Problem 554: Brick Wall

Problem Information

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

Problem Description

There is a rectangular brick wall in front of you with

n

rows of bricks. The

i

th

row has some number of bricks each of the same height (i.e., one unit) but they can be of different widths. The total width of each row is the same.

Draw a vertical line from the top to the bottom and cross the least bricks. If your line goes through the edge of a brick, then the brick is not considered as crossed. You cannot draw a line just along one of the two vertical edges of the wall, in which case the line will obviously cross no bricks.

Given the 2D array

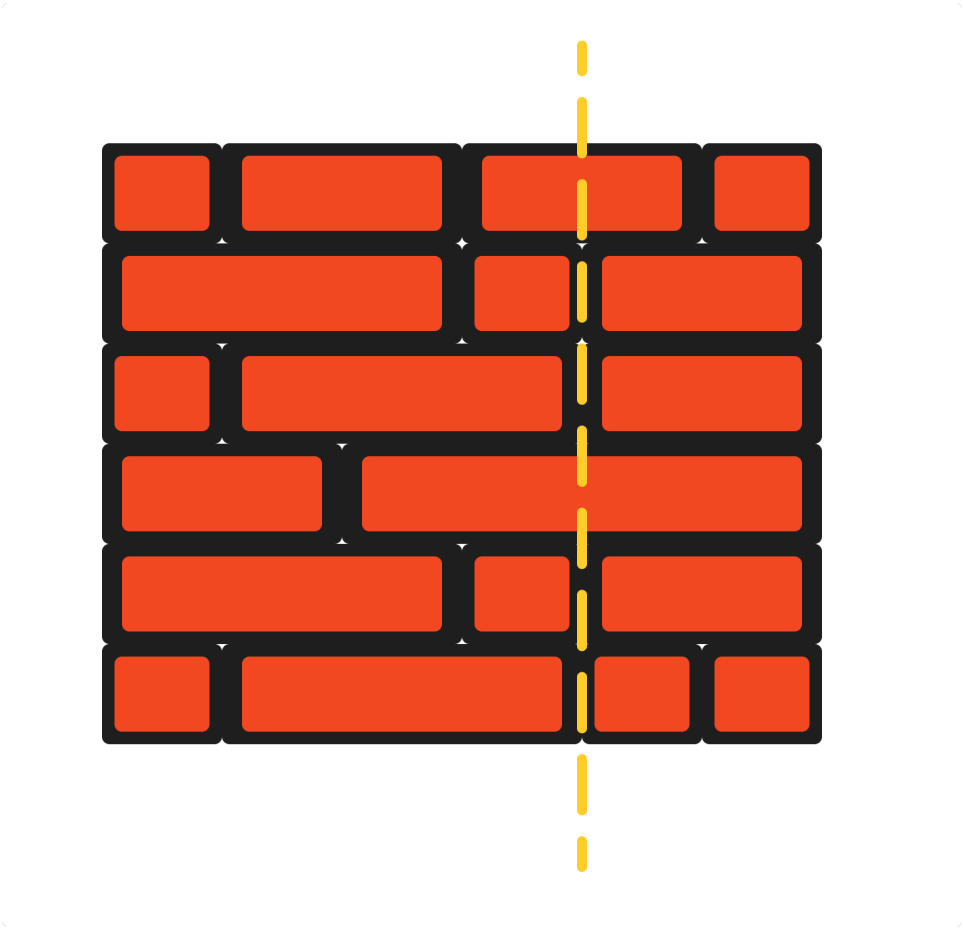
wall

that contains the information about the wall, return

the minimum number of crossed bricks after drawing such a vertical line

.

Example 1:



Input:

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

Output:

2

Example 2:

Input:

wall = [[1],[1],[1]]

Output:

3

Constraints:

n == wall.length

1 <= n <= 10

4

1 <= wall[i].length <= 10

4

1 <= sum(wall[i].length) <= 2 * 10

4

sum(wall[i])

is the same for each row

i

.

1 <= wall[i][j] <= 2

31

- 1

Code Snippets

C++:

```
class Solution {
public:
int leastBricks(vector<vector<int>>& wall) {

}
};
```

Java:

```
class Solution {
public int leastBricks(List<List<Integer>> wall) {

}
}
```

Python3:

```
class Solution:
def leastBricks(self, wall: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def leastBricks(self, wall):
"""
:type wall: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} wall
* @return {number}
*/
var leastBricks = function(wall) {

};
```

TypeScript:

```
function leastBricks(wall: number[][]): number {
```

```
};
```

C#:

```
public class Solution {
public int LeastBricks(IList<IList<int>> wall) {

}
}
```

C:

```
int leastBricks(int** wall, int wallSize, int* wallColSize) {

}
```

Go:

```
func leastBricks(wall [][]int) int {

}
```

Kotlin:

```
class Solution {
fun leastBricks(wall: List<List<Int>>): Int {

}
}
```

Swift:

```
class Solution {
func leastBricks(_ wall: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn least_bricks(wall: Vec<Vec<i32>>) -> i32 {
```

```
}
}
```

Ruby:

```
# @param {Integer[][]} wall
# @return {Integer}
def least_bricks(wall)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $wall
* @return Integer
*/
function leastBricks($wall) {

}
}
```

Dart:

```
class Solution {
int leastBricks(List<List<int>> wall) {

}
}
```

Scala:

```
object Solution {
def leastBricks(wall: List[List[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec least_bricks(wall :: [[integer]]) :: integer
def least_bricks(wall) do

end
end
```

Erlang:

```
-spec least_bricks(Wall :: [[integer()]]) -> integer().
least_bricks(Wall) ->
.
```

Racket:

```
(define/contract (least-bricks wall)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Brick Wall
* Difficulty: Medium
* 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:
int leastBricks(vector<vector<int>>& wall) {

}
};
```

Java Solution:

```
/**
* Problem: Brick Wall
* Difficulty: Medium
* 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 int leastBricks(List<List<Integer>> wall) {

}
}
```

Python3 Solution:

```
"""
Problem: Brick Wall
Difficulty: Medium
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 leastBricks(self, wall: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def leastBricks(self, wall):
"""
:type wall: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Brick Wall
* Difficulty: Medium
* 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[][]} wall
* @return {number}
*/
var leastBricks = function(wall) {

};
```

TypeScript Solution:

```
/**
* Problem: Brick Wall
* Difficulty: Medium
* 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 leastBricks(wall: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Brick Wall
* Difficulty: Medium
* 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 int LeastBricks(IList<IList<int>> wall) {

}
}
```

C Solution:

```
/*
* Problem: Brick Wall
* Difficulty: Medium
* 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
*/

int leastBricks(int** wall, int wallSize, int* wallColSize) {

}
```

Go Solution:

```
// Problem: Brick Wall
// Difficulty: Medium
// 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 leastBricks(wall [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun leastBricks(wall: List<List<Int>>): Int {

}
}
```

Swift Solution:

```
class Solution {
func leastBricks(_ wall: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Brick Wall
// Difficulty: Medium
// 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 least_bricks(wall: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} wall
# @return {Integer}
def least_bricks(wall)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $wall
* @return Integer
*/
function leastBricks($wall) {

}
}
```

Dart Solution:

```
class Solution {
int leastBricks(List<List<int>> wall) {

}
}
```

Scala Solution:

```
object Solution {
def leastBricks(wall: List[List[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec least_bricks(wall :: [[integer]]) :: integer
def least_bricks(wall) do

end
end
```

Erlang Solution:

```
-spec least_bricks(Wall :: [[integer()]]) -> integer().
least_bricks(Wall) ->
.
```

Racket Solution:

```
(define/contract (least-bricks wall)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```