1134. Armstrong Number

Easy Math

Problem Description

An Armstrong number (also known as a narcissistic number) is a number that is the sum of its own digits each raised to the power of the number of digits. In this problem, you are given an integer `n`, and the task is to determine whether `n` is an Armstrong number.

To explain it with an example, let's take the number 153 which is a 3-digit number. If we raise each digit to the power of 3 (since it's a 3-digit number) and sum them, we get: `1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153`, which is equal to the original number. Therefore, 153 is an Armstrong number.

This definition generalizes to any number of digits. For any given `n`, we wish to find out whether the number is equal to the sum of its own digits each raised to the power of `k`, where `k` is the number of digits in `n`.

Intuition

The intuition behind the solution follows from the definition of an Armstrong number. We need to:

1. Determine the number of digits `k` in `n`, which can be done by converting the number `n` to a string and finding its length.
2. Sum up each digit raised to the `k`th power. This can be done by repeatedly taking the last digit of `n` using the modulo operator `% 10`, raising it to the `k`th power, and adding it to a sum.
3. After processing a digit, we can remove it from `n` by using integer division by 10 (`// 10`) to shift all the digits to the right.
4. Continue this process until all digits have been processed (i.e., until `n` becomes 0).
5. Finally, compare the obtained sum with the original number `n`, and if they are equal, `n` is an Armstrong number; otherwise, it is not.

This approach systematically deconstructs the number `n` to access its individual digits while keeping track of the sum of the digits each raised to the `k`th power. If at the end, this sum is equal to `n`, then the number is confirmed to be an Armstrong number, and the function should return `true`. Otherwise, it should return `false`.

Solution Approach

The implementation of the solution for determining if a number is an Armstrong number involves a straightforward approach that iterates through the digits of the number while computing a sum that follows the rule for Armstrong numbers.

Here is the step by step breakdown of the algorithm:

1. First, find the number of digits `k` in our number `n`. This is done by converting `n` to a string and getting the length of that string:

```
k = len(str(n))
```

2. Initialize two variables: one for the sum (`s`) which we will use to accumulate the k-th powers of the digits, and a copy of `n` (`x`) which we will use to manipulate and access each digit without altering the original number `n`:

```
s, x = 0, n
```

3. Use a while loop to iterate over each digit in the number. The loop continues as long as `x` is not zero:

```
while x:
```

4. Inside the loop, extract the last digit of `x` using `% 10`, raise it to the power of `k`, and add the result to the sum `s`:

```
s += (x % 10) ** k
```

5. Then remove the last digit from `x` by dividing `x` by 10 and keeping the integer part only (effectively shifting the digits to the right):

```
x //= 10
```

6. After finishing the loop, we check if the calculated sum `s` is equal to the original number `n`. If they match, it means that `n` is indeed an Armstrong number:

```
return s == n
```

The data structures used in this approach are quite simple - just integers for keeping track of the sum, the number of digits, and the working copy of the original number.

The pattern followed here is an iterative approach that decomposes the number digit by digit to perform the necessary arithmetic operations, a common technique when dealing with numbers in programming.

This solution does not require any complex algorithms or data structures and efficiently arrives at the answer by harnessing elementary arithmetic and control flow constructs in Python.

Example Walkthrough

Let us use the number 371 as an example to illustrate the solution approach.

1. First, we find the number of digits `k` in our number `n`, which is `371`. We convert `371` to a string and get the length of that string:

```
k = len(str(371))  # k is 3 because there are 3 digits in the number
```

2. We initialize a variable for the sum `s` at 0 to accumulate the k-th powers of the digits, and we make a copy of `n` (`x`) to manipulate without altering the original number `371`:

```
s, x = 0, 371
```

3. We use a while loop to iterate over each digit in the number. The loop continues as long as `x` (which begins as 371) is not zero:

```
while x:  # True initially because x is 371
```

4. Inside the loop, we extract the last digit of `x` (which is 1 in the first iteration) using `% 10`, raise it to the power of `k` (which is 3), and add the result to the sum `s`:

```
s += (x % 10) ** k  # Adds 1**3 to s, so s becomes 1
```

5. We remove the last digit from `x` by dividing `x` by 10 and keeping the integer part only, which shifts all digits to the right:

```
x //= 10  # x becomes 37 after the first iteration
```

6. The loop continues with `x` now being 37, and in the second iteration, the last digit 7 (`37 % 10`) will be raised to the power of 3 and added to `s`, resulting in `s` being `1 + 7**3` which is `1 + 343` making `s` now 344.
7. With `x` updated to 3 (`37 // 10`), the loop goes for one more iteration with 3 raised to the power of 3 and added to `s`, resulting in `344 + 3**3`, which equals `344 + 27` making `s` 371.
8. After finishing the loop (since `x` is now 0), we check if the calculated sum `s` (371) is equal to the original number `n` (371). Since they match, it means that `371` is indeed an Armstrong number:

```
return s == n  # returns True, as 371 is indeed an Armstrong number
```

By following these steps, we have checked that 371 is an Armstrong number using the method described in the solution approach. The while loop terminated after processing all digits of number 371, raising each digit to the power of the number of digits, summing them, and comparing the result with the original number.

Solution Implementation

Python

```python
class Solution:

    def is_armstrong(self, n: int) -> bool:
        # Convert the number to a string and calculate the length to determine k
        k = len(str(n))

        # Initialize the sum and a temporary variable with the original number
        sum_of_powers, temp_number = 0, n

        # Loop through each digit of the number
        while temp_number > 0:
            # Get the last digit and raise it to the power of k
            last_digit = temp_number % 10
            sum_of_powers += last_digit ** k

            # Remove the last digit from temp_number
            temp_number //= 10

        # Check if the sum of the digits to the power of k is equal to the original number
        return sum_of_powers == n
```

Java

```java
class Solution {
    public boolean isArmstrong(int number) {
        // Calculate the number of digits in 'number' by converting it to a string and getting its length
        int numOfDigits = String.valueOf(number).length();

        // Initialize sum to store the sum of digits raised to the power of 'numOfDigits'
        int sum = 0;

        // Temporary variable to hold the current number as we compute the Armstrong sum
        int tempNumber = number;

        // Loop to calculate the sum of each digit raised to the power numOfDigits
        while (tempNumber > 0) {
            // Get the last digit of tempNumber
            int digit = tempNumber % 10;

            // Add the digit raised to the power of 'numOfDigits' to the sum
            sum += Math.pow(digit, numOfDigits);

            // Remove the last digit from tempNumber
            tempNumber /= 10;
        }

        // Check if the calculated sum is equal to the original number
        return sum == number;
    }
}
```

C++

```cpp
#include <cmath>        // Include cmath for the pow function
#include <string>       // Include string to convert integers to strings

class Solution {
public:
    bool isArmstrong(int n) {
        // Convert the integer n to a string to compute its length (number of digits)
        int numDigits = std::to_string(n).size();
        int sumOfPowers = 0; // This will hold the sum of the digits raised to the power of numDigits

        // Iterate over the digits of the number
        for (int temp = n; temp != 0; temp /= 10) {
            // Calculate the current digit to the power of numDigits and add it to sumOfPowers
            sumOfPowers += std::pow(temp % 10, numDigits);
        }

        // Return true if the sum of the powers equals the original number, false otherwise
        return sumOfPowers == n;
    }
};
```

TypeScript

```typescript
function isArmstrong(number: number): boolean {
    // Calculate the length of the number to determine the power to be used.
    const numberOfDigits = String(number).length;

    // Initialize the sum of the digits raised to the power of the number of digits.
    let sum = 0;

    // Initialize a variable 'value' to iterate through the digits of 'number'.
    for (let value = number; value; value = Math.floor(value / 10)) {
        // Add the current digit raised to the 'numberOfDigits' power to the sum.
        sum += Math.pow(value % 10, numberOfDigits);
    }

    // Return true if the sum is equal to the original number (Armstrong number), otherwise return false.
    return sum === number;
}
```


```python
class Solution:

    def is_armstrong(self, n: int) -> bool:
        # Convert the number to a string and calculate the length to determine k
        k = len(str(n))

        # Initialize the sum and a temporary variable with the original number
        sum_of_powers, temp_number = 0, n

        # Loop through each digit of the number
        while temp_number > 0:
            # Get the last digit and raise it to the power of k
            last_digit = temp_number % 10
            sum_of_powers += last_digit ** k

            # Remove the last digit from temp_number
            temp_number //= 10

        # Check if the sum of the digits to the power of k is equal to the original number
        return sum_of_powers == n
```


Time and Space Complexity

The given Python code defines a method `isArmstrong` which determines whether a given integer `n` is an Armstrong number or not. An Armstrong number (also known as a narcissistic number) of `k` digits is an integer such that the sum of its own digits each raised to the power of `k` equals the number itself.

To analyze its computational complexity, let's consider the following:

- Let `d` be the number of digits in the integer `n`, which is equal to `len(str(n))`.

Time Complexity:

- Determining the number of digits, `k = len(str(n))`, requires `O(d)` time.
- The while loop runs as long as `x` is not zero. Since `x` is reduced by a factor of 10 in each iteration, there will be `O(d)` iterations.
- Within each iteration, the modulus operation `x % 10`, the power operation `(x % 10) ** k` and integer division `x //= 10` are performed. The power operation has a time complexity of `O(k)` because it involves multiplying the number `x % 10` `k` times.
- Therefore, the total time complexity within the while loop is `O(d * k)`.

Since `k` is itself `O(d)` because the number of digits `d` is `log10(n)`, and `k ≈ log10(n)`, the overall time complexity when combining these operations is `O(d^2)`.

Space Complexity:

- The space complexity is determined by the extra space used by the variables `k`, `s`, and `x`.
- No additional data structures that grow with input size are used.
- The variables themselves only occupy constant space, `O(1)`.

Thus, the overall space complexity of the method is `O(1)`.

The provided Python code's time complexity is `O(d^2)` and space complexity is `O(1)`.