Problem 3509: Maximum Product of Subsequences With an Alternating Sum Equal to K

Problem Information

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

Problem Description

You are given an integer array

nums

and two integers,

k

and

limit

. Your task is to find a non-empty

subsequence

of

nums

that:

Has an

alternating sum

equal to

k

.

Maximizes

the product of all its numbers

without the product exceeding

limit

.

Return the

product

of the numbers in such a subsequence. If no subsequence satisfies the requirements, return -1.

The

alternating sum

of a

0-indexed

array is defined as the

sum

of the elements at

even

indices

minus

the

sum

of the elements at

odd

indices.

Example 1:

Input:

nums = [1,2,3], k = 2, limit = 10

Output:

6

Explanation:

The subsequences with an alternating sum of 2 are:

[1, 2, 3]

Alternating Sum:

1 - 2 + 3 = 2

Product:

1 * 2 * 3 = 6

[2]

Alternating Sum: 2

Product: 2

The maximum product within the limit is 6.

Example 2:

Input:

nums = [0,2,3], k = -5, limit = 12

Output:

-1

Explanation:

A subsequence with an alternating sum of exactly -5 does not exist.

Example 3:

Input:

nums = [2,2,3,3], k = 0, limit = 9

Output:

9

Explanation:

The subsequences with an alternating sum of 0 are:

[2, 2]

Alternating Sum:

2 - 2 = 0

Product:

2 * 2 = 4

[3, 3]

Alternating Sum:

3 - 3 = 0

Product:

3 * 3 = 9

[2, 2, 3, 3]

Alternating Sum:

2 - 2 + 3 - 3 = 0

Product:

2 * 2 * 3 * 3 = 36

The subsequence

[2, 2, 3, 3]

has the greatest product with an alternating sum equal to

k

, but

36 > 9

. The next greatest product is 9, which is within the limit.

Constraints:

1 <= nums.length <= 150

0 <= nums[i] <= 12

-10

5

<= k <= 10

5

1 <= limit <= 5000

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int maxProduct(int[] nums, int k, int limit) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MaxProduct(int[] nums, int k, int limit) {

}
}
```

C:

```
int maxProduct(int* nums, int numsSize, int k, int limit) {

}
```

Go:

```
func maxProduct(nums []int, k int, limit int) int {

}
```

Kotlin:

```
class Solution {
fun maxProduct(nums: IntArray, k: Int, limit: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-product nums k limit)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Product of Subsequences With an Alternating Sum Equal to K
* Difficulty: Hard
* Tags: array, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxProduct(vector<int>& nums, int k, int limit) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Product of Subsequences With an Alternating Sum Equal to K
* Difficulty: Hard
* Tags: array, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxProduct(int[] nums, int k, int limit) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Product of Subsequences With an Alternating Sum Equal to K
Difficulty: Hard
Tags: array, dp, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxProduct(self, nums: List[int], k: int, limit: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Product of Subsequences With an Alternating Sum Equal to K
* Difficulty: Hard
* Tags: array, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Product of Subsequences With an Alternating Sum Equal to K
* Difficulty: Hard
* Tags: array, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxProduct(nums: number[], k: number, limit: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Product of Subsequences With an Alternating Sum Equal to K
* Difficulty: Hard
* Tags: array, dp, hash
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxProduct(int[] nums, int k, int limit) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Product of Subsequences With an Alternating Sum Equal to K
* Difficulty: Hard
* Tags: array, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxProduct(int* nums, int numsSize, int k, int limit) {

}
```

Go Solution:

```
// Problem: Maximum Product of Subsequences With an Alternating Sum Equal to K
// Difficulty: Hard
// Tags: array, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxProduct(nums []int, k int, limit int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxProduct(nums: IntArray, k: Int, limit: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Product of Subsequences With an Alternating Sum Equal to K
// Difficulty: Hard
// Tags: array, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_product(nums: Vec<i32>, k: i32, limit: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-product nums k limit)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```