Assignment overview

In this lab you will write a simple spell checker. Actually, you will write two spell checkers:

- One using brute force (sequential search)
- One using decrease-and-conquer (binary search)

For complete information, see the "Assignment details" section below.

Be sure to read the "Tips and FAQs" section, too.

This lab requires Java programming. You may (and should!) discuss the lab and coding techniques with your classmates, but all of the code you submit to Learning Hub must be your own.

Submission information

Due date: As shown on Learning Hub. Late assignments will not be graded.

What to submit:

- Java source file
- Screen print of your output
- Short report with discussion/conclusions

Please do not zip or compress your submissions.

Grading

This lab is graded on a 10-point scale. Point breakdown:

- Sequential search solution to the problem [4 points]
- Binary search solution to the problem [4 points]
- Summary report [2 points]

Assignment details

Input

Data files available on Learning Hub:

- lab4_wordlist.txt – contains a long, sorted list of words that are to be treated as the spell checker dictionary. You should read and store this data in an array of Strings.
- lab4_testdata.txt – contains the data that you are to test for correctly/incorrectly spelled words.

Methodology

Create a file SpellChecker.java with three primary methods: main(), seqSearch(), binSearch().

The seqSearch() method searches the dictionary for every word in the test data using Sequential Search. If a word is not in the dictionary, it is considered misspelled. Count the total number of words that are not found.

The binSearch() method searches the dictionary for every word in the test data using Binary Search. If a word is not in the dictionary, it is considered misspelled. Count the total number of words that are not found.

Ignore capitalization in all of this searching.

Note: It is OK (and you are strongly encouraged!) to write additional helper functions as needed for your program design.

Output

For each search method, output to the console the total number of misspelled words and the total processing time it required in seconds or milliseconds.

Make sure you time only the spell checking, not the I/O or file processing.

Discussions/conclusion

Write a short written report that includes:

- The overall efficiency class for each algorithm (such as O(n^2), O(nlogn), ...).
- Be very clear about what is the problem size, i.e. what the "n" represents.
- Which algorithm performed faster in your experiment.

Tips and FAQs

All code examples in this section are only suggestions. There are other ways that these bits and pieces can be accomplished in Java. You are not required to use any of the methods given here. They are intended only to help you if you are getting stuck on parts of the assignment where you are not meant to get stuck.

Comparing Strings

You can use `s1.equals(s2)` to compare Strings s1 and s2 for equality.

You can use `s1.compareToIgnoreCase(s2)` to compare s1 and s2 for less-than or greater-than. This method also returns 0 if the strings are equal.

Midpoint

The Math library has a floor() function:

```
Integer midpt = (int) Math.floor((startpt+endpt)/2.0);
```

How to read the data file

Here is one way to read a "word list" sort of data file (one word per line) into an array of Strings:

```
String dictfiletext = new
      String(Files.readAllBytes(Paths.get("lab4_wordlist.txt")));
String[] dict = dictfiletext.split("\n");
System.out.println("Dictionary file size: " + dict.length + " words");
```

The last line of code is not required, it's just there to verify our input. By the way, there are 235,922 words in this dictionary.

Yes, the dictionary is weird

Even though there are 235,922 words in the dictionary, there is a lot missing. For example, *misspelled* is not in this dictionary. I'm not making this up. This bothers me, but it doesn't bother me enough to try to find a better dictionary.