Problem 416: Partition Equal Subset Sum

Problem Information

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

Problem Description

Given an integer array

nums

, return

true

if you can partition the array into two subsets such that the sum of the elements in both subsets is equal or

false

otherwise

.

Example 1:

Input:

nums = [1,5,11,5]

Output:

true

Explanation:

The array can be partitioned as [1, 5, 5] and [11].

Example 2:

Input:

nums = [1,2,3,5]

Output:

false

Explanation:

The array cannot be partitioned into equal sum subsets.

Constraints:

1 <= nums.length <= 200

1 <= nums[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
bool canPartition(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public boolean canPartition(int[] nums) {
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function canPartition(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool CanPartition(int[] nums) {

}
}
```

C:

```
bool canPartition(int* nums, int numsSize) {

}
```

Go:

```
func canPartition(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun canPartition(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn can_partition(nums: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Boolean}
def can_partition(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean
*/
function canPartition($nums) {

}
}
```

Dart:

```
class Solution {
bool canPartition(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def canPartition(nums: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_partition(nums :: [integer]) :: boolean
def can_partition(nums) do

end
end
```

Erlang:

```
-spec can_partition(Nums :: [integer()]) -> boolean().
can_partition(Nums) ->
.
```

Racket:

```
(define/contract (can-partition nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Partition Equal Subset Sum
* Difficulty: Medium
* Tags: array, dp
*
* 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:
bool canPartition(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Partition Equal Subset Sum
* Difficulty: Medium
* Tags: array, dp
*
* 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 boolean canPartition(int[] nums) {
}
}
```

Python3 Solution:

```
"""
Problem: Partition Equal Subset Sum
Difficulty: Medium
Tags: array, dp

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Partition Equal Subset Sum
* Difficulty: Medium
* Tags: array, dp
*
* 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
* @return {boolean}
*/
var canPartition = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Partition Equal Subset Sum
* Difficulty: Medium
* Tags: array, dp
*
* 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 canPartition(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Partition Equal Subset Sum
* Difficulty: Medium
* Tags: array, dp
*
* 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 bool CanPartition(int[] nums) {

}

}
```

C Solution:

```
/*
* Problem: Partition Equal Subset Sum
* Difficulty: Medium
* Tags: array, dp
*
* 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
*/

bool canPartition(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Partition Equal Subset Sum
// Difficulty: Medium
// Tags: array, dp
//
// 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 canPartition(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canPartition(nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Partition Equal Subset Sum
// Difficulty: Medium
// Tags: array, dp
//
// 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 can_partition(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Boolean}
def can_partition(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean
*/
function canPartition($nums) {

}
}
```

Dart Solution:

```
class Solution {
bool canPartition(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def canPartition(nums: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_partition(nums :: [integer]) :: boolean
def can_partition(nums) do

end
end
```

Erlang Solution:

```
-spec can_partition(Nums :: [integer()]) -> boolean().
can_partition(Nums) ->
.
```

Racket Solution:

```
(define/contract (can-partition nums)
(-> (listof exact-integer?) boolean?)
)
```