1456. Maximum Number of Vowels in a Substring of Given Length

Medium String Sliding Window

Leetcode Link

Problem Description

The problem asks for the maximum number of vowel letters that can be found in any substring of a given string `s` with a fixed length `k`. The vowels in English are defined as 'a', 'e', 'i', 'o', and 'u'. A substring is any continuous sequence of characters in the string. For example, if `s` is "banana" and `k` is 3, we need to find the substring of length 3 that has the most vowels. This problem is solved by checking each possible substring of length `k` and finding the one with the most vowels.

Intuition

To solve this problem efficiently, we use a technique called the sliding window algorithm. The idea is to maintain a window of size `k` that slides over the string `s` from the beginning to the end. This window keeps track of the number of vowels in the current substring of length `k`.

Here are the steps to implement this approach:

1. Initialize a set of vowels for quick look-up.
2. Start by counting the number of vowels in the first substring of length `k`.
3. As you slide the window by one position to the right, add one to the count if the incoming character is a vowel and subtract one if the outgoing character is a vowel.
4. Update the maximum number of vowels found so far.

This way, you don't have to check each substring from scratch; you just update the count based on the characters that are entering and exiting the window as it slides. This makes the solution very efficient because each character in `s` is processed exactly once.

Solution Approach

The implementation uses the sliding window technique alongside a set for fast vowel checking, and simple arithmetic operations for counting. Let's break down the steps and logic used in the solution:

1. **Create a Set of Vowels:** First, a set containing the vowels `'aeiou'` is created for O(1) lookup times. This is done to quickly determine if a character is a vowel.

```
vowels = set('aeiou')
```

2. **Count Vowels in the First Window:** Next, the number of vowels in the first window (the first `k` characters) of the string is counted. This forms our initial maximum.

```
t = sum(c in vowels for c in s[:k])
ans = t
```

3. **Slide the Window:** The algorithm then iterates through the string starting from the `k`th character. For each new character that enters the window, we add `1` to `t` if it's a vowel. Simultaneously, we subtract `1` from `t` if the character that is exiting the window (which is `i - k` characters behind the current character) is a vowel.

```
for i in range(k, len(s)):
    t += s[i] in vowels
    t -= s[i - k] in vowels
```

4. **Update Maximum:** After adjusting the count for the new window position, we check if the current count `t` is greater than our recorded maximum `ans`. If it is, we update `ans` to be equal to `t`.

```
ans = max(ans, t)
```

5. **Return Result:** After sliding through the entire string, `ans` will hold the maximum number of vowels found in any substring of length `k`. This result is then returned.

```
return ans
```

The simplicity and efficiency of this solution come from the fact that each character is looked at exactly twice per iteration (once when it enters the window and once when it leaves), leading to an O(n) runtime complexity where `n` is the length of the string `s`.

Example Walkthrough

Let's illustrate the solution approach using the string `s = "celebration"` and `k = 5`. We are looking for the maximum number of vowels in any substring of length 5.

1. **Create a Set of Vowels:** We initialize a set containing the vowels for quick look-up.

```
vowels = set('aeiou')
```

2. **Count Vowels in the First Window:** Our first window is 'celeb'. We count the number of vowels in this window.

```
t = sum(c in vowels for c in "celeb")  # t = 2 ('e' and 'e')
ans = t  # ans = 2
```

3. **Slide the Window:** Now we start sliding the window one character at a time to the right and adjust the count `t`.

   - Move 1: New window is 'elebr'. We add 1 for 'r' (new) and do not subtract any because 'c' (old) is not a vowel.

     `t` now becomes `3` (from 'e', 'e', 'a'), and `ans` remains `3` because `3 > 2`.

   - Move 2: New window is 'lebra'. We add 1 for 'a' (new) and subtract 1 for 'e' (old).

     `t` remains `3` (from 'e', 'a'), and `ans` remains `3`.

   - Move 3: New window is 'ebrat'. We add 1 for 'a' (new) and subtract 1 for 'e' (old).

     `t` remains `3` (from 'a', 'a', 'a'), and `ans` remains `3`.

   - Move 4: New window is 'brati'. We add 1 for 'i' (new) and do not subtract any because 'r' (old) is not a vowel.

     `t` now becomes `3` (from 'a', 'a', 'i'), and `ans` remains `3`.

   - Move 5: New window is 'ratio'. We add 1 for 'o' (new) and subtract 1 for 'b' (old).

     `t` now becomes `3` (from 'a', 'i', 'o'), and `ans` remains `3`.

   - Move 6: New window 'ation'. We add 1 for 'n' (new) and subtract 1 for 'r' (old).

     `t` now becomes `3` (from 'a', 'i', 'o'), and `ans` remains `3`.

   In each move, we adjust `t` and update `ans` if necessary:

```
for i in range(5, len(s)):
    t += s[i] in vowels
    t -= s[i - 5] in vowels
    ans = max(ans, t)
```

4. **Return Result:** After sliding through the entire string `s`, we find that `ans = 3` is the maximum number of vowels we can find in any substring of length `5`.

The final result for this example is `3`, which means the maximum number of vowels found in any substring of length `5` in the string "celebration" is three.

Python Solution

```
class Solution:
    def maxVowels(self, s: str, k: int) -> int:
        # Define a set containing all vowels for easy access and checking
        vowels = set('aeiou')

        # Initial count of vowels in the first window of size k
        current_vowel_count = sum(char in vowels for char in s[:k])

        # Initialize maximum vowels found in a window
        max_vowel_count = current_vowel_count

        # Slide the window by one character from the kth element to the end
        for i in range(k, len(s)):
            # Increment count if the incoming character is a vowel
            current_vowel_count += s[i] in vowels
            # Decrease count since we are leaving the character at the start of the window
            current_vowel_count -= s[i - k] in vowels

            # Update maximum if the current window has more vowels than previous maximum
            max_vowel_count = max(max_vowel_count, current_vowel_count)

        # Return the maximum number of vowels found in any window of size k
        return max_vowel_count
```

Java Solution

```
class Solution {
    /**
     * Calculates the maximum number of vowels in any substring of length k.
     *
     * @param s The input string.
     * @param k The length of the substring.
     * @return The maximum number of vowels found in any substring of length k.
     */
    public int maxVowels(String s, int k) {
        // Initialize the total vowel count for the first window of size k
        int totalVowelsInWindow = 0;
            stringLength = s.length();

        // Count the number of vowels in the initial window of size k
        for (int i = 0; i < k; ++i) {
            if (isVowel(s.charAt(i))) {
                ++totalVowelsInWindow;
            }
        }

        // Initialize the answer with the vowel count of the first window
        int maxVowels = totalVowelsInWindow;

        // Slide the window of size k across the string
        for (int i = k; i < stringLength; ++i) {
            // If the newly included character is a vowel, increase the count
            if (isVowel(s.charAt(i))) {
                ++totalVowelsInWindow;
            }

            // If the character that got excluded from the window is a vowel, decrease the count
            if (isVowel(s.charAt(i - k))) {
                --totalVowelsInWindow;
            }

            // Update maxVowels if the current window has more vowels than the previous ones
            maxVowels = Math.max(maxVowels, totalVowelsInWindow);
        }

        // Return the maximum number of vowels found
        return maxVowels;
    }

    /**
     * Helper method to check if a character is a vowel.
     *
     * @param character The character to be checked.
     * @return true if the character is a vowel, false otherwise.
     */
    private boolean isVowel(char character) {
        // A character is a vowel if it is one of 'a', 'e', 'i', 'o', or 'u'
        return character == 'a' ||
               character == 'e' ||
               character == 'i' ||
               character == 'o' ||
               character == 'u';
    }
}
```

C++ Solution

```
class Solution {
public:
    // Function to find the maximum number of vowels in any substring of length k
    int maxVowels(string s, int k) {
        int count = 0; // Initialize counter for vowels
        int maxVowelCount = 0; // This will store the maximum count of vowels found
        int strLength = s.size(); // Store the length of the string

        // Count the number of vowels in the first window of size k
        for (int i = 0; i < k; ++i)
            count += isVowel(s[i]);

        maxVowelCount = count; // Initialize maximum vowel count to be the count from the first window

        // Slide the window by one position to the right each time and update counts
        for (int i = k; i < strLength; ++i) {
            count += isVowel(s[i]); // Add a vowel count for new character in the window
            count -= isVowel(s[i - k]); // Subtract a vowel count for the character that is no longer in the window
            maxVowelCount = max(maxVowelCount, count); // Update maximum count if current window has more vowels
        }

        return maxVowelCount; // Return the maximum number of vowels found in any substring of length k
    }

    // Helper function to check if a character is a vowel
    bool isVowel(char c) {
        // A character is a vowel if it is 'a', 'e', 'i', 'o', or 'u'
        return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
    }
};
```

Typescript Solution

```
// Define a function that checks if a character is a vowel
function isVowel(char: string): boolean {
    return ['a', 'e', 'i', 'o', 'u'].includes(char.toLowerCase());
}

// Define a function that finds the maximum number of vowels in a substring of length k within string s
function maxVowels(s: string, k: number): number {
    // Initialize total vowel count for the first window of size k
    let totalVowels = 0;
    for (let i = 0; i < k; ++i) {
        if (isVowel(s[i])) {
            totalVowels++;
        }
    }

    // Store the maximum number of vowels found in a window of size k
    let maxVowelCount = totalVowels;

    // Use a sliding window to count vowels in the remaining windows of size k
    for (let i = k; i < s.length; ++i) {
        // Add a vowel count if the newly included character is a vowel
        if (isVowel(s[i])) {
            totalVowels++;
        }
        // Subtract a vowel count if the excluded character from the left of the window was a vowel
        if (isVowel(s[i - k])) {
            totalVowels--;
        }
        // Update maximum vowel count if the current window has more vowels
        maxVowelCount = Math.max(maxVowelCount, totalVowels);
    }

    // Return the maximum number of vowels found in any window of size k
    return maxVowelCount;
}
```

Time and Space Complexity

Time Complexity

The provided code has a time complexity of `O(n)` where `n` is the length of the string `s`. This is because the code iterates over each character of the string exactly once beyond the initial window setup. The initial sum calculation for the first `k` characters is `O(k)`, and each subsequent step in the loop is constant time `O(1)` because it involves adding or subtracting one and checking for the existence of a character in a set, which is also `O(1)`. Since `k` is at most `n`, the entire operation is bounded by `O(n)`.

Space Complexity

The space complexity of the code is `O(1)`. The space used does not grow with the size of the input string `s`. The set of vowels is of constant size (containing 5 elements) and does not change. The variables `t` and `ans` use a constant amount of space and there are no data structures that grow with the size of the input.