709. To Lower Case

Easy String

Leetcode Link

Problem Description

The problem requires us to take an input string `s` and return a new string where all uppercase letters have been converted to their corresponding lowercase counterparts. The task focuses on modifying the letter cases without altering any other characters in the string. For instance, if the input string is `"LeetCode"`, the output should be `"leetcode"`.

Intuition

To solve this problem, we need to iterate over each character in the string `s` and check if it's an uppercase letter. If it is, we convert it to lowercase. The solution uses a list comprehension to create a new list of characters, where each uppercase character is converted by using the bitwise `OR` operator with `32`. This works because, in the ASCII character encoding, adding `32` to the ASCII value of an uppercase letter gives us the ASCII value of the corresponding lowercase letter. The `chr(ord(c) | 32)` expression inside the list comprehension checks if a character `c` is uppercase using the `isupper()` method, and if it is, applies the `| 32` operation which adds `32` to the character's ASCII value, effectively converting it to lowercase.

This approach is efficient because:

1. It avoids branching logic like if-else statements within the list comprehension, which can be slower in Python.
2. The bitwise operation is an efficient way to convert characters from uppercase to lowercase.
3. The use of a list comprehension allows for the creation of the new lowercased string in a single line of code.

The final string is obtained by joining all characters in the resultant list with an empty string separator `""`.

Solution Approach

The implementation uses a straightforward and efficient algorithm to convert uppercase letters to lowercase. The process involves the following steps:

1. Iterate over each character in the input string `s` using a list comprehension.
2. For each character `c`, check if it is an uppercase letter by calling the `isupper()` method.
3. If `c` is uppercase, apply the bitwise OR operation `|` with `32` to the ASCII value of `c` to get the ASCII value of the lowercase equivalent. This is done by first using `ord(c)` to get the ASCII value of `c`, then applying the bitwise operation, and finally converting back to a character with `chr()`.
4. If `c` is not uppercase, it is left unchanged.
5. The list comprehension creates a list of characters, with uppercase characters now converted to lowercase.
6. The final lowercase string is produced by concatenating all characters in the list using the `join` method with an empty string `""` as the separator.

No additional data structures are required aside from the list created by the list comprehension and the final string returned.

This pattern is based on the property of ASCII values for letters:

- The ASCII values of uppercase and lowercase letters have a specific numerical relationship: the difference between the ASCII values of an uppercase letter and its corresponding lowercase letter is exactly 32.
- The bitwise OR operation with `32` effectively adds `32` to the ASCII value if the number is within the range of uppercase letters in ASCII (since their binary representations are such that the value `32` would only change the bit corresponding to the lowercase bit).

Here is the core line of code that performs the conversion:

```
return "".join([chr(ord(c) | 32) if c.isupper() else c for c in s])
```

This line is Pythonic and leverages the features of the language for a concise and efficient implementation.

Example Walkthrough

Let's illustrate the solution approach with a simple example.

Suppose our input string `s` is `"Python3.8"`. We aim to convert any uppercase letters to lowercase while leaving other characters unchanged. To make it easier to follow, let's walk through the process step by step for this input:

1. The string "Python3.8" is composed of characters: 'P', 'y', 't', 'h', 'o', 'n', '3', '.', and '8'.
2. We start iterating over each character, starting with 'P'.
3. 'P' is an uppercase letter, so we check its ASCII value. The ASCII value for 'P' is 80.
4. We then apply the bitwise OR operation (`|`) with `32`, which is `80 | 32`. The binary representation of 80 is `1010000`, and the binary representation of 32 is `0100000`. The OR operation yields `1110000`, which is the binary representation of 112 - the ASCII value of 'p'.
5. We convert the ASCII value back to a character with `chr()`, resulting in 'p'.
6. We continue the process with 'y', which isn't an uppercase letter, so it remains unchanged.
7. We proceed with the rest of the characters, and since there are no more uppercase letters, they all remain unchanged.
8. After processing all characters, we obtain a list of characters: ['p', 'y', 't', 'h', 'o', 'n', '3', '.', '8'].
9. We join all characters in the list to produce the final string, using `"".join([...])`.
10. The final output is `"python3.8"`.

The core line of Python code that performs this operation looks like this:

```
return "".join([chr(ord(c) | 32) if c.isupper() else c for c in s])
```

For our example `"Python3.8"`, this line of code will return `"python3.8"`, as expected, with the uppercase 'P' converted to a lowercase 'p'.

Python Solution

```
class Solution:
    def toLowerCase(self, string: str) -> str:
        # Create a lowercase version of the given string
        # by iterating over each character in the string
        lowercase_string = "".join([
            # Convert uppercase letters to lowercase by using bitwise OR with 32
            # which effectively adds 32 to ASCII value of uppercase letters to get
            # their lowercase counterparts because in ASCII table, the difference
            # between uppercase and lowercase letters is 32
            chr(ord(char) | 32) if 'A' <= char <= 'Z' else char
            for char in string
        ])
        return lowercase_string

```

Java Solution

```
class Solution {

    // Method to convert all uppercase letters in a string to lowercase
    public String toLowerCase(String str) {
        // Convert the input string to an array of characters
        char[] characters = str.toCharArray();

        // Iterate over each character of the array
        for (int i = 0; i < characters.length; ++i) {
            // Check if the current character is an uppercase letter
            if (characters[i] >= 'A' && characters[i] <= 'Z') {
                // Perform a bitwise OR operation with 32 to convert
                // the uppercase letter to a lowercase letter
                characters[i] |= 32;
            }
        }

        // Return the new string with all characters converted to lowercase
        return String.valueOf(characters);
    }
}

```

C++ Solution

```
class Solution {
public:
    // Function to convert a string to lowercase
    string toLowerCase(string str) {
        // Iterate over each character in the string
        for (char& character : str) {
            // Check if the character is an uppercase letter
            if (character >= 'A' && character <= 'Z') {
                // Convert it to lowercase by adding the difference in ASCII values
                character |= 32; // Bitwise OR operation with 32 to make the character lowercase
            }
        }
        // Return the modified string
        return str;
    }
};

```

Typescript Solution

```
/**
 * Converts a string to lowercase.
 * @param {string} str - The string to be converted to lowercase.
 * @returns {string} The lowercase equivalent of the input string.
 */
function toLowerCase(str: string): string {
    // Convert the string to an array of characters, then map each character
    // to its lowercase equivalent by OR-ing the character's char code with 32.
    // This works because in the ASCII table, the difference between uppercase
    // and lowercase letters is 32.
    // Finally, join the array back into a string.
    const lowerCaseString = [...str].map(char =>
        // Convert char to lowercase by OR-ing with 32, and then convert back to string
        String.fromCharCode(char.charCodeAt(0) | 32)
    ).join('');

    // Return the resulting lowercase string
    return lowerCaseString;
}

```

Time and Space Complexity

The given Python code snippet converts each uppercase letter in a string to lowercase by using a bitwise OR operation with the space character, which is 32 in decimal (`' '` has an ASCII value of 32). The bit manipulation here relies on the fact that in ASCII, setting the sixth bit of an uppercase letter's ASCII value to 1 will convert it to its lowercase equivalent. Here's how the code breaks down in terms of time and space complexity:

Time Complexity:

- The function iterates over each character in the string `s` once.
- During each iteration, it performs a constant-time check to see if the character is uppercase (`c.isupper()`) and then applies the bitwise operation (`ord(c) | 32`) if needed.
- The overall time complexity is `O(n)`, where `n` is the length of the input string `s`. This is because the operations inside the loop are constant time and the loop runs `n` times, where n is the number of characters in the string.

Space Complexity:

- The space complexity of the code is also `O(n)`. This is due to the list comprehension creating a new list that stores the resultant characters before joining them into a final string. The size of this list scales linearly with the number of characters in the input string, making the space required proportional to `n`.