2222. Number of Ways to Select Buildings

Medium String Dynamic Programming Prefix Sum Leetcode Link

Problem Description

In LeetCode's problem, you are provided with a binary string `s` representing a street of buildings. A '0' represents an office, and a '1' represents a restaurant. The task is to count the number of valid ways to select a sequence of three buildings for inspection. However, a valid sequence cannot include two consecutive buildings of the same type. This means you cannot have "000" or "111" as part of your selected sequence. For example, if `s = "0101"`, there are two valid sequences: "010" and "101". The goal is to calculate the total number of such valid sequences across the entire string.

Intuition

The intuition behind the solution is to leverage the constraints provided. Since we cannot select two buildings of the same type consecutively, a valid sequence can only be "010" or "101". We need to count the occurrences of these patterns.

We know the total counts of '0's and '1's in the string upfront as `cnt0` and `cnt1`. For each character in the string, if it's a '0', then it can be the middle building in a "101" pattern. The number of valid patterns with this '0' in the middle is the count of '1's encountered so far times the count of '1's that are yet to be seen (`cnt1 - c1`). Similarly, if the character is a '1', it can be the middle building in a "010" pattern, and the number of valid patterns is the count of '0's seen so far times the count of '0's yet to be seen (`cnt0 - c0`).

By iterating over the string and updating the counts of '0's and '1's seen so far (`c0` and `c1`), we can accumulate the number of valid sequences into `ans`.

Solution Approach

The solution makes use of a single-pass algorithm to count the number of valid sequences. Here is a step-by-step breakdown of the algorithm using the code provided:

1. Calculate the total number of '0's (`cnt0`) in the string: `cnt0 = s.count("0")`.
2. Calculate the total number of '1's (`cnt1`) by subtracting `cnt0` from the length of the string, `n`.
3. Initialize two counters `c0` and `c1` to keep track of the number of '0's and '1's encountered so far as we iterate through the string.
4. Initialize a variable `ans` to accumulate the answer, which is the total count of valid ways.
5. Iterate over each character `c` in the string `s`:
   - If `c` is '0', it can potentially be the middle of a "101" sequence. The number of such sequences involving this particular '0' is `c1` (the number of '1's already seen) multiplied by `(cnt1 - c1)` (the number of '1's after this '0'). Add this to `ans`.
   - Increment `c0` to indicate that we've seen an additional '0'.
   - If `c` is '1', the same logic applies, only now it could be the center of a "010" sequence. Calculate the valid sequences by multiplying `c0` by `(cnt0 - c0)` and add it to `ans`.
   - Increment `c1` accordingly.
6. Once the iteration is complete, `ans` will contain the total number of valid sequences, which we return.

This approach is efficient because it requires only a single pass through the string, and thus has a time complexity of O(n), where n is the length of the string. There is no need for nested loops, which would increase the complexity. The space complexity is O(1) since only a constant amount of extra space is used—no additional data structures are needed. The insight that each '0' or '1' can potentially be the center of a valid sequence, and the precalculated total counts of each type of building, allow us to compute the answer with this direct method.

Example Walkthrough

Let's walk through an example to illustrate the solution approach, using the binary string `s = "0101010"`.

First, we count the number of '0's (`cnt0`) and '1's (`cnt1`) in the string:

- `cnt0 = s.count("0") = 4`
- `cnt1 = len(s) - cnt0 = 7 - 4 = 3`

Now, we initialize two counters `c0` and `c1` to keep track of the number of '0's and '1's encountered as we iterate:

- `c0 = 0`
- `c1 = 0`

We also initialize `ans` to accumulate the total count of valid ways:

- `ans = 0`

Now, we start iterating over the string `s`.

- Iteration 1: We encounter a '0'. It cannot form a "101" sequence as no '1's have been seen so far (`c1 = 0`). We increment `c0`.
  - `c0 = 1`
  - `c1 = 0`
  - `ans` remains `0`
- Iteration 2: We encounter a '1'. It can be the middle of a "010" sequence. There is 1 '0' before it and 3 '0's after it.
  - Valid "010" sequences with this '1': `c0 * (cnt0 - c0) = 1 * (4 - 1) = 3`
  - `c0 = 1`
  - `c1 = 1`
  - `ans = ans + 3 = 3`
- Iteration 3: We encounter a '0'. We've seen 1 '1' so far and have 2 '1's remaining.
  - Valid "101" sequences with this '0': `c1 * (cnt1 - c1) = 1 * (3 - 1) = 2`
  - `c0 = 2`
  - `c1 = 1`
  - `ans = ans + 2 = 5`
- Iteration 4: We encounter a '1'. There are 2 '0's before and 2 '0's after.
  - Valid "010" sequences with this '1': `c0 * (cnt0 - c0) = 2 * (4 - 2) = 4`
  - `c0 = 2`
  - `c1 = 2`
  - `ans = ans + 4 = 9`
- Iteration 5: We encounter a '0'. Now we have 2 '1's before and 1 '1' after.
  - Valid "101" sequences with this '0': `c1 * (cnt1 - c1) = 2 * (3 - 2) = 2`
  - `c0 = 3`
  - `c1 = 2`
  - `ans = ans + 2 = 11`
- Iteration 6: We encounter a '1'. There are 3 '0's before and 1 '0' after.
  - Valid "010" sequences with this '1': `c0 * (cnt0 - c0) = 3 * (4 - 3) = 3`
  - `c0 = 3`
  - `c1 = 3`
  - `ans = ans + 3 = 14`
- Iteration 7: We encounter the final '0'. There are 3 '1's before but no '1's are after, so it cannot form a "101" sequence.
  - `c0 = 4`
  - `c1 = 3`
  - `ans` remains `14`

After completing the iteration, `ans = 14`. So there are 14 valid ways to select sequences of buildings for inspection from the given string "0101010".

Python Solution

```python
class Solution:
    def numberOfWays(self, s: str) -> int:
        # Calculate the length of the string.
        length_of_string = len(s)

        # Count the number of '0's in the string.
        count_of_zeros = s.count("0")

        # Calculate the number of '1's in the string by subtracting the number of '0's from the total length.
        count_of_ones = length_of_string - count_of_zeros

        # Initialize counters for the number of '0's and '1's that have been encountered so far.
        running_count_zeros = 0
        running_count_ones = 0

        # Initialize the answer to track the number of ways.
        number_of_ways = 0

        # Iterate over the characters in the string.
        for char in s:
            if char == "0":
                # If the current character is '0', calculate the contribution to the number of ways by considering
                # the number of '1's encountered so far and the number of '1's yet to be encountered.
                number_of_ways += running_count_ones * (count_of_ones - running_count_ones)

                # Increment the counter for '0's encountered.
                running_count_zeros += 1
            else:
                # If the current character is '1', calculate the contribution to the number of ways by considering
                # the number of '0's encountered so far and the number of '0's yet to be encountered.
                number_of_ways += running_count_zeros * (count_of_zeros - running_count_zeros)

                # Increment the counter for '1's encountered.
                running_count_ones += 1

        # Return the total number of ways.
        return number_of_ways

```

Java Solution

```java
class Solution {
    // Method to count the number of ways to form a "010" or "101" pattern in the given string.
    public long numberOfWays(String s) {
        // Length of the input string.
        int length = s.length();
        // Counter for zeros in the input string.
        int countZeros = 0;

        // Count the number of zeros in the input string.
        for (char c : s.toCharArray()) {
            if (c == '0') {
                countZeros++;
            }
        }

        // Counter for ones, which is the total length minus the number of zeros.
        int countOnes = length - countZeros;
        // Variable to store the total number of patterns found.
        long totalWays = 0;
        // Temp counters for zeros and ones as we iterate through the string.
        int tempCountZeros = 0, tempCountOnes = 0;

        // Iterate through the characters of the string to count the patterns.
        for (char c : s.toCharArray()) {
            if (c == '0') {
                // When we find a '0', we increase the total count of valid patterns found
                // by the number of '1's found before multiplied by the number of '1's that
                // can potentially come after this '0' to complete the pattern.
                totalWays += tempCountOnes * (countOnes - tempCountOnes);
                // Increase the temporary count of zeros since we encountered a '0'.
                tempCountZeros++;
            } else {
                // Similarly, when we find a '1', we increase the count of valid patterns by
                // the temporary count of '0's multiplied by the number of '0's that can come
                // after to complete the pattern.
                totalWays += tempCountZeros * (countZeros - tempCountZeros);
                // Increase the temporary count of ones since we encountered a '1'.
                tempCountOnes++;
            }
        }

        // Return the total number of patterns found.
        return totalWays;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    long long numberOfWays(string s) {
        // Get the length of the string
        int stringLength = s.size();

        // Initialize count of zeros in the string
        int countOfZeros = 0;

        // Loop through the string to count the number of zeros
        for (char character : s) {
            countOfZeros += character == '0';
        }

        // Calculate the count of ones as the remaining characters
        int countOfOnes = stringLength - countOfZeros;

        // Initialize counters for zeros and ones processed so far
        int zerosProcessed = 0, onesProcessed = 0;

        // Initialize the answer which will store the total number of ways
        long long totalWays = 0;

        // Loop through the string to calculate the total count of valid sequences
        for (char character : s) {
            if (character == '0') {
                // For each zero, pair it with previously encountered ones
                // and potential ones that could come later in the string
                totalWays += onesProcessed * (countOfOnes - onesProcessed);
                // Increment the count of processed zeros
                ++zerosProcessed;
            } else {
                // For each one, pair it with previously encountered zeros
                // and potential zeros that could come later in the string
                totalWays += zerosProcessed * (countOfZeros - zerosProcessed);
                // Increment the count of processed ones
                ++onesProcessed;
            }
        }

        // Return the total number of valid ways to order substrings "010" and "101"
        return totalWays;
    }
};

```

Typescript Solution

```typescript
// TypeScript function to count number of ways to split a string into "010" and "101" substrings
function numberOfWays(s: string): number {
    // Get the length of the string
    let stringLength: number = s.length;

    // Initialize count of zeros in the string
    let countOfZeros: number = 0;

    // Loop through the string to count the number of zeros
    for (let character of s) {
        countOfZeros += character === '0' ? 1 : 0;
    }

    // Calculate the count of ones as the remaining characters
    let countOfOnes: number = stringLength - countOfZeros;

    // Initialize counters for zeros and ones processed so far
    let zerosProcessed: number = 0;
    let onesProcessed: number = 0;

    // Initialize the total number of ways to pair "010" and "101"
    let totalWays: number = 0;

    // Loop through the string to calculate the total count of valid sequences
    for (let character of s) {
        if (character === '0') {
            // For each zero, pair it with previously encountered ones
            // and potential ones that could come later in the string
            totalWays += onesProcessed * (countOfOnes - onesProcessed);
            // Increment the count of processed zeros
            zerosProcessed++;
        } else {
            // For each one, pair it with previously encountered zeros
            // and potential zeros that could come later in the string
            totalWays += zerosProcessed * (countOfZeros - zerosProcessed);
            // Increment the count of processed ones
            onesProcessed++;
        }
    }

    // Return the total number of valid ways to order substrings "010" and "101"
    return totalWays;
}

// Example Usage
// const ways: number = numberOfWays("001101");

```

Time and Space Complexity

The given Python code aims to count the number of ways to select three characters from the string `s`, such that the selected characters form the pattern "010" or "101".

Time Complexity

The time complexity of the code is O(`n`), where `n` is the length of the string `s`.

- Calculating `cnt0` using `s.count("0")` requires a single pass over the string, which is O(`n`).
- The subsequent loop iterates over each character in the string once, which is O(`n`).
- Inside the loop, the operations are constant time, such as updating counters and calculating the values for `ans`.

Therefore, the overall time complexity is the sum of the two O(`n`) operations, which is still O(`n`) since constant factors are ignored.

Space Complexity

The space complexity of the code is O(1).

- The variables `cnt0`, `cnt1`, `c0`, `c1`, and `ans` are all integer counters that use a fixed amount of space.
- No additional data structures that grow with the input size are used.

Thus, the space required does not scale with the size of the input `s`, resulting in a constant space complexity.