Problem 3363: Find the Maximum Number of Fruits Collected

Problem Information

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

Problem Description

There is a game dungeon comprised of

n x n

rooms arranged in a grid.

You are given a 2D array

fruits

of size

n x n

, where

fruits[i][j]

represents the number of fruits in the room

(i, j)

. Three children will play in the game dungeon, with

initial

positions at the corner rooms

(0, 0)

,

(0, n - 1)

, and

(n - 1, 0)

.

The children will make

exactly

n - 1

moves according to the following rules to reach the room

(n - 1, n - 1)

:

The child starting from

(0, 0)

must move from their current room

(i, j)

to one of the rooms

(i + 1, j + 1)

,

(i + 1, j)

, and

(i, j + 1)

if the target room exists.

The child starting from

(0, n - 1)

must move from their current room

(i, j)

to one of the rooms

(i + 1, j - 1)

,

(i + 1, j)

, and

(i + 1, j + 1)

if the target room exists.

The child starting from

(n - 1, 0)

must move from their current room

(i, j)

to one of the rooms

(i - 1, j + 1)

,

(i, j + 1)

, and

(i + 1, j + 1)

if the target room exists.

When a child enters a room, they will collect all the fruits there. If two or more children enter the same room, only one child will collect the fruits, and the room will be emptied after they leave.

Return the

maximum

number of fruits the children can collect from the dungeon.

Example 1:

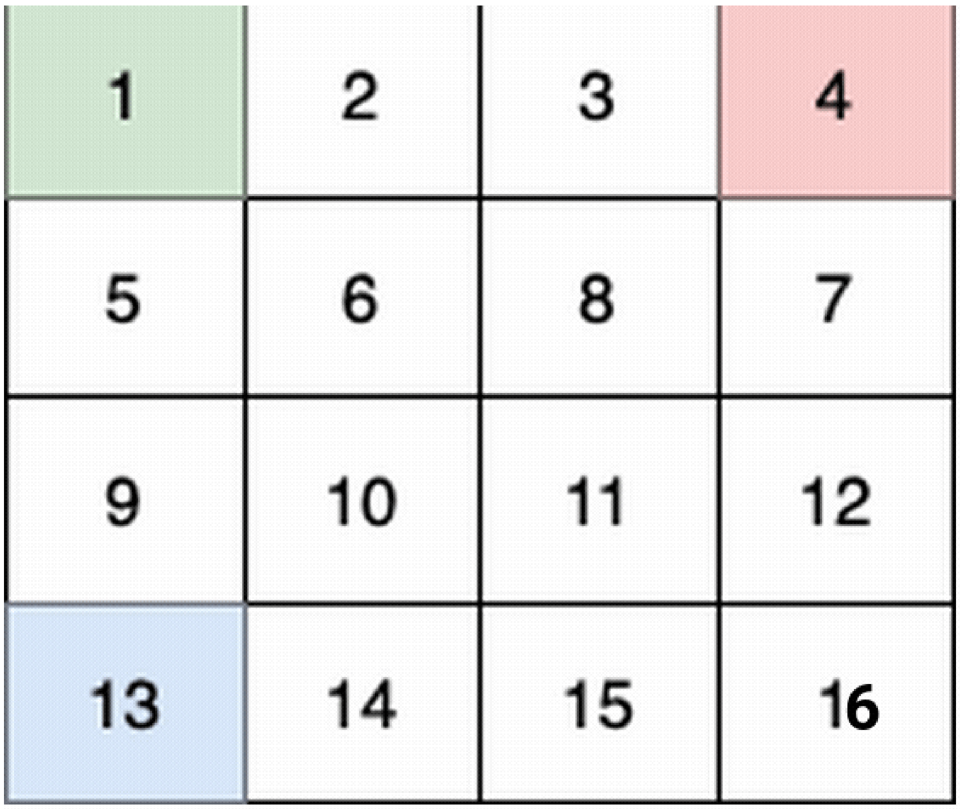
Input:

fruits = [[1,2,3,4],[5,6,8,7],[9,10,11,12],[13,14,15,16]]

Output:

100

Explanation:



1	2	3	4
5	6	8	7
9	10	11	12
13	14	15	16

In this example:

The 1

st

child (green) moves on the path

(0,0) -> (1,1) -> (2,2) -> (3, 3)

.

The 2

nd

child (red) moves on the path

(0,3) -> (1,2) -> (2,3) -> (3, 3)

.

The 3

rd

child (blue) moves on the path

(3,0) -> (3,1) -> (3,2) -> (3, 3)

.

In total they collect

1 + 6 + 11 + 16 + 4 + 8 + 12 + 13 + 14 + 15 = 100

fruits.

Example 2:

Input:

fruits = [[1,1],[1,1]]

Output:

4

Explanation:

In this example:

The 1

st

child moves on the path

(0,0) -> (1,1)

.

The 2

nd

child moves on the path

(0,1) -> (1,1)

.

The 3

rd

child moves on the path

(1,0) -> (1,1)

.

In total they collect

1 + 1 + 1 + 1 = 4

fruits.

Constraints:

2 <= n == fruits.length == fruits[i].length <= 1000

0 <= fruits[i][j] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int maxCollectedFruits(vector<vector<int>>& fruits) {

}
};
```

Java:

```
class Solution {
public int maxCollectedFruits(int[][] fruits) {

}
}
```

Python3:

```
class Solution:
def maxCollectedFruits(self, fruits: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def maxCollectedFruits(self, fruits):
"""
:type fruits: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} fruits
* @return {number}
*/
var maxCollectedFruits = function(fruits) {

};
```

TypeScript:

```
function maxCollectedFruits(fruits: number[][]): number {
};
```

C#:

```
public class Solution {
public int MaxCollectedFruits(int[][] fruits) {

}
}
```

C:

```
int maxCollectedFruits(int** fruits, int fruitsSize, int* fruitsColSize) {

}
```

Go:

```
func maxCollectedFruits(fruits [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxCollectedFruits(fruits: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maxCollectedFruits(_ fruits: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_collected_fruits(fruits: Vec<Vec<i32>>) -> i32 {
}
}
```

Ruby:

```
# @param {Integer[][]} fruits
# @return {Integer}
def max_collected_fruits(fruits)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $fruits
* @return Integer
*/
function maxCollectedFruits($fruits) {

}
}
```

Dart:

```
class Solution {
int maxCollectedFruits(List<List<int>> fruits) {

}
}
```

Scala:

```
object Solution {
def maxCollectedFruits(fruits: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_collected_fruits(fruits :: [[integer]]) :: integer
def max_collected_fruits(fruits) do

end
end
```

Erlang:

```
-spec max_collected_fruits(Fruits :: [[integer()]]) -> integer().
max_collected_fruits(Fruits) ->
.
```

Racket:

```
(define/contract (max-collected-fruits fruits)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Maximum Number of Fruits Collected
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxCollectedFruits(vector<vector<int>>& fruits) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Maximum Number of Fruits Collected
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxCollectedFruits(int[][] fruits) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Maximum Number of Fruits Collected
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxCollectedFruits(self, fruits: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxCollectedFruits(self, fruits):
"""
:type fruits: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Find the Maximum Number of Fruits Collected
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} fruits
* @return {number}
*/
var maxCollectedFruits = function(fruits) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Find the Maximum Number of Fruits Collected
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxCollectedFruits(fruits: number[][]): number {

};
```

C# Solution:

```csharp
/*
* Problem: Find the Maximum Number of Fruits Collected
* Difficulty: Hard
* Tags: array, dp
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxCollectedFruits(int[][] fruits) {

}
}
```

C Solution:

```
/*
* Problem: Find the Maximum Number of Fruits Collected
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxCollectedFruits(int** fruits, int fruitsSize, int* fruitsColSize) {

}
```

Go Solution:

```
// Problem: Find the Maximum Number of Fruits Collected
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxCollectedFruits(fruits [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxCollectedFruits(fruits: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxCollectedFruits(_ fruits: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Maximum Number of Fruits Collected
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_collected_fruits(fruits: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} fruits
# @return {Integer}
def max_collected_fruits(fruits)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $fruits
* @return Integer
*/
function maxCollectedFruits($fruits) {

}
}
```

Dart Solution:

```
class Solution {
int maxCollectedFruits(List<List<int>> fruits) {

}
}
```

Scala Solution:

```
object Solution {
def maxCollectedFruits(fruits: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_collected_fruits(fruits :: [[integer]]) :: integer
def max_collected_fruits(fruits) do

end
end
```

Erlang Solution:

```
-spec max_collected_fruits(Fruits :: [[integer()]]) -> integer().
max_collected_fruits(Fruits) ->
.
```

Racket Solution:

```
(define/contract (max-collected-fruits fruits)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```