166. Fraction to Recurring Decimal (/problems/fraction-to-recurring-decimal/)

March 10, 2016 | 4.4K views

Given two integers representing the numerator and denominator of a fraction, return the fraction in string format.

If the fractional part is repeating, enclose the repeating part in parentheses.

For example,

- Given numerator = 1, denominator = 2, return "0.5".
- Given numerator = 2, denominator = 1, return "2".
- Given numerator = 2, denominator = 3, return "0.(6)".

Credits:
Special thanks to @Shangrila (https://oj.leetcode.com/discuss/user/Shangrila) for adding this problem and creating all test cases.

Summary

This is a straight forward coding problem but with a fair amount of details to get right.

Hints

1. No scary math, just apply elementary math knowledge. Still remember how to perform a *long division*?
2. Try a long division on $\frac{4}{9}$, the repeating part is obvious. Now try $\frac{4}{333}$. Do you see a pattern?
3. Be wary of edge cases! List out as many test cases as you can think of and test your code thoroughly.

Solution

Approach #1 (Long Division) [Accepted]

Intuition

The key insight here is to notice that once the remainder starts repeating, so does the divided result.

$$
\begin{array}{r}
0.16 \\
6\overline{)1.00} \\
\underline{0} \\
10 \\
\underline{6} \\
40 \\
\underline{36} \\
4
\end{array}
$$

$\Leftarrow$ *remainder* $= 1$, mark 1 as seen at *position* $= 0$.

$\Leftarrow$ *remainder* $= 4$, mark 4 as seen at *position* $= 1$.

$\Leftarrow$ *remainder* $= 4$ was seen before at *position* $= 1$, so the fractional part which is 16 starts repeating at *positic*

Algorithm

You will need a hash table that maps from the remainder to its position of the fractional part. Once you found a repeating remainder, you may enclose the reoccurring fractional part with parentheses by consulting the position from the table.

The remainder could be zero while doing the division. That means there is no repeating fractional part and you should stop right away.

Just like the question Divide Two Integers (https://leetcode.com/problems/divide-two-integers/), be wary of edge cases such as negative fractions and nasty extreme case such as $\frac{-2147483648}{-1}$.

Here are some good test cases:

Test case	Explanation
$\frac{0}{1}$	Numerator is zero.
$\frac{1}{0}$	Divisor is 0, should handle it by throwing an exception but here we ignore for simplicity sake.
$\frac{20}{4}$	Answer is a whole integer, should not contain the fractional part.
$\frac{1}{2}$	Answer is 0.5, no recurring decimal.
$\frac{-1}{4}$ or $\frac{1}{-4}$	One of the numerator or denominator is negative, fraction is negative.
$\frac{-1}{-4}$	Both numerator and denominator are negative, should result in positive fraction.
$\frac{-2147483648}{-1}$	Beware of overflow if you cast to positive.

Java

Copy

```java
public String fractionToDecimal(int numerator, int denominator) {
    if (numerator == 0) {
        return "0";
    }
    StringBuilder fraction = new StringBuilder();
    // If either one is negative (not both)
    if (numerator < 0 ^ denominator < 0) {
        fraction.append("-");
    }
    // Convert to Long or else abs(-2147483648) overflows
    long dividend = Math.abs(Long.valueOf(numerator));
    long divisor = Math.abs(Long.valueOf(denominator));
    fraction.append(String.valueOf(dividend / divisor));
    long remainder = dividend % divisor;
    if (remainder == 0) {
        return fraction.toString();
    }
    fraction.append(".");
    Map<Long, Integer> map = new HashMap<>();
    while (remainder != 0) {
        if (map.containsKey(remainder)) {
            fraction.insert(map.get(remainder), "(");
            fraction.append(")");
            break;
        }
        map.put(remainder, fraction.length());
        remainder *= 10;
        fraction.append(String.valueOf(remainder / divisor));
```

Rate this article: **(/ratings/107/19/?return=/articles/fraction-recurring-decimal/) (/ratings/107/19/?return=/articles/f**

Previous (/articles/intersection-two-linked-lists/) Next (/articles/two-sum-ii-input-array-sorted/)

Join the conversation

Login to Reply

H

hiwade commented 3 months ago

what's the space complexity?
(https://discuss.leetcode.com/user/hiwade)

J

jabberwok commented 6 months ago

@Geminiiiiii (https://discuss.leetcode.com/uid/38854) @oyugioho (https://discuss.leetcode.com/uid/183893)
(https://discuss.leetcode.com/user/jabberwok)

I managed to find my solution, it is actually easier than the algorithm I described. You don't even need to compare digits, you just need to store the first remainder of the loop and compare current remainder to it. @oyugioho (https://discuss.leetcode.com/uid/183893) you were right, one-two-stepping would also work, but it will walk over the loop twice.

```
class Solution(object):
    def fractionToDecimal(self, numerator, denominator):
        return fractionToDecimal(numerator, denominator)

def digits_before_loop(den):
    n = den
    digits = 0
    while not n % 10:
        n /= 10
        digits += 1

    while not n % 5:
        n /= 5
        digits += 1

    while not n % 2:
        n /= 2
        digits += 1

    return digits

def list_to_str(l):
    return ''.join(str(d) for d in l)

def fractionToDecimal(n, d):
    if d == 0:
        raise ZeroDivisionError()

    if n == 0:
        return '0'

    if (n < 0) == (d < 0):
        sign = ''
    else:
        sign = '-'
        n = abs(n)
        d = abs(d)

    whole_part = str(n / d)
    n = n % d
    if n == 0:
        return sign + whole_part

    pre_loop_part = []
    loop_part = []

    first_remainder = None
    digits_to_skip = digits_before_loop(d)
```

```
    while True:
        if digits_to_skip == 0:
            first_remainder = n

        n *= 10

        digit = n / d

        if digits_to_skip > 0:
            pre_loop_part.append(digit)
        else:
            loop_part.append(digit)

        digits_to_skip -= 1
        n = n % d

        if first_remainder == n:
            template = '{}{}.{}({})'
            break

        if n == 0:
            template = '{}{}.{}{}'
            break

    return template.format(
        sign, whole_part,
        list_to_str(pre_loop_part),
        list_to_str(loop_part),
    )
```


(https://discuss.leetcode.com/user/manuelp)

ManuelP commented 6 months ago

@jabberwok (https://discuss.leetcode.com/uid/172759) said in Fraction to Recurring Decimal (https://discuss.leetcode.com/post/178535):

> all reminders
> first reminder
> which reminder
> first reminder
> x+y+z reminders

remainders
remainder
remainder
remainder
remainders

J

jabberwok commented 6 months ago

@Geminiiiiii (https://discuss.leetcode.com/uid/38854)
(https://discuss.leetcode.com/user/jabberwok)

I can't read the original article because my subscription ran out, so I can't explain in details what is wrong with it, the only thing I remember that it used HashMap which was a huge overkill performance-wise. My idea was that all fractions are either finite like 0.123 or cyclic like 0.123333...(3) . Cyclic fractions have a part that is not repeated like 0.12 in my example and the part that is getting repeated: 3. So my solution was

- Determine length of non-cyclic part
- Now we guaranteed to know the first digit of the cycle
- Keep dividing until we hit the first digit of the cycle, then compare next known digit with the new digit from the division. Repeat until you either have different digits or you hit the digit you started with. In the first case - continue, in the second case you got a hit.

The first point was the tricky one so I came up with this formula:

`A / B`, where B can be represented as `B = 10^x * 5^y * 2^z * rest` then you add `x + y + z` and you get how many digits come before cycle.

Example:

```
1 / 280= 0.00357142857142857142857142857...
```

```
280 = 10^1 * 5^0 * 2^2 * 7
```

```
x + y + z = 1 + 0 + 2 = 3
```

It means that the cycle starts at the 4th digit - 5

```
0.00357142857142857142857142857
-----^
```

We go forward until we hit 5 and store the index of that position 10 (potential cycle end)

```
0.00357142857142857142857142857
----------^
```

We continue going forward and compare digits: 5, 7, 1, 4, 2, 8

```
0.00357142857142857142857142857
----------^-----^ and so on
```

and we hit the position 10 (potential cycle end) on the left. The loop is closed, we detected a cycle. So the result shall be

```
0.003(571428)
```

O

oyugioho commented 6 months ago

I think he is referring to the technique of loop detection that we all know how to do for the question "Find the node where the loop starts for a singly linked list". Basically, try to get reminders two at a time for one and one at a time for another. When they are equal, we detect a loop, etc.
(https://discuss.leetcode.com/user/oyugioho)

G

Geminiiiiii commented 6 months ago

@jabberwok (https://discuss.leetcode.com/uid/172759) Can you explain more details?
(https://discuss.leetcode.com/user/geminiiiiii)

J

jabberwok commented last year

(https://discuss.leetcode.com/user/jabberwok)

You can avoid using a map. You don't need all reminders to test for a cycle, you need only the first reminder of the cycle. For example, if you do 1 / 3 result is 0.(3). Now the tricky part - how to figure out which reminder is first in the cycle, because you can have 1 / 6 = 0.1(6), 1 is outside of the cycle. Think that numerator = 10^x * 2^x * 5^z * rest, where x, y, z can be 0. To get the first reminder of the cycle you need to skip x+y+z reminders. It works. This way you skip using HashMap because using HashMap is MAGNITUDES slower than integer operations, this code spends 90% if not more on HashMap calls.

View original thread (https://discuss.leetcode.com/topic/42)

Load more comments...