972. Equal Rational Numbers

Problem Explanation

The problem states that you have two strings, which represent a non-negative rational number. The task is to figure out if both these strings represent the same number or not. Basically, the problem requires comparing two rational numbers which are represented in string format. The rational number itself is expressed in three different ways.

1. Only the integer part. For example - "123"
2. An integer followed by a decimal and a non-repeating part. For example - "123.456"
3. An integer followed by a decimal, a non-repeating part and a repeating part in parentheses. For example - "123.456(789)"

We need to evaluate these strings and compare their floating numbers. Due to the repeating part, the evaluation of the third type of number can be tricky.

Taking an example - Let's say the two strings are "0.1(6)" and "0.1666(6)".

String 1 -> "0.1(6)": This represents the number 0.1666... String 2 -> "0.1666(6)": As already mentioned this represents 0.1666..., so both represent the same number.

Therefore the evaluation is True.

Solution Approach

The approach for the solution for this problem involve converting both strings into floating numbers and then comparing them. The solution uses simple string manipulation methods to convert these strings into actual float numbers.

The function `valueOf()` computes the floating number represented by the string. We get the slice of the string before the occurrence of parentheses (if any) and convert it to a double number. Then, calculate the repeating part of the string (if any), multiply it by the appropriate ratio according to its length, and add it to the previous double number. Finally, we compare the float values by checking if the absolute difference is less than a very small number (1e-9) as we're dealing with floating point precision.

Implementation

C++

```
cpp
class Solution {
 public:
  bool isRationalEqual(string s, string t) {
    return abs(valueOf(s) - valueOf(t)) < 1e-9;
  }

 private:
  const static vector<double> ratios{1.0, 1.0 / 9, 1.0 / 99, 1.0 / 999,
                                     1.0 / 9999};

  double valueOf(const string& s) {
    if (s.find('(') == string::npos)
      return stod(s); // stod function converts string to double

    const int leftParenIndex = s.find first of('(');
    const int rightParenIndex = s.find first of(')');
    const int dotIndex = s.find_first_of('.');

    const double integerAndNonRepeating = stod(s.substr(0, leftParenIndex)); // Get the non-repeating string part ar
    const int nonRepeatingLength = leftParenIndex - dotIndex - 1;

    const int repeating = stoi(s.substr(leftParenIndex + 1, rightParenIndex - leftParenIndex - 1)); // Get the repea
    const int repeatingLength = rightParenIndex - leftParenIndex - 1;
    return integerAndNonRepeating +
           repeating * pow(0.1, nonRepeatingLength) * ratios[repeatingLength]; // Compute the total number by adding
  }
};
```

Python

```
python
class Solution:
    def isRationalEqual(self, s: str, t: str) -> bool:
        def value(s):
            if '(' not in s:
                return float(s)
            i = s.index('(')
                nonRepeating, repeating = s[:i], s[i + 1:-1]
                return float(nonRepeating) + float(repeating) / (10 ** len(repeating) - 1) / 10 ** (i - s.index('.')
        return abs(value(s) - value(t)) < 1e-9
```

In this code, we first define a function `value()` which will convert the string into a floating number. If there's no repeating part in the string, Python's inbuilt function is sufficient and directly converts the string to a floating number. However, if it contains a repeating part, we divide the section inside the parentheses by the number of nines equal to the length of the repeating string and the appropriate power of ten. Finally, we subtract the resulting floating number from `s` and check if it's less than a very small number (1e-9)

JavaScript

```
javascript
const isRationalEqual = (s, t) => {
  return Math.abs(valueOf(s) - valueOf(t)) < 1e-9;
}

const valueOf = (s) => {
  let leftParenIndex = s.indexOf('(');
  if (leftParenIndex === -1) {
      return parseFloat(s);
  }
  let rightParenIndex = s.indexOf(')');
  let dotIndex = s.indexOf('.');
  let nonRepeating = parseFloat(s.substring(0, leftParenIndex));
  let nonRepeatingLength = leftParenIndex - dotIndex - 1;
  let repeating = parseInt(s.substring(leftParenIndex + 1, rightParenIndex));
  let repeatingLength = rightParenIndex - leftParenIndex - 1;
  return nonRepeating +
      repeating * Math.pow(0.1, nonRepeatingLength) * (1.0 / (Math.pow(10, repeatingLength) - 1))
}
```

In JavaScript, we're implementing a similar strategy. If string `s` doesn't contain a repeating part, we parse it as a float. If it does, we slice the non-repeating and the repeating part of the string. The non-repeating part is parsed to a float and the repeating part to an int. Then, we calculate the sum of the non-repeating part plus the repeating part divided by `10 ^ repeatingLength - 1`.

Java

```
java
class Solution {
    public boolean isRationalEqual(String S, String T) {
        return Math.abs(value(S) - value(T)) < 1E-9;
    }

    public double value(String S) {
        int i = S.indexOf("(");
        if (i > 0) {
            String base = S.substring(0, i);
            String rep = S.substring(i + 1, S.length() - 1);
            for (int j = 0; j < 20; ++j) base += rep;
            return Double.valueOf(base);
        } else {
            return Double.valueOf(S);
        }
    }
}
```

In the Java solution, we convert the string to a double value. If the string includes a repeating part, we append the repeating part 20 times to the non-repeating part to approximate the repeating decimal.