Problem 2184: Number of Ways to Build Sturdy Brick Wall

Problem Information

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

Problem Description

You are given integers

height

and

width

which specify the dimensions of a brick wall you are building. You are also given a

0-indexed

array of

unique

integers

bricks

, where the

i

th

brick has a height of

1

and a width of

bricks[i]

. You have an

infinite

supply of each type of brick and bricks may

not

be rotated.

Each row in the wall must be exactly

width

units long. For the wall to be

sturdy

, adjacent rows in the wall should

not

join bricks at the same location, except at the ends of the wall.

Return

the number of ways to build a

sturdy

wall.

Since the answer may be very large, return it

modulo

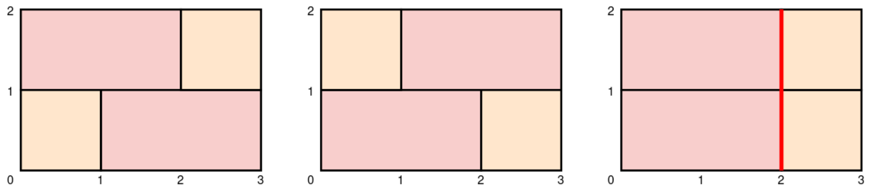
10

9

+ 7

.

Example 1:



Input:

height = 2, width = 3, bricks = [1,2]

Output:

2

Explanation:

The first two walls in the diagram show the only two ways to build a sturdy brick wall. Note that the third wall in the diagram is not sturdy because adjacent rows join bricks 2 units from the left.

Example 2:

Input:

height = 1, width = 1, bricks = [5]

Output:

0

Explanation:

There are no ways to build a sturdy wall because the only type of brick we have is longer than the width of the wall.

Constraints:

1 <= height <= 100

1 <= width <= 10

1 <= bricks.length <= 10

1 <= bricks[i] <= 10

All the values of

bricks

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int buildWall(int height, int width, vector<int>& bricks) {
}
};
```

Java:

```
class Solution {
public int buildWall(int height, int width, int[] bricks) {

}
}
```

Python3:

```
class Solution:
def buildWall(self, height: int, width: int, bricks: List[int]) -> int:
```

Python:

```
class Solution(object):
def buildWall(self, height, width, bricks):
"""
:type height: int
:type width: int
:type bricks: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} height
* @param {number} width
* @param {number[]} bricks
* @return {number}
*/
var buildWall = function(height, width, bricks) {

};
```

TypeScript:

```
function buildWall(height: number, width: number, bricks: number[]): number {

};
```

C#:

```
public class Solution {
public int BuildWall(int height, int width, int[] bricks) {

}
}
```

C:

```
int buildWall(int height, int width, int* bricks, int bricksSize) {

}
```

Go:

```
func buildWall(height int, width int, bricks []int) int {

}
```

Kotlin:

```
class Solution {
fun buildWall(height: Int, width: Int, bricks: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func buildWall(_ height: Int, _ width: Int, _ bricks: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn build_wall(height: i32, width: i32, bricks: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} height
# @param {Integer} width
# @param {Integer[]} bricks
# @return {Integer}
def build_wall(height, width, bricks)

end
```

PHP:

```
class Solution {

/**
* @param Integer $height
* @param Integer $width
* @param Integer[] $bricks
* @return Integer
*/
function buildWall($height, $width, $bricks) {

}
}
```

Dart:

```
class Solution {
int buildWall(int height, int width, List<int> bricks) {

}
}
```

Scala:

```
object Solution {
def buildWall(height: Int, width: Int, bricks: Array[Int]): Int = {
}
}
```

Elixir:

```
defmodule Solution do
@spec build_wall(height :: integer, width :: integer, bricks :: [integer]) ::
integer
def build_wall(height, width, bricks) do

end
end
```

Erlang:

```
-spec build_wall(Height :: integer(), Width :: integer(), Bricks ::
[integer()]) -> integer().
build_wall(Height, Width, Bricks) ->
.
```

Racket:

```
(define/contract (build-wall height width bricks)
(-> exact-integer? exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Build Sturdy Brick Wall
* 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:
int buildWall(int height, int width, vector<int>& bricks) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Build Sturdy Brick Wall
* 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 int buildWall(int height, int width, int[] bricks) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Build Sturdy Brick Wall
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 buildWall(self, height: int, width: int, bricks: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def buildWall(self, height, width, bricks):
"""
:type height: int
:type width: int
:type bricks: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways to Build Sturdy Brick Wall
* 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} height
* @param {number} width
* @param {number[]} bricks
* @return {number}
*/
var buildWall = function(height, width, bricks) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Build Sturdy Brick Wall
* 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 buildWall(height: number, width: number, bricks: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Build Sturdy Brick Wall
* 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 int BuildWall(int height, int width, int[] bricks) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Build Sturdy Brick Wall
* 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
*/

int buildWall(int height, int width, int* bricks, int bricksSize) {
}
```

Go Solution:

```
// Problem: Number of Ways to Build Sturdy Brick Wall
// 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 buildWall(height int, width int, bricks []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun buildWall(height: Int, width: Int, bricks: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func buildWall(_ height: Int, _ width: Int, _ bricks: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Build Sturdy Brick Wall
// 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 build_wall(height: i32, width: i32, bricks: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} height
# @param {Integer} width
# @param {Integer[]} bricks
# @return {Integer}
def build_wall(height, width, bricks)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $height
* @param Integer $width
* @param Integer[] $bricks
* @return Integer
*/
function buildWall($height, $width, $bricks) {

}
}
```

Dart Solution:

```
class Solution {
int buildWall(int height, int width, List<int> bricks) {

}
}
```

Scala Solution:

```
object Solution {
def buildWall(height: Int, width: Int, bricks: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec build_wall(height :: integer, width :: integer, bricks :: [integer]) ::
integer
def build_wall(height, width, bricks) do

end
end
```

Erlang Solution:

```
-spec build_wall(Height :: integer(), Width :: integer(), Bricks ::
[integer()]) -> integer().
build_wall(Height, Width, Bricks) ->
.
```

Racket Solution:

```
(define/contract (build-wall height width bricks)
(-> exact-integer? exact-integer? (listof exact-integer?) exact-integer?)
)
```