171. Excel Sheet Column Number

Easy ~~Math~~ String Leetcode Link

Problem Description

The problem is akin to translating a specialized numbering system, akin to a base-26 number system, into a decimal (base-10) system. In this system instead of using digits, Excel sheets use uppercase English letters where 'A' corresponds to 1, 'B' to 2, ..., and 'Z' to 26. After 'Z', the sequence continues with combinations of these letters, starting with 'AA', 'AB', etc., similar to how after 9 in decimal comes 10. Our task is to convert an input string `columnTitle` representing an Excel column title to its equivalent column number. For instance, 'A' is 1, 'Z' is 26, 'AA' is 27, 'AB' is 28, and so on.

Intuition

The solution is founded on understanding how the Excel column title translation to a number works. We recognize the pattern that it is almost like a base-26 number system but with a key distinction: there is no zero in the Excel system (it starts from 'A'=1, whereas in base-26, the least digit is '0').

Each character in the string is a "digit" in the base-26 system, with the rightmost character being the least significant "digit". To find the corresponding number, we start processing the string from left to right (the most significant "digit").

For each character, we follow these steps:

1. Convert the letter to its corresponding numerical value, which is its position in the alphabet ('A' is 1, 'B' is 2, ..., 'Z' is 26). We do this by subtracting the ASCII value of 'A' from the ASCII value of the character and then adding 1.
2. Multiply the current total by 26 to make space for the value of the next character (since each position is 26 times more significant than the position to its right).
3. Add the numerical value of the current character to the running total.

We keep a running total and repeat these steps for each character. By the end of the loop, the running total represents the column number of the `columnTitle`.

Solution Approach

The implementation of the solution is fairly straightforward and does not require complex data structures. It applies a simple for-loop and arithmetic operations to achieve the conversion from the Excel column title to a number.

Here's a step-by-step walkthrough of the solution implementation:

1. We initialize a variable `res` to `0`. This will hold the cumulative result as we process each character in the Excel column title.
2. We loop over each character `c` in the `columnTitle` string, processing from left to right.
   - For each character, the algorithm makes use of the `ord()` function to get the ASCII value of `c`, then subtracts the ASCII value of `'A'` to align it with a base-26 system. But since Excel's numbering starts with 1 and not 0 (e.g., 'A' is 1, not 0), we add 1 to the result.
   - Then, we update the result `res` by multiplying it by 26 to make space for the next "digit" coming from `c`. This is similar to shifting digits to the left in base-10 mathematics when you multiply by 10 (`123 * 10` becomes `1230`).
   - Now, we add the value of `c` (now properly converted to a number from 1 to 26) to `res`.
3. Once the loop concludes, `res` contains the final column number that corresponds to the input column title. This value is then returned from the function.

The solution uses a single `for` loop and operates in `O(n)` time complexity, where `n` is the length of the string `columnTitle`, because it reads each character exactly once. The space complexity is `O(1)` since it uses only a fixed amount of extra space (the `res` variable).

Example Walkthrough

Let's walk through an example to illustrate the solution approach using the input `columnTitle = "CAB"`.

1. Initialize `res` to 0. This variable will accumulate our result.
2. We process the first character, 'C':
   - Convert 'C' to its numerical value: `ord('C') - ord('A') + 1 = 3 - 1 + 1 = 3`.
   - Multiply `res` by 26 (since it's the first character, `res` is still 0): `res = 0 * 26 + 3 = 3`.
3. Move to the second character, 'A':
   - Convert 'A' to its numerical value: `ord('A') - ord('A') + 1 = 1 - 1 + 1 = 1`.
   - Multiply `res` by 26 to make space for the 'A' value: `res = 3 * 26 = 78`.
   - Add the value of 'A': `res = 78 + 1 = 79`.
4. Process the third character, 'B':
   - Convert 'B' to its numerical value: `ord('B') - ord('A') + 1 = 2 - 1 + 1 = 2`.
   - Multiply `res` by 26 to accommodate the 'B' value: `res = 79 * 26 = 2054`.
   - Add the value of 'B': `res = 2054 + 2 = 2056`.
5. Now that we've processed all characters, `res` holds the final result (`2056`), which represents the column number of `columnTitle = "CAB"`.

The column title 'CAB' translates to the column number 2056. Each character is weighed by its position's significance, akin to place value in our traditional base-10 number system, only in this case, it follows a base-26 logic without the digit '0'.

Python Solution

```python
class Solution:
    def titleToNumber(self, column_title: str) -> int:
        # Initialize the result to 0
        result = 0

        # Iterate through each character in the column title
        for char in column_title:
            # Convert the character to a number (A=1, B=2, ..., Z=26)
            # and shift the previous result by multiplying by 26 (since it's base 26)
            result = result * 26 + (ord(char) - ord('A') + 1)

        # Return the final numerical result
        return result

```

Java Solution

```java
class Solution {

    /**
     * This method converts a column title from a spreadsheet to its corresponding number.
     *
     * @param columnTitle A string representing the column title to be converted.
     * @return The numeric value of the column title.
     */
    public int titleToNumber(String columnTitle) {
        int result = 0; // Initialize result to 0

        // Iterate over each character in the columnTitle string
        for (char ch : columnTitle.toCharArray()) {
            // Convert character to corresponding number ('A' -> 1, 'B' -> 2, ..., 'Z' -> 26)
            int numericValue = ch - 'A' + 1;

            // Update the result by shifting it 26 places (base 26 number system) and adding the numeric value of the current charact
            result = result * 26 + numericValue;
        }

        // Return the computed numeric value of the column title
        return result;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to convert a column title to its corresponding number
    int titleToNumber(string columnTitle) {
        int result = 0; // Initialize result to zero to store the column number

        // Iterate over each character of the column title
        for (char c : columnTitle) {
            // Convert char to its position in the alphabet and add it to the result
            // 'A' maps to 1, 'B' maps to 2, ..., so 'A' - 'A' + 1 starts at 1.
            // Multiply the current result by 26 for base-26 calculation.
            result = result * 26 + (c - 'A' + 1);
        }
        return result; // Return the resulting column number
    }
};

```

Typescript Solution

```typescript
/**
 * Converts a column title from an Excel sheet (e.g., "A", "B", ..., "Z", "AA", etc.)
 * to its corresponding column number.
 *
 * @param {string} columnTitle - The title of the column in Excel sheet notation.
 * @return {number} The column number corresponding to the column title.
 */
function titleToNumber(columnTitle: string): number {
    // Initialize the result variable to 0. This will be the final column number.
    let columnNumber: number = 0;

    // Loop through each character in the columnTitle string.
    for (let character of columnTitle) {
        // Convert the current character to its corresponding column number
        // taking 'A' as 1, 'B' as 2, and so on by subtracting 64 from the ASCII code
        // (since 'A' is ASCII 65, 'B' is 66, etc.).
        // Multiply the current result by 26 (number of letters in the alphabet)
        // to "shift" its value to the left before adding the new value.
        columnNumber = columnNumber * 26 + character.charCodeAt(0) - 64;
    }

    // Return the final computed column number.
    return columnNumber;
}

```

Time and Space Complexity

The time complexity of the given code is `O(n)`, where `n` is the length of the input string `columnTitle`. This is because the code iterates through each character in the input string exactly once, and for each character, it performs a constant-time operation (calculating the value of `res` based on ASCII values).

The space complexity of the code is `O(1)`. This is because the space required to store the intermediate results and the final output does not depend on the size of the input string. The same amount of space (for variables `res` and `c`) is used regardless of the length of `columnTitle`.