Problem 3567: Minimum Absolute Difference in Sliding Submatrix

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

grid

and an integer

k

.

For every contiguous

k x k

submatrix

of

grid

, compute the

minimum absolute

difference between any two

distinct

values within that

submatrix

.

Return a 2D array

ans

of size

(m - k + 1) x (n - k + 1)

, where

ans[i][j]

is the minimum absolute difference in the submatrix whose top-left corner is

(i, j)

in

grid

.

Note

: If all elements in the submatrix have the same value, the answer will be 0.

A submatrix

(x1, y1, x2, y2)

is a matrix that is formed by choosing all cells

matrix[x][y]

where

x1 <= x <= x2

and

y1 <= y <= y2

.

Example 1:

Input:

grid = [[1,8],[3,-2]], k = 2

Output:

[[2]]

Explanation:

There is only one possible

k x k

submatrix:

[[1, 8], [3, -2]]

.

Distinct values in the submatrix are

[1, 8, 3, -2]

.

The minimum absolute difference in the submatrix is

|1 - 3| = 2

. Thus, the answer is

[[2]]

.

Example 2:

Input:

grid = [[3,-1]], k = 1

Output:

[[0,0]]

Explanation:

Both

k x k

submatrix has only one distinct element.

Thus, the answer is

[[0, 0]]

.

Example 3:

Input:

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

Output:

[[1,2]]

Explanation:

There are two possible

k × k

submatrix:

Starting at

(0, 0)

:

[[1, -2], [2, 3]]

.

Distinct values in the submatrix are

[1, -2, 2, 3]

.

The minimum absolute difference in the submatrix is

|1 - 2| = 1

.

Starting at

(0, 1)

:

[[-2, 3], [3, 5]]

.

Distinct values in the submatrix are

[-2, 3, 5]

.

The minimum absolute difference in the submatrix is

|3 - 5| = 2

.

Thus, the answer is

[[1, 2]]

.

Constraints:

1 <= m == grid.length <= 30

1 <= n == grid[i].length <= 30

-10

5

<= grid[i][j] <= 10

5

1 <= k <= min(m, n)

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> minAbsDiff(vector<vector<int>>& grid, int k) {

}
};
```

Java:

```
class Solution {
public int[][] minAbsDiff(int[][] grid, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minAbsDiff(grid: number[][], k: number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] MinAbsDiff(int[][] grid, int k) {

}
}
```

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** minAbsDiff(int** grid, int gridSize, int* gridColSize, int k, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func minAbsDiff(grid [][]int, k int) [][]int {

}
```

Kotlin:

```
class Solution {
fun minAbsDiff(grid: Array<IntArray>, k: Int): Array<IntArray> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer} k
# @return {Integer[][]}
def min_abs_diff(grid, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $k
* @return Integer[][]
*/
function minAbsDiff($grid, $k) {


}
}
```

Dart:

```
class Solution {
List<List<int>> minAbsDiff(List<List<int>> grid, int k) {

}
}
```

Scala:

```
object Solution {
def minAbsDiff(grid: Array[Array[Int]], k: Int): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_abs_diff(grid :: [[integer]], k :: integer) :: [[integer]]
def min_abs_diff(grid, k) do

end
end
```

Erlang:

```
-spec min_abs_diff(Grid :: [[integer()]], K :: integer()) -> [[integer()]].
min_abs_diff(Grid, K) ->
.
```

Racket:

```
(define/contract (min-abs-diff grid k)
(-> (listof (listof exact-integer?)) exact-integer? (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Absolute Difference in Sliding Submatrix
* Difficulty: Medium
* Tags: array, sort
*
* 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>> minAbsDiff(vector<vector<int>>& grid, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Absolute Difference in Sliding Submatrix
* Difficulty: Medium
* Tags: array, sort
*
* 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[][] minAbsDiff(int[][] grid, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Absolute Difference in Sliding Submatrix
Difficulty: Medium
Tags: array, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Absolute Difference in Sliding Submatrix
* Difficulty: Medium
* Tags: array, sort
*
* 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} k
* @return {number[][]}
*/
```

```
var minAbsDiff = function(grid, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Absolute Difference in Sliding Submatrix
* Difficulty: Medium
* Tags: array, sort
*
* 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 minAbsDiff(grid: number[][], k: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Minimum Absolute Difference in Sliding Submatrix
* Difficulty: Medium
* Tags: array, sort
*
* 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[][] MinAbsDiff(int[][] grid, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Absolute Difference in Sliding Submatrix
* Difficulty: Medium
* Tags: array, sort
*
* 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** minAbsDiff(int** grid, int gridSize, int* gridColSize, int k, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Minimum Absolute Difference in Sliding Submatrix
// Difficulty: Medium
// Tags: array, sort
//
// 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 minAbsDiff(grid [][]int, k int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun minAbsDiff(grid: Array<IntArray>, k: Int): Array<IntArray> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Absolute Difference in Sliding Submatrix
// Difficulty: Medium
// Tags: array, sort
//
// 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 min_abs_diff(grid: Vec<Vec<i32>>, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer} k
# @return {Integer[][]}
def min_abs_diff(grid, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $k
* @return Integer[][]
*/
function minAbsDiff($grid, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> minAbsDiff(List<List<int>> grid, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minAbsDiff(grid: Array[Array[Int]], k: Int): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_abs_diff(grid :: [[integer]], k :: integer) :: [[integer]]
def min_abs_diff(grid, k) do

end
end
```

Erlang Solution:

```
-spec min_abs_diff(Grid :: [[integer()]], K :: integer()) -> [[integer()]].
min_abs_diff(Grid, K) ->
.
```

Racket Solution:

```
(define/contract (min-abs-diff grid k)
(-> (listof (listof exact-integer?)) exact-integer? (listof (listof
exact-integer?)))
)
```