Problem 1700: Number of Students Unable to Eat Lunch

Problem Information

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

Problem Description

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers

0

and

1

respectively. All students stand in a queue. Each student either prefers square or circular sandwiches.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a

stack

. At each step:

If the student at the front of the queue

prefers

the sandwich on the top of the stack, they will take it

and leave the queue.

Otherwise, they will

leave it

and go to the queue's end.

This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

You are given two integer arrays

students

and

sandwiches

where

sandwiches[i]

is the type of the

i

th

sandwich in the stack (

i = 0

is the top of the stack) and

students[j]

is the preference of the

j

th

student in the initial queue (

j = 0

is the front of the queue). Return

the number of students that are unable to eat.

Example 1:

Input:

students = [1,1,0,0], sandwiches = [0,1,0,1]

Output:

0

Explanation:

- Front student leaves the top sandwich and returns to the end of the line making students = [1,0,0,1]. - Front student leaves the top sandwich and returns to the end of the line making students = [0,0,1,1]. - Front student takes the top sandwich and leaves the line making students = [0,1,1] and sandwiches = [1,0,1]. - Front student leaves the top sandwich and returns to the end of the line making students = [1,1,0]. - Front student takes the top sandwich and leaves the line making students = [1,0] and sandwiches = [0,1]. - Front student leaves the top sandwich and returns to the end of the line making students = [0,1]. - Front student takes the top sandwich and leaves the line making students = [1] and sandwiches = [1]. - Front student takes the top sandwich and leaves the line making students = [] and sandwiches = []. Hence all students are able to eat.

Example 2:

Input:

students = [1,1,1,0,0,1], sandwiches = [1,0,0,0,1,1]

Output:

3

Constraints:

1 <= students.length, sandwiches.length <= 100

students.length == sandwiches.length

sandwiches[i]

is

0

or

1

.

students[i]

is

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int countStudents(vector<int>& students, vector<int>& sandwiches) {

}
};
```

Java:

```
class Solution {
public int countStudents(int[] students, int[] sandwiches) {

}
}
```

Python3:

```
class Solution:
def countStudents(self, students: List[int], sandwiches: List[int]) -> int:
```

Python:

```
class Solution(object):
def countStudents(self, students, sandwiches):
"""
:type students: List[int]
:type sandwiches: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} students
* @param {number[]} sandwiches
* @return {number}
*/
var countStudents = function(students, sandwiches) {

};
```

TypeScript:

```
function countStudents(students: number[], sandwiches: number[]): number {

};
```

C#:

```
public class Solution {
public int CountStudents(int[] students, int[] sandwiches) {

}
}
```

C:

```
int countStudents(int* students, int studentsSize, int* sandwiches, int
sandwichesSize) {

}
```

Go:

```
func countStudents(students []int, sandwiches []int) int {

}
```

Kotlin:

```
class Solution {
fun countStudents(students: IntArray, sandwiches: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func countStudents(_ students: [Int], _ sandwiches: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_students(students: Vec<i32>, sandwiches: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} students
# @param {Integer[]} sandwiches
# @return {Integer}
def count_students(students, sandwiches)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $students
* @param Integer[] $sandwiches
* @return Integer
*/
function countStudents($students, $sandwiches) {

}
}
```

Dart:

```
class Solution {
int countStudents(List<int> students, List<int> sandwiches) {

}
}
```

Scala:

```
object Solution {
def countStudents(students: Array[Int], sandwiches: Array[Int]): Int = {
}
}
```

Elixir:

```
defmodule Solution do
@spec count_students(students :: [integer], sandwiches :: [integer]) ::
integer
def count_students(students, sandwiches) do

end
end
```

Erlang:

```
-spec count_students(Students :: [integer()], Sandwiches :: [integer()]) ->
integer().
count_students(Students, Sandwiches) ->
.
```

Racket:

```
(define/contract (count-students students sandwiches)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Students Unable to Eat Lunch
* Difficulty: Easy
* Tags: array, stack, queue
*
* 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 countStudents(vector<int>& students, vector<int>& sandwiches) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Students Unable to Eat Lunch
* Difficulty: Easy
* Tags: array, stack, queue
*
* 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 countStudents(int[] students, int[] sandwiches) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Students Unable to Eat Lunch
Difficulty: Easy
Tags: array, stack, queue

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 countStudents(self, students: List[int], sandwiches: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def countStudents(self, students, sandwiches):
"""
:type students: List[int]
:type sandwiches: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Students Unable to Eat Lunch
* Difficulty: Easy
* Tags: array, stack, queue
*
* 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
*/

/**
* @param {number[]} students
* @param {number[]} sandwiches
* @return {number}
*/
var countStudents = function(students, sandwiches) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Students Unable to Eat Lunch
* Difficulty: Easy
* Tags: array, stack, queue
*
* 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 countStudents(students: number[], sandwiches: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Students Unable to Eat Lunch
* Difficulty: Easy
* Tags: array, stack, queue
*
* 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 CountStudents(int[] students, int[] sandwiches) {

}
}
```

C Solution:

```
/*
* Problem: Number of Students Unable to Eat Lunch
* Difficulty: Easy
* Tags: array, stack, queue
*
* 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 countStudents(int* students, int studentsSize, int* sandwiches, int
sandwichesSize) {

}
```

Go Solution:

```
// Problem: Number of Students Unable to Eat Lunch
// Difficulty: Easy
// Tags: array, stack, queue
//
// 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 countStudents(students []int, sandwiches []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countStudents(students: IntArray, sandwiches: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func countStudents(_ students: [Int], _ sandwiches: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Students Unable to Eat Lunch
// Difficulty: Easy
// Tags: array, stack, queue
//
// 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 count_students(students: Vec<i32>, sandwiches: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} students
# @param {Integer[]} sandwiches
# @return {Integer}
def count_students(students, sandwiches)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $students
* @param Integer[] $sandwiches
* @return Integer
*/
function countStudents($students, $sandwiches) {

}
}
```

Dart Solution:

```
class Solution {
int countStudents(List<int> students, List<int> sandwiches) {

}
}
```

Scala Solution:

```
object Solution {
def countStudents(students: Array[Int], sandwiches: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_students(students :: [integer], sandwiches :: [integer]) ::
integer
def count_students(students, sandwiches) do

end
end
```

Erlang Solution:

```
-spec count_students(Students :: [integer()], Sandwiches :: [integer()]) ->
integer().
count_students(Students, Sandwiches) ->
.
```

Racket Solution:

```
(define/contract (count-students students sandwiches)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```