Problem 3001: Minimum Moves to Capture The Queen

Problem Information

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

Problem Description

There is a

1-indexed

8 x 8

chessboard containing

3

pieces.

You are given

6

integers

a

,

b

,

c

,

d

,

e

, and

f

where:

(a, b)

denotes the position of the white rook.

(c, d)

denotes the position of the white bishop.

(e, f)

denotes the position of the black queen.

Given that you can only move the white pieces, return

the

minimum

number of moves required to capture the black queen

.

Note

that:

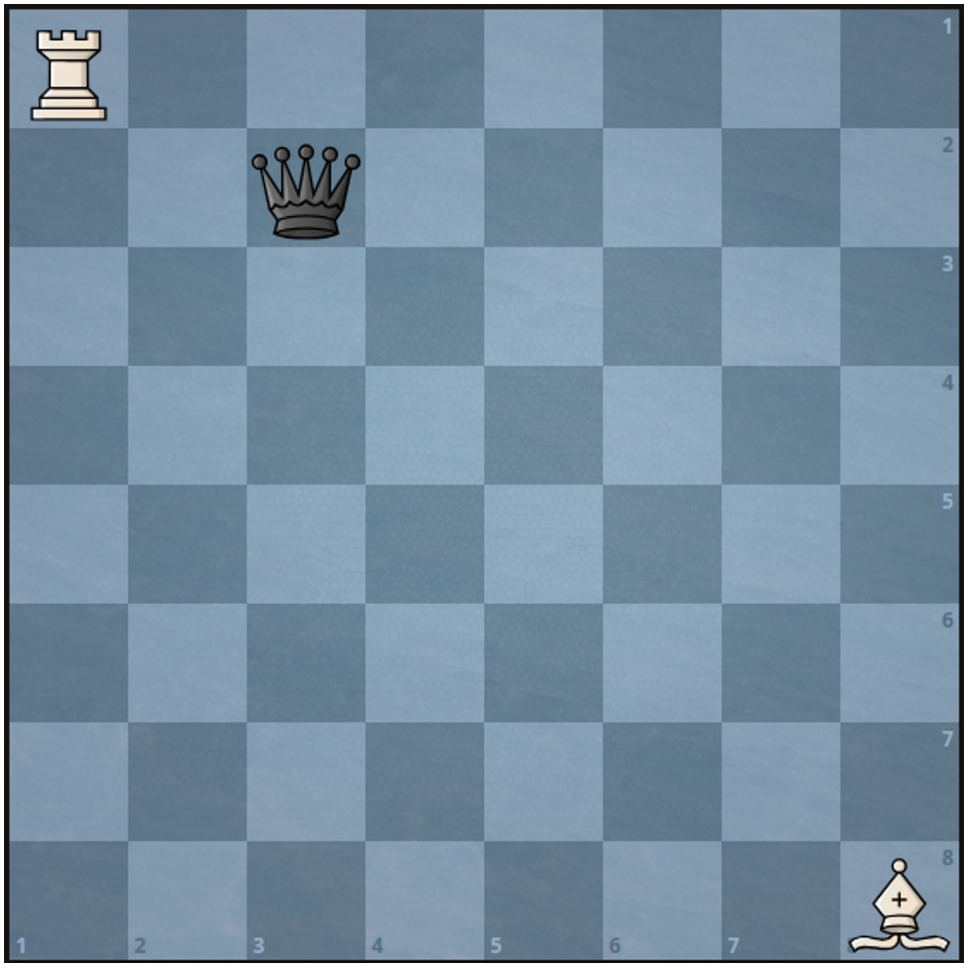
Rooks can move any number of squares either vertically or horizontally, but cannot jump over other pieces.

Bishops can move any number of squares diagonally, but cannot jump over other pieces.

A rook or a bishop can capture the queen if it is located in a square that they can move to.

The queen does not move.

Example 1:



Input:

a = 1, b = 1, c = 8, d = 8, e = 2, f = 3

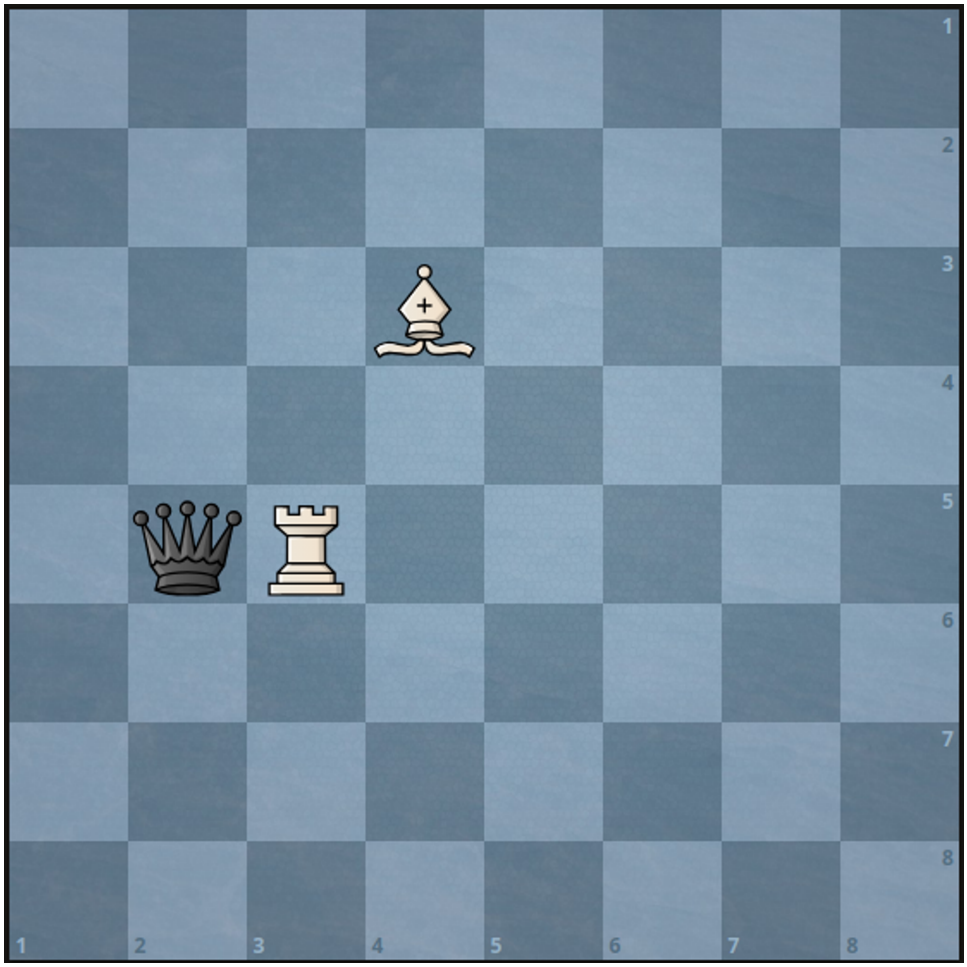
Output:

2

Explanation:

We can capture the black queen in two moves by moving the white rook to (1, 3) then to (2, 3). It is impossible to capture the black queen in less than two moves since it is not being attacked by any of the pieces at the beginning.

Example 2:



Input:

a = 5, b = 3, c = 3, d = 4, e = 5, f = 2

Output:

1

Explanation:

We can capture the black queen in a single move by doing one of the following: - Move the white rook to (5, 2). - Move the white bishop to (5, 2).

Constraints:

1 <= a, b, c, d, e, f <= 8

No two pieces are on the same square.

Code Snippets

C++:

```
class Solution {
public:
int minMovesToCaptureTheQueen(int a, int b, int c, int d, int e, int f) {

}
};
```

Java:

```
class Solution {
public int minMovesToCaptureTheQueen(int a, int b, int c, int d, int e, int
f) {

}
}
```

Python3:

```
class Solution:
def minMovesToCaptureTheQueen(self, a: int, b: int, c: int, d: int, e: int,
f: int) -> int:
```

Python:

```
class Solution(object):
def minMovesToCaptureTheQueen(self, a, b, c, d, e, f):
"""
:type a: int
:type b: int
:type c: int
:type d: int
:type e: int
:type f: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} a
* @param {number} b
* @param {number} c
* @param {number} d
* @param {number} e
* @param {number} f
* @return {number}
*/
var minMovesToCaptureTheQueen = function(a, b, c, d, e, f) {

};
```

TypeScript:

```
function minMovesToCaptureTheQueen(a: number, b: number, c: number, d:
number, e: number, f: number): number {

};
```

C#:

```
public class Solution {
public int MinMovesToCaptureTheQueen(int a, int b, int c, int d, int e, int
f) {

}
}
```

C:

```
int minMovesToCaptureTheQueen(int a, int b, int c, int d, int e, int f) {

}
```

Go:

```
func minMovesToCaptureTheQueen(a int, b int, c int, d int, e int, f int) int
{

}
```

Kotlin:

```
class Solution {
fun minMovesToCaptureTheQueen(a: Int, b: Int, c: Int, d: Int, e: Int, f:
Int): Int {

}
}
```

Swift:

```
class Solution {
func minMovesToCaptureTheQueen(_ a: Int, _ b: Int, _ c: Int, _ d: Int, _ e:
Int, _ f: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_moves_to_capture_the_queen(a: i32, b: i32, c: i32, d: i32, e: i32,
f: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} a
# @param {Integer} b
# @param {Integer} c
# @param {Integer} d
# @param {Integer} e
# @param {Integer} f
# @return {Integer}
def min_moves_to_capture_the_queen(a, b, c, d, e, f)

end
```

PHP:

```
class Solution {

/**
* @param Integer $a
* @param Integer $b
* @param Integer $c
* @param Integer $d
* @param Integer $e
* @param Integer $f
* @return Integer
*/
function minMovesToCaptureTheQueen($a, $b, $c, $d, $e, $f) {

}
}
```

Dart:

```
class Solution {
int minMovesToCaptureTheQueen(int a, int b, int c, int d, int e, int f) {

}
}
```

Scala:

```
object Solution {
def minMovesToCaptureTheQueen(a: Int, b: Int, c: Int, d: Int, e: Int, f:
Int): Int = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec min_moves_to_capture_the_queen(a :: integer, b :: integer, c ::
integer, d :: integer, e :: integer, f :: integer) :: integer
def min_moves_to_capture_the_queen(a, b, c, d, e, f) do

end
end
```

Erlang:

```
-spec min_moves_to_capture_the_queen(A :: integer(), B :: integer(), C ::
integer(), D :: integer(), E :: integer(), F :: integer()) -> integer().
min_moves_to_capture_the_queen(A, B, C, D, E, F) ->
.
```

Racket:

```
(define/contract (min-moves-to-capture-the-queen a b c d e f)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Moves to Capture The Queen
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int minMovesToCaptureTheQueen(int a, int b, int c, int d, int e, int f) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Moves to Capture The Queen
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int minMovesToCaptureTheQueen(int a, int b, int c, int d, int e, int
f) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Moves to Capture The Queen
Difficulty: Medium
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def minMovesToCaptureTheQueen(self, a: int, b: int, c: int, d: int, e: int,
```

```
f: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minMovesToCaptureTheQueen(self, a, b, c, d, e, f):
"""
:type a: int
:type b: int
:type c: int
:type d: int
:type e: int
:type f: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Moves to Capture The Queen
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} a
* @param {number} b
* @param {number} c
* @param {number} d
* @param {number} e
* @param {number} f
* @return {number}
*/
var minMovesToCaptureTheQueen = function(a, b, c, d, e, f) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Moves to Capture The Queen
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function minMovesToCaptureTheQueen(a: number, b: number, c: number, d:
number, e: number, f: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Moves to Capture The Queen
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MinMovesToCaptureTheQueen(int a, int b, int c, int d, int e, int
f) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Moves to Capture The Queen
* Difficulty: Medium
* Tags: math
```

```
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int minMovesToCaptureTheQueen(int a, int b, int c, int d, int e, int f) {

}
```

Go Solution:

```
// Problem: Minimum Moves to Capture The Queen
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func minMovesToCaptureTheQueen(a int, b int, c int, d int, e int, f int) int
{

}
```

Kotlin Solution:

```
class Solution {
fun minMovesToCaptureTheQueen(a: Int, b: Int, c: Int, d: Int, e: Int, f:
Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minMovesToCaptureTheQueen(_ a: Int, _ b: Int, _ c: Int, _ d: Int, _ e:
Int, _ f: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Minimum Moves to Capture The Queen
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn min_moves_to_capture_the_queen(a: i32, b: i32, c: i32, d: i32, e: i32,
f: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} a
# @param {Integer} b
# @param {Integer} c
# @param {Integer} d
# @param {Integer} e
# @param {Integer} f
# @return {Integer}
def min_moves_to_capture_the_queen(a, b, c, d, e, f)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $a
* @param Integer $b
* @param Integer $c
* @param Integer $d
```

```
* @param Integer $e
* @param Integer $f
* @return Integer
*/
function minMovesToCaptureTheQueen($a, $b, $c, $d, $e, $f) {

}
}
```

Dart Solution:

```
class Solution {
int minMovesToCaptureTheQueen(int a, int b, int c, int d, int e, int f) {

}
}
```

Scala Solution:

```
object Solution {
def minMovesToCaptureTheQueen(a: Int, b: Int, c: Int, d: Int, e: Int, f:
Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_moves_to_capture_the_queen(a :: integer, b :: integer, c ::
integer, d :: integer, e :: integer, f :: integer) :: integer
def min_moves_to_capture_the_queen(a, b, c, d, e, f) do

end
end
```

Erlang Solution:

```
-spec min_moves_to_capture_the_queen(A :: integer(), B :: integer(), C ::
integer(), D :: integer(), E :: integer(), F :: integer()) -> integer().
min_moves_to_capture_the_queen(A, B, C, D, E, F) ->
```

```
.
```

Racket Solution:

```
(define/contract (min-moves-to-capture-the-queen a b c d e f)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer? exact-integer? exact-integer?)
)
```