Problem 2503: Maximum Number of Points From Grid Queries

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

grid

and an array

queries

of size

k

.

Find an array

answer

of size

k

such that for each integer

queries[i]

you start in the

top left

cell of the matrix and repeat the following process:

If

queries[i]

is

strictly

greater than the value of the current cell that you are in, then you get one point if it is your first time visiting this cell, and you can move to any

adjacent

cell in all

4

directions: up, down, left, and right.

Otherwise, you do not get any points, and you end this process.

After the process,

answer[i]

is the

maximum

number of points you can get.

Note

that for each query you are allowed to visit the same cell

multiple

times.

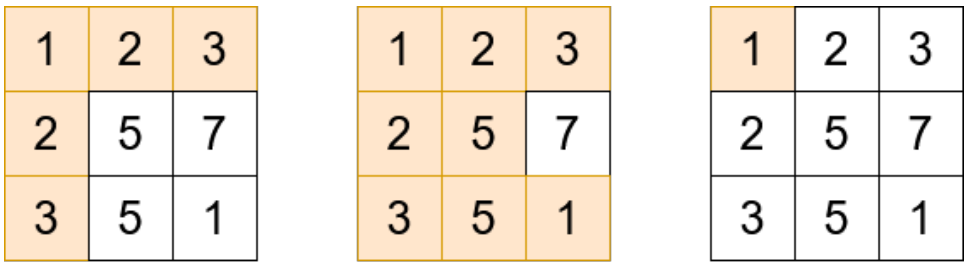
Return

the resulting array

answer

.

Example 1:



Input:

grid = [[1,2,3],[2,5,7],[3,5,1]], queries = [5,6,2]

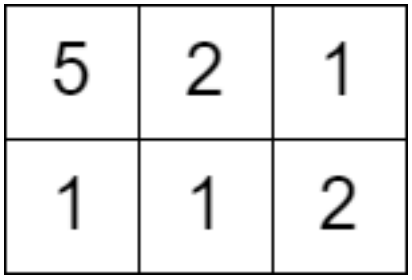
Output:

[5,8,1]

Explanation:

The diagrams above show which cells we visit to get points for each query.

Example 2:



Input:

grid = [[5,2,1],[1,1,2]], queries = [3]

Output:

[0]

Explanation:

We can not get any points because the value of the top left cell is already greater than or equal to 3.

Constraints:

m == grid.length

n == grid[i].length

2 <= m, n <= 1000

4 <= m * n <= 10

5

k == queries.length

1 <= k <= 10

4

1 <= grid[i][j], queries[i] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
vector<int> maxPoints(vector<vector<int>>& grid, vector<int>& queries) {

}
};
```

Java:

```
class Solution {
public int[] maxPoints(int[][] grid, int[] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} grid
* @param {number[]} queries
* @return {number[]}
*/
var maxPoints = function(grid, queries) {

};
```

TypeScript:

```
function maxPoints(grid: number[][], queries: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] MaxPoints(int[][] grid, int[] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maxPoints(int** grid, int gridSize, int* gridColSize, int* queries, int
queriesSize, int* returnSize) {

}
```

Go:

```
func maxPoints(grid [][]int, queries []int) []int {

}
```

Kotlin:

```
class Solution {
fun maxPoints(grid: Array<IntArray>, queries: IntArray): IntArray {
```

```
}
}
```

Swift:

```
class Solution {
func maxPoints(_ grid: [[Int]], _ queries: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn max_points(grid: Vec<Vec<i32>>, queries: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer[]} queries
# @return {Integer[]}
def max_points(grid, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer[] $queries
* @return Integer[]
*/
function maxPoints($grid, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> maxPoints(List<List<int>> grid, List<int> queries) {

}
}
```

Scala:

```
object Solution {
def maxPoints(grid: Array[Array[Int]], queries: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_points(grid :: [[integer]], queries :: [integer]) :: [integer]
def max_points(grid, queries) do

end
end
```

Erlang:

```
-spec max_points(Grid :: [[integer()]], Queries :: [integer()]) ->
[integer()].
max_points(Grid, Queries) ->
.
```

Racket:

```
(define/contract (max-points grid queries)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Points From Grid Queries
* Difficulty: Hard
* Tags: array, graph, sort, search, queue, heap
*
* 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> maxPoints(vector<vector<int>>& grid, vector<int>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Points From Grid Queries
* Difficulty: Hard
* Tags: array, graph, sort, search, queue, heap
*
* 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[] maxPoints(int[][] grid, int[] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Points From Grid Queries
Difficulty: Hard
Tags: array, graph, sort, search, queue, heap
```

```
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 maxPoints(self, grid: List[List[int]], queries: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Points From Grid Queries
* Difficulty: Hard
* Tags: array, graph, sort, search, queue, heap
*
* 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
* @param {number[]} queries
* @return {number[]}
*/
var maxPoints = function(grid, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Points From Grid Queries
* Difficulty: Hard
* Tags: array, graph, sort, search, queue, heap
*
* 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 maxPoints(grid: number[][], queries: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Points From Grid Queries
* Difficulty: Hard
* Tags: array, graph, sort, search, queue, heap
*
* 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[] MaxPoints(int[][] grid, int[] queries) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Points From Grid Queries
* Difficulty: Hard
* Tags: array, graph, sort, search, queue, heap
*
* 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* maxPoints(int** grid, int gridSize, int* gridColSize, int* queries, int
queriesSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Points From Grid Queries
// Difficulty: Hard
// Tags: array, graph, sort, search, queue, heap
//
// 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 maxPoints(grid [][]int, queries []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun maxPoints(grid: Array<IntArray>, queries: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func maxPoints(_ grid: [[Int]], _ queries: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Points From Grid Queries
// Difficulty: Hard
// Tags: array, graph, sort, search, queue, heap
//
// 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 max_points(grid: Vec<Vec<i32>>, queries: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer[]} queries
# @return {Integer[]}
def max_points(grid, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer[] $queries
* @return Integer[]
*/
function maxPoints($grid, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> maxPoints(List<List<int>> grid, List<int> queries) {

}
}
```

Scala Solution:

```
object Solution {
def maxPoints(grid: Array[Array[Int]], queries: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_points(grid :: [[integer]], queries :: [integer]) :: [integer]
def max_points(grid, queries) do

end
end
```

Erlang Solution:

```
-spec max_points(Grid :: [[integer()]], Queries :: [integer()]) ->
[integer()].
max_points(Grid, Queries) ->
.
```

Racket Solution:

```
(define/contract (max-points grid queries)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?))
)
```