12. Integer to Roman

Medium Hash Table Math String

Problem Description

In this problem, we are given the task to convert an integer to its equivalent Roman numeral representation. Roman numerals use a specific set of symbols that correspond to certain values. The seven symbols used are I, V, X, L, C, D, and M. Here's their corresponding values:

- I represents 1
- V represents 5
- X represents 10
- L represents 50
- C represents 100
- D represents 500
- M represents 1000

Roman numerals are written by combining these symbols starting from the largest to the smallest from left to right. However, instead of repeating a symbol four times, Roman numerals use a subtractive notation for certain numbers. For instance, the number 4 isn't represented as "IIII" but as "IV" (1 before 5 signifies subtraction, giving us 4). This rule of subtraction applies to several pairs, such as I before V or X, X before L or C, and C before D or M. The problem requires us to implement the logic that can convert any given integer within the permissible range into this Roman numeral format.

Intuition

The solution approach is a greedy one. We work through the integer from the largest possible Roman numeral to the smallest. We start by creating an ordered list of Roman numeral characters (`cs`) and their respective integer values (`vs`), including those representing the subtraction cases. For example, "CM" for 900, "XL" for 40 and so on. These need to be in descending order so that we can start with the largest possible values and work our way down.

We initialize an empty list `ans` that will be used to store the Roman numeral symbols sequentially as we build up our answer. Now, for each pair (symbol and its value) we check if our integer (`num`) is greater than or equal to the current value (`v`). If it is, we subtract this value from `num` and append the corresponding Roman numeral symbol to our `ans` list. We repeat this process, checking and subtracting the same value, until `num` is smaller than the current value. Then we proceed to the next value-symbol pair and continue the process.

Essentially, we are always trying to subtract the largest possible values from our given integer, which is why this method is considered greedy. This continues until our integer (`num`) has been reduced to 0, meaning we've found a Roman numeral representation for each part of the initial number. Finally, we concatenate the symbols in our `ans` list and return the result as the Roman numeral representation of the initial integer.

Solution Approach

The solution to this problem employs a simple but efficient algorithm using Greedy approach, array manipulation, and sequential iteration. The key data structures used are tuples for storing Roman numeral characters and their corresponding values, and a list to construct the final Roman numeral string.

Here's an in-depth look at each step of the algorithm:

1. **Define Symbol-Value Pairs**: Two tuples are created, one holding the Roman numeral characters `cs` and the other holding their corresponding values `vs`. These are aligned by indices and include entries for the subtractive combinations like "CM" for 900, and "IV" for 4, among others. These tuples are ordered from the largest value to the smallest.
2. **Initialize the Result Holder**: An empty list, `ans`, is created to hold the characters that will eventually form the final Roman numeral string.
3. **Iterate Through Symbol-Value Pairs**: We iterate through the pairs in the order they are defined, starting with the largest value.
4. **Greedy Subtraction and Accumulation**: In each iteration, we check if our current number `num` is greater than or equal to the value `v` at the current iteration. If it is, this means we can represent a part of our number with the Roman numeral symbol `c`. We subtract this value `v` from `num` and append the symbol `c` to our `ans` list.
5. **Repeat Subtractions**: We continue to subtract and append the same symbol until `num` is less than `v`. This process follows the Greedy approach because it always consumes the largest possible "chunk" of the number in each step, hence reducing the number in the largest steps possible.
6. **Proceed To Next Symbol-Value Pair**: Once `num` is smaller than the current value `v`, we move on to the next pair. This essentially transitions to the next largest Roman symbol that `num` can accommodate.
7. **Concatenate the Result**: After the loop exits, we combine all characters in the `ans` list into a string, which gives us the final Roman numeral.

The pseudocode makes this process clear:

```
function intToRoman(num):
    initialize cs as tuples ("M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I")
    initialize vs as tuples (1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1)
    initialize ans as an empty list

    for each (c, v) in zip(cs, vs):
        while num >= v:
            subtract v from num
            append c to ans

    return the concatenation of items in ans
```

In essence, this algorithm is a straightforward application of the Greedy algorithm, suitable for the problem's requirements, as the representation of Roman numerals is inherently "Greedy" — favoring the usage of larger denominations whenever possible. The use of tuples and list operations makes the implementation concise and efficient.

Example Walkthrough

Let's go through a specific example using the integer 1437 to illustrate the solution approach described earlier. Here's how the algorithm would convert the number 1437 into a Roman numeral:

1. **Start With the Largest Symbol-Value Pair**: The first value and symbol pair in our pre-defined tuples that 1437 is greater than or equal to is "M" which corresponds to 1000.
2. **Subtract and Accumulate**: We subtract 1000 from 1437, leaving us with 437, and add "M" to our `ans` list.
3. **Repeat with the Remaining Number**: Looking at the next largest symbol-value pair, no value is greater than or equal to 437 until we get to "CD" for 400.
4. **Subtract and Accumulate**: Subtract 400 from 437, getting 37, and add "CD" to our `ans` list.
5. **Continue the Process**: There's no symbol for 37, so we look for the next highest which is "X" corresponding to 10.
6. **Subtract and Accumulate**: We can subtract 10 from 37 three times, adding "X" to our `ans` list three times, and are left with 7.
7. **Finish Up**: The highest value for 7 is "V" which is 5. We subtract 5 from 7 and add "V" to our `ans` list, leaving us with 2.
8. **Final Steps**: We can subtract 1 from 2 twice, so we add "I" to our list twice.
9. **Concatenate the Result**: Now that our number has been reduced to 0, we concatenate everything we've added to our `ans` list and end up with "MCDXXXVII" as the Roman numeral representation of 1437.

So the step-by-step process for the number 1437 goes like this:

- Start at 1437, subtract 1000, list is now ["M"].
- 437 remains, subtract 400, list is now ["M", "CD"].
- 37 remains, subtract 30 ($3 * 10$), list is now ["M", "CD", "X", "X", "X"].
- 7 remains, subtract 5, list is now ["M", "CD", "X", "X", "X", "V"].
- 2 remains, subtract 2 ($2 * 1$), list is now ["M", "CD", "X", "X", "X", "V", "I", "I"].
- Concatenate list items to get "MCDXXXVII".

This example captures the essence of the greedy algorithm in converting an integer to a Roman numeral, consistently taking the largest possible value that fits into what remains of the number until the entire number is converted.

Python Solution

```python
class Solution:
    def intToRoman(self, num: int) -> str:
        # Tuple of Roman numerals in descending order
        roman_symbols = ('M', 'CM', 'D', 'CD', 'C', 'XC', 'L', 'XL', 'X', 'IX', 'V', 'IV', 'I')
        # Tuple of the respective values of the Roman numerals
        roman_values = (1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1)

        # Will hold the resulting Roman numeral string
        roman_string = []

        # Pair Roman symbols with their values
        for symbol, value in zip(roman_symbols, roman_values):
            # As long as the number is greater than or equal to the value
            while num >= value:
                # Subtract the value from the number
                num -= value
                # Append the corresponding Roman symbol to the list
                roman_string.append(symbol)

        # Join the list of symbols to get the final Roman numeral string
        return ''.join(roman_string)

```

Java Solution

```java
class Solution {
    public String intToRoman(int num) {
        // Define arrays for Roman numeral characters and their corresponding values.
        String[] romanNumerals = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"};
        int[] values = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};

        // Initialize a StringBuilder to build the Roman numeral string.
        StringBuilder romanString = new StringBuilder();

        // Iterate over the Roman numerals and values to construct the Roman numeral.
        for (int i = 0; i < romanNumerals.length; i++) {
            while (num >= values[i]) { // While the number is greater than the current value
                num -= values[i]; // Subtract the value from the number
                romanString.append(romanNumerals[i]); // Append the corresponding Roman numeral to the string
            }
        }

        // Return the constructed Roman numeral string.
        return romanString.toString();
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to convert an integer to a Roman numeral
    string intToRoman(int num) {
        // Define the Roman numeral symbols and their corresponding values
        vector<string> romanSymbols = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"};
        vector<int> values = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
        string result;  // This will hold the resulting Roman numeral

        // Go through each symbol starting with the largest
        for (int i = 0; i < romanSymbols.size(); ++i) {
            // As long as the number is larger than the romanSymbols'value
            while (num >= values[i]) {
                // Subtract the value from the number
                num -= values[i];
                // And append the symbol to the result string
                result += romanSymbols[i];
            }
        }

        // Return the final Roman numeral string
        return result;
    }
};

```

Typescript Solution

```typescript
// Function to convert an integer to a Roman numeral
// num: the integer to convert
// returns: a string representing the Roman numeral
function intToRoman(num: number): string {
    // Array of Roman numeral symbols
    const romanSymbols: string[] = [
        'M', 'CM', 'D', 'CD', 'C', 'XC', 'L', 'XL', 'X', 'IX', 'V', 'IV', 'I'
    ];

    // Corresponding values for the Roman numeral symbols
    const values: number[] = [
        1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1
    ];

    // The resulting array to store Roman numeral parts
    const result: string[] = [];

    // Iterate over the values and symbols arrays
    for (let i = 0; i < values.length; ++i) {
        // While the current number is greater than or equal to the current value
        while (num >= values[i]) {
            // Subtract the value from the number
            num -= values[i];
            // Add the corresponding symbol to the result array
            result.push(romanSymbols[i]);
        }
    }

    // Convert the result array to a string to get the final Roman numeral
    return result.join('');
}

```

Time and Space Complexity

The time complexity of the given Python function `intToRoman` is O(1), because the number of operations performed does not depend on the size of the input number but on the fixed set of Roman numeral symbols. Since Roman numerals have a finite number of symbols (`cs`) that are considered to provide the greatest value in the Roman numeral system, the loop within the function will iterate at most a constant number of times, which is equivalent to the number of symbols in the set `cs` (13 symbols in this case).

The space complexity of the function is also O(1) for similar reasons. The amount of memory used by the program does not increase proportionally to the input `num`. It rather depends on the size of the two constant arrays `cs` and `vs`, and the list `ans`. Regardless of the size of `num`, `ans` will have at most a constant number of elements related to the number of possible Roman numeral symbols. Therefore, the space used by the list `ans` will not grow arbitrarily with the input value.

To summarize, both the time complexity and the space complexity of the function are constant, represented as:

```
Time Complexity: O(1)
Space Complexity: O(1)
```