Problem 1460: Make Two Arrays Equal by Reversing Subarrays

Problem Information

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

Problem Description

You are given two integer arrays of equal length

target

and

arr

. In one step, you can select any

non-empty subarray

of

arr

and reverse it. You are allowed to make any number of steps.

Return

true

if you can make

arr

equal to

target

or

false

otherwise

.

Example 1:

Input:

target = [1,2,3,4], arr = [2,4,1,3]

Output:

true

Explanation:

You can follow the next steps to convert arr to target: 1- Reverse subarray [2,4,1], arr becomes [1,4,2,3] 2- Reverse subarray [4,2], arr becomes [1,2,4,3] 3- Reverse subarray [4,3], arr becomes [1,2,3,4] There are multiple ways to convert arr to target, this is not the only way to do so.

Example 2:

Input:

target = [7], arr = [7]

Output:

true

Explanation:

arr is equal to target without any reverses.

Example 3:

Input:

target = [3,7,9], arr = [3,7,11]

Output:

false

Explanation:

arr does not have value 9 and it can never be converted to target.

Constraints:

target.length == arr.length

1 <= target.length <= 1000

1 <= target[i] <= 1000

1 <= arr[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
bool canBeEqual(int* target, int targetSize, int* arr, int arrSize) {

}
```

Go:

```
func canBeEqual(target []int, arr []int) bool {

}
```

Kotlin:

```
class Solution {
fun canBeEqual(target: IntArray, arr: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (can-be-equal target arr)
(-> (listof exact-integer?) (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make Two Arrays Equal by Reversing Subarrays
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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:
bool canBeEqual(vector<int>& target, vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Make Two Arrays Equal by Reversing Subarrays
```

```
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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 boolean canBeEqual(int[] target, int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Make Two Arrays Equal by Reversing Subarrays
Difficulty: Easy
Tags: array, hash, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Make Two Arrays Equal by Reversing Subarrays
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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 {boolean}
*/
var canBeEqual = function(target, arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Make Two Arrays Equal by Reversing Subarrays
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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 canBeEqual(target: number[], arr: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Make Two Arrays Equal by Reversing Subarrays
* Difficulty: Easy
* Tags: array, hash, sort
```

```
*
* 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 bool CanBeEqual(int[] target, int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Make Two Arrays Equal by Reversing Subarrays
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool canBeEqual(int* target, int targetSize, int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Make Two Arrays Equal by Reversing Subarrays
// Difficulty: Easy
// Tags: array, hash, sort
//
// 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 canBeEqual(target []int, arr []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canBeEqual(target: IntArray, arr: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Make Two Arrays Equal by Reversing Subarrays
// Difficulty: Easy
// Tags: array, hash, sort
//
// 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 can_be_equal(target: Vec<i32>, arr: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (can-be-equal target arr)
(-> (listof exact-integer?) (listof exact-integer?) boolean?)
)
```