

Problem 827: Making A Large Island

Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an

$n \times n$

binary matrix

grid

. You are allowed to change

at most one

0

to be

1

.

Return

the size of the largest

island

in

grid

after applying this operation

.

An

island

is a 4-directionally connected group of

1

s.

Example 1:

Input:

grid = [[1,0],[0,1]]

Output:

3

Explanation:

Change one 0 to 1 and connect two 1s, then we get an island with area = 3.

Example 2:

Input:

grid = [[1,1],[1,0]]

Output:

4

Explanation:

Change the 0 to 1 and make the island bigger, only one island with area = 4.

Example 3:

Input:

```
grid = [[1,1],[1,1]]
```

Output:

4

Explanation:

Can't change any 0 to 1, only one island with area = 4.

Constraints:

```
n == grid.length
```

```
n == grid[i].length
```

```
1 <= n <= 500
```

```
grid[i][j]
```

is either

0

or

1

Code Snippets

C++:

```
class Solution {
public:
    int largestIsland(vector<vector<int>>& grid) {

    }
};
```

Java:

```
class Solution {
    public int largestIsland(int[][] grid) {

    }
}
```

Python3:

```
class Solution:
    def largestIsland(self, grid: List[List[int]]) -> int:
```

Python:

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

JavaScript:

```
/**
 * @param {number[][]} grid
 * @return {number}
 */
```

```
var largestIsland = function(grid) {  
  
};
```

TypeScript:

```
function largestIsland(grid: number[][]): number {  
  
};
```

C#:

```
public class Solution {  
    public int LargestIsland(int[][] grid) {  
  
    }  
}
```

C:

```
int largestIsland(int** grid, int gridSize, int* gridColSize) {  
  
}
```

Go:

```
func largestIsland(grid [][]int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun largestIsland(grid: Array<IntArray>): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func largestIsland(_ grid: [[Int]]) -> Int {
```

```
}  
}
```

Rust:

```
impl Solution {  
    pub fn largest_island(grid: Vec<Vec<i32>>) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[][]} grid  
# @return {Integer}  
def largest_island(grid)  
  
end
```

PHP:

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

Dart:

```
class Solution {  
    int largestIsland(List<List<int>> grid) {  
  
    }  
}
```

Scala:

```

object Solution {
  def largestIsland(grid: Array[Array[Int]]): Int = {

  }
}

```

Elixir:

```

defmodule Solution do
  @spec largest_island(grid :: [[integer]]) :: integer
  def largest_island(grid) do

  end
end

```

Erlang:

```

-spec largest_island(Grid :: [[integer()]]) -> integer().
largest_island(Grid) ->
.

```

Racket:

```

(define/contract (largest-island grid)
  (-> (listof (listof exact-integer?)) exact-integer?)
)

```

Solutions

C++ Solution:

```

/*
 * Problem: Making A Large Island
 * Difficulty: Hard
 * Tags: array, graph, 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 largestIsland(vector<vector<int>>& grid) {

    }
};

```

Java Solution:

```

/**
 * Problem: Making A Large Island
 * Difficulty: Hard
 * Tags: array, graph, 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 largestIsland(int[][] grid) {

    }
}

```

Python3 Solution:

```

"""
Problem: Making A Large Island
Difficulty: Hard
Tags: array, graph, 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 largestIsland(self, grid: List[List[int]]) -> int:
        # TODO: Implement optimized solution
        pass

```


Python Solution:

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

JavaScript Solution:

```
/**
 * Problem: Making A Large Island
 * Difficulty: Hard
 * Tags: array, graph, 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[][]} grid
 * @return {number}
 */
var largestIsland = function(grid) {

};
```

TypeScript Solution:

```
/**
 * Problem: Making A Large Island
 * Difficulty: Hard
 * Tags: array, graph, 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 largestIsland(grid: number[][]): number {
```

```
};
```

C# Solution:

```
/*
 * Problem: Making A Large Island
 * Difficulty: Hard
 * Tags: array, graph, 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 LargestIsland(int[][] grid) {

    }
}
```

C Solution:

```
/*
 * Problem: Making A Large Island
 * Difficulty: Hard
 * Tags: array, graph, 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
 */

int largestIsland(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Making A Large Island
// Difficulty: Hard
```

```
// Tags: array, graph, 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 largestIsland(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
    fun largestIsland(grid: Array<IntArray>): Int {

    }
}
```

Swift Solution:

```
class Solution {
    func largestIsland(_ grid: [[Int]]) -> Int {

    }
}
```

Rust Solution:

```
// Problem: Making A Large Island
// Difficulty: Hard
// Tags: array, graph, 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 largest_island(grid: Vec<Vec<i32>>) -> i32 {

    }
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer}
def largest_island(grid)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $grid
     * @return Integer
     */
    function largestIsland($grid) {

    }

}
```

Dart Solution:

```
class Solution {
  int largestIsland(List<List<int>> grid) {

  }
}
```

Scala Solution:

```
object Solution {
  def largestIsland(grid: Array[Array[Int]]): Int = {

  }
}
```

Elixir Solution:

```
defmodule Solution do
  @spec largest_island(grid :: [[integer]]) :: integer
  def largest_island(grid) do
```

```
end  
end
```

Erlang Solution:

```
-spec largest_island(Grid :: [[integer()]]) -> integer().  
largest_island(Grid) ->  
.
```

Racket Solution:

```
(define/contract (largest-island grid)  
  (-> (listof (listof exact-integer?)) exact-integer?)  
)
```