Problem 3013: Divide an Array Into Subarrays With Minimum Cost II

Problem Information

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

Problem Description

You are given a

0-indexed

array of integers

nums

of length

n

, and two

positive

integers

k

and

dist

.

The

cost

of an array is the value of its

first

element. For example, the cost of

[1,2,3]

is

1

while the cost of

[3,4,1]

is

3

.

You need to divide

nums

into

k

disjoint contiguous

subarrays

, such that the difference between the starting index of the

second

subarray and the starting index of the

kth

subarray should be

less than or equal to

dist

. In other words, if you divide

nums

into the subarrays

nums[0..(i

1

- 1)], nums[i

1

..(i

2

- 1)], ..., nums[i

k-1

..(n - 1)]

, then

i

k-1

- i

1

<= dist

.

Return

the

minimum

possible sum of the cost of these

subarrays

.

Example 1:

Input:

nums = [1,3,2,6,4,2], k = 3, dist = 3

Output:

5

Explanation:

The best possible way to divide nums into 3 subarrays is: [1,3], [2,6,4], and [2]. This choice is valid because i

k-1

- i

1

is 5 - 2 = 3 which is equal to dist. The total cost is nums[0] + nums[2] + nums[5] which is 1 + 2 + 2 = 5. It can be shown that there is no possible way to divide nums into 3 subarrays at a cost lower than 5.

Example 2:

Input:

nums = [10,1,2,2,2,1], k = 4, dist = 3

Output:

15

Explanation:

The best possible way to divide nums into 4 subarrays is: [10], [1], [2], and [2,2,1]. This choice is valid because i

k-1

- i

1

is 3 - 1 = 2 which is less than dist. The total cost is nums[0] + nums[1] + nums[2] + nums[3] which is 10 + 1 + 2 + 2 = 15. The division [10], [1], [2,2,2], and [1] is not valid, because the difference between i

k-1

and i

1

is 5 - 1 = 4, which is greater than dist. It can be shown that there is no possible way to divide nums into 4 subarrays at a cost lower than 15.

Example 3:

Input:

nums = [10,8,18,9], k = 3, dist = 1

Output:

36

Explanation:

The best possible way to divide nums into 4 subarrays is: [10], [8], and [18,9]. This choice is valid because i

k-1

- i

1

is 2 - 1 = 1 which is equal to dist.The total cost is nums[0] + nums[1] + nums[2] which is 10 + 8 + 18 = 36. The division [10], [8,18], and [9] is not valid, because the difference between i

k-1

and i

1

is 3 - 1 = 2, which is greater than dist. It can be shown that there is no possible way to divide nums into 3 subarrays at a cost lower than 36.

Constraints:

3 <= n <= 10

5

1 <= nums[i] <= 10

9

3 <= k <= n

k - 2 <= dist <= n - 2

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def minimumCost(self, nums, k, dist):
```

```
"""
:type nums: List[int]
:type k: int
:type dist: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
long long minimumCost(int* nums, int numsSize, int k, int dist) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-cost nums k dist)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Divide an Array Into Subarrays With Minimum Cost II
* Difficulty: Hard
* Tags: array, hash, queue, heap
*
* 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 minimumCost(vector<int>& nums, int k, int dist) {

}
};
```

Java Solution:

```
/**
* Problem: Divide an Array Into Subarrays With Minimum Cost II
* Difficulty: Hard
* Tags: array, hash, queue, heap
*
* 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 minimumCost(int[] nums, int k, int dist) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Divide an Array Into Subarrays With Minimum Cost II
Difficulty: Hard
Tags: array, hash, queue, heap

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 minimumCost(self, nums: List[int], k: int, dist: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Divide an Array Into Subarrays With Minimum Cost II
* Difficulty: Hard
* Tags: array, hash, queue, heap
*
* 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
* @param {number} dist
* @return {number}
*/
var minimumCost = function(nums, k, dist) {

};
```

TypeScript Solution:

```
/**
* Problem: Divide an Array Into Subarrays With Minimum Cost II
* Difficulty: Hard
* Tags: array, hash, queue, heap
*
* 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 minimumCost(nums: number[], k: number, dist: number): number {

};
```

C# Solution:

```
/*
* Problem: Divide an Array Into Subarrays With Minimum Cost II
* Difficulty: Hard
* Tags: array, hash, queue, heap
*
* 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 MinimumCost(int[] nums, int k, int dist) {

}
}
```

C Solution:

```
/*
* Problem: Divide an Array Into Subarrays With Minimum Cost II
* Difficulty: Hard
* Tags: array, hash, queue, heap
*
* 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 minimumCost(int* nums, int numsSize, int k, int dist) {

}
```

Go Solution:

```
// Problem: Divide an Array Into Subarrays With Minimum Cost II
// Difficulty: Hard
// Tags: array, hash, queue, heap
//
// 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 minimumCost(nums []int, k int, dist int) int64 {

}
```

Kotlin Solution:

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

```
}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Divide an Array Into Subarrays With Minimum Cost II
// Difficulty: Hard
// Tags: array, hash, queue, heap
//
// 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 minimum_cost(nums: Vec<i32>, k: i32, dist: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
minimum_cost(Nums, K, Dist) ->
.
```

Racket Solution:

```
(define/contract (minimum-cost nums k dist)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```