Problem 1958: Check if Move is Legal

Problem Information

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

Problem Description

You are given a

0-indexed

8 x 8

grid

board

, where

board[r][c]

represents the cell

(r, c)

on a game board. On the board, free cells are represented by

'.'

, white cells are represented by

'W'

, and black cells are represented by

'B'

.

Each move in this game consists of choosing a free cell and changing it to the color you are playing as (either white or black). However, a move is only

legal

if, after changing it, the cell becomes the

endpoint of a good line

(horizontal, vertical, or diagonal).

A

good line

is a line of

three or more cells (including the endpoints)

where the endpoints of the line are

one color

, and the remaining cells in the middle are the

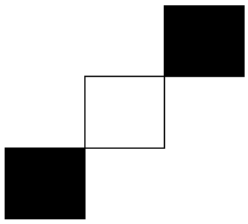
opposite color

(no cells in the line are free). You can find examples for good lines in the figure below:

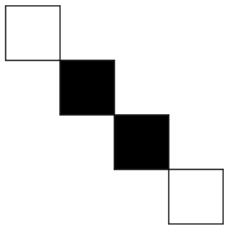
Good Line with black end points

Good Line with white end points

No good lines



Good diagonal line with black end points



Good diagonal line with white end points

Given two integers

rMove

and

cMove

and a character

color

representing the color you are playing as (white or black), return

true

if changing cell

(rMove, cMove)

to color

color

is a

legal

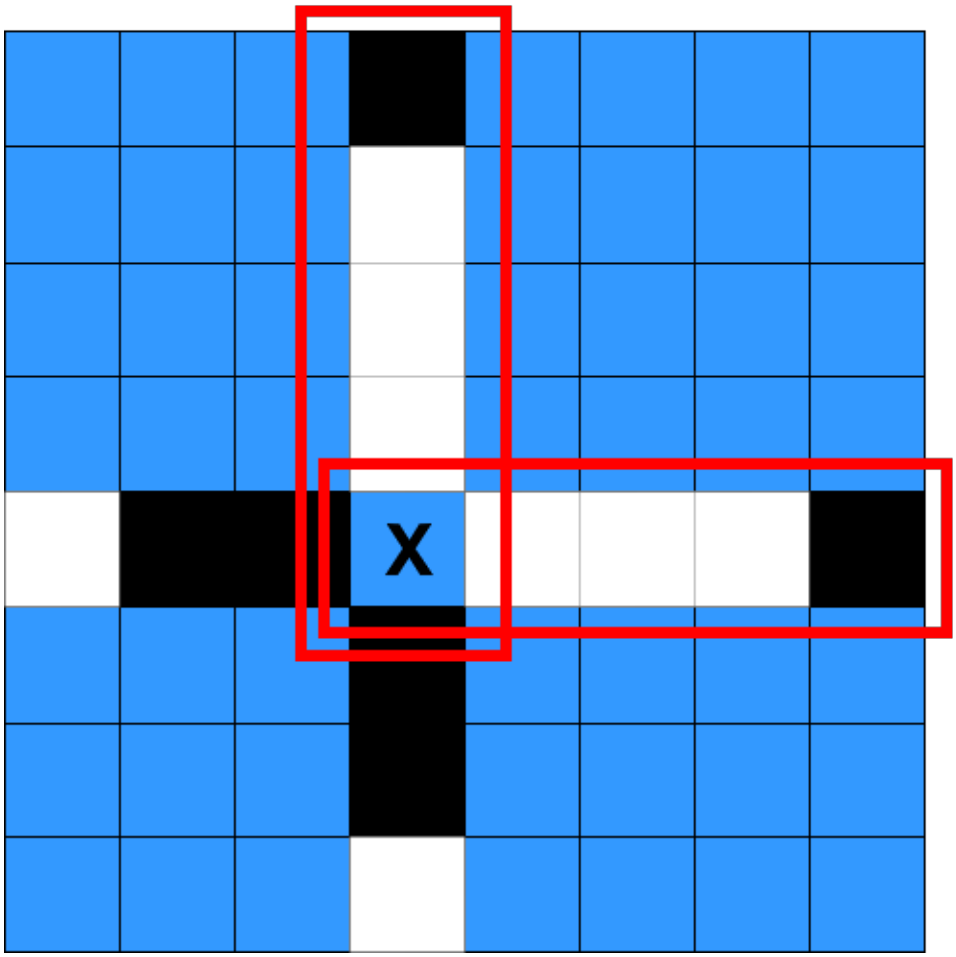
move, or

false

if it is not legal

.

Example 1:



Input:

board = [[".",".",".","B",".",".",".",".",".",".",],[".",".",".",".","W",".",".",".",".",".",],[".",".",".",".","W",".",".",".",".",".",],[".",".",".",".","W",".",".",".",".",".",],["W","B","B",".",".","W","W","W","B"],[".",".",".",".","B",".",".",".",".",".",],[".",".",".",".","B",".",".",".",".",".",],[".",".",".",".","W",".",".",".",".",".",]], rMove = 4, cMove = 3, color = "B"

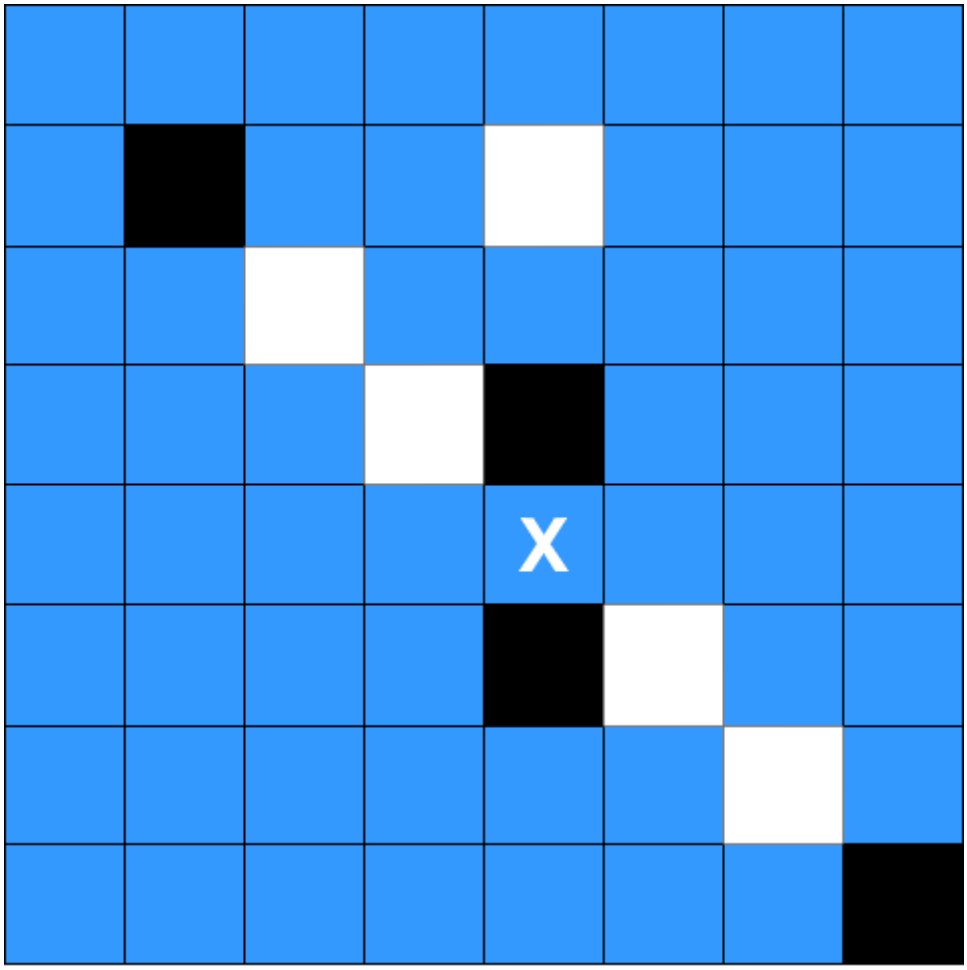
Output:

true

Explanation:

'.', 'W', and 'B' are represented by the colors blue, white, and black respectively, and cell (rMove, cMove) is marked with an 'X'. The two good lines with the chosen cell as an endpoint are annotated above with the red rectangles.

Example 2:



Input:

board = [[".",".",".",".",".",".",".",".",".",."],[".","B",".",".",".","W",".",".",".","."],[".",".",".","W",".",".",".",".",".","."],[".",".",".","W","B",".",".",".",".","."],[".",".",".",".",".",".",".",".",".","."],[".",".",".",".",".",".",".","B","W",".",".","."],[".",".",".",".",".",".",".",".",".","W","."],[".",".",".",".",".",".",".",".",".","B"]], rMove = 4, cMove = 4, color = "W"

Output:

false

Explanation:

While there are good lines with the chosen cell as a middle cell, there are no good lines with the chosen cell as an endpoint.

Constraints:

board.length == board[r].length == 8

0 <= rMove, cMove < 8

board[rMove][cMove] == '.'

color

is either

'B'

or

'W'

.

Code Snippets

C++:

```
class Solution {
public:
bool checkMove(vector<vector<char>>& board, int rMove, int cMove, char color)
{
```

```
}
};
```

Java:

```
class Solution {
public boolean checkMove(char[][] board, int rMove, int cMove, char color) {

}
}
```

Python3:

```
class Solution:
def checkMove(self, board: List[List[str]], rMove: int, cMove: int, color:
str) -> bool:
```

Python:

```
class Solution(object):
def checkMove(self, board, rMove, cMove, color):
"""
:type board: List[List[str]]
:type rMove: int
:type cMove: int
:type color: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {character[][]} board
* @param {number} rMove
* @param {number} cMove
* @param {character} color
* @return {boolean}
*/
var checkMove = function(board, rMove, cMove, color) {

};
```

TypeScript:

```
function checkMove(board: string[][], rMove: number, cMove: number, color:
string): boolean {

};
```

C#:

```
public class Solution {
public bool CheckMove(char[][] board, int rMove, int cMove, char color) {

}
}
```

C:

```
bool checkMove(char** board, int boardSize, int* boardColSize, int rMove, int
cMove, char color) {

}
```

Go:

```
func checkMove(board [][]byte, rMove int, cMove int, color byte) bool {

}
```

Kotlin:

```
class Solution {
fun checkMove(board: Array<CharArray>, rMove: Int, cMove: Int, color: Char):
Boolean {

}
}
```

Swift:

```
class Solution {
func checkMove(_ board: [[Character]], _ rMove: Int, _ cMove: Int, _ color:
Character) -> Bool {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn check_move(board: Vec<Vec<char>>, r_move: i32, c_move: i32, color:
char) -> bool {

}
}
```

Ruby:

```
# @param {Character[][]} board
# @param {Integer} r_move
# @param {Integer} c_move
# @param {Character} color
# @return {Boolean}
def check_move(board, r_move, c_move, color)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $board
* @param Integer $rMove
* @param Integer $cMove
* @param String $color
* @return Boolean
*/
function checkMove($board, $rMove, $cMove, $color) {

}
}
```

Dart:

```
class Solution {
bool checkMove(List<List<String>> board, int rMove, int cMove, String color)
{

}
}
```

Scala:

```
object Solution {
def checkMove(board: Array[Array[Char]], rMove: Int, cMove: Int, color:
Char): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec check_move(board :: [[char]], r_move :: integer, c_move :: integer,
color :: char) :: boolean
def check_move(board, r_move, c_move, color) do

end
end
```

Erlang:

```
-spec check_move(Board :: [[char()]], RMove :: integer(), CMove :: integer(),
Color :: char()) -> boolean().
check_move(Board, RMove, CMove, Color) ->
.
```

Racket:

```
(define/contract (check-move board rMove cMove color)
(-> (listof (listof char?)) exact-integer? exact-integer? char? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if Move is Legal
* 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:
bool checkMove(vector<vector<char>>& board, int rMove, int cMove, char color)
{

}
};
```

Java Solution:

```
/**
* Problem: Check if Move is Legal
* 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 boolean checkMove(char[][] board, int rMove, int cMove, char color) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if Move is Legal
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 checkMove(self, board: List[List[str]], rMove: int, cMove: int, color:
str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def checkMove(self, board, rMove, cMove, color):
"""
:type board: List[List[str]]
:type rMove: int
:type cMove: int
:type color: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Check if Move is Legal
* 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 {character[][]} board
* @param {number} rMove
* @param {number} cMove
```

```
* @param {character} color
* @return {boolean}
*/
var checkMove = function(board, rMove, cMove, color) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if Move is Legal
* 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 checkMove(board: string[][], rMove: number, cMove: number, color:
string): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if Move is Legal
* 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 bool CheckMove(char[][] board, int rMove, int cMove, char color) {

}
}
```

C Solution:

```
/*
* Problem: Check if Move is Legal
* 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
*/

bool checkMove(char** board, int boardSize, int* boardColSize, int rMove, int
cMove, char color) {

}
```

Go Solution:

```
// Problem: Check if Move is Legal
// 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 checkMove(board [][]byte, rMove int, cMove int, color byte) bool {

}
```

Kotlin Solution:

```
class Solution {
fun checkMove(board: Array<CharArray>, rMove: Int, cMove: Int, color: Char):
Boolean {

}
}
```

Swift Solution:

```
class Solution {
func checkMove(_ board: [[Character]], _ rMove: Int, _ cMove: Int, _ color:
Character) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check if Move is Legal
// 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 check_move(board: Vec<Vec<char>>, r_move: i32, c_move: i32, color:
char) -> bool {

}
}
```

Ruby Solution:

```
# @param {Character[][]} board
# @param {Integer} r_move
# @param {Integer} c_move
# @param {Character} color
# @return {Boolean}
def check_move(board, r_move, c_move, color)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $board
* @param Integer $rMove
```

```
* @param Integer $cMove
* @param String $color
* @return Boolean
*/
function checkMove($board, $rMove, $cMove, $color) {

}
}
```

Dart Solution:

```
class Solution {
bool checkMove(List<List<String>> board, int rMove, int cMove, String color)
{

}
}
```

Scala Solution:

```
object Solution {
def checkMove(board: Array[Array[Char]], rMove: Int, cMove: Int, color:
Char): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec check_move(board :: [[char]], r_move :: integer, c_move :: integer,
color :: char) :: boolean
def check_move(board, r_move, c_move, color) do

end
end
```

Erlang Solution:

```
-spec check_move(Board :: [[char()]], RMove :: integer(), CMove :: integer(),
Color :: char()) -> boolean().
```

```
check_move(Board, RMove, CMove, Color) ->
.
```

Racket Solution:

```
(define/contract (check-move board rMove cMove color)
(-> (listof (listof char?)) exact-integer? exact-integer? char? boolean?)
)
```