Problem 1983: Widest Pair of Indices With Equal Range Sum

Problem Information

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

Problem Description

You are given two

0-indexed

binary arrays

nums1

and

nums2

. Find the

widest

pair of indices

(i, j)

such that

i <= j

and

nums1[i] + nums1[i+1] + ... + nums1[j] == nums2[i] + nums2[i+1] + ... + nums2[j]

.

The

widest

pair of indices is the pair with the

largest

distance

between

i

and

j

. The

distance

between a pair of indices is defined as

j - i + 1

.

Return

the

distance

of the

widest

pair of indices. If no pair of indices meets the conditions, return

0

.

Example 1:

Input:

nums1 = [1,1,0,1], nums2 = [0,1,1,0]

Output:

3

Explanation:

If i = 1 and j = 3: nums1[1] + nums1[2] + nums1[3] = 1 + 0 + 1 = 2. nums2[1] + nums2[2] + nums2[3] = 1 + 1 + 0 = 2. The distance between i and j is j - i + 1 = 3 - 1 + 1 = 3.

Example 2:

Input:

nums1 = [0,1], nums2 = [1,1]

Output:

1

Explanation:

If i = 1 and j = 1: nums1[1] = 1. nums2[1] = 1. The distance between i and j is j - i + 1 = 1 - 1 + 1 = 1.

Example 3:

Input:

nums1 = [0], nums2 = [1]

Output:

0

Explanation:

There are no pairs of indices that meet the requirements.

Constraints:

n == nums1.length == nums2.length

1 <= n <= 10

5

nums1[i]

is either

0

or

1

.

nums2[i]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int widestPairOfIndices(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java:

```
class Solution {
public int widestPairOfIndices(int[] nums1, int[] nums2) {

}
}
```

Python3:

```
class Solution:
def widestPairOfIndices(self, nums1: List[int], nums2: List[int]) -> int:
```

Python:

```
class Solution(object):
def widestPairOfIndices(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var widestPairOfIndices = function(nums1, nums2) {

};
```

TypeScript:

```
function widestPairOfIndices(nums1: number[], nums2: number[]): number {

};
```

C#:

```
public class Solution {
public int WidestPairOfIndices(int[] nums1, int[] nums2) {

}
}
```

C:

```
int widestPairOfIndices(int* nums1, int nums1Size, int* nums2, int nums2Size)
{

}
```

Go:

```
func widestPairOfIndices(nums1 []int, nums2 []int) int {

}
```

Kotlin:

```
class Solution {
fun widestPairOfIndices(nums1: IntArray, nums2: IntArray): Int {
```

```
}
}
```

Swift:

```
class Solution {
func widestPairOfIndices(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn widest_pair_of_indices(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def widest_pair_of_indices(nums1, nums2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function widestPairOfIndices($nums1, $nums2) {

}
}
```

Dart:

```
class Solution {
int widestPairOfIndices(List<int> nums1, List<int> nums2) {

}
}
```

Scala:

```
object Solution {
def widestPairOfIndices(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec widest_pair_of_indices(nums1 :: [integer], nums2 :: [integer]) ::
integer
def widest_pair_of_indices(nums1, nums2) do

end
end
```

Erlang:

```
-spec widest_pair_of_indices(Nums1 :: [integer()], Nums2 :: [integer()]) ->
integer().
widest_pair_of_indices(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (widest-pair-of-indices nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Widest Pair of Indices With Equal Range Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int widestPairOfIndices(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Widest Pair of Indices With Equal Range Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int widestPairOfIndices(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Widest Pair of Indices With Equal Range Sum
Difficulty: Medium
Tags: array, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def widestPairOfIndices(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def widestPairOfIndices(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Widest Pair of Indices With Equal Range Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var widestPairOfIndices = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Widest Pair of Indices With Equal Range Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function widestPairOfIndices(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Widest Pair of Indices With Equal Range Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int WidestPairOfIndices(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Widest Pair of Indices With Equal Range Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int widestPairOfIndices(int* nums1, int nums1Size, int* nums2, int nums2Size)
{

}
```

Go Solution:

```
// Problem: Widest Pair of Indices With Equal Range Sum
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func widestPairOfIndices(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun widestPairOfIndices(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func widestPairOfIndices(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Widest Pair of Indices With Equal Range Sum
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn widest_pair_of_indices(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def widest_pair_of_indices(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function widestPairOfIndices($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
int widestPairOfIndices(List<int> nums1, List<int> nums2) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def widestPairOfIndices(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec widest_pair_of_indices(nums1 :: [integer], nums2 :: [integer]) ::
integer
def widest_pair_of_indices(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec widest_pair_of_indices(Nums1 :: [integer()], Nums2 :: [integer()]) ->
integer().
widest_pair_of_indices(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (widest-pair-of-indices nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```