Problem 531: Lonely Pixel I

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, return

the number of

black

lonely pixels

.

A black lonely pixel is a character

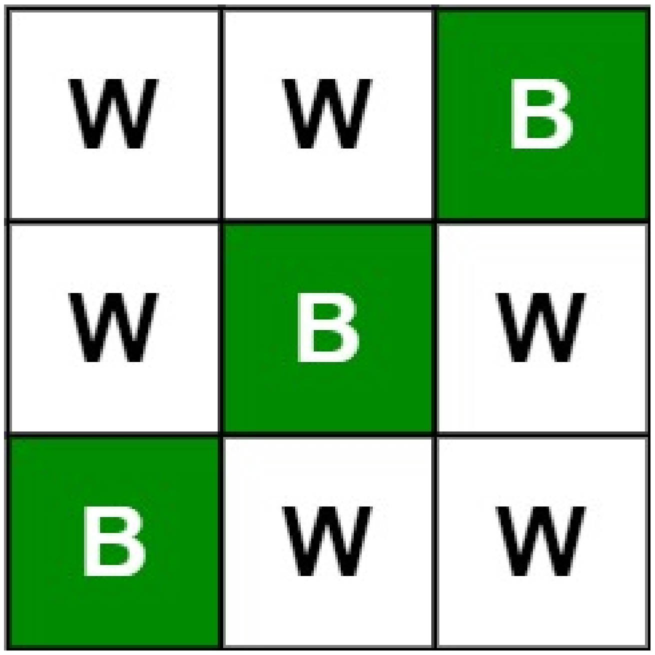
'B'

that located at a specific position where the same row and same column don't have

any other

black pixels.

Example 1:



Input:

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

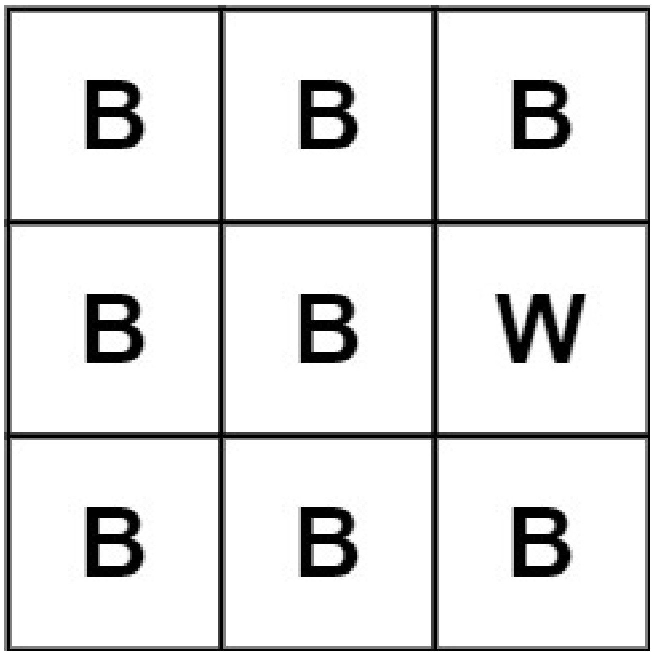
Output:

3

Explanation:

All the three 'B's are black lonely pixels.

Example 2:



B	B	B
B	B	W
B	B	B

Input:

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

Output:

0

Constraints:

m == picture.length

n == picture[i].length

1 <= m, n <= 500

picture[i][j]

is

'W'

or

'B'

.

Code Snippets

C++:

```
class Solution {
public:
int findLonelyPixel(vector<vector<char>>& picture) {

}
};
```

Java:

```
class Solution {
public int findLonelyPixel(char[][] picture) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findLonelyPixel(picture: string[][]): number {

};
```

C#:

```
public class Solution {
public int FindLonelyPixel(char[][] picture) {

}
}
```

C:

```
int findLonelyPixel(char** picture, int pictureSize, int* pictureColSize) {

}
```

Go:

```
func findLonelyPixel(picture [][]byte) int {

}
```

Kotlin:

```
class Solution {
fun findLonelyPixel(picture: Array<CharArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Character[][]} picture
# @return {Integer}
def find_lonely_pixel(picture)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $picture
* @return Integer
*/
function findLonelyPixel($picture) {

}
}
```

Dart:

```
class Solution {
int findLonelyPixel(List<List<String>> picture) {

}
}
```

Scala:

```
object Solution {
def findLonelyPixel(picture: Array[Array[Char]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_lonely_pixel(picture :: [[char]]) :: integer
def find_lonely_pixel(picture) do

end
end
```

Erlang:

```
-spec find_lonely_pixel(Picture :: [[char()]]) -> integer().
find_lonely_pixel(Picture) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Lonely Pixel I
* 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 findLonelyPixel(vector<vector<char>>& picture) {

}
};
```

Java Solution:

```
/**
* Problem: Lonely Pixel I
* 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 findLonelyPixel(char[][] picture) {

}
}
```

Python3 Solution:

```
"""
Problem: Lonely Pixel I
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 findLonelyPixel(self, picture: List[List[str]]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lonely Pixel I
* 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
* @return {number}
*/
var findLonelyPixel = function(picture) {

};
```

TypeScript Solution:

```
/**
* Problem: Lonely Pixel I
* 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 findLonelyPixel(picture: string[][]): number {

};
```

C# Solution:

```
/*
* Problem: Lonely Pixel I
* 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 FindLonelyPixel(char[][] picture) {

}
}
```

C Solution:

```
/*
* Problem: Lonely Pixel I
* 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 findLonelyPixel(char** picture, int pictureSize, int* pictureColSize) {

}
```

Go Solution:

```
// Problem: Lonely Pixel I
// 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 findLonelyPixel(picture [][]byte) int {

}
```

Kotlin Solution:

```
class Solution {
fun findLonelyPixel(picture: Array<CharArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Lonely Pixel I
// 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_lonely_pixel(picture: Vec<Vec<char>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Character[][]} picture
# @return {Integer}
def find_lonely_pixel(picture)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $picture
* @return Integer
*/
function findLonelyPixel($picture) {

}
}
```

Dart Solution:

```
class Solution {
int findLonelyPixel(List<List<String>> picture) {

}
}
```

Scala Solution:

```
object Solution {
def findLonelyPixel(picture: Array[Array[Char]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_lonely_pixel(picture :: [[char]]) :: integer
def find_lonely_pixel(picture) do

end
end
```

Erlang Solution:

```
-spec find_lonely_pixel(Picture :: [[char()]]) -> integer().
find_lonely_pixel(Picture) ->
.
```

Racket Solution:

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