

Problem 1992: Find All Groups of Farmland

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

$m \times n$

binary matrix

land

where a

0

represents a hectare of forested land and a

1

represents a hectare of farmland.

To keep the land organized, there are designated rectangular areas of hectares that consist

entirely

of farmland. These rectangular areas are called

groups

. No two groups are adjacent, meaning farmland in one group is

not

four-directionally adjacent to another farmland in a different group.

land

can be represented by a coordinate system where the top left corner of

land

is

$(0, 0)$

and the bottom right corner of

land

is

$(m-1, n-1)$

. Find the coordinates of the top left and bottom right corner of each

group

of farmland. A

group

of farmland with a top left corner at

$(r$

, c

1

)

and a bottom right corner at

(r

2

, c

2

)

is represented by the 4-length array

[r

1

, c

1

, r

2

, c

2

].

Return

a 2D array containing the 4-length arrays described above for each

group

of farmland in

land

. If there are no groups of farmland, return an empty array. You may return the answer in

any order

.

Example 1:

1	0	0
0	1	1
0	1	1

Input:

land = [[1,0,0],[0,1,1],[0,1,1]]

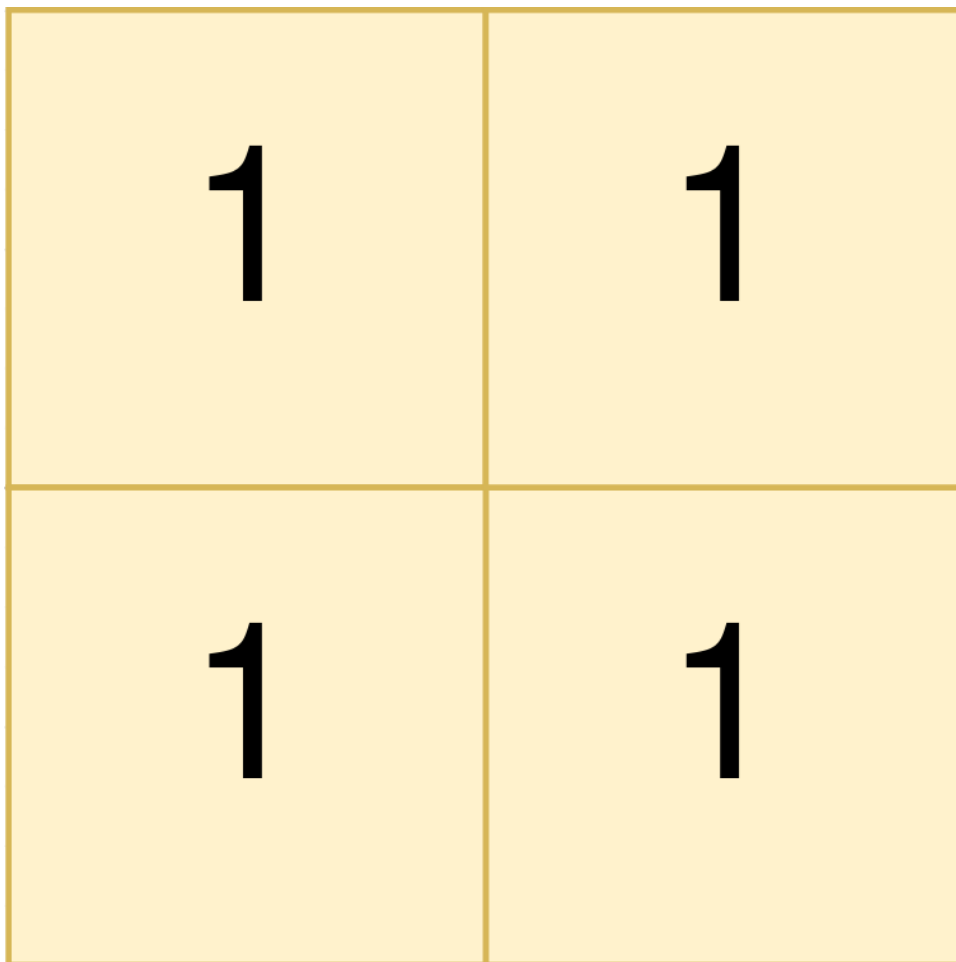
Output:

[[0,0,0,0],[1,1,2,2]]

Explanation:

The first group has a top left corner at land[0][0] and a bottom right corner at land[0][0]. The second group has a top left corner at land[1][1] and a bottom right corner at land[2][2].

Example 2:



Input:

```
land = [[1,1],[1,1]]
```

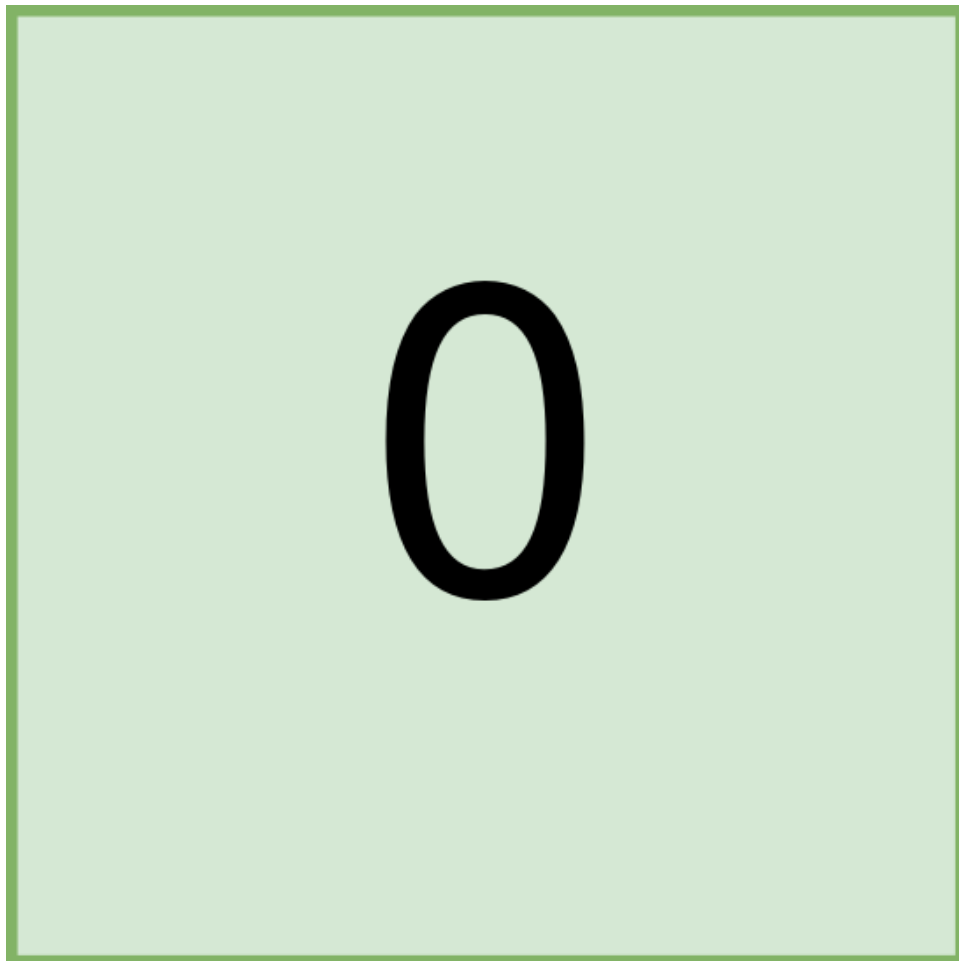
Output:

```
[[0,0,1,1]]
```

Explanation:

The first group has a top left corner at `land[0][0]` and a bottom right corner at `land[1][1]`.

Example 3:



Input:

```
land = [[0]]
```

Output:

[]

Explanation:

There are no groups of farmland.

Constraints:

$m == \text{land.length}$

$n == \text{land}[i].\text{length}$

$1 \leq m, n \leq 300$

land

consists of only

0

's and

1

's.

Groups of farmland are

rectangular

in shape.

Code Snippets

C++:

```
class Solution {  
public:
```

```
vector<vector<int>> findFarmland(vector<vector<int>>& land) {

}

};
```

Java:

```
class Solution {
public int[][] findFarmland(int[][] land) {

}

}
```

Python3:

```
class Solution:
def findFarmland(self, land: List[List[int]]) -> List[List[int]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findFarmland(land: number[][]): number[][] {

};
```

C#:

```
public class Solution {  
    public int[][] FindFarmland(int[][] land) {  
  
    }  
}
```

C:

```
/**  
 * Return an array of arrays of size *returnSize.  
 * The sizes of the arrays are returned as *returnColumnSizes array.  
 * Note: Both returned array and *columnSizes array must be malloced, assume  
 caller calls free().  
 */  
int** findFarmland(int** land, int landSize, int* landColSize, int*  
returnSize, int** returnColumnSizes) {  
  
}
```

Go:

```
func findFarmland(land [][]int) [][]int {  
  
}
```

Kotlin:

```
class Solution {  
    fun findFarmland(land: Array<IntArray>): Array<IntArray> {  
  
    }  
}
```

Swift:

```
class Solution {  
    func findFarmland(_ land: [[Int]]) -> [[Int]] {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn find_farmland(land: Vec<Vec<i32>>) -> Vec<Vec<i32>> {  
  
    }  
}
```

Ruby:

```
# @param {Integer[][]} land  
# @return {Integer[][]}  
def find_farmland(land)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $land  
     * @return Integer[][]  
     */  
    function findFarmland($land) {  
  
    }  
}
```

Dart:

```
class Solution {  
    List<List<int>> findFarmland(List<List<int>> land) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def findFarmland(land: Array[Array[Int]]): Array[Array[Int]] = {  
  
    }  
}
```

```
}
```

Elixir:

```
defmodule Solution do
  @spec find_farmland(land :: [[integer]]) :: [[integer]]
  def find_farmland(land) do

  end
end
```

Erlang:

```
-spec find_farmland(Land :: [[integer()]]) -> [[integer()]].
find_farmland(Land) ->
.
```

Racket:

```
(define/contract (find-farmland land)
  (-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Find All Groups of Farmland
 * Difficulty: Medium
 * Tags: array, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public:
    vector<vector<int>> findFarmland(vector<vector<int>>& land) {
```

```
}  
};
```

Java Solution:

```
/**  
 * Problem: Find All Groups of Farmland  
 * Difficulty: Medium  
 * Tags: array, search  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(1) to O(n) depending on approach  
 */  
  
class Solution {  
    public int[][] findFarmland(int[][] land) {  
  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Find All Groups of Farmland  
Difficulty: Medium  
Tags: array, search  
  
Approach: Use two pointers or sliding window technique  
Time Complexity: O(n) or O(n log n)  
Space Complexity: O(1) to O(n) depending on approach  
"""  
  
class Solution:  
    def findFarmland(self, land: List[List[int]]) -> List[List[int]]:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

```

class Solution(object):
def findFarmland(self, land):
    """
    :type land: List[List[int]]
    :rtype: List[List[int]]
    """

```

JavaScript Solution:

```

/**
 * Problem: Find All Groups of Farmland
 * Difficulty: Medium
 * Tags: array, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * @param {number[][]} land
 * @return {number[][]}
 */
var findFarmland = function(land) {

};

```

TypeScript Solution:

```

/**
 * Problem: Find All Groups of Farmland
 * Difficulty: Medium
 * Tags: array, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

function findFarmland(land: number[][]): number[][] {

};

```

C# Solution:

```
/*
 * Problem: Find All Groups of Farmland
 * Difficulty: Medium
 * Tags: array, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

public class Solution {
    public int[][] FindFarmland(int[][] land) {

    }
}
```

C Solution:

```
/*
 * Problem: Find All Groups of Farmland
 * Difficulty: Medium
 * Tags: array, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * Return an array of arrays of size *returnSize.
 * The sizes of the arrays are returned as *returnColumnSizes array.
 * Note: Both returned array and *columnSizes array must be malloced, assume
 * caller calls free().
 */
int** findFarmland(int** land, int landSize, int* landColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```

// Problem: Find All Groups of Farmland
// Difficulty: Medium
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func findFarmland(land [][]int) [][]int {

}

```

Kotlin Solution:

```

class Solution {
    fun findFarmland(land: Array<IntArray>): Array<IntArray> {

    }
}

```

Swift Solution:

```

class Solution {
    func findFarmland(_ land: [[Int]]) -> [[Int]] {

    }
}

```

Rust Solution:

```

// Problem: Find All Groups of Farmland
// Difficulty: Medium
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
    pub fn find_farmland(land: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

    }
}

```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} land
# @return {Integer[][]}
def find_farmland(land)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $land
     * @return Integer[][]
     */
    function findFarmland($land) {

    }

}
```

Dart Solution:

```
class Solution {
  List<List<int>> findFarmland(List<List<int>> land) {

  }

}
```

Scala Solution:

```
object Solution {
  def findFarmland(land: Array[Array[Int]]): Array[Array[Int]] = {

  }

}
```

Elixir Solution:

```
defmodule Solution do
  @spec find_farmland(land :: [[integer]]) :: [[integer]]
  def find_farmland(land) do

  end
end
```

Erlang Solution:

```
-spec find_farmland(Land :: [[integer()]]) -> [[integer()]].
find_farmland(Land) ->
.
```

Racket Solution:

```
(define/contract (find-farmland land)
  (-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```