541. Reverse String II

Easy Two Pointers String

Problem Description

The problem presents a string manipulation scenario where we are asked to modify the string in a specific pattern based on a parameter `k`. You are given the string `s` and an integer `k`. The task is to reverse every first `k` characters in each `2k` segment of the string, starting from the beginning.

Here's the pattern we need to follow:

- **If a segment has `2k` characters**: reverse the first `k` characters and leave the next `k` characters in the original order.
- **If a segment has between `k` and `2k-1` characters**: reverse the first `k` characters and leave the rest in the original order.
- **If a segment has fewer than `k` characters**: reverse all of them.

It's important to note that reversing less than `k` characters happens only if those are the last characters in the string and their count is less than `k`.

Intuition

To solve this problem, we can iterate through the string in increments of `2k` since we know that every `2k` interval will require the same operation – reverse the first `k` characters. To implement this in code, we'll follow these steps:

1. Convert the string `s` into a list of characters since strings in Python are immutable and we want to efficiently manipulate individual characters.
2. Use a loop to work through the string in chunks of `2k`. For each iteration, only pay attention to the current section of the string that is `2k` in length. Within this current section, we only need to reverse the first `k` characters.
3. When reversing, we'll use the slice notation in Python, `t[i : i + k]`, where `t` is the list of characters from the original string, and `i` is the start of the current `2k` segment. Reverse this slice and replace the original contents with the reversed ones.
4. After the loop has processed the entire list, convert the list of characters back into a string using `''.join(t)` and return the new string.

The provided solution takes advantage of list slicing and the `reversed` function in Python, which makes it a clean and efficient approach to solving the problem.

Solution Approach

The provided solution uses several key programming concepts to solve the problem efficiently—list manipulation, slicing, looping, and in-place reversing of a sequence.

Here is how the solution is implemented:

1. **Convert the String to a List**: Since strings in Python are immutable and cannot be changed in place, the first step is to convert the string `s` into a list of characters, `t = list(s)`.
2. **Loop through the List in Chunks of `2k`:** The next step is to iterate over the list in chunks of `2k`. This is achieved using a `for` loop with a range that starts at 0 and ends at the length of the list `len(t)`, with step increments of `2k`. In Python, `k << 1` is a bitwise left shift operation that multiplies `k` by 2, giving us the required step size `2k`.
3. **Reverse the First `k` Characters**: For each chunk, reverse the first `k` characters. This is done by taking a slice `t[i : i + k]` from the current position `i` to the position `i + k`. The `reversed()` function is then used to reverse this slice. The reversed slice is then used to replace the original section of the list using slice assignment, which is an in-place operation.
4. **Join the List into a String**: After the loop finishes, a new string is constructed from the list of characters using `''.join(t)`. This step converts the list with the modified characters back into a string.
5. **Return the Result**: Finally, this new modified string is returned.

The algorithm relies on the efficiency of list operations and the ability to manipulate slices in Python to achieve the desired result without the need for additional data structures or complex patterns. By using the slice notation and `reversed()` function, the solution is concise and easy to read, while also minimizing the number of operations performed.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach.

Suppose we have the string `s = "abcdefgh"` and the value of `k = 2`. We are tasked to reverse every first `k` characters for each `2k` segment of the string.

Following our solution approach:

1. **Convert the String to a List**: We first convert the string into a list of characters `t = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']`.
2. **Loop through the List in Chunks of `2k`:** We will iterate over the list using steps of `2k = 4`. Our loop will be over the indices `0, 4`.
3. **Reverse the First `k` Characters**: We reverse the first `k` characters for each segment:
   - At the first iteration (`i=0`), our first `2k` segment is `['a', 'b', 'c', 'd']`. The first `k` characters `['a', 'b']` are reversed to `['b', 'a']`. The list now becomes `t = ['b', 'a', 'c', 'd', 'e', 'f', 'g', 'h']`.
   - At the second iteration (`i=4`), our second `2k` segment is `['e', 'f', 'g', 'h']`. The first `k` characters `['e', 'f']` are reversed to `['f', 'e']`. The list now becomes `t = ['b', 'a', 'c', 'd', 'f', 'e', 'g', 'h']`.
4. **Join the List into a String**: We join the list back into a string resulting in `'bacdfegh'`.
5. **Return the Result**: The final output we return will be the string `"bacdfegh"`.

Each step of the solution works together to efficiently manipulate the sections of the string as required, ensuring that we only reverse what's needed and maintain the order of the rest of the characters.

Solution Implementation

Python

```python
class Solution:
    def reverseStr(self, s: str, k: int) -> str:
        # Convert the input string to a list of characters for in-place modification.
        chars = list(s)

        # Process the list in blocks of 2k characters.
        for i in range(0, len(chars), 2*k):
            # Reverse the first k characters in the current block.
            # If the block is smaller than k, reverse the entire block.
            chars[i : i + k] = reversed(chars[i : i + k])

        # Join the list of characters back into a string and return it.
        return ''.join(chars)

# The use of 'k << 1' in the original code is equivalent to '2*k'.
# Using '2*k' makes the intention clearer - to increment i by chunks of 2k.
```

Java

```java
class Solution {
    public String reverseStr(String s, int k) {
        // Convert the input string 's' into a character array for in-place manipulation
        char[] charArray = s.toCharArray();

        // Iterate over the array in blocks of size 2k
        for (int startIndex = 0; startIndex < charArray.length; startIndex += (k * 2)) {
            // Initialize 'endIndex' to the minimum of the last index of the block or the end of the array
            int endIndex = Math.min(charArray.length - 1, startIndex + k - 1);

            // Reverse the characters in the current block from 'startIndex' to 'endIndex'
            reverseCharacters(charArray, startIndex, endIndex);
        }

        // Convert the reversed character array back to a string and return it
        return new String(charArray);
    }

    // Helper method to reverse a portion of the character array in place
    private void reverseCharacters(char[] charArray, int startIndex, int endIndex) {
        // Use two pointers to reverse the characters in the array
        while (startIndex < endIndex) {
            // Swap characters at 'startIndex' and 'endIndex'
            char temp = charArray[startIndex];
            charArray[startIndex] = charArray[endIndex];
            charArray[endIndex] = temp;

            // Move the pointers closer to the middle of the array
            startIndex++;
            endIndex--;
        }
    }
}
```

C++

```cpp
class Solution {
public:
    string reverseStr(string str, int k) {
        // Iterate over the string in chunks of 2k characters
        for (int start = 0, len = str.size(); start < len; start += (k * 2)) {
            // Calculate the end of the segment to be reversed,
            // which should not exceed the string's length
            int end = min(start + k, len);

            // Reverse the first k characters in the current 2k segment
            // If the remaining characters are less than k, reverse all of them
            reverse(str.begin() + start, str.begin() + end);
        }
        // Return the modified string
        return str;
    }
};
```

TypeScript

```typescript
// Define the function to reverse every k characters of a string,
// in each 2k interval.
function reverseStr(str: string, k: number): string {
    // Convert the string to an array of characters to manipulate
    let arr = str.split('');

    // Get the length of the string for later use
    const len = arr.length;

    // Iterate over the string in chunks of 2k characters
    for (let start = 0; start < len; start += 2 * k) {
        // Calculate the end index for the segment of the string that will be reversed,
        // ensuring that it does not exceed the string's length
        let end = Math.min(start + k, len) - 1;

        // Reverse the specified segment of the array using a two-pointer approach
        for (let i = start, j = end; i < j; i++, j--) {
            // Swap the characters at the i-th and j-th positions
            let temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // Return the modified string by joining the array back into a string
    return arr.join('');
}

// Example usage:
// reverseStr("abcdefg", 2) would return "bacdfeg"
```

```python
class Solution:
    def reverseStr(self, s: str, k: int) -> str:
        # Convert the input string to a list of characters for in-place modification.
        chars = list(s)

        # Process the list in blocks of 2k characters.
        for i in range(0, len(chars), 2*k):
            # Reverse the first k characters in the current block.
            # If the block is smaller than k, reverse the entire block.
            chars[i : i + k] = reversed(chars[i : i + k])

        # Join the list of characters back into a string and return it.
        return ''.join(chars)

# The use of 'k << 1' in the original code is equivalent to '2*k'.
# Using '2*k' makes the intention clearer - to increment i by chunks of 2k.
```

Time and Space Complexity

The time complexity of the given code can be broken down into two major operations: the loop that iterates over the string in chunks and the reversing of each chunk.

- The loop iterates over the entire length of the string "`s`" with a step of "`k << 1`" which is equivalent to "`2k`" because the left-shift operator "`<<`" effectively multiplies the number by "`2`" for every shift. The loop thus runs approximately "`len(s) / (2k)`" times.
- Inside each iteration, it reverses a sublist of "`t`" that is at most "`k`" elements long. The reverse operation takes "`O(k)`" time in the worst case.

Multiplying the number of iterations by the complexity of each iteration's operation gives us the total time complexity. Therefore, assuming "`n`" to be the length of the string "`s`", the overall time complexity of the code is approximately "`O(n/k * k)`" which simplifies to "`O(n)`".

The space complexity of the solution includes the additional space allocated for the conversion of the string "`s`" to a list "`t`". Since a list is created with the same number of elements as the input string, thus occupying "`O(n)`" space. No additional significant space is used that grows with the size of the input as the reversal is done in-place within the "`t`" list. Consequently, the space complexity of the code is "`O(n)`".