Problem 3426: Manhattan Distances of All Arrangements of Pieces

Problem Information

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

Problem Description

You are given three integers

m

,

n

, and

k

.

There is a rectangular grid of size

m × n

containing

k

identical pieces. Return the sum of Manhattan distances between every pair of pieces over all

valid arrangements

of pieces.

A

valid arrangement

is a placement of all

k

pieces on the grid with

at most

one piece per cell.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

The Manhattan Distance between two cells

(x

i

, y

i

)

and

(x

j

, y

j

)

is

|x

i

- x

j

| + |y

i

- y

j

|

.

Example 1:

Input:

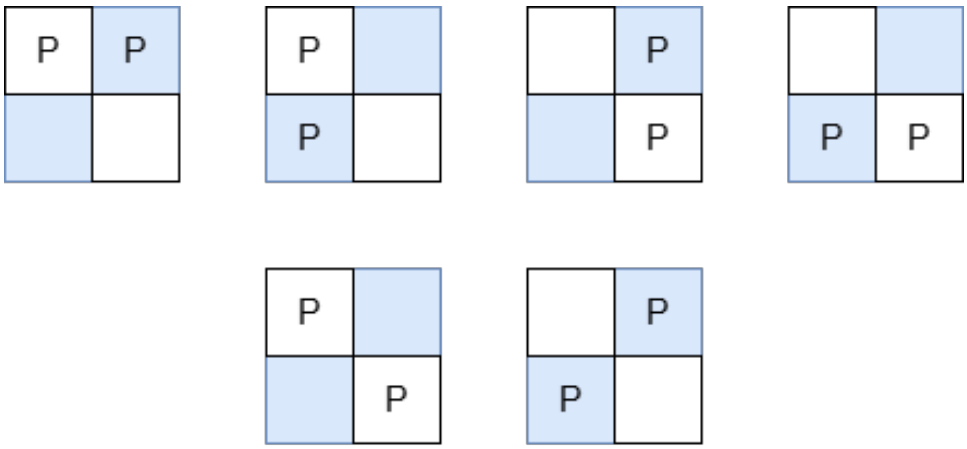
m = 2, n = 2, k = 2

Output:

8

Explanation:

The valid arrangements of pieces on the board are:



In the first 4 arrangements, the Manhattan distance between the two pieces is 1.

In the last 2 arrangements, the Manhattan distance between the two pieces is 2.

Thus, the total Manhattan distance across all valid arrangements is

1 + 1 + 1 + 1 + 2 + 2 = 8

.

Example 2:

Input:

m = 1, n = 4, k = 3

Output:

20

Explanation:

The valid arrangements of pieces on the board are:

The first and last arrangements have a total Manhattan distance of

1 + 1 + 2 = 4

.

The middle two arrangements have a total Manhattan distance of

1 + 2 + 3 = 6

.

The total Manhattan distance between all pairs of pieces across all arrangements is

4 + 6 + 6 + 4 = 20

.

Constraints:

1 <= m, n <= 10

5

2 <= m * n <= 10

5

2 <= k <= m * n

Code Snippets

C++:

```
class Solution {
public:
int distanceSum(int m, int n, int k) {

}
};
```

Java:

```
class Solution {
public int distanceSum(int m, int n, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} m
* @param {number} n
* @param {number} k
* @return {number}
*/
```

```
var distanceSum = function(m, n, k) {

};
```

TypeScript:

```
function distanceSum(m: number, n: number, k: number): number {

};
```

C#:

```
public class Solution {
public int DistanceSum(int m, int n, int k) {

}
}
```

C:

```
int distanceSum(int m, int n, int k) {

}
```

Go:

```
func distanceSum(m int, n int, k int) int {

}
```

Kotlin:

```
class Solution {
fun distanceSum(m: Int, n: Int, k: Int): Int {

}
}
```

Swift:

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

```
}
}
```

Rust:

```
impl Solution {
pub fn distance_sum(m: i32, n: i32, k: i32) -> i32 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int distanceSum(int m, int n, int k) {
```

```
}
}
```

Scala:

```
object Solution {
def distanceSum(m: Int, n: Int, k: Int): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (distance-sum m n k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Manhattan Distances of All Arrangements of Pieces
* Difficulty: Hard
```

```
* 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:
int distanceSum(int m, int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Manhattan Distances of All Arrangements of Pieces
* Difficulty: Hard
* 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 int distanceSum(int m, int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Manhattan Distances of All Arrangements of Pieces
Difficulty: Hard
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 distanceSum(self, m: int, n: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Manhattan Distances of All Arrangements of Pieces
* Difficulty: Hard
* 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} m
* @param {number} n
* @param {number} k
* @return {number}
*/
var distanceSum = function(m, n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Manhattan Distances of All Arrangements of Pieces
* Difficulty: Hard
* 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 distanceSum(m: number, n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Manhattan Distances of All Arrangements of Pieces
* Difficulty: Hard
* 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 int DistanceSum(int m, int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Manhattan Distances of All Arrangements of Pieces
* Difficulty: Hard
* 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
*/

int distanceSum(int m, int n, int k) {

}
```

Go Solution:

```
// Problem: Manhattan Distances of All Arrangements of Pieces
// Difficulty: Hard
// 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 distanceSum(m int, n int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun distanceSum(m: Int, n: Int, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Manhattan Distances of All Arrangements of Pieces
// Difficulty: Hard
```

```
// 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 distance_sum(m: i32, n: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int distanceSum(int m, int n, int k) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def distanceSum(m: Int, n: Int, k: Int): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (distance-sum m n k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```