Problem 3021: Alice and Bob Playing Flower Game

Problem Information

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

Problem Description

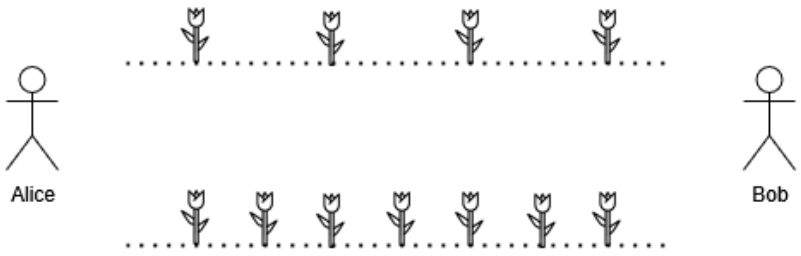
Alice and Bob are playing a turn-based game on a field, with two lanes of flowers between them. There are

x

flowers in the first lane between Alice and Bob, and

y

flowers in the second lane between them.



The game proceeds as follows:

Alice takes the first turn.

In each turn, a player must choose either one of the lane and pick one flower from that side.

At the end of the turn, if there are no flowers left at all in either lane, the

current

player captures their opponent and wins the game.

Given two integers,

n

and

m

, the task is to compute the number of possible pairs

(x, y)

that satisfy the conditions:

Alice must win the game according to the described rules.

The number of flowers

x

in the first lane must be in the range

$[1,n]$

.

The number of flowers

y

in the second lane must be in the range

[1,m]

.

Return

the number of possible pairs

(x, y)

that satisfy the conditions mentioned in the statement

.

Example 1:

Input:

n = 3, m = 2

Output:

3

Explanation:

The following pairs satisfy conditions described in the statement: (1,2), (3,2), (2,1).

Example 2:

Input:

n = 1, m = 1

Output:

0

Explanation:

No pairs satisfy the conditions described in the statement.

Constraints:

1 <= n, m <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long flowerGame(int n, int m) {

}
};
```

Java:

```
class Solution {
public long flowerGame(int n, int m) {

}
}
```

Python3:

```
class Solution:
def flowerGame(self, n: int, m: int) -> int:
```

Python:

```
class Solution(object):
def flowerGame(self, n, m):
"""
:type n: int
```

```
:type m: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} m
* @return {number}
*/
var flowerGame = function(n, m) {

};
```

TypeScript:

```
function flowerGame(n: number, m: number): number {

};
```

C#:

```
public class Solution {
public long FlowerGame(int n, int m) {

}
}
```

C:

```
long long flowerGame(int n, int m) {

}
```

Go:

```
func flowerGame(n int, m int) int64 {

}
```

Kotlin:

```
class Solution {
fun flowerGame(n: Int, m: Int): Long {

}
}
```

Swift:

```
class Solution {
func flowerGame(_ n: Int, _ m: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn flower_game(n: i32, m: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} m
# @return {Integer}
def flower_game(n, m)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @return Integer
*/
function flowerGame($n, $m) {

}
```

```
}
```

Dart:

```
class Solution {
int flowerGame(int n, int m) {

}
}
```

Scala:

```
object Solution {
def flowerGame(n: Int, m: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec flower_game(n :: integer, m :: integer) :: integer
def flower_game(n, m) do

end
end
```

Erlang:

```
-spec flower_game(N :: integer(), M :: integer()) -> integer().
flower_game(N, M) ->
.
```

Racket:

```
(define/contract (flower-game n m)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Alice and Bob Playing Flower Game
* 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:
long long flowerGame(int n, int m) {

}
};
```

Java Solution:

```
/**
* Problem: Alice and Bob Playing Flower Game
* 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 long flowerGame(int n, int m) {

}
}
```

Python3 Solution:

```
"""
Problem: Alice and Bob Playing Flower Game
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 flowerGame(self, n: int, m: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def flowerGame(self, n, m):
"""
:type n: int
:type m: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Alice and Bob Playing Flower Game
* 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} n
* @param {number} m
* @return {number}
*/
var flowerGame = function(n, m) {

};
```

TypeScript Solution:

```
/**
* Problem: Alice and Bob Playing Flower Game
* 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 flowerGame(n: number, m: number): number {

};
```

C# Solution:

```
/*
* Problem: Alice and Bob Playing Flower Game
* 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 long FlowerGame(int n, int m) {

}
}
```

C Solution:

```
/*
* Problem: Alice and Bob Playing Flower Game
* 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
*/

long long flowerGame(int n, int m) {

}
```

Go Solution:

```
// Problem: Alice and Bob Playing Flower Game
// 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 flowerGame(n int, m int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun flowerGame(n: Int, m: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func flowerGame(_ n: Int, _ m: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Alice and Bob Playing Flower Game
// 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 flower_game(n: i32, m: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} m
# @return {Integer}
def flower_game(n, m)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @return Integer
*/
function flowerGame($n, $m) {

}
}
```

Dart Solution:

```
class Solution {
int flowerGame(int n, int m) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def flowerGame(n: Int, m: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec flower_game(n :: integer, m :: integer) :: integer
def flower_game(n, m) do

end
end
```

Erlang Solution:

```
-spec flower_game(N :: integer(), M :: integer()) -> integer().
flower_game(N, M) ->
.
```

Racket Solution:

```
(define/contract (flower-game n m)
(-> exact-integer? exact-integer? exact-integer?)
)
```