Problem 999: Available Captures for Rook

Problem Information

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

Problem Description

You are given an

8 x 8

matrix

representing a chessboard. There is

exactly one

white rook represented by

'R'

, some number of white bishops

'B'

, and some number of black pawns

'p'

. Empty squares are represented by

'.'

.

A rook can move any number of squares horizontally or vertically (up, down, left, right) until it reaches another piece

or

the edge of the board. A rook is

attacking

a pawn if it can move to the pawn's square in one move.

Note: A rook cannot move through other pieces, such as bishops or pawns. This means a rook cannot attack a pawn if there is another piece blocking the path.

Return the

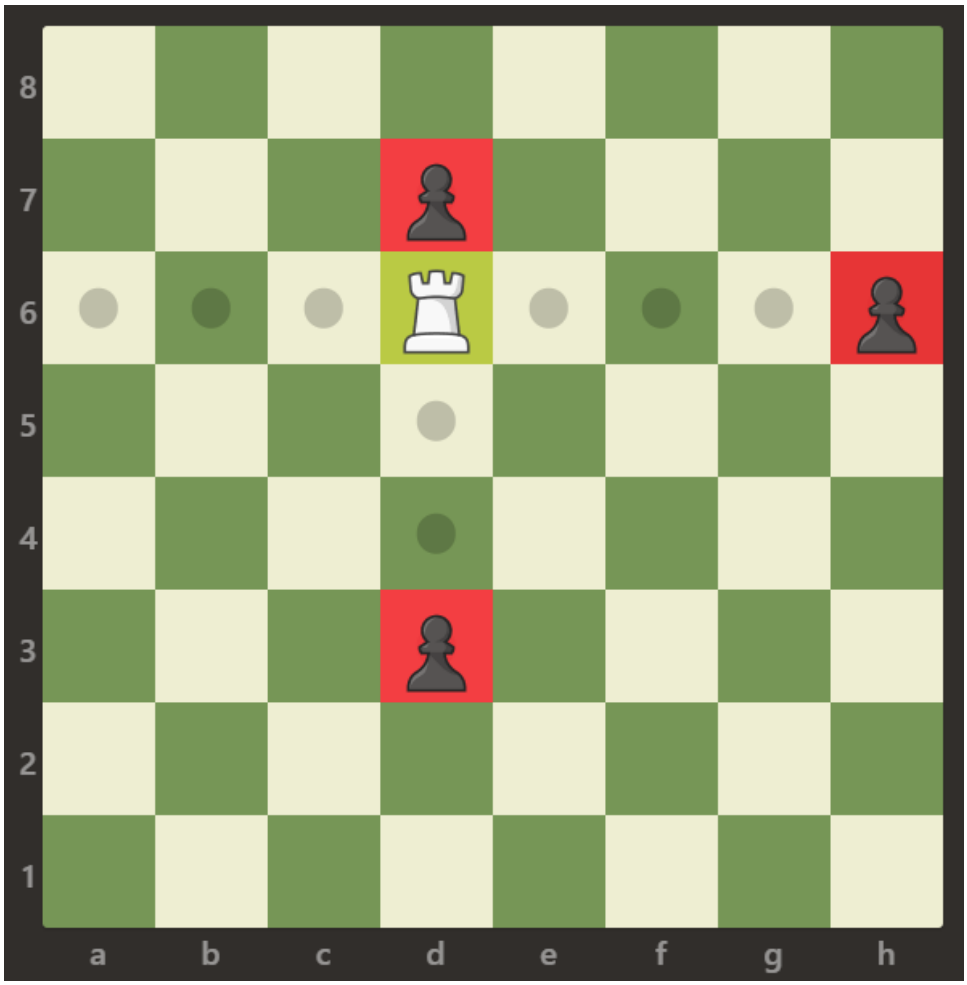
number of pawns

the white rook is

attacking

.

Example 1:



Input:

board = [[".",".",".",".",".",".",".",".",],[".",".",".",".","p",".",".",".",".",".",],[".",".",".",".","R",".",".",".",".","p"],[".",".",".",".",".",".",".",".",],[".",".",".",".",".",".",".",".",],[".",".",".",".","p",".",".",".",".",".",],[".",".",".",".",".",".",".",".",".",".",],[".",".",".",".",".",".",".",".",".",".",]]

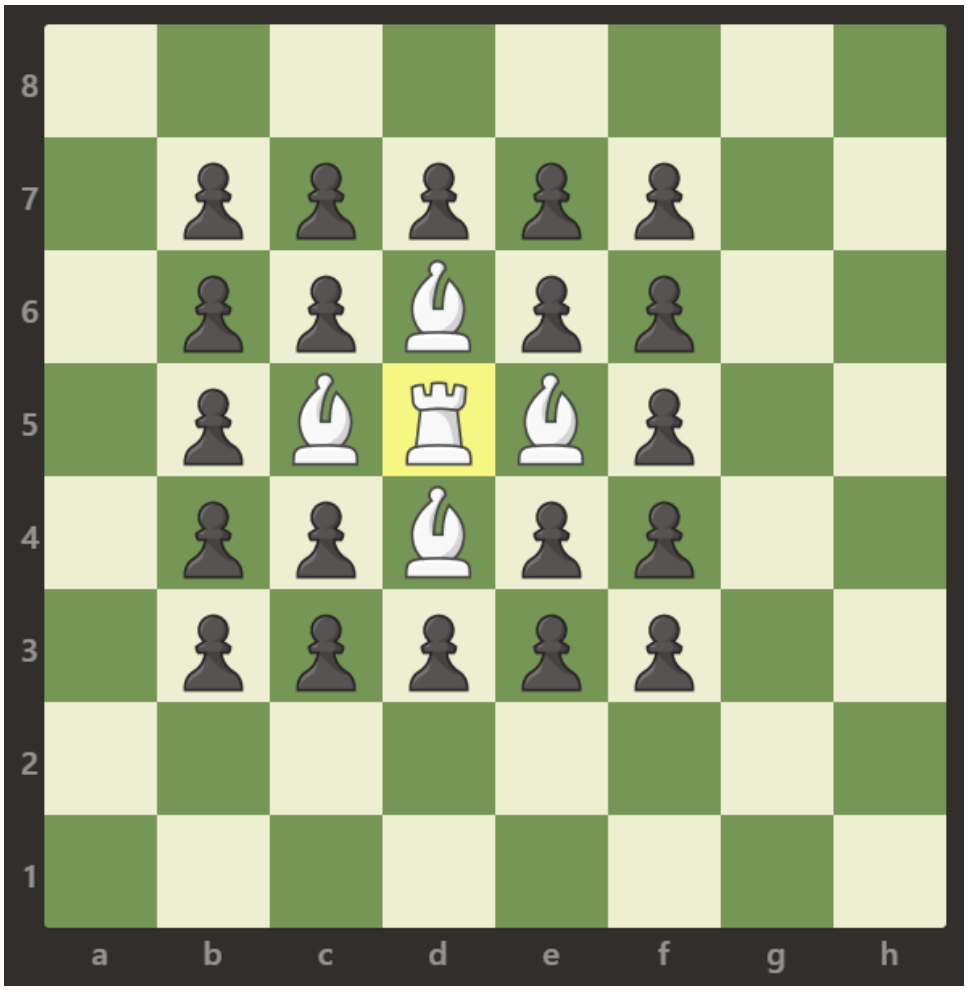
Output:

3

Explanation:

In this example, the rook is attacking all the pawns.

Example 2:



Input:

board = [[".",".",".",".",".",".",".","."],[".","p","p","p","p","p",".","."],[".","p","p","B","p","p",".","."],[".","p","B","R","B","p",".","."],[".","p","p","B","p","p",".","."],[".","p","p","p","p","p",".","."],[".",".",".",".",".",".",".","."],[".",".",".",".",".",".",".","."]]

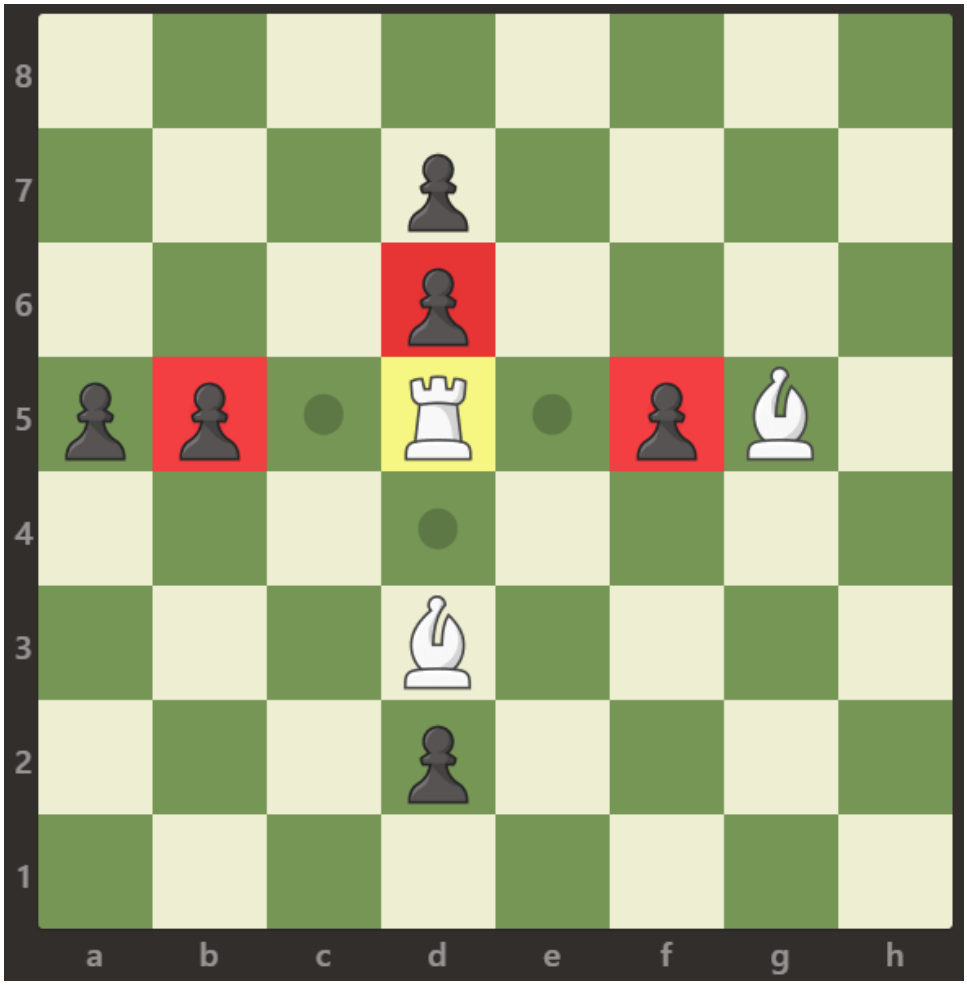
Output:

0

Explanation:

The bishops are blocking the rook from attacking any of the pawns.

Example 3:



Input:

board = [[".",".",".",".",".",".",".",".",".",."],[".",".",".",".","p",".",".",".",".","."],[".",".",".",".","p",".",".",".",".","."],["p","p",".","R",".","p","B","."],[".",".",".",".",".",".",".",".","."],[".",".",".",".","B",".",".",".",".","."],[".",".",".",".","p",".",".",".",".","."],[".",".",".",".",".",".",".",".",".","."]]

Output:

3

Explanation:

The rook is attacking the pawns at positions b5, d6, and f5.

Constraints:

board.length == 8

board[i].length == 8

board[i][j]

is either

'R'

,

'.'

,

'B'

, or

'p'

There is exactly one cell with

board[i][j] == 'R'

Code Snippets

C++:

```
class Solution {
public:
int numRookCaptures(vector<vector<char>>& board) {

}
};
```

Java:

```
class Solution {
public int numRookCaptures(char[][] board) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numRookCaptures(board: string[][]): number {

};
```

C#:

```
public class Solution {
public int NumRookCaptures(char[][] board) {

}
}
```

C:

```
int numRookCaptures(char** board, int boardSize, int* boardColSize) {

}
```

Go:

```
func numRookCaptures(board [][]byte) int {

}
```

Kotlin:

```
class Solution {
fun numRookCaptures(board: Array<CharArray>): Int {

}
}
```

Swift:

```
class Solution {
func numRookCaptures(_ board: [[Character]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_rook_captures(board: Vec<Vec<char>>) -> i32 {

}
}
```

Ruby:

```
# @param {Character[][]} board
# @return {Integer}
def num_rook_captures(board)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $board
* @return Integer
*/
function numRookCaptures($board) {

}
}
```

Dart:

```
class Solution {
int numRookCaptures(List<List<String>> board) {

}
}
```

Scala:

```
object Solution {
def numRookCaptures(board: Array[Array[Char]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_rook_captures(board :: [[char]]) :: integer
def num_rook_captures(board) do

end
end
```

Erlang:

```
-spec num_rook_captures(Board :: [[char()]]) -> integer().
num_rook_captures(Board) ->
.
```

Racket:

```
(define/contract (num-rook-captures board)
(-> (listof (listof char?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Available Captures for Rook
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int numRookCaptures(vector<vector<char>>& board) {

}
};
```

Java Solution:

```
/**
* Problem: Available Captures for Rook
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int numRookCaptures(char[][] board) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Available Captures for Rook
Difficulty: Easy
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def numRookCaptures(self, board: List[List[str]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Available Captures for Rook
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Available Captures for Rook
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function numRookCaptures(board: string[][]): number {

};
```

C# Solution:

```
/*
* Problem: Available Captures for Rook
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int NumRookCaptures(char[][] board) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Available Captures for Rook
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int numRookCaptures(char** board, int boardSize, int* boardColSize) {

}
```

Go Solution:

```
// Problem: Available Captures for Rook
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func numRookCaptures(board [][]byte) int {

}
```

Kotlin Solution:

```
class Solution {
fun numRookCaptures(board: Array<CharArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func numRookCaptures(_ board: [[Character]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Available Captures for Rook
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn num_rook_captures(board: Vec<Vec<char>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Character[][]} board
# @return {Integer}
def num_rook_captures(board)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $board
* @return Integer
*/
function numRookCaptures($board) {

}
}
```

Dart Solution:

```
class Solution {
int numRookCaptures(List<List<String>> board) {

}
}
```

Scala Solution:

```
object Solution {
def numRookCaptures(board: Array[Array[Char]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_rook_captures(board :: [[char]]) :: integer
def num_rook_captures(board) do

end
end
```

Erlang Solution:

```
-spec num_rook_captures(Board :: [[char()]]) -> integer().
num_rook_captures(Board) ->
.
```

Racket Solution:

```
(define/contract (num-rook-captures board)
(-> (listof (listof char?)) exact-integer?)
)
```