Problem 1901: Find a Peak Element II

Problem Information

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

Problem Description

A

peak

element in a 2D grid is an element that is

strictly greater

than all of its

adjacent

neighbors to the left, right, top, and bottom.

Given a

0-indexed

m x n

matrix

mat

where

no two adjacent cells are equal

, find

any

peak element

mat[i][j]

and return

the length 2 array

[i,j]

.

You may assume that the entire matrix is surrounded by an

outer perimeter

with the value

-1

in each cell.

You must write an algorithm that runs in

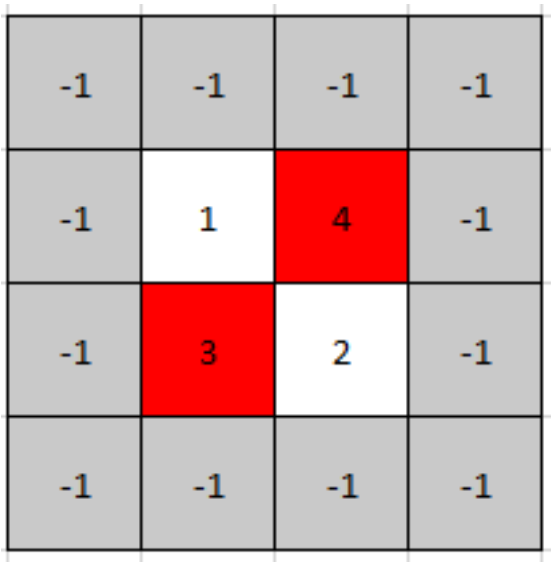
O(m log(n))

or

O(n log(m))

time.

Example 1:



Input:

mat = [[1,4],[3,2]]

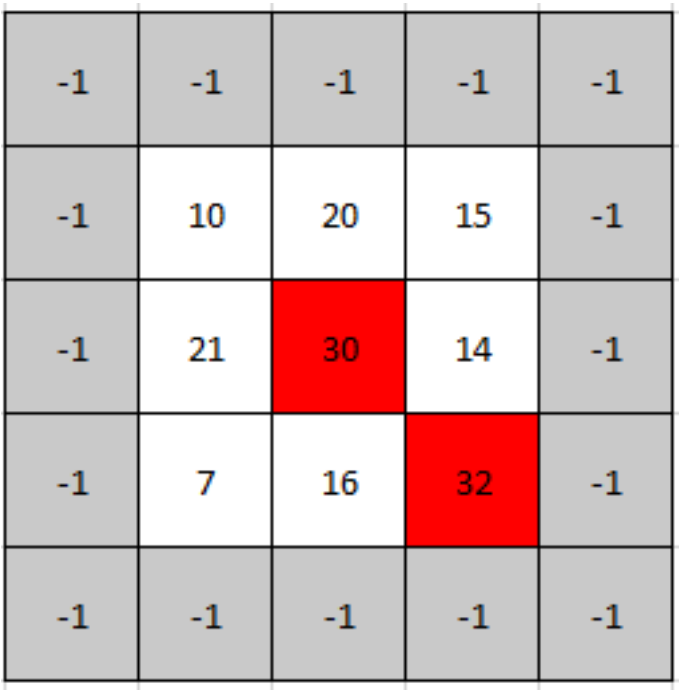
Output:

[0,1]

Explanation:

Both 3 and 4 are peak elements so [1,0] and [0,1] are both acceptable answers.

Example 2:



Input:

mat = [[10,20,15],[21,30,14],[7,16,32]]

Output:

[1,1]

Explanation:

Both 30 and 32 are peak elements so [1,1] and [2,2] are both acceptable answers.

Constraints:

m == mat.length

n == mat[i].length

1 <= m, n <= 500

1 <= mat[i][j] <= 10

5

No two adjacent cells are equal.

Code Snippets

C++:

```
class Solution {
public:
vector<int> findPeakGrid(vector<vector<int>>& mat) {

}
};
```

Java:

```
class Solution {
public int[] findPeakGrid(int[][] mat) {

}
}
```

Python3:

```
class Solution:
def findPeakGrid(self, mat: List[List[int]]) -> List[int]:
```

Python:

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

JavaScript:

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

```
var findPeakGrid = function(mat) {

};
```

TypeScript:

```
function findPeakGrid(mat: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] FindPeakGrid(int[][] mat) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findPeakGrid(int** mat, int matSize, int* matColSize, int* returnSize) {

}
```

Go:

```
func findPeakGrid(mat [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun findPeakGrid(mat: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func findPeakGrid(_ mat: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_peak_grid(mat: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} mat
# @return {Integer[]}
def find_peak_grid(mat)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $mat
* @return Integer[]
*/
function findPeakGrid($mat) {

}
}
```

Dart:

```
class Solution {
List<int> findPeakGrid(List<List<int>> mat) {

}
}
```

Scala:

```
object Solution {
def findPeakGrid(mat: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_peak_grid(mat :: [[integer]]) :: [integer]
def find_peak_grid(mat) do

end
end
```

Erlang:

```
-spec find_peak_grid(Mat :: [[integer()]]) -> [integer()].
find_peak_grid(Mat) ->
.
```

Racket:

```
(define/contract (find-peak-grid mat)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find a Peak Element II
* 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<int> findPeakGrid(vector<vector<int>>& mat) {

}
};
```

Java Solution:

```
/**
* Problem: Find a Peak Element II
* 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[] findPeakGrid(int[][] mat) {

}
}
```

Python3 Solution:

```
"""
Problem: Find a Peak Element II
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 findPeakGrid(self, mat: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find a Peak Element II
* 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[][]} mat
* @return {number[]}
*/
var findPeakGrid = function(mat) {

};
```

TypeScript Solution:

```
/**
* Problem: Find a Peak Element II
* 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 findPeakGrid(mat: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find a Peak Element II
* 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[] FindPeakGrid(int[][] mat) {

}
}
```

C Solution:

```
/*
* Problem: Find a Peak Element II
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findPeakGrid(int** mat, int matSize, int* matColSize, int* returnSize) {
```

```
}
```

Go Solution:

```
// Problem: Find a Peak Element II
// 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 findPeakGrid(mat [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findPeakGrid(mat: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func findPeakGrid(_ mat: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find a Peak Element II
// 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_peak_grid(mat: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} mat
# @return {Integer[]}
def find_peak_grid(mat)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $mat
* @return Integer[]
*/
function findPeakGrid($mat) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findPeakGrid(List<List<int>> mat) {

}
}
```

Scala Solution:

```
object Solution {
def findPeakGrid(mat: Array[Array[Int]]): Array[Int] = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_peak_grid(mat :: [[integer]]) :: [integer]
def find_peak_grid(mat) do

end
end
```

Erlang Solution:

```
-spec find_peak_grid(Mat :: [[integer()]]) -> [integer()].
find_peak_grid(Mat) ->
.
```

Racket Solution:

```
(define/contract (find-peak-grid mat)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```