Problem 3208: Alternating Groups II

Problem Information

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

Problem Description

There is a circle of red and blue tiles. You are given an array of integers

colors

and an integer

k

. The color of tile

i

is represented by

colors[i]

:

colors[i] == 0

means that tile

i

is

red

.

colors[i] == 1

means that tile

i

is

blue

.

An

alternating

group is every

k

contiguous tiles in the circle with

alternating

colors (each tile in the group except the first and last one has a different color from its

left

and

right

tiles).

Return the number of

alternating

groups.

Note

that since

colors

represents a

circle

, the

first

and the

last

tiles are considered to be next to each other.

Example 1:

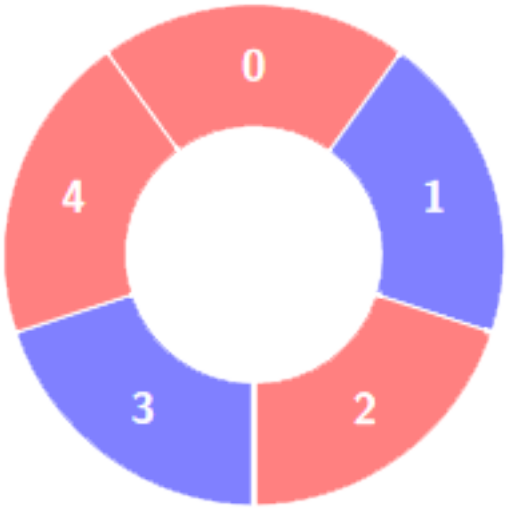
Input:

colors = [0,1,0,1,0], k = 3

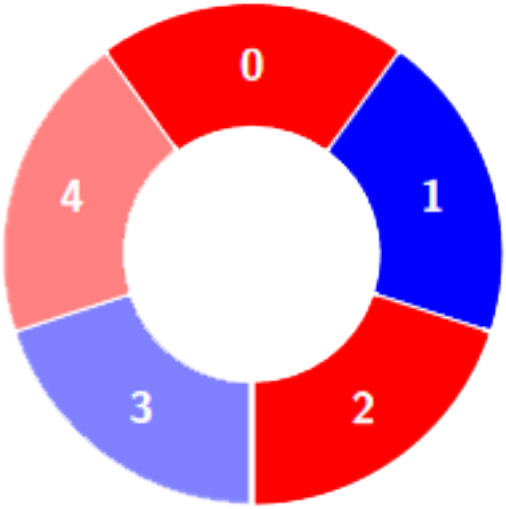
Output:

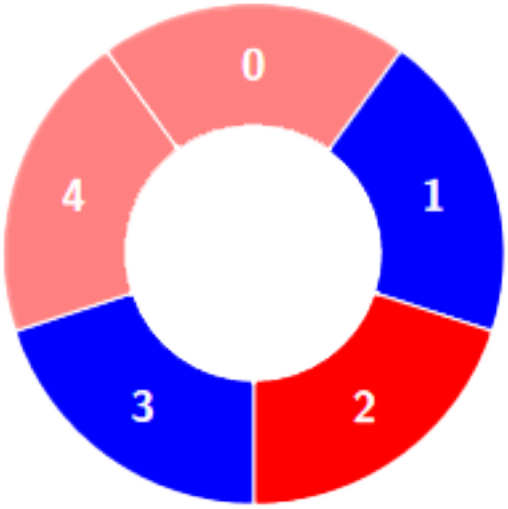
3

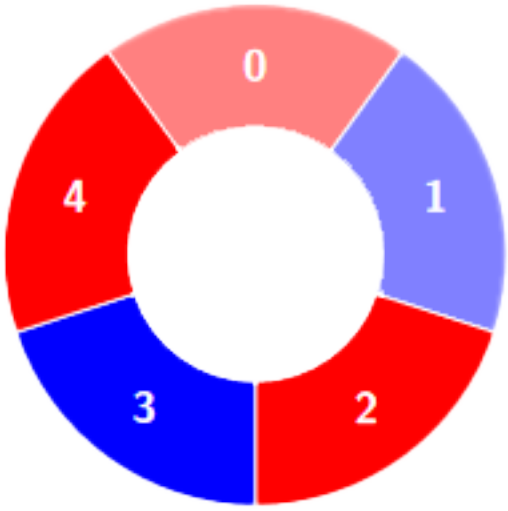
Explanation:



Alternating groups:







Example 2:

Input:

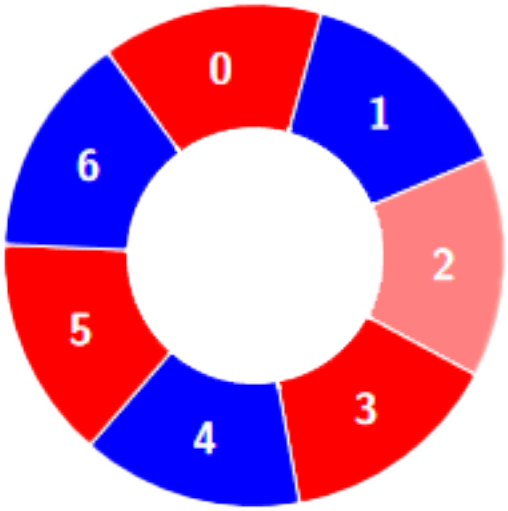
colors = [0,1,0,0,1,0,1], k = 6

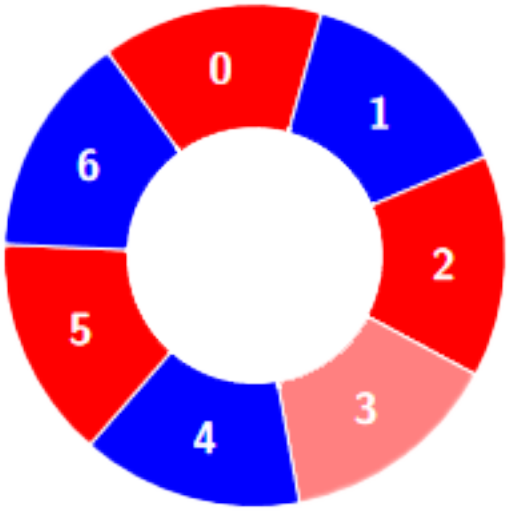
Output:

2

Explanation:

Alternating groups:





Example 3:

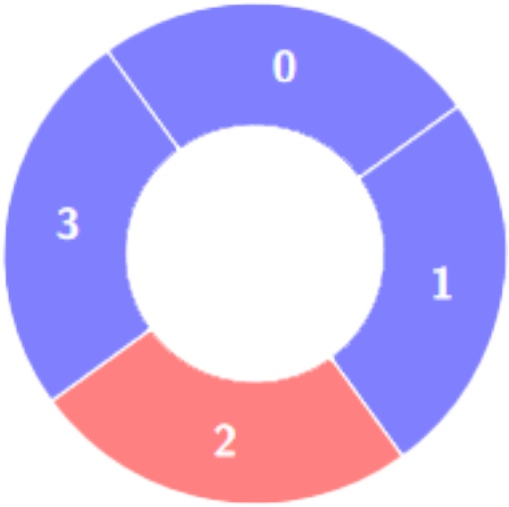
Input:

colors = [1,1,0,1], k = 4

Output:

0

Explanation:



Constraints:

3 <= colors.length <= 10

5

0 <= colors[i] <= 1

3 <= k <= colors.length

Code Snippets

C++:

```
class Solution {
public:
int numberOfAlternatingGroups(vector<int>& colors, int k) {

}
};
```

Java:

```
class Solution {
public int numberOfAlternatingGroups(int[] colors, int k) {

}
```

```
}
```

Python3:

```
class Solution:
def numberOfAlternatingGroups(self, colors: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def numberOfAlternatingGroups(self, colors, k):
"""
:type colors: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} colors
* @param {number} k
* @return {number}
*/
var numberOfAlternatingGroups = function(colors, k) {

};
```

TypeScript:

```
function numberOfAlternatingGroups(colors: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int NumberOfAlternatingGroups(int[] colors, int k) {

}
}
```

C:

```
int numberOfAlternatingGroups(int* colors, int colorsSize, int k) {

}
```

Go:

```
func numberOfAlternatingGroups(colors []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfAlternatingGroups(colors: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func numberOfAlternatingGroups(_ colors: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_alternating_groups(colors: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} colors
# @param {Integer} k
# @return {Integer}
def number_of_alternating_groups(colors, k)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $colors
* @param Integer $k
* @return Integer
*/
function numberOfAlternatingGroups($colors, $k) {

}
}
```

Dart:

```
class Solution {
int numberOfAlternatingGroups(List<int> colors, int k) {

}
}
```

Scala:

```
object Solution {
def numberOfAlternatingGroups(colors: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_alternating_groups(colors :: [integer], k :: integer) ::
integer
def number_of_alternating_groups(colors, k) do

end
end
```

Erlang:

```
-spec number_of_alternating_groups(Colors :: [integer()], K :: integer()) ->
integer().
number_of_alternating_groups(Colors, K) ->
.
```

Racket:

```
(define/contract (number-of-alternating-groups colors k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Alternating Groups II
* Difficulty: Medium
* 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:
int numberOfAlternatingGroups(vector<int>& colors, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Alternating Groups II
* Difficulty: Medium
* 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 int numberOfAlternatingGroups(int[] colors, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Alternating Groups II
Difficulty: Medium
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 numberOfAlternatingGroups(self, colors: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numberOfAlternatingGroups(self, colors, k):
"""
:type colors: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Alternating Groups II
* Difficulty: Medium
* 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[]} colors
* @param {number} k
* @return {number}
*/
var numberOfAlternatingGroups = function(colors, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Alternating Groups II
* Difficulty: Medium
* 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 numberOfAlternatingGroups(colors: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Alternating Groups II
* Difficulty: Medium
* 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 int NumberOfAlternatingGroups(int[] colors, int k) {

}
}
```

C Solution:

```
/*
* Problem: Alternating Groups II
* Difficulty: Medium
* 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
*/

int numberOfAlternatingGroups(int* colors, int colorsSize, int k) {

}
```

Go Solution:

```
// Problem: Alternating Groups II
// Difficulty: Medium
// 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 numberOfAlternatingGroups(colors []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfAlternatingGroups(colors: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfAlternatingGroups(_ colors: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Alternating Groups II
// Difficulty: Medium
// 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 number_of_alternating_groups(colors: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} colors
# @param {Integer} k
# @return {Integer}
def number_of_alternating_groups(colors, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $colors
* @param Integer $k
* @return Integer
*/
function numberOfAlternatingGroups($colors, $k) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfAlternatingGroups(List<int> colors, int k) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfAlternatingGroups(colors: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_alternating_groups(colors :: [integer], k :: integer) ::
integer
def number_of_alternating_groups(colors, k) do

end
end
```

Erlang Solution:

```
-spec number_of_alternating_groups(Colors :: [integer()], K :: integer()) ->
integer().
```

```
number_of_alternating_groups(Colors, K) ->
.
```

Racket Solution:

```
(define/contract (number-of-alternating-groups colors k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```