BasicSpellChecker

By :

Naveenraj Devaraj (ndevaraj17@ubishops.ca)

A report submitted to the **Dr Layachi Bentabet** of Department of Computer Science in conformity with the requirements for the degree of Master of Science.

Bishop's University
QC, Canada
August 2018

Abstract

The aim of this project is to identify the basic typo perform by the users in their regular documentations. We achieved the requirement in the python platform without using any dependent API's only by performing the manual check for each and every letter with the given corpus document. We mainly focused on the four type of scenario in this project and we optimized the project to our best to achieve the best performance in the implantation.

I. Introduction

In the python platform we implemented this project by initializing the file reader to read the corpus dictionary and it acts as source file for the word of dictionary. Once the source file of dictionary is recognized we open it and read it line by line then we separate the words from the line using the blank space and also we implemented some regex pattern to remove the unwanted special character's which are part of the word. Once the words are separated we added that in the list later we use the counter from the collection package to convert it into a key value pair of map, where the words act as a key because of its uniqueness and the occurrence of the word will act as a value pair of the map.

II. Implementation

In this project we mainly focused on the four different scenario's of mistakes which usually followed by the end user's during their documentation. All the four scenarios are handled without using any external libraries only by implementing our own idea towards achieving the best result for the given input.

Scenario 1: One character in the string gets deleted incorrectly

In this scenario the given word may missing one letter from the actual word, Example: The user enters *Ordnary* instead of *Ordinary* (i.e. leaves out the i). During this case first we grab the actual input word from the user and made the iteration with the list of words in our map during the iteration we take each and every character and check for the match if one character is not matches with the actual character we skip that execution iteration for once and continue the iteration for the next letter of the processing word with the same letter of the user word, We maintain the jump count and grab all the words with one possible jump and we will output the word which has more occurrence in the given corpus dictionary.

Scenario 2: One character in the string is incorrectly replaced by another one

In this scenario the given word may have one incorrect letter of the another letter, Example: The user enters *Accedent* instead of *Accident*. In this case as like the scenario do the iteration with the list of words in our map during the iteration we take each and every character and check for the match if one character is not matches with the actual character we make a break count as one and follow the step again with the rest of the character. We grab all the words with the break count as one and we will output the word which has more occurrence in the given corpus dictionary.

Scenario 3: Swapping one pair of consecutive characters

In this case, user may incorrectly type a word by made a typo of swapping between two consecutive letters in the word. We achieved this scenario by performing the same kind of iteration and when comparing the each and every letter of the word if we noticed one letter is wrong the then we will make a temp compare with the next letter processing word if it is matches then we will perform the temp swap between the letter and continue the execution with one break down count if we noticed more than one swap of letter is happening we will skip that word. We collect all the words which only matches with one swap and output the most occurrence word as an output.

Scenario 4: One extra character somewhere in the word

In this scenario, User will type the word with one additional repetition of the same letter again. Example: The user enters *Heello* instead of *Hello,* In this case as the normal work flow we will compare the each and every letter in the sequence manner if we notice any letter doesn't match then we will check whether is there is a repetition letter of the previous letter if so we will repeat the same iteration with the next letter of the user word and follow this execution of one break if it happen again we will skip that processing word and go for next one. In this way we will obtain all the word which matches the one break case and display the most occurrence word from the corpus dictionary.

Drawback:

We achieve the above mentioned scenario in our best way to option the expected result but during the implementation we need to come across some circumstance that make some impact in the result in that way we identified the following drawback in our implementation.

1. When we try to make the manual iteration to converting the list of words to key value pair takes more time than expected so to make a little improvement we use the inbuilt Counter concept from the collection framework.
2. We followed the scenarios in the sequence manner and so if any scenario got satisfied it wont move further for the next scenario and so if any word which comes for both scenario 1 and scenario 4, it wont do the check for scenario 4 once it satisfied the scenario 1 it will go out of the loop.
3. For every scenario we made a whole map iteration separately and it causes some time delay in the milliseconds to option the result for the scenario 4 but it wont cause much delay in displaying the result as the corpus dictionary compactable with few words. If source dictionary is big file it may cause some delay.

Conclusion:

This implementation is achieved by the custom architecture of the spell check without using any external library so the iteration operation makes the project executable in all the above scenarios so it will obtain the best result with the good performance.