1323. Maximum 69 Number

Easy Greedy Math

Leetcode Link

Problem Description

You are provided with a positive integer named `num` which contains only the digits `6` and `9`. Your task is to determine the highest possible number you can create by changing at most one digit. This means you can change one `6` to a `9`, or vice versa, but it is important to note that changing a `9` to a `6` is not optimal for obtaining the maximum number. Therefore, you should look for the first occurrence of the digit `6` and change it to `9`. If the number contains only `9`s, then it is already the highest possible number and can be returned as it is.

Intuition

The intuition behind the solution is to form the largest possible number by changing at most one digit from `6` to `9`. To maximize the number, you should aim to change the most significant `6` (the one that appears first when reading the number from left to right) because:

- Changing the first `6` to `9` will add the most value to the overall number.
- There's no need to consider changing `9` to `6`, as this will only decrease the number.

Here's the step-by-step approach to arrive at the solution:

1. Convert the integer `num` to a string so that we can employ string manipulation techniques.
2. Use the `replace` method of a string which allows us to specify how many occurrences of a substring we want to replace. In this case, we want to replace only the first occurrence of `6` with `9`.
3. Once the first `6` has been replaced (or if there are no `6`s), convert the string back to an integer.
4. Return the new integer which is now the maximum number we can get by changing at most one `6` to a `9`.

This is a simple and efficient solution as it performs all the operations in-place without the need for additional data structures or iterating through all the digits of the number.

Solution Approach

The solution provided uses basic string manipulation and type conversion techniques to solve the problem. The implementation does not explicitly require the use of complex algorithms, data structures, or patterns. Instead, it relies on a straightforward approach leveraging Python's built-in string and integer functionalities. Here's how the solution works:

1. The integer `num` is converted into a string to access the replace method: `str(num)`. This is necessary because in Python, strings are iterable and we can perform operations like find or replace on them, but we cannot do that directly on integers.

2. The string method `replace` is used to replace the first occurrence of "6" with "9". By specifying the third parameter of the replace method as `1`, it ensures that at most one digit is changed, which adheres to the problem's constraints.

   ```
   str(num).replace("6", "9", 1)
   ```

3. The `replace` method would return a new string with the first "6" changed to "9". If the original number does not contain any "6", the string remains unchanged.

4. The modified string which represents our maximum number is then converted back to an integer using `int()`.

5. The result, now as an integer, is then returned as the final output of the function.

This concise solution is made possible because of Python's powerful standard library, which allows tasks like string replacement and type casting to be done in a single line of code. The complexity of the solution is primarily O(n) where n is the number of digits in the `num`, as `replace()` in worst-case will scan the entire string. However, since we are only replacing at most one occurrence, its performance is likely to be very close to O(1) for practically sized integers.

Example Walkthrough

Let's go through an example to demonstrate the solution approach using a small example.

Suppose the provided positive integer `num` is `9669`.

Following the solution steps:

1. Convert the integer to a string: str(9669) results in the string "9669".

2. Use the string method `replace` to replace the first occurrence of "6" with "9":

   ```
   "9669".replace("6", "9", 1)
   ```

   This method will scan the string from left to right and upon finding the first "6", it will replace it with "9", resulting in the string "9969". The `1` as the third argument in the replace method ensures that only the first occurrence is replaced.

3. The new string "9969" is then converted back into an integer using `int()`:

   ```
   int("9969")
   ```

   This results in the integer 9969.

4. The resulting integer 9969 is returned as it is now the maximum number that can be achieved by changing at most one "6" to "9".

In this example, the initial `num` is `9669`, and by applying the steps outlined in the solution approach, the function would return `9969` as the highest possible number after changing the first `6` to a `9`. This approach ensures that the added value is maximized, adhering to the goal of making the number as large as possible with a single-digit change.

Python Solution

```python
class Solution:
    def maximum69Number(self, num: int) -> int:
        # Convert the integer to a string to manipulate individual digits
        num_str = str(num)

        # Replace the first occurrence of '6' with '9' in the string
        # This ensures we make the maximum number by changing the leftmost '6'
        # Use only one replacement to maximize the number
        modified_num_str = num_str.replace('6', '9', 1)

        # Convert the modified string back to an integer and return it
        return int(modified_num_str)

# Example of usage:
# solution = Solution()
# print(solution.maximum69Number(9669)) # Output: 9969

```

Java Solution

```java
class Solution {
    public int maximum69Number(int num) {
        // Convert the integer num to a String
        String numberAsString = String.valueOf(num);

        // Replace the first occurrence of the character '6' with '9' in the string
        String modifiedNumber = numberAsString.replaceFirst("6", "9");

        // Convert the modified string back to an integer and return it
        return Integer.valueOf(modifiedNumber);
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // This method maximizes the value of the given number by changing at most
    // one digit - change the first '6' to a '9'.
    int maximum69Number(int num) {
        // Convert the integer to a string to manipulate individual characters
        string numStr = to_string(num);

        // Iterate over the characters in the string
        for (char& ch : numStr) {
            // If the character is '6', change it to '9'
            if (ch == '6') {
                ch = '9';
                // After the first change, break out of the loop since we are
                // allowed to change at most one digit
                break;
            }
        }

        // Convert the string back to an integer to obtain the final result
        return stoi(numStr);
    }
};

```

Typescript Solution

```typescript
/**
 * Converts the first occurrence of the digit '6' to '9' in the given number
 * if it has '6' and returns the maximum number possible.
 * If there is no '6', the original number is returned.
 *
 * @param {number} num The original number to be converted.
 * @return {number} The maximum number obtained after conversion.
 */
function maximum69Number(num: number): number {
    // Convert the number to a string to utilize string replacement
    const numStr: string = num.toString();

    // Replace the first occurrence of '6' with '9' in the string
    const convertedNumStr: string = numStr.replace('6', '9');

    // Convert the modified string back to a number and return
    return Number(convertedNumStr);
}

```

Time and Space Complexity

The time complexity of the code is `O(n)` where `n` is the number of digits in the input number. This is because the `str` function is used to convert the integer to a string which takes `O(n)` time, and the `replace` function also takes `O(n)` time to scan the string and replace the first occurrence of "6" with "9".

The space complexity is also `O(n)`, since we're creating a new string when we convert the number to its string representation. The `replace` method does not increase the space complexity as it still generates a string of the same order of size `n`.