HW7: Programming with trees

Due Monday Oct. 27 1:59PM through turnin
Total: 20pts

You may work on this assignment with one other student. A team of two members must practice pair programming. Pair programming "is a practice in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test." Both members in a team must read the article All I really need to know about pair programming I learned in kindergarten. This article can be downloaded to any computer that is in the Marquette network.

If you work in a team of two students, make sure to write both names in the beginning of each java file as JavaDoc and always let the same user submit the files. The lower grade from two submissions will be used if both team members submit files.

Preparation

The textbook includes a program, namely `BinarySearchTree.java` that implements a binary search tree in pages 547-575. The `BinarySeachTree.java` uses class `BSTNode`, which is implemented on pages 546-547. These two .java files are included in the zip file posted on D2L. You are recommended to study and understand these programs before you attempt the assignment.

Java files you can reuse

You can reuse `BSTNode.java` from the textbook, use `BinarySearchTree.java` as a main reference.

Task

For this assignment, you are going to implement a simple SpellCheck that uses a binary search tree to implement a DictionaryADT and checks if the words in a given file are correctly spelled.

The classes that you will implement:

1. The `BSTDictionary` Class placed in `BSTDictionary.java`. This class implements interface `DictionaryADT`, which is provided in `DictionaryADT.java`. One can add, remove, and lookup a key in a dictionary ADT. One can also get the average number of comparisons of the tree. It is calculated as the sum of comparisons over all the words in the tree divided by the number of words.

2. The `SpellCheck` Class placed in `SpellCheck.java`. This class checks if the words in a given file is correctly spelled by comparing the words against keys in the dictionary. If a word is not correctly spelled, a spell error is printed to the console.

Test your design and implementation

You should try to exactly match the solution output for a given input.
A driver `SpellCheckRunner.java` is provided. This file includes a `main(String[] args)` that accepts two file names from the command line arguments. For example,

```
java SpellCheckRunner unsorteddictionary.in wordstocheck.in
```

This driver 1) uses all the words in the first file to build a binary search tree, which implements a dictionary, and prints the tree size and height, 2) deletes the tenth word in every 10 words from the input file and prints the resulting tree size and height, and 3) checks each word in the second file and reports incorrectly spelled words in order.

Input: A sample content of `unsorteddictionary.in` can be downloaded. There are thousands of words in this file, one word in each line. You can safely assume that the words only have English letters. The words are case insensitive, that is, "CAT" and "cat" are the same.

The words in file wordstocheck.in have the same format: each word in one line, and each word contains only English letters.

Output: As the height of the binary search tree, which is built as the dictionary out of words in `unsorteddictionary.in`, determines the running time of your program, your program prints the height of the tree. Your program also reports incorrectly spelled words in file `wordstocheck.in` based on the dictionary. A sample output is as follows:

```
Original tree
  # words: words
  Largest number of comparisons: max
  Average number of comparisons: average

Resulting tree after removing 10% of words
  # words: words
  Largest number of comparisons: max
  Average number of comparisons: average

Incorrectly spelled words:
condedded
sgove
…
```

Where *words* is the number of words in the dictionary, *max* is the largest number of comparisons any key search can possibly perform, and *average* is the *(total comparisons)/(#words)*. More examples are included in TA-bot.

Submission

Submit your `BSTDictionary.java` and `SpellCheck.java`, and other .java files you have programmed to turnin system. **Do not submit `SpellCheckRunner.java`, `DictionaryADT.java`, and `BSTNode.java`.** The provided java files will be used to compile and run your submission. If your submission requires different versions of these files, it won't pass compilation and tests.

You should use a single command to turnin all files and separate the files with space. For example:

```
turnin -c cosc2010-Ge -p SpellCheck SpellCheck.java BSTDictionary.java
```

You can submit multiple times but only the last submission will be saved by turnin.