Problem 533: Lonely Pixel II

Problem Information

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

Problem Description

Given an

m x n

picture

consisting of black

'B'

and white

'W'

pixels and an integer target, return

the number of

black

lonely pixels

.

A black lonely pixel is a character

'B'

that located at a specific position

(r, c)

where:

Row

r

and column

c

both contain exactly

target

black pixels.

For all rows that have a black pixel at column

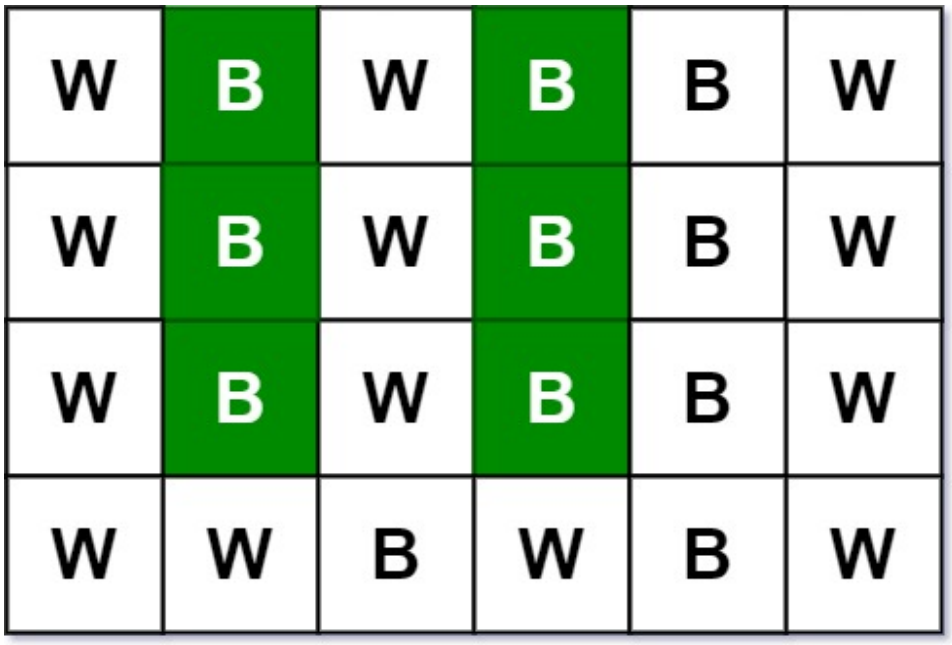
c

, they should be exactly the same as row

r

.

Example 1:



Input:

picture = [["W","B","W","B","B","W"],["W","B","W","B","B","W"],["W","B","W","B","B","W"],["W","W","B","W","B","W"]], target = 3

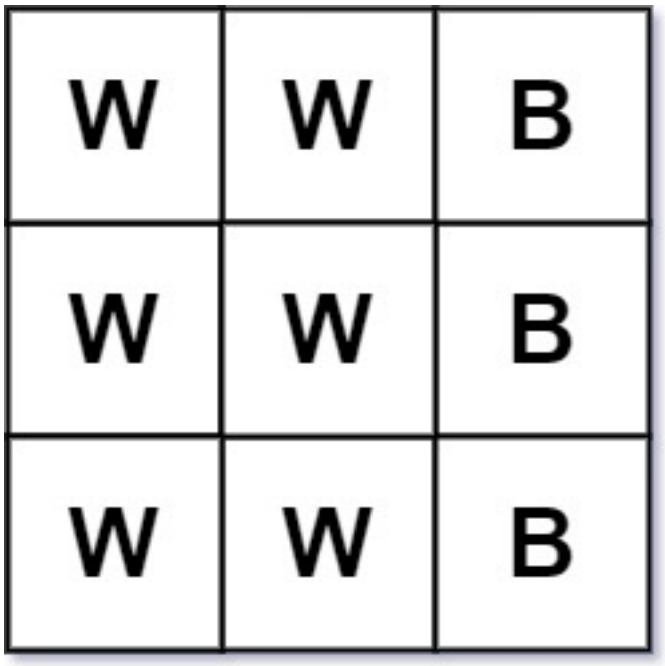
Output:

6

Explanation:

All the green 'B' are the black pixels we need (all 'B's at column 1 and 3). Take 'B' at row r = 0 and column c = 1 as an example: - Rule 1, row r = 0 and column c = 1 both have exactly target = 3 black pixels. - Rule 2, the rows have black pixel at column c = 1 are row 0, row 1 and row 2. They are exactly the same as row r = 0.

Example 2:



Input:

picture = [["W","W","B"],["W","W","B"],["W","W","B"]], target = 1

Output:

0

Constraints:

m == picture.length

n == picture[i].length

1 <= m, n <= 200

picture[i][j]

is

'W'

or

'B'

.

1 <= target <= min(m, n)

Code Snippets

C++:

```
class Solution {
public:
int findBlackPixel(vector<vector<char>>& picture, int target) {

}
};
```

Java:

```
class Solution {
public int findBlackPixel(char[][] picture, int target) {

}
}
```

Python3:

```
class Solution:
def findBlackPixel(self, picture: List[List[str]], target: int) -> int:
```

Python:

```
class Solution(object):
def findBlackPixel(self, picture, target):
"""
:type picture: List[List[str]]
:type target: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {character[][]} picture
* @param {number} target
* @return {number}
*/
var findBlackPixel = function(picture, target) {

};
```

TypeScript:

```
function findBlackPixel(picture: string[][], target: number): number {

};
```

C#:

```
public class Solution {
public int FindBlackPixel(char[][] picture, int target) {

}
}
```

C:

```
int findBlackPixel(char** picture, int pictureSize, int* pictureColSize, int
target) {

}
```

Go:

```
func findBlackPixel(picture [][]byte, target int) int {

}
```

Kotlin:

```
class Solution {
fun findBlackPixel(picture: Array<CharArray>, target: Int): Int {

}
}
```

Swift:

```
class Solution {
func findBlackPixel(_ picture: [[Character]], _ target: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_black_pixel(picture: Vec<Vec<char>>, target: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Character[][]} picture
# @param {Integer} target
# @return {Integer}
def find_black_pixel(picture, target)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $picture
* @param Integer $target
* @return Integer
*/
function findBlackPixel($picture, $target) {

}
}
```

Dart:

```
class Solution {
int findBlackPixel(List<List<String>> picture, int target) {
```

```
}
}
```

Scala:

```
object Solution {
def findBlackPixel(picture: Array[Array[Char]], target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_black_pixel(picture :: [[char]], target :: integer) :: integer
def find_black_pixel(picture, target) do

end
end
```

Erlang:

```
-spec find_black_pixel(Picture :: [[char()]], Target :: integer()) ->
integer().
find_black_pixel(Picture, Target) ->
.
```

Racket:

```
(define/contract (find-black-pixel picture target)
(-> (listof (listof char?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Lonely Pixel II
```

```
* 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 findBlackPixel(vector<vector<char>>& picture, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Lonely Pixel II
* 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 findBlackPixel(char[][] picture, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Lonely Pixel II
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 findBlackPixel(self, picture: List[List[str]], target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findBlackPixel(self, picture, target):
"""
:type picture: List[List[str]]
:type target: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Lonely Pixel II
* 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 {character[][]} picture
* @param {number} target
* @return {number}
*/
var findBlackPixel = function(picture, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Lonely Pixel II
* 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 findBlackPixel(picture: string[][], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Lonely Pixel II
* 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 FindBlackPixel(char[][] picture, int target) {

}
}
```

C Solution:

```
/*
* Problem: Lonely Pixel II
* 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 findBlackPixel(char** picture, int pictureSize, int* pictureColSize, int
target) {

}
```

Go Solution:

```
// Problem: Lonely Pixel II
// 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 findBlackPixel(picture [][]byte, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findBlackPixel(picture: Array<CharArray>, target: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findBlackPixel(_ picture: [[Character]], _ target: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Lonely Pixel II
// 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 find_black_pixel(picture: Vec<Vec<char>>, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Character[][]} picture
# @param {Integer} target
# @return {Integer}
def find_black_pixel(picture, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $picture
* @param Integer $target
* @return Integer
*/
function findBlackPixel($picture, $target) {

}
}
```

Dart Solution:

```
class Solution {
int findBlackPixel(List<List<String>> picture, int target) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def findBlackPixel(picture: Array[Array[Char]], target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_black_pixel(picture :: [[char]], target :: integer) :: integer
def find_black_pixel(picture, target) do

end
end
```

Erlang Solution:

```
-spec find_black_pixel(Picture :: [[char()]], Target :: integer()) ->
integer().
find_black_pixel(Picture, Target) ->
.
```

Racket Solution:

```
(define/contract (find-black-pixel picture target)
(-> (listof (listof char?)) exact-integer? exact-integer?)
)
```