9. Palindrome Number

Solution

Approach #1 Revert half of the number [Accepted]

Intuition

The first idea that comes to mind is to convert the number into string, and check if the string is a palindrome, but this would require extra non-constant space for creating the string which is not allowed by the problem description.

Second idea would be reverting the number itself, and then compare the number with original number, if they are the same, then the number is a palindrome. However, if the reversed number is larger than $\mathrm{int.MAX}$, we will hit integer overflow problem.

Following the thoughts based on the second idea, to avoid the overflow issue of the reverted number, what if we only revert half of the int number? After all, the reverse of the last half of the palindrome should be the same as the first half of the number, if the number is a palindrome.

For example, if the input is `1221`, if we can revert the last part of the number "12**21**" from "**21**" to "**12**", and compare it with the first half of the number "12", since 12 is the same as 12, we know that the number is a palindrome.

Let's see how we could translate this idea into an algorithm.

Algorithm

First of all we should take care of some edge cases. All negative numbers are not palindrome, for example: -123 is not a palindrome since the '-' does not equal to '3'. So we can return false for all negative numbers.

Now let's think about how to revert the last half of the number. For number `1221`, if we do `1221 % 10`, we get the last digit `1`, to get the second to the last digit, we need to remove the last digit from `1221`, we could do so by dividing it by 10, `1221 / 10 = 122`. Then we can get the last digit again by doing a modulus by 10, `122 % 10 = 2`, and if we multiply the last digit by 10 and add the second last digit, `1 * 10 + 2 = 12`, it gives us the reverted number we want. Continuing this process would give us the reverted number with more digits.

Now the question is, how do we know that we've reached the half of the number?

Since we divided the number by 10, and multiplied the reversed number by 10, when the original number is less than the reversed number, it means we've processed half of the number digits.

C#

Complexity Analysis

- Time complexity : $O(log_{10}n)$. We divided the input by 10 for every iteration, so the time complexity is $O(log_{10}n)$
- Space complexity : $O(1)$.

Analysis written by: @ccwei (https://leetcode.com/ccwei)

Join the conversation

Signed in as **Xiaotian_Fu**.

Post a Reply

D (https://discuss.leetcode.com/user/donnyc219)

donnyc219 commented 3 days ago

@Weinb14 (https://discuss.leetcode.com/uid/341256) using extra space means "no matter how the input is, you only use a fixed number of space"? So if you only declare int before the loop, even if your input changes, it only still uses constant space.
However if you declare int array, it may shrink or expand due to your input...
This is what I understand as "extra space". Please point me out if I am wrong....

K (https://discuss.leetcode.com/user/kaashmonee)

kaashmonee commented 5 days ago

@Weinb14 (https://discuss.leetcode.com/uid/341256) My solution is still faulty because it takes O(n) time which is pretty bad. There should be a better way to do it. Also, it takes O(n) space as well if I'm not wrong.

W (https://discuss.leetcode.com/user/weinb14)

Weinb14 commented 6 days ago

In Python you can still convert x into String and check if x is Palindrome Number without declaring any new variables. On the other hand, the solution provided above does declare new variable, yet that is not count as "using extra space"?

D (https://discuss.leetcode.com/user/dmanran)

dmanran commented 6 days ago

where(x > revertedNumber) is very clever! perfect!