Problem 718: Maximum Length of Repeated Subarray

Problem Information

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

Problem Description

Given two integer arrays

nums1

and

nums2

, return

the maximum length of a subarray that appears in

both

arrays

.

Example 1:

Input:

nums1 = [1,2,3,2,1], nums2 = [3,2,1,4,7]

Output:

3

Explanation:

The repeated subarray with maximum length is [3,2,1].

Example 2:

Input:

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

Output:

5

Explanation:

The repeated subarray with maximum length is [0,0,0,0,0].

Constraints:

1 <= nums1.length, nums2.length <= 1000

0 <= nums1[i], nums2[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (find-length nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Length of Repeated Subarray
* Difficulty: Medium
* Tags: array, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Maximum Length of Repeated Subarray
```

```
* Difficulty: Medium
* Tags: array, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Length of Repeated Subarray
Difficulty: Medium
Tags: array, dp, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Length of Repeated Subarray
* Difficulty: Medium
* Tags: array, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Length of Repeated Subarray
* Difficulty: Medium
* Tags: array, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Maximum Length of Repeated Subarray
* Difficulty: Medium
* Tags: array, dp, hash, search
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Maximum Length of Repeated Subarray
* Difficulty: Medium
* Tags: array, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Maximum Length of Repeated Subarray
// Difficulty: Medium
// Tags: array, dp, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Length of Repeated Subarray
// Difficulty: Medium
// Tags: array, dp, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-length nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```