Problem 3548: Equal Sum Grid Partition II

Problem Information

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

Problem Description

You are given an

m x n

matrix

grid

of positive integers. Your task is to determine if it is possible to make

either one horizontal or one vertical cut

on the grid such that:

Each of the two resulting sections formed by the cut is

non-empty

.

The sum of elements in both sections is

equal

, or can be made equal by discounting

at most

one single cell in total (from either section).

If a cell is discounted, the rest of the section must

remain connected

.

Return

true

if such a partition exists; otherwise, return

false

.

Note:

A section is

connected

if every cell in it can be reached from any other cell by moving up, down, left, or right through other cells in the section.

Example 1:

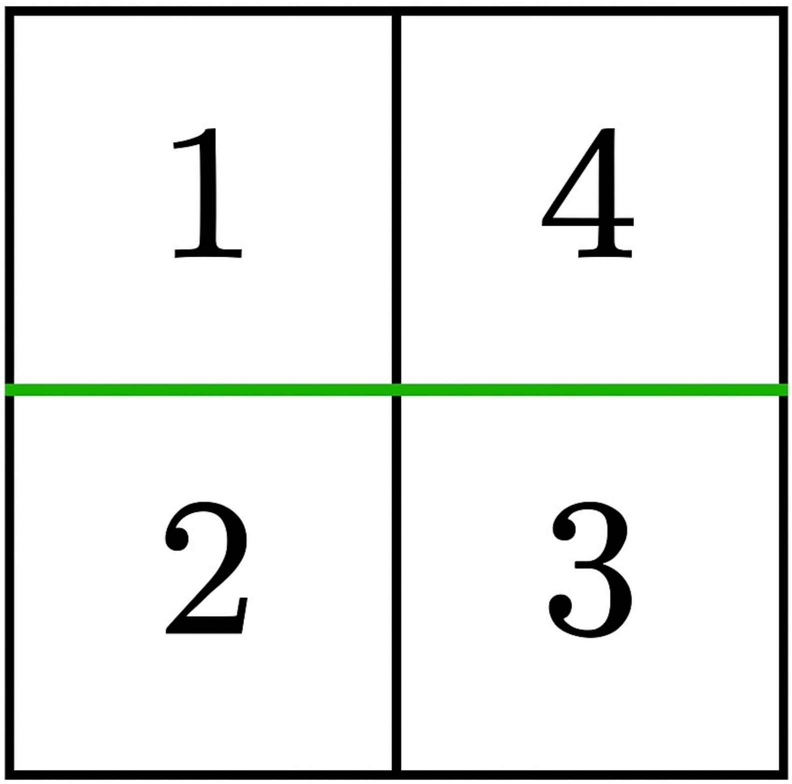
Input:

grid = [[1,4],[2,3]]

Output:

true

Explanation:



A horizontal cut after the first row gives sums

1 + 4 = 5

and

2 + 3 = 5

, which are equal. Thus, the answer is

true

.

Example 2:

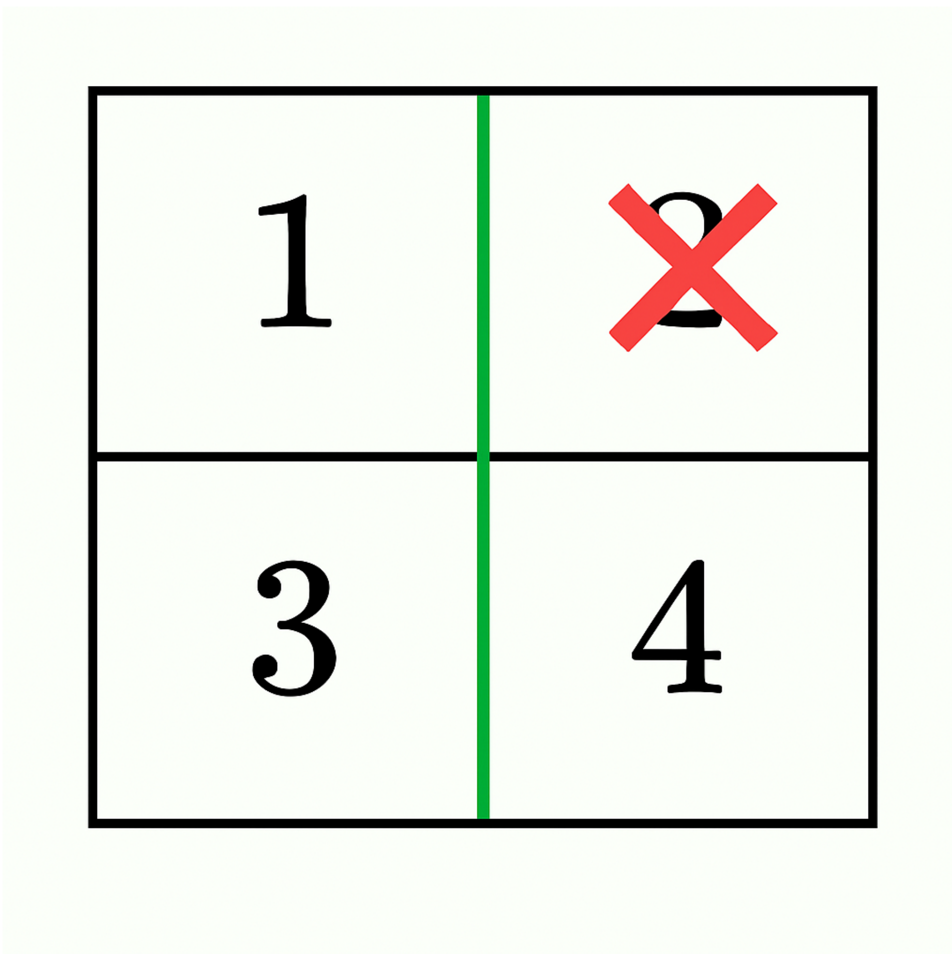
Input:

grid = [[1,2],[3,4]]

Output:

true

Explanation:



A vertical cut after the first column gives sums

1 + 3 = 4

and

2 + 4 = 6

.

By discounting 2 from the right section (

6 - 2 = 4

), both sections have equal sums and remain connected. Thus, the answer is

true

.

Example 3:

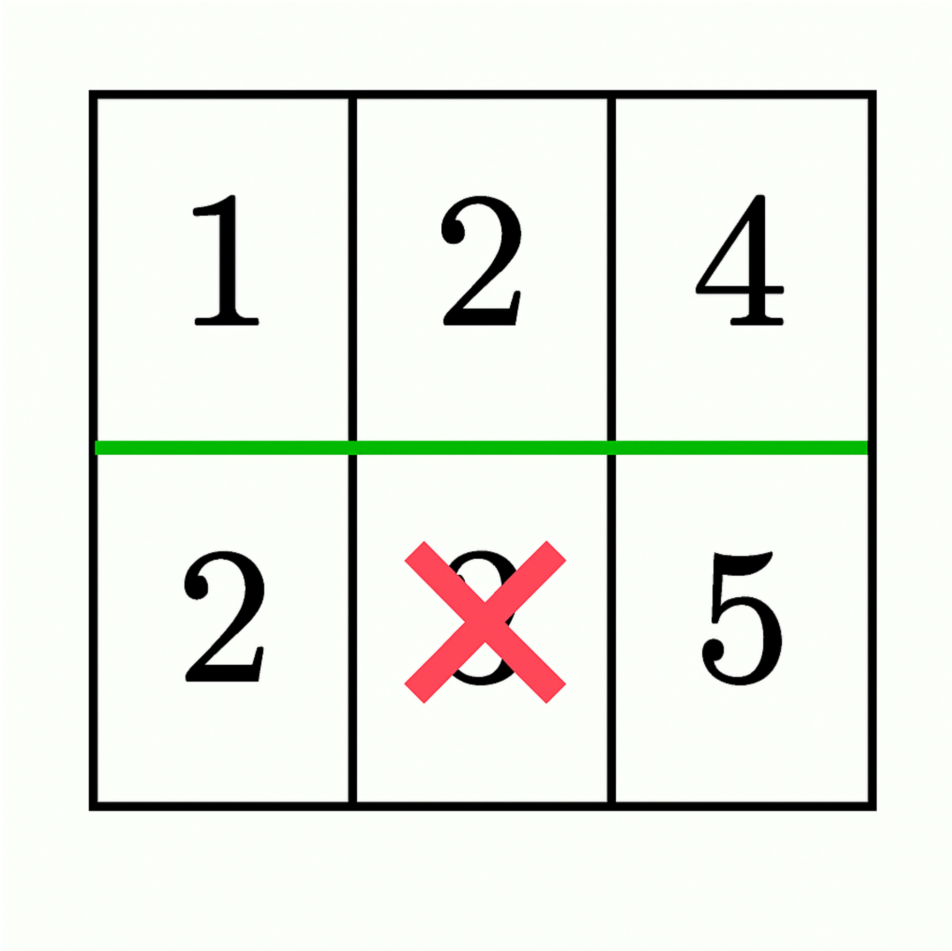
Input:

grid = [[1,2,4],[2,3,5]]

Output:

false

Explanation:



A horizontal cut after the first row gives

$1 + 2 + 4 = 7$

and

$2 + 3 + 5 = 10$

.

By discounting 3 from the bottom section (

$10 - 3 = 7$

), both sections have equal sums, but they do not remain connected as it splits the bottom section into two parts (

[2]

and

[5]

). Thus, the answer is

false

.

Example 4:

Input:

grid = [[4,1,8],[3,2,6]]

Output:

false

Explanation:

No valid cut exists, so the answer is

false

.

Constraints:

1 <= m == grid.length <= 10

5

1 <= n == grid[i].length <= 10

5

2 <= m * n <= 10

5

1 <= grid[i][j] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
bool canPartitionGrid(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public boolean canPartitionGrid(int[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function canPartitionGrid(grid: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool CanPartitionGrid(int[][] grid) {

}
}
```

C:

```
bool canPartitionGrid(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func canPartitionGrid(grid [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun canPartitionGrid(grid: Array<IntArray>): Boolean {

}
}
```

Swift:

```
class Solution {
func canPartitionGrid(_ grid: [[Int]]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_partition_grid(grid: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Boolean}
def can_partition_grid(grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Boolean
*/
function canPartitionGrid($grid) {

}
}
```

Dart:

```
class Solution {
bool canPartitionGrid(List<List<int>> grid) {

}
```

```
}
```

Scala:

```
object Solution {
def canPartitionGrid(grid: Array[Array[Int]]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_partition_grid(grid :: [[integer]]) :: boolean
def can_partition_grid(grid) do

end
end
```

Erlang:

```
-spec can_partition_grid(Grid :: [[integer()]]) -> boolean().
can_partition_grid(Grid) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Equal Sum Grid Partition II
* Difficulty: Hard
* 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:
bool canPartitionGrid(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Equal Sum Grid Partition II
* Difficulty: Hard
* 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 boolean canPartitionGrid(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Equal Sum Grid Partition II
Difficulty: Hard
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 canPartitionGrid(self, grid: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Equal Sum Grid Partition II
* Difficulty: Hard
* 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[][]} grid
* @return {boolean}
*/
var canPartitionGrid = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Equal Sum Grid Partition II
* Difficulty: Hard
* 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 canPartitionGrid(grid: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Equal Sum Grid Partition II
* Difficulty: Hard
* 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 bool CanPartitionGrid(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Equal Sum Grid Partition II
* Difficulty: Hard
* 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
*/

bool canPartitionGrid(int** grid, int gridSize, int* gridColSize) {
```

```
}
```

Go Solution:

```go
// Problem: Equal Sum Grid Partition II
// Difficulty: Hard
// 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 canPartitionGrid(grid [][]int) bool {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun canPartitionGrid(grid: Array<IntArray>): Boolean {

}
}
```

Swift Solution:

```swift
class Solution {
func canPartitionGrid(_ grid: [[Int]]) -> Bool {

}
}
```

Rust Solution:

```rust
// Problem: Equal Sum Grid Partition II
// Difficulty: Hard
// 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 can_partition_grid(grid: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Boolean}
def can_partition_grid(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Boolean
*/
function canPartitionGrid($grid) {

}
}
```

Dart Solution:

```
class Solution {
bool canPartitionGrid(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def canPartitionGrid(grid: Array[Array[Int]]): Boolean = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_partition_grid(grid :: [[integer]]) :: boolean
def can_partition_grid(grid) do

end
end
```

Erlang Solution:

```
-spec can_partition_grid(Grid :: [[integer()]]) -> boolean().
can_partition_grid(Grid) ->
.
```

Racket Solution:

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