Search for the best

Before you can start with your internship the company asks you to take their developer test which every new developer has to do as part of their onboarding program. It consists of two small assignments.

Assignment 1: Backwards search

You have to implement a text-based searching algorithm that finds the right most occurrence of the text searched for. This in contrast to most algorithms that find the left most occurrence. This brings you to a beautiful insight. Why not use a reverse version of the Boyer-Moore algorithm. This way you will always find the right most occurrence as fast as possible. Besides determining where the needle can be found, your code must be able to return the number of character comparisons of the last search.

You may assume that all text is in lowercase and does not contain spaces, punctuations or special characters.
No clever tricks such as reversing the haystack and/or the needle, converting to StringBuffer, Streams or whatever and using the standard Java search implementations are allowed!

You are allowed to look for examples on the internet, when you do you must however clearly state the original code's origin!

Deliverables

- The code that implements the reverse Boyer-Moore search
- Describe clearly what you have changed compared to the original Boyer-Moore implementation and most importantly why. (you are allowed to use the code from Sedgewick)
- Provide tests that clearly show that your reverse Boyer-Moore implementation is more efficient (needs less character comparisons) than the code from the book. (You might need to find some large bodies of text that can be used for these tests.)

Tips

Although the implementation does not need a lot of modifications here are some tips to help you.

- Since you can assume only lowercase characters are used in both the pattern and the text you can place the shift value for 'a' at index 0.
- Since you are implementing the algorithm in reverse order you might need to change some minuses into pluses and visa versa and change the initial value and condition in some loops.
- The initial values for the shifts are different from the standard implementation. Start by investigating with what should happen when the first character (the left most) results in a mismatch with the mismatching character does not appear in the pattern. How many characters should the patterns shift to the left? Once you figured that out, what should the shift value be when the mismatch happens somewhere else.

Assignment 2: Code quality

Because English is the common language in the IT industry the company wants to ensure that all the documentation (JavaDoc) is written in English. They've asked you to come up with a simple and elegant solution for detecting if the JavaDoc of any given Java source file is indeed written in English. This language detection must be able to run offline, so using any of the services provided by Google and other companies is out of the question.
One other aspect the company wants to be checked is the number of method calls being done in a Java source file. This number is not allowed to exceed a certain (yet to be determined) threshold.

Lucky for you, you watched a movie about Bletchley Park lately. One of the easiest to break encryption schemes mentioned turns out to be quite useful. Every language has a unique character frequency distribution. So, you could compare the frequency distribution of the characters in the JavaDoc with that of a known distribution of the English language.

It is your task to write the language detection algorithm and the determination of the number of method-calls for a given Java source file. There is however one very strict restricting: *all this has to be done using regular expressions only*!

Deliverables

Write a method that determines the frequency distribution of the combined multi-line JavaDoc and compare it graphically with a standard distribution for the English language and another language, Dutch for example. And explain if this method turns out to be useful.
Write a method that prints the number of method calls, and the calls themselfs, for a Java source file.

Tips

- For working with regular expressions in Java there are plenty of online tutorials, so make use of them.
- Have a look at the `compile`, `find`, and `group` methods from the `Pattern` and `Matcher` classes.
- Transform the JavaDoc to either lower-case or upper-case before determining the frequency distribution. (You can leave out any non-alphabet character.)
- For the frequency distribution you determine per character the number of times it appears within the combined JavaDoc and divide it by the total number of characters in the JavaDoc
- Provide a list of method calls (including the instance variable on with the method was called, `myname.toLowerCase()` for example) that your code finds, and more importantly describe which methods (due to the number of parameters or the usage of expressions as part of the method call) your code is not able to detect properly.

Provide code

There is some code that you can use for this assignment. It is not much but it gives you a head start. For the second part a Java file is provided for which the language in the JavaDoc and the exact number of method calls is known. This file is read automatically by the `LanguageDetectorTest` and can be found at `src/main/resources`.

Grading

In order for the assignment to pass you must meet all the following criteria.

Your report meets the requirements as described in the study-manual.
Implemented the reverse version of Boyer-Moore.
Correctly searches backwards.
Reports the number of comparisons correctly.
Clearly describes the changes made to the original implementation.
Clearly state the original source origin.
Implement the language detection.
Proof (graphically) that your approach works and clearly describe why it works.
Show an overview of the method calls your code is able to detect.
When a method is called on an variable(instance), the name of the variable appears in the overview list.
Clearly describe which type of method calls your code doesn't detect.
The number of method-calls not detected is small compared to the list of detected method-calls.
Your upload contains all the code/files needed to run the program.