Problem 782: Transform to Chessboard

Problem Information

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

Problem Description

You are given an

n x n

binary grid

board

. In each move, you can swap any two rows with each other, or any two columns with each other.

Return

the minimum number of moves to transform the board into a

chessboard board

. If the task is impossible, return

-1

.

A

chessboard board

is a board where no

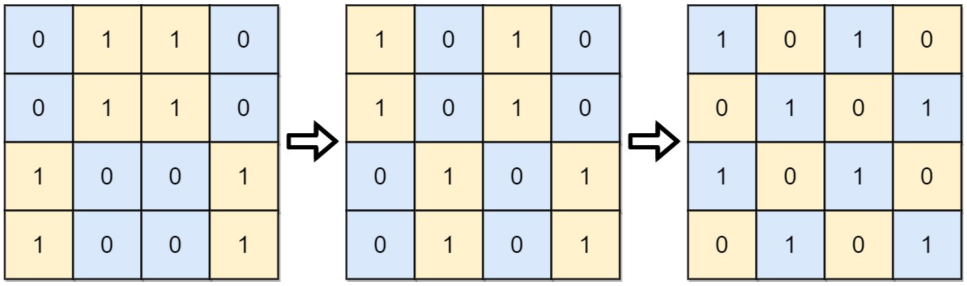
0

's and no

1

's are 4-directionally adjacent.

Example 1:



Input:

board = [[0,1,1,0],[0,1,1,0],[1,0,0,1],[1,0,0,1]]

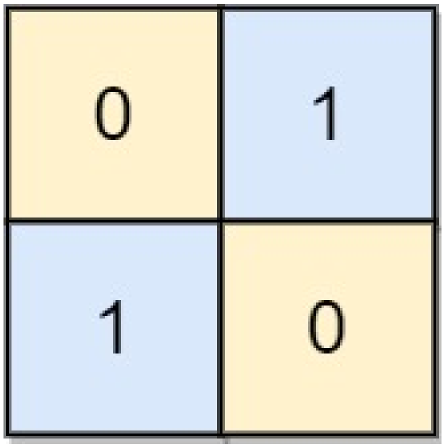
Output:

2

Explanation:

One potential sequence of moves is shown. The first move swaps the first and second column. The second move swaps the second and third row.

Example 2:



0	1
1	0

Input:

board = [[0,1],[1,0]]

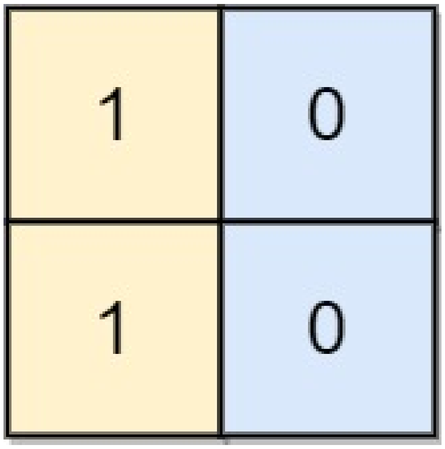
Output:

0

Explanation:

Also note that the board with 0 in the top left corner, is also a valid chessboard.

Example 3:



1	0
1	0

Input:

board = [[1,0],[1,0]]

Output:

-1

Explanation:

No matter what sequence of moves you make, you cannot end with a valid chessboard.

Constraints:

n == board.length

n == board[i].length

2 <= n <= 30

board[i][j]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int movesToChessboard(int[][] board) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function movesToChessboard(board: number[][]): number {

};
```

C#:

```
public class Solution {
public int MovesToChessboard(int[][] board) {


}
}
```

C:

```
int movesToChessboard(int** board, int boardSize, int* boardColSize) {

}
```

Go:

```
func movesToChessboard(board [][]int) int {

}
```

Kotlin:

```
class Solution {
fun movesToChessboard(board: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int movesToChessboard(List<List<int>> board) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (moves-to-chessboard board)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Transform to Chessboard
* Difficulty: Hard
* Tags: array, math
*
* 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 movesToChessboard(vector<vector<int>>& board) {

}
};
```

Java Solution:

```
/**
* Problem: Transform to Chessboard
* Difficulty: Hard
* Tags: array, math
*
* 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 movesToChessboard(int[][] board) {

}
}
```

Python3 Solution:

```
"""
Problem: Transform to Chessboard
Difficulty: Hard
Tags: array, math

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 movesToChessboard(self, board: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Transform to Chessboard
* Difficulty: Hard
* Tags: array, math
*
* 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 {number[][]} board
* @return {number}
*/
var movesToChessboard = function(board) {

};
```

TypeScript Solution:

```
/**
* Problem: Transform to Chessboard
* Difficulty: Hard
* Tags: array, math
*
* 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 movesToChessboard(board: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Transform to Chessboard
* Difficulty: Hard
* Tags: array, math
*
* 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 MovesToChessboard(int[][] board) {

}
}
```

C Solution:

```
/*
* Problem: Transform to Chessboard
* Difficulty: Hard
* Tags: array, math
*
* 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 movesToChessboard(int** board, int boardSize, int* boardColSize) {

}
```

Go Solution:

```
// Problem: Transform to Chessboard
// Difficulty: Hard
// Tags: array, math
//
// 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 movesToChessboard(board [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun movesToChessboard(board: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Transform to Chessboard
// Difficulty: Hard
// Tags: array, math
//
// 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 moves_to_chessboard(board: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[][] $board
* @return Integer
*/
function movesToChessboard($board) {

}
}
```

Dart Solution:

```
class Solution {
int movesToChessboard(List<List<int>> board) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (moves-to-chessboard board)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```