Spell Checker with C++ and Python Testing

Introduction

This document presents a spell checker program written in C++ and a Python testing script to verify its functionality. The spell checker takes an input word and checks if it exists in a predefined dictionary. If the word is not correctly spelled, it suggests the closest matching word from the dictionary using the Levenshtein distance algorithm, which calculates the minimum number of edits needed to transform one word into another.

Source Code

main.cpp

The following C++ code implements the spell checker. The program uses Levenshtein distance to find the closest matching word in the dictionary when the input word is not spelled correctly.

Listing 1: main.cpp

```cpp
#include <iostream>
#include <unordered_set>
#include <vector>
#include <string>
#include <algorithm>
#include <limits>

// SpellChecker Class Definition
class SpellChecker {
public:
    SpellChecker(const std::vector<std::string>& dictionary);
    std::vector<std::string> spellCheck(const std::string& word);

private:
    std::unordered_set<std::string> wordSet;
    int calculateLevenshteinDistance(const std::string& source, const std::string&
        target);
    std::string findClosestMatch(const std::string& word);
};

// SpellChecker Implementation
SpellChecker::SpellChecker(const std::vector<std::string>& dictionary) {
    for (const auto& word : dictionary) {
        wordSet.insert(word);
    }
}

std::vector<std::string> SpellChecker::spellCheck(const std::string& word) {
    if (wordSet.find(word) != wordSet.end()) {
        return {word};  // Word is correctly spelled
    }

    std::string closestMatch = findClosestMatch(word);
    return closestMatch.empty() ? std::vector<std::string>{} : std::vector<std::string>{
        closestMatch};
}

int SpellChecker::calculateLevenshteinDistance(const std::string& source, const std::
    string& target) {
    std::vector<std::vector<int>> dp(source.size() + 1, std::vector<int>(target.size() +
         1));
```

```
38
39      for (size_t i = 0; i <= source.size(); ++i) dp[i][0] = i;
40      for (size_t j = 0; j <= target.size(); ++j) dp[0][j] = j;
41
42      for (size_t i = 1; i <= source.size(); ++i) {
43          for (size_t j = 1; j <= target.size(); ++j) {
44              int cost = (source[i - 1] == target[j - 1]) ? 0 : 1;
45              dp[i][j] = std::min({dp[i - 1][j] + 1, dp[i][j - 1] + 1, dp[i - 1][j - 1] +
                    cost});
46          }
47      }
48      return dp[source.size()][target.size()];
49  }
50
51  std::string SpellChecker::findClosestMatch(const std::string& word) {
52      int minDistance = std::numeric_limits<int>::max();
53      std::string closestMatch;
54
55      for (const auto& dictWord : wordSet) {
56          int distance = calculateLevenshteinDistance(word, dictWord);
57          if (distance < minDistance) {
58              minDistance = distance;
59              closestMatch = dictWord;
60          }
61      }
62
63      // Return suggestion only if it's reasonably close
64      return (minDistance <= word.size() / 2 + 1) ? closestMatch : "";
65  }
66
67  // Main Application
68  int main() {
69      std::vector<std::string> dictionary = {"apple", "banana", "orange", "grape", "peach"
            , "pear", "mango", "melon"};
70      SpellChecker checker(dictionary);
71
72      std::string input;
73      while (true) {
74          std::cout << "Enter a word to spell-check (or 'exit' to quit): ";
75          std::cin >> input;
76          if (input == "exit") break;
77
78          std::vector<std::string> result = checker.spellCheck(input);
79          if (result.empty()) {
80              std::cout << "No suitable suggestion found." << std::endl;
81          } else if (result[0] == input) {
82              std::cout << "The word '" << input << "' is spelled correctly." << std::endl
                    ;
83          } else {
84              std::cout << "Did you mean: " << result[0] << "?" << std::endl;
85          }
86      }
87      return 0;
88  }
```

test.py

The following Python script tests the C++ spell checker by simulating various input cases and checking the output.

Listing 2: test.py

```
1  import subprocess
2
3  def run_spellchecker_test(word, expected_output):
4      process = subprocess.Popen(["./spellchecker.exe"], stdin=subprocess.PIPE, stdout=
           subprocess.PIPE, text=True)
5      out, _ = process.communicate(input=f"{word}\nexit\n")
6      actual_output = out.splitlines()[1]  # Grabbing the result line after input prompt
7      result = "PASS" if actual_output.strip() == expected_output.strip() else "FAIL"
8      print(f"Test: {word}\nExpected: {expected_output}\nActual: {actual_output}\nResult:
           {result}\n")
```

```python

# Define test cases
tests = [
    ("apple", "The word 'apple' is spelled correctly."),
    ("applle", "Did you mean: apple?"),
    ("bananna", "Did you mean: banana?"),
    ("orrange", "Did you mean: orange?"),
    ("grap", "Did you mean: grape?"),
    ("pech", "Did you mean: peach?"),
    ("a", "No suitable suggestion found."),
    ("ra", "No suitable suggestion found."),
    ("ora", "Did you mean: orange?"),
]

# Run all tests
for word, expected in tests:
    run_spellchecker_test(word, expected)
```

Testing Process and Results

The testing script executes several test cases with various input words to verify the correctness of the spell checker. The output of each test is compared with the expected output, and the result is recorded as either "PASS" or "FAIL".

```
gorme@kombyoga MINGW64 ~/projects/algorithms-and-datastuctures/assignments/spell-checker (main)
$ python test.py
Test: apple
Expected: Enter a word to spell-check (or 'exit' to quit): The word 'apple' is spelled correctly
Actual: Enter a word to spell-check (or 'exit' to quit): The word 'apple' is spelled correctly.
Result: PASS

Test: applle
Expected: Enter a word to spell-check (or 'exit' to quit): Did you mean: apple?
Actual: Enter a word to spell-check (or 'exit' to quit): Did you mean: apple?
Result: PASS

Test: bananna
Expected: Enter a word to spell-check (or 'exit' to quit): Did you mean: banana?
Actual: Enter a word to spell-check (or 'exit' to quit): Did you mean: banana?
Result: PASS

Test: orrange
Expected: Enter a word to spell-check (or 'exit' to quit): Did you mean: orange?
Actual: Enter a word to spell-check (or 'exit' to quit): Did you mean: orange?
Result: PASS

Test: grap
Expected: Enter a word to spell-check (or 'exit' to quit): Did you mean: grape?
Actual: Enter a word to spell-check (or 'exit' to quit): Did you mean: grape?
Result: PASS

Test: pech
Expected: Enter a word to spell-check (or 'exit' to quit): Did you mean: peach?
Actual: Enter a word to spell-check (or 'exit' to quit): Did you mean: peach?
Result: PASS

Test: a
Expected: Enter a word to spell-check (or 'exit' to quit): No suitable suggestion found.
Actual: Enter a word to spell-check (or 'exit' to quit): No suitable suggestion found.
Result: PASS
```

```
Test: ra
Expected: Enter a word to spell-check (or 'exit' to quit): No suitable suggestion found.
Actual: Enter a word to spell-check (or 'exit' to quit): No suitable suggestion found.
Result: PASS

Test: ora
Expected: Enter a word to spell-check (or 'exit' to quit): Did you mean: orange?
Actual: Enter a word to spell-check (or 'exit' to quit): No suitable suggestion found.
Result: FAIL


gorme@kombyoga MINGW64 ~/projects/algorithms-and-datastuctures/assignments/spell-checker (main)
$
```

Time Complexity Analysis

The spell checker algorithm primarily relies on the calculation of the Levenshtein distance, which has a time complexity of $O(m \times n)$, where m and n are the lengths of the source and target strings. the Levenshtein distance calculates the minimum number of edits needed to transform one word into another. Checking the dictionary for the closest match requires calculating the distance for each word in the dictionary, resulting in an overall complexity of $O(d \times m \times n)$, where d is the number of words in the dictionary. This way works efficiently for small dictionaries, but bigger dictionaries may require optimizations, maybe using a trie or limiting suggestions by minimum distance.

Conclusion

The spell checker program works as expected for most test cases, meaning the correct words or indicating correct spelling. However, it fails for the word "ra," which could be due to limitations in the Levenshtein distance threshold or the dictionary's contents.