Dhirubhai Ambani
Institute of Information and Communication Technology

Subject: Data structures

Subject code: IT205

CAPSTONE PROJECT

Name of group: Nibble Ninjas

Topic: Spell Checker   **Project I'D: P8**

Name of Project Members with Student ID And GITHUB ACCOUNT :

1 . Abhishek Guna   (202301148)   (HINOKAM-ii)

2 . Manthan Jethva   (202301175)   (MANTHANJETHVA)

3 . Meet Dobariya   (202301115)   (meetdobariya)

4 . Shubham Kankotiya   (202301155)   (Shubham-Kankotiya)

Github Repository Link :

LINK

➡ THE REPOSITORY INCLUDES TWO HEADER FILES (dictionary.h & HIGHLIGHT.h) , ONE CPP FILE (main.cpp) , THREE TEXTS FILE (dictionary.txt & input.txt & shortcut.txt) AND ONE README FILE .

➡ IN dictionary.h OUR CLASS OF DATA STRUCTURES IS PRESENT , IN main.cpp THE MAIN FUNCTION IS PRESENT AND ALL THE OTHER HEADER FILES ARE INCLUDED IN THE main.cpp FILE , dictionary.txt HAS ALL DICTIONARY WORDS , input.txt FILE IS THE FILE ON WHICH OUR SPELL CHECKER WORKS (IF THERE IS ERROR IN ANY WORD IN input.txt THE main.cpp PROGRAM WILL RESOLVE IT BY PROVIDING SUGGESTIONS AND THEN CHANGE THE WORD IN INPUT FILE USING FILE HANDLING) .

BRIEF OVERVIEW ON PROJECT : (STEPS TO MAKE SPELL CHECKER ....)

1 . Implementation of dictionary :

creation of data structure to store dictionaries (all corrected words) . We could use data structures like hash tables, trie .

2 . Spell Checking Algorithm :

Developing an algorithm to compare words from the input text with those in the dictionary to identify misspelled words. This algorithm may include techniques such as Levenshtein distance, which measures the similarity between two words by calculating the minimum number of operations.

3 . Suggestions Generation:

Implementing a mechanism to generate suggestions for correcting misspelled words based on their similarity to correctly spelled words in the dictionary. This could involve techniques like generating all possible variations of a word .

4 . Creating console based User Interface .

PSEUDOCODE OF THE PROJECT WITH TIME COMPLEXITIES AND SPACE COMPLEXITIES :

Needed function in class of data structure :

1 . Two functions to insert words into the dictionary.
2 . Function to check if a word is in the dictionary.
3 . Function to implement Levenshtein distance.
4 . Function to generate suggestions for misspelled words(which is using another function Levenshtein distance).
5 . Function to check and showing spelling errors in a input file and suggest corrections.
6 . Function to replace misspelled words.

Some function for additional features (BONUS) :

→ **Functions to add shortcuts to words and replace it in the input file .**
→ **Function to add any particular new word to dictionary based on user experience .**

Pseudocode -

```
// Define a class for spell checker
Class spellchecker:
    Define an unordered_map dictionary to store correct words
    Define an unordered_map short_cut to store shortcuts
    Define main_dictionary and shortcut_location strings
      // Constructor
    Function spellchecker():
        Call insert_dictionary(main_dictionary)
        Call insert_shortcut_file(shortcut_location)
```

→ **We have used unordered_map instead of map because insertion, deletion and search operation in map is O(log n) whereas O(1) on average case .**

```
// (1.1) . Function to insert a word into the dictionary
    Function insert_spell(word):
        dictionary[word] = true
// (1.2) . Function to insert words from a dictionary file
    Function insert_dictionary(dictionary_name):
        Open the dictionary file
        If the file is open:
            Read each word from the file and insert it into the
dictionary
            Close the file
        Else:
            Print an error message
```

For inserting a word -
— Time Complexity:
O(1) on average for inserting a word
— Space Complexity:
 O(1) for storing the word

For inserting the whole dictionary -
— Time Complexity:
O(N), where N is the number of words in the dictionary.
— Space Complexity:
O(N) to store the word

```
// 2 . Function to check if a word is in the dictionary
   Function is_in_dictionary(word):
       Return true if word is in dictionary, else false
```

— Time Complexity: O(1) on average for unordered_map
— Space Complexity: O(1)

```
// 3 . function to calculate Levenshtein distance between two strings
Function Distance(str1, str2):
    n = length of str1
    m = length of str2
    Create a 2D array dis[n+1][m+1]

    // Initialize the first row and column
    For i from 0 to n:
        dis[i][0] = i
    For j from 0 to m:
        dis[0][j] = j

    // Calculate distances using dynamic programming
    For i from 1 to n:
        For j from 1 to m:
            If str1[i-1] equals str2[j-1]:
                dis[i][j] = dis[i-1][j-1]
            Else:
                dis[i][j] = 1 + minimum of {dis[i-1][j], dis[i][j-1],
dis[i-1][j-1]}

    Return dis[n][m]
```

— Time Complexity :
O(n * m), where n is the length of str1 and m is the length of str2 .

— Space Complexity : O(n * m) for the 2D array 'dis' .

```
// 4 . Function to generate suggestions for a misspelled word
    Function suggestion(MisSpellWord):
        Initialize an empty vector final_suggestion
        Open the main dictionary file
        If the file is open:
            Read each word from the file
            Calculate Levenshtein distance between each word and
MisSpellWord
            Calculate priority based on distance and matching
characters
            Store the word and priority in suggestion_word vector
            Sort suggestion_word based on priority
            Add top 5 suggestions to final_suggestion
            If suggestion_word is empty, print "zero suggestion"
        Else:
            Print an error message

        Return final_suggestion
```

— Time Complexity :

O(N * (M + log M)), where N is the number of words in the dictionary file, and M is the length of the longest word. The O(N) comes from iterating over all words in the dictionary, and the O(M + log M) comes from sorting the suggestions.

— Space Complexity : O(N) for storing the suggestions

```
// 5 . Function to check spelling errors in a file and suggest
corrections
    Function checking_file(filename):
        Open the input file
        If the file is open:
            Read each character from the file
            If the character is a letter, add it to a word
            If a word is formed:
                If the word is in the dictionary, print it
                Else, print the word underlined
                Reset the word
            Else, print the character
        Else:
            Print an error message

        Return count of misspelled words
```

— Time Complexity :
O(N * M), where N is the number of characters in the file and M is the average length of the words.

— Space Complexity : O(1)

```
// 6 . Function to replace misspelled words in a file
    Function replace_spell(file_name):
        Open the input file
        If the file is open:
            Create a temporary file
            Read each character from the file
            If the character is a letter, add it to a word
            If a word is formed:
                If the word is a shortcut, ask for replacement
                Else if the word is in the dictionary, write it to the
temporary file
                Else, suggest corrections and write the chosen word to
the temporary file
                Reset the word
            Else, write the character to the temporary file
            Close both files, remove the original file, and rename the
temporary file
```

— Time Complexity :
O(N * M), where N is the number of characters in the file and M is the average length of the words
— Space Complexity :
O(N), where N is the size of the file

```
// functions to add shortcuts to words and replace it in the input file
  // Function to insert a shortcut word into the shortcuts dictionary
    Function insert_shortcut_word(key, ans):
        Open the shortcut file
        If the file is open:
            Write the key and answer to the file
            Close the file
        Else:
            Print an error message
 // Function to read shortcut words from a file
    Function insert_shortcut_file(shortcut_filename):
        Open the shortcut file
        If the file is open:
            Read each key and answer from the file and store them in
the shortcuts dictionary
            Close the file
```

For inserting N shortcut words in file :
—Time Complexity : O(N)
—Space Complexity : O(N)

For replacing words :
—Time Complexity : O(1)
—Space Complexity : O(1)

```
// function to add any particular new word to dictionary
function insert_spell_to_dictionary(word):
    // Open the main_dictionary file in append mode
    file = open(main_dictionary, append mode)

    // Write the word to the file followed by a newline character
    write word to file

    // Close the file
    close file
```

Time complexity : O(1) , Constant time
Spaces complexity : O(1) , Constant time

WHY WE USE HASH TABLES OVER TRIE DATA STRUCTURES :

- **BOTH THE STRUCTURES HAVE THEIR ON ADVANTAGES AND APPLICATIONS , BUT WE FOUND SOME EDGE OF USING HASH TABLE OVER TRIE BECAUSE OF REASONS DISCUSS BELOW .**

- **Efficiency :**

  HASH TABLE CONSTANT TIME COMPLEXITY FOR MOST BASIC OPERATIONS LIKE INSERTION , DELETION AND SEARCH ON AVERAGE CASES .

- **Search Speed:**

  WHILE TRIES OFFER EFFICIENT PREFIX-BASED SEARCHES, HASH TABLES CAN BE EQUALLY EFFICIENT FOR EXACT MATCHING SEARCHES, WHICH IS OFTEN THE PRIMARY REQUIREMENT FOR SPELL CHECKERS.

- **Ease of Implementation :**

  HASH TABLES ARE GENERALLY SIMPLER TO IMPLEMENT AND UNDERSTAND COMPARED TO TRIES, WHICH REQUIRE A MORE COMPLEX STRUCTURE INVOLVING MULTIPLE NODES AND POINTERS.

- **Flexibility :**

  HASH TABLES CAN BE EASILY ADAPTED TO HANDLE VARIOUS TYPES OF KEYS, NOT JUST STRINGS. THIS MAKES THEM SUITABLE FOR SPELL CHECKERS THAT MAY NEED TO HANDLE DIFFERENT TYPES OF DATA BEYOND JUST WORDS

DRAWBACK OF NOT USING TRIE DATA STRUCTURE :

Space Efficiency for Similar Keys: Tries can be more space-efficient than hash tables when storing a large number of keys
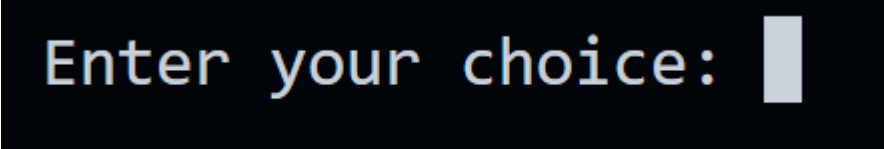
DEMO SCREENSHOT OF THE PROGRAM -

```
** Spellchecker Menu **
1. Check Spelling in a File and replace wrong words
2. Add your word in dictonary
3. Add a Shortcut
4. Exit
Enter your choice: 1
Enter File Name to check spelling: input.txt
 ~> Following underlined words are wrong words
in the process of developing new algorithms, it's important to ensure that the code is both efficient and reliable. ho
wever, sometimes misspelled words or typos can slip through the cracks and lead to uexpected behaviour. that's why hav
ing a reliable spellcheckeer built into your developement environment can greatly aid in catching these errors early i
n the development cycl. it's also a good idea to periodically review your code for spelling misstates and other common
 grammatical issues, as these can impact the readability and maintainability of your codebase in the long run.
3 Misspelled words found.
 ~> wrong words have been corrected in the input.txt you can check in the file
 - Mistake in the word uexpected
 - dictionary open successfully
 - suggestion sorted properly
 - We found these suggestions for you:
1 expected
2 unexpected
3 expecter
4 experted
5 exsected
 ~> Choose any of them or enter 0 to keep the word unchanged: 2
```

1 . SPELLCHECKER MENU :

```
** Spellchecker Menu **
1. Check Spelling in a File and replace wrong words
2. Add your word in dictonary
3. Add a Shortcut
4. Exit
```

THIS MENU HAS 4 PARAMETERS SPELL CHECKING IN A PARTICULAR FILE , ADDING WORDS TO EXISTING DICTIONARY , ADD SHORTCUTS AND TERMINATE(EXIT).

2 . ENTERING THE CHOICE (WHAT USER WANTED TO DO) :

Enter your choice:

WHICH IN THIS CASE IS ONE (MAIN SPELL CHECKER).

3 . THE PROGRAM AS PER THE CHOICE :

THE SPELL CHECKER AND SUGGESTIONS -

FIRSTLY ENTER THE FILE FOR SPELL CHECKING THEN IT IS SHOWING THE WHOLE TEXTS WITH WRONG WORDS HAVING UNDERLINE .
THEN STEP BY STEP IT IS SHOWING CORRECTED SUGGESTIONS BASED ON OUR ALGORITHM .

```
Enter your choice: 1
Enter File Name to check spelling: input.txt
 ~> Following underlined words are wrong words
as the sun began to set behind the silhouetted trees, casting a warm glow over the landscape, sarah sat on the porch s
wing, lost in both. she pondered the mysteries of the univers, wondering about the true nature of distance. suddenly,
a gentle breeze blew, rustling the leaves and interrupting her reverie.sarah smiled, feeling a sense of belace accend
upon her.in that moment, she realized that life's greatest journeys often begin with a singl step, and she resolved to
 embrace each new adventure with open arms.
2 Misspelled words found.
 - Mistake in the word univers
 - dictionary open successfully
 - suggestion sorted properly
 - We found these suggestions for you:
1 uniters
2 universe
3 aivers
4 clivers
5 divers
 ~> Choose any of them or enter 0 to keep the word unchanged: 2
 - Mistake in the word singl
 - dictionary open successfully
 - suggestion sorted properly
 - We found these suggestions for you:
1 sinal
2 sing
3 singe
4 singh
5 single
 ~> Choose any of them or enter 0 to keep the word unchanged: 5
all spell corrected
 ~> wrong words have been corrected in the input.txt you can check in the file
```

NOW BOTH WORDS IN THE INPUT FILE WILL BE CORRECTED .

CHOICE N0 . 2 — NOW FOR INSERTING NEW WORD IN DICTIONARY :

```
Enter your choice: 2
ENTER WORD YOU WANTED TO INSERT -
manthan
ENTERED WORD ADDED SUCCESSFULLY !!
```

WE ARE GIVING CHOICE 2 AS INPUT THEN GIVING INPUT AS THE WORD WE WANTED TO ADD , IN THIS CASE IT IS "manthan" . THEN IT IS GIVING A POSITIVE FEEDBACK AS WORD ADDED .....

NOW CHOICE NO.3 —

```
Enter your choice: 3
Enter the short cut key: jsr
Enter full form: jai shree ram
 ~> shortcut added successfully.
```

WE HAVE ADDED JSR AS "JAI SHREE RAM" , WHENEVER THE INPUT FILE IS HAVING THE SHORTCUT WORD JSR IT WILL CONVERT IT INTO "JAI SHREE RAM".
WE HAVE DEVELOPED THIS IN SUCH A WAY THAT , IT CONTAINS MANY SHORTCUTS AS DEFAULT .
SUCH AS GN➡️GOOD NIGHT , ...

CHOICE NO . 4 FOR TAKING EXIT FROM THE PROGRAM

WE HAVE TRIED TO MAKE IT AS USEFUL AS WE CAN BASED ON THE SKILL WE HAVE DEVELOPED AND THE KNOWLEDGE WE GOT ABOUT DSA .

HERE ARE SOME SOME REFERENCES THAT WE WANTED TO GIVE THAT HELPED US TO IMPLEMENT THE CODE :

1. YOUTUBE VIDEO ON ALGORITHM :
   CHECK IT OUT ↗

2. YOUTUBER STRIVER FOR Levenshtein distance :
   CHECK IT OUT ↗

3. SOME ARTICELSE ON GFG AND STACKOVERFLOW :
   CHECK IT OUT ↗ (GFG)
   CHECK IT OUT ↗ (STACK OVERFLOW)