Problem 1375: Number of Times Binary String Is Prefix-Aligned

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You have a

1-indexed

binary string of length

n

where all the bits are

0

initially. We will flip all the bits of this binary string (i.e., change them from

0

to

1

) one by one. You are given a

1-indexed

integer array

flips

where

flips[i]

indicates that the bit at index

flips[i]

will be flipped in the

i

th

step.

A binary string is

prefix-aligned

if, after the

i

th

step, all the bits in the

inclusive

range

[1, i]

are ones and all the other bits are zeros.

Return

the number of times the binary string is

prefix-aligned

during the flipping process

.

Example 1:

Input:

flips = [3,2,4,1,5]

Output:

2

Explanation:

The binary string is initially "00000". After applying step 1: The string becomes "00100", which is not prefix-aligned. After applying step 2: The string becomes "01100", which is not prefix-aligned. After applying step 3: The string becomes "01110", which is not prefix-aligned. After applying step 4: The string becomes "11110", which is prefix-aligned. After applying step 5: The string becomes "11111", which is prefix-aligned. We can see that the string was prefix-aligned 2 times, so we return 2.

Example 2:

Input:

flips = [4,1,2,3]

Output:

1

Explanation:

The binary string is initially "0000". After applying step 1: The string becomes "0001", which is not prefix-aligned. After applying step 2: The string becomes "1001", which is not prefix-aligned. After applying step 3: The string becomes "1101", which is not prefix-aligned. After applying step 4: The string becomes "1111", which is prefix-aligned. We can see that the string was prefix-aligned 1 time, so we return 1.

Constraints:

n == flips.length

1 <= n <= 5 * 10

4

flips

is a permutation of the integers in the range

[1, n]

.

Code Snippets

C++:

```
class Solution {
public:
int numTimesAllBlue(vector<int>& flips) {

}
};
```

Java:

```
class Solution {
public int numTimesAllBlue(int[] flips) {
}
}
```

Python3:

```
class Solution:
def numTimesAllBlue(self, flips: List[int]) -> int:
```

Python:

```
class Solution(object):
def numTimesAllBlue(self, flips):
"""
:type flips: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} flips
* @return {number}
*/
var numTimesAllBlue = function(flips) {

};
```

TypeScript:

```
function numTimesAllBlue(flips: number[]): number {

};
```

C#:

```
public class Solution {
public int NumTimesAllBlue(int[] flips) {

}
}
```

C:

```
int numTimesAllBlue(int* flips, int flipsSize) {

}
```

Go:

```
func numTimesAllBlue(flips []int) int {

}
```

Kotlin:

```
class Solution {
fun numTimesAllBlue(flips: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func numTimesAllBlue(_ flips: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_times_all_blue(flips: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} flips
# @return {Integer}
def num_times_all_blue(flips)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $flips
* @return Integer
*/
function numTimesAllBlue($flips) {

}
}
```

Dart:

```
class Solution {
int numTimesAllBlue(List<int> flips) {

}
}
```

Scala:

```
object Solution {
def numTimesAllBlue(flips: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_times_all_blue(flips :: [integer]) :: integer
def num_times_all_blue(flips) do

end
end
```

Erlang:

```
-spec num_times_all_blue(Flips :: [integer()]) -> integer().
num_times_all_blue(Flips) ->
.
```

Racket:

```
(define/contract (num-times-all-blue flips)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Times Binary String Is Prefix-Aligned
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int numTimesAllBlue(vector<int>& flips) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Times Binary String Is Prefix-Aligned
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int numTimesAllBlue(int[] flips) {
}
}
```

Python3 Solution:

```
"""
Problem: Number of Times Binary String Is Prefix-Aligned
Difficulty: Medium
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def numTimesAllBlue(self, flips: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numTimesAllBlue(self, flips):
"""
:type flips: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Times Binary String Is Prefix-Aligned
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```javascript
/**
* @param {number[]} flips
* @return {number}
*/
var numTimesAllBlue = function(flips) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Number of Times Binary String Is Prefix-Aligned
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function numTimesAllBlue(flips: number[]): number {

};
```

C# Solution:

```csharp
/*
* Problem: Number of Times Binary String Is Prefix-Aligned
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int NumTimesAllBlue(int[] flips) {

}

}
```

C Solution:

```
/*
* Problem: Number of Times Binary String Is Prefix-Aligned
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int numTimesAllBlue(int* flips, int flipsSize) {

}
```

Go Solution:

```
// Problem: Number of Times Binary String Is Prefix-Aligned
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func numTimesAllBlue(flips []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numTimesAllBlue(flips: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func numTimesAllBlue(_ flips: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Times Binary String Is Prefix-Aligned
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn num_times_all_blue(flips: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} flips
# @return {Integer}
def num_times_all_blue(flips)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $flips
* @return Integer
*/
function numTimesAllBlue($flips) {

}
}
```

Dart Solution:

```
class Solution {
int numTimesAllBlue(List<int> flips) {

}
}
```

Scala Solution:

```
object Solution {
def numTimesAllBlue(flips: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_times_all_blue(flips :: [integer]) :: integer
def num_times_all_blue(flips) do

end
end
```

Erlang Solution:

```
-spec num_times_all_blue(Flips :: [integer()]) -> integer().
num_times_all_blue(Flips) ->
.
```

Racket Solution:

```
(define/contract (num-times-all-blue flips)
(-> (listof exact-integer?) exact-integer?)
)
```