504. Base 7

Easy Math

Problem Description

The problem requires writing a function that converts an integer `num` to its base 7 representation. In a base 7 number system, each digit in a number represents an increasing power of 7, starting from the rightmost digit (7^0, 7^1, 7^2, and so forth). The digits can only be from 0 to 6, inclusive. The output of the function should be a string representing the number in base 7.

For example, if the input is `100`, the output should be `'202'` because `100` equals `2 * 7^2 + 0 * 7^1 + 2 * 7^0`.

Edge cases to consider include when the input number is `0`, where the base 7 representation is also `'0'`, and when the number is negative, in which case the base 7 representation should include a negative sign (`-`) before the digits.

Intuition

To arrive at a solution, we break down the problem and approach it step by step:

1. **Handling Zero**: If the input number is `0`, then the base 7 representation is straightforwardly `'0'`.
2. **Handling Negatives**: If the number is negative, we work with its absolute value and prepend a `'-'` to the result after the base 7 conversion.
3. **Conversion to Base 7**: For a positive number, we can iteratively divide the number by `7` and keep track of the remainders. Each remainder gives us the base 7 digit starting from the least significant digit to the most significant digit.
4. **Reversal and Joining Digits**: Since the remainders are obtained in reverse order (least significant digit first), we reverse the sequence of remainders and concatenate them to form the base 7 string. This is where we utilize the array structure to store digits and the `join` and `[::-1]` operations to form the final string.

Solution Approach

The solution utilizes a simple algorithm that converts an integer from base 10 to base 7. Here's a step-by-step walkthrough of the implementation corresponding to the provided solution code:

1. **Check for Zero**: The code checks whether the input `num` is `0`. If it is, it immediately returns the string `'0'` because zero is represented the same in any base.
2. **Handling Negative Numbers**: If `num` is negative, we convert it to a positive number using `-num` and perform a recursive call to `convertToBase7`. We prepend a `'-'` to the result of the recursive call to account for the negative sign. This ensures that the base 7 conversion logic only deals with positive numbers.
3. **Conversion Loop**: A while loop is used to perform the base 7 conversion. Inside the loop, the remainder of `num` divided by `7` is calculated using `num % 7`. This gives us the next digit of the base 7 number (from right to left).
4. **Storing Digits**: The remainders/digits are stored in an array called `ans`. Each digit is converted to a string before appending it to the array. This is done because the output needs to be a string, and it's more efficient to build an array of strings and then join them together than to perform string concatenation in each iteration.
5. **Updating the Number**: The number `num` is then updated by performing integer division by `7` using `num //= 7`. Integer division ensures that we get the next set of digits for the remaining number without decimals.
6. **Reversing and Joining**: Once the number `num` is reduced to `0`, all of its base 7 digits are contained in `ans` array but in reverse order. The `ans[::-1]` expression reverses the array. To form the final base 7 string, the code joins the elements of this reversed array using `''.join(ans[::-1])`.

The concepts of recursion, string manipulation, and number base conversion are applied in this solution. The problem is solved without using any complex data structures or patterns, focusing on simple loop iteration and string array handling.

Example Walkthrough

Let's consider the number `58` to illustrate the solution approach. We want to convert `58` into its base 7 representation.

1. **Check for Zero**: Since `58` is not `0`, this step is skipped.
2. **Handling Negative Numbers**: `58` is positive, so there is no need to handle negatives. We proceed directly with the base 7 conversion.
3. **Conversion Loop**: We start by dividing `58` by `7`. The remainder is `58 % 7 = 2`. This is the least significant digit of the base 7 representation.
4. **Storing Digits**: The remainder `2` is added to the array `ans`. It's stored as a string `'2'`, so `ans = ['2']` now.
5. **Updating the Number**: Next, we update `58` by doing integer division by `7`: `58 // 7 = 8`. We repeat the loop with this new number `8`.
   - Divide `8` by `7` again. The remainder is `8 % 7 = 1`. Add `'1'` to `ans`, which becomes `ans = ['2', '1']`.
   - Update `8` by doing integer division by `7`: `8 // 7 = 1`.

   Now we have `1`, which is still positive, so we continue.
   - Divide `1` by `7`. The remainder is `1 % 7 = 1`. Add `'1'` to `ans`, which now becomes `ans = ['2', '1', '1']`.
   - Update `1` by doing integer division by `7`: `1 // 7 = 0`. Now that `num` is `0`, the conversion loop ends.
6. **Reversing and Joining**: The array `ans` currently contains `['2', '1', '1']`. This is the reverse of what we want, so we need to reverse it back to get the correct order, which gives us `['1', '1', '2']`.

Finally, we join these digits together to get the base 7 representation of `58`: `''.join(['1', '1', '2'])` which results in `'112'`.

So, the base 7 representation of the decimal number `58` is `'112'`.

Solution Implementation

Python

```python
class Solution:
    def convertToBase7(self, num: int) -> str:
        # Handle the special case where num is zero
        if num == 0:
            return '0'

        # Check if the number is negative, if so, convert the number to positive
        # and prepend the negative sign after the base 7 conversion.
        if num < 0:
            return '-' + self.convertToBase7(-num)

        # Initialize a list to hold the digits of the base 7 representation.
        base7_digits = []

        # Convert the number to base 7.
        while num > 0:
            # Append the remainder (base 7 digit) to the list.
            base7_digits.append(str(num % 7))
            # Update num by dividing it by 7.
            num //= 7

        # Join the base 7 digits in reverse order to form the final base 7 number.
        # Since we append digits from least significant to most significant,
        # we need to reverse the list before joining into a string.
        return ''.join(reversed(base7_digits))

# Example usage:
# solution = Solution()
# base7_number = solution.convertToBase7(100)
# print(base7_number) # Output would be the base 7 representation of 100
```

Java

```java
class Solution {

    /**
     * This method converts an integer to its base 7 representation.
     *
     * @param num The integer to be converted to base 7.
     * @return A string representing the base 7 equivalent of the input number.
     */
    public String convertToBase7(int num) {
        // Base case: If the input is zero, return "0" as its base 7 representation.
        if (num == 0) {
            return "0";
        }

        // If the number is negative, return the negative sign concatenated
        // with the base 7 representation of the positive counterpart.
        if (num < 0) {
            return "-" + convertToBase7(-num);
        }

        // Create a StringBuilder to construct the base 7 representation
        StringBuilder builder = new StringBuilder();

        // Continue the process until we have completely converted the number to base 7
        while (num != 0) {
            // Append the remainder (base 7 digit) to the StringBuilder
            builder.append(num % 7);

            // Divide the number by 7 to get to the next digit
            num /= 7;
        }

        // Reverse the string since the construction was from least significant digit to most
        // significant.
        return builder.reverse().toString();
    }
}
```

C++

```cpp
class Solution {
public:
    // Function to convert an integer to its base 7 string representation
    string convertToBase7(int num) {
        // Handle the base case where number is 0
        if (num == 0) {
            return "0";
        }

        // If the number is negative, recursively call the function on its absolute value
        // and append a minus sign to the result.
        if (num < 0) {
            return "-" + convertToBase7(-num);
        }

        // Initialize an empty string to hold the base 7 representation
        string base7Representation = "";

        // Loop to convert the number to base 7
        while (num > 0) {
            // Prepend the remainder of the division by 7 to the string
            base7Representation = to_string(num % 7) + base7Representation;
            // Divide the number by 7 to get the next digit
            num /= 7;
        }

        // Return the resulting base 7 string
        return base7Representation;
    }
};
```

TypeScript

```typescript
/**
 * Converts an integer to its base-7 string representation.
 * @param {number} num - The integer to be converted.
 * @return {string} The base-7 representation of the given number.
 */
function convertToBase7(num: number): string {
    // If the number is zero, return the string "0".
    if (num === 0) {
        return '0';
    }

    // Initialize the result string.
    let result = '';
    // Determine if the number is negative.
    const isNegative = num < 0;

    // If the number is negative, convert it to positive.
    if (isNegative) {
        num = -num;
    }

    // Loop to divide the number by 7 and take the remainder until num is 0.
    while (num !== 0) {
        // Calculate the remainder of num divided by 7.
        const remainder = num % 7;
        // Prepend the remainder to the result string.
        result = remainder.toString() + result;
        // Update num to be the quotient of division by 7.
        num = (num - remainder) / 7;
    }

    // If the original number was negative, add the negative sign to the result.
    return isNegative ? '-' + result : result;
}

class Solution:
    def convertToBase7(self, num: int) -> str:
        # Handle the special case where num is zero
        if num == 0:
            return '0'

        # Check if the number is negative, if so, convert the number to positive
        # and prepend the negative sign after the base 7 conversion.
        if num < 0:
            return '-' + self.convertToBase7(-num)

        # Initialize a list to hold the digits of the base 7 representation.
        base7_digits = []

        # Convert the number to base 7.
        while num > 0:
            # Append the remainder (base 7 digit) to the list.
            base7_digits.append(str(num % 7))
            # Update num by dividing it by 7.
            num //= 7

        # Join the base 7 digits in reverse order to form the final base 7 number.
        # Since we append digits from least significant to most significant,
        # we need to reverse the list before joining into a string.
        return ''.join(reversed(base7_digits))

# Example usage:
# solution = Solution()
# base7_number = solution.convertToBase7(100)
# print(base7_number) # Output would be the base 7 representation of 100
```

Time and Space Complexity

Time Complexity

The time complexity of the `convertToBase7` function mainly depends on the number of digit extractions performed in the loop. Since each iteration of the loop processes one digit of the number in base 7, the number of iterations is $O(\log_7(num))$. However, when evaluating time complexity in terms of base 2 (which is common since our input size is commonly measured in bits), it's safe to say that it is approximately $O(\log(num))$ because the base of the logarithm can be changed with a constant multiplier due to the logarithm change of base property ($\log_b(a) = \log_c(a) / \log_c(b)$), and constants are discarded in big O notation.

The methods `append` and `str` have constant time complexity $O(1)$, and `join` is $O(n)$ where n is the length of the list `ans`. But since the length of `ans` contributes to the overall time it takes to construct the base 7 representation, this doesn't add more than a constant multiple to the $\log(num)$ term.

Therefore, the time complexity of the given code is $O(\log(num))$.

Space Complexity

Space complexity includes the extra space required besides the input. In this case, it is determined by the space needed to hold the `ans` list and the characters it contains.

For a decimal number `num`, the number of digits `d` in its base 7 representation will be roughly `log_7(num)`, which is about $O(\log(num))$. Each digit converts to a string element in the `ans` list, so the space complexity is directly proportional to the number of digits in the base 7 representation.

So, the space complexity of the given code is also $O(\log(num))$ as it is solely dependent on the length of the `ans` list.