445. Add Two Numbers II

Medium Stack Linked List Math

Leetcode Link

Problem Description

In this problem, we are given two linked lists representing two non-negative integer numbers. The digits are stored in reverse order, meaning that the least significant digit is at the head of the list, and each node contains a single digit. Our goal is to add these two numbers together and return a linked list that represents the sum of these two numbers. Importantly, the most significant digit is at the front of the result linked list. Leading zeros are not present in the numbers, except in the case of the number 0 itself.

Intuition

The intuitive approach to this problem is to follow the way we normally add numbers on paper. Normally, we add numbers starting from the least significant digit and move towards the most significant digit, carrying over any overflowing value to the next digit. However, in this scenario, since the most significant digit is at the head, we need a way to process the numbers from least significant to most significant.

To accomplish this, we can use stacks for each linked list to reverse the digits. By pushing the digits on to the stacks, the last digit pushed will be the least significant, which allows us to pop from the stack to get digits in the reverse order, allowing us to add the two numbers from least to most significant digits just as we would on paper.

Once we have the two stacks, we add the digits, keeping track of the carry. Every time we pop a digit from a stack, we add it to our running sum along with the carried value from the previous addition. If the sum is greater than 9, we calculate the new carry that will be added to the next pair of digits. Else, the carry is set to zero.

The added complication compared to adding numbers directly is dealing with different lengths of the linked lists. We handle this by considering a value of 0 for a list if it has no more digits to pop.

Once we have processed both stacks, we may still have a carry left over, which should form a new digit in the result. For example, adding 95 and 7 would require a new '1' at the front after processing the two initial digits.

The solution creates the resulting list in reverse (starting with the least significant digit), using a dummy head to simplify the edge case of the leading digit. This way, we insert each new digit at the head of the resulting list. At the end, we simply need to return the list starting from the dummy head's next node, which represents the most significant digit of the result.

Solution Approach

The solution to this problem consists of several key steps, which utilize data structures such as stacks, as well as a dummy node pattern to simplify the list construction. The algorithm proceeds as follows:

1. **Initialize Stacks**: Two stacks `s1` and `s2` are created to hold the digits from the two linked lists, ensuring we can process them in a least-significant to most-significant order.
2. **Populate Stacks**: We iterate through each linked list (`l1` and `l2`), pushing the value of each node onto the respective stack. Thus, the last node's value will be at the top of the stack.
3. **Prepare for Addition**: A dummy node `dummy` is created to facilitate the easy addition of new nodes at the front of the result linked list. A variable `carry` is initialized to `0`.
4. **Add Numbers**:
   - While there are still values in either `s1` or `s2`, or there is a nonzero `carry`, we continue to process the addition.
   - We pop values from `s1` and `s2` respectively (if available, or default to `0`) and add them together along with the `carry`.
   - The sum is split using `divmod(s, 10)`, where `sum = s` is the result of the addition, and `divmod()` returns a tuple `(carry, val)`. Here, `carry` is the amount to be carried to the next most significant digit, and `val` is the value of the current digit.
5. **Build Result List**:
   - For each summation step, a new node with value `val` is created and prepended to the linked list starting at `dummy.next`.
   - This ensures that we are building the result list from least-significant to most-significant, starting with the dummy node, which does not contain a digit.
6. **Return Result**: After completing the iteration, we return the linked list starting from `dummy.next`, which skips over the dummy node to return the correct leading digit of the result.

The code makes use of standard linked list manipulation techniques, leveraging the stack data structure to reverse the processing order of digits in the linked list. By inserting nodes at the head (using the dummy node pattern), we efficiently construct the list in reverse, following our algorithm for addition. This effectively tackles the constraint that the most significant digit appears first in each input linked list.

The mathematical concepts invoked are primarily those related to simple addition and carry-over rules from elementary arithmetic. The particular use of the Python built-in function `divmod()` is a straightforward way to separate the value of the current digit and the carry in a single line of code, enhancing clarity and efficiency.

Example Walkthrough

Let's illustrate the solution approach with a small example where we have two linked lists representing the numbers 243 and 564. The linked lists for these numbers are given in reverse order: `l1 = 3 -> 4 -> 2` and `l2 = 4 -> 6 -> 5`.

Step-by-step process:

1. **Initialize Stacks**: Create two empty stacks, `s1` and `s2`.
2. **Populate Stacks**: We traverse `l1` and push each digit onto `s1`, and do the same with `l2` to `s2`. After this, `s1` holds the sequence `2, 4, 3` and `s2` holds `5, 6, 4`.

```
l1: [3, 4, 2]
l2: [4, 6, 5]
```

3. **Prepare for Addition**: Create a dummy node `dummy` and set `carry` to `0`.
4. **Add Numbers**: The steps within the loop:
   - Pop `s1` (3) and `s2` (4), add them with `carry` (0) giving us `3 + 4 + 0 = 7`. Set `carry` to 0, and the digit to add is 7.
   - Pop `s1` (4) and `s2` (6), add them with `carry` (0) giving us `4 + 6 + 0 = 10`. Set `carry` to 1 (divmod(10, 10)), and the digit to add is 0.
   - Pop `s1` (2) and `s2` (5), add them with `carry` (1) giving us `2 + 5 + 1 = 8`. Set `carry` to 0, and the digit to add is 8.
   - At this point, both stacks are empty, and `carry` is 0, so we end the loop.
5. **Build Result List**:
   - Add nodes in front of the dummy node with the digits in the following order: 7, 0, 8. So the list after these steps would look like `dummy -> 8 -> 0 -> 7`.
6. **Return Result**: The final result is returned by skipping the dummy node, so we get `8 -> 0 -> 7`, representing the number 807, which is the sum of 243 and 564.

Python Solution

```python
# Definition for singly-linked list.
class ListNode:
    def __init__(self, value=0, next_node=None):
        self.value = value
        self.next = next_node

class Solution:
    def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:
        # Initialize stacks for the two numbers
        stack1, stack2 = [], []

        # Push all values from the first linked list onto stack1
        while l1:
            stack1.append(l1.value)
            l1 = l1.next

        # Push all values from the second linked list onto stack2
        while l2:
            stack2.append(l2.value)
            l2 = l2.next

        # Initialize a dummy node to which we'll add the resulting digits
        dummy_node = ListNode()

        # Initialize carry to handle sums greater than 9
        carry = 0

        # Iterate while there are values in either of the stacks or there is a carry value
        while stack1 or stack2 or carry:
            # Pop from each stack; if a stack is empty, use 0
            sum1 = stack1.pop() if stack1 else 0
            sum2 = stack2.pop() if stack2 else 0

            # Calculate the total sum and update carry
            total = sum1 + sum2 + carry
            carry, digit = divmod(total, 10)

            # Create a new node with the resulting digit and insert it at the front
            # at the list since we're adding numbers from the least significant digit)
            new_node = ListNode(digit, dummy_node.next)
            dummy_node.next = new_node

        # Return the next node after dummy node as it only serves as an anchor
        return dummy_node.next
```

Java Solution

```java
class Solution {
    /**
     * Adds two numbers represented by two linked lists, where each node contains a single digit.
     * The digits are stored in reverse order, such that the 1's digit is at the head of the list.
     *
     * @param l1 first linked list representing the first number.
     * @param l2 second linked list representing the second number.
     * @return ListNode list representing the sum of the two numbers.
     */
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        // Stacks to store the digits of the two numbers
        Stack<Integer> stack1 = new Stack<>();
        Stack<Integer> stack2 = new Arrays<>();

        // Push all digits of l1 into stack1
        while (l1 != null) {
            stack1.push(l1.val);
            l1 = l1.next;
        }

        // Push all digits of l2 into stack2
        while (l2 != null) {
            stack2.push(l2.val);
            l2 = l2.next;
        }

        // "dummyHead" is the placeholder for the result linked list
        ListNode dummyHead = new ListNode();
        int carry = 0; // Initialize carry to zero

        // Loop until both stacks are empty and there is no carry left
        while (!stack1.isEmpty() || !stack2.isEmpty() || carry != 0) {
            // If a stack is empty, use 0 as the digit, otherwise pop the top digit
            int sum = stack1.isEmpty() ? 0 : stack1.pop() + (stack2.isEmpty() ? 0 : stack2.pop()) + carry;

            // Create a new node with the sum's least significant digit and add it to the front of the linked list "dummyHead"
            ListNode newNode = new ListNode(sum % 10);
            newNode.next = dummyHead.next;
            dummyHead.next = newNode;

            // Calculate the new carry value
            carry = sum / 10;
        }

        // Return the sum linked list, which is "dummyHead.next" because "dummyHead" is a dummy node
        return dummyHead.next;
    }
}
```

C++ Solution

```cpp
// Definition for singly-linked list.
struct ListNode {
    int val;  // Value of the node
    ListNode *next;  // Pointer to the next node
    ListNode() : val(0), next(nullptr) {}  // Default value
    ListNode(int x) : val(x), next(nullptr) {}
    // Constructor to initialize with a specific value
    ListNode(int x, ListNode *next) : val(x), next(next) {}
    // Constructor to initialize with a specific value and next node
    ListNode(int x, ListNode *next) : val(x), next(next) {}
};

class Solution {
public:
    // Function to add two numbers represented by two linked lists
    ListNode* addTwoNumbers(ListNode *l1, ListNode *l2) {
        stack<int> stack1;  // Stack to store digits of the first number.
        stack<int> stack2;  // Stack to store digits of the second number

        // Push all digits of the first number onto stack1
        for (; l1; l1 = l1->next) {
            stack1.push(l1->val);
        }
        // Push all digits of the second number onto stack2
        for (; l2; l2 = l2->next) {
            stack2.push(l2->val);
        }

        ListNode dummyHead = new ListNode(); // Dummy head for the result list
        int carry = 0; // Initialize carry to 0

        // Continue looping until both stacks are empty or there is a carry
        while (!stack1.empty() || !stack2.empty() || carry) {
            int sum = carry; // Start with carry even from previous iteration

            // Add the top of stack1 to sum and pop it
            if (!stack1.empty()) {
                sum += stack1.top();
                stack1.pop();
            }
            // Add the top of stack2 to sum and pop it
            if (!stack2.empty()) {
                sum += stack2.top();
                stack2.pop();
            }

            // Create a new node with the digit part of sum and insert it at the front
            // Assign the created node to the next of the dummy head
            dummyHead->next = new ListNode(sum % 10, dummyHead->next);
            // Update carry
            carry = sum / 10;
        }

        // Return the next of the dummy head which points to the actual result list
        return dummyHead->next;
    }
};
```

Typescript Solution

```typescript
// Definition for singly-linked list.
type ListNode = {
    val: number
    next: ListNode | null
};
// Constructor function to create a new ListNode instance
function ListNode(val?: number, next?: ListNode | null): ListNode {
    // ...
}

/**
 * Sums two numbers represented by linked lists, where each node contains a single digit.
 * The digits are stored in reverse order, such that the 1's digit is at the head of the list.
 * @param {ListNode} l1 - The first linked list representing the first number.
 * @param {ListNode} l2 - The second linked list representing the second number.
 * @returns {ListNode} A linked list representing the sum of the two numbers.
 */
function addTwoNumbers(l1: ListNode | null, l2: ListNode | null): ListNode | null {
    // Stacks to hold the digits of the two numbers
    const stack1: number[] = [];
    const stack2: number[] = [];

    // Build up the stacks with the digits from the linked lists
    while (l1) {
        stack1.push(l1.val);
        l1 = l1.next;
    }
    while (l2) {
        stack2.push(l2.val);
        l2 = l2.next;
    }

    // A dummy head for the result linked list
    let resultHead = new ListNode();

    // Variable to keep track of the carry
    let carry = 0;

    // Process the stacks until they are empty or there is a carry left
    while (stack1.length > 0 || stack2.length > 0 || carry > 0) {
        // Calculate the sum of the current digits and the carry
        const sum = (stack1.pop() || 0) + (stack2.pop() || 0) + carry;

        // Create a new node with the digit part of the sum
        resultHead.next = new ListNode(sum % 10, resultHead.next);

        // Update the carry for the next iteration
        carry = Math.floor(sum / 10);
    }

    // Return the result linked list, discarding the dummy head
    return resultHead.next;
};
```

Time and Space Complexity

Time Complexity

The time complexity of the code is determined by several factors:

1. The time it takes to iterate through the two linked lists (l1 and l2) to build the stacks s1 and s2. Since we're traversing each list once, the time complexity for this part is $O(n)$ for l1 and $O(m)$ for l2, where n is the number of nodes in l1 and m is the number of nodes in l2.
2. The time it takes to pop elements from the stacks s1 and s2 and to add the carry if it exists. In the worst case, we'll pop each element once from both stacks, which gives us another $O(n + m)$ operations.

As the two stages are sequential, we can add these complexities together, which results in a total time complexity of $O(n + m)$.

Space Complexity

The space complexity is determined by the additional space we need aside from the input lists:

1. We're creating two stacks s1 and s2 that can potentially contain all the elements of l1 and l2. This yields a space complexity of $O(n + m)$.
2. We also create a dummy node and potentially new nodes equivalent to the total number of nodes that are in the two lists plus one additional node in case there is a carry at the last node. However, creating nodes for the resulting linked list does not add to the space complexity when considering the total space used since the output space is not considered in space complexity analysis for this problem.

As there's no recursive calls or dynamic memory allocation that exceeds the size of input lists, the space complexity of the code remains $O(n + m)$.