1704. Determine if String Halves Are Alike

Easy String Counting

Problem Description

In this problem, you are given a string `s` with an even number of characters. Your task is to divide the string into two equal parts: `a` which is the first half, and `b` which is the second half of the string `s`. After the split, you must determine whether the two strings are "alike." Two strings are considered alike if they contain the same number of vowels. Vowels are the characters 'a', 'e', 'i', 'o', and 'u' in both lowercase and uppercase forms. You need to return `true` if the substrings `a` and `b` are alike (i.e., have the same number of vowels). Otherwise, return `false`.

Intuition

To solve this problem, you need to count the number of vowels in both halves of the string and compare them. The most straightforward approach is to iterate through each half of the string separately, count the vowels in each half, and then compare the counts.

However, the provided solution takes a more efficient approach by calculating the difference in the number of vowels between both halves in a single pass. It uses a counter `cnt` that is incremented when a vowel is encountered in the first half and decremented when a vowel is found in the second half. By doing this, we combine the counting and comparison steps.

To check if a character is a vowel, the solution creates a set `vowels` containing all the vowels in both lowercase and uppercase. Using a set provides an efficient O(1) time complexity for checking if an element is present.

The variable `n` represents half the length of the string. During the iteration, for each character in the first half (indexed by `i`), `cnt` is increased by 1 if it's a vowel. Correspondingly, for each character in the second half (indexed by `i + n`), `cnt` is decreased by 1 if it's a vowel.

Solution Approach

The provided solution uses a counting approach combined with a set to efficiently check for vowels in each half of the string. Here's a detailed walkthrough of the algorithm and the data structures used:

- Initialization:
  - A set `vowels` is created containing all vowel characters. This data structure is chosen because it allows for O(1) time complexity for vowel checks.
  - A variable `cnt` is initialized to `0`. It represents the net difference in the number of vowels between the first half and the second half of the string.
  - The length of the first half of the string, denoted as `n`, is calculated using `len(s) >> 1`, which is equivalent to dividing the length of the string `s` by 2 using a bitwise right shift operator.
- Iteration:
  - We loop through `i` from the start to the midpoint (`range(n)`). For each character in the first half:
    - We increment `cnt` if the character is found in the `vowels` set (indicating that it's a vowel).
  - Simultaneously, for the corresponding character in the second half (indexed by `i + n`):
    - We decrement `cnt` if this character is a vowel.

During this single loop over the string, the `cnt` variable effectively counts the number of vowels in the first half and subtracts the number of vowels in the second half. If the two halves contain an equal number of vowels, `cnt` will be `0` at the end of the loop:

- Checking the result:
  - The `cnt` variable is compared to `0`. If they are equal, it means that there were an equal number of vowels in both halves, and we return `true`.
  - If `cnt` is not `0`, it indicates that the number of vowels in the two halves was different, and we return `false`.

This method is efficient because it traverses the string only once and performs the counting and comparison at the same time, which gives it a time complexity of O(n), where n is the length of the string. Additionally, since the solution requires a fixed amount of extra space for the vowel set and the counter, its space complexity is O(1).

Example Walkthrough

Let's illustrate the solution approach using a small example. Suppose we are given the string `s = "book"`.

1. **Initialization:**
   - We create a set `vowels = {'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'}`.
   - We initialize `cnt` to `0`.
   - Calculate half the length of the string `n = len(s) >> 1` which is `2`.
2. **Iteration:**
   - As we loop through the first half of the string `s`, we have 'b' and 'o'.
     - 'b' is not in `vowels`, so `cnt` remains `0`.
     - 'o' is in `vowels`, so we increment `cnt` by `1`. Now `cnt = 1`.
   - For the second half of the string, the corresponding indices will be `i + n = 2` and `i + n = 3`, which gives us 'o' and 'k'.
     - 'o' is in `vowels`, so we decrement `cnt` by `1`. Now `cnt` returns to `0`.
     - 'k' is not in `vowels`, so `cnt` remains `0`.
3. **Checking the result:**
   - After the iteration, `cnt` is `0`.
   - Since `cnt` is `0`, it indicates that both halves of the string contain the same number of vowels.
   - Therefore, we return `true`. The string "book" has equal numbers of vowels in both halves, making them alike.

In this example, the single pass counting of vowels in both halves efficiently concludes that the string halves are alike without the need for separate counts.

Solution Implementation

Python

```python
class Solution:
    def halvesAreAlike(self, s: str) -> bool:
        # Initialize a count variable for tracking vowel balance and
        # calculate the midpoint of the string.
        vowel_count_balance, string_length_half = 0, len(s) // 2

        # Define a set containing all lowercase and uppercase vowels.
        vowels = set('aeiouAEIOU')

        # Iterate through the first half of the string.
        for i in range(string_length_half):
            # If the character in the first half is a vowel, increment the balance counter.
            vowel_count_balance += s[i] in vowels

            # If the corresponding character in the second half is a vowel, decrement the balance counter.
            vowel_count_balance -= s[i + string_length_half] in vowels

        # If the balance counter is zero after the loop, both halves have the same number of vowels.
        return vowel_count_balance == 0
```

Java

```java
class Solution {
    // Create a set that holds all the vowel characters (both lower and upper case)
    private static final Set<Character> VOWELS = Set.of('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U');

    /**
     * This method checks if a string has equal number of vowels in both halves.
     *
     * @param s The input string to be checked.
     * @return Returns true if both halves have the same number of vowels, false otherwise.
     */
    public boolean halvesAreAlike(String s) {
        // Initialize a variable to keep the cumulative difference of vowels count
        int vowelCountDifference = 0;
        // n represents half the length of the input string, using bitwise right shift for division by 2
        int n = s.length() >> 1;

        // Loop through each character of the first half and the corresponding character of the second half
        for (int i = 0; i < n; ++i) {
            // If the character at current position of first half is vowel, increment the difference count
            vowelCountDifference += VOWELS.contains(s.charAt(i)) ? 1 : 0;
            // If the character at the corresponding position in second half is vowel, decrement the difference count
            vowelCountDifference -= VOWELS.contains(s.charAt(i + n)) ? 1 : 0;
        }

        // If vowelCountDifference is zero, it means both halves have the same number of vowels
        return vowelCountDifference == 0;
    }
}
```

C++

```cpp
#include <string>
#include <unordered_set>
using namespace std;

class Solution {
public:
    bool halvesAreAlike(string s) {
        // Define a set of vowels including both uppercase and lowercase.
        unordered_set<char> vowels = {'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'};

        // Initialize a counter to compare the number of vowels in each half.
        int vowelBalance = 0;

        // Calculate the midpoint of the string.
        int midPoint = s.size() / 2;

        // Loop to compare vowels in two halves of the string.
        for (int i = 0; i < midPoint; ++i) {
            // If the character in the first half is a vowel, increment the counter.
            vowelBalance += vowels.count(s[i]);

            // If the character in the second half is a vowel, decrement the counter.
            vowelBalance -= vowels.count(s[i + midPoint]);
        }

        // The halves are alike if the vowelBalance is 0 (equal number of vowels).
        return vowelBalance == 0;
    }
};
```

TypeScript

```typescript
// Function to check if two halves of a string contain the same number of vowels
function halvesAreAlike(s: string): boolean {
    // Define a set of vowels for quick lookup
    const vowelsSet = new Set(['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']);

    // Calculate the half length of the string
    const halfLength = s.length >> 1; // Equivalent to division by 2 using bitwise right shift

    // Initialize a counter to compare vowel occurrences in both halves
    let counter = 0;

    // Loop through each character of the first half of the string
    for (let i = 0; i < halfLength; i++) {
        // If the character is a vowel, increment the counter
        if (vowelsSet.has(s[i])) {
            counter++;
        }

        // Simultaneously check the corresponding character in the second half
        // If it is a vowel, decrement the counter
        if (vowelsSet.has(s[halfLength + i])) {
            counter--;
        }
    }

    // If the counter is zero, both halves have the same number of vowels
    return counter === 0;
}
```

```python
class Solution:
    def halvesAreAlike(self, s: str) -> bool:
        # Initialize a count variable for tracking vowel balance and
        # calculate the midpoint of the string.
        vowel_count_balance, string_length_half = 0, len(s) // 2

        # Define a set containing all lowercase and uppercase vowels.
        vowels = set('aeiouAEIOU')

        # Iterate through the first half of the string.
        for i in range(string_length_half):
            # If the character in the first half is a vowel, increment the balance counter.
            vowel_count_balance += s[i] in vowels

            # If the corresponding character in the second half is a vowel, decrement the balance counter.
            vowel_count_balance -= s[i + string_length_half] in vowels

        # If the balance counter is zero after the loop, both halves have the same number of vowels.
        return vowel_count_balance == 0
```

Time and Space Complexity

The given code snippet defines a function `halvesAreAlike` that checks if two halves of a string contain the same number of vowels.

Time Complexity:

To analyze the time complexity, we need to consider the operations that depend on the size of the input string `s`.

- The length of the string `s` is determined with `len(s)`, which is `O(1)`.
- The right shift operation `len(s) >> 1` is also `O(1)`.
- Creating the `vowels` set is `O(1)` as it is initialized with a fixed number of characters.
- The for loop runs `n` times, where `n` is half the length of the string. Inside the loop, we check the membership of `s[i]` and `s[i + n]` in the `vowels` set, and then update the `cnt` variable accordingly. The set membership checking is `O(1)` on average. Thus, the loop has a complexity of `O(n)`.

Given that `n` is `len(s) / 2`, the time complexity of the entire function is dominated by the for loop, which gives us `O(n/2)`. However, in big O notation, we drop constant factors, so the time complexity can be expressed as `O(n)` where `n` is the length of the string `s`.

Space Complexity:

- The space complexity is determined by the additional space used by the algorithm independent of the input size.
- The `vowels` set requires `O(1)` space because it contains a fixed number of vowels, which does not grow with the input.
- The `cnt` variable also takes `O(1)` space.
- The function does not use any data structures that grow with the input size.

Therefore, the space complexity of the function is `O(1)`.

In conclusion, the time complexity is `O(n)` and the space complexity is `O(1)`.