Problem 576: Out of Boundary Paths

Problem Information

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

Problem Description

There is an

m x n

grid with a ball. The ball is initially at the position

[startRow, startColumn]

. You are allowed to move the ball to one of the four adjacent cells in the grid (possibly out of the grid crossing the grid boundary). You can apply

at most

maxMove

moves to the ball.

Given the five integers

m

,

n

,

maxMove

,

startRow

,

startColumn

, return the number of paths to move the ball out of the grid boundary. Since the answer can be very large, return it

modulo

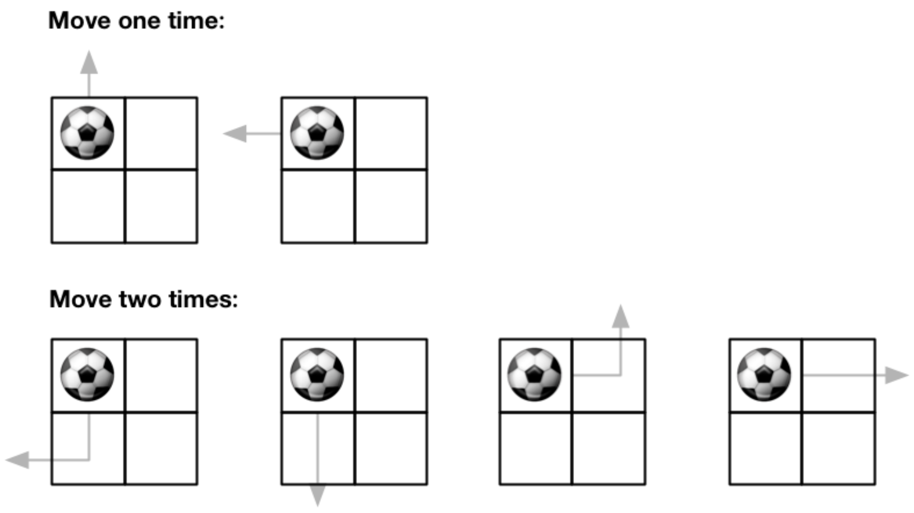
10

9

+ 7

.

Example 1:



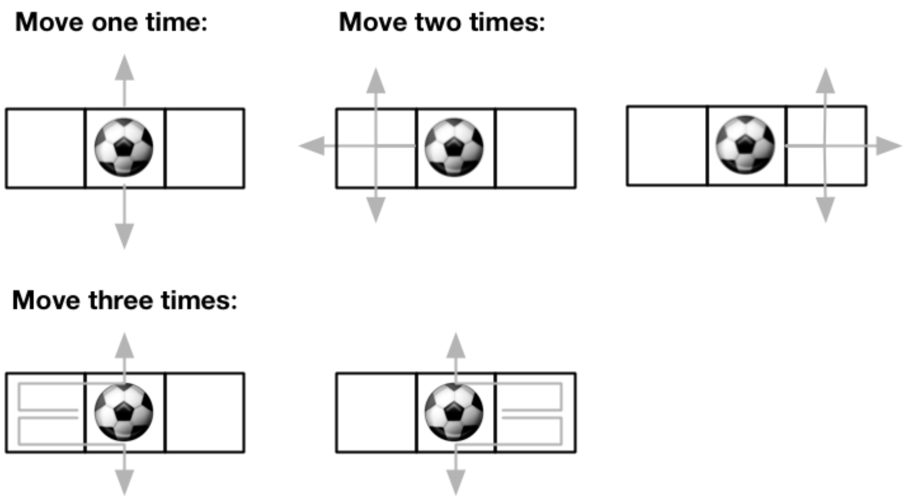
Input:

m = 2, n = 2, maxMove = 2, startRow = 0, startColumn = 0

Output:

6

Example 2:



Input:

m = 1, n = 3, maxMove = 3, startRow = 0, startColumn = 1

Output:

12

Constraints:

1 <= m, n <= 50

0 <= maxMove <= 50

0 <= startRow < m

0 <= startColumn < n

Code Snippets

C++:

```
class Solution {
public:
int findPaths(int m, int n, int maxMove, int startRow, int startColumn) {

}
};
```

Java:

```
class Solution {
public int findPaths(int m, int n, int maxMove, int startRow, int
startColumn) {

}
}
```

Python3:

```
class Solution:
def findPaths(self, m: int, n: int, maxMove: int, startRow: int, startColumn:
int) -> int:
```

Python:

```
class Solution(object):
def findPaths(self, m, n, maxMove, startRow, startColumn):
"""
:type m: int
:type n: int
:type maxMove: int
:type startRow: int
:type startColumn: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} m
* @param {number} n
* @param {number} maxMove
* @param {number} startRow
* @param {number} startColumn
* @return {number}
*/
var findPaths = function(m, n, maxMove, startRow, startColumn) {

};
```

TypeScript:

```
function findPaths(m: number, n: number, maxMove: number, startRow: number,
startColumn: number): number {

};
```

C#:

```
public class Solution {
public int FindPaths(int m, int n, int maxMove, int startRow, int
startColumn) {

}
}
```

C:

```
int findPaths(int m, int n, int maxMove, int startRow, int startColumn) {

}
```

Go:

```
func findPaths(m int, n int, maxMove int, startRow int, startColumn int) int
{

}
```

Kotlin:

```
class Solution {
fun findPaths(m: Int, n: Int, maxMove: Int, startRow: Int, startColumn: Int):
Int {

}
}
```

Swift:

```
class Solution {
func findPaths(_ m: Int, _ n: Int, _ maxMove: Int, _ startRow: Int, _
startColumn: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_paths(m: i32, n: i32, max_move: i32, start_row: i32,
start_column: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} m
# @param {Integer} n
# @param {Integer} max_move
# @param {Integer} start_row
# @param {Integer} start_column
# @return {Integer}
def find_paths(m, n, max_move, start_row, start_column)

end
```

PHP:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer $maxMove
* @param Integer $startRow
* @param Integer $startColumn
* @return Integer
*/
function findPaths($m, $n, $maxMove, $startRow, $startColumn) {

}
}
```

Dart:

```
class Solution {
int findPaths(int m, int n, int maxMove, int startRow, int startColumn) {

}
}
```

Scala:

```
object Solution {
def findPaths(m: Int, n: Int, maxMove: Int, startRow: Int, startColumn: Int):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_paths(m :: integer, n :: integer, max_move :: integer, start_row
:: integer, start_column :: integer) :: integer
def find_paths(m, n, max_move, start_row, start_column) do

end
end
```

Erlang:

```
-spec find_paths(M :: integer(), N :: integer(), MaxMove :: integer(),
StartRow :: integer(), StartColumn :: integer()) -> integer().
```

```
find_paths(M, N, MaxMove, StartRow, StartColumn) ->
.
```

Racket:

```
(define/contract (find-paths m n maxMove startRow startColumn)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Out of Boundary Paths
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int findPaths(int m, int n, int maxMove, int startRow, int startColumn) {

}
};
```

Java Solution:

```
/**
* Problem: Out of Boundary Paths
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int findPaths(int m, int n, int maxMove, int startRow, int
startColumn) {

}
}
```

Python3 Solution:

```
"""
Problem: Out of Boundary Paths
Difficulty: Medium
Tags: dp

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def findPaths(self, m: int, n: int, maxMove: int, startRow: int, startColumn:
int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findPaths(self, m, n, maxMove, startRow, startColumn):
"""
:type m: int
:type n: int
:type maxMove: int
:type startRow: int
:type startColumn: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Out of Boundary Paths
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} m
* @param {number} n
* @param {number} maxMove
* @param {number} startRow
* @param {number} startColumn
* @return {number}
*/
var findPaths = function(m, n, maxMove, startRow, startColumn) {

};
```

TypeScript Solution:

```
/**
* Problem: Out of Boundary Paths
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function findPaths(m: number, n: number, maxMove: number, startRow: number,
startColumn: number): number {

};
```

C# Solution:

```
/*
* Problem: Out of Boundary Paths
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int FindPaths(int m, int n, int maxMove, int startRow, int
startColumn) {

}
}
```

C Solution:

```
/*
* Problem: Out of Boundary Paths
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int findPaths(int m, int n, int maxMove, int startRow, int startColumn) {

}
```

Go Solution:

```
// Problem: Out of Boundary Paths
// Difficulty: Medium
// Tags: dp
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table
```

```
func findPaths(m int, n int, maxMove int, startRow int, startColumn int) int
{

}
```

Kotlin Solution:

```
class Solution {
fun findPaths(m: Int, n: Int, maxMove: Int, startRow: Int, startColumn: Int):
Int {

}
}
```

Swift Solution:

```
class Solution {
func findPaths(_ m: Int, _ n: Int, _ maxMove: Int, _ startRow: Int, _
startColumn: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Out of Boundary Paths
// Difficulty: Medium
// Tags: dp
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn find_paths(m: i32, n: i32, max_move: i32, start_row: i32,
start_column: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} n
# @param {Integer} max_move
# @param {Integer} start_row
# @param {Integer} start_column
# @return {Integer}
def find_paths(m, n, max_move, start_row, start_column)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer $maxMove
* @param Integer $startRow
* @param Integer $startColumn
* @return Integer
*/
function findPaths($m, $n, $maxMove, $startRow, $startColumn) {

}
}
```

Dart Solution:

```
class Solution {
int findPaths(int m, int n, int maxMove, int startRow, int startColumn) {

}
}
```

Scala Solution:

```
object Solution {
def findPaths(m: Int, n: Int, maxMove: Int, startRow: Int, startColumn: Int):
Int = {
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_paths(m :: integer, n :: integer, max_move :: integer, start_row
:: integer, start_column :: integer) :: integer
def find_paths(m, n, max_move, start_row, start_column) do

end
end
```

Erlang Solution:

```
-spec find_paths(M :: integer(), N :: integer(), MaxMove :: integer(),
StartRow :: integer(), StartColumn :: integer()) -> integer().
find_paths(M, N, MaxMove, StartRow, StartColumn) ->
.
```

Racket Solution:

```
(define/contract (find-paths m n maxMove startRow startColumn)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer? exact-integer?)
)
```