2264. Largest 3-Same-Digit Number in String

Easy String

Leetcode Link

Problem Description

In this problem, we are provided with a string `num` that represents a large integer. We need to determine if there exists a 'good' integer within this string. A 'good' integer is defined by two specific criteria:

1. It must be a substring of `num` with exactly a length of 3 characters.
2. All three characters of this substring must be the same digit, meaning it consists of only one unique digit.

The task is to return the largest such 'good' integer in string format. If no such 'good' integer exists, we are required to return an empty string `""`.

A 'substring' is defined as a continuous sequence of characters within the string. It's also important to note that 'good' integers can contain leading zeroes, as they are part of the string `num`.

The problem is essentially asking us to look for triplets within the string where all three characters are the same and return the highest numerical value triplet. For example, if single-digit triplets '222', '555', and '777' are all found in `num`, then we should return '777' as it represents the largest good integer.

Intuition

The solution employs a straightforward approach. Since we are asked for the largest good integer, we can employ a strategy where we start looking for triplets from the largest digit (9) and work our way down to the smallest digit (0).

The key insights that lead us to this solution are:

- Given that we're looking for sequences of the same digit, we know that '999' is the largest possible good integer and '000' is the smallest.
- The problem statement defines that a good integer has a fixed length of 3. Hence, we don't need to consider substrings of any other length.
- If a larger triplet is found, we don't need to continue searching since we are guaranteed that no larger good integer can exist.
- Due to the ordering of digits (9 to 0), the first matching triplet we come across will be the largest one, and we can terminate the search immediately and return the result.

Taking these points into consideration, the algorithm iterates over digits from 9 to 0 and checks if a substring of three similar digits exists in `num`. As soon as such a substring is found, the algorithm stops and we return the substring. If no such triplet is found after checking all digits, we conclude that no good integer is present, and we return an empty string.

Solution Approach

The solution approach utilizes a simple but effective strategy which is a countdown loop that checks for the presence of triplets of the same digit within the input string `num`. The approach is essentially a search problem with a greedy method for finding the maximum good integer by checking the highest possible digits first.

Algorithm:

We iterate over the possible digits in descending order, starting from 9, since we want to find the maximum good integer. This is a greedy choice since we aim to return the largest possible triplet of digits.

For each digit `i` in the range from 9 down to 0, we do the following:

1. We create a string `s` that consists of three identical characters of the digit `i`. This is achieved by multiplying the string representation of `i` by 3 (i.e., `s = str(i) * 3`).
2. Check if the string `s` is a substring of `num`. To do this, we use the `in` keyword in Python which checks for substring existence.
3. If `s` is found within `num`, we immediately return `s` as we have located the largest possible good integer up to that point.
4. If no match is found after checking all digits down to 0, the function returns an empty string, indicating no good integer exists in `num`.

Data Structures:

No additional data structures are used in this approach since we are directly checking substrings within the given string and there is no need for auxiliary storage.

Patterns:

This approach uses a **greedy algorithm** pattern, where the solution builds the optimum choice step by step. In this case, the optimal choice at each step is the largest digit that forms a good integer. By starting from 9 and working downwards, we ensure that the first matching substring is also the largest possible one.

Here is the code block which represents our solution approach:

```
class Solution:
    def largestGoodInteger(self, num: str) -> str:
        for i in range(9, -1, -1):  # Loop over digits from 9 to 0
            if (s := str(i) * 3) in num:  # Check if the triplet is in num
                return s  # Return the largest good integer found
        return ""  # Return an empty string if no good integer exists
```

The simplicity of this solution lies in the fact that it combines enumeration with a greedy strategy to minimize the number of checks required. It doesn't require complex algorithms or data structures, making it both easy to understand and efficient to execute.

Example Walkthrough

Let's consider the input string `num = "4233312221555"`. We are tasked with finding the largest 'good' integer. A 'good' integer is a triplet of the same digit.

Step 1: We begin by checking for the triplet '999'. Since '999' does not appear in our input string, we move on to the next digit.

Step 2: We then check for the triplet '888'. This triplet also does not appear in our input string.

Step 3: We continue this process with '777', '666', and '555'. When we check for '555', we find that it does exist in the input string '4233312221555'.

Step 4: We return '555' immediately since it is the largest 'good' integer that can be found by our search and we do not need to check for triplets of any smaller digit. There is no need to search for '444', '333', '222', or '111', let alone '000', because we are guaranteed they won't give us a larger 'good' integer than what we've already found ('555').

Therefore, using our solution approach, for the input `num = "4233312221555"`, the largest good integer is '555'.

Python Solution

```
class Solution:
    def largestGoodInteger(self, num: str) -> str:
        # Iterate over the digits in descending order, starting from 9 to 0.
        for digit in range(9, -1, -1):
            # Create a string of three consecutive identical digits.
            triple_digit_str = str(digit) * 3

            # Check if this string of three identical digits is in the input 'num'.
            if triple_digit_str in num:
                # If found, return this largest 'good' integer as the result.
                return triple_digit_str

        # If no such triple is found in the number, return an empty string.
        return ""
```

Java Solution

```
class Solution {
    // Method to find the largest good integer in the given string.
    public String largestGoodInteger(String num) {
        // Loop backwards from '9' to '0'
        // This will ensure that the first good integer found is the largest
        for (int i = 9; i >= 0; i--) {
            // Create a string consisting of three identical digits
            String tripleDigit = String.valueOf(i).repeat(3);

            // Check if the num string contains the tripleDigit
            if (num.contains(tripleDigit)) {
                // Return the found good integer (since we started from 9, it's the largest)
                return tripleDigit;
            }
        }
        // If no good integer is found, return an empty string
        return "";
    }
}
```

C++ Solution

```
class Solution {
public:
    // Function to find the largest 3-digit substring with identical digits
    string largestGoodInteger(string num) {
        // Start checking from '9' down to '0' to find max "good" integer snippet
        for (char digit = '9'; digit >= '0'; --digit) {
            // Create a string consisting of three identical characters (digits)
            string triplet(3, digit);

            // If the triplet is found in 'num', return it as it's the largest "good" integer
            if (num.find(triplet) != string::npos) {
                // Return the first instance of the largest "good" integer found
                return triplet;
            }
        }
        // If no triplet found, return an empty string
        return "";
    }
};
```

Typescript Solution

```
// This function finds the largest "good integer" in the given number string.
// A "good integer" is a substring that consists of three identical consecutive digits.
function largestGoodInteger(num: string): string {
    // Iterate from the largest possible digit ('9') to the smallest ('0').
    for (let currentDigit = 9; currentDigit >= 0; currentDigit--) {
        // Create a string with the current digit repeated three times.
        const tripleDigit = String(currentDigit).repeat(3);
        // Check if the current triple digit string exists in the input number string.
        if (num.includes(tripleDigit)) {
            // If found, return the current triple digit string as it is the largest "good integer" found so far.
            return tripleDigit;
        }
    }
    // If no "good integer" is found, return an empty string.
    return '';
}
```

Time and Space Complexity

The given Python code snippet defines a function `largestGoodInteger` that searches for the largest 3-digit same numeric substring within a given string `num`. The function loops decrementally from `9` to `0`, constructs a string `s` of three identical digits, and checks whether this string is contained within the input `num`.

Time Complexity

The time complexity of the provided code can be analyzed as follows:

- The `for` loop runs from `9` to `0`, which is `10` iterations.
- Within each iteration, the `in` operator is used to check for the presence of a 3-digit substring `s` within the string `num`. The `in` operator, in the worst case, has to check each character in `num` against `s`, leading to an `O(n)` operation for each check, where `n` is the length of `num`.
- Since we perform this check for `10` potential substrates (`999`, `888`, ..., `000`), the overall time complexity is `O(10 * n)`.

Thus, the time complexity is indeed `O(10 * n)` as mentioned in the reference answer.

Space Complexity

For space complexity, we consider the following:

- The variable `s` is created within the loop, consisting of a 3-character string. This space is reused and does not depend on the input size.
- No additional data structures or dynamic memory allocations are used that grow with the size of the input.
- The input `num` and the constant 3-character string `s` do not count towards additional space since the space complexity analysis accounts for only the extra space required by the algorithm beyond the input.

Therefore, the space complexity is `O(1)`, which means that the amount of memory required does not change with the size of the input string `num`.