43. MULTIPLY STRINGS

22 June 2024 Saturday
-Kaushal Pratap

In this problem, we will be given two non-negative integers *num1* and *num2* represented as strings. We need to find the multiplication of these two integers without directly converting them to integer using built-in *int()* function.

Example :
"2" x "3" = "6"
(Double quotes here means that these values are actually a string)

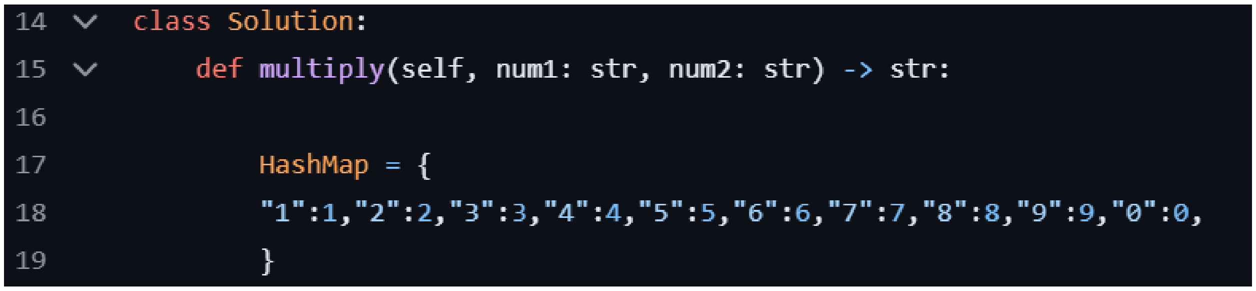
Logic-

As we already know that we can't directly convert them to integer. So, let's discuss it in a step-by-step process.

1. First, we will create our HashMap and map all the string values to its respective integer values from 0 to 9. Like- "1" : 1
2. Then, we'll iterate over every digit in the num1 and num2 string. And at every iteration, we'll be accessing the integer value of its respective string value using HashMap. This will work as a conversion of string into integer without directly using *int()* function.
3. After the conversion of string into integer using HashMap, we will start adding the digits one-by-one using the long integer representation method. For example, we can represent 451 as 400 + 50 + 1.
4. And then, we will have the final integer values of num1 and num2. And now we'll simply return the multiplication of two. But as the question stated that the answer should be string, we will return the string of multiplication using built-in *str()* function.

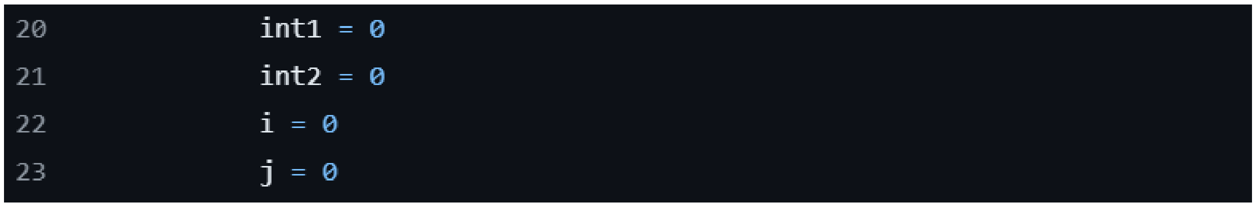
Let's look at the code for better understanding of the algorithm.

Code-



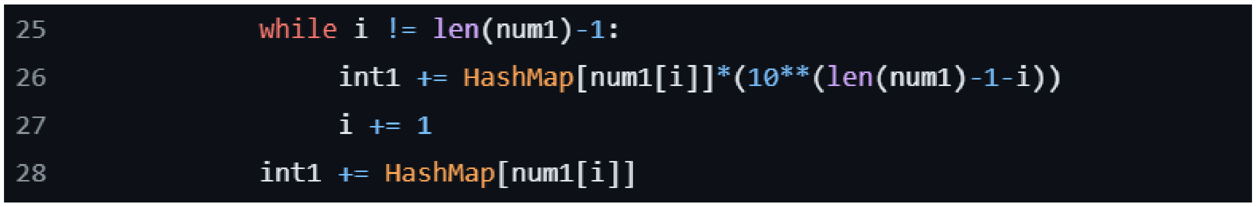
```
14  class Solution:
15      def multiply(self, num1: str, num2: str) -> str:
16
17          HashMap = {
18          "1":1,"2":2,"3":3,"4":4,"5":5,"6":6,"7":7,"8":8,"9":9,"0":0,
19          }
```

- In lines 14 and 15, we will begin by creating our class and function called *multiply()* which takes two parameters *num1* and *num2 as* string.
- Then we will initialize our HashMap from line 17 to 19.



```
20          int1 = 0
21          int2 = 0
22          i = 0
23          j = 0
```

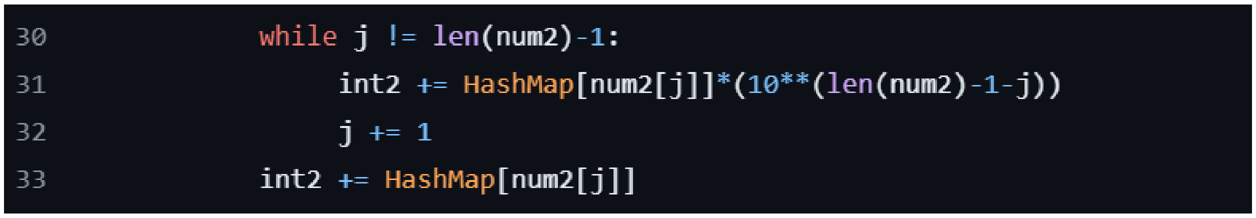
- Here in line 20 and 21, int1 and int2 represents the sum of integers of *num1* and *num2* respectively. Like we can represent 345 in 300 + 40 + 5. In the same way, *int1* and *int2* represents the sum.
- And in line 22 and 23, *i* and *j* variables will be used as index of *num1* and *num2* respectively and for *while loop* conditions.



```
25          while i != len(num1)-1:
26              int1 += HashMap[num1[i]]*(10**(len(num1)-1-i))
27              i += 1
28          int1 += HashMap[num1[i]]
```

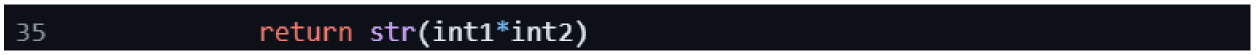
- First, we will understand the algorithm for *num1* because it works same for both the inputs.
- In line 25, we will execute the loop till the *i* (index of *num1*) reaches the last second digit. The motive behind executing till the last second digit is that when you represent an integer in the terms of multiple of 10, the last digit of any integer is only and only multiplied by 1 or we say is simply added. That's why, we're going to access till the last second digit of *num1* because we want the last digit to be added simply.
- In line 26, we will first access the integer value of that particular digit of *num1* from our HashMap. ***num[i]*** will get the string value at *i*th index.
- And ***HashMap[Num[i]]*** will get the respective integer value of string digit at *i*th index.

- Now, we want to multiply the values in the terms of 10s and 100s and keep adding them. Let’s understand this with an example.
- The number 423 have a length of 3. And we will represent it like this- 400 + 20 +3.
- As you can notice the pattern, 4 is multiplied with 100 and 100 have 3-1 zeros. And 20 with 3-2 zeros.
- For continuing this pattern, we will first have 1 unit less length of *num1* i.e. *len(num1) -1* and then we will subtract the *i* also, which is responsible in decreasing the number of zeroes one-by-one.
- And after this step of finding the pattern, that how many zeroes we need, we will make them as exponential of 10.
- After this, we will multiply it with the respective integer value in line 26.
- Also in line 26, after we have an integer in the terms of 10s and 100s, it will add it into *int1* as we’ve already discussed above.
- Line 27 will increment the *i* with 1.
- After the complete execution of *while loop* we will get onto line 28. As we already know that one last digit of integer is not multiplied in the term of 10s and 100s. So, we will simply add it to *int1*. *While loop* will add the values in *int1* in the terms of 10s and 100s and after the end of the loop, it will simply add the last digit.



```
30          while j != len(num2)-1:
31              int2 += HashMap[num2[j]]*(10**(len(num2)-1-j))
32              j += 1
33          int2 += HashMap[num2[j]]
```

- Now, we will do the same task for *num2* which we’ve done for *num1* earlier.



```
35          return str(int1*int2)
```

- After completing all the steps, we’ll have *int1* and *int2* as integers which we’ve converted using our algorithm.
- Now at the end in line 35, we will simply return the product of *int1* and *int2* as a string.

<u>Time complexity analysis-</u>

Line 17-23: Initializing the HashMap and all the other variables take constant time O(1).
Line 25-28: The loop will iterate over the ever digit of *num1* and keep adding it to *int1* which has length *n1* or digits *n1*. So, the time complexity for this operation is O(n1)
Line 30-33: Same as for *num1*. So, its time complexity will be O(n2) where n2 is the length of *num2*.
Line 35: Return works in O(1).

So, the overall time complexity will be O(n1 + n2) which is O(n).