Project Report of CS590 Challenge 5

Table of Contents

1. Report Revision History

Name	Date	Reason for changes	Version
Xiaoping Yu	2019.06.05	Initial draft	1.0
Yunxiu Zhang	2019.07.02	More details and enhancements	1.5
Tianye Zhao, Xiaoping Yu	2019.07.25	Style, editing, additional information after research	2.0

2. Team members and Contribution

Name	Contribution Description
Xiaoping Yu	1.Research and technology selection. 2.Coding(transposition and insertion scenarios of spellCheck method) 3. Testing, code review 4. Write report.
Yunxiu Zhang	1.Research and algorithms selection. 2. Coding(spellCheck method and integration) 3. Testing, code review 4. Write report.
Tianye Zhao	1.Research and write proof of concept code. 2. Coding(buildDict method) 3. Testing, code review 4. Write report.

3. Introduction

The objective of this challenge is to implement a basic spell checker, which recommend the likeliest known word from a dictionary built from a corpus file, for the input mistyped words.

The term basic means this spell checker only handles the most common mistakes which account for 60%-70% of all spelling mistakes people might make while typing. These most common spelling mistakes occur for the following reasons: Deletion, Replacement, Transposition, Insertion of only one character(2 consecutive characters for transposition).

4. Comparison of possible solutions

There are a few possible solutions to implement this basic spell checker.

4.1 Solution A: Brute Force

It is not difficult to come up a naïve brute force solution. For each incoming word, we compare it with each word in the dictionary and compute their edit distance. After computing, we get a list of candidates for an incoming word, and pick one word from the candidate as a recommendation as per the requirement. The time complexity of this solution is O(n*m) (n is the number of words in dictionary and m is the length of the incoming word) . This solution is also illustrated in Figure 1.

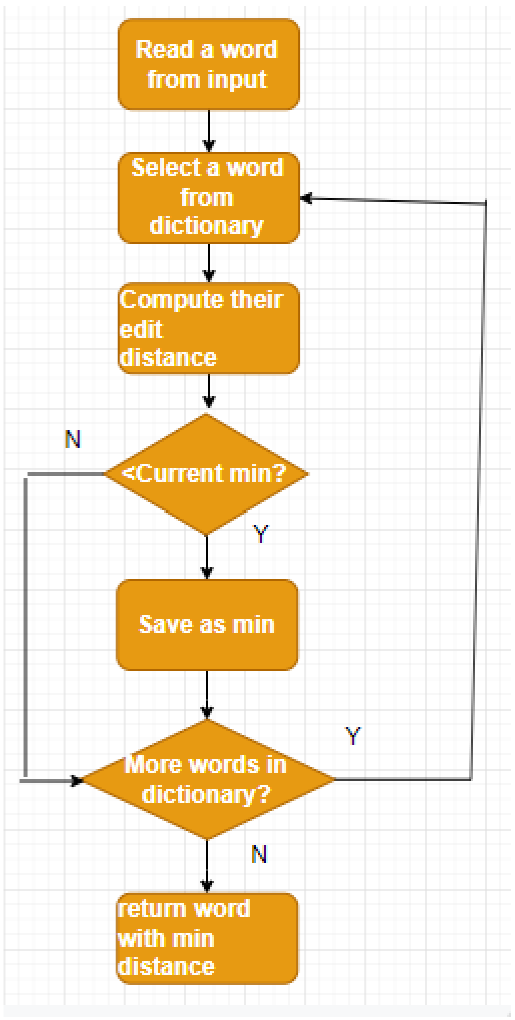

Read a word from input
Select a word from dictionary
Compute their edit distance
<Current min?
N
Y
Save as min
More words in dictionary?
Y
N
return word with min distance

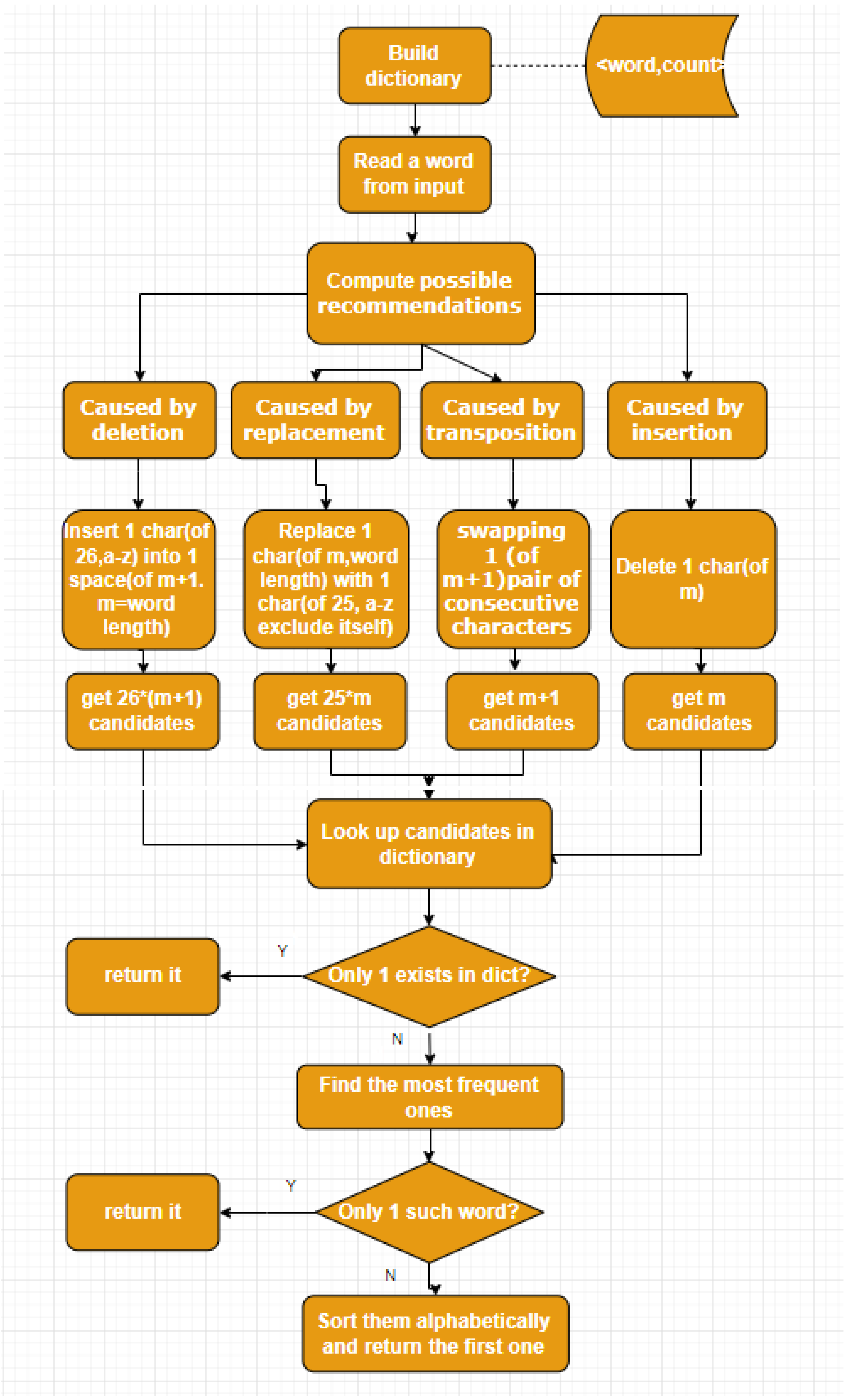
Figure 1. Flowchart of Brute Force solution

4.2 Solution B: Compute all possible recommendations and look up

Because the possible mistakes that this spell checker would handle are countable, and the time complexity of computing their possible original words is O(m) (m is the length of the incoming word). And for each possible original word, we look up it in the dictionary, which is a HashMap, to see if it exists. If yes, we put it into the candidate list. The time complexity of look-up is O(1), so the overall time complexity of this solution is O(m). Since the length of a word is very short, O(m) is almost O(1), which is much better than above solution. This solution is also illustrated in Figure 2.

5. Solution description

As per above comparison, solution B is much better than solution A, so in this section let's talk about the implementation of solution B.



Build dictionary
<word,count>
Read a word from input
Compute possible recommendations
Caused by deletion
Caused by replacement
Caused by transposition
Caused by insertion
Insert 1 char(of 26,a-z) into 1 space(of m+1. m=word length)
Replace 1 char(of m,word length) with 1 char(of 25, a-z exclude itself)
swapping 1 (of m+1)pair of consecutive characters
Delete 1 char(of m)
get 26*(m+1) candidates
get 25*m candidates
get m+1 candidates
get m candidates
Look up candidates in dictionary
Only 1 exists in dict?
Y
return it
N
Find the most frequent ones
Only 1 such word?
Y
return it
N
Sort them alphabetically and return the first one

Figure 2. Flowchart of selected solution B

5.1 Dictionary building

The dictionary can be built as a HashMap, in which the key is a word and the value is the frequency it occurs in corpus text file.

To read all the words from the text file, we need to remove some noises, including space, symbols, and so on. Only the strings with alphabetic characters and apostrophes are valid words. Even for a word that is connected with a hyphen, we split it into two words. We use a regular expression to remove the noises. After that, we can simply read the valid words one by one, count their frequency and save the data into the dictionary.

5.2 Possible recommendations computation

As per the requirement, the most common spelling mistakes occur for 4 reasons: Deletion, Replacement, Transposition, and Insertion. If the word exists in dictionary, then it is not a mistype, we return it as it is. Otherwise, we compute its possible original words according to the possible ways it was mistyped.

5.2.1 Deletion

One character in the string gets deleted incorrectly. If the mistype is caused by deletion, then we can get the possible recommendation by insert one character into the word. If the incoming word is of length m, then there are m+1 places to insert a character, and for each place we have 26 characters to choose from. This is because only the letters a-z are involved, as per the requirement.

5.2.2 Replacement

If the mistype is because one character in the string is incorrectly replaced by another one, we can get the possible original words by replacing each one character of the word with another 25 characters (a-z excludes itself).

5.2.3 Transposition

If the mistype is caused by swapping one pair of consecutive characters, we can get the possible original words by swapping each pair of consecutive characters again in the incoming word. If the incoming word is of length m, then we can get m-1 pairs of consecutive characters with first m-1 characters and their subsequent character.

5.2.4 Insertion

If the mistype is caused by inserting one extra character somewhere in the string, then each character in the word is possibly the extra one. we can get the possible original words by deleting one of the characters in the string. If the incoming word is of length m, then there are m possible original words.

5.3 Candidates selection

If we got multiple candidates for one incoming word after above step of possible recommendations computation, we need to select one as a result.

First of all, we check their frequency in the dictionary (the key is the word, and the value is the frequency), and return the one occurred most frequently.

Then if we found multiple candidates with same frequency, we sort it alphabetically and return the first one in the list.

In our code, we combine above two steps into one, where we sort the candidates by their frequency and alphabetical order.

6. Technical implementation

Our solution is implemented in Java (JDK8) without external framework, library or tool. Source code will be provided separately.

Steps to run the program

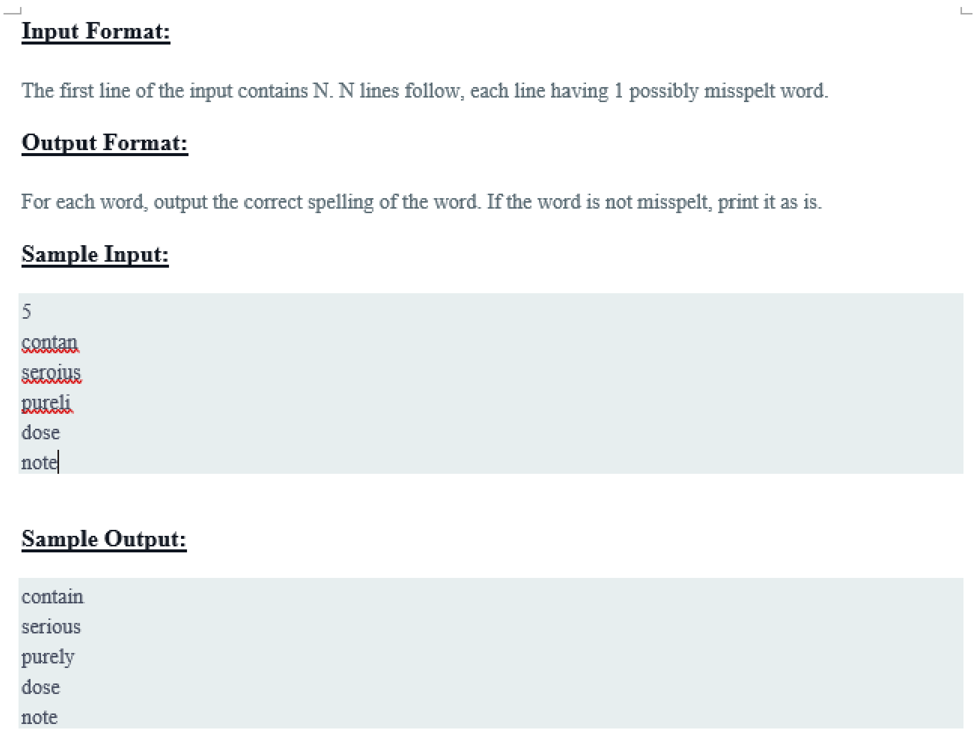
(1) Install JDK 8 if not yet

(4) Run SpellChecker.java as Java application in IDE(e.g. Eclipse) Or Run *mvn exec:java* on DOS command line under project folder.

(5) Provide inputs and press Enter

(6) Check out the result correct spelling of the words which are printed immediately after the input.

Because the challenge document has made it very clear what expect input and output should be(Figure 3) and we guess this application might be used for some automatic assessment, we did not implement any user interface or extra hint message(See Figure 4 and Figure 5).



Input Format:

The first line of the input contains N. N lines follow, each line having 1 possibly misspelt word.

Output Format:

For each word, output the correct spelling of the word. If the word is not misspelt, print it as is.

Sample Input:

```
5
contan
seroius
pureli
dose
note
```

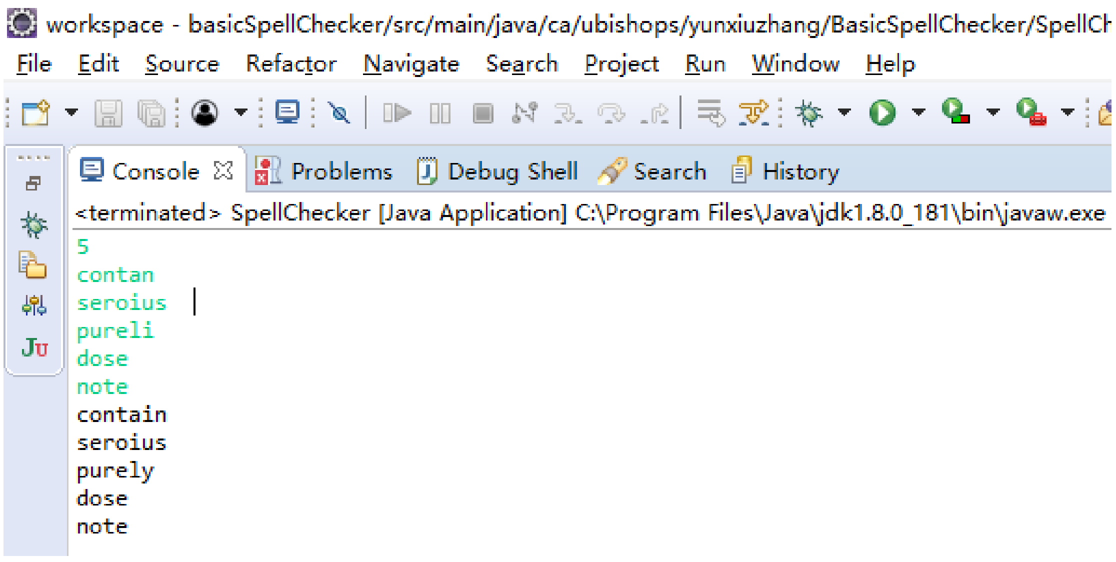
Sample Output:

```
contain
serious
purely
dose
note
```

Figure 3. Expected input and output (from Challenge doc)

7. Results

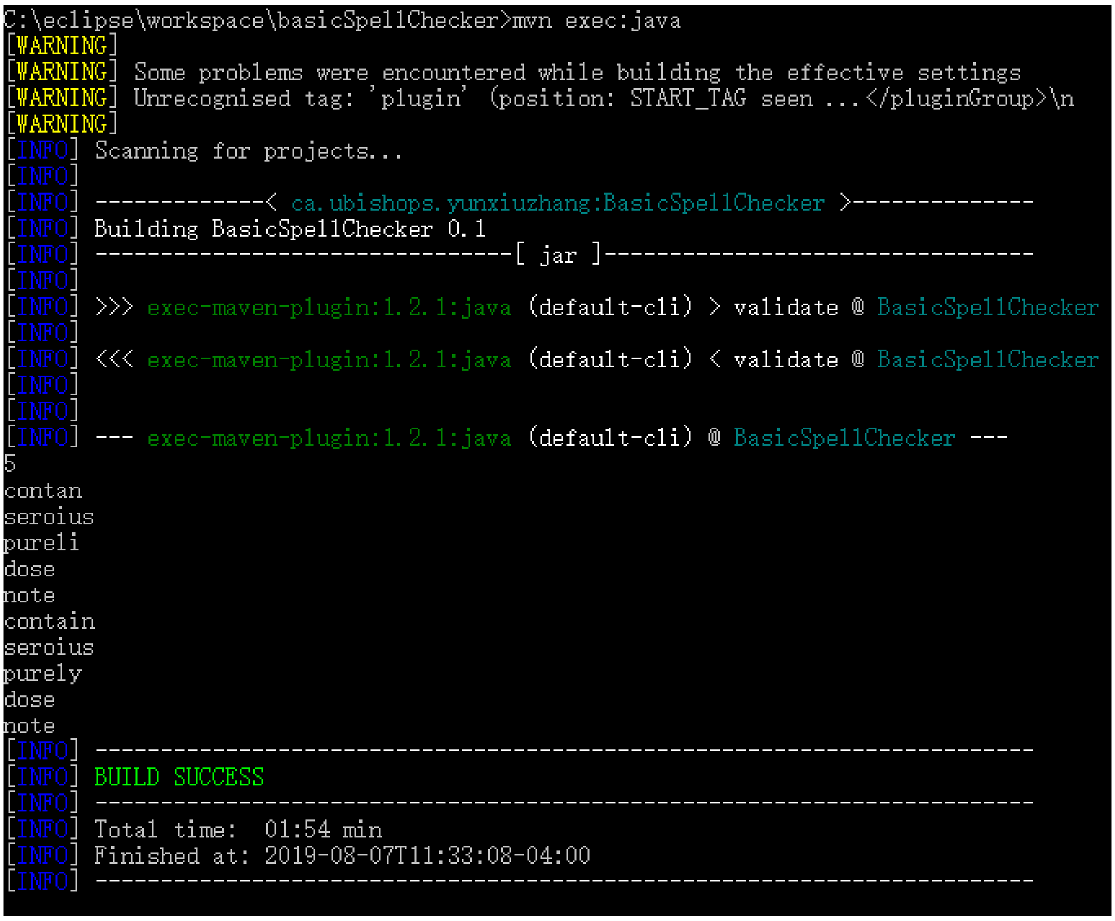
The program is a standalone Java application. As below are the screenshots when it's run on IDE and command line.



workspace - basicSpellChecker/src/main/java/ca/ubishops/yunxiuzhang/BasicSpellChecker/SpellCh
File Edit Source Refactor Navigate Search Project Run Window Help
Console
Problems
Debug Shell
Search
History

```
<terminated> SpellChecker [Java Application] C:\Program Files\Java\jdk1.8.0_181\bin\javaw.exe
5
contan
seroius
pureli
dose
note
contain
seroius
purely
dose
note
```

Figure 4. Running on IDE



```
C:\eclipse\workspace\basicSpellChecker>mvn exec:java
[WARNING]
[WARNING] Some problems were encountered while building the effective settings
[WARNING] Unrecognised tag: 'plugin' (position: START_TAG seen ...</pluginGroup>\n
[WARNING]
[INFO] Scanning for projects...
[INFO]
[INFO] -------------< ca.ubishops.yunxiuzhang:BasicSpellChecker >--------------
[INFO] Building BasicSpellChecker 0.1
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> exec-maven-plugin:1.2.1:java (default-cli) > validate @ BasicSpellChecker
[INFO]
[INFO] <<< exec-maven-plugin:1.2.1:java (default-cli) < validate @ BasicSpellChecker
[INFO]
[INFO]
[INFO] --- exec-maven-plugin:1.2.1:java (default-cli) @ BasicSpellChecker ---
5
contan
seroius
pureli
dose
note
contain
seroius
purely
dose
note
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:54 min
[INFO] Finished at: 2019-08-07T11:33:08-04:00
[INFO] ------------------------------------------------------------------------
```

Figure 5. Running on command line

8. Conclusion

To tackle this challenge, we have implemented the basic spell checker by computing all possible recommendations according to the 4 major reasons of mistakes in O(m) time(m is the length of a incoming word): Deletion, Replacement, Transposition, and Insertion, and looking up each of the recommendation words in the dictionary HashMap in O(1) time. The overall time complexity of this solution is O(m). The number m is very small, so the time complexity of this solution is almost O(1).