Problem 2056: Number of Valid Move Combinations On Chessboard

Problem Information

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

Problem Description

There is an

8 x 8

chessboard containing

n

pieces (rooks, queens, or bishops). You are given a string array

pieces

of length

n

, where

pieces[i]

describes the type (rook, queen, or bishop) of the

i

th

piece. In addition, you are given a 2D integer array

positions

also of length

n

, where

positions[i] = [r

i

, c

i

]

indicates that the

i

th

piece is currently at the

1-based

coordinate

(r

i

, c

i

)

on the chessboard.

When making a

move

for a piece, you choose a

destination

square that the piece will travel toward and stop on.

A rook can only travel

horizontally or vertically

from

(r, c)

to the direction of

(r+1, c)

,

(r-1, c)

,

(r, c+1)

, or

(r, c-1)

.

A queen can only travel

horizontally, vertically, or diagonally

from

(r, c)

to the direction of

(r+1, c)

,

(r-1, c)

,

(r, c+1)

,

(r, c-1)

,

(r+1, c+1)

,

(r+1, c-1)

,

(r-1, c+1)

,

(r-1, c-1)

.

A bishop can only travel

diagonally

from

(r, c)

to the direction of

(r+1, c+1)

,

(r+1, c-1)

,

(r-1, c+1)

,

(r-1, c-1)

.

You must make a

move

for every piece on the board simultaneously. A

move combination

consists of all the

moves

performed on all the given pieces. Every second, each piece will instantaneously travel

one square

towards their destination if they are not already at it. All pieces start traveling at the

0

th

second. A move combination is

invalid

if, at a given time,

two or more

pieces occupy the same square.

Return

the number of

valid

move combinations

.

Notes:

No two pieces

will start in the

same

square.

You may choose the square a piece is already on as its

destination

.

If two pieces are

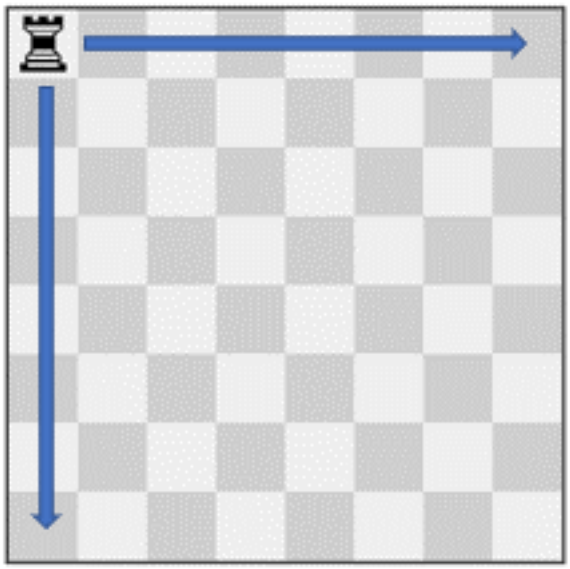
directly adjacent

to each other, it is valid for them to

move past each other

and swap positions in one second.

Example 1:



Input:

pieces = ["rook"], positions = [[1,1]]

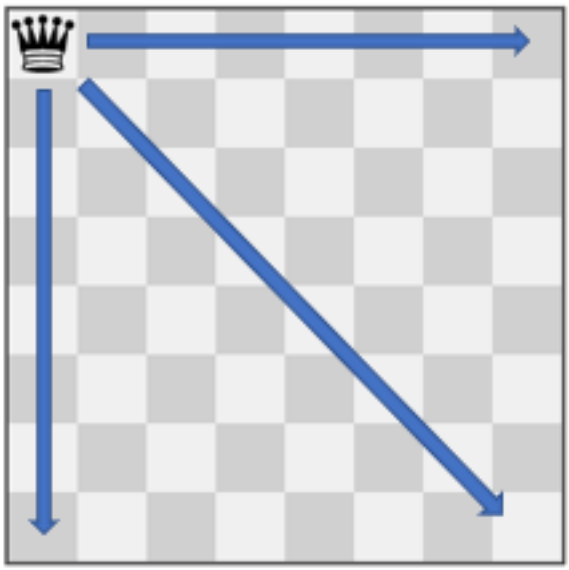
Output:

15

Explanation:

The image above shows the possible squares the piece can move to.

Example 2:



Input:

pieces = ["queen"], positions = [[1,1]]

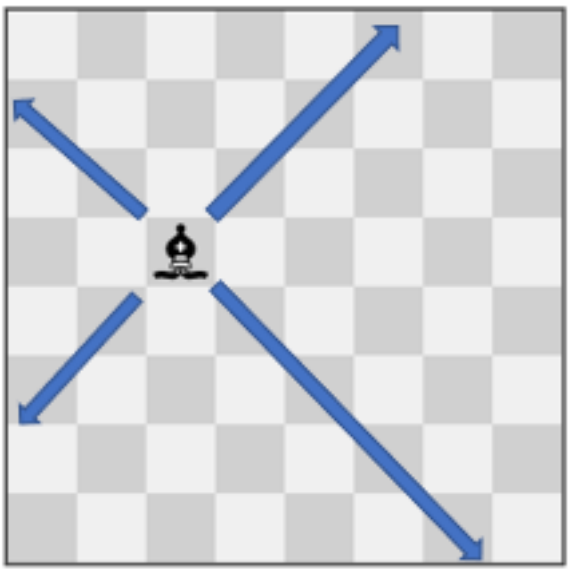
Output:

22

Explanation:

The image above shows the possible squares the piece can move to.

Example 3:



Input:

pieces = ["bishop"], positions = [[4,3]]

Output:

12

Explanation:

The image above shows the possible squares the piece can move to.

Constraints:

n == pieces.length

n == positions.length

1 <= n <= 4

pieces

only contains the strings

"rook"

,

"queen"

, and

"bishop"

.

There will be at most one queen on the chessboard.

1 <= r

i

, c

i

<= 8

Each

positions[i]

is distinct.

Code Snippets

C++:

```
class Solution {
public:
int countCombinations(vector<string>& pieces, vector<vector<int>>& positions)
{

}
};
```

Java:

```
class Solution {
public int countCombinations(String[] pieces, int[][] positions) {

}
}
```

Python3:

```
class Solution:
def countCombinations(self, pieces: List[str], positions: List[List[int]]) ->
int:
```

Python:

```
class Solution(object):
def countCombinations(self, pieces, positions):
"""
:type pieces: List[str]
:type positions: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} pieces
* @param {number[][]} positions
* @return {number}
*/
var countCombinations = function(pieces, positions) {

};
```

TypeScript:

```
function countCombinations(pieces: string[], positions: number[][]): number {

};
```

C#:

```
public class Solution {
public int CountCombinations(string[] pieces, int[][] positions) {

}
}
```

C:

```
int countCombinations(char** pieces, int piecesSize, int** positions, int
positionsSize, int* positionsColSize) {

}
```

Go:

```
func countCombinations(pieces []string, positions [][]int) int {

}
```

Kotlin:

```
class Solution {
fun countCombinations(pieces: Array<String>, positions: Array<IntArray>): Int
{

}
}
```

Swift:

```
class Solution {
func countCombinations(_ pieces: [String], _ positions: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_combinations(pieces: Vec<String>, positions: Vec<Vec<i32>>) ->
i32 {

}
```

```
}
```

Ruby:

```
# @param {String[]} pieces
# @param {Integer[][]} positions
# @return {Integer}
def count_combinations(pieces, positions)

end
```

PHP:

```
class Solution {

/**
* @param String[] $pieces
* @param Integer[][] $positions
* @return Integer
*/
function countCombinations($pieces, $positions) {

}
}
```

Dart:

```
class Solution {
int countCombinations(List<String> pieces, List<List<int>> positions) {

}
}
```

Scala:

```
object Solution {
def countCombinations(pieces: Array[String], positions: Array[Array[Int]]):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_combinations(pieces :: [String.t], positions :: [[integer]]) ::
integer
def count_combinations(pieces, positions) do

end
end
```

Erlang:

```
-spec count_combinations(Pieces :: [unicode:unicode_binary()], Positions ::
[[integer()]]) -> integer().
count_combinations(Pieces, Positions) ->
.
```

Racket:

```
(define/contract (count-combinations pieces positions)
(-> (listof string?) (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Valid Move Combinations On Chessboard
* Difficulty: Hard
* Tags: array, string
*
* 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 countCombinations(vector<string>& pieces, vector<vector<int>>& positions)
{
```

```
}
};
```

Java Solution:

```
/**
* Problem: Number of Valid Move Combinations On Chessboard
* Difficulty: Hard
* Tags: array, string
*
* 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 countCombinations(String[] pieces, int[][] positions) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Valid Move Combinations On Chessboard
Difficulty: Hard
Tags: array, string

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 countCombinations(self, pieces: List[str], positions: List[List[int]]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countCombinations(self, pieces, positions):
"""
:type pieces: List[str]
:type positions: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Valid Move Combinations On Chessboard
* Difficulty: Hard
* Tags: array, string
*
* 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 {string[]} pieces
* @param {number[][]} positions
* @return {number}
*/
var countCombinations = function(pieces, positions) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Valid Move Combinations On Chessboard
* Difficulty: Hard
* Tags: array, string
*
* 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 countCombinations(pieces: string[], positions: number[][]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Number of Valid Move Combinations On Chessboard
* Difficulty: Hard
* Tags: array, string
*
* 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 CountCombinations(string[] pieces, int[][] positions) {

}
}
```

C Solution:

```
/*
* Problem: Number of Valid Move Combinations On Chessboard
* Difficulty: Hard
* Tags: array, string
*
* 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 countCombinations(char** pieces, int piecesSize, int** positions, int
positionsSize, int* positionsColSize) {

}
```

Go Solution:

```
// Problem: Number of Valid Move Combinations On Chessboard
// Difficulty: Hard
// Tags: array, string
//
// 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 countCombinations(pieces []string, positions [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countCombinations(pieces: Array<String>, positions: Array<IntArray>): Int
{

}
}
```

Swift Solution:

```
class Solution {
func countCombinations(_ pieces: [String], _ positions: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Valid Move Combinations On Chessboard
// Difficulty: Hard
// Tags: array, string
//
// 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 count_combinations(pieces: Vec<String>, positions: Vec<Vec<i32>>) ->
i32 {
```

```
}
}
```

Ruby Solution:

```
# @param {String[]} pieces
# @param {Integer[][]} positions
# @return {Integer}
def count_combinations(pieces, positions)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $pieces
* @param Integer[][] $positions
* @return Integer
*/
function countCombinations($pieces, $positions) {

}
}
```

Dart Solution:

```
class Solution {
int countCombinations(List<String> pieces, List<List<int>> positions) {

}
}
```

Scala Solution:

```
object Solution {
def countCombinations(pieces: Array[String], positions: Array[Array[Int]]):
Int = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_combinations(pieces :: [String.t], positions :: [[integer]]) ::
integer
def count_combinations(pieces, positions) do

end
end
```

Erlang Solution:

```
-spec count_combinations(Pieces :: [unicode:unicode_binary()], Positions ::
[[integer()]]) -> integer().
count_combinations(Pieces, Positions) ->
.
```

Racket Solution:

```
(define/contract (count-combinations pieces positions)
(-> (listof string?) (listof (listof exact-integer?)) exact-integer?)
)
```