Problem 1713: Minimum Operations to Make a Subsequence

Problem Information

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

Problem Description

You are given an array

target

that consists of

distinct

integers and another integer array

arr

that

can

have duplicates.

In one operation, you can insert any integer at any position in

arr

. For example, if

arr = [1,4,1,2]

, you can add

3

in the middle and make it

[1,4,

3

,1,2]

. Note that you can insert the integer at the very beginning or end of the array.

Return

the

minimum

number of operations needed to make

target

a

subsequence

of

arr

.

A

subsequence

of an array is a new array generated from the original array by deleting some elements (possibly none) without changing the remaining elements' relative order. For example,

[2,7,4]

is a subsequence of

[4,

2

,3,

7

,2,1,

4

]

(the underlined elements), while

[2,4,2]

is not.

Example 1:

Input:

target = [5,1,3],

arr

= [9,4,2,3,4]

Output:

2

Explanation:

You can add 5 and 1 in such a way that makes

arr

= [

5

,9,4,

1

,2,3,4], then target will be a subsequence of

arr

.

Example 2:

Input:

target = [6,4,8,1,3,2],

arr

= [4,7,6,2,3,8,6,1]

Output:

3

Constraints:

1 <= target.length, arr.length <= 10

5

1 <= target[i], arr[i] <= 10

9

target

contains no duplicates.

Code Snippets

C++:

```
class Solution {
public:
int minOperations(vector<int>& target, vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public int minOperations(int[] target, int[] arr) {

}
}
```

Python3:

```
class Solution:
def minOperations(self, target: List[int], arr: List[int]) -> int:
```

Python:

```
class Solution(object):
def minOperations(self, target, arr):
"""
:type target: List[int]
```

```
:type arr: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} target
* @param {number[]} arr
* @return {number}
*/
var minOperations = function(target, arr) {

};
```

TypeScript:

```
function minOperations(target: number[], arr: number[]): number {

};
```

C#:

```
public class Solution {
public int MinOperations(int[] target, int[] arr) {

}
}
```

C:

```
int minOperations(int* target, int targetSize, int* arr, int arrSize) {

}
```

Go:

```
func minOperations(target []int, arr []int) int {

}
```

Kotlin:

```
class Solution {
fun minOperations(target: IntArray, arr: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minOperations(_ target: [Int], _ arr: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_operations(target: Vec<i32>, arr: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} target
# @param {Integer[]} arr
# @return {Integer}
def min_operations(target, arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $target
* @param Integer[] $arr
* @return Integer
*/
function minOperations($target, $arr) {

}
```

```
}
```

Dart:

```
class Solution {
int minOperations(List<int> target, List<int> arr) {

}
}
```

Scala:

```
object Solution {
def minOperations(target: Array[Int], arr: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(target :: [integer], arr :: [integer]) :: integer
def min_operations(target, arr) do

end
end
```

Erlang:

```
-spec min_operations(Target :: [integer()], Arr :: [integer()]) -> integer().
min_operations(Target, Arr) ->
.
```

Racket:

```
(define/contract (min-operations target arr)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make a Subsequence
* Difficulty: Hard
* Tags: array, greedy, hash, search
*
* 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 minOperations(vector<int>& target, vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make a Subsequence
* Difficulty: Hard
* Tags: array, greedy, hash, search
*
* 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 minOperations(int[] target, int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make a Subsequence
Difficulty: Hard
Tags: array, greedy, hash, search
```

```
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 minOperations(self, target: List[int], arr: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minOperations(self, target, arr):
"""
:type target: List[int]
:type arr: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make a Subsequence
* Difficulty: Hard
* Tags: array, greedy, hash, search
*
* 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[]} target
* @param {number[]} arr
* @return {number}
*/
var minOperations = function(target, arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make a Subsequence
* Difficulty: Hard
* Tags: array, greedy, hash, search
*
* 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 minOperations(target: number[], arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make a Subsequence
* Difficulty: Hard
* Tags: array, greedy, hash, search
*
* 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 MinOperations(int[] target, int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make a Subsequence
* Difficulty: Hard
* Tags: array, greedy, hash, search
*
* 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 minOperations(int* target, int targetSize, int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make a Subsequence
// Difficulty: Hard
// Tags: array, greedy, hash, search
//
// 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 minOperations(target []int, arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(target: IntArray, arr: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minOperations(_ target: [Int], _ arr: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make a Subsequence
// Difficulty: Hard
// Tags: array, greedy, hash, search
//
// 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 min_operations(target: Vec<i32>, arr: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} target
# @param {Integer[]} arr
# @return {Integer}
def min_operations(target, arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $target
* @param Integer[] $arr
* @return Integer
*/
function minOperations($target, $arr) {

}
}
```

Dart Solution:

```
class Solution {
int minOperations(List<int> target, List<int> arr) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def minOperations(target: Array[Int], arr: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(target :: [integer], arr :: [integer]) :: integer
def min_operations(target, arr) do

end
end
```

Erlang Solution:

```
-spec min_operations(Target :: [integer()], Arr :: [integer()]) -> integer().
min_operations(Target, Arr) ->
.
```

Racket Solution:

```
(define/contract (min-operations target arr)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```