1844. Replace All Digits with Characters

Easy String Leetcode Link

Problem Description

In this problem, we are given a string where lowercase English letters are placed at even indices (0, 2, 4, ...), and digits at odd indices (1, 3, 5, ...). We need to transform this string by using a special `shift` function. The `shift` function takes a character `c` and a digit `x`, and returns the character which is `x` positions after `c` in the alphabet sequence.

The task is to replace every digit in the string at an odd index `i` with the character obtained by shifting the preceding letter (at index `i-1`) by the digit at `i`. The process will transform the string into a new version with no digits, and only letters. Additionally, we are assured that applying the `shift` operation will not produce a character beyond 'z'.

For example, if we have `s = "a1b2"`, after replacement, the resulting string should be `s = "abbc"`, because 'a' shifted by 1 position gives 'b', and 'b' shifted by 2 positions gives 'c'.

Intuition

To solve this problem, we can iterate over the string and perform the `shift` operation on every odd index. We know that the indices of the string start at 0, and we can access each character of the string using these indices. The key idea is to convert each digit at an odd index `i` into the corresponding shift character by using the previous character at index `i-1`. We repeat this procedure for all the odd indices in the string.

For the actual shift operation, we leverage the ASCII values of characters. The ASCII value of a letter can be obtained with Python's `ord()` function, and we can add the digit directly to this value because the digit at odd indices is already a number. We must convert the digit from string to integer before adding. After shifting, we can get the character back from its ASCII value using the `chr()` function. We replace the digit with this new character in the string.

To facilitate these operations, we convert the string `s` into a list because strings in Python are immutable, while lists are mutable. After iterating through the string and applying the shift operation to all digits, we join the list to form the final transformed string with no digits left.

Solution Approach

The core of our approach is based on the idea that we need to modify only the digits in the original string, and each digit needs to be offset by its own value from the preceding character. The flow of the solution is as follows:

1. Convert the string `s` into a list to allow for modification, since Python strings are immutable and don't allow changes to individual characters directly.
2. Iterate over the list starting from the first odd index (which is 1) to the end of the list, and increment by 2 each time to only visit the odd indices where the digits are located.
3. Retrieve the character that comes before the current digit (`s[i - 1]`) and find its ASCII value using `ord()`. ASCII values are numeric representations of characters that can be manipulated mathematically.
4. Convert the current digit (`s[i]`) from a string to an integer so that it can be used in the arithmetic operation.
5. Calculate the ASCII value of the new character by adding the integer value of the digit to the ASCII value of the preceding character.
6. Convert the new ASCII value back into a character using `chr()`.
7. Replace the digit in the list with the new character.
8. After the iteration, join the elements of the list together into a string using `''.join(s)`.

In this solution:

- A **for loop** is employed to iterate over the string indices. The range starts at 1 (the first odd index) and goes up to `len(s)`, which is the length of the string, with steps of 2 to ensure we're only looking at the odd indices.
- Python's built-in **`ord` function** is used to get the ASCII value of a character. It converts the given string of length 1 to an integer representing the Unicode code point of the character. For example, `ord('a')` returns 97 which is the Unicode point for the character 'a'.
- Python's built-in **`chr` function** is used to convert the adjusted ASCII value back into a character. This function returns a string representing a character whose Unicode code point is the integer. For example, `chr(98)` returns 'b'.
- An implicit **data structure**, a list, is used to allow mutation of the string.

Here is a step-by-step description of the algorithm applied within the `for` loop for the digit at odd index `i`:

```
1. ASCII_preceding_char = ord(s[i - 1])
2. offset = int(s[i])
3. ASCII_new_char = ASCII_preceding_char + offset
4. new_char = chr(ASCII_new_char)
5. s[i] = new_char
```

These steps are repeated until every digit at an odd index in the given string is replaced with a corresponding character as per the `shift` function. This is the essence of the solution, and the output is the modified string.

Example Walkthrough

Let's consider a small example using the string `s = "a5c3"` to illustrate the solution approach described above.

1. First, we convert the string `s` into a list to allow modifications. The string as a list will be `['a', '5', 'c', '3']`.
2. We will then start iterating over the list from the first odd index, which is 1. This index points to the digit '5'.
3. According to the steps in the solution approach, we first find the ASCII value of the preceding character 'a' (which is at index 0):

   ```
   ASCII_preceding_char = ord('a') // ASCII_preceding_char = 97
   ```

4. We then convert the current digit '5' into an integer and add it to the ASCII value of the preceding character:

   ```
   offset = int('5') // offset = 5
   ASCII_new_char = ASCII_preceding_char + offset // ASCII_new_char = 97 + 5 = 102
   ```

5. Next, we convert the ASCII value back into a character:

   ```
   new_char = chr(ASCII_new_char) // new_char = chr(102) = 'f'
   ```

6. Replace the digit with the new character in our list:

   ```
   ['a', '5', 'c', '3'] becomes ['a', 'f', 'c', '3']
   ```

7. We move to the next odd index, which is 3. Repeat the steps:

   ```
   ASCII_preceding_char = ord('c') // ASCII_preceding_char = 99
   offset = int('3') // offset = 3
   ASCII_new_char = ASCII_preceding_char + offset // ASCII_new_char = 99 + 3 = 102
   new_char = chr(ASCII_new_char) // new_char = chr(102) = 'f'
   ```

8. We replace the digit at index 3 with the new character in our list:

   ```
   ['a', 'f', 'c', '3'] becomes ['a', 'f', 'c', 'f']
   ```

9. After completing the iteration for all odd indices, we join the elements of the list into a string:

   ```
   ''.join(['a', 'f', 'c', 'f']) // The final transformed string is "afcf"
   ```

And there we have it! The original string `s = "a5c3"` has been successfully transformed into `s = "afcf"` following the provided solution approach, correctly applying the shift operation to the digits.

Python Solution

```python
class Solution:
    def replaceDigits(self, s: str) -> str:
        # Convert the string to a list of characters for easy manipulation
        char_list = list(s)

        # Iterate over the character list, jumping in steps of 2 (starting from index 1)
        for i in range(1, len(char_list), 2):
            # Replace the digit at the odd index with a char that is `digit` away from the char at the even index
            # before it, using ASCII values.
            char_list[i] = chr(ord(char_list[i - 1]) + int(char_list[i]))

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

```

Java Solution

```java
class Solution {
    // This method replaces all digits in the string with a character shifted by the digit value.
    public String replaceDigits(String s) {
        // Convert the input string into a character array for easy manipulation.
        char[] charArray = s.toCharArray();

        // Loop through the character array starting from index 1, incrementing by 2,
        // to process only the digit characters.
        for (int i = 1; i < charArray.length; i += 2) {
            // Shift the character before the digit ('0' through '9') by the digit value.
            // The digit character is converted to the actual digit by subtracting '0'.
            // Then add this numerical value to the preceding character to shift it.
            charArray[i] = (char) (charArray[i - 1] + (charArray[i] - '0'));
        }

        // Convert the altered character array back to a string and return it.
        return String.valueOf(charArray);
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to replace digits in a string with characters shifted accordingly
    string replaceDigits(string s) {
        // Determine the size of the string to iterate through it
        int strSize = s.size();

        // Loop through the string, incrementing by 2 to only handle the digits
        for (int i = 1; i < strSize; i += 2) {
            // Replace each digit at an odd index with the character that is
            // shifted by the digit's value from the previous character.
            // The expression s[i] - '0' converts the digit character to its
            // integer value, and we add this value to the previous character.
            s[i] = s[i - 1] + (s[i] - '0');
        }

        // Return the modified string
        return s;
    }
};

```

Typescript Solution

```typescript
function replaceDigits(s: string): string {
    // Get the length of the input string
    const stringLength = s.length;

    // Split the input string into an array to manipulate individual characters
    const answerArray = [...s];

    // Iterate over the characters of the string, skipping every other character
    for (let index = 1; index < stringLength; index += 2) {
        // Replace the current character (digit) with the character that is 'digit' positions
        // after the letter that precedes it in the alphabet.
        // 'charCodeAt(0)' gets the ASCII code of the letter at 'index - 1',
        // 'Number(answerArray[index])' converts the character at 'index' into a number,
        // and 'String.fromCharCode' converts the new ASCII code back to a character.
        answerArray[index] = String.fromCharCode(answerArray[index - 1].charCodeAt(0) + Number(answerArray[index]));
    }

    // Join the array of characters back into a string and return it
    return answerArray.join('');
}

```

Time and Space Complexity

Time Complexity

The time complexity of the code is `O(n)`, where `n` is the length of the string `s`. This is because the for loop iterates over every other element of the string, performing a constant-time operation for half the characters (the digits). Each operation inside the loop (calculating `ord`, adding an integer, converting back to a character with `chr`, and assigning it back to the list) takes constant time. Therefore, the time taken is directly proportional to the number of characters in the string.

Space Complexity

The space complexity of the code is `O(n)`, as the input string `s` is converted to a list, which takes up space proportional to the length of `s`. No additional data structures are used that grow with the input size, and the space used for the output is the same as the space used to store the modified list created from the input string.