1291. Sequential Digits

Medium Enumeration Leetcode Link

Problem Description

We are provided with two integers, `low` and `high`, which define a range. The task is to find all integers within this range that have sequential digits. An integer is said to have sequential digits if each digit in the number is exactly one more than the previous digit. For example, 123 has sequential digits, but 132 does not. The final list of found integers should be returned in increasing order.

Intuition

To tackle the problem of finding integers with sequential digits within a given range, we need to generate these numbers rather than checking every number in the range. Starting with the smallest digit '1', we can iteratively add the next sequential digit by multiplying the current number by 10 (to shift the digits to the left) and then adding the next digit. For instance, start with `1`, then `12`, `123`, and so on, till we reach a number greater than `high`. We repeat this for starting digits 1 through 9.

The intuitive steps in the solution approach are:

- Loop through all possible starting digits (`1` to `9`).
- In a nested loop, add the next sequential digit to the current number and check if it falls within the range [low, high]. If it does, add it to our answers list.
- Repeat this process until the number exceeds `high` or we reach the digit '9'.
- Since the process generates the sequential digit numbers in increasing order, but not necessary starting with the lowest in the range, a final sort of the answer list ensures that the output is correctly ordered.

The provided solution follows this methodology, using two nested loops, to generate the numbers with sequential digits and collect the ones that lie within the range.

Solution Approach

The algorithm takes a direct approach by constructing numbers with sequential digits and then checking if they lie within the provided range.

Here's the implementation process:

1. **Initialization**: We start by creating an empty list `ans`, which will be used to store all the numbers that match our condition and are within the `[low, high]` range.

2. **Outer Loop**: This loop starts with the digit `1` and goes up to `9`, representing the starting digit of our sequential numbers.

3. **Inner Loop**: For each digit `i` started by the outer loop, we initialize an integer `x` with `i`. This inner loop adds the next digit (`j`) to `i`, so that `x` becomes a number with sequential digits. For example, if `i` is `1`, then `x` will be transformed through the sequence: `1 -> 12 -> 123 -> ...` and so on. The loop continues until `j` reaches `10`, as `9` is the last digit we can add.

4. **Number Construction and Range Checking**: In each iteration of the inner loop, `x` is updated as `x = x * 10 + j`; this adds the next digit at the end of `x`. The newly formed number is then checked to ascertain if it's within the `[low, high]` range. If `x` falls within the range, the number is added to the `ans` list.

5. **Loop Termination**: The loop terminates in one of two cases. The first is when `x` exceeds `high`, since all subsequent numbers that could be formed by adding more sequential digits will also be out of the range. The second case is when the last digit (which is `9`) has been added to the sequence, and no further sequential digit can follow.

6. **Returning the Sorted List**: Although the algorithm itself generates these numbers in a sequential order, they are added to the list in an order based on the starting digit. Therefore, a final sort is applied before returning the list to ensure that numbers are sorted according to their value.

The utilized data structures are simple: an `int` variable to keep track of the currently formed number with sequential digits, and a list to store and return the eligible numbers.

The patterns involved in this algorithm include:

- **Constructive algorithms**: Rather than checking all numbers within a range, we constructively build the eligible numbers.
- **Brute force with a twist**: Instead of brute force checking all numbers, we only consider plausible candidates.
- **Range-based iteration**: Both loops are based on clearly defined ranges (1 to 9 and the dynamically generated `x` within `[low, high]`).

Here's a brief code snippet for reference:

```python
class Solution:
    def sequentialDigits(self, low: int, high: int) -> List[int]:
        ans = []
        for i in range(1, 9):
            x = i
            for j in range(i + 1, 10):
                x = x * 10 + j
                if low <= x <= high:
                    ans.append(x)
        return sorted(ans)
```

As you can see, the code closely follows the outlined algorithm, utilizing two nested for-loops to build and check the sequential digit numbers.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach based on the problem of finding integers with sequential digits within a given range. Say our range is defined by the integers `low = 100` and `high = 300`.

Following the steps of the solution approach:

1. **Initialization**: Start with an empty list `ans`, ready to store numbers with sequential digits within our range `[100, 300]`.

2. **Outer Loop**: We consider starting digits from `1` through `9`.

3. **Inner Loop**: Starting with the initial digit `i`. For example, when `i` is `1`, we will construct numbers beginning with `1`.

4. **Number Construction and Range Checking**:

   - With `i = 1`, we construct `x = 1`.
   - Increment the next digit by one and add it to `x`, so now `x = 12` (because `1 * 10 + 2 = 12`).
   - Continue adding the next digit: `x = 123`. This number exceeds our `low` but is still less than `high`, so we add `123` to our `ans` list.
   - When we add `4`, `x = 1234`, which is now greater than `high`. We stop here for this sequence and do not add `1234` to `ans`.
5. **Loop Termination**: This inner loop stops for this particular starting digit since `x` has exceeded `high`, and we won't attempt to add more sequential digits.

6. **Proceed with Other Starting Digits**: Loop with the next starting digit, `i = 2`.

   - Repeat the process with `x = 2`, then `x = 23`.
   - We find `x = 234` exceeds our range, so we don't add `234` to `ans` and stop the inner loop for the starting digit `2`.

This process continues for all starting digits from `1` through `9`. However, given our `–` high`, we won't find any valid numbers with sequential digits starting with `3`or greater within the range`[100, 300]`.

7. **Returning Sorted List**: Once the loops have terminated, we sort the list `ans`. In this case, the only number we would have added to the list is `123`, which is already in sorted order, but if there were more numbers, sorting would ensure they are returned in increasing order.

Thus, our final list `ans` is `[123]`, and it would be returned as the solution.

Quick Recap:

- Initialized with `ans = []`.
- Our loops constructed and checked sequential digits from numbers starting with `1`, then `2`, then `3`, and so forth.
- We found `123` as a valid number and added it to `ans`.
- The loops stopped when the constructed number exceeded `high` for each starting digit.
- We would return a sorted `ans`, which in this case is simply `[123]`.

Python Solution

```python
from typing import List

class Solution:
    def sequentialDigits(self, low: int, high: int) -> List[int]:
        # Initialize an empty list to store the sequential digits
        sequential_numbers = []

        # Iterate over digits from 1 to 8 (since 9 cannot start a sequence)
        for start_digit in range(1, 9):
            # Initialize the current number with the starting digit
            current_number = start_digit

            # Build the sequence by appending digits to the right
            for next_digit in range(start_digit + 1, 10):  # Only digits 1-9 are valid
                # Create the next number in the sequence by shifting left and adding the next_digit
                current_number = current_number * 10 + next_digit

                # Check if the current number is within the given range [low, high]
                if low <= current_number <= high:
                    # If it is, append it to the list of sequential numbers
                    sequential_numbers.append(current_number)

        # Return the sorted list of sequential numbers
        return sorted(sequential_numbers)

# Example usage:
# solution = Solution()
# print(solution.sequentialDigits(100, 300))  # Output: [123, 234]

```

Java Solution

```java
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

class Solution {
    public List<Integer> sequentialDigits(int low, int high) {
        // Initialize the answer list to hold sequential digit numbers
        List<Integer> sequentialNumbers = new ArrayList<>();

        // Start generating numbers from each digit 1 through 8
        // A sequential digit number cannot start with 9 as it would not have a consecutive next digit
        for (int startDigit = 1; startDigit < 9; ++startDigit) {
            // Initialize the sequential number with the current starting digit
            int sequentialNum = startDigit;

            // Append the next digit to the sequential number, starting from startDigit + 1
            for (int nextDigit = startDigit + 1; nextDigit < 10; ++nextDigit) {
                // Append the next digit to the current sequential number
                sequentialNum = sequentialNum * 10 + nextDigit;

                // Check if the newly formed sequential number is within the range [low, high]
                if (sequentialNum >= low && sequentialNum <= high) {
                    // If it is within the range, add it to the answer list
                    sequentialNumbers.add(sequentialNum);
                }
            }
        }

        // Sort the list of sequential numbers
        Collections.sort(sequentialNumbers);

        // Return the list containing all valid sequential digit numbers in the range
        return sequentialNumbers;
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <algorithm>

class Solution {
public:
    // Function to find all the unique numbers that consist of sequential digits
    // and are within the range [low, high] inclusive.
    vector<int> sequentialDigits(int low, int high) {
        vector<int> sequential_numbers; // Stores the resulting sequential numbers
        // Loop to start forming sequential digits from each number 1 through 8.
        for (int start_digit = 1; start_digit < 9; ++start_digit) {
            int num = start_digit; // Initialize the current number with the start digit.
            // Add digits sequentially after the start digit to form a number.
            for (int next_digit = start_digit + 1; next_digit < 10; ++next_digit) {
                num = num * 10 + next_digit; // Append the next digit at the unit place.
                // Check if the number formed is within the range.
                if (num >= low && num <= high) {
                    sequential_numbers.push_back(num); // Add it to our answer if it's within the range.
                }
            }
        }
        // Sort the vector containing all the qualifying numbers in ascending order.
        sort(sequential_numbers.begin(), sequential_numbers.end());
        // Return the vector of sequential numbers within the given range.
        return sequential_numbers;
    }
};

```

Typescript Solution

```typescript
// Function to find all the sequential digit numbers within the range [low, high].
function sequentialDigits(low: number, high: number): number[] {
    // Initialize an empty array to hold the result.
    let result: number[] = [];

    // Outer loop to start the sequence with numbers from 1 to 8.
    for (let startDigit = 1; startDigit < 9; ++startDigit) {
        // Initialize the first number of the sequence with the starting digit.
        let num = startDigit;

        // Inner loop to build the sequential digit numbers by appending digits.
        for (let nextDigit = startDigit + 1; nextDigit < 10; ++nextDigit) {
            // Create the new sequential number by shifting the current
            // number by one place to the left and adding the next digit.
            num = num * 10 + nextDigit;

            // Check if the newly formed number is within the given range.
            if (num >= low && num <= high) {
                // If the number is within the range, add it to the result array.
                result.push(num);
            }
        }
    }

    // Sort the result array in ascending order before returning.
    result.sort((a, b) => a - b);
    return result;
}

```

Time and Space Complexity

The given Python code generates sequential digits between a low and high value by constructing such numbers starting from each digit between 1 and 9. Here is an analysis of its time and space complexity:

Time Complexity:

The time complexity of the code can be evaluated by considering the two nested loops. The outer loop runs for a constant number of times (8 times to be precise, as it starts from 1 and goes up to 8). The inner loop depends on the value of `i` from the outer loop and can iteratively execute at most 9 times (when `i=1`). However, not all iterations will be full, as it breaks off once it exceeds `high`. But in the worst-case scenario, if `low` is 1 and `high` is the maximum possible value within the constraints, each loop can be considered to run `O(1)` times as the number of iterations does not depend on the input values 'low' and 'high'.

With each iteration of the inner loop, we're performing `O(1)` operations (arithmetic and a conditional check). Therefore, the time complexity is `O(1)`.

Space Complexity:

The space complexity is dependent on the number of valid sequential digit numbers we can have in the predefined range. The `ans` list contains these numbers and is returned as the output. In the worst case, this could be all 'sequential digit' numbers from one to nine digits long. However, since the count of such numbers is limited (there are only 45 such numbers in total: 9 one-digit, 8 two-digit, ..., 1 nine-digit), the space needed to store them does not depend on the value of `low` or `high`. Therefore, the space complexity is also `O(1)`.