guaranteed to be properly formed coordinate strings. None of them will contain spaces, capital letters, punctuation, or other characters that would violate the coordinate notation system described in this writeup. Similarly, we will never pass a `boardSize` value less than 1 or greater than `Integer.MAX_VALUE` to your `allTheKnightsAreSafe` method.

However, please be aware that the test cases included with this assignment writeup are by no means comprehensive. Please be sure to create your own test cases and thoroughly test your code. Sharing test cases with other students is allowed, but you should challenge yourself to think of edge cases before reading other students' test cases.

Start early! Work hard! Ask questions! Good luck!