819. Most Common Word

String

Counting

Problem Description

Hash Table

Easy

The problem requires that you find the most frequent word from a given string, paragraph, which is not included in a list of banned words, banned. The string paragraph can contain uppercase and lowercase letters. Finally, the result should be a lowercase word that occurs the most frequently in paragraph and is not present in the banned list. It is assured that there is at

least one such word, and there is only one unique answer.

Intuition

The challenge is to accurately count word occurrences while excluding banned words and managing different cases (uppercase/lowercase). The approach involves several steps. First, normalize the paragraph by converting it to lowercase to ensure case insensitivity. Next, use regular expressions to extract words by ignoring punctuation and other non-alphabetic characters. After the words have been extracted, use the Counter class from the collections library to count the frequency of each word. We

then iterate through the most common words while skipping those present in the set of banned words. A set is used for the banned words to achieve O(1) lookup times.

Solution Approach

Normalization of Text: First, we convert the entire paragraph to lowercase using the lower() method. This ensures that all words are counted in a case-insensitive manner.

Word Extraction: We then use the re.findall() function from the re module (which stands for "regular expression") to

through this list to find and retrieve the first word that is not present in the set of banned words.

To illustrate the proposed solution approach, imagine you have the following paragraph and banned list:

The solution to the problem utilizes several Python features and data structures to approach the task efficiently:

- extract all the words in the text. The regular expression '[a-z]+' is used to match continuous sequences of lowercase letters ('a' to 'z'), effectively skipping over any punctuation or other characters.
- word. This data structure is ideal for this use case because it allows for efficient counting and retrieval of the most common elements. Set for Banned Words: The banned words are stored in a set, which allows us to check if a word is banned in constant time,

Word Frequency Counting: Python's Counter class from the collections module is used to count the frequency of each

thanks to the underlying hash table implementation of sets in Python. Finding the Most Common Non-Banned Word: We use the most_common() method of the Counter class to get a list of words

sorted by their frequency in descending order. The next() function, in conjunction with a generator expression, iterates

- The final line return next(word for word, _ in p.most_common() if word not in s) uses a generator expression to go through the words in the order of decreasing frequency and checks if the word is not in the banned set s. The underscore _ is used to
- With the help of these tools and constructs, the solution efficiently finds the most frequent, non-banned word in the input paragraph.

ignore the frequency in the tuple returned by p.most_common() since we only care about the word itself in this context.

paragraph: "Bob hit a ball, the hit BALL flew far after it was hit." banned: ["hit"] Following the solution steps:

Result: "bob hit a ball, the hit ball flew far after it was hit." Word Extraction: Use a regular expression to extract all the words.

'hit'])

Result: {"hit"}

not banned.

Result: 'ball'

included in the banned list.

from collections import Counter

from typing import List

import re

class Solution:

Example Walkthrough

Code Example: re.findall(r'[a-z]+', "bob hit a ball, the hit ball flew far after it was hit.")

Result: Counter({'hit': 3, 'ball': 2, 'bob': 1, 'a': 1, 'the': 1, 'flew': 1, 'far': 1, 'after': 1, 'it': 1, 'was': 1})

Word Frequency Counting: Use the Counter to count the frequency of each word.

Result: ['bob', 'hit', 'a', 'ball', 'the', 'hit', 'ball', 'flew', 'far', 'after', 'it', 'was', 'hit']

Normalization of Text: Convert the entire paragraph to lowercase.

Set for Banned Words: Create a set of the banned words.

def mostCommonWord(self, paragraph: str, banned: List[str]) -> str:

Use a generator expression to iterate over the most common words

and return the first one which is not in the set of banned words

banned_words_set = set(word.lower() for word in banned)

// Initialize variables to track the most frequent word

if (entry.getValue() > maxFrequency) {

maxFrequency = entry.getValue();

mostFrequentWord = entry.getKey();

// Return the most common word that is not banned

// Method to find the most common non-banned word in a paragraph.

string mostCommonWord(string paragraph, vector<string>& banned) {

if (!isalpha(paragraph[i]) && (++i > 0)) continue;

unordered_set<string> bannedWords(banned.begin(), banned.end());

for (int i = 0, maxCount = 0, length = paragraph.size(); i < length;) {</pre>

// Skip non-alpha characters and continue to the next iteration.

// Convert the word to lowercase and store in word variable.

// Return the most common word after processing the entire paragraph.

// Split the paragraph into words using a regex that matches non-letter characters.

// Create a set of banned words for quick lookup.

// Use a map to count the occurrence of each word.

// Loop through the characters of the paragraph.

unordered_map<string, int> wordCount;

// Variable to store the most common word.

// Find the start of the next word.

if (wordCount[word] > maxCount) {

maxCount = wordCount[word];

function mostCommonWord(paragraph: string, banned: string[]): string {

const lowercaseParagraph = paragraph.toLocaleLowerCase();

// A set containing the banned words for quick look-up.

for (const word of lowercaseParagraph.split(/[^A-Za-z]/)) {

wordCount.set(word, (wordCount.get(word) ?? 0) + 1);

// Increment the word count for each occurrence of the word.

// Initialize a placeholder for the most common word and its count.

// Iterate through the map entries to find the word with the highest count.

// A map to keep track of the count of each word.

const wordCount = new Map<string, number>();

const bannedWords = new Set<string>(banned);

// Skip empty strings or banned words.

for (const [currentWord, count] of wordCount) {

mostCommon = [currentWord, count];

if (count > mostCommon[1]) {

if (word === '' || bannedWords.has(word)) {

// Convert the paragraph to lower case for consistent comparison.

mostCommon = word;

// Iterate through the map entries to find the most common word

for (Map.Entry<String, Integer> entry : wordFrequency.entrySet()) {

words = re.findall(r'[a-z]+', paragraph.lower())

Return the most common non-banned word

return most_common_word

Convert the set of banned words to lowercase and create a set for fast look-up

The '+' indicates that we're looking for one or more alphabetic characters together as a word

most_common_word = next(word for word, _ in word_count.most_common() if word not in banned_words_set)

Code Example: next(word for word, _ in Counter(['bob', 'hit', 'a', 'ball', 'the', 'hit', 'ball', 'flew', 'far', 'after', 'it', 'was', 'hit']).most_common() if word not in {"hit"})

So, in this example, ball would be the most frequent, non-banned word from the paragraph. It appears 2 times and is not

Finding the Most Common Non-Banned Word: Iterate through the list of most common words and return the first one that is

Code Example: Counter(['bob', 'hit', 'a', 'ball', 'the', 'hit', 'ball', 'flew', 'far', 'after', 'it', 'was',

Solution Implementation Python

Use regular expression to find all words (sequences of alphabetic characters) and convert them all to lowercase

Create a Counter object to count the occurrences of each word in the paragraph word_count = Counter(words)

```
Java
import java.util.HashSet;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
class Solution {
   // Compilation of pattern to match words with lowercase letters
    private static final Pattern PATTERN = Pattern.compile("[a-z]+");
    public String mostCommonWord(String paragraph, String[] banned) {
       // Initialize a set to keep track of banned words for quick lookup
       Set<String> bannedWords = new HashSet<>();
        for (String word : banned) {
            bannedWords.add(word);
       // Initialize a map to count the occurrence of each word
       Map<String, Integer> wordFrequency = new HashMap<>();
       // Preprocess the paragraph to make it lowercase for case insensitivity
       Matcher matcher = PATTERN.matcher(paragraph.toLowerCase());
       // Find all matches and populate the word frequency map
       while (matcher.find()) {
           String currentWord = matcher.group();
           // Skip the word if it is banned
           if (!bannedWords.contains(currentWord)) {
                wordFrequency.put(currentWord, wordFrequency.getOrDefault(currentWord, 0) + 1);
```

C++

int maxFrequency = 0;

return mostFrequentWord;

#include <unordered_set>

#include <unordered_map>

#include <cctype>

#include <string>

#include <vector>

class Solution {

String mostFrequentWord = null;

};

TypeScript

public:

```
while (j < length && isalpha(paragraph[j])) {</pre>
    word.push_back(tolower(paragraph[j]));
    ++j;
// Update the starting position for the next word.
i = j + 1;
// If the word is in the banned list, skip to the next word.
if (bannedWords.find(word) != bannedWords.end()) continue;
// Increment the count of the word in the map.
++wordCount[word];
// If the current word's count is greater than maxCount, update the most common.
```

return mostCommon;

continue;

let mostCommon = ['', 0];

string mostCommon;

int j = i;

string word;

```
// Return the most common word.
return mostCommon[0];
```

import re

class Solution:

from collections import Counter

from typing import List

Time Complexity

```
def mostCommonWord(self, paragraph: str, banned: List[str]) -> str:
       # Convert the set of banned words to lowercase and create a set for fast look-up
       banned_words_set = set(word.lower() for word in banned)
       # Use regular expression to find all words (sequences of alphabetic characters) and convert them all to lowercase
       # The '+' indicates that we're looking for one or more alphabetic characters together as a word
       words = re.findall(r'[a-z]+', paragraph.lower())
       # Create a Counter object to count the occurrences of each word in the paragraph
       word_count = Counter(words)
       # Use a generator expression to iterate over the most common words
       # and return the first one which is not in the set of banned words
       most_common_word = next(word for word, _ in word_count.most_common() if word not in banned_words_set)
       # Return the most common non-banned word
       return most_common_word
Time and Space Complexity
  The provided code snippet defines a function that returns the most common non-banned word from a string paragraph, ignoring
  words in the banned list banned. It utilizes regular expressions to identify words and the collections. Counter class to count
  occurrences.
```

which in the worst case is 0(m log m) if the Counter implementation uses Timsort, a variation of sorting that has this worstcase complexity. next with generator expression: In the worst case, we might need to iterate through all m words to find one that is not in the

banned set. Checking if a word is in the banned set is 0(1) due to using a set.

The time complexity of the mostCommonWord function is primarily determined by several operations:

paragraph, as it must traverse the entire paragraph and perform pattern matching.

paragraph.lower(): Lowercasing the entire paragraph has a time complexity of O(n).

The overall time complexity T(n) therefore is dominated by the sorting step in Counter.most_common, which gives us: $T(n) = O(n) + O(n) + O(m) + O(m \log m)$ Since m, the number of unique words, is typically much less than n, we can consider O(n) as a more practical worst-case estimate

re.findall: This function is used to find all substrings in paragraph that match the regular expression [a-z]+, effectively

extracting all words composed of lowercase letters. The complexity of this operation is O(n), where n is the length of

Counter.most_common: The Counter object is created from the list of words found using re.findall, which also has a

complexity of O(m) where m is the number of words in the paragraph. The most_common method then sorts these word counts,

Space Complexity The space complexity is also determined by several factors:

for large paragraphs.

S(m, b) = O(m + b)

The counter p: In the worst case, if every word in paragraph is unique, the space needed for the counter would be 0(m).

The set of banned words s: If there are b banned words, the space complexity for the set is O(b).

The list of words produced by re.findall: This list also has a space complexity of O(m) in the worst case.

Therefore, the total space complexity S(m, b) is a combination of the space needed for these data structures: S(m, b) = O(b) + O(m) + O(m)

In conclusion, the mostCommonWord function has a time complexity of O(n) and a space complexity of O(m + b), where n is the

length of paragraph, m is the number of unique words in the paragraph, and b is the number of banned words.