1374. Generate a String With Characters That Have Odd Counts

Easy String

Problem Description

The task in this problem is to construct a string that consists of exactly `n` characters where `n` is an integer input provided to the function. Importantly, **each character** within the resulting string must appear an **odd number of times**. For instance, if the string is composed of `3 'a'` characters and `1 'b'` character, both `'a'` and `'b'` occur an odd number of times, which is valid. The string can only be formed using lowercase English letters. There may be several valid strings that satisfy these conditions, and any one of them is considered an acceptable solution.

Intuition

To solve this problem, we first recognize a fundamental property of integers: every integer is either odd or even. We can leverage this property by building a string where all but potentially one character has an even count, and the remaining character (if necessary) has an odd count. This design ensures all characters collectively appear an odd number of times.

The solution's intuition is quite straightforward:

- If `n` is odd, we can simply create a string consisting of `n` instances of the same character (e.g., `'a'`). Since `n` is odd, our condition is immediately satisfied with just one character.
- If `n` is even, we cannot use a single character an even number of times because it would not meet the odd occurrence requirement. In this case, we construct a string with `n - 1` instances of one character (e.g., `'a'`) and `1` instance of another character (e.g., `'b'`). The `'a'`s appear `n - 1` times, which is odd since `n` is even, and the occurrence of `'b'` is odd as well since it's only there once.

Solution Approach

The implementation of the solution uses a simple conditional check to determine whether `n` is odd or even and then constructs the string accordingly.

Here's a breakdown of the solution algorithm:

1. Use the bitwise AND operator `&` to check if `n` is odd or even. For any integer `n`, `n & 1` will be `1` if `n` is odd and `0` if `n` is even.
   - `n & 1` essentially checks the least significant bit of `n`. If it is `1`, `n` is odd; if it is `0`, `n` is even.
2. If `n` is odd (`n & 1 == 1`), return a string of `n 'a'` characters.
   - This satisfies the condition since there is only one character, and it occurs `n` times, which is an odd number.
3. If `n` is even (`n & 1 == 0`), return a string of `n - 1 'a'` characters concatenated with `'b'`.
   - Here, `'a'` occurs an odd number of times (`n - 1`), and `'b'` occurs exactly once, which is also odd.

This approach does not require any complex data structures or algorithms. It simply uses two string concatenation operations and a bitwise operator for the conditional check. The time complexity is O(1) since the string construction involves a constant number of operations that do not depend on the size of `n`. The space complexity is also O(n), where n is the length of the result string, because we need to store the result string.

The code in context looks like this:

```
class Solution:
    def generateTheString(self, n: int) -> str:
        # Check if n is odd using bitwise AND with 1
        if n & 1:
            # Return n 'a' chars if n is odd
            return 'a' * n
        else:
            # Return n-1 'a' chars plus 'b' if n is even
            return 'a' * (n - 1) + 'b'
```

Each part of the code correlates directly with the steps laid out in the algorithm above.

Example Walkthrough

Let's go through the solution approach with a small example where `n = 5` which is an odd number, and `n = 6` which is an even number.

- For `n = 5`:
  1. We check if `n` is odd or even. Here, `5 & 1` equals `1` indicating that `n` is odd.
  2. Since `n` is odd, we return a string that consists of `5` instances of the `'a'` character. Therefore, the resulting string is `'aaaaa'`.
  3. The string `'aaaaa'` contains the character `'a'` exactly 5 times, which is odd, satisfying the condition.
- For `n = 6`:
  1. We check if `n` is odd or even. Here, `6 & 1` equals `0` indicating that `n` is even.
  2. Since `n` is even, we need to construct a string of `6 - 1` instances of `'a'` and add `1` instance of `'b'` at the end. Therefore, the resulting string is `'aaaaab'`.
  3. In the string `'aaaaab'`, the character `'a'` appears `5` times which is an odd number, and the character `'b'` appears `1` time, which is also an odd number. This fulfills the requirement.

Using the methodology provided in the solution, we can quickly determine the correct form of the string for any given `n` by simply checking if it's odd or even and constructing the string accordingly. The simplicity of this approach ensures it is easy to understand and efficient to execute.

Solution Implementation

Python

```
class Solution:
    def generate_the_string(self, n: int) -> str:
        # If n is even, we generate a string with (n-1) 'a' characters and 1 'b' to make the length odd
        # If n is odd, we generate a string with all 'a' characters, as odd-length palindrome is already non-equal
        return 'a' * (n - 1) + 'b' if n % 2 == 0 else 'a' * n
```

Java

```
class Solution {
    // Method to generate a string based on the input integer n
    public String generateTheString(int n) {
        StringBuilder stringBuilder = new StringBuilder(); // Initialize a StringBuilder to construct the string

        if (n % 2 == 1) { // Check if n is odd
            // If n is odd, append 'a' n times to the StringBuilder
            for (int i = 0; i < n; i++) {
                stringBuilder.append("a");
            }
        } else { // If n is even
            // Append 'a' (n-1) times to make the length (n-1), ensuring the final string will have an odd count of 'a'
            for (int i = 0; i < n - 1; i++) {
                stringBuilder.append("a");
            }
            // Append 'b' once to make the total length of the string equal to n
            stringBuilder.append("b");
        }

        // Convert StringBuilder to String and return the result
        return stringBuilder.toString();
    }
}
```

C++

```
class Solution {
public:
    // This function generates a string with 'n' characters
    // such that it has an odd count of at least one unique character
    string generateTheString(int n) {
        // Create a string 'result' with 'n' characters initialized to 'a'
        string result(n, 'a');

        // If 'n' is even, change the first character to 'b'
        // to satisfy the condition of having odd count of unique characters
        if (n % 2 == 0) {
            result[0] = 'b';
        }

        // Return the resultant string
        return result;
    }
};
```

TypeScript

```
// Function to generate a string with a given length 'n' where each character is 'a'
// but if 'n' is even, the first character will be 'b' to ensure that all characters
// are not the same.
function generateTheString(n: number): string {
    // Initialize an array of 'n' elements, all set to 'a'
    const resultArray = Array(n).fill('a');

    // If 'n' is even, set the first element to 'b' to meet the condition
    if (n % 2 === 0) {
        resultArray[0] = 'b';
    }

    // Join the array into a string and return
    return resultArray.join('');
}
```

```
class Solution:
    def generate_the_string(self, n: int) -> str:
        # If n is even, we generate a string with (n-1) 'a' characters and 1 'b' to make the length odd
        # If n is odd, we generate a string with all 'a' characters, as odd-length palindrome is already non-equal
        return 'a' * (n - 1) + 'b' if n % 2 == 0 else 'a' * n
```

Time and Space Complexity

The time complexity of the function `generateTheString` is `O(n)` where `n` is the input to the function. This is because the function involves concatenating `n` or `n-1` characters which takes linear time in the size of the number of characters to concatenate.

The space complexity of the function is also `O(n)` because it generates a string which requires space proportional to the number of characters in the string, which in this case is `n` or `n-1`.