660. Remove 9

Leetcode Link

Problem Description

You are to create a sequence of integers, but with a twist. When counting up from 1, any number containing a 9 should be skipped. For instance, you would go from 8, to 10, to 11, skipping 9. The sequence will therefore look like: 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, ..., 18, 20, ..., etc. You are given an integer n, and your task is to return the n-th number in this sequence.

For example, if you're asked for the 9th number in the sequence, the sequence would look as follows: 1, 2, 3, 4, 5, 6, 7, 8, 10. Hence, you would return 10.

Solution Approach

The solution to this problem lies in number systems. Specifically, this problem mirrors counting in base 9, where there are only digits from 0-8. This is because our problem essentially forms a new sequence excluding 9, so the count is similarly performed in base 9.

To explain this further with an example, let's take n = 10. When counted in base 10 (which is the normal counting system we use), the 10th integer is 10. However, in our problem, we would need to exclude the number 9 (integers containing 9) from the sequence. If we count in base 9, the 10th integer is 11 (because 9 is not counted).

Python Solution

```
python
class Solution:
    def newInteger(self, n: int) -> int:
        ans = ""
        while n:
            ans = str(n % 9) + ans
            n //= 9
        return int(ans)
```

Java Solution

```
java
class Solution {
    public int newInteger(int n) {
        String ans = "";
        while (n > 0) {
            ans = (n % 9) + ans;
            n /= 9;
        }
        return Integer.parseInt(ans);
    }
}
```

Javascript Solution

```
javascript
class Solution {
    newInteger(n) {
        let ans = "";
        while (n) {
            ans = (n % 9).toString() + ans;
            n = Math.floor(n / 9);
        }
        return parseInt(ans);
    }
};
```

C++ Solution

```
cpp
class Solution {
public:
    int newInteger(int n) {
        string ans = "";
        while (n) {
            ans = to_string(n % 9) + ans;
            n /= 9;
        }
        return stoi(ans);
    }
};
```

C# Solution

```
csharp
public class Solution {
    public int NewInteger(int n) {
        string ans = "";
        while (n > 0) {
            ans = (n % 9).ToString() + ans;
            n /= 9;
        }
        return Int32.Parse(ans);
    }
}
```

In all these solutions, we're effectively transforming the number form base 10 to base 9 by replacing the digit in the tenth place with the remainder of the division by 9, and then updating the number to be the quotient of the division by 9. We keep doing this in a loop until our original number is reduced to 0. The final answer is obtained by converting the string representation back to integer.The implementations of this algorithm are fairly straightforward across different programming languages. The differences are mainly related to specific language syntax and functionalities.

For example, in Python, the "//" operator is used for integer division. In Java and C#, integer division is the default, and the "%" operator is used for getting the remainder of the division. In Javascript, a separate function Math.floor() is used for integer division.

One important aspect to note in these implementations is how we do the conversion from integers to strings and then back to integers. In Java, C#, and Javascript, we can use the built-in functions Integer.parseInt(), Int32.Parse(), and parseInt() respectively for converting the string representation back to an integer.

During implementation, we don't have to consider the number 9 while generating the sequence. This is due to the property of base 9 numbers. Hence this solution saves us from explicitly checking for numbers containing the digit 9. This would make the solution extremely efficient even for large input values of n.

Finally, note that we concatenate the remainder string on the left of the answer string. This is because the least significant digit in the base 9 representation is obtained first in our algorithm, and we need to append the next digits on its left, not its right.

In conclusion, this problem illustrates how a simple change in perspective - from base 10 to base 9 - can lead to an elegant and efficient solution. By harnessing the power of number theory, we effectively solve this problem with minimal computation and without explicitly checking for digits.