1271. Hexspeak

Easy Math String Leetcode Link

Problem Description

The problem provided requires converting a decimal number to a special hexadecimal representation known as "Hexspeak." Hexspeak is a whimsical representation of hexadecimal numbers where the digits '0' and '1' are replaced with the letters 'O' and 'I' respectively. Therefore, the valid characters in Hexspeak are only 'A', 'B', 'C', 'D', 'E', 'F', 'I', and 'O'.

The task involves the following steps:

1. Take the decimal number as a string and convert it to a hexadecimal string.
2. Change all the '0's to 'O's and all the '1's to 'I's in the hexadecimal string.
3. Check if the resulting string only contains the Hexspeak characters mentioned above.

If the string after performing the conversion contains only the valid Hexspeak characters, it is considered a valid Hexspeak representation. If any other character is present, the function should return the string "ERROR."

Intuition

The intuition behind the solution is straightforward. First, we convert the given decimal number to its hexadecimal equivalent using the built-in functions of Python. Then we apply string manipulation techniques to interchange '0's and '1's with 'O's and 'I's as per the Hexspeak rules.

We use the `hex` function to convert the decimal number to hexadecimal then slice the `0x` prefix provided by Python's `hex` conversion. To ensure Hexspeak format, we convert the hexadecimal string to uppercase. Following that, we replace '0's with 'O's and '1's with 'I's. We make sure that the transformed string complies with the Hexspeak standard by comparing each character to an approved set of characters 'ABCDEFIO'.

The check involves iterating through each character of the transformed string and verifying it exists in the set of valid Hexspeak characters. If all characters are valid, the transformed string is returned. If any character doesn't match the valid set, we know that it isn't a correct Hexspeak representation, and we return "ERROR."

Solution Approach

The implementation of the solution can be broken down into a few steps involving algorithms, data structures, and programming patterns:

1. **Hexadecimal Conversion**: We use Python's built-in `hex()` function, which converts a decimal number to its hexadecimal equivalent. Since the `hex()` function returns a string prefixed with "`0x`" representing that the number is hexadecimal, we slice the result to omit the "`0x`" part using array slicing (`[2:]`).

2. **String Manipulation**: We carry out a series of string manipulations to convert the hexadecimal number to follow Hexspeak rules:

   - Convert to uppercase with `.upper()` method to align with Hexspeak's requirement that all letters must be uppercase.
   - Replace all occurrences of '0' with 'O' and '1' with 'I' using the `.replace('0', 'O').replace('1', 'I')` method chain.
3. **Validation Against Hexspeak Specification**: To ensure the conversion is valid as per Hexspeak rules, we check each character of the transformed string:

   - We introduce a set `s` containing the allowed Hexspeak characters - `ABCDEFIO`.
   - We then iterate over each character in the transformed string `t` to confirm if it exists in the set `s`. This is done using a list comprehension in combination with the `all()` function, which checks if all elements of the iteration meet the condition (`c in s for c in t`).
4. **Conditional Output**: The result of the validation checks decides the output:

   - If all characters are in the set `s` (that is, they're all Hexspeak valid characters), the transformed string `t` is returned as the result.
   - If any character is not found in the set `s`, the string "ERROR" is returned.

The choice of data structures and algorithms is suitable for the given task, with an emphasis on simplicity and readability. String operations and a set are well-suited for this conversion and validation task due to their efficiency and the direct support for the required operations in the Python standard library.

Example Walkthrough

Let's take an example where the decimal number is `257`. We will walk through the solution approach step by step to convert this number into a Hexspeak representation.

1. Hexadecimal Conversion

First, we convert the decimal number `257` into hexadecimal. We use Python's `hex()` function:

```python
hex_number = hex(257)
```

This returns `0x101`. We slice off the `0x` part to obtain just the hexadecimal representation:

```python
hex_number = hex_number[2:]
```

So, `hex_number` will now be `"101"`.

2. String Manipulation

To convert `hex_number` to Hexspeak, we make the following string manipulations:

- We convert the string to uppercase.
- We replace '0' with 'O' and '1' with 'I'.

Performing these operations, we have:

```python
hexspeak_number = hex_number.upper().replace('0', 'O').replace('1', 'I')
```

After this step, `hexspeak_number` is `"IOI"`.

3. Validation Against Hexspeak Specification

Now, we need to check if `hexspeak_number` contains only valid Hexspeak characters. We create a set `s` that includes the allowed characters 'A', 'B', 'C', 'D', 'E', 'F', 'I', and 'O':

```python
s = {'A', 'B', 'C', 'D', 'E', 'F', 'I', 'O'}
```

We iterate over each character in `hexspeak_number` and check if all of them are in the set `s`:

```python
is_valid_hexspeak = all(c in s for c in hexspeak_number)
```

Since 'I' and 'O' are valid Hexspeak characters, `is_valid_hexspeak` will be `True`.

4. Conditional Output

Based on the validity check, we decide the output. Since `is_valid_hexspeak` is `True`, we return the `hexspeak_number`:

```python
if is_valid_hexspeak:
    result = hexspeak_number
else:
    result = "ERROR"
```

In this case, the output `result` will be `"IOI"`, which is the correct Hexspeak representation of the decimal number `257`.

Through these steps, we have successfully converted a decimal number into the Hexspeak representation following the outlined solution approach.

Python Solution

```python
class Solution:
    def toHexspeak(self, num: str) -> str:
        # Define a set of valid Hexspeak letters plus the replacements for 0 and 1.
        valid_hexspeak_characters = {'A', 'B', 'C', 'D', 'E', 'F', 'I', 'O'}

        # Convert the input string 'num' to an integer, then to a hexadecimal string,
        # convert to uppercase, replace '0' with 'O', and '1' with 'I'.
        hex_value = hex(int(num))[2:].upper().replace('0', 'O').replace('1', 'I')

        # Check if all characters in the hexadecimal string are in the set of valid characters.
        # If they are, return the hexadecimal string, else return 'ERROR'
        if all(character in valid_hexspeak_characters for character in hex_value):
            return hex_value
        else:
            return 'ERROR'

```

Java Solution

```java
class Solution {

    // Define a constant set containing valid HexSpeak characters.
    private static final Set<Character> VALID_CHARS = Set.of('A', 'B', 'C', 'D', 'E', 'F', 'I', 'O');

    public String toHexspeak(String num) {
        // Convert the given number to a hexadecimal string in uppercase.
        // Replace '0' with 'O' and '1' with 'I' to adhere to HexSpeak rules.
        String hexSpeak = Long.toHexString(Long.valueOf(num)).toUpperCase()
                              .replace("0", "O")
                              .replace("1", "I");

        // Check each character of the string to ensure it's valid HexSpeak.
        for (char c : hexSpeak.toCharArray()) {
            if (!VALID_CHARS.contains(c)) {
                // If a character is not valid, return "ERROR".
                return "ERROR";
            }
        }
        // If all characters are valid, return the HexSpeak string.
        return hexSpeak;
    }
}

```

C++ Solution

```cpp
#include <sstream>
#include <string>

class Solution {
public:
    // Function to convert a decimal number represented as a string to "hexspeak".
    // Hexspeak is a representation of hexadecimal values where only the letters
    // 'A' to 'F' and the digits '0' and '1' are allowed. '0' is replaced with 'O' and '1' with 'I'.
    string toHexspeak(string num) {
        stringstream stream; // create a stringstream object
        stream << hex << stol(num); // convert the string number to a long and output as hexadecimal
        string hexString = stream.str(); // get the hex string
        for (int i = 0; i < hexString.size(); ++i) { // iterate over each character
            // If the character is a digit between '2' and '9', return "ERROR".
            if (hexString[i] >= '2' && hexString[i] <= '9') return "ERROR";
            // If the character is '0', replace with 'O'
            if (hexString[i] == '0')
                hexString[i] = 'O';
            // If the character is '1', replace with 'I'
            else if (hexString[i] == '1')
                hexString[i] = 'I';
            // Convert lowercase letters to uppercase (for the range 'a' to 'f')
            else
                hexString[i] = toupper(hexString[i]); // using the standard toupper function for clarity
        }
        return hexString; // return the converted string
    }
};

```

Typescript Solution

```typescript
// Function to convert a decimal number represented as a string to "hexspeak".
// Hexspeak is a representation of hexadecimal values where only the letters
// 'A' to 'F' and the digits '0' and '1' are allowed. '0' is replaced with 'O' and '1' with 'I'.
function toHexspeak(numStr: string): string {
    // Convert the input string to a number in base 10, and then to a hexadecimal string
    let hexString: string = parseInt(numStr).toString(16);
    // Loop over each character of the hex string to apply transformations
    for (let i = 0; i < hexString.length; ++i) {
        let char = hexString[i];
        // If the character is a digit between '2' and '9', return "ERROR".
        if (char >= '2' && char <= '9') {
            return "ERROR";
        }
        hexString = hexString.split(''); // Convert string to array for mutation
        // If the character is '0', replace with 'O'
        if (char === '0') {
            hexString[i] = 'O';
        }
        // If the character is '1', replace with 'I'
        else if (char === '1') {
            hexString[i] = 'I';
        }
        // Convert lowercase letters to uppercase (for the range 'a' to 'f')
        else {
            hexString[i] = char.toUpperCase();
        }
        hexString = hexString.join(''); // Convert array back to string
    }
    // Return the converted hex string in hexspeak
    return hexString;
}

```

Time and Space Complexity

The given Python code converts a decimal number to "hexspeak," which is a hex representation that only contains certain letters (A, B, C, D, E, F, I, O) and excludes all digits except for '0' and '1', which are replaced by 'O' and 'I', respectively. If the converted hex contains characters not allowed in hexspeak, it returns 'ERROR'.

Time Complexity:

The time complexity of the code is determined by several factors including converting the decimal to hexadecimal, replacing characters, and checking whether all characters belong to a set of allowed characters.

1. `hex(int(num))`: Converting the decimal number to a hexadecimal string has a time complexity of `O(N)`, where `N` is the length of the number in bits.

2. `[2:]`, `.upper()`, `.replace()`, `.replace()`: The subsequent operations on the string occur in linear time relative to the size of the string. These have a complexity of `O(K)` collectively, where `K` is the length of the hexadecimal string which is proportional to `N`.

3. `all(c in s for c in t)`: This list comprehension with containment check in a set has `O(K)` time complexity, as set containment check operations have constant average time complexity (`O(1)`), and it is done for all characters in the hex string `K`.

Since each of these steps is linear with respect to its input, and recognizing that `K`, the length of the hexadecimal representation, does not exceed `O(log(N))` (the number of digits in a base `b` representation of a number is proportional to `log_b(N)`), the overall time complexity of the function is dominated by the decimal to hexadecimal conversion and can be considered `O(N)`.

Space Complexity:

The space complexity of the code includes the space used by the input, the set `s`, the intermediate string `t`, and the output.

1. The space taken by the set `s` is a constant `O(1)` because it only includes a fixed number of characters.

2. The intermediate string `t` holds the hexadecimal representation of the number, which is proportional to the number of digits in the input number's bit representation; this is `O(log(N))`.

3. For the output, in the worst case (when `t` does not include any forbidden characters), the space complexity remains `O(log(N))`.

Therefore, the total space complexity of the function is also dominated by the storage required for the hexadecimal string, resulting in an overall space complexity of `O(log(N))`.