168. Excel Sheet Column Title

Easy Math String

Leetcode Link

Problem Description

The given problem requires us to convert an integer, `columnNumber`, into a string that represents the corresponding column title as seen in an Excel sheet. In Excel, columns are labeled alphabetically from A to Z, then continuing with AA, AB, etc., after Z. This sequence creates a pattern similar to base-26 numeral system, except that it doesn't contain a zero and it starts from A (1) rather than zero (0).

In this pattern, `A` corresponds to 1, `B` to 2, and so on up to `Z` which corresponds to 26. Then, the sequence continues with two-letter combinations such as `AA` for 27, `AB` for 28, and so on. Unlike our decimal system, where the digit '0' represents zero and serves as a place-holder, there is no 'zero' character in the Excel column titles.

The challenge is to convert the given `columnNumber` into this special alphabet-based naming system. This includes figuring out how many times we can divide the number by 26 to advance letters and how to correctly wrap around after reaching the end of the alphabet.

Intuition

To solve this problem, we can use a method similar to converting a number from base-10 to base-26, with the twist that Excel's column naming scheme is 1-indexed, not 0-indexed. This means the sequence goes 1, 2, ..., 25, 26 (A, B, ..., Y, Z) and then wraps to 27 (AA), rather than the standard 0, 1, ..., 24, 25 (A, B, ..., Y, Z) and then wrap to 26 (AA).

Here's the approach step-by-step:

1. Initialize an empty list `res` to accumulate the characters (from the least important digit to the most important one).
2. Enter a loop that will continue as long as `columnNumber` is greater than zero. This iterative process will divide the `columnNumber` until it cannot be divided anymore, indicating completion.
3. Inside the loop, since the numbering is 1-indexed, we decrease `columnNumber` by 1 to convert it to a 0-indexed number.
4. Obtain the remainder of the current `columnNumber` when divided by 26. This modulo operation gives us an index corresponding to the letters from A to Z.
5. Convert this index into a character by adding it to the ASCII value of 'A' and append the resulting character to the list `res`.
6. Prepare for the next iteration by dividing `columnNumber` by 26 since we have already handled one "digit" of our base-26 number.
7. Once the loop is done, we reverse `res` because we have been appending the less significant characters first, and then we join the characters together to form a string.
8. Return the resulting string, which is the column title that corresponds to the original `columnNumber`.

By repeatedly taking modulus and division, we handle the number 'column by column', just like an actual base conversion, but always taking into consideration the offset caused by the lack of 'zero' in the Excel system. This approach keeps us within Excel's 1-indexed column naming system.

Solution Approach

The implementation of the solution involves a straightforward approach that resembles a base-conversion algorithm. Here's a detailed walk-through of the implementation steps.

1. Create a class `Solution` with a method `convertToTitle`, which takes an integer `columnNumber` as its parameter and returns a string.
2. Inside the `convertToTitle` method, initialize an empty list `res` which will store the characters of our result in reverse order (the least significant 'digit' first).
3. Use a `while` loop to process the `columnNumber` as long as it's greater than 0. This loop will be used to iteratively break down the number from base-10 to the equivalent 'base-26'.
4. As Excel columns start from 1 (A) instead of 0, we subtract 1 from `columnNumber` each iteration to properly align our values with the index for character mapping.
5. Calculate the remainder with `columnNumber % 26` to determine which character in the range A-Z corresponds to the current least significant 'digit' of the number.
6. Use `ord('A')` to get the ASCII value of 'A', then add the remainder to it. This will give us the ASCII value of the character we want. Use `chr()` to get the character from this ASCII value and append it to our list `res`.
7. Prepare for the next iteration by performing integer division `columnNumber //= 26` to reduce our `columnNumber` for the subsequent 'digit'.
8. Once the `while` loop concludes, we have all the characters of our column title in reverse order. We use `[::1]` to reverse the list `res` back to the correct order.
9. Finally, we join the list of characters with `join()` into a string, which represents the column title equivalent to the initial `columnNumber`.
10. The method `convertToTitle` returns this string, completing the conversion process.

This solution doesn't use any complex libraries or specific data structures beyond a basic list and standard Python string operations. The pattern here mirrors a familiar concept of converting between numerical systems, accommodating the unique Excel spreadsheet 1-indexed base. It is an elegant and efficient solution that works within the confines of Python's simple data manipulation capabilities.

Example Walkthrough

Let's walk through an example to illustrate the solution approach by converting the `columnNumber` 705 to its corresponding Excel column title.

1. We begin with `columnNumber` = 705. Our result `res` is currently an empty list.
2. Enter the `while` loop since `columnNumber` > 0.
3. Subtract 1 from `columnNumber` to make it 0-indexed: `columnNumber` = 705 - 1 = 704.
4. Calculate the remainder of `columnNumber` divided by 26 to determine the character for this 'digit': `remainder` = `columnNumber` % 26 = 704 % 26 = 24.
5. Convert this remainder to a character by finding the ASCII value of 'A', adding the remainder to it, and then converting back to a character with `chr()`: `character` = `chr(ord('A') + remainder)` = chr(65 + 24) = 'Y'. Append 'Y' to `res`.
6. Update `columnNumber` for the next iteration: `columnNumber` //= 26 = 704 // 26 = 27.
7. As `columnNumber` is still greater than 0, we repeat steps 3-6:
   - `columnNumber` = 27 - 1 = 26
   - remainder = 26 % 26 = 0
   - 'A' + 0 = 'A', append 'A' to `res`. Now, `res` = ['Y', 'A'].
   - `columnNumber` = 26 // 26 = 1
8. We loop again:
   - `columnNumber` = 1 - 1 = 0
   - remainder = 0 % 26 = 0
   - 'A' + 0 = 'A', append 'A' to `res`. Now, `res` = ['Y', 'A', 'A'].
   - `columnNumber` = 0 // 26 = 0
9. The loop ends because `columnNumber` is now 0.
10. Reverse `res` and join the characters to form a string: `res.reverse()` = ['A', 'A', 'Y'], `result` = "".join(res) = "AAY".
11. Return the result, "AAY", which is the corresponding Excel column title for column number 705.

We have converted the input `columnNumber` 705 into the Excel column title "AAY" following the solution steps detailed above.

Python Solution

```python
class Solution:
    def convertToTitle(self, column_number: int) -> str:
        # Initialize an empty list to store the characters that will
        # represent the column title in reverse order.
        title_chars = []

        # Continue to iterate as long as the column_number is greater than zero.
        while column_number:
            # Adjust column_number to zero-indexed for modulo operation.
            column_number -= 1

            # Calculate the character that corresponds to the current modulo of
            # column_number and 'A'. Append the character to the title_chars list.
            title_chars.append(chr(ord('A') + column_number % 26))

            # Update column_number by dividing it by 26, for the next iteration
            # as we move to the next most significant digit in the title.
            column_number //= 26

        # Since the title_chars list was constructed backwards,
        # reverse it and join the characters into a single string
        # to form the column title, then return the result.
        return ''.join(title_chars[::-1])

```

Java Solution

```java
class Solution {

    // This method converts a given column number to its corresponding Excel column title.
    public String convertToTitle(int columnNumber) {
        // StringBuilder to build the result in reverse order
        StringBuilder result = new StringBuilder();

        // Loop until the columnNumber is 0
        while (columnNumber > 0) {
            // Decrement columnNumber to adjust for 1-indexing in Excel columns
            columnNumber--;

            // Calculate the character to append using the remainder
            // and concatenate it to our result string
            result.append((char) ('A' + columnNumber % 26));

            // Update columnNumber by dividing by 26 for the next iteration
            columnNumber /= 26;
        }

        // Reverse the result since we've been appending characters from least significant (right most) to most significant
        return result.reverse().toString();
    }
}

```

C++ Solution

```cpp
#include <string>
#include <algorithm> // for std::reverse

// This function converts a given column number to a title as it would appear in an Excel sheet.
// For example, 1 becomes 'A', 27 becomes 'AA', and so on.
std::string convertToTitle(int columnNumber) {
    // Initialize a string to hold the characters of the final result.
    std::string title;

    // Continue the loop until the columnNumber is reduced to 0.
    while (columnNumber > 0) {
        // Decrement to map the number to a zero-indexed scale where A=0, B=1, ..., Z=25.
        columnNumber--;

        // Get the remainder when columnNumber is divided by 26 to find
        // the character that corresponds to the current place value.
        int remainder = columnNumber % 26;

        // Convert the remainder to the corresponding uppercase character
        // by adding it to 'A' and then prepend it to the 'title' string.
        // The ASCII value of 'A' is 65, but adding remainder to 'A' gives the correct character.
        title.insert(title.begin(), 'A' + remainder);

        // Go to the next place value by dividing columnNumber by the next iteration.
        columnNumber /= 26;
    }

    // The characters are inserted in reverse order, so we must reverse the final string.
    // This can be done during insertion as in the line above or by reversing the string here.
    // std::reverse(title.begin(), title.end());

    // Return the column title in the correct format.
    return title;
}

```

Typescript Solution

```typescript
// This function converts a given column number to a title as it would appear in an Excel sheet.
// For example, 1 becomes 'A', 27 becomes 'AA' and so on.
function convertToTitle(columnNumber: number): string {
    // Initialize an array to hold the characters of the final result.
    let titleCharacters: string[] = [];

    // Continue the loop until the columnNumber is reduced to 0.
    while (columnNumber > 0) {
        // Decrement to map the number to a zero-indexed scale where A=0, B=1, ..., Z=25.
        columnNumber--;

        // Get the remainder when columnNumber is divided by 26 to find
        // the character that corresponds to the current place value.
        let remainder: number = columnNumber % 26;

        // Convert the remainder to the corresponding ASCII uppercase character
        // and unshift it to the start of the titleCharacters array.
        // ASCII value of 'A' is 65, so adding remainder to it gives the correct character.
        titleCharacters.unshift(String.fromCharCode(remainder + 65));

        // Go to the next place value by dividing columnNumber by 26 before the next iteration.
        columnNumber = Math.floor(columnNumber / 26);
    }

    // Join the titleCharacters array into a string to get the column title in the correct format.
    return titleCharacters.join("");
}

```

Time and Space Complexity

The time complexity of the function is $O(\log_{26}(n))$, where n is the `columnNumber`. This is because the loop divides the `columnNumber` by 26 in each iteration, effectively converting the number from base 10 to base 26. The number of iterations is proportional to the number of times n can be divided by 26 before it becomes 0, which is essentially the number of digits in the base-26 representation of n.

The space complexity of the function is $O(\log_{26}(n))$ as well. The reason behind this is that space is allocated for the `res` list, which stores each character corresponding to a digit in the base-26 number. The length of the `res` list is equal to the number of digits in the base-26 representation of n, which results from the same logarithmic relationship as in the time complexity analysis.