Problem 2107: Number of Unique Flavors After Sharing K Candies

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

candies

, where

candies[i]

represents the flavor of the

i

th

candy. Your mom wants you to share these candies with your little sister by giving her

k

consecutive

candies, but you want to keep as many flavors of candies as possible.

Return

the

maximum

number of

unique

flavors of candy you can keep after sharing

with your sister.

Example 1:

Input:

candies = [1,

2,2,3

,4,3], k = 3

Output:

3

Explanation:

Give the candies in the range [1, 3] (inclusive) with flavors [2,2,3]. You can eat candies with flavors [1,4,3]. There are 3 unique flavors, so return 3.

Example 2:

Input:

candies = [2,2,2,

2,3

,3], k = 2

Output:

2

Explanation:

Give the candies in the range [3, 4] (inclusive) with flavors [2,3]. You can eat candies with flavors [2,2,2,3]. There are 2 unique flavors, so return 2. Note that you can also share the candies with flavors [2,2] and eat the candies with flavors [2,2,3,3].

Example 3:

Input:

candies = [2,4,5], k = 0

Output:

3

Explanation:

You do not have to give any candies. You can eat the candies with flavors [2,4,5]. There are 3 unique flavors, so return 3.

Constraints:

0 <= candies.length <= 10

5

1 <= candies[i] <= 10

5

0 <= k <= candies.length

Code Snippets

C++:

```
class Solution {
public:
int shareCandies(vector<int>& candies, int k) {

}
};
```

Java:

```
class Solution {
public int shareCandies(int[] candies, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function shareCandies(candies: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int ShareCandies(int[] candies, int k) {

}
}
```

C:

```
int shareCandies(int* candies, int candiesSize, int k) {

}
```

Go:

```
func shareCandies(candies []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun shareCandies(candies: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} candies
# @param {Integer} k
# @return {Integer}
def share_candies(candies, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $candies
* @param Integer $k
* @return Integer
*/
function shareCandies($candies, $k) {

}
}
```

Dart:

```
class Solution {
int shareCandies(List<int> candies, int k) {

}
}
```

Scala:

```
object Solution {
def shareCandies(candies: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec share_candies(candies :: [integer], k :: integer) :: integer
def share_candies(candies, k) do

end
end
```

Erlang:

```
-spec share_candies(Candies :: [integer()], K :: integer()) -> integer().
share_candies(Candies, K) ->
.
```

Racket:

```
(define/contract (share-candies candies k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Unique Flavors After Sharing K Candies
* Difficulty: Medium
* Tags: array, hash
*
```

```
* 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 shareCandies(vector<int>& candies, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Unique Flavors After Sharing K Candies
* Difficulty: Medium
* Tags: array, hash
*
* 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 shareCandies(int[] candies, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Unique Flavors After Sharing K Candies
Difficulty: Medium
Tags: array, hash

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 shareCandies(self, candies: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Unique Flavors After Sharing K Candies
* Difficulty: Medium
* Tags: array, hash
*
* 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[]} candies
* @param {number} k
* @return {number}
*/
var shareCandies = function(candies, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Unique Flavors After Sharing K Candies
* Difficulty: Medium
* Tags: array, hash
*
* 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 shareCandies(candies: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Unique Flavors After Sharing K Candies
* Difficulty: Medium
* Tags: array, hash
*
* 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 ShareCandies(int[] candies, int k) {

}
}
```

C Solution:

```
/*
* Problem: Number of Unique Flavors After Sharing K Candies
* Difficulty: Medium
* Tags: array, hash
*
* 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 shareCandies(int* candies, int candiesSize, int k) {

}
```

Go Solution:

```
// Problem: Number of Unique Flavors After Sharing K Candies
// Difficulty: Medium
// Tags: array, hash
//
// 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 shareCandies(candies []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun shareCandies(candies: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Unique Flavors After Sharing K Candies
// Difficulty: Medium
// Tags: array, hash
//
```

```
// 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 share_candies(candies: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} candies
# @param {Integer} k
# @return {Integer}
def share_candies(candies, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $candies
* @param Integer $k
* @return Integer
*/
function shareCandies($candies, $k) {

}
}
```

Dart Solution:

```
class Solution {
int shareCandies(List<int> candies, int k) {

}
}
```

Scala Solution:

```
object Solution {
def shareCandies(candies: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec share_candies(candies :: [integer], k :: integer) :: integer
def share_candies(candies, k) do

end
end
```

Erlang Solution:

```
-spec share_candies(Candies :: [integer()], K :: integer()) -> integer().
share_candies(Candies, K) ->
.
```

Racket Solution:

```
(define/contract (share-candies candies k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```