Programming Assignment #3: SneakyKnights

COP 3503, Spring 2021

Due: Sunday, February 14, *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(*nk*), and a worst-case space complexity that does not exceed O(*nk*) (where *n* is the number of knights and *k* is the max length of any of their coordinate strings).

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! The capitalization and spelling of your filename matter!

Note! Code must be tested on Eustis, but submitted via Webcourses.

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:

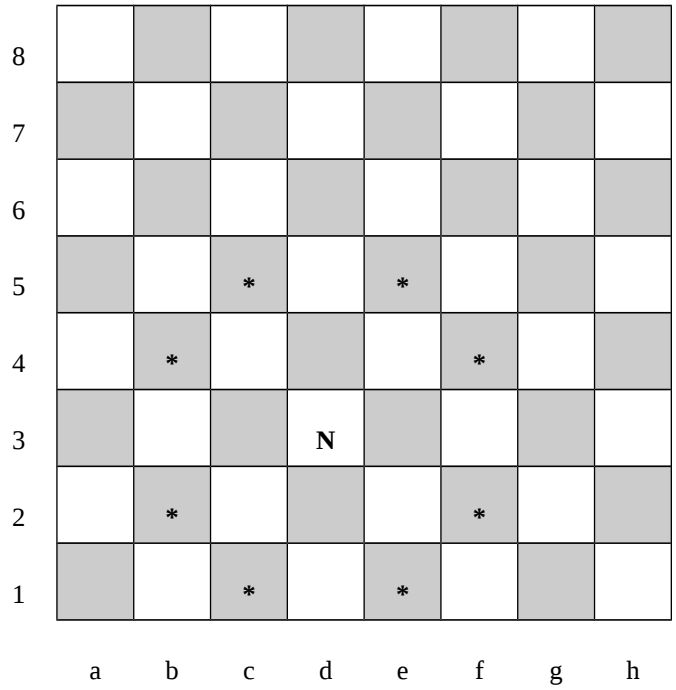

Figure 1: *The knight at position d3 can move to any square marked with an asterisk.*

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

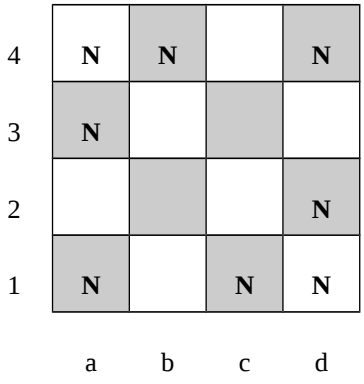

Figure 2: *A 4x4 board in which none of the knights can attack one another.*

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*:

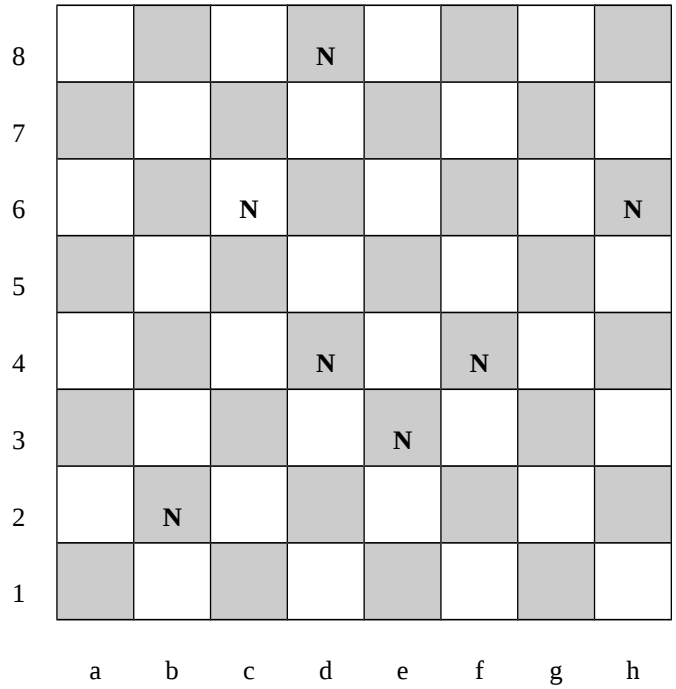

Figure 3: *An 8x8 board in which some of the knights can attack one another.*

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(*nk*), and the worst-case space complexity cannot exceed O(*nk*) (where *n* is the number of coordinate strings to be processed and *k* is the maximum length of any of those coordinate strings). So, you can only process the full length of each coordinate string some small, constant number of times.

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 a realistic and reasonable estimate (greater than zero) of the number of hours you spent on this assignment.

5. Style Restrictions (Same as in Program #1) (*Super Important!*)

Please conform as closely as possible to the style I use while coding in class. To encourage everyone to develop a commitment to writing consistent and readable code, the following restrictions will be strictly enforced:

- ★ Capitalize the first letter of all class names. Use lowercase for the first letter of all method names.
- ★ Any time you open a curly brace, that curly brace should start on a new line.
- ★ Any time you open a new code block, indent all the code within that code block one level deeper than you were already indenting.
- ★ Be consistent with the amount of indentation you're using, and be consistent in using either spaces or tabs for indentation throughout your source file. If you're using spaces for indentation, please use at least

two spaces for each new level of indentation, because trying to read code that uses just a single space for each level of indentation is downright painful.

- ★ Please avoid block-style comments: /* *comment* */
- ★ Instead, please use inline-style comments: // *comment*
- ★ Always include a space after the "//" in your comments: "// *comment*" instead of "//*comment*"
- ★ The header comments introducing your source file (including the comment(s) with your name, course number, semester, NID, and so on), should always be placed ***above*** your import statements.
- ★ Use end-of-line comments sparingly. Comments longer than three words should always be placed ***above*** the lines of code to which they refer. Furthermore, such comments should be indented to properly align with the code to which they refer. For example, if line 16 of your code is indented with two tabs, and line 15 contains a comment referring to line 16, then line 15 should also be intended with two tabs.
- ★ Please do not write excessively long lines of code. Lines must be no longer than 100 characters wide.
- ★ Avoid excessive consecutive blank lines. In general, you should never have more than one or two consecutive blank lines.
- ★ Please leave a space on both sides of any binary operators you use in your code (i.e., operators that take two operands). For example, use *(a + b) - c* instead of *(a+b)-c*. (The only place you do ***not*** have to follow this restriction is within the square brackets used to access an array index, as in: *array[i+j]*.)
- ★ When defining or calling a method, do not leave a space before its opening parenthesis. For example: use *System.out.println("Hi!")* instead of *System.out.println ("Hi!")*.
- ★ Do leave a space before the opening parenthesis in an *if* statement or a loop. For example, use use *for (i = 0; i < n; i++)* instead of *for(i = 0; i < n; i++)*, and use *if (condition)* instead of *if(condition)* or *if(condition)*.
- ★ Use meaningful variable names that convey the purpose of your variables. (The exceptions here are when using variables like *i*, *j*, and *k* for looping variables or *m* and *n* for the sizes of some inputs.)
- ★ Do not use *var* to declare variables.

6. Compiling and Testing SneakyKnights on Eustis (and the *test-all.sh* Script!)

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.

4. I've also included a script, *test-all.sh*, that will compile and run all test cases for you. You can run it on Eustis by placing it in a directory with *SneakyKnights.java* and all the test case files and typing:

   ```
   bash test-all.sh
   ```

   Note! Because the last four 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 five tests on Eustis. If you're running these test cases on your own system, you'll want to force the *test-all.sh* script to run all test cases (including the very large ones) by typing:

   ```
   bash test-all.sh --include-the-really-big-test-cases
   ```

   Super Important: Using the *test-all.sh* script to test your code on Eustis is the safest, most sure-fire way to make sure your code is working properly before submitting.

7. Grading Criteria and Miscellaneous Requirements

> ***Important Note:*** When grading your programs, we will use different test cases from the ones we've released with this assignment, to ensure that no one can game the system and earn credit by simply hard-coding the expected output for the test cases we've released to you. You should create additional test cases of your own in order to thoroughly test your code. In creating your own test cases, you should always ask yourself, "What kinds of inputs could be passed to this program that don't violate any of the input specifications, but which haven't already been covered in the test cases included with the assignment?"

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

80%	Passes test cases in O(*nk*) time with 100% correct output formatting. This portion of the grade includes tests of the *difficultyRating()* and *hoursSpent()* methods.
20%	Adequate comments and whitespace. To earn these points, you must adhere to the style restrictions set forth above. We will likely impose huge penalties for small deviations. Please include a header comment with your name and NID.

Your program must be submitted via Webcourses.

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 on Eustis 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, sample output files, and the input text files associated with the test cases), 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! No file I/O. In the methods you write, please do not read or write to any files.

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!