1837. Sum of Digits in Base K

Easy Math

Problem Description

The problem asks for converting an integer `n` from base `10` to a given base `k` and then to find the sum of the digits of the new number in base `k`. It's important to note that after conversion, each digit of the base `k` number should be treated as an individual base `10` number. Finally, the sum of these base `10` values of the digits should be returned also in base `10`.

To illustrate this with an example, if `n` is `34` and `k` is `6`, the base `6` representation of `34` would be `54` (`5*6^1 + 4*6^0`). The sum needed would be the sum of the digits `5` and `4` in base `10`, which equals `9`.

Intuition

The solution follows a simple mathematical approach, consisting of repeatedly dividing the given number `n` by the base `k` and taking the remainder as the next base `k` digit. This is a common mathematical strategy to convert numbers from base `10` to another base.

The core intuition is that in each division step, `n % k` gives us the digit in the ones place of the base `k` number, and `n // k` reduces `n` to remove that digit. We keep track of the sum of these digits as we find them by adding `n % k` to `ans` at each step, which is initialized as `0`. This process continues until `n` is reduced to `0`, at which point we've looked at each digit of the number in base `k`, and `ans` holds their sum.

By treating each digit as a base `10` number immediately upon discovery, we avoid the need to first fully convert `n` to base `k` as a separate step before summing the digits. We're effectively converting and summing in one pass, which is efficient and straightforward.

Solution Approach

The implementation of the solution makes use of a simple loop and arithmetic operations to achieve the conversion and digit summation. No complex data structures or patterns are necessary, which makes the algorithm easy to understand and efficient in terms of both time and space complexity.

Here's an explanation of the steps taken in the given Python code:

1. We initialize `ans` to `0`, which will be used to keep track of the cumulative sum of the base `k` digits.
2. We start a `while` loop that will run as long as `n` is not `0`. Within each iteration of the loop, we are dealing with two tasks: converting to base `k` and summing up the digits.
3. The expression `n % k` is evaluated, which will give us the least significant digit in the base `k` representation of our number. For example, if `n` is `34` and `k` is `6`, in the first iteration, we will get `4` because `34 % 6` equals `4`.
4. We add this digit to `ans`, effectively summing the digits as we generate them in the base `k` representation.
5. Next, we need to update `n` for the next iteration. We do this by floor-dividing `n` by `k` using the expression `n //= k`. Floor division (`//`) gives us the quotient when `n` is divided by `k`, which effectively shifts our base `k` number to the right, discarding the digit we just added to `ans`.
6. The loop continues until `n` is reduced to `0`, meaning we have processed all digits in the base `k` number.

After the loop exits, the final result, which is the sum of the digits in base `k`, is stored in `ans`. This value is then returned.

This approach is grounded in arithmetic and the properties of number base conversion. It is efficient, as it does not require any additional memory beyond a few integer variables, and all operations are basic arithmetic which can be performed in constant time relative to the number of digits in the base `k` representation.

Example Walkthrough

Let's take `n = 29` and `k = 7` as a small example to illustrate the solution approach. We are tasked with converting the integer `n` from base `10` to base `k` and then finding the sum of the digits of the resulting number when each digit is considered as a separate base `10` number.

1. We start by initializing `ans` to `0`. This will keep track of the cumulative sum of the base `k` digits.
2. Since `n` is not `0`, we enter the while loop.
3. During the first iteration, we calculate `n % k`. For our example, `29 % 7` is `1`. The number `1` is the least significant digit in the base `7` representation of the number `29`.
4. We then add this digit to `ans`, so `ans` is now `1`.
5. We prepare for the next iteration by updating `n` with `n // k`. For our example, `29 // 7` is `4`. So now `n` becomes `4`.
6. The while loop executes again because `n` is not `0`.
7. During the second iteration, `n % k` gives `4 % 7`, which is `4`. This is the next digit in the base `7` representation, which is also the most significant digit since `n` is now less than `k`.
8. We add the `4` to `ans`, resulting in `ans` being `1 + 4` or `5`.
9. We update `n` with `n // k` again. This time, `4 // 7` is `0`, since `4` is less than `7`.
10. The while loop terminates because `n` is now `0`.

The final result stored in `ans` is `5`, which is the sum of the digits `4` and `1` of the number `29` when converted to base `7`. This sum `5` is returned as the result.

Solution Implementation

Python

```python
class Solution:
    def sumBase(self, n: int, k: int) -> int:
        """
        Calculate the sum of digits of a number 'n' represented in base 'k'.

        Parameters:
        n (int): The integer number to be converted.
        k (int): The base to which the number 'n' is to be converted.

        Returns:
        int: The sum of the digits of the number 'n' in base 'k'.
        """

        # Initialize the variable to store the sum of digits
        digit_sum = 0

        # Continue looping until the number becomes 0
        while n > 0:
            # Add the last digit of 'n' in base 'k' to 'digit_sum'
            digit_sum += n % k

            # Remove the last digit by dividing 'n' by the base 'k'
            n //= k

        # Return the sum of the digits
        return digit_sum
```

Java

```java
class Solution {
    // Method to calculate the sum of digits of 'n' when represented in base 'k'
    public int sumBase(int n, int k) {
        // Initialize the result variable to store the sum of digits
        int result = 0;

        // Continue the process until 'n' becomes 0
        while (n != 0) {
            // Add the last digit of 'n' in base 'k' to the result
            result += n % k;  // 'n % k' gives the last digit when 'n' is represented in base 'k'

            // Divide 'n' by 'k' to remove the last digit, reducing 'n'
            n /= k;  // 'n' is now equal to 'n' without its last digit in base 'k'
        }

        // Return the sum of digits of 'n' in base 'k'
        return result;
    }
}
```

C++

```cpp
class Solution {
public:
    // Function to calculate the sum of digits of the number 'n' when it is represented in base 'k'.
    int sumBase(int n, int k) {
        int sum = 0; // Initialize the sum of digits to 0
        while (n > 0) { // Continue until the number becomes 0
            sum += n % k; // Add the last digit of 'n' in base 'k' to sum
            n /= k; // Divide 'n' by 'k' to remove the last digit
        }
        return sum; // Return the calculated sum of digits in base 'k'
    }
};
```

TypeScript

```typescript
// This function calculates the sum of the digits of a number n when represented in base k.
function sumBase(n: number, k: number): number {
    let sum = 0; // Initialize sum to store the sum of the digits in base k.

    // Loop continues until n is reduced to 0.
    while (n) {
        sum += n % k; // Add the remainder of n divided by k to sum, this is the rightmost digit in base k.
        n = Math.floor(n / k); // Update n to be the quotient of n divided by k, removing the rightmost digit in base k.
    }

    // The function returns the sum of digits of n in base k.
    return sum;
}
```


```python
class Solution:
    def sumBase(self, n: int, k: int) -> int:
        """
        Calculate the sum of digits of a number 'n' represented in base 'k'.

        Parameters:
        n (int): The integer number to be converted.
        k (int): The base to which the number 'n' is to be converted.

        Returns:
        int: The sum of the digits of the number 'n' in base 'k'.
        """

        # Initialize the variable to store the sum of digits
        digit_sum = 0

        # Continue looping until the number becomes 0
        while n > 0:
            # Add the last digit of 'n' in base 'k' to 'digit_sum'
            digit_sum += n % k

            # Remove the last digit by dividing 'n' by the base 'k'
            n //= k

        # Return the sum of the digits
        return digit_sum
```


Time and Space Complexity

The time complexity of the provided code is `O(log_k(n))`. This result is due to the fact that in each iteration, the number `n` is divided by the base `k`, which decreases `n` exponentially until it reaches 0. The number of iterations required is proportional to the number of digits of `n` in base `k`, which is why the logarithm comes into play.

As for the space complexity, it is `O(1)` which means it is constant. The reason for this is that the variables `ans` and `n` are being reused and updated with each iteration, and no additional space is required that scales with the input size.