Problem 2461: Maximum Sum of Distinct Subarrays With Length K

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

. Find the maximum subarray sum of all the subarrays of

nums

that meet the following conditions:

The length of the subarray is

k

, and

All the elements of the subarray are

distinct

.

Return

the maximum subarray sum of all the subarrays that meet the conditions

.

If no subarray meets the conditions, return

0

.

A

subarray

is a contiguous non-empty sequence of elements within an array.

Example 1:

Input:

nums = [1,5,4,2,9,9,9], k = 3

Output:

15

Explanation:

The subarrays of nums with length 3 are: - [1,5,4] which meets the requirements and has a sum of 10. - [5,4,2] which meets the requirements and has a sum of 11. - [4,2,9] which meets the requirements and has a sum of 15. - [2,9,9] which does not meet the requirements because the element 9 is repeated. - [9,9,9] which does not meet the requirements because the element 9 is repeated. We return 15 because it is the maximum subarray sum of all the subarrays that meet the conditions

Example 2:

Input:

nums = [4,4,4], k = 3

Output:

0

Explanation:

The subarrays of nums with length 3 are: - [4,4,4] which does not meet the requirements because the element 4 is repeated. We return 0 because no subarrays meet the conditions.

Constraints:

1 <= k <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long maximumSubarraySum(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public long maximumSubarraySum(int[] nums, int k) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumSubarraySum(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long MaximumSubarraySum(int[] nums, int k) {

}
}
```

C:

```
long long maximumSubarraySum(int* nums, int numsSize, int k) {

}
```

Go:

```
func maximumSubarraySum(nums []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximumSubarraySum(nums: IntArray, k: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn maximum_subarray_sum(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def maximum_subarray_sum(nums, k)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maximumSubarraySum($nums, $k) {

}
}
```

Dart:

```
class Solution {
int maximumSubarraySum(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def maximumSubarraySum(nums: Array[Int], k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_subarray_sum(nums :: [integer], k :: integer) :: integer
def maximum_subarray_sum(nums, k) do

end
end
```

Erlang:

```
-spec maximum_subarray_sum(Nums :: [integer()], K :: integer()) -> integer().
maximum_subarray_sum(Nums, K) ->
.
```

Racket:

```
(define/contract (maximum-subarray-sum nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum of Distinct Subarrays With Length K
* 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:
long long maximumSubarraySum(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum of Distinct Subarrays With Length K
* 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 long maximumSubarraySum(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum of Distinct Subarrays With Length K
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 maximumSubarraySum(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Sum of Distinct Subarrays With Length K
* 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[]} nums
* @param {number} k
* @return {number}
*/
var maximumSubarraySum = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum of Distinct Subarrays With Length K
* 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 maximumSubarraySum(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Sum of Distinct Subarrays With Length K
* 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 long MaximumSubarraySum(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Sum of Distinct Subarrays With Length K
* 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
*/

long long maximumSubarraySum(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Sum of Distinct Subarrays With Length K
// 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 maximumSubarraySum(nums []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumSubarraySum(nums: IntArray, k: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Sum of Distinct Subarrays With Length K
// 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 maximum_subarray_sum(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def maximum_subarray_sum(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maximumSubarraySum($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maximumSubarraySum(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maximumSubarraySum(nums: Array[Int], k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_subarray_sum(nums :: [integer], k :: integer) :: integer
def maximum_subarray_sum(nums, k) do

end
end
```

Erlang Solution:

```
-spec maximum_subarray_sum(Nums :: [integer()], K :: integer()) -> integer().
maximum_subarray_sum(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (maximum-subarray-sum nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```