Problem 2664: The Knight’s Tour

Problem Information

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

Problem Description

Given two positive integers

m

and

n

which are the height and width of a

0-indexed

2D-array

board

, a pair of positive integers

(r, c)

which is the starting position of the knight on the board.

Your task is to find an order of movements for the knight, in a manner that every cell of the

board

gets visited

exactly

once (the starting cell is considered visited and you

shouldn't

visit it again).

Return

the array

board

in which the cells' values show the order of visiting the cell starting from 0 (the initial place of the knight).

Note that a

knight

can

move

from cell

(r1, c1)

to cell

(r2, c2)

if

0 <= r2 <= m - 1

and

0 <= c2 <= n - 1

and

min(abs(r1 - r2), abs(c1 - c2)) = 1

and

max(abs(r1 - r2), abs(c1 - c2)) = 2

.

Example 1:

Input:

m = 1, n = 1, r = 0, c = 0

Output:

[[0]]

Explanation:

There is only 1 cell and the knight is initially on it so there is only a 0 inside the 1x1 grid.

Example 2:

Input:

m = 3, n = 4, r = 0, c = 0

Output:

[[0,3,6,9],[11,8,1,4],[2,5,10,7]]

Explanation:

By the following order of movements we can visit the entire board.
(0,0)->(1,2)->(2,0)->(0,1)->(1,3)->(2,1)->(0,2)->(2,3)->(1,1)->(0,3)->(2,2)->(1,0)

Constraints:

1 <= m, n <= 5

0 <= r <= m - 1

0 <= c <= n - 1

The inputs will be generated such that there exists at least one possible order of movements with the given condition

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> tourOfKnight(int m, int n, int r, int c) {

}
};
```

Java:

```
class Solution {
public int[][] tourOfKnight(int m, int n, int r, int c) {

}
}
```

Python3:

```
class Solution:
def tourOfKnight(self, m: int, n: int, r: int, c: int) -> List[List[int]]:
```

Python:

```
class Solution(object):
def tourOfKnight(self, m, n, r, c):
"""
:type m: int
:type n: int
:type r: int
:type c: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number} m
* @param {number} n
* @param {number} r
* @param {number} c
* @return {number[][]}
*/
var tourOfKnight = function(m, n, r, c) {

};
```

TypeScript:

```
function tourOfKnight(m: number, n: number, r: number, c: number): number[][]
{

};
```

C#:

```
public class Solution {
public int[][] TourOfKnight(int m, int n, int r, int c) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
```

```
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** tourOfKnight(int m, int n, int r, int c, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

```
func tourOfKnight(m int, n int, r int, c int) [][]int {

}
```

Kotlin:

```
class Solution {
fun tourOfKnight(m: Int, n: Int, r: Int, c: Int): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func tourOfKnight(_ m: Int, _ n: Int, _ r: Int, _ c: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn tour_of_knight(m: i32, n: i32, r: i32, c: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} m
# @param {Integer} n
```

```
# @param {Integer} r
# @param {Integer} c
# @return {Integer[][]}
def tour_of_knight(m, n, r, c)

end
```

PHP:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer $r
* @param Integer $c
* @return Integer[][]
*/
function tourOfKnight($m, $n, $r, $c) {

}
}
```

Dart:

```
class Solution {
List<List<int>> tourOfKnight(int m, int n, int r, int c) {

}
}
```

Scala:

```
object Solution {
def tourOfKnight(m: Int, n: Int, r: Int, c: Int): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec tour_of_knight(m :: integer, n :: integer, r :: integer, c :: integer)
:: [[integer]]
def tour_of_knight(m, n, r, c) do

end
end
```

Erlang:

```
-spec tour_of_knight(M :: integer(), N :: integer(), R :: integer(), C ::
integer()) -> [[integer()]].
tour_of_knight(M, N, R, C) ->
.
```

Racket:

```
(define/contract (tour-of-knight m n r c)
(-> exact-integer? exact-integer? exact-integer? exact-integer? (listof
(listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Knight’s Tour
* 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 Solution {
public:
vector<vector<int>> tourOfKnight(int m, int n, int r, int c) {

}
```

```
};
```

Java Solution:

```
/**
* Problem: The Knight’s Tour
* 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 Solution {
public int[][] tourOfKnight(int m, int n, int r, int c) {

}
}
```

Python3 Solution:

```
"""
Problem: The Knight’s Tour
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 Solution:
def tourOfKnight(self, m: int, n: int, r: int, c: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def tourOfKnight(self, m, n, r, c):
```

```
"""
:type m: int
:type n: int
:type r: int
:type c: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: The Knight’s Tour
* 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} m
* @param {number} n
* @param {number} r
* @param {number} c
* @return {number[][]}
*/
var tourOfKnight = function(m, n, r, c) {

};
```

TypeScript Solution:

```
/**
* Problem: The Knight’s Tour
* 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
```

```
*/

function tourOfKnight(m: number, n: number, r: number, c: number): number[][]
{

};
```

C# Solution:

```
/*
* Problem: The Knight’s Tour
* 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 Solution {
public int[][] TourOfKnight(int m, int n, int r, int c) {

}
}
```

C Solution:

```
/*
* Problem: The Knight’s Tour
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
```

```
caller calls free().
*/
int** tourOfKnight(int m, int n, int r, int c, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: The Knight’s Tour
// 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

func tourOfKnight(m int, n int, r int, c int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun tourOfKnight(m: Int, n: Int, r: Int, c: Int): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func tourOfKnight(_ m: Int, _ n: Int, _ r: Int, _ c: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: The Knight’s Tour
// 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

impl Solution {
pub fn tour_of_knight(m: i32, n: i32, r: i32, c: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} n
# @param {Integer} r
# @param {Integer} c
# @return {Integer[][]}
def tour_of_knight(m, n, r, c)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer $r
* @param Integer $c
* @return Integer[][]
*/
function tourOfKnight($m, $n, $r, $c) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> tourOfKnight(int m, int n, int r, int c) {

}
}
```

Scala Solution:

```
object Solution {
def tourOfKnight(m: Int, n: Int, r: Int, c: Int): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec tour_of_knight(m :: integer, n :: integer, r :: integer, c :: integer)
:: [[integer]]
def tour_of_knight(m, n, r, c) do

end
end
```

Erlang Solution:

```
-spec tour_of_knight(M :: integer(), N :: integer(), R :: integer(), C ::
integer()) -> [[integer()]].
tour_of_knight(M, N, R, C) ->
.
```

Racket Solution:

```
(define/contract (tour-of-knight m n r c)
(-> exact-integer? exact-integer? exact-integer? exact-integer? (listof
(listof exact-integer?)))
)
```