Problem 3566: Partition Array into Two Equal Product Subsets

Problem Information

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

Problem Description

You are given an integer array

nums

containing

distinct

positive integers and an integer

target

.

Determine if you can partition

nums

into two

non-empty

disjoint

subsets

, with each element belonging to

exactly one

subset, such that the product of the elements in each subset is equal to

target

.

Return

true

if such a partition exists and

false

otherwise.

A

subset

of an array is a selection of elements of the array.

Example 1:

Input:

nums = [3,1,6,8,4], target = 24

Output:

true

Explanation:

The subsets

[3, 8]

and

[1, 6, 4]

each have a product of 24. Hence, the output is true.

Example 2:

Input:

nums = [2,5,3,7], target = 15

Output:

false

Explanation:

There is no way to partition

nums

into two non-empty disjoint subsets such that both subsets have a product of 15. Hence, the output is false.

Constraints:

3 <= nums.length <= 12

1 <= target <= 10

15

1 <= nums[i] <= 100

All elements of

nums

are

distinct

.

Code Snippets

C++:

```
class Solution {
public:
bool checkEqualPartitions(vector<int>& nums, long long target) {

}
};
```

Java:

```
class Solution {
public boolean checkEqualPartitions(int[] nums, long target) {

}
}
```

Python3:

```
class Solution:
def checkEqualPartitions(self, nums: List[int], target: int) -> bool:
```

Python:

```
class Solution(object):
def checkEqualPartitions(self, nums, target):
"""
:type nums: List[int]
```

```
:type target: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} target
* @return {boolean}
*/
var checkEqualPartitions = function(nums, target) {

};
```

TypeScript:

```
function checkEqualPartitions(nums: number[], target: number): boolean {

};
```

C#:

```
public class Solution {
public bool CheckEqualPartitions(int[] nums, long target) {

}
}
```

C:

```
bool checkEqualPartitions(int* nums, int numsSize, long long target) {

}
```

Go:

```
func checkEqualPartitions(nums []int, target int64) bool {

}
```

Kotlin:

```
class Solution {
fun checkEqualPartitions(nums: IntArray, target: Long): Boolean {

}
}
```

Swift:

```
class Solution {
func checkEqualPartitions(_ nums: [Int], _ target: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn check_equal_partitions(nums: Vec<i32>, target: i64) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Boolean}
def check_equal_partitions(nums, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Boolean
*/
function checkEqualPartitions($nums, $target) {

}
```

```
}
```

Dart:

```
class Solution {
bool checkEqualPartitions(List<int> nums, int target) {

}
}
```

Scala:

```
object Solution {
def checkEqualPartitions(nums: Array[Int], target: Long): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec check_equal_partitions(nums :: [integer], target :: integer) :: boolean
def check_equal_partitions(nums, target) do

end
end
```

Erlang:

```
-spec check_equal_partitions(Nums :: [integer()], Target :: integer()) ->
boolean().
check_equal_partitions(Nums, Target) ->
.
```

Racket:

```
(define/contract (check-equal-partitions nums target)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Partition Array into Two Equal Product Subsets
* 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:
bool checkEqualPartitions(vector<int>& nums, long long target) {

}
};
```

Java Solution:

```
/**
* Problem: Partition Array into Two Equal Product Subsets
* 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 boolean checkEqualPartitions(int[] nums, long target) {

}
}
```

Python3 Solution:

```
"""
Problem: Partition Array into Two Equal Product Subsets
```

```
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 checkEqualPartitions(self, nums: List[int], target: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def checkEqualPartitions(self, nums, target):
"""
:type nums: List[int]
:type target: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Partition Array into Two Equal Product Subsets
* 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[]} nums
* @param {number} target
* @return {boolean}
*/
var checkEqualPartitions = function(nums, target) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Partition Array into Two Equal Product Subsets
* 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 checkEqualPartitions(nums: number[], target: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Partition Array into Two Equal Product Subsets
* 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 bool CheckEqualPartitions(int[] nums, long target) {

}
}
```

C Solution:

```
/*
* Problem: Partition Array into Two Equal Product Subsets
```

```
* 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
*/

bool checkEqualPartitions(int* nums, int numsSize, long long target) {

}
```

Go Solution:

```
// Problem: Partition Array into Two Equal Product Subsets
// 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 checkEqualPartitions(nums []int, target int64) bool {

}
```

Kotlin Solution:

```
class Solution {
fun checkEqualPartitions(nums: IntArray, target: Long): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func checkEqualPartitions(_ nums: [Int], _ target: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Partition Array into Two Equal Product Subsets
// 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 check_equal_partitions(nums: Vec<i32>, target: i64) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Boolean}
def check_equal_partitions(nums, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Boolean
*/
function checkEqualPartitions($nums, $target) {

}
}
```

Dart Solution:

```
class Solution {
bool checkEqualPartitions(List<int> nums, int target) {

}
}
```

Scala Solution:

```
object Solution {
def checkEqualPartitions(nums: Array[Int], target: Long): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec check_equal_partitions(nums :: [integer], target :: integer) :: boolean
def check_equal_partitions(nums, target) do

end
end
```

Erlang Solution:

```
-spec check_equal_partitions(Nums :: [integer()], Target :: integer()) ->
boolean().
check_equal_partitions(Nums, Target) ->
.
```

Racket Solution:

```
(define/contract (check-equal-partitions nums target)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```