Problem 2711: Difference of Number of Distinct Values on Diagonals

Problem Information

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

Problem Description

Given a 2D

grid

of size

m x n

, you should find the matrix

answer

of size

m x n

.

The cell

answer[r][c]

is calculated by looking at the diagonal values of the cell

grid[r][c]

:

Let

leftAbove[r][c]

be the number of

distinct

values on the diagonal to the left and above the cell

grid[r][c]

not including the cell

grid[r][c]

itself.

Let

rightBelow[r][c]

be the number of

distinct

values on the diagonal to the right and below the cell

grid[r][c]

, not including the cell

grid[r][c]

itself.

Then

answer[r][c] = |leftAbove[r][c] - rightBelow[r][c]|

.

A

matrix diagonal

is a diagonal line of cells starting from some cell in either the topmost row or leftmost column and going in the bottom-right direction until the end of the matrix is reached.

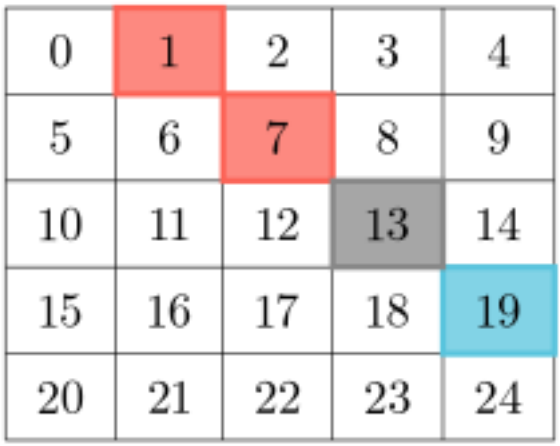
For example, in the below diagram the diagonal is highlighted using the cell with indices

(2, 3)

colored gray:

Red-colored cells are left and above the cell.

Blue-colored cells are right and below the cell.



Return the matrix

answer

.

Example 1:

Input:

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

Output:

Output: [[1,1,0],[1,0,1],[0,1,1]]

Explanation:

To calculate the

answer

cells:

answer

left-above elements

leftAbove

right-below elements

rightBelow

|leftAbove - rightBelow|

[0][0]

[]

0

[grid[1][1], grid[2][2]]

|{1, 1}| = 1

1

[0][1]

[]

0

[grid[1][2]]

|{5}| = 1

1

[0][2]

[]

0

[]

0

0

[1][0]

[]

0

[grid[2][1]]

|{2}| = 1

1

[1][1]

[grid[0][0]]

|{1}| = 1

[grid[2][2]]

|{1}| = 1

0

[1][2]

[grid[0][1]]

|{2}| = 1

[]

0

1

[2][0]

[]

0

[]

0

0

[2][1]

[grid[1][0]]

|{3}| = 1

[]

0

1

[2][2]

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

|{1, 1}| = 1

[]

0

1

Example 2:

Input:

grid = [[1]]

Output:

Output: [[0]]

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n, grid[i][j] <= 50

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
*/
var differenceOfDistinctValues = function(grid) {

};
```

TypeScript:

```
function differenceOfDistinctValues(grid: number[][]): number[][] {

};
```

C#:

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

}
}
```

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** differenceOfDistinctValues(int** grid, int gridSize, int* gridColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func differenceOfDistinctValues(grid [][]int) [][]int {

}
```

Kotlin:

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

```
}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (difference-of-distinct-values grid)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Difference of Number of Distinct Values on Diagonals
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Difference of Number of Distinct Values on Diagonals
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int[][] differenceOfDistinctValues(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Difference of Number of Distinct Values on Diagonals
Difficulty: Medium
Tags: array, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Difference of Number of Distinct Values on Diagonals
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[][]} grid
* @return {number[][]}
*/
var differenceOfDistinctValues = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Difference of Number of Distinct Values on Diagonals
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function differenceOfDistinctValues(grid: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Difference of Number of Distinct Values on Diagonals
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int[][] DifferenceOfDistinctValues(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Difference of Number of Distinct Values on Diagonals
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

/**
* 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** differenceOfDistinctValues(int** grid, int gridSize, int* gridColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Difference of Number of Distinct Values on Diagonals
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func differenceOfDistinctValues(grid [][]int) [][]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
```

```
}
```

Rust Solution:

```
// Problem: Difference of Number of Distinct Values on Diagonals
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn difference_of_distinct_values(grid: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (difference-of-distinct-values grid)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```