43. Multiply Strings String Medium Simulation

Problem Description

calculate the product of these two numbers and return the result as a string. The constraint is that we are not allowed to use any built-in library that can handle big integers, nor can we simply convert the strings to integers and multiply them in the standard way. This challenges us to think about how we can perform multiplication manually, mimicking the way one might do it on paper.

In this problem, we are given two non-negative integer numbers represented as strings, num1 and num2, and our task is to

Intuition

results in a series of partial products, which are then added together to form the final product. To implement this in code, we need a data structure to store intermediate results. The approach is to use an array arr to store the digits of the partial products. The length of this array is the sum of the lengths of num1 and num2 because that's the

The intuition behind the solution is based on how we perform multiplication by hand between two numbers. More precisely, when

we multiply, say, a three-digit number by a two-digit number, we do it digit by digit and keep track of the carries. This process

maximum possible length of the result (e.g., 99 * 99 is 9801, four digits long, which is the sum of the lengths of the numbers being multiplied). Next, we iterate over each digit of num1 and num2 in nested loops, and for each pair of digits, we multiply them and add the result

to the corresponding position in arr. The key formula for the index in arr where we should accumulate the product of digits at positions i and j is arr[i+j+1]. After we have all the partial products, we then iterate through the arr array to handle carries at each position, adjusting each

digit so that it represents a proper digit in a number (less than 10), and propagating the carry to the next position. Finally, we need to return the string representation of the number stored in the array, but we skip any leading zeroes, as they don't contribute to the magnitude of the number. We join the remaining digits together to form the resulting product string to be

Solution Approach The solution follows these steps:

Check for Zero: If either num1 or num2 is "0", the product is "0". We catch this case early to simplify further logic.

Initialization: Determine the lengths m and n of num1 and num2, respectively. Initialize an array arr of length m + n to hold the digits of the product.

returned.

- Digit-by-Digit Multiplication:
- ∘ Iterate through the digits of num1 and num2 in descending order using two nested loops, with indices i and j. Convert current digits to integers and multiply them: a * b. \circ Add the multiplication result to an appropriate position in the array arr: arr[i + j + 1] += a * b.

∘ The position i + j + 1 comes from the fact that when you multiply a digit at position i of num1 with a digit at position j of num2, the

Handling Carries:

Converting Array to String:

- For each position, if the value is greater than 9, divide by 10 to find the carry and keep the remainder: ■ arr[i - 1] += arr[i] // 10: This propagates the carry to the next higher position.
- If the digit at the highest position (arr[0]) is zero, it's a leading zero and should be omitted. Determine the starting index for the conversion

To illustrate the solution approach, let's walk through a small example where num1 = "13" and num2 = "24".

 \circ We iterate over num1 and num2 in reverse order so we will have the indices i=1,0 for num1 and j=1,0 for num2.

 Join the digits from the arr starting at the right index i to form a string without leading zeros: "".join(str(x) for x in arr[i:]). This implementation doesn't use any special algorithms or data structures—it uses simple arrays and elementary math operations

Check for Zero: Neither num1 nor num2 are "0", so we proceed.

(i), which is 0 if there's no leading zero, and 1 otherwise.

Iterate backwards through arr, starting from the end, to process carries.

result will contribute to the digits at positions i + j and i + j + 1 of the product.

arr[i] %= 10: This ensures that the current position has only a single digit.

just like manual multiplication on paper.

Digit-by-Digit Multiplication:

Example Walkthrough

* 4) = 12.

Handling Carries:

Solution Implementation

return "0"

Python

class Solution:

Initialization: The length m of num1 is 2, and the length n of num2 is also 2. We initialize an array arr of length m + n = 4 to [0, 0, 0, 0].

∘ For i=1 (num1[1] = "3") and j=1 (num2[1] = "4"), we multiply the digits and add the result to arr[i + j + 1], which is arr[3] = 0 + (3

to simulate digit-by-digit multiplication, carefully considering the placement of each partial product and the handling of carries,

 We repeat this for all pairs of digits: • i=1, j=0: arr[2] += 3 (num1[1]) * 2 (num2[0]); <math>arr becomes [0, 0, 7, 2].

Starting from the end of arr, we process the carries:

def multiply(self, num1: str, num2: str) -> str:

Determine the lengths of the input strings

for i in range(length num1 - 1, -1, -1):

digit num2 = int(num2[i])

Skip leading zeros in the result list

// Convert the product array into a string.

product.append(productArray[i]);

string multiply(string num1, string num2) {

if (num1 == "0" || num2 == "0") {

// Initialize the sizes of the input numbers

vector<int> result(length1 + length2, 0);

for (int $i = length1 - 1; i >= 0; --i) {$

int length1 = num1.size(), length2 = num2.size();

for (int i = length2 - 1; i >= 0; --i) {

for (int $i = result.size() - 1; i > 0; --i) {$

// Skip any leading zeros in the result vector

int startIndex = result[0] == 0 ? 1 : 0;

// Convert the result vector to a string

string resultStr:

// Create a vector to store the multiplication result

// Multiply each digit of num1 with each digit of num2

result[i + j + 1] += digit1 * digit2;

// Handle carrying over the value for digits greater than 9

result[i] %= 10; // Remainder stays at current position

result[i - 1] += result[i] / 10; // Carry over

int digit1 = num1[i] - '0'; // Convert char to integer

int digit2 = num2[j] - '0'; // Convert char to integer

// Add to the corresponding position in the result vector

return "0";

return product.toString();

StringBuilder product = new StringBuilder();

for (int i = startIndex; i < productArray.length; ++i) {</pre>

// Check if either input is "0", if yes, then result is "0"

for j in range(length num2 - 1, -1, -1):

digit num1 = int(num1[i])

if num1 == "0" or num2 == "0":

• i=0, j=1: arr[1] += 1 (num1[0]) * 4 (num2[1]); arr becomes [0, 4, 7, 2]. • i=0, j=0: arr[1] += 1 (num1[0]) * 2 (num2[0]); arr becomes [0, 6, 7, 2].

■ At arr[2] (7): We add 7 // 10 to arr[1] and set arr[2] to 7 % 10. arr stays [0, 6, 7, 2] as 7 is less than 10.

At arr[1] (6): We do the same. There's no carry since 6 is also less than 10. **Converting Array to String:** o arr is [0, 6, 7, 2], so we omit the leading zero.

• The final product string is "672" as we join the digits from arr starting from index 1.

If either number is "0", return "0" because the product will also be "0"

Reverse process of multiplication, processing digits from the end

result[i + j + 1] += digit_num1 * digit_num2

○ We store 2 in arr[3] and carry over 1 to arr[2], so arr becomes [0, 0, 1, 2].

The final result of multiplying "13" by "24" using our manual algorithm results in "672", which matches what we expect from standard multiplication.

Add product of current digits to the previously stored value in result list

keep only the last digit

length_num1, length_num2 = len(num1), len(num2) # Create a result list to store the product digits result = [0] * (length_num1 + length_num2)

Handle carrying over digits > 9 to the next place for i in range(length num1 + length num2 - 1, 0, -1): result[i - 1] += result[i] // 10 # carry over

result[i] %= 10

```
start_index = 0 if result[0] != 0 else 1
        # Convert the result list to string
        return "".join(str(digit) for digit in result[start_index:])
Java
class Solution {
    public String multiply(String num1, String num2) {
        // If either number is 0, the product will be 0.
        if ("0".equals(num1) || "0".equals(num2)) {
            return "0";
        // Get lengths of both numbers.
        int length1 = num1.length(), length2 = num2.length();
        // Initialize an array to store the product of each digit multiplication.
        int[] productArray = new int[length1 + length2];
        // Loop over each digit in num1 and num2 and multiply them.
        for (int i = length1 - 1; i >= 0; --i) {
            int digit1 = num1.charAt(i) - '0';
            for (int i = length2 - 1; i >= 0; --i) {
                int digit2 = num2.charAt(j) - '0';
                // Add the product of the two digits to the corresponding position.
                productArray[i + j + 1] += digit1 * digit2;
        // Normalize the productArray so that each position is a single digit.
        for (int i = productArray.length - 1; i > 0; --i) {
            productArray[i - 1] += productArray[i] / 10; // Carry over the tens to the next left cell.
            productArray[i] %= 10; // Keep the units in the current cell.
        // Skip the leading 0 in the product array if it exists.
        int startIndex = productArray[0] == 0 ? 1 : 0;
```

C++

public:

class Solution {

```
for (int i = startIndex; i < result.size(); ++i) {</pre>
            resultStr += '0' + result[i]; // Convert integer to char and append to resultStr
        // Return the final product as a string
        return resultStr;
};
TypeScript
// Multiplies two non-negative integers represented as number strings and returns the product as a string.
function multiply(num1: string, num2: string): string {
    // If either number is '0', the product will be '0'.
    if ([num1, num2].includes('0')) return '0';
    // Get the lengths of the two number strings.
    const length1 = num1.length;
    const length2 = num2.length;
    let answer = '';
    // Iterate over each digit in the first number.
    for (let i = 0; i < length1; i++) {</pre>
        let currentDigit1 = parseInt(num1.charAt(length1 - i - 1), 10);
        let partialSum = '';
        // Iterate over each digit in the second number.
        for (let i = 0; i < length2; i++) {</pre>
            let currentDigit2 = parseInt(num2.charAt(length2 - j - 1), 10);
            // Multiply current digits and add them to the partial sum with proper offset.
            partialSum = addString(partialSum, (currentDigit1 * currentDigit2) + '0'.repeat(j));
        // Construct the answer with the accumulated partial sum with proper offset.
        answer = addString(answer, partialSum + '0'.repeat(i));
    // Return the final product as a string.
    return answer;
// Adds two number strings and returns the sum as a string.
function addString(numString1: string, numString2: string): string {
    const length1 = numString1.length;
    const length2 = numString2.length;
    let answerArray = [];
    let carry = 0;
    // Add the two strings together, digit by digit, from the end to the beginning.
    for (let i = 0; i < Math.max(length1, length2) || carry > 0; <math>i++) {
        let digit1 = i < length1 ? parseInt(numString1.charAt(length1 - <math>i - 1), 10) : 0;
        let digit2 = i < length2 ? parseInt(numString2.charAt(length2 - i - 1), 10) : 0;</pre>
        // Calculate the sum for current place and maintain the carry for the next iteration.
```

Handle carrying over digits > 9 to the next place for i in range(length num1 + length num2 - 1, 0, -1): result[i - 1] += result[i] // 10 # carry over result[i] %= **10**

let sum = digit1 + digit2 + carry;

// Join the computed digits to form the final sum string.

If either number is "0", return "0" because the product will also be "0"

Reverse process of multiplication, processing digits from the end

result[i + j + 1] += digit_num1 * digit_num2

return "".join(str(digit) for digit in result[start_index:])

However, since 0(m * n) dominates 0(m + n), the overall time complexity stays 0(m * n).

Add product of current digits to the previously stored value in result list

keep only the last digit

def multiply(self, num1: str, num2: str) -> str:

Determine the lengths of the input strings

result = [0] * (length num1 + length num2)

for i in range(length num1 - 1, -1, -1):

digit num2 = int(num2[i])

Skip leading zeros in the result list

Convert the result list to string

start_index = 0 if result[0] != 0 else 1

digit num1 = int(num1[i])

length_num1, length_num2 = len(num1), len(num2)

Create a result list to store the product digits

for j in range(length num2 - 1, -1, -1):

answerArrav.unshift(sum % 10);

carry = Math.floor(sum / 10);

if num1 == "0" or num2 == "0":

return answerArray.join('');

return "0"

class Solution:

Time and Space Complexity **Time Complexity**: The time complexity of the given code is 0(m * n), where m and n are the lengths of the input strings num1 and num2 respectively. The code involves a double loop where the outer loop runs m times and the inner loop runs n times, leading to m * n multiplication operations. Additionally, there is a loop for carrying over the values, which runs in 0(m + n) time.

Space Complexity: The space complexity is 0(m + n), as an additional array arr of size m + n is used to store the intermediate results of the multiplication before they are converted to the final string result.