Assessed Individual Coursework 1 — Spell-Check

Learning Outcomes

- *Ability to analyse and hence choose suitable algorithms and data structures for a given problem*
- *To design and implement medium sized programs based on a range of standard algorithms*
- *Understanding the distinction between Abstract Data Type (ADT) properties and concrete ADT realisations*
- *Appreciation of need for integration of multiple ADTs in substantial programs*
- *Appreciation of efficiencies/reassurances from ADT reuse*
- *To develop practical problem-solving skills in the context of programming*
- *To be able to critically analyse and hence choose suitable algorithms and data structures for a given problem*

1 Overview

In this assignment, you are requested to write a simple spell-checker program. Your program should be named `SpellCheck` and it will take as command line arguments two file names. The first name is the dictionary file which contains correctly spelled words (for example, the provided `dictionary.txt` file). The second file contains the text to be spell-checked (for example, the provided `text-to-check.txt` file). Your program should first read all words from the dictionary file and insert them into a set data structure. Then your program should read words from the second file and check if they are in the set. For words that do exist in the set nothing needs to be done. For words which are not in the set, your program should suggest possible correct spellings by printing to the standard output. You should perform modifications of a misspelled word given in Section 3. In this assignment you will implement and compare (experimentally) a linked list based and a hash table based set.

Indicative Timing Guidelines

Step 1: from Week 3 to Week 4

develop your linked list based set, implement and test some misspelled word modifications

Step 2: from Week 4 to Week 5

develop your hash table based set (you can first implement linear probing collision handling before later changing to a double hashing), implement and test the remaining misspelled word modifications

Step 3: from Week 5 to Week 6

finalise you double hashing collision handling, run tests, run comparison of list-based and hash-table based implementations, finalise your report

2 Implementation

You are to implement the program based on a set, let us call it W. You will use W for storing the words in the dictionary input file specified by the first command line argument. After storing all the words in W, you should open the second file (the one to be spell-checked) for reading and look up the words in the second file in set W. If any word w of the second file is not in W, you have to try all possible modifications of w suggested above, in Section 3. Notice that different modifications may result in the same word. The implementation of the set, see sections below, insures that each set element is a unique word. Therefore, to make sure that there are no duplicates on the list of possible spellings, create a second set S (for each misspelled word), and insert all modifications of the misspelled words that are in set W. After you went over all modifications, print out all the words stored in set S.

3 Modifications of Misspelled Word

You should perform the modifications of a misspelled word to handle commonly made mistakes:

- *Letter substitution*: go over all the characters in the misspelled word, and try to replace a character by any other character. In this case, if there are k characters in a word, the number of modifications to try is $26k$. For example, in a misspelled word 'lat', substituting 'c' instead of 'l' will produce a word 'cat', which is in the dictionary.
- *Letter omission*: try to omit (in turn, one by one) a single character in the misspelled word and see if the word with omitted character is in the dictionary. In this case, there are k modifications to try where k is the number of characters in the word. For example, if the misspelled word is 'catt', omitting the last character 't' will produce a word 'cat' which is in the dictionary.
- *Letter insertion*: try to insert a letter in the misspelled word. In this case, if the word is k characters long, there are $26 * (k + 1)$ modifications to try, since there are 26 characters to try to insert and $k + 1$ places (including the beginning and the end of the word) to insert a character. For example, for word 'plce', inserting letter 'a' in the middle will produce a correctly spelled word 'place'.
- *Letter reversal*: Try swapping every 2 adjacent characters. For a word of length k, there are $k - 1$ pairs to try to swap. For example in a misspelled word 'paernt', swapping letters 'e' and 'r' will produce a correctly spelled word 'parent'.

For each word which was misspelled, on a separate line, print out the misspelled word and all possible correct spellings that you found in the dictionary. For example, if your dictionary file contains the words 'cats like on of to play', and the file to spell check contains 4 words: 'Catts lik o play', the output should be:

```
catts => cats
lik => like
o => on, to, of
```

Notice that the list of possible correct spelling must contain only unique words. In the example above, for the misspelled word 'catts', removing the first 't' or the second 't' leads to the same word 'cats', but this word appears in the output only once. For the modifications of a word above, the Java class `StringBuffer` and its built-in methods are very useful.
A file `FileWordRead.java` which will read the next word from an opened file is provided. See comments in the file `FileWordRead.java` for the usage. In this implementation, all words are converted to lower case so that the words 'Cat' and 'cat' are treated as one word. Thus all the

words you read from the file using the program will be lowercase words. The method `suggestions` declared by the `Spelling` interface and to implement in the `SpellingImpl` class should produce the list of suggestions from a given word and a given dictionary.

4 Classes and Interfaces

You should fork the `F28DA-20-21-CW1` project on GitLab Student and regularly commit and push your work (remember to add new files). You should **not** invite any other students to your project nor share your code with other students (see Section 7). Remember that the final submission is through **Vision** (see Section 8).

`SpellCheck` **(class to complete)**

This is the class which contains the main program. You have to write most of the main program, the code in `SpellCheck.java` currently reads the command line arguments (file names). The name of the class must stay the same, `SpellCheck`. In the version that you hand in, you should be using the hash table set implementation. The linked list based implementation should be used only for comparison with the hash table implementation (see Section 5).

`Spelling` **(interface provided)**

Interface for spelling suggestions.

`SpellingImpl` **(class to complete)**

This class implements the spelling suggestions (see Section 3). The class must implement the `suggestions` method of the provided `Spelling` interface, all other methods and any member variables must be private.

`public WordsSet suggestions(String word, WordsSet dict)`

Suggests word modifications for a given word and a given word dictionary.

`WordsSet` **(interface provided)**

The interface of your set of words should implement (both the linked list based set and the hash table based set).

`SpellCheckException` **(class provided)**

This exception should be thrown by your set in case of unexpected conditions, see classes `HTWordsSet` and `LLWordsSet` for cases in which to throw this exception.

`LLWordsSet` **(class to implement)**

This class implements a set of words based on linked list. You can use Java's build in *LinkedList* class (importing `java.util.LinkedList`). This class must implement the methods of the provided `WordsSet` interface. You must implement the following public methods, and all other methods which you might implement for this class must be private. Also any member variables must be private.

`public LLWordsSet()`

Constructor for the class.

`public void insWord(String word) throws SpellCheckException`

Adds an word to the set. Throws an `SpellCheckException` exception if the word is already present.

`public void rmWord(String word) throws SpellCheckException`

Deletes a word from the set. Throws exception if no such word exists.

`public boolean wordExists(String word)`

Returns true if the word is present.

`public int getWordsCount()`

Returns the number of words stored in the set.

`public Iterator<String> getWordsIterator()`

Returns an *Iterator* over all words stored in the set. The iteration is over objects of class *String*. You can use `java.util.Iterator` which gives the *Iterator* interface (to use this interface, import `java.util.Iterator` at the beginning of your `LLWordsSet.java` file).

`Monitor` **(interface provided)**

This is an interface your hash table based set should implement.

`Hashing` **(interface provided)**

This is an interface your hash table based set should implement.

`HTWordsSet` **(class to implement)**

This class implements a words set based on hash table, and should implement the provided `WordsSet` and `Hashing` interfaces. It should also implement the `Monitor` interface (needed by `HTWordsSetProvidedExp`). You should use open addressing with double hashing strategy. Start with an initial hash table of size 7. Increase its size to the next prime number at least twice larger than the current array size (which is N) when the load factor gets larger than the maximum allowed load factor (maximum allowed load factor is to be given to the constructor to the hash table). You must design your hash function so that it produces few collisions.

You should implement the following constructors.

`public HTWordsSet()`

A constructor for the class which sets the maximum load factor to `0.5`.

`public HTWordsSet(float maxLF)`

A constructor for the class which allows to set the maximum load factor at construction time.

You must implement the following public methods, and all other methods which you might implement for this class must be private. Any member variables must also be private.

`public void insWord(String word) throws SpellCheckException`

Adds an word to the set. Throws an `SpellCheckException` exception if the word is already present.

`public void rmWord(String word) throws SpellCheckException`

Deletes a word from the set. Throws exception if no such word exists.

`public boolean wordExists(String word)`

Returns true if the word is present.

`public int getWordsCount()`

Returns the number of words stored in the set.

`public Iterator<String> getWordsIterator()`

Returns an *Iterator* over all words stored in the set. The iteration is over objects of class *String*. You can use `java.util.Iterator` which gives the *Iterator* interface (to use this interface, import `java.util.Iterator` at the beginning of your `HTWordsSet.java` file).

`public int giveHashCode(String s)`

This method (declared by the `Hashing` interface) should be used to get a hash code for a string. You have to use the polynomial accumulation hash code for strings we talked about in class. You should not use Java's `hashCode` method.

`public float getMaxLoadFactor()`

This method (declared by the `Monitor` interface) returns the maximum authorised load factor.

`public float getLoadFactor()`

This method (declared by the `Monitor` interface) returns the current load factor.

`public float getAverageProbes()`

This method (declared by the `Monitor` interface) returns an average number of probes performed by your hash table so far. You should count the total number of operations performed by the hash table (each of find, insert, remove count as one operation, do not count any other operations) and also the total number of probes performed so far by the hash table. When `getAverageProbes()` is called, it should return `(float) numberOfProbes/numberOfOperations`. As you decrease the maximum allowed load factor, the average number of probes should go down. When you run the `HTWordsSetProvidedExp` program, it will run your hash table at different load factors and will print out the average probe numbers versus the running time. If you see that the average probe number goes up as the max load factor goes up, you are probably computing probes/implementing hash table correctly. You can implement any other methods that you want, but they must be declared as private methods.

`HTWordsSetProvidedExp` **(class provided)**

The main method of this class makes some experimental run, see above.

`HTWordsSetProvidedTest` **(test class provided)**

This is a JUnit 4 test class which we will use to test your hash table implementation. Compile and run it once you have implemented your `HTWordsSet` class. It will run some tests on your hash table and will let you know which tests are passed/failed by your hash table. Read the source code of this class to understand what each test if doing and to fix your implementation in case of failed tests. To get the full score on the assignment, you must pass all the tests.

`HTWordsSetTest` **(test class to implement)**

Write you own JUnit 4 test cases for your hash table implementation in this test class.

`ModificationsProvidedTest` **(test class provided)**

This is a JUnit 4 test class which runs a single test on your implementation of the word modifications suggestions (see Section 3).

`ModificationsTest` **(test class implement)**

Write you own JUnit 4 test cases for your word modifications/suggestions implementation in this test class.

5 Hash Table vs. Linked List Set Implementation

Dictionary files of different sizes (`d1.txt`, ..., `d6.txt`) are provided. Run your program with these different dictionaries and the same `text-to-check.txt` file to check the spelling of words. That is the second command line argument stays the same, while the first one goes through `d1.txt`, ..., `d6.txt`. Get the running time using the Java method:
`System.currentTimeMillis()`. Note that this method will return the current time, **NOT** the running time from the start of the program. Therefore, to get the total time (in milliseconds) your program took to complete, you should measure the current time at the very start of the program, then at the very end, and subtract the two. Since what changes between the different runs is the size of the dictionary file, we should plot the running time vs. the size of the dictionary file, that is the number of words in the dictionary file. Count the number of words in each dictionary file and plot, on the same chart, the number of words versus the running time for the list and hash based dictionary implementations.
The running time is essentially the time it takes to insert all the dictionary words, since the second file for spell checking has only 2 words to check. When we insert a word into a set, we also have to check if that word is already in the set.
For a hash table, checking and inserting is expected to take a constant amount of time, and therefore inserting all elements in the set should take a linear time. For a linked list, inserting is constant amount of time, but checking if the element is already in the list is linear amount of time, and therefore inserting all elements in the set should take quadratic time. Thus hash table based implementation running time plot should resemble a linear function, linked list based implementation should resemble a quadratic function.

6 Coding Style

Your mark will be based partly on your coding style. Here are some recommendations:

- Variable and method names should be chosen to reflect their purpose in the program.
- Comments, indenting, and whitespaces should be used to improve readability.
- No variable declarations should appear outside methods (“instance variables”) unless they contain data which is to be maintained in the object from call to call. In other words, variables which are needed only inside methods, whose value does not have to be remembered until the next method call, should be declared inside those methods.

- All variables declared outside methods (“instance variables”) should be declared `private` (not `protected`) to maximise information hiding. Any access to the variables should be done with accessor methods (like `getVar()` and `setVar(...)` for a private variable `var`).

- Use appropriate stream when printing output: normal output should be on the standard output (using `System.out`). Error or warning notifications should be on the standard error output (using `System.err`).

7 Note on plagiarism and collusion

- The coursework is an **individual** coursework.

- You are permitted to **discuss** the coursework with your with your classmates. You can **get help** from lecturer and lab helpers in lab sessions. You can get help and **ask questions** to lecturer, via GitLab-Student or by email or at the beginning or end of lecture sessions, or during the office hour of the lecturer.

- Coursework reports must be written in your **own words** and any code in their coursework must be your **own code**. If some text or code in the coursework has been taken from other sources, these sources must be **properly referenced**. Failure to reference work that has been obtained from other sources or to copy the words and/or code of another student is **plagiarism** and if detected, this will be reported to the School’s Discipline Committee. If a student is found guilty of plagiarism, the penalty could involve voiding the course.

- Students must **never** give hard or soft copies of their coursework reports or code to another student. Students must always **refuse** any request from another student for a copy of their report and/or code.

- Sharing a coursework report and/or code with another student is **collusion**, and if detected, this will be reported to the School’s Discipline Committee. If found guilty of collusion, the penalty could involve voiding the course.

- **Special note for re-using available code**: If you are re-using code that you have not yourself written, then this must clearly be indicated. At all time, you have to make clear what part is not yours, and what in fact is your contribution. Re-using existing code and amending it is perfectly fine, as long as you are not trying to pass it on as your own work, so you must clearly state where it is taken from. A brief additional explanation of why you chose this code would be an added benefit and even adds value to your work. If your code is found elsewhere by the person marking your work, and you have not mentioned this, you may find yourself having to go before a disciplinary committee and face grave consequences.

8 Submission

Submit on Vision the followings:

Sources Submit an archive of your source code from GitLab Student. To do so, 1) make sure you have added/committed/pushed all your Java files to GitLab Student, 2) download an archive of your sources (see image below), submit the archive on Vision. The submission should include all necessary Java files, it should **not** include compiled files (**no** `.jar`, **no** `.class` files), and you should not modify the interfaces provided or the test files called `Provided`. If you consider you need to add or modify some method signatures in the interfaces, please speak first to the lecturer of the course.

Report A short report (not more than 4 pages). Your report should:

1. Indicate your name, campus and programme in the first page,
2. Explain briefly your design choices, if your program meets the specification fully, if your program has known limitations,
3. Provide and discuss the chart comparison between Linked List and Hash Table implementations.

Coursework submission deadline: Week 7, Tuesday 23rd of February, 2021, 3:30pm.
The course applies the University's coursework policy.

- No individual extension for coursework submissions.
- Deduction of 30% from the mark awarded for up to 5 working days late submission.
- Submission more than 5 working days late will not get a mark.
- If you have mitigating circumstances for an extension, talk to your Personal Tutor and submit a Mitigating Circumstances (MC) form with supporting documentation online[1].

You will be required to take part in **peer-testing** after submission. You will be using your classes and the unit test cases you have prepared. At the end of the peer-testing period you will be asked to submit a short **reflective summary** on Vision. In some circumstances, a **demonstration** of your program could be organised, this needs to be approved by the lecturer of the course.

9 Marking Scheme

Your **overall mark** will be computed as follows.

• Coding style, `SpellCheck`, `SpellingImpl` and `LLWordsSet` implementations	20 marks
• `HTWordsSet` implementation	20 marks
• `HTWordsSetProvidedTest` pass	20 marks
• Report and comparison chart of Linked List vs. Hash Table running times	20 marks
• Test cases submitted, peer-testing involvement and reflective summary[2]	20 marks

[1] https://www.hw.ac.uk/uk/students/studies/examinations/mitigating-circumstances.htm

[2] or demonstration of your program if circumstances require