Problem 1566: Detect Pattern of Length M Repeated K or More Times

Problem Information

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

Problem Description

Given an array of positive integers

arr

, find a pattern of length

m

that is repeated

k

or more times.

A

pattern

is a subarray (consecutive sub-sequence) that consists of one or more values, repeated multiple times

consecutively

without overlapping. A pattern is defined by its length and the number of repetitions.

Return

true

if there exists a pattern of length

m

that is repeated

k

or more times, otherwise return

false

.

Example 1:

Input:

arr = [1,2,4,4,4,4], m = 1, k = 3

Output:

true

Explanation:

The pattern

(4)

of length 1 is repeated 4 consecutive times. Notice that pattern can be repeated k or more times but not less.

Example 2:

Input:

arr = [1,2,1,2,1,1,1,3], m = 2, k = 2

Output:

true

Explanation:

The pattern

(1,2)

of length 2 is repeated 2 consecutive times. Another valid pattern

(2,1) is

also repeated 2 times.

Example 3:

Input:

arr = [1,2,1,2,1,3], m = 2, k = 3

Output:

false

Explanation:

The pattern (1,2) is of length 2 but is repeated only 2 times. There is no pattern of length 2 that is repeated 3 or more times.

Constraints:

2 <= arr.length <= 100

1 <= arr[i] <= 100

1 <= m <= 100

2 <= k <= 100

Code Snippets

C++:

```
class Solution {
public:
bool containsPattern(vector<int>& arr, int m, int k) {

}
};
```

Java:

```
class Solution {
public boolean containsPattern(int[] arr, int m, int k) {

}
}
```

Python3:

```
class Solution:
def containsPattern(self, arr: List[int], m: int, k: int) -> bool:
```

Python:

```
class Solution(object):
def containsPattern(self, arr, m, k):
"""
:type arr: List[int]
:type m: int
:type k: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} arr
* @param {number} m
* @param {number} k
* @return {boolean}
*/
var containsPattern = function(arr, m, k) {

};
```

TypeScript:

```
function containsPattern(arr: number[], m: number, k: number): boolean {

};
```

C#:

```
public class Solution {
public bool ContainsPattern(int[] arr, int m, int k) {

}
}
```

C:

```
bool containsPattern(int* arr, int arrSize, int m, int k) {

}
```

Go:

```
func containsPattern(arr []int, m int, k int) bool {

}
```

Kotlin:

```
class Solution {
fun containsPattern(arr: IntArray, m: Int, k: Int): Boolean {
```

```
}
}
```

Swift:

```
class Solution {
func containsPattern(_ arr: [Int], _ m: Int, _ k: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn contains_pattern(arr: Vec<i32>, m: i32, k: i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} m
# @param {Integer} k
# @return {Boolean}
def contains_pattern(arr, m, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $m
* @param Integer $k
* @return Boolean
*/
function containsPattern($arr, $m, $k) {

}
```

```
}
```

Dart:

```
class Solution {
bool containsPattern(List<int> arr, int m, int k) {

}
}
```

Scala:

```
object Solution {
def containsPattern(arr: Array[Int], m: Int, k: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec contains_pattern(arr :: [integer], m :: integer, k :: integer) ::
boolean
def contains_pattern(arr, m, k) do

end
end
```

Erlang:

```
-spec contains_pattern(Arr :: [integer()], M :: integer(), K :: integer()) ->
boolean().
contains_pattern(Arr, M, K) ->
.
```

Racket:

```
(define/contract (contains-pattern arr m k)
(-> (listof exact-integer?) exact-integer? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Detect Pattern of Length M Repeated K or More Times
* Difficulty: Easy
* Tags: array
*
* 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:
bool containsPattern(vector<int>& arr, int m, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Detect Pattern of Length M Repeated K or More Times
* Difficulty: Easy
* Tags: array
*
* 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 boolean containsPattern(int[] arr, int m, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Detect Pattern of Length M Repeated K or More Times
```

```
Difficulty: Easy
Tags: array

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

Python Solution:

```
class Solution(object):
def containsPattern(self, arr, m, k):
"""
:type arr: List[int]
:type m: int
:type k: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Detect Pattern of Length M Repeated K or More Times
* Difficulty: Easy
* Tags: array
*
* 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[]} arr
* @param {number} m
* @param {number} k
* @return {boolean}
```

```
*/
var containsPattern = function(arr, m, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Detect Pattern of Length M Repeated K or More Times
* Difficulty: Easy
* Tags: array
*
* 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 containsPattern(arr: number[], m: number, k: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Detect Pattern of Length M Repeated K or More Times
* Difficulty: Easy
* Tags: array
*
* 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 bool ContainsPattern(int[] arr, int m, int k) {

}
}
```

C Solution:

```c
/*
* Problem: Detect Pattern of Length M Repeated K or More Times
* Difficulty: Easy
* Tags: array
*
* 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
*/

bool containsPattern(int* arr, int arrSize, int m, int k) {

}
```

Go Solution:

```go
// Problem: Detect Pattern of Length M Repeated K or More Times
// Difficulty: Easy
// Tags: array
//
// 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 containsPattern(arr []int, m int, k int) bool {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun containsPattern(arr: IntArray, m: Int, k: Int): Boolean {

}
}
```

Swift Solution:

```swift
class Solution {
func containsPattern(_ arr: [Int], _ m: Int, _ k: Int) -> Bool {

}
```

```
}
```

Rust Solution:

```
// Problem: Detect Pattern of Length M Repeated K or More Times
// Difficulty: Easy
// Tags: array
//
// 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 contains_pattern(arr: Vec<i32>, m: i32, k: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} m
# @param {Integer} k
# @return {Boolean}
def contains_pattern(arr, m, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $m
* @param Integer $k
* @return Boolean
*/
function containsPattern($arr, $m, $k) {

}
```

```
}
```

Dart Solution:

```
class Solution {
bool containsPattern(List<int> arr, int m, int k) {

}
}
```

Scala Solution:

```
object Solution {
def containsPattern(arr: Array[Int], m: Int, k: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec contains_pattern(arr :: [integer], m :: integer, k :: integer) ::
boolean
def contains_pattern(arr, m, k) do

end
end
```

Erlang Solution:

```
-spec contains_pattern(Arr :: [integer()], M :: integer(), K :: integer()) ->
boolean().
contains_pattern(Arr, M, K) ->
.
```

Racket Solution:

```
(define/contract (contains-pattern arr m k)
(-> (listof exact-integer?) exact-integer? exact-integer? boolean?)
)
```