CSE 1322L – Assignment 2 (Spring 2026)

Introduction

When performing a spellcheck, your computer checks if the word you typed can be found in a dictionary. If the word isn't in the dictionary, the computer will suggest a few words from the dictionary which it believes could be the word that you tried to type. There are several algorithms to determine this and, in this assignment, you will write a program which implements one such algorithm by Wagner and Fisher, described in a 1974 paper called "The String-to-String Correction Problem". This will be done by calculating the edit distance between words that the user types against words in a dictionary, suggesting the 5 words whose edit distance is the smallest.

The String-to-String Correction Problem

In order to convert a string to another string, we allow ourselves 3 operations: insertion of a character, deletion of a character, and substitution of a character. For example:

- loop → sloop: insert the s
- loop → loo: delete the p
- loop → loot: replace the p with a t

The examples above all have an edit distance of 1, as only one operation needs to be performed to convert the string on the left with the string on the right. We can, however, perform as many operations as we want:

- loot → sloop: replace the t with a p, insert the s. **Edit distance of 2**
- flowers → bee: delete the s, delete the r, replace the w with an e, replace the o with a b, delete the l, delete the f. **Edit distance of 6**

While there are usually several combinations of the 3 operations which allow us to convert a string into another string, we are only interested in the ones that minimize the number of operations used. To do so, the Wagner-Fisher algorithm first creates a table whose length is the length of one of the strings plus 1, and its height is the length of the other string plus 1. The white part of the table below is the table between the words "sloop" and "loot". The grey rows have been added for clarity:

		S	L	O	O	P
L						
O						
O						
T						

The table's first roll is then filled with the numbers from 0 to X, where X is the length of the first string plus 1. The table's first column is filled the same way, but relative to the other word:

		S	L	O	O	P
	0	1	2	3	4	5
L	1					
O	2					
O	3					
T	4					

Starting on the first white empty box on the top left, we compare the letters in that row and column. If the letters are different, we pick the smallest of the following boxes, relative to the box we are currently working on:

- The box to the left
- The box above
- The box above and to the left

In our example, between 0, 1, and 1, we pick 0 as it is the smallest. We then add 1 to it and then use this result to fill in the box we are working on:

		S	L	O	O	P
	0	1	2	3	4	5
L	1	**1**				
O	2					
O	3					
T	4					

If, however, the two letters we are comparing are the same, then we fill in the box we are working on with the value of the box on the top left:

		S	L	O	O	P
	0	1	2	3	4	5
L	1	1	**1**			
O	2					
O	3					
T	4					

We continue filling the row in this manner, left to right. Once we are done with a row, we go on to the next row. Once the table is completely filled, the value at the bottom right of the table is the smallest edit distance between the two words:

		S	L	O	O	P
	0	1	2	3	4	5
L	1	1	1	2	3	4
O	2	2	2	1	2	3
O	3	3	3	2	1	2
T	4	4	4	3	2	**2**

The reason why this works is as follows: this table is keeping track of how many operations we need to convert one sub-string to another. Any given box represents the edit distance between the substring of the word on the top up to and including the letter on that column, and the word on the left, up to and including the letter on that row.

If we check the table, we can see that converting SLO to LO only requires a single operation (deleting the S). We determined this because, between the three numbers available to that box (1 on the top left, 2 above, and 2 to the left), the smallest one was the one on the top left. That box, in turn, was determining how to convert L to SL which, itself, only requires one deletion.

		S	**L**	**O**	O	P
	0	1	2	3	4	5
L	1	1	1	2	3	4
O	2	2	2	**1**	2	3
O	3	3	3	2	1	2
T	4	4	4	3	2	2

Lastly, the reason why the box on the top left has a 0 is because that represents converting an empty string into the empty string, which requires 0 operations. In our example, converting the empty string into an S requires 1 operation (adding the S). Converting the empty string into SL requires 2 operations, and so on.

You can find a visual demonstration of the algorithm in the link below, starting at 7:03:

https://youtu.be/Cu7Tl7FGigQ?t=423

This algorithm can be implemented using a 2D array of integers to represent the table, which is how you will implement it in your program.

Requirements

Download the zip file that is near these instructions and place both of its contents, Dictionary.java and dictionary.txt, into your project's "src" folder. Together, these two files will provide your program with a dictionary which you can use to spellcheck the user's input.

Then, add the following to your project:

A Word class, which will be used to keep track of the edit distance between 2 strings. This class has 3 fields, all of which are public:

- 2 strings called "originalWord" and "candidateWord", respectively
- An integer called "editDistance"

Your driver class (the class which contains main()) must have the following methods:

- **static int calculateEditDistance(String, String)**: This method creates a 2D array of integers. The length of the 2 dimensions matches the lengths arguments + 1. As an example, if the two arguments are "flower" and "bee", the array will have lengths 7 (length_of_flower + 1) and 4 (length_of_bee + 1), respectively. The method then fills the array according to the Wagner-Fisher algorithm described above. Once the array is filled, the value stored in the "bottom-right cell" is returned.
- **static void updateCandidates(ArrayList<Word>, Word):** The arraylist argument contains a list of 5 candidate words with the smallest edit distance from a given word, in ascending order. The given word is stored in the Word objects' "originalWord" field, whereas the candidates are stored in the "candidateWord" fields.

  Insert the Word object argument into the arraylist at the correct spot. The correct spot is determined by finding the first spot, from left to right, where the edit distance of the Word in the argument is less than or equal to the edit distance of the Word in that spot. If done correctly, this ensures the arraylist always has its Word objects sorted from smallest edit distance to largest. For example: suppose we have an arraylist with four Words, with edit lengths 2, 2, 4, and 5, respectively. If we try to insert a Word into this arraylist whose editDistance is 4, we must insert it at index 2, between the second 2 and the 4. The arraylist would then have the following edit lengths: 2, 2, 4, 4, 5.

  Note that the arraylist must have its size capped at 5, as we are only interested in the 5 best edit distances in the whole dictionary. As such, before returning, check the size of the arraylist: if it has 6 elements, remove the rightmost one.

- **static void spellCheckSentence(ArrayList<String>, String, Scanner):** This method will perform the spellchecking by comparing each word in the sentence (the second argument) with every word in the dictionary (the first argument), offering suggestions as it goes along. To achieve this, do the following:
    - Convert the sentence to lower-case
    - Use the sentence's split() using a space (" ") as the argument to convert the sentence into an array of strings, removing all the spaces
    - For each string in the array above, do the following:
        - Create an arraylist of Words called "candidates"
        - If the dictionary contains the string, go to the next string in the array
        - Otherwise, for each string in the dictionary, create a Word object using the string currently being evaluated in the array, the string currently being evaluated in the dictionary, and their edit distance using calculateEditDistance().
        - Pass the Word object and "candidates" as arguments to updateCandidates()
        - Once the dictionary has been fully traversed, show the user the candidateWord fields of all the Words in "candidates"
        - Ask the user if they want to replace the string being evaluated in the array with one of the words in "candidates" or if they want to add the word in the array to the dictionary
    - Finally, print all the words in the array on the same line, separated by spaces. If all was done correctly, the array should now only contain words that are in the dictionary, i.e.: the sentence has been spell-checked.
- **static void main():** This method does the following:
    - Create a Scanner, which will be used to read user input
    - Create an arraylist of Strings using the public method in the provided Dictionary class. This arraylist contains your dictionary.
    - In a loop, do the following:
        - ask the user to enter a sentence
        - If the sentence is empty, terminate the program
        - Otherwise, pass the sentence, the arraylist above, and the Scanner to spellCheckSentence()

Deliverables

- Assignment2.java (driver)
- Word.java

Considerations

- You will get partial credit for partial work, as long as the rubric permits it.
- Despite what deliverables say, all of your classes can be submitted in a single file.
- You will have to make use of String and ArrayList methods to complete this assignment. You can find their documentation in the links below:
    - https://docs.oracle.com/en/java/javase/23/docs/api/java.base/java/lang/String.html
    - https://docs.oracle.com/en/java/javase/23/docs/api/java.base/java/util/ArrayList.html
- There is no need to submit Dictionary.java or dictionary.txt. Your grader already has both files.
- Unless otherwise stated, class fields should be declared as private whereas class methods should be declared as public.
    - For this assignment, the fields in Word have been set to public to reduce the amount of typing required
- Given that the fields of the Word objects aren't edited after the object is created, it would be more appropriate to declare Word as a record instead of a class. You can read more about records in the link below:
    - https://docs.oracle.com/en/java/javase/17/language/records.html
    - **Do not declare Word as a record**. You will lose points if you do so. Word must be kept as a class
- Normally, you would remove the punctuation from a text before performing a spellcheck. For this assignment, we'll pretend the user never uses punctuation in their inputs. Otherwise, our dictionary will end up with lots of words containing punctuation, as the program would, for example, see "say" and "say," as two different words.
- The dictionary words were retrieved from the repository in the link below:
    - https://github.com/first20hours/google-10000-english
- The original Wagner-Fisher algorithm can be found in the link below:
    - https://dl.acm.org/doi/10.1145/321796.321811

Sample Output (user input in red)

```
[Spell Checker]

Enter a sentence to spell-check, or nothing to quit: The quick brown
fox leaps over the lazy dog
You've entered 'The quick brown fox leaps over the lazy dog'

'leaps' not in dictionary. Pick an option:
0. Replace with 'leads'
1. Replace with 'laos'
2. Replace with 'seas'
3. Replace with 'sleeps'
4. Replace with 'loops'
5. Add 'leaps' to dictionary
5
'leaps' added to dictionary

The final sentence is: 'the quick brown fox leaps over the lazy dog'

Enter a sentence to spell-check, or nothing to quit: A few small steps
for a man several giant leaps for humanity
You've entered 'A few small steps for a man several giant leaps for
humanity'

The final sentence is: 'a few small steps for a man several giant
leaps for humanity'

Enter a sentence to spell-check, or nothing to quit: Beter a birb in
the hand than two in the busch
You've entered 'Beter a birb in the hand than two in the busch'

'beter' not in dictionary. Pick an option:
0. Replace with 'beer'
1. Replace with 'meter'
2. Replace with 'peter'
3. Replace with 'better'
4. Replace with 'bother'
5. Add 'beter' to dictionary
3
Replaced 'beter' with 'better'

'birb' not in dictionary. Pick an option:
```

```
0. Replace with 'bird'
1. Replace with 'kirk'
2. Replace with 'ira'
3. Replace with 'cir'
4. Replace with 'rb'
5. Add 'birb' to dictionary
0
Replaced 'birb' with 'bird'

'busch' not in dictionary. Pick an option:
0. Replace with 'bunch'
1. Replace with 'bush'
2. Replace with 'buses'
3. Replace with 'buck'
4. Replace with 'punch'
5. Add 'busch' to dictionary
1
Replaced 'busch' with 'bush'

The final sentence is: 'better a bird in the hand than two in the
bush'

Enter a sentence to spell-check, or nothing to quit:
You've entered ''


Shutting off...
```