445. Add Two Numbers II Medium Stack Linked List Math Leetcode Link

Problem Description

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.

In this problem, we are given two linked lists representing two non-negative integer numbers. The digits are stored in reverse order,

to most significant.

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

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.

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

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.

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

2. Populate Stacks: We iterate through each linked list (11 and 12), pushing the value of each node onto the respective stack.

least-significant to most-significant order.

which does not contain a digit.

node to return the correct leading digit of the result.

carry in a single line of code, enhancing clarity and efficiency.

1. Initialize Stacks: Create two empty stacks, s1 and s2.

4. Add Numbers: The steps within the loop:

like dummy -> 8 -> 0 -> 7.

which is the sum of 243 and 564.

1 # Definition for singly-linked list.

self.value = value

self.next = next_node

stack1, stack2 = [], []

l1 = l1.next

dummy_node = ListNode()

stack1.append(l1.value)

while stack1 or stack2 or carry:

carry, digit = divmod(total, 10)

newNode.next = dummyHead.next;

// Calculate the new carry value

// Value of the node

// Constructor to initialize with default values

// Constructor to initialize with a specific value

ListNode *next; // Pointer to the next node

ListNode(): val(0), next(nullptr) {}

dummyHead.next = newNode;

carry = sum / 10;

return dummyHead.next;

1 // Definition for singly-linked list.

def __init__(self, value=0, next_node=None):

Initialize stacks for the two numbers

Push all values from the first linked list onto stack1

Initialize carry to handle sums greater than 9

Pop from each stack; if a stack is empty, use 0

new_node = ListNode(digit, dummy_node.next)

Initialize a dummy node to which we'll add the resulting digits

3. Prepare for Addition: Create a dummy node dummy and set carry to 0.

At this point, both stacks are empty, and carry is 0, so we end the loop.

Solution Approach

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 \$1 or \$2, 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.

For each summation step, a new node with value val is created and prepended to the linked list starting at dummy.next.

5. Build Result List:

each input linked list.

Example Walkthrough

Step-by-step process:

6. Return Result: After completing the iteration, we return the linked list starting from dummy. next, which skips over the dummy

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

This ensures that we are building the result list from least-significant to most-significant, starting with the dummy node,

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

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: $11 = 3 \rightarrow 4 \rightarrow 2$ and $12 = 4 \rightarrow 6 \rightarrow 5$.

2. Populate Stacks: We traverse 11 and push each digit onto \$1, and do the same with 12 to \$2. After this, \$1 holds the sequence 2, 4, 3 and s2 holds 5, 6, 4. 1 s1: [3, 4, 2] 2 s2: [4, 6, 5]

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.

5. Build Result List:

2 class ListNode:

class Solution:

while 11:

carry = 0

13

14

15

21

23

24

25

26

27

28

29

30

36

37

38

39

40

38

39

40

41

43

44

45

46

47

49

48 }

6

C++ Solution

2 struct ListNode {

int val;

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

def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:

Iterate while there are values in either of the stacks or there is a carry value

Create a new node with the resulting digit and insert it at the front

of the list (since we are adding numbers from the least significant digit)

Python Solution

6. Return Result: The final result is returned by skipping the dummy node, so we get 8 -> 0 -> 7, representing the number 807,

∘ 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.

16 17 # Push all values from the second linked list onto stack2 while 12: 18 stack2.append(l2.value) 19 20 12 = 12.next

31 sum1 = stack1.pop() if stack1 else 0 32 sum2 = stack2.pop() if stack2 else 0 33 34 # Calculate the total sum and update carry 35 total = sum1 + sum2 + carry

```
dummy_node.next = new_node
41
42
43
           # Return the next node after dummy node as it only serves as an anchor
           return dummy_node.next
44
45
Java Solution
   class Solution {
       /**
        * Adds two numbers represented by two linked lists, where each node contains a single digit.
3
        * The digits are stored in reverse order, such that the 1's digit is at the head of the list.
5
        * @param ll First linked list representing the first number.
        * @param 12 Second linked list representing the second number.
        * @return A linked list representing the sum of the two numbers.
8
9
        */
       public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
10
           // Stacks to store the digits of the two numbers
11
12
           Deque<Integer> stack1 = new ArrayDeque<>();
13
           Deque<Integer> stack2 = new ArrayDeque<>();
14
15
           // Push all digits of ll into stackl
16
           while (l1 != null) {
17
               stack1.push(l1.val);
18
               l1 = l1.next;
19
20
21
           // Push all digits of l2 into stack2
22
           while (l2 != null) {
23
               stack2.push(l2.val);
24
               12 = 12.next;
25
26
27
           // 'dummyHead' is the placeholder for the result linked list
28
           ListNode dummyHead = new ListNode();
           int carry = 0; // Initialize carry to zero
29
30
31
           // Loop until both stacks are empty and there is no carry left
32
           while (!stack1.isEmpty() || !stack2.isEmpty() || carry != 0) {
33
               // If a stack is empty, use 0 as the digit, otherwise pop the top digit
34
               int sum = (stack1.isEmpty() ? 0 : stack1.pop()) + (stack2.isEmpty() ? 0 : stack2.pop()) + carry;
35
36
               // 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);
37
```

// Return the sum linked list, which is 'dummyHead.next' because 'dummyHead' is a dummy node

```
ListNode(int x) : val(x), next(nullptr) {}
  8
         // Constructor to initialize with a specific value and next node
  9
 10
         ListNode(int x, ListNode *next) : val(x), next(next) {}
 11 };
 12
 13 class Solution {
 14 public:
         // Function to add two numbers represented by two linked lists
 15
         ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
 16
 17
             stack<int> stack1; // Stack to store digits of the first number
 18
             stack<int> stack2; // Stack to store digits of the second number
 19
 20
             // Push all digits of the first number onto stack1
             for (; l1; l1 = l1->next) {
 21
 22
                 stack1.push(l1->val);
 23
 24
             // Push all digits of the second number onto stack2
 25
             for (; l2; l2 = l2->next) {
                 stack2.push(l2->val);
 26
 27
 28
 29
             ListNode* dummyHead = new ListNode(); // Dummy head for the result list
 30
             int carry = 0; // Initialize carry to 0
 31
 32
             // Continue adding until both stacks are empty or there is a carry
             while (!stack1.empty() || !stack2.empty() || carry) {
 33
 34
                 int sum = carry; // Start with carry over from previous iteration
 35
 36
                 // Add the top of stack1 to sum and pop it
                 if (!stack1.empty()) {
 37
 38
                     sum += stack1.top();
                     stack1.pop();
 39
 40
                 // Add the top of stack2 to sum and pop it
 41
 42
                 if (!stack2.empty()) {
 43
                     sum += stack2.top();
 44
                     stack2.pop();
 45
 46
                 // Create a new node with the digit part of sum and insert it at the front
 47
                 // Assign the created node to the next of the dummy head
                 dummyHead->next = new ListNode(sum % 10, dummyHead->next);
 48
 49
                 // Update carry
 50
                 carry = sum / 10;
 51
 52
 53
             // Return the next of the dummy head which points to the actual result list
 54
             return dummyHead->next;
 55
 56
    };
 57
Typescript Solution
   // Definition for singly-linked list.
   type ListNode = {
       val: number;
       next: ListNode | null;
       // Constructor function to create a new ListNode instance
       new(val?: number, next?: ListNode | null): ListNode;
8
```

* Sums two numbers represented by linked lists, where each node contains a single digit.

function addTwoNumbers(l1: ListNode | null, l2: ListNode | null): ListNode | null {

* @param ll - The first linked list representing the first number.

* @param 12 - The second linked list representing the second number.

* @returns - A linked list representing the sum of the two numbers.

// Build up the stacks with the digits from the linked lists

// Process the stacks until they are empty or there is a carry left

// Calculate the sum of the topmost elements and the carry

resultHead.next = new ListNode(sum % 10, resultHead.next);

const sum = (stack1.pop() ?? 0) + (stack2.pop() ?? 0) + carry;

while (stack1.length > 0 || stack2.length > 0 || carry > 0) {

// Create a new node with the digit part of the sum

// Update the carry for the next iteration

// Stacks to hold the digits of the two numbers

// A dummy head for the result linked list

// Variable to keep track of the carry

const stack1: number[] = [];

const stack2: number[] = [];

stack1.push(l1.val);

stack2.push(l2.val);

let resultHead = new ListNode();

Time and Space Complexity

l1 = l1.next;

l2 = l2.next;

while (l1) {

while (12) {

let carry = 0;

* The digits are stored in reverse order, such that the 1's digit is at the head of the list.

carry = Math.floor(sum / 10); // Return the result linked list, discarding the dummy head 49 50 return resultHead.next; 51 }

Time Complexity

Space Complexity

analysis for this problem.

9

15

17

18

19

20

21

22

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

52

/**

*/

1. The time it takes to iterate through the two linked lists (I1 and I2) to build the stacks s1 and s2. Since we're traversing each list once, the time complexity for this part is O(n) for 11 and O(m) for 12, where n is the number of nodes in 11 and m is the number of nodes in 12.

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

- 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 0(n + m).

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

- 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 I1 and I2. This yields a space complexity of
 - 0(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

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).