Problem 1476: Subrectangle Queries

Problem Information

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

Problem Description

Implement the class

SubrectangleQueries

which receives a

rows x cols

rectangle as a matrix of integers in the constructor and supports two methods:

1.

updateSubrectangle(int row1, int col1, int row2, int col2, int newValue)

Updates all values with

newValue

in the subrectangle whose upper left coordinate is

(row1,col1)

and bottom right coordinate is

(row2,col2)

.

2.

getValue(int row, int col)

Returns the current value of the coordinate

(row,col)

from the rectangle.

Example 1:

Input

["SubrectangleQueries","getValue","updateSubrectangle","getValue","getValue","updateSubrectangle","getValue","getValue"]
[[[[1,2,1],[4,3,4],[3,2,1],[1,1,1]]],[0,2],[0,0,3,2,5],[0,2],[3,1],[3,0,3,2,10],[3,1],[0,2]]

Output

[null,1,null,5,5,null,10,5]

Explanation

SubrectangleQueries subrectangleQueries = new SubrectangleQueries([[1,2,1],[4,3,4],[3,2,1],[1,1,1]]); // The initial rectangle (4x3) looks like: // 1 2 1 // 4 3 4 // 3 2 1 // 1 1 1 subrectangleQueries.getValue(0, 2); // return 1 subrectangleQueries.updateSubrectangle(0, 0, 3, 2, 5); // After this update the rectangle looks like: // 5 5 5 // 5 5 5 // 5 5 5 // 5 5 5 subrectangleQueries.getValue(0, 2); // return 5 subrectangleQueries.getValue(3, 1); // return 5 subrectangleQueries.updateSubrectangle(3, 0, 3, 2, 10); // After this update the rectangle looks like: // 5 5 5 // 5 5 5 // 5 5 5 // 10 10 10 subrectangleQueries.getValue(3, 1); // return 10 subrectangleQueries.getValue(0, 2); // return 5

Example 2:

Input

["SubrectangleQueries","getValue","updateSubrectangle","getValue","getValue","updateSubrectangle","getValue"] [[[[1,1,1],[2,2,2],[3,3,3]]],[0,0],[0,0,2,2,100],[0,0],[2,2],[1,1,2,2,20],[2,2]]

Output

[null,1,null,100,100,null,20]

Explanation

SubrectangleQueries subrectangleQueries = new SubrectangleQueries([[1,1,1],[2,2,2],[3,3,3]]); subrectangleQueries.getValue(0, 0); // return 1 subrectangleQueries.updateSubrectangle(0, 0, 2, 2, 100); subrectangleQueries.getValue(0, 0); // return 100 subrectangleQueries.getValue(2, 2); // return 100 subrectangleQueries.updateSubrectangle(1, 1, 2, 2, 20); subrectangleQueries.getValue(2, 2); // return 20

Constraints:

There will be at most

500

operations considering both methods:

updateSubrectangle

and

getValue

.

1 <= rows, cols <= 100

rows == rectangle.length

cols == rectangle[i].length

0 <= row1 <= row2 < rows

0 <= col1 <= col2 < cols

1 <= newValue, rectangle[i][j] <= 10^9

0 <= row < rows

0 <= col < cols

Code Snippets

C++:

```
class SubrectangleQueries {
public:
SubrectangleQueries(vector<vector<int>>& rectangle) {

}

void updateSubrectangle(int row1, int col1, int row2, int col2, int newValue)
{

}

int getValue(int row, int col) {

}
};

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* SubrectangleQueries* obj = new SubrectangleQueries(rectangle);
* obj->updateSubrectangle(row1,col1,row2,col2,newValue);
* int param_2 = obj->getValue(row,col);
*/
```

Java:

```
class SubrectangleQueries {

public SubrectangleQueries(int[][] rectangle) {
```

```
}

public void updateSubrectangle(int row1, int col1, int row2, int col2, int
newValue) {

}

public int getValue(int row, int col) {

}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* SubrectangleQueries obj = new SubrectangleQueries(rectangle);
* obj.updateSubrectangle(row1,col1,row2,col2,newValue);
* int param_2 = obj.getValue(row,col);
*/
```

Python3:

```
class SubrectangleQueries:

def __init__(self, rectangle: List[List[int]]):


def updateSubrectangle(self, row1: int, col1: int, row2: int, col2: int,
newValue: int) -> None:


def getValue(self, row: int, col: int) -> int:




# Your SubrectangleQueries object will be instantiated and called as such:
# obj = SubrectangleQueries(rectangle)
# obj.updateSubrectangle(row1,col1,row2,col2,newValue)
# param_2 = obj.getValue(row,col)
```

Python:

```
class SubrectangleQueries(object):

def __init__(self, rectangle):
"""
:type rectangle: List[List[int]]
"""


def updateSubrectangle(self, row1, col1, row2, col2, newValue):
"""
:type row1: int
:type col1: int
:type row2: int
:type col2: int
:type newValue: int
:rtype: None
"""


def getValue(self, row, col):
"""
:type row: int
:type col: int
:rtype: int
"""



# Your SubrectangleQueries object will be instantiated and called as such:
# obj = SubrectangleQueries(rectangle)
# obj.updateSubrectangle(row1,col1,row2,col2,newValue)
# param_2 = obj.getValue(row,col)
```

JavaScript:

```
/**
* @param {number[][]} rectangle
*/
var SubrectangleQueries = function(rectangle) {

};

/**
```

```
* @param {number} row1
* @param {number} col1
* @param {number} row2
* @param {number} col2
* @param {number} newValue
* @return {void}
*/
SubrectangleQueries.prototype.updateSubrectangle = function(row1, col1, row2,
col2, newValue) {

};

/**
* @param {number} row
* @param {number} col
* @return {number}
*/
SubrectangleQueries.prototype.getValue = function(row, col) {

};

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* var obj = new SubrectangleQueries(rectangle)
* obj.updateSubrectangle(row1,col1,row2,col2,newValue)
* var param_2 = obj.getValue(row,col)
*/
```

TypeScript:

```
class SubrectangleQueries {
constructor(rectangle: number[][]) {

}

updateSubrectangle(row1: number, col1: number, row2: number, col2: number,
newValue: number): void {

}

getValue(row: number, col: number): number {
```

```
}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* var obj = new SubrectangleQueries(rectangle)
* obj.updateSubrectangle(row1,col1,row2,col2,newValue)
* var param_2 = obj.getValue(row,col)
*/
```

C#:

```
public class SubrectangleQueries {

public SubrectangleQueries(int[][] rectangle) {

}

public void UpdateSubrectangle(int row1, int col1, int row2, int col2, int
newValue) {

}

public int GetValue(int row, int col) {

}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* SubrectangleQueries obj = new SubrectangleQueries(rectangle);
* obj.UpdateSubrectangle(row1,col1,row2,col2,newValue);
* int param_2 = obj.GetValue(row,col);
*/
```

C:

```
typedef struct {
```

```
} SubrectangleQueries;


SubrectangleQueries* subrectangleQueriesCreate(int** rectangle, int
rectangleSize, int* rectangleColSize) {

}

void subrectangleQueriesUpdateSubrectangle(SubrectangleQueries* obj, int
row1, int col1, int row2, int col2, int newValue) {

}

int subrectangleQueriesGetValue(SubrectangleQueries* obj, int row, int col) {

}

void subrectangleQueriesFree(SubrectangleQueries* obj) {

}

/**
* Your SubrectangleQueries struct will be instantiated and called as such:
* SubrectangleQueries* obj = subrectangleQueriesCreate(rectangle,
rectangleSize, rectangleColSize);
* subrectangleQueriesUpdateSubrectangle(obj, row1, col1, row2, col2,
newValue);

* int param_2 = subrectangleQueriesGetValue(obj, row, col);

* subrectangleQueriesFree(obj);
*/
```

Go:

```
type SubrectangleQueries struct {

}


func Constructor(rectangle [][]int) SubrectangleQueries {
```

```
}


func (this *SubrectangleQueries) UpdateSubrectangle(row1 int, col1 int, row2
int, col2 int, newValue int) {

}


func (this *SubrectangleQueries) GetValue(row int, col int) int {

}


/**
* Your SubrectangleQueries object will be instantiated and called as such:
* obj := Constructor(rectangle);
* obj.UpdateSubrectangle(row1,col1,row2,col2,newValue);
* param_2 := obj.GetValue(row,col);
*/
```

Kotlin:

```
class SubrectangleQueries(rectangle: Array<IntArray>) {

fun updateSubrectangle(row1: Int, col1: Int, row2: Int, col2: Int, newValue:
Int) {

}

fun getValue(row: Int, col: Int): Int {

}

}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* var obj = SubrectangleQueries(rectangle)
* obj.updateSubrectangle(row1,col1,row2,col2,newValue)
* var param_2 = obj.getValue(row,col)
*/
```

Swift:

```
class SubrectangleQueries {

init(_ rectangle: [[Int]]) {

}

func updateSubrectangle(_ row1: Int, _ col1: Int, _ row2: Int, _ col2: Int, _
newValue: Int) {

}

func getValue(_ row: Int, _ col: Int) -> Int {

}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* let obj = SubrectangleQueries(rectangle)
* obj.updateSubrectangle(row1, col1, row2, col2, newValue)
* let ret_2: Int = obj.getValue(row, col)
*/
```

Rust:

```
struct SubrectangleQueries {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl SubrectangleQueries {

fn new(rectangle: Vec<Vec<i32>>) -> Self {

}
```

```
fn update_subrectangle(&self, row1: i32, col1: i32, row2: i32, col2: i32,
new_value: i32) {

}

fn get_value(&self, row: i32, col: i32) -> i32 {

}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* let obj = SubrectangleQueries::new(rectangle);
* obj.update_subrectangle(row1, col1, row2, col2, newValue);
* let ret_2: i32 = obj.get_value(row, col);
*/
```

Ruby:

```
class SubrectangleQueries

=begin
:type rectangle: Integer[][]
=end
def initialize(rectangle)

end


=begin
:type row1: Integer
:type col1: Integer
:type row2: Integer
:type col2: Integer
:type new_value: Integer
:rtype: Void
=end
def update_subrectangle(row1, col1, row2, col2, new_value)

end
```

```
=begin
:type row: Integer
:type col: Integer
:rtype: Integer
=end
def get_value(row, col)

end



end

# Your SubrectangleQueries object will be instantiated and called as such:
# obj = SubrectangleQueries.new(rectangle)
# obj.update_subrectangle(row1, col1, row2, col2, new_value)
# param_2 = obj.get_value(row, col)
```

PHP:

```
class SubrectangleQueries {
/**
* @param Integer[][] $rectangle
*/
function __construct($rectangle) {

}

/**
* @param Integer $row1
* @param Integer $col1
* @param Integer $row2
* @param Integer $col2
* @param Integer $newValue
* @return NULL
*/
function updateSubrectangle($row1, $col1, $row2, $col2, $newValue) {

}

/**
* @param Integer $row
```

```
* @param Integer $col
* @return Integer
*/
function getValue($row, $col) {

}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* $obj = SubrectangleQueries($rectangle);
* $obj->updateSubrectangle($row1, $col1, $row2, $col2, $newValue);
* $ret_2 = $obj->getValue($row, $col);
*/
```

Scala:

```
class SubrectangleQueries(_rectangle: Array[Array[Int]]) {

def updateSubrectangle(row1: Int, col1: Int, row2: Int, col2: Int, newValue:
Int) {

}

def getValue(row: Int, col: Int): Int = {

}

}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* var obj = new SubrectangleQueries(rectangle)
* obj.updateSubrectangle(row1,col1,row2,col2,newValue)
* var param_2 = obj.getValue(row,col)
*/
```

Racket:

```
(define subrectangle-queries%
(class object%
(super-new)
```

```
; rectangle : (listof (listof exact-integer?))
(init-field
rectangle)

; update-subrectangle : exact-integer? exact-integer? exact-integer?
exact-integer? exact-integer? -> void?
(define/public (update-subrectangle row1 col1 row2 col2 newValue)

)
; get-value : exact-integer? exact-integer? -> exact-integer?
(define/public (get-value row col)

)))

;; Your subrectangle-queries% object will be instantiated and called as such:
;; (define obj (new subrectangle-queries% [rectangle rectangle]))
;; (send obj update-subrectangle row1 col1 row2 col2 new-value)
;; (define param_2 (send obj get-value row col))
```

Solutions

C++ Solution:

```
/*
* Problem: Subrectangle Queries
* Difficulty: Medium
* 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 SubrectangleQueries {
public:
SubrectangleQueries(vector<vector<int>>& rectangle) {

}
```

```
void updateSubrectangle(int row1, int col1, int row2, int col2, int newValue)
{

}

int getValue(int row, int col) {

}
};

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* SubrectangleQueries* obj = new SubrectangleQueries(rectangle);
* obj->updateSubrectangle(row1,col1,row2,col2,newValue);
* int param_2 = obj->getValue(row,col);
*/
```

Java Solution:

```
/**
* Problem: Subrectangle Queries
* Difficulty: Medium
* 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 SubrectangleQueries {

public SubrectangleQueries(int[][] rectangle) {

}

public void updateSubrectangle(int row1, int col1, int row2, int col2, int
newValue) {

}

public int getValue(int row, int col) {
```

```
}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* SubrectangleQueries obj = new SubrectangleQueries(rectangle);
* obj.updateSubrectangle(row1,col1,row2,col2,newValue);
* int param_2 = obj.getValue(row,col);
*/
```

Python3 Solution:

```
"""
Problem: Subrectangle Queries
Difficulty: Medium
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 SubrectangleQueries:

def __init__(self, rectangle: List[List[int]]):


def updateSubrectangle(self, row1: int, col1: int, row2: int, col2: int,
newValue: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class SubrectangleQueries(object):

def __init__(self, rectangle):
"""
:type rectangle: List[List[int]]
"""
```

```
def updateSubrectangle(self, row1, col1, row2, col2, newValue):
"""
:type row1: int
:type col1: int
:type row2: int
:type col2: int
:type newValue: int
:rtype: None
"""


def getValue(self, row, col):
"""
:type row: int
:type col: int
:rtype: int
"""



# Your SubrectangleQueries object will be instantiated and called as such:
# obj = SubrectangleQueries(rectangle)
# obj.updateSubrectangle(row1,col1,row2,col2,newValue)
# param_2 = obj.getValue(row,col)
```

JavaScript Solution:

```
/**
* Problem: Subrectangle Queries
* Difficulty: Medium
* 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 {number[][]} rectangle
```

```
*/
var SubrectangleQueries = function(rectangle) {

};

/**
* @param {number} row1
* @param {number} col1
* @param {number} row2
* @param {number} col2
* @param {number} newValue
* @return {void}
*/
SubrectangleQueries.prototype.updateSubrectangle = function(row1, col1, row2,
col2, newValue) {

};

/**
* @param {number} row
* @param {number} col
* @return {number}
*/
SubrectangleQueries.prototype.getValue = function(row, col) {

};

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* var obj = new SubrectangleQueries(rectangle)
* obj.updateSubrectangle(row1,col1,row2,col2,newValue)
* var param_2 = obj.getValue(row,col)
*/
```

TypeScript Solution:

```
/**
* Problem: Subrectangle Queries
* Difficulty: Medium
* 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 SubrectangleQueries {
constructor(rectangle: number[][]) {

}

updateSubrectangle(row1: number, col1: number, row2: number, col2: number,
newValue: number): void {

}

getValue(row: number, col: number): number {

}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* var obj = new SubrectangleQueries(rectangle)
* obj.updateSubrectangle(row1,col1,row2,col2,newValue)
* var param_2 = obj.getValue(row,col)
*/
```

C# Solution:

```
/*
* Problem: Subrectangle Queries
* Difficulty: Medium
* 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 SubrectangleQueries {
```

```
public SubrectangleQueries(int[][] rectangle) {

}

public void UpdateSubrectangle(int row1, int col1, int row2, int col2, int
newValue) {

}

public int GetValue(int row, int col) {

}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* SubrectangleQueries obj = new SubrectangleQueries(rectangle);
* obj.UpdateSubrectangle(row1,col1,row2,col2,newValue);
* int param_2 = obj.GetValue(row,col);
*/
```

C Solution:

```
/*
* Problem: Subrectangle Queries
* Difficulty: Medium
* 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
*/




typedef struct {

} SubrectangleQueries;
```

```
SubrectangleQueries* subrectangleQueriesCreate(int** rectangle, int
rectangleSize, int* rectangleColSize) {

}

void subrectangleQueriesUpdateSubrectangle(SubrectangleQueries* obj, int
row1, int col1, int row2, int col2, int newValue) {

}

int subrectangleQueriesGetValue(SubrectangleQueries* obj, int row, int col) {

}

void subrectangleQueriesFree(SubrectangleQueries* obj) {

}

/**
* Your SubrectangleQueries struct will be instantiated and called as such:
* SubrectangleQueries* obj = subrectangleQueriesCreate(rectangle,
rectangleSize, rectangleColSize);
* subrectangleQueriesUpdateSubrectangle(obj, row1, col1, row2, col2,
newValue);

* int param_2 = subrectangleQueriesGetValue(obj, row, col);

* subrectangleQueriesFree(obj);
*/
```

Go Solution:

```
// Problem: Subrectangle Queries
// Difficulty: Medium
// 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

type SubrectangleQueries struct {
```

```
}


func Constructor(rectangle [][]int) SubrectangleQueries {

}


func (this *SubrectangleQueries) UpdateSubrectangle(row1 int, col1 int, row2
int, col2 int, newValue int) {

}


func (this *SubrectangleQueries) GetValue(row int, col int) int {

}


/**
* Your SubrectangleQueries object will be instantiated and called as such:
* obj := Constructor(rectangle);
* obj.UpdateSubrectangle(row1,col1,row2,col2,newValue);
* param_2 := obj.GetValue(row,col);
*/
```

Kotlin Solution:

```
class SubrectangleQueries(rectangle: Array<IntArray>) {

fun updateSubrectangle(row1: Int, col1: Int, row2: Int, col2: Int, newValue:
Int) {

}

fun getValue(row: Int, col: Int): Int {

}

}
```

```
/**
* Your SubrectangleQueries object will be instantiated and called as such:
* var obj = SubrectangleQueries(rectangle)
* obj.updateSubrectangle(row1,col1,row2,col2,newValue)
* var param_2 = obj.getValue(row,col)
*/
```

Swift Solution:

```
class SubrectangleQueries {

init(_ rectangle: [[Int]]) {

}

func updateSubrectangle(_ row1: Int, _ col1: Int, _ row2: Int, _ col2: Int, _
newValue: Int) {

}

func getValue(_ row: Int, _ col: Int) -> Int {

}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* let obj = SubrectangleQueries(rectangle)
* obj.updateSubrectangle(row1, col1, row2, col2, newValue)
* let ret_2: Int = obj.getValue(row, col)
*/
```

Rust Solution:

```
// Problem: Subrectangle Queries
// Difficulty: Medium
// 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

struct SubrectangleQueries {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl SubrectangleQueries {

fn new(rectangle: Vec<Vec<i32>>) -> Self {

}

fn update_subrectangle(&self, row1: i32, col1: i32, row2: i32, col2: i32,
new_value: i32) {

}

fn get_value(&self, row: i32, col: i32) -> i32 {

}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* let obj = SubrectangleQueries::new(rectangle);
* obj.update_subrectangle(row1, col1, row2, col2, newValue);
* let ret_2: i32 = obj.get_value(row, col);
*/
```

Ruby Solution:

```
class SubrectangleQueries

=begin
:type rectangle: Integer[][]
```

```
=end
def initialize(rectangle)

end


=begin
:type row1: Integer
:type col1: Integer
:type row2: Integer
:type col2: Integer
:type new_value: Integer
:rtype: Void
=end
def update_subrectangle(row1, col1, row2, col2, new_value)

end


=begin
:type row: Integer
:type col: Integer
:rtype: Integer
=end
def get_value(row, col)

end



end

# Your SubrectangleQueries object will be instantiated and called as such:
# obj = SubrectangleQueries.new(rectangle)
# obj.update_subrectangle(row1, col1, row2, col2, new_value)
# param_2 = obj.get_value(row, col)
```

PHP Solution:

```
class SubrectangleQueries {
/**
* @param Integer[][] $rectangle
```

```
*/
function __construct($rectangle) {

}

/**
* @param Integer $row1
* @param Integer $col1
* @param Integer $row2
* @param Integer $col2
* @param Integer $newValue
* @return NULL
*/
function updateSubrectangle($row1, $col1, $row2, $col2, $newValue) {

}

/**
* @param Integer $row
* @param Integer $col
* @return Integer
*/
function getValue($row, $col) {

}
}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* $obj = SubrectangleQueries($rectangle);
* $obj->updateSubrectangle($row1, $col1, $row2, $col2, $newValue);
* $ret_2 = $obj->getValue($row, $col);
*/
```

Scala Solution:

```
class SubrectangleQueries(_rectangle: Array[Array[Int]]) {

def updateSubrectangle(row1: Int, col1: Int, row2: Int, col2: Int, newValue:
Int) {
```

```
}

def getValue(row: Int, col: Int): Int = {

}

}

/**
* Your SubrectangleQueries object will be instantiated and called as such:
* var obj = new SubrectangleQueries(rectangle)
* obj.updateSubrectangle(row1,col1,row2,col2,newValue)
* var param_2 = obj.getValue(row,col)
*/
```

Racket Solution:

```
(define subrectangle-queries%
(class object%
(super-new)

; rectangle : (listof (listof exact-integer?))
(init-field
rectangle)

; update-subrectangle : exact-integer? exact-integer? exact-integer?
exact-integer? exact-integer? -> void?
(define/public (update-subrectangle row1 col1 row2 col2 newValue)

)
; get-value : exact-integer? exact-integer? -> exact-integer?
(define/public (get-value row col)

)))

;; Your subrectangle-queries% object will be instantiated and called as such:
;; (define obj (new subrectangle-queries% [rectangle rectangle]))
;; (send obj update-subrectangle row1 col1 row2 col2 new-value)
;; (define param_2 (send obj get-value row col))
```