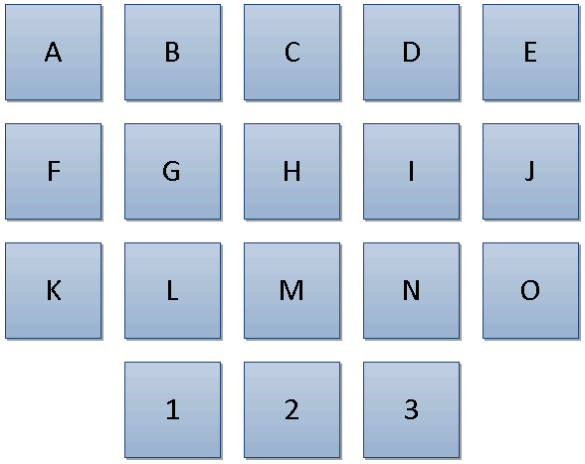
Knight Moves Problem

Pictured here is a keypad:



We want you to count all key sequences of length *n* that can be entered into the keypad in the following manner:

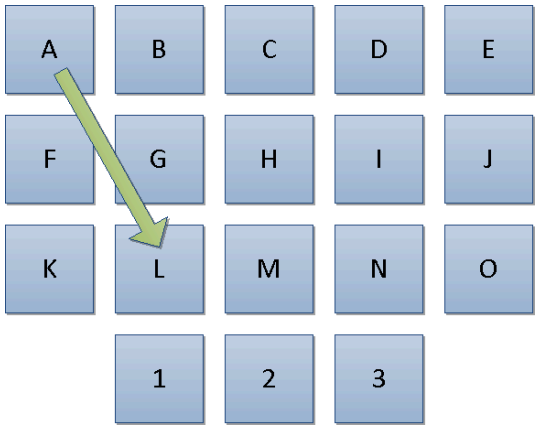
- The initial keypress can be any one of the keys.
- Each subsequent keypress must be a *knight move* from the previous keypress.
- There can be at most 2 vowels in the sequence.
- *n* can take any value between 1 and 32, inclusive

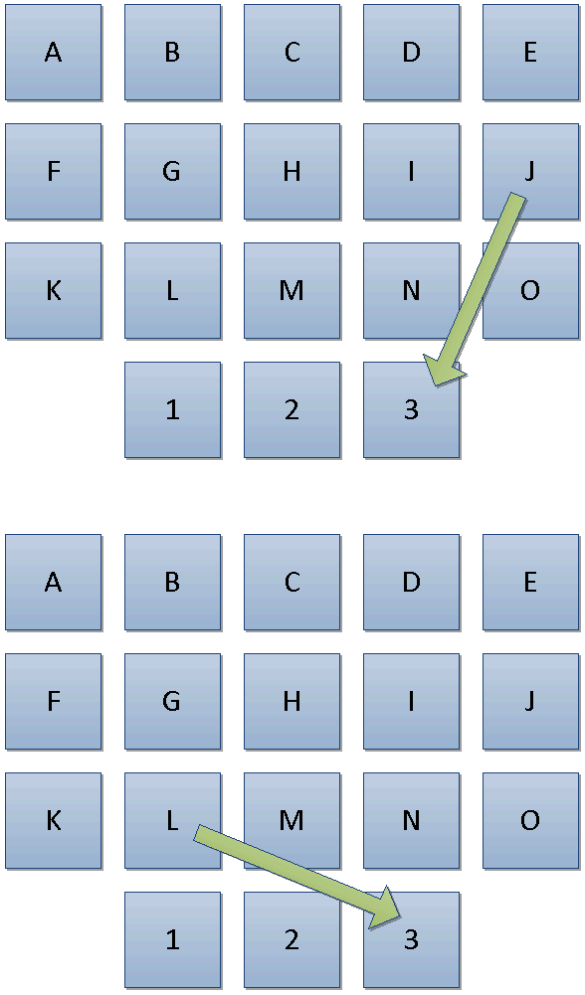
A knight move is made in one of the following ways:

- Move two steps horizontally and one step vertically.
- Move two steps vertically and one step horizontally.

It does not matter which direction or length the knight moves first. For example, a knight could not move two steps down from F and then one over to reach the 1 key, because it would leave the board. However, it could reach the 1 key legally by instead moving one step to the right of F and then two down. Wrapping from top to bottom or from one side of the keypad to another is not allowed.

Here are some examples of knight moves:





Your program will be run with a single integer *n* (between 1 and 32, inclusive) as a command-line parameter. Given this value for *n*, your program should print out the total count of valid *n*-key sequences on a single line to standard out. Note that *n* is the number of keys pressed, not the number of knight moves made.

Frequently Asked Questions

1. What language should I use?

   We strongly suggest you use the Java language unless there's a language you feel stronger with.

2. What should I submit?

   A zip file with the solution inside a directory is the best way to submit your response. If you want to throw in a readme file to explain anything, a simple .txt file is best. Send the zip file as an attachment on a reply to this email message.

3. I use third party library X for everything that I do. Can I use it in this solution?

   Sure. Please let us know that you have done that in your submission. If you do, you should include a gradle or maven project file so that we can easily build your solution.

4. What will you do with my submission?

   We will compile and run it. We will also be spending a fair bit of time looking at the code itself and that may lead to some interesting conversations in later steps. We will attempt to run your solution at lengths 10, 16, and 32.

5. Is there a “right” solution?

   The program should produce the correct number on the output, but failing to get the number right will not immediately disqualify you. Beyond getting the correct answer, we are interested in seeing how you solve the problem and how you program. There are many ways to solve this problem, and each has advantages and disadvantages. Choose the approach you feel is best.

6. How much time do I have?

   The sooner you send in a solution, the sooner we can move forward with the interview process. However, we are using your submission to judge your programming skills and habits, so take whatever time you need to submit something representative of how you would write code if you worked here. We know that you may be very busy, so we do not impose any particular deadline on your submission.

7. Are there any tricky parts to understanding the problem?

   There are no tricks here. If you think you have spotted a cleverly hidden edge case we put into the problem to trip you up, then you are probably overcomplicating things and you should re-read the problem statement. The only thing that is tricky is writing an efficient implementation.

8. Can the same key be used more than once?

   It is legal for a key to be used multiple times in the same sequence, subject to the constraints on sequence length and number of vowels. People familiar with the related-sounding "Knight's Tour" problem sometimes read in a requirement that the path never visit the same key twice. That is NOT a part of this problem.

9. What should the output of my program be?

   The only thing required on the output is the number of sequences. If you want to include some particularly interesting additional output, that is allowed, but the program should clearly print this single number when it runs.

10. What do you mean when you say that paths “do not wrap?”

    What this means is that a knight cannot step off one side the board and reappear on the opposite side. If a move would go off of the board, then it is not a legal move.

11. Is there any way to tell if I have the right solution?

    For $n = 10$, we expect the number which is printed out to contain each of the following digits at least once: 1, 3, 8, 9. If your solution does not contain each of these digits, then it is probably not exactly what we are looking for.

If you’d like to include any notes which explain the computational complexity of your algorithm, please feel free.