Solution: Fraction to Recurring Decimal

Let's solve the Fraction to Recurring Decimal problem using the Hash Map pattern.

We'll cover the following

Statement

Given the two integer values of a fraction, `numerator` and `denominator`, implement a function that returns the fraction in string format. If the fractional part repeats, enclose the repeating part in parentheses.

Constraints:

- `denominator` $! = 0$
- $-2^{31} \leq$ `numerator`, `denominator` $\leq 2^{31} - 1$

Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

A naive approach is to use an array to store the remainders to determine the repetition of remainders. We can detect repetitions by checking if the remainder already exists in the array during each calculation. This approach involves using a nested loop, with the outer loop calculating the remainder and the inner loop searching for the remainder in the array.

The time complexity of this approach is $O(|d|^2)$, where d is the number of digits in the denominator. This is because we're using a nested loop. Let's see if we can use the hash map pattern to find a faster solution.

Optimized approach using hash maps

To solve this problem, we can use the hash map pattern to develop a faster solution. The idea is to use a hashmap to store the remainder, and every time we calculate the remainder, we can check if the remainder is already in the hash map in $O(1)$ time complexity.

> **Note**: In the following section, we will gradually build the solution. Alternatively, you can skip straight to just the code.

Step-by-step solution construction

Let's start with the initial conditions:

- If the numerator is zero, simply return `0`.
- If either of the numerator or denominator is negative, append the minus character (`-`) to the `result` string, and make the numerator and denominator positive.

Java

```
class FractionToDecimal{

    public static String fractionToDecimal(int numerator, int denominator)
    {
        String result = "";
        HashMap<Integer, Integer> remainderMap = new HashMap<Integer, Integer>();
        System.out.println("\n\tnumerator: " + numerator);
        System.out.println("\tdenominator: " + denominator);
        // if numerator is 0, then return 0 in the string
        if(numerator == 0)
        {
            System.out.println("\tAs numerator is 0, so return 0.");
            return "0";
        }
        // If numerator or denominator is negative, then append "-" to the result
        if(numerator < 0 ^ denominator < 0)
        {
            System.out.println("\tAs numerator or denominator is negative. Appending - to the result");
            result += '-';
        }
        // Make numerator and denominator positive after adding "-" to the result
        numerator = Math.abs(numerator);
        denominator = Math.abs(denominator);
        return result;
    }
    // Driver code
    public static void main(String[] args){
        int [][]inputs = {{0, 4}, {4, 2}, {5, 333}, {2, 3}, {47, 18}, {93, 7}, { 5, 333}, {47, 18}, {4
```

Fraction to Recurring Decimal

We calculate the quotient and remainder of the division and then perform the following steps:

- Append the quotient of the division to the `result` string.
- If the remainder is 0, return the `result` string, since the denominator completely divides the numerator.
- Otherwise, if the remainder is not 0, append the dot (`.`) character to the `result` string.

Java

```
class FractionToDecimal{

    public static String fractionToDecimal(int numerator, int denominator)
    {
        String result = "";
        HashMap<Integer, Integer> remainderMap = new HashMap<Integer, Integer>();
        // if numerator is 0, then return 0 in the string
        if(numerator == 0)
        {
            return "0";
```

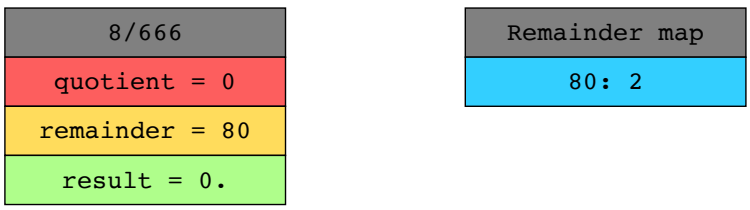
```
        }
        // If numerator or denominator is negative, then append "-" to the result
        if(numerator < 0 ^ denominator < 0)
        {
            result += '-';
        }
          // Make numerator and denominator positive after adding "-" to the result
        numerator = Math.abs(numerator);
        denominator = Math.abs(denominator);
        // Calculate the quotient
        int quotient = numerator / denominator;
        System.out.println("\n\tquotient: " + quotient);
        int remainder = (numerator % denominator) * 10;
        System.out.println("\tremainder: " + remainder);
        // Append the quotient part in the result
        result += Integer.toString(quotient);
        System.out.println("\tresult: " + result);
        if(remainder == 0)
```

Fraction to Recurring Decimal

Start a loop until the remainder is 0 and every time check, if the remainder exists in the keys of the hash map:

- If it does, do the following:
  - Store the value of the corresponding key in a variable, `beginning`. This variable will store the starting index of the recurring part of the decimal from the `result` string.
  - Store the non-recurring part of the `result` variable in the `left` variable. This is done by slicing the `result` string from the index `0` to the index `beginning`.
  - Store the recurring part of the `result` variable in the `right` variable. This is done by slicing the `result` string from the index `beginning` to its last index.
  - Update the `result` variable by storing the following concatenated string in it: `left` + `'('` + `right` + `')'`.
- Otherwise, if the remainder is not present in the hashmap, do the following:
  - Insert the remainder (key) and the current length of the `result` string (value) in the hash map.
  - Calculate the new quotient and remainder by dividing the current remainder by the denominator.
  - Append the quotient to the `result` string.

The slides below illustrate how we'd like the algorithm to run:



quotient = numerator / denominator = 8/666 = 0

remainder = (numerator % denominator) * 10 = 8%666 *10 = 80

Since quotient is 0, append it to the result. Next, append "." to the result. Since the remainder is not in the hash map, put it in the hash map with its length.

For the first time, the quotient and remainder will be calculated using the formula above. For the next iterations, it will be changed.

Java

```java
import java.util.*;

class FractionToDecimal {
    public static String fractionToDecimal(int numerator, int denominator) {
        String result = "";
        HashMap<Integer, Integer> remainderMap = new HashMap<Integer, Integer>();
        System.out.println("\n\tnumerator: " + numerator);
        System.out.println("\tdenominator: " + denominator);

        // if numerator is 0, then return 0 in the string
        if (numerator == 0) {
            System.out.println("\tAs numerator is 0, so return 0.");
            return "0";
        }

        // If numerator or denominator is negative, then append "-" to the result
        if (numerator < 0 ^ denominator < 0) {
            System.out.println("\tAs numerator or denominator is negative. Appending - to the result");
            result += '-';
        }

        // Make numerator and denominator positive after adding "-" to the result
        numerator = Math.abs(numerator);
        denominator = Math.abs(denominator);

        // Calculate the quotient
        int quotient = numerator / denominator;
        System.out.println("\n\tquotient: " + quotient);
```

Fraction to Recurring Decimal

Just the code

Here's the complete solution to this problem:

Java

```java
import java.util.*;

class FractionToDecimal {

    public static String fractionToDecimal(int numerator, int denominator) {
        String result = "";
        HashMap<Integer, Integer> remainderMap = new HashMap<Integer, Integer>();
        if (numerator == 0) {
            return "0";
        }

        if (numerator < 0 ^ denominator < 0) {
            result += '-';
        }

        numerator = Math.abs(numerator);
        denominator = Math.abs(denominator);
        int quotient = numerator / denominator;
        int remainder = (numerator % denominator) * 10;
        result += Integer.toString(quotient);

        if (remainder == 0)
            return result;
        else {
            result += ".";
            while (remainder != 0) {
```

```
27                  if (remainderMap.containsKey(remainder)) {
28                      int beginning = remainderMap.get(remainder);
```

Fraction to Recurring Decimal

Solution summary

1. If the numerator is zero, simply return 0.
2. If either of the numerator or denominator is negative, append the minus character (-) to the result string, and make the numerator and denominator positive.
3. Calculate the quotient and the remainder. Append the quotient part to the result.
4. Check if the remainder is 0, then return the result.
5. If the remainder is not 0, append the dot (`'.'`) character to the result.
6. Start a loop until the remainder is 0, and every time check the remainder in the hash map. If the remainder already exists in the hash map, then make the recurring decimal from the fraction. If the remainder does not exist in the hash map, put it in the hash map.

Time complexity

The time complexity of the above solution is $O(|d|)$, where d is the number of digits in the denominator. The algorithm will have at most $|d|$ different remainders. Therefore, $|d|$ iterations of the loop will be completed in the worst case.

Space complexity

The space complexity of the above solution is $O(|d|)$. The algorithm will have at most $|d|$ different remainders. So $|d|$ entries will be filled in the hash map in the worst case.

Back
Fraction to Recurring ...

Next
Logger Rate Limiter

Mark as Completed