748. Shortest Completing Word

Easy Array Hash Table String

Leetcode Link

Problem Description

In this LeetCode problem, we are given a string `licensePlate` and an array of strings `words`. Our task is to find the shortest word in `words` that contains all the letters in `licensePlate`, disregarding any numbers and spaces, and treating letters as case-insensitive. Notably, if a letter repeats in the `licensePlate`, it must appear at least as many times in the word. The essence of the problem is to identify the most concise word from an array that encapsulates all alphabetic characters of the given license plate string.

Intuition

To arrive at the solution for this problem, we breakdown the requirements into smaller steps and handle each part. First, we create a function called `count` that converts a given word into a frequency array that represents how many times each letter appears in the word.

Once we have `count`, we need a way to determine if a word in `words` can be considered a "completing word". To do this, we craft a `check` function that compares two frequency arrays: one from the cleaned `licensePlate` and one from a candidate word. The `check` function goes over each letter's frequency and ensures the candidate word has equal or more occurrences of each letter found in `licensePlate`.

Then we iterate over each word in `words` and apply the `count` function to it. If it passes the `check` with the `licensePlate`'s frequency array, it means we've found a potential completing word. We maintain a variable for the shortest word found (`ans`) and its length (`n`). If a new completing word is shorter than the current best, we update `ans` with this new word.

This way, by prioritizing words that meet the completing criteria and are shorter than any others we've seen, we can solve the problem effectively.

Solution Approach

The solution approach for finding the shortest completing word involves the use of frequency arrays and efficient checks for each word in the `words` array against the `licensePlate`. Here's a step-by-step explanation of the implementation:

The count function

We define a `count` function that takes a word and transforms it into a frequency array, which is an array of 26 integers (corresponding to the 26 letters of the English alphabet), each element representing the count of a specific letter from 'a' to 'z'. The `ord` function is used to get the ASCII value of a character, and we subtract the ASCII value of 'a' to map 'a' to index 0, 'b' to index 1, and so on up to 'z'.

```python
def count(word):
    counter = [0] * 26
    for c in word:
        counter[ord(c) - ord('a')] += 1
    return counter
```

The check function

The `check` function compares two frequency arrays: one from the license plate (`counter1`) and the other from the current word (`counter2`). If for any letter the count in the license plate is greater than the count in the word, `check` returns `False` indicating that the word is not a completing word. Otherwise, if all letter counts are equal or higher in the word than in the license plate, it returns `True`.

```python
def check(counter1, counter2):
    for i in range(26):
        if counter1[i] > counter2[i]:
            return False
    return True
```

The main loop

The main part of the algorithm is a loop through each word in `words`. It initializes two variables: `ans`, to store the currently found shortest completing word, and `n`, to store its length.

```python
counter = count(c.lower() for c in licensePlate if c.isalpha())
ans, n = None, 16
for word in words:
    if n <= len(word):
        continue
    t = count(word)
    if check(counter, t):
        n = len(word)
        ans = word
```

The process for each word in the loop is as follows:

1. We skip the word if its length is not less than the current shortest completing word's length (`n`) to optimize performance.
2. We generate a frequency array of the word (`t`) using `count`.
3. We use `check` to see if `t` meets the requirements compared to `counter`, which is the frequency array for `licensePlate`.
4. If it does, this is the new shortest completing word, and we update `ans` with this word and `n` with its length.

The loop proceeds until all words have been checked, and the shortest completing word (`ans`) is returned as the answer.

This algorithm effectively leverages data structures (arrays) and functions to compare character frequencies, significantly simplifying and optimizing the process of determining completing words.

Example Walkthrough

To illustrate the solution approach, let's take a small example. Suppose `licensePlate` is "aBc 1123" and `words` is `["step", "steps", "stripe", "stepple"]`.

The first step would be to create a frequency array for the cleaned `licensePlate`, which disregards numbers and spaces and is case-insensitive. This means "aBc" becomes "abc" and our count function generates an array `[1, 1, 1, 0, ..., 0]`, indicating that there's one 'a', one 'b', and one 'c'.

Iterating Over Each Word

We then iterate over each word in `words` and check if it's a completing word for `licensePlate`.

- For "step", we get the frequency array `[0, 0, 0, 0, 1, 0, ..., 1]`, which has an 'e' and a 'p', but lacks 'a' and 'b'.
- For "steps", the frequency array is `[0, 0, 0, 0, 1, 0, ..., 1, 1]`. This word also covers 's', but still misses 'a' and 'b'.
- "stripe" has a frequency array `[0, 0, 0, 0, 1, 0, ..., 1, 1, 1]`, containing 'e', 'p', 'r', 's', 't', and 'i', and it does have at least one 'a', 'b', and 'c'.
- Finally, "stepple" has the array `[0, 0, 0, 0, 1, 0, ..., 2, 2, 1]`, where there is more than one occurrence of some letters, but it's longer than "stripe".

Through our `check` function, we see that only "stripe" has equal or more occurrences of the letters in `licensePlate`, which are 'a', 'b', and 'c', so it is a potential completing word.

Finding the Shortest Completing Word

Since "stripe" is shorter than "stepple" and it contains all the required letters (ignoring the unnecessary 'i', 'r', and 't'), "stripe" becomes our `ans` and its length, 6, becomes `n`.

As no other word in `words` is both a completing word for "aBc 1123" and shorter than "stripe", "stripe" is the answer we return. It successfully encapsulates all alphabetic characters of the license plate "aBc 1123" in the most concise form found in the list `words`.

This example demonstrates how the `count` and `check` functions, together with an iterating loop, can be used to efficiently solve the problem.

Python Solution

```python
class Solution:
    def shortestCompletingWord(self, licensePlate: str, words: List[str]) -> str:

        # Helper function to count the frequency of each letter in a word.
        def count_letters(word):
            # Initialize a counter for 26 letters of the alphabet.
            counter = [0] * 26
            # Increment the letter's corresponding counter.
            for char in word:
                counter[ord(char) - ord('a')] += 1
            return counter

        # Helper function to check if the word contains all the
        # needed letters with required frequency.
        def contains_needed_letters(license_counter, word_counter):
            # Iterate through each letter in the alphabet.
            for i in range(26):
                # If the license has more letters than the word,
                # then it's not a completing word.
                if license_counter[i] > word_counter[i]:
                    return False
            return True

        # Transforms the license plate into lowercase and filters out non-alphabetic characters.
        # Then, counts the frequency of each letter in the license plate.
        license_counter = count_letters(char.lower() for char in licensePlate if char.isalpha())

        # Initialize the variables for tracking the shortest completing word.
        shortest_completing_word = None
        min_length = 16 # Given constraint - words length doesn't exceed 15.

        # Iterate through the words list to find the shortest completing word.
        for word in words:
            # Skip the iteration if the current word is not shorter than min_length.
            if min_length <= len(word):
                continue
            # Count the frequency of letters in the current word.
            word_counter = count_letters(word)
            # Check if the current word has all the required letters
            # with at least the frequencies in the license plate.
            if contains_needed_letters(license_counter, word_counter):
                # Update min_length and shortest_completing_word with the current word's length and the word itself.
                min_length = len(word)
                shortest_completing_word = word

        return shortest_completing_word

```

Java Solution

```java
class Solution {
    // Finds the shortest completing word in words that covers all letters in licensePlate
    public String shortestCompletingWord(String licensePlate, String[] words) {
        // Count the frequency of letters in the license plate, ignoring case and non-letters
        int[] licensePlateCounter = countLetters(licensePlate.toLowerCase());

        // Initialize the answer variable to hold the shortest completing word
        String shortestCompletingWord = null;

        // Initialize the shortest length found so far to an upper bound
        int minLength = 16; // As per the problem, words are at most 15 letters long

        // Iterate through each word in the array
        for (String word : words) {
            // Skip the word if its length is not less than the shortest length found so far
            if (minLength <= word.length()) {
                continue;
            }

            // Count the frequency of letters in the current word
            int[] wordCounter = countLetters(word);

            // Check if the current word covers all letters in the license plate
            if (doesWordCoverLicensePlate(licensePlateCounter, wordCounter)) {
                // Update the shortest completing word and the minimum length found so far
                minLength = word.length();
                shortestCompletingWord = word;
            }
        }

        // Return the shortest completing word found
        return shortestCompletingWord;
    }

    // Counts the frequency of each letter in a given word
    private int[] countLetters(String word) {
        int[] letterCounter = new int[26];
        // Iterate through each character in the word
        for (char c : word.toCharArray()) {
            // Increment the counter for the letter if it's a letter
            if (Character.isLetter(c)) {
                letterCounter[c - 'a']++;
            }
        }
        return letterCounter;
    }

    // Checks if wordCounter covers all the letters in licensePlateCounter
    private boolean doesWordCoverLicensePlate(int[] licensePlateCounter, int[] wordCounter) {
        // Iterate over each letter count
        for (int i = 0; i < 26; ++i) {
            // If the current letter's count in licensePlateCounter exceeds that in wordCounter,
            // the word does not cover the license plate, return false
            if (licensePlateCounter[i] > wordCounter[i]) {
                return false;
            }
        }
        // If all letter counts are covered, return true
        return true;
    }
}

```

C++ Solution

```cpp
#include <string>
#include <vector>
#include <cctype> // For isalpha and tolower

class Solution {
public:
    // Finds the shortest word in 'words' that contains all the letters in 'licensePlate'
    string shortestCompletingWord(string licensePlate, vector<string>& words) {
        // Count the frequency of each letter in the license plate
        vector<int> licenseCounter = countLetters(licensePlate);
        int minLength = INT_MAX;
        string shortestWord;

        // Iterate over each word in the list
        for (auto& word : words) {
            // If current word is longer than the shortest found, skip it
            if (minLength <= word.size()) continue;

            // Count the frequency of each letter in the word
            vector<int> wordCounter = countLetters(word);

            // Check if the word contains all letters in the license plate
            if (containsAllLetters(licenseCounter, wordCounter)) {
                // Update minimum length and answer if this word is shorter
                minLength = word.size();
                shortestWord = word;
            }
        }
        return shortestWord;
    }

private:
    // Counts the frequency of each letter in a word
    vector<int> countLetters(string& word) {
        vector<int> counter(26, 0); // Initiate a counter for 26 letters

        // Increment the count for each letter in the word
        for (char& c : word) {
            if (isalpha(c)) {
                ++counter[tolower(c) - 'a'];
            }
        }
        return counter;
    }

    // Checks if 'counterSource' contains at least as many of each letter as 'counterTarget'
    bool containsAllLetters(vector<int>& counterSource, vector<int>& counterTarget) {
        for (int i = 0; i < 26; ++i) {
            // If the target has more of a letter than the source, return false
            if (counterSource[i] > counterTarget[i]) {
                return false;
            }
        }
        // If the source contains at least as many of each letter, return true
        return true;
    }
};

```

Typescript Solution

```typescript
function shortestCompletingWord(licensePlate: string, words: string[]): string {
  // Count the frequency of each letter in the license plate
  const licenseCounter: number[] = countLetters(licensePlate);
  let minLength: number = Number.MAX_SAFE_INTEGER;
  let shortestWord: string = '';

  // Iterate over each word in the list
  for (let word of words) {
    // If the current word is longer than the shortest so far, skip it
    if (minLength <= word.length) continue;

    // Count the frequency of each letter in the current word
    const wordCounter: number[] = countLetters(word);

    // Check if the word contains all the letters in the license plate
    if (containsAllLetters(licenseCounter, wordCounter)) {
      // Update minimum length and shortest word if this word is shorter
      minLength = word.length;
      shortestWord = word;
    }
  }
  return shortestWord;
}

// Helper function to count the frequency of each letter in a word
function countLetters(word: string): number[] {
  const counter: number[] = new Array(26).fill(0); // Initialize a counter for 26 letters

  // Increment the count for each alphabet letter in the word
  for (let char of word) {
    if (isAlphabetic(char)) {
      counter[char.toLowerCase().charCodeAt(0) - 'a'.charCodeAt(0)]++;
    }
  }
  return counter;
}

// Helper function to verify if sourceCounter contains at least
// as many of each letter as targetCounter
function containsAllLetters(sourceCounter: number[], targetCounter: number[]): boolean {
  for (let i = 0; i < 26; ++i) {
    // If the target has more of any particular letter than the source, return false
    if (sourceCounter[i] < targetCounter[i]) {
      return false;
    }
  }
  // If source contains at least as many of each letter, return true
  return true;
}

// Helper function to check if a character is an alphabetic letter
function isAlphabetic(char: string): boolean {
  const code: number = char.charCodeAt(0);
  return (code >= 65 && code <= 90) || (code >= 97 && code <= 122); // A-Z or a-z
}

```

Time and Space Complexity

The time complexity of the `shortestCompletingWord` function primarily depends on two factors: the number of words in the input `words` list and the average length of each word. Here is the breakdown:

- The `count` function has a time complexity of `O(k)` where `k` is the length of the word. It iterates once through all the characters of the word.
- This `count` function is called for every character in the normalized `licensePlate` once, resulting in a time complexity of `O(m)` where `m` is the number of alphabetic characters in `licensePlate`.
- For each word in `words`, the `count` function is called again, giving a time complexity of `O(k * n)` where `n` is the number of words and `k` is the average length of the words.
- The function `check` has a time complexity of `O(1)` because it checks a fixed size array (26 letters of English alphabet).
- Since `check` is called for every word, we multiply it by `n`, but it doesn't affect the overall time complexity significantly due to the constant size.

Combining the above steps, the total time complexity for the function is `O(m + k * n)`.

The space complexity is determined by:

- The counter arrays used to store character frequencies for the `licensePlate` and each word. Since these are of fixed size (26), they occupy `O(1)` space.
- The space used to store the answer and the length of the shortest word found so far (`n` and `ans`) is `O(1)`.

Thus, the overall space complexity is `O(1)` because it does not scale with the size of the input.

In summary:

- Time Complexity: `O(m + k * n)`
- Space Complexity: `O(1)`