1694. Reformat Phone Number

Easy String

Problem Description

In the given problem, we are tasked with reformatting a phone number provided to us as a string. The input `number` may include digits, spaces `' '`, and/or dashes `'-'`. Our objective is to reorganize the digits into a specific pattern and return the reformatted number. The following steps are a part of the reformatting process:

1. First, we need to strip out all the spaces and dashes from the number.
2. Then, we must group the digits into blocks containing 3 digits each, proceeding from left to right.
3. We continue creating blocks of 3 digits until we're left with either 4, 3, or 2 digits.
4. The remaining digits are then grouped based on the amount left.
   - If there are 2 digits left, we keep them together as one block.
   - If there are 3 digits, they also stay as a single block.
   - If there are 4 digits, we split them into two blocks of 2 digits each.
5. Finally, we join these blocks together with dashes to produce the final formatted number.

The challenge is to ensure that we never end up with a block of length 1 and that we have a maximum of two blocks of length 2.

Intuition

To solve this problem, we can divide the approach into several logical steps that can be programmatically implemented:

1. **Cleaning the Input:** We want our number string free from any non-numeric characters like spaces and dashes. We do this by applying the `replace` method to remove these characters.
2. **Creating Groups of Three:** We then create as many groups of three as we can by iterating through the cleaned number. To facilitate this, we can use list comprehension, creating a sub-string for each group of three until we've handled all digits that can be grouped in threes.
3. **Handling Remaining Digits:** We always have to be careful about how we handle the last few digits of the phone number. Depending on whether there's a remainder of one or two after grouping digits by three, we handle the last blocks differently:
   - If there is only one left over, we need to adjust the last block to only contain two of those digits, and combine the last digit with the penultimate pair, creating two blocks of two.
   - If there are two left over, we simply add another block of two without needing any adjustments.
4. **Joining the Blocks:** After the blocks are correctly established, we can join them together into a single string with dashes in between using the `join` method.

Solution Approach

The solution iterates over the logic and patterns to reformat the phone number, as described under the intuition section. Here's how the implementation aligns with that strategy:

1. **Removing Non-numeric Characters:** The first step is straightforward – remove any dashes and spaces from the input number, which is achieved with two `replace` function calls.

```
number = number.replace("-", "").replace(" ", "")
```

2. **Creating Groups of Three:** Here we utilize Python's list comprehension combined with slicing. The comprehension iterates over a range `0` to `n // 3` where `n` is the length of the cleaned number, and for each iteration `i`, it slices a sub-string of the number starting from `i * 3` to `i * 3 + 3`. Each slice is a block with up to 3 digits.

```
ans = [number[i * 3 : i * 3 + 3] for i in range(n // 3)]
```

3. **Handling Remaining Digits:** Depending on the remainder when the cleaned number's length is divided by 3, we handle the final groups differently.
   - If there's a remainder of 1 (`n % 3 == 1`), it suggests our last block in `ans` currently has 3 digits. We split this block to grab the first 2 digits, and then append another block with the last 2 digits, which include the last digit from the previous block and another digit from the end of the number string.

   ```
   if n % 3 == 1:
       ans[-1] = ans[-1][:2]
       ans.append(number[-2:])
   ```

   - If the remainder is 2 (`n % 3 == 2`), we simply append one last block with the final two digits.

   ```
   elif n % 3 == 2:
       ans.append(number[-2:])
   ```

4. **Joining the Blocks with Dashes:** The final step is to join all the blocks with dashes to give us the formatted phone number. The `join` method is perfect for this purpose, turning our list of blocks into a single string.

```
return "-".join(ans)
```

The described solution approach efficiently handles the process using simple built-in operations—a combination of string manipulations through `replace`, `slice`, and `join`, along with list comprehension for constructing sub-strings—delivering an elegant and intuitive algorithm for the given task.

Example Walkthrough

Let's take the following phone number string as an example: "`123-45 6789`". We want to reformat it according to the rules in the problem description.

1. **Removing Non-numeric Characters:** First, we remove spaces and dashes.

```
Original: "123-45 6789"
Cleaned: "123456789"
```

2. **Creating Groups of Three:** We then create blocks of three digits from the cleaned number:

```
Blocks of three: ["123", "456"]
Remaining digits: "789"
```

3. **Handling Remaining Digits:** Since we have 3 remaining digits ("789"), we can add them as a single block:

```
Final blocks: ["123", "456", "789"]
```

4. **Joining the Blocks with Dashes:** Now, we combine the blocks with dashes to get the final phone number format:

```
Result: "123-456-789"
```

If we had a different number, say "`123-45 678`", the process would differ in step 3, where we handle the remaining digits:

1. **Removing Non-numeric Characters:**

```
Original: "123-45 678"
Cleaned: "12345678"
```

2. **Creating Groups of Three:**

```
Blocks of three: ["123", "456"]
Remaining digits: "78"
```

3. **Handling Remaining Digits:** Since there are 2 remaining digits, they form the final block on their own:

```
Final blocks: ["123", "456", "78"]
```

4. **Joining the Blocks with Dashes:**

```
Result: "123-456-78"
```

In both cases, the solution approach systematically applies the reformatting rules to produce the correct phone number format.

Solution Implementation

Python

```python
class Solution:
    def reformatNumber(self, number: str) -> str:
        # Remove dashes and spaces from the input string
        cleaned_number = number.replace("-", "").replace(" ", "")

        # Calculate the length of the cleaned number
        length = len(cleaned_number)

        # Break the cleaned number into chunks of three and store them in a list
        chunks_of_three = [cleaned_number[i * 3: i * 3 + 3] for i in range(length // 3)]

        # Depending on the remainder of the length divided by 3, process the end of the number
        if length % 3 == 1:
            # When there is one digit left after dividing into chunks of three,
            # we need to break the last chunk into two digits and two digits
            last_chunk = chunks_of_three[-1][:2]  # Get first two digits of the last chunk
            second_last_chunk = cleaned_number[-4:-2]  # Get the two digits before the last chunk
            chunks_of_three[-1] = second_last_chunk  # Replace the last chunk with the two digits before it
            chunks_of_three.append(last_chunk + cleaned_number[-2:])  # Add the final two pairs of two digits
        elif length % 3 == 2:
            # When there are two digits left after dividing into chunks of three,
            # simply append them as the final chunk.
            chunks_of_three.append(cleaned_number[-2:])

        # Join the chunks with dashes and return the reformatted number
        return "-".join(chunks_of_three)
```

Java

```java
class Solution {
    public String reformatNumber(String number) {
        // Remove all dashes and spaces from the input number.
        number = number.replace("-", "").replace(" ", "");

        // Calculate the length of the cleaned number string.
        int length = number.length();

        // Use a list to store the parts of the reformatted number.
        List<String> reformattedParts = new ArrayList<>();

        // Divide the number into chunks of three and add them to the list.
        for (int i = 0; i < length / 3; ++i) {
            reformattedParts.add(number.substring(i * 3, i * 3 + 3));
        }

        // Handle the remaining digits after dividing by three.
        int remainder = length % 3;
        if (remainder == 1) {
            // If there is exactly one digit left, take two from the last block and join with the last digit.
            int lastBlockIndex = reformattedParts.size() - 1;
            // Remove the last block and save the first two digits.
            String lastBlockFirstPart = reformattedParts.remove(lastBlockIndex).substring(0, 2);
            // Add the two digit part back to the list.
            reformattedParts.add(lastBlockFirstPart);
            // Add the final two digits as a separate block.
            reformattedParts.add(number.substring(length - 2));
        } else if (remainder == 2) {
            // If there are two digits left, they form the final block on their own.
            reformattedParts.add(number.substring(length - 2));
        }

        // Join all parts with dashes and return the reformatted number.
        return String.join("-", reformattedParts);
    }
}
```

C++

```cpp
class Solution {
public:
    string reformatNumber(string number) {
        // Remove spaces and hyphens from the input number
        string digitsOnly;
        for (char digit : number) {
            if (digit != ' ' && digit != '-') {
                digitsOnly.push_back(digit);
            }
        }

        // Determine the length of the new string with only digits
        int length = digitsOnly.size();
        vector<string> formattedParts;
        // We divide the digits into blocks of 3
        for (int i = 0; i < length / 3; ++i) {
            formattedParts.push_back(digitsOnly.substr(i * 3, 3));
        }

        // Handle the remainder of the digits after division by 3
        if (length % 3 == 1) {
            // If there's one digit left after division by 3, we split the last block into two blocks of 2
            // So we need to adjust the last block to have two digits
            formattedParts.back() = formattedParts.back().substr(0, 2);
            // Then, we take the last two digits and put them into a new block
            formattedParts.push_back(digitsOnly.substr(length - 2));
        } else if (length % 3 == 2) {
            // If there are two digits left, simply add them as a block
            formattedParts.push_back(digitsOnly.substr(length - 2));
        }

        // Combine the blocks into final formatted number with hyphens
        string reformattedNumber;
        for (auto& part : formattedParts) {
            reformattedNumber += part;
            reformattedNumber += "-";
        }
        // Remove the trailing hyphen
        reformattedNumber.pop_back();

        return reformattedNumber;
    }
};
```

TypeScript

```typescript
function reformatNumber(number: string): string {
    // Filter out spaces and hyphens from the input string.
    const cleanedNumbers = [...number].filter(c => c !== ' ' && c !== '-');
    const totalDigits = cleanedNumbers.length;

    // Transform the array of cleaned digits to a string formatted in blocks of digits, separated by hyphens.
    return cleanedNumbers
        .map((digit, index) => {
            // Add a hyphen after every third digit except in specific conditions.
            // Condition 1: When last block can be split into 2 blocks of 2 digits, add a hyphen before last 4 digits.
            // Condition 2: Add a hyphen, if the current position is such that it's not the last or second last block.
            if (
                (index !== 0 && (index + 1) % 3 === 0 && index < totalDigits - 2) ||
                (totalDigits % 3 === 1 && index === totalDigits - 4)
            ) {
                return digit + '-';
            }
            return digit;
        })
        .join(''); // Join the transformed array elements into a string.
}
```


```python
class Solution:
    def reformatNumber(self, number: str) -> str:
        # Remove dashes and spaces from the input string
        cleaned_number = number.replace("-", "").replace(" ", "")

        # Calculate the length of the cleaned number
        length = len(cleaned_number)

        # Break the cleaned number into chunks of three and store them in a list
        chunks_of_three = [cleaned_number[i * 3: i * 3 + 3] for i in range(length // 3)]

        # Depending on the remainder of the length divided by 3, process the end of the number
        if length % 3 == 1:
            # When there is one digit left after dividing into chunks of three,
            # we need to break the last chunk into two digits and two digits
            last_chunk = chunks_of_three[-1][:2]  # Get first two digits of the last chunk
            second_last_chunk = cleaned_number[-4:-2]  # Get the two digits before the last chunk
            chunks_of_three[-1] = second_last_chunk  # Replace the last chunk with the two digits before it
            chunks_of_three.append(last_chunk + cleaned_number[-2:])  # Add the final two pairs of two digits
        elif length % 3 == 2:
            # When there are two digits left after dividing into chunks of three,
            # simply append them as the final chunk.
            chunks_of_three.append(cleaned_number[-2:])

        # Join the chunks with dashes and return the reformatted number
        return "-".join(chunks_of_three)
```


Time and Space Complexity

The time complexity of the code is `O(n)`, where `n` is the length of the input string. This complexity arises from the iteration and replacement methods called on the input string to remove spaces and hyphens, as well as the slicing to create the new format. Each of these operations has a linear complexity relative to the size of the input. The loop to construct the `ans` list can iterate at most `n/3` times, each iteration performing a constant amount of work, contributing to linear complexity as well.

The space complexity of the code is also `O(n)`. This is because we create a new string that has approximately the same length as the input when we remove non-numeric characters. The `ans` list itself will contain at most `n/3 + 2` elements (in the case where the remaining length after slicing in threes is 1), which still contributes linearly to the space complexity due to the storage requirements of these substrings.