989. Add to Array-Form of Integer

Easy Array Math

Problem Description

The problem is to implement the addition of a non-negative integer `k` to a number represented as an array, where each element of the array represents a single digit of the number in the correct order. For instance, if `num = 1321`, then the array form would be `[1,3,2,1]`. If we were to add `k = 9` to this number, we're trying to find the array form of `1321 + 9`, which would result in `[1,3,3,0]`.

This is akin to how we perform addition by hand, starting from the rightmost digit (the least significant digit) and adding it to the corresponding digit of the other number (if available), carrying over the excess to the next digit on the left if the sum exceeds 9.

Intuition

This solution approach leverages the simple idea of adding two numbers together just as we would on paper, digit by digit from right to left, handling carry as needed.

We initialize a pointer `i` to the last index of the `num` array and set `carry` to `0`. In a loop, we perform the following until we have exhausted all digits in the `num` array and fully added `k` and accounted for any remaining carry:

- We calculate the total sum for the current digit by adding the appropriate digit from `num` (or `0` if we've passed the beginning of `num`), the last digit of `k`, and any carry from the previous step.
- We then use the `divmod()` function to both determine the digit to append to our answer in reverse (`v`) and the new carry value, ensuring that we only carry over when the sum is 10 or greater.
- Append `v` to the answer array `ans`.
- Update the value of `k` to reflect the leftover part after extracting the last digit by performing integer division by `10`.
- Decrement the index `i` to move to the next digit to the left.

Once the loop concludes (all digits and the carry have been processed), we reverse the `ans` array to represent the correct number as the digits have been appended in reverse order, starting with the least significant digit. This reversed `ans` array represents the `num + k` in array form, which is what we return as the solution.

Solution Approach

The implementation of the solution is a straightforward simulation of the addition operation that we perform manually between numbers. Here is the step-by-step break down:

1. Initialize the index `i` to point to the last digit of the input array `num` which represents the least significant digit of the number we are given. Also, initialize a `carry` variable to `0`, which will keep track of any carryover during the addition process.
2. Create an empty list `ans` that will store the result of the addition in reverse order.
3. Execute a `while` loop with the condition to continue as long as there is a digit remaining in the `num` array (`i >= 0`), or there is still a part of `k` that hasn't been added (`k`), or there is a carry from the previous addition (`carry`).
4. The sum for the current position is calculated by adding the current digit of `num` array, if any (`num[i]` or `0` if `i` is less than `0`), the last digit of `k` (`k % 10`), and the current `carry`. This is done using the line:

```
carry += (0 if i < 0 else num[i]) + (k % 10)
```

5. The `carry` and the value to append to the result list, `v`, are determined by using `divmod()`:

```
carry, v = divmod(carry, 10)
```

   This operation returns the quotient that becomes the next `carry` and the remainder `v` which is the digit to be appended to the answer.

6. Append `v` to the `ans` list:

```
ans.append(v)
```

7. The next digit of `k` is obtained by reducing `k` by one decimal place using the floor division operation:

```
k //= 10
```

8. Decrement the index `i` to move the pointer to the left, preparing for the next iteration to add the next higher order digit.
9. After the loop exits, the answer list `ans` contains the digits of the result in reversed order since we started adding from the least significant digit. Therefore, we reverse `ans` to correct the order:

```
return ans[::-1]
```

10. Return the reversed `ans` list as the final output, which now represents the `num + k` in the correct array-form.

In summary, the algorithm uses simple arithmetic and an iterative approach to simulate the addition process with careful bookkeeping of carries. The data structure used is a list to store the digits of the result which is eventually reversed to match the expected output format.

Example Walkthrough

Let's take a small example to illustrate the solution approach.

Suppose we have the array `num = [2,5,6]` which represents the number `256` and we want to add `k = 34` to this number. We expect to find the array form of `256 + 34`, which would result in `[2,9,0]`.

Here is how the algorithm would process this example:

1. Initialize the index `i` to `2` since the last digit of `num` is at index `2`. Also, the `carry` variable is initialized to `0`.
2. An empty list `ans` is created to store the result.
3. Enter the `while` loop which continues as long as `i >= 0`, `k > 0`, or `carry` is not `0`.
4. In the first iteration, `i` is `2`, there is no carry, and `k` is `34`. So, we calculate:

```
carry += num[2] + k % 10   # carry += 6 + 4
```

   `carry` now becomes `10`. The next step is to split this into `carry` and `v`:

```
carry, v = divmod(10, 10)   # carry = 1, v = 0
```

   We append `0` to `ans` (since `v = 0`) and adjust `k` and `i`:

```
k //= 10   # k becomes 3
i -= 1     # i becomes 1
```

5. The next iteration starts with `i = 1`, `carry = 1`, and `k = 3`. The sum is:

```
carry += num[1] + k % 10   # carry += 5 + 3
```

   `carry` now becomes `9`. Again, we split this into `carry` and `v`:

```
carry, v = divmod(9, 10)   # carry = 0, v = 9
```

   We append `9` to `ans` and adjust `k` and `i`:

```
k //= 10   # k becomes 0
i -= 1     # i becomes 0
```

6. The final iteration starts with `i = 0`, `carry = 0`, and `k = 0`. We perform the addition:

```
carry += num[0] + k % 10   # carry += 2 + 0
```

   `carry` now becomes `2`. Split into `carry` and `v`:

```
carry, v = divmod(2, 10)   # carry = 0, v = 2
```

   We append `2` to `ans`:

```
ans.append(v)
```

7. The loop finishes since `i` is now less than `0`, `k` is `0`, and `carry` is also `0`. The result list `ans` is currently `[0, 9, 2]`.
8. We reverse `ans` to get the correct order:

```
ans[::-1]   # becomes [2, 9, 0]
```

9. The final output is `[2, 9, 0]`, which is the correct array representation of `256 + 34`.

This walkthrough demonstrates each step of the algorithm on a small example, successfully showing how the solution approach adds two numbers when one is given as an array and the other as an integer (`k`).

Solution Implementation

Python

```python
from typing import List

class Solution:
    def addToArrayForm(self, num: List[int], k: int) -> List[int]:
        current_index, carry_over = len(num) - 1, 0
        result = []

        # Loop until we've processed each digit of `num` and `k`
        # as well as any remaining carry over value.
        while current_index >= 0 or k or carry_over:
            # If the current index is valid, extract the digit from `num`,
            # otherwise use 0. Add the last digit of `k` and the carry_over.
            carry_over += (num[current_index] if current_index >= 0 else 0) + (k % 10)
            # Use divmod to get the new carry_over and the digit to be added to the result.
            carry_over, digit = divmod(carry_over, 10)
            result.append(digit)
            # Update `k` and `current_index` for the next iteration.
            k //= 10
            current_index -= 1

        # The `result` list is currently in reverse, as digits were added from
        # the least significant digit (rightmost). We need to reverse it before returning.
        return result[::-1]
```

Java

```java
class Solution {
    public List<Integer> addToArrayForm(int[] num, int k) {
        int n = num.length - 1; // Initialize the index to the last element of the input array
        int carry = 0; // To hold carry-over values during addition
        LinkedList<Integer> result = new LinkedList<>(); // Using LinkedList to utilize addFirst method

        // Loop until we've processed all elements of num, or until k becomes 0, or until there's no carry left
        while (n >= 0 || k > 0 || carry > 0) {
            // If n is within bounds, add num[n] to carry else add 0.
            // Also add the last digit of k to carry
            carry += (n >= 0 ? num[n--] : 0) + k % 10;

            // Add the unit's place of carry to the front of result
            result.addFirst(carry % 10);

            // Remove the unit's place digit from carry and k
            carry /= 10;
            k /= 10;
        }

        return result;
    }
}
```

C++

```cpp
#include <vector>
#include <algorithm>

class Solution {
public:
    // Adds an integer k to a large integer represented as a vector of digits.
    vector<int> addToArrayForm(vector<int>& num, int k) {
        int numIndex = num.size() - 1; // Start from the last digit of the number.
        int carry = 0; // Initialize carry for addition.
        vector<int> result; // The result vector to store the sum.

        // Iterate until all digits are processed, or there is no carry, or k is not zero.
        while (numIndex >= 0 || k > 0 || carry > 0) {
            // If numIndex is within bounds, add the current digit to carry.
            // Otherwise, add 0 (when numIndex < 0).
            carry += (numIndex >= 0 ? num[numIndex] : 0) + (k % 10);
            result.push_back(carry % 10); // Extract the last digit of carry and add it to result.
            carry /= 10; // Update carry for the next iteration.
            k /= 10; // Move to the next digit in k.
            numIndex--; // Move to the previous digit in num.
        }

        // The result currently contains the digits in reverse order.
        reverse(result.begin(), result.end()); // Reverse the result to get the correct order.

        return result;
    }
};
```

TypeScript

```typescript
function addToArrayForm(numArray: number[], k: number): number[] {
    // Convert `k` to an array of its digits
    let kArray: number[] = [...k.toString()].map(Number);

    // Initialize the answer array to hold the result
    let answer: number[] = [];

    // Sum is used to handle the current value while adding digits
    let carry: number = 0;

    // Process while there are digits in `numArray` or `kArray`, or if there is a carry value
    while (numArray.length || kArray.length || carry) {
        // Pop the last digit from `numArray` or use 0 if empty
        let numDigit = numArray.pop() || 0;

        // Pop the last digit from `kArray` or use 0 if empty
        let kDigit = kArray.pop() || 0;

        // Add the digits and the carry
        carry += numDigit + kDigit;

        // Unshift (insert at the beginning) the last digit of the sum into `answer`
        answer.unshift(carry % 10);

        // Floor division by 10 to get a new carry value
        carry = Math.floor(carry / 10);
    }

    // Return the result stored in `answer`
    return answer;
}
```

```python
from typing import List

class Solution:
    def addToArrayForm(self, num: List[int], k: int) -> List[int]:
        current_index, carry_over = len(num) - 1, 0
        result = []

        # Loop until we've processed each digit of `num` and `k`
        # as well as any remaining carry over value.
        while current_index >= 0 or k or carry_over:
            # If the current index is valid, extract the digit from `num`,
            # otherwise use 0. Add the last digit of `k` and the carry_over.
            carry_over += (num[current_index] if current_index >= 0 else 0) + (k % 10)
            # Use divmod to get the new carry_over and the digit to be added to the result.
            carry_over, digit = divmod(carry_over, 10)
            result.append(digit)
            # Update `k` and `current_index` for the next iteration.
            k //= 10
            current_index -= 1

        # The `result` list is currently in reverse, as digits were added from
        # the least significant digit (rightmost). We need to reverse it before returning.
        return result[::-1]
```

Time and Space Complexity

Time Complexity

The time complexity of the code is `O(max(N, logK))` where `N` is the number of digits in `num` and `logK` represents the number of digits in `k`. This is because the while loop runs once for each digit of the number `num` and for each digit of the number `k`. When `k` has more digits than `num`, the loop will continue until all digits of `k` have been processed. In the reverse situation, it will process all the digits of `num` even if `k` is already reduced to zero.

Space Complexity

The space complexity of the code is `O(max(N, logK))` because we create a list `ans` that stores the result, which in the worst case will have max(N, logK) digits (the maximum out of the number of digits in `num` and the number of digits in `k` after addition). Furthermore, space is used for variables `i`, `carry`, `v`, and the stack space for the recursive call when reversing the answer list, but these do not depend on the size of the inputs and thus contribute only a constant factor, which is negligible when analyzing space complexity.