2729. Check if The Number is Fascinating

Easy | Hash Table | Math

Problem Description

The given LeetCode problem defines a fascinating number as a three-digit integer `n` such that when you concatenate `n`, `2*n`, and `3*n`, the resulting nine-digit number contains every digit from 1 to 9 exactly once, with no zeros present. The task is to determine if a given number is fascinating according to this definition. If it is, the function should return `true`, otherwise `false`.

Intuition

To solve this problem, we can follow a straightforward approach:

1. Calculate `2*n` and `3*n` to find the products we need to concatenate with `n`.
2. Concatenate the string representations of `n`, `2*n`, and `3*n` in order.
3. Sort the concatenated string. For `n` to be fascinating, this sorted string must be "123456789" since a fascinating number contains all digits from 1 to 9 exactly once.
4. The final step is to compare the sorted string with "123456789" and return the result. If they are equal, `n` is fascinating and we return `true`; if not, return `false`.

The implementation in the provided code directly translates this reasoning into a simple one-liner function, where the sorting of concatenated strings is succinctly compared with the string "123456789".

Solution Approach

The implementation of the solution to determine if a number `n` is fascinating involves a simple algorithm without the need for complex data structures.

Here's the breakdown of the algorithm:

1. **String Conversion**: The first step is to convert the original number `n` and its multiples `2*n` and `3*n` into strings. This allows for easy concatenation.
2. **Concatenation**: The next step involves concatenating the string representations of `n`, `2*n`, and `3*n`. Concatenation is the process of joining strings end to end. In Python, this is done using the `+` operator.
3. **Sorting**: After concatenating, the algorithm sorts the characters in the resulting string. Sorting rearranges the characters so that when compared to the string "123456789", we can easily check if all digits are present exactly once and in the correct order.
4. **Comparison**: The sorted string is then compared to "123456789". For the number `n` to be fascinating, these two strings must be identical.

The code uses the following Python features:

- `str()` for converting an integer to a string.
- `sorted()` for sorting the characters in the string.
- `''.join()` for joining the sorted list of characters back into a string.
- `==` operator to compare two strings for equality.

The line `return "".join(sorted(s)) == "123456789"` executes the sorting and joining operations and compares the result with "123456789" in a single, concise expression. This line is the crux of the solution, utilizing Python's ability to chain operations in a readable and efficient manner.

The reason we do not need any more complex data structures or algorithms here is due to Python's powerful built-in functions which take care of the heavy lifting in the background, like sorting and string manipulation.

Example Walkthrough

Let's use the number `192` as an example to illustrate the solution approach.

1. **String Conversion**: Convert the number `192` and its multiples by `2` and `3` into strings.
   - `n` = 192 → `'192'`
   - `2*n` = 384 → `'384'`
   - `3*n` = 576 → `'576'`
2. **Concatenation**: Concatenate the strings of `n`, `2*n`, and `3*n`.
   - Concatenated String = `'192' + '384' + '576'` = `'192384576'`
3. **Sorting**: Sort the characters in the concatenated string.
   - Before Sorting: `'192384576'`
   - After Sorting: `''.join(sorted('192384576'))` = `'123456789'`
4. **Comparison**: Compare the sorted string with the string `'123456789'`.
   - Since the sorted string is `'123456789'`, and it matches exactly with the string `'123456789'`, the number `192` is a fascinating number.

According to the steps above, if we apply this process to the number `192`, we get `'123456789'` after the sorting step, which confirms that `192` is indeed a fascinating number. The function according to our solution approach would return `true` for this input.

Solution Implementation

Python

```python
class Solution:
    def isFascinating(self, n: int) -> bool:
        # Concatenate the string representation of the number 'n'
        # with its double ('2 * n') and its triple ('3 * n')
        concatenated_str = str(n) + str(2 * n) + str(3 * n)

        # Sort the concatenated string and join it to check if it forms
        # the sequence "123456789", which means that each digit from
        # 1 to 9 occurs exactly once.
        # This is the condition for a number to be fascinating.
        sorted_str = "".join(sorted(concatenated_str))

        # Compare the sorted string with "123456789" to determine if the number is fascinating.
        # Return True if it is fascinating, False otherwise.
        return sorted_str == "123456789"
```

Java

```java
class Solution {
    // Method to check whether a number is fascinating or not
    public boolean isFascinating(int number) {
        // Create a string concatenating 'number', 'number * 2', and 'number * 3'
        String concatenatedResult = "" + number + (2 * number) + (3 * number);

        // Array to keep count of digits 1-9 appearing in the concatenated result
        int[] digitCount = new int[10];

        // Iterate over each character in the concatenatedResult string
        for (char digit : concatenatedResult.toCharArray()) {
            // Incrementing the count of the digit in the array. If any digit is repeated, return false.
            if (++digitCount[digit - '0'] > 1) {
                return false;
            }
        }

        // Check that digit '0' does not appear and the length of concatenatedResult is 9
        // Since a fascinating number must include every digit from 1 to 9 exactly once.
        return digitCount[0] == 0 && concatenatedResult.length() == 9;
    }
}
```

C++

```cpp
#include <algorithm> // Required for std::sort
#include <string>    // Required for std::string and to_string

class Solution {
public:
    // Function to check if a number is fascinating
    // A fascinating number is one which when concatenated with its multiples of 2 and 3 results in all digits from 1 to 9 exacti
    bool isFascinating(int number) {
        // Convert number, number*2, and number*3 to string and concatenate them
        std::string concatenated = std::to_string(number) +
                                   std::to_string(number * 2) +
                                   std::to_string(number * 3);

        // Sort the concatenated string to check for the sequence "123456789"
        std::sort(concatenated.begin(), concatenated.end());

        // Return true if the sorted string matches "123456789", indicating that all digits are present exactly once.
        return concatenated == "123456789";
    }
};
```

TypeScript

```typescript
// This function checks if the number is fascinating or not.
// A number is fascinating if when concatenated with its multiples of 2 and 3,
// the resulting string contains all digits from 1 to 9 exactly once.
function isFascinating(n: number): boolean {
    // Concatenate `n` with its multiples of 2 and 3
    const concatenatedString = `${n}${n * 2}${n * 3}`;

    // Split the concatenated string into an array, sort it and join back to a string
    const sortedString = concatenatedString.split('').sort().join('');

    // Check if the sorted string matches '123456789' which would mean it contains all digits once
    return sortedString === '123456789';
}


class Solution:
    def isFascinating(self, n: int) -> bool:
        # Concatenate the string representation of the number 'n'
        # with its double ('2 * n') and its triple ('3 * n')
        concatenated_str = str(n) + str(2 * n) + str(3 * n)

        # Sort the concatenated string and join it to check if it forms
        # the sequence "123456789", which means that each digit from
        # 1 to 9 occurs exactly once.
        # This is the condition for a number to be fascinating.
        sorted_str = "".join(sorted(concatenated_str))

        # Compare the sorted string with "123456789" to determine if the number is fascinating.
        # Return True if it is fascinating, False otherwise.
        return sorted_str == "123456789"
```

Time and Space Complexity

Time Complexity: The time complexity of the code is `O(n log n)`, where `n` is the number of digits in the number `9 * n`, since `9 * n` has the most digits among `n`, `2 * n`, and `3 * n` and will dominate the time complexity. The `sorted()` function has `O(n log n)` complexity, and since the maximum value of `n` after concatenation of `n`, `2 * n`, and `3 * n` is `9 * n` (assuming the input does not exceed 9,999 because beyond that, the concatenated string will exceed 9 digits and can't be fascinating), we use `9 * digits_in_n` to denote `n`.

Space Complexity: The space complexity is `O(n)` which corresponds to the space needed to store the concatenated string `s`. The number of digits in this string is at most 9 if we consider the largest 4-digit number that could make a fascinating number. There is also the space used by the `sorted` function to create a list of characters, which however does not exceed 9 characters. Thus, the number of digits doesn't grow with the input `n`, leading to a constant space complexity when considering the actual storage for digits.