Problem 1659: Maximize Grid Happiness

Problem Information

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

Problem Description

You are given four integers,

m

,

n

,

introvertsCount

, and

extrovertsCount

. You have an

m x n

grid, and there are two types of people: introverts and extroverts. There are

introvertsCount

introverts and

extrovertsCount

extroverts.

You should decide how many people you want to live in the grid and assign each of them one grid cell. Note that you

do not

have to have all the people living in the grid.

The

happiness

of each person is calculated as follows:

Introverts

start

with

120

happiness and

lose

30

happiness for each neighbor (introvert or extrovert).

Extroverts

start

with

40

happiness and

gain

20

happiness for each neighbor (introvert or extrovert).

Neighbors live in the directly adjacent cells north, east, south, and west of a person's cell.

The

grid happiness

is the

sum

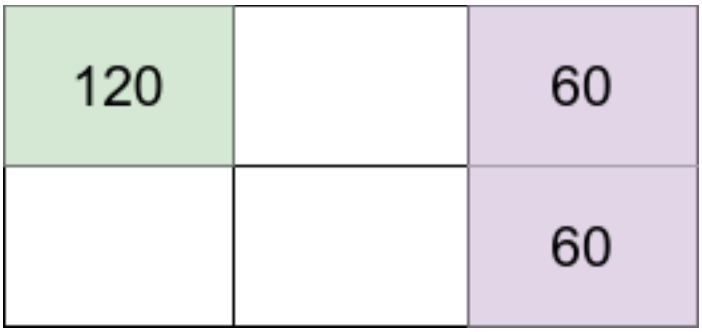
of each person's happiness. Return

the

maximum possible grid happiness

.

Example 1:



Input:

m = 2, n = 3, introvertsCount = 1, extrovertsCount = 2

Output:

240

Explanation:

Assume the grid is 1-indexed with coordinates (row, column). We can put the introvert in cell (1,1) and put the extroverts in cells (1,3) and (2,3). - Introvert at (1,1) happiness: 120 (starting happiness) - (0 * 30) (0 neighbors) = 120 - Extrovert at (1,3) happiness: 40 (starting happiness) + (1 * 20) (1 neighbor) = 60 - Extrovert at (2,3) happiness: 40 (starting happiness) + (1 * 20) (1 neighbor) = 60 The grid happiness is 120 + 60 + 60 = 240. The above figure shows the grid in this example with each person's happiness. The introvert stays in the light green cell while the extroverts live on the light purple cells.

Example 2:

Input:

m = 3, n = 1, introvertsCount = 2, extrovertsCount = 1

Output:

260

Explanation:

Place the two introverts in (1,1) and (3,1) and the extrovert at (2,1). - Introvert at (1,1) happiness: 120 (starting happiness) - (1 * 30) (1 neighbor) = 90 - Extrovert at (2,1) happiness: 40 (starting happiness) + (2 * 20) (2 neighbors) = 80 - Introvert at (3,1) happiness: 120 (starting happiness) - (1 * 30) (1 neighbor) = 90 The grid happiness is 90 + 80 + 90 = 260.

Example 3:

Input:

m = 2, n = 2, introvertsCount = 4, extrovertsCount = 0

Output:

240

Constraints:

1 <= m, n <= 5

0 <= introvertsCount, extrovertsCount <= min(m * n, 6)

Code Snippets

C++:

```
class Solution {
public:
int getMaxGridHappiness(int m, int n, int introvertsCount, int
extrovertsCount) {

}
};
```

Java:

```
class Solution {
public int getMaxGridHappiness(int m, int n, int introvertsCount, int
extrovertsCount) {

}
}
```

Python3:

```
class Solution:
def getMaxGridHappiness(self, m: int, n: int, introvertsCount: int,
extrovertsCount: int) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function getMaxGridHappiness(m: number, n: number, introvertsCount: number,
extrovertsCount: number): number {

};
```

C#:

```
public class Solution {
public int GetMaxGridHappiness(int m, int n, int introvertsCount, int
extrovertsCount) {

}
}
```

C:

```
int getMaxGridHappiness(int m, int n, int introvertsCount, int
extrovertsCount) {
```

```
}
```

Go:

```
func getMaxGridHappiness(m int, n int, introvertsCount int, extrovertsCount
int) int {

}
```

Kotlin:

```
class Solution {
fun getMaxGridHappiness(m: Int, n: Int, introvertsCount: Int,
extrovertsCount: Int): Int {

}
}
```

Swift:

```
class Solution {
func getMaxGridHappiness(_ m: Int, _ n: Int, _ introvertsCount: Int, _
extrovertsCount: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn get_max_grid_happiness(m: i32, n: i32, introverts_count: i32,
extroverts_count: i32) -> i32 {

}
}
```

Ruby:

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

```
# @param {Integer} extroverts_count
# @return {Integer}
def get_max_grid_happiness(m, n, introverts_count, extroverts_count)

end
```

PHP:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer $introvertsCount
* @param Integer $extrovertsCount
* @return Integer
*/
function getMaxGridHappiness($m, $n, $introvertsCount, $extrovertsCount) {

}
}
```

Dart:

```
class Solution {
int getMaxGridHappiness(int m, int n, int introvertsCount, int
extrovertsCount) {

}
}
```

Scala:

```
object Solution {
def getMaxGridHappiness(m: Int, n: Int, introvertsCount: Int,
extrovertsCount: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_max_grid_happiness(m :: integer, n :: integer, introverts_count ::
integer, extroverts_count :: integer) :: integer
def get_max_grid_happiness(m, n, introverts_count, extroverts_count) do

end
end
```

Erlang:

```
-spec get_max_grid_happiness(M :: integer(), N :: integer(), IntrovertsCount
:: integer(), ExtrovertsCount :: integer()) -> integer().
get_max_grid_happiness(M, N, IntrovertsCount, ExtrovertsCount) ->
.
```

Racket:

```
(define/contract (get-max-grid-happiness m n introvertsCount extrovertsCount)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Grid Happiness
* Difficulty: Hard
* 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 getMaxGridHappiness(int m, int n, int introvertsCount, int
extrovertsCount) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Maximize Grid Happiness
* Difficulty: Hard
* 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 getMaxGridHappiness(int m, int n, int introvertsCount, int
extrovertsCount) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Grid Happiness
Difficulty: Hard
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 getMaxGridHappiness(self, m: int, n: int, introvertsCount: int,
extrovertsCount: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Grid Happiness
* Difficulty: Hard
* 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} introvertsCount
* @param {number} extrovertsCount
* @return {number}
*/
var getMaxGridHappiness = function(m, n, introvertsCount, extrovertsCount) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Grid Happiness
* Difficulty: Hard
* 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 getMaxGridHappiness(m: number, n: number, introvertsCount: number,
extrovertsCount: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Grid Happiness
* Difficulty: Hard
* 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 GetMaxGridHappiness(int m, int n, int introvertsCount, int
extrovertsCount) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Grid Happiness
* Difficulty: Hard
* 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 getMaxGridHappiness(int m, int n, int introvertsCount, int
extrovertsCount) {
```

```
}
```

Go Solution:

```
// Problem: Maximize Grid Happiness
// Difficulty: Hard
// 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 getMaxGridHappiness(m int, n int, introvertsCount int, extrovertsCount
int) int {

}
```

Kotlin Solution:

```
class Solution {
fun getMaxGridHappiness(m: Int, n: Int, introvertsCount: Int,
extrovertsCount: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func getMaxGridHappiness(_ m: Int, _ n: Int, _ introvertsCount: Int, _
extrovertsCount: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize Grid Happiness
// Difficulty: Hard
// 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 get_max_grid_happiness(m: i32, n: i32, introverts_count: i32,
extroverts_count: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} n
# @param {Integer} introverts_count
# @param {Integer} extroverts_count
# @return {Integer}
def get_max_grid_happiness(m, n, introverts_count, extroverts_count)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer $introvertsCount
* @param Integer $extrovertsCount
* @return Integer
*/
function getMaxGridHappiness($m, $n, $introvertsCount, $extrovertsCount) {

}
}
```

Dart Solution:

```
class Solution {
int getMaxGridHappiness(int m, int n, int introvertsCount, int
extrovertsCount) {

}
}
```

Scala Solution:

```
object Solution {
def getMaxGridHappiness(m: Int, n: Int, introvertsCount: Int,
extrovertsCount: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_max_grid_happiness(m :: integer, n :: integer, introverts_count ::
integer, extroverts_count :: integer) :: integer
def get_max_grid_happiness(m, n, introverts_count, extroverts_count) do

end
end
```

Erlang Solution:

```
-spec get_max_grid_happiness(M :: integer(), N :: integer(), IntrovertsCount
:: integer(), ExtrovertsCount :: integer()) -> integer().
get_max_grid_happiness(M, N, IntrovertsCount, ExtrovertsCount) ->
.
```

Racket Solution:

```
(define/contract (get-max-grid-happiness m n introvertsCount extrovertsCount)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```