Problem 1314: Matrix Block Sum

Problem Information

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

Problem Description

Given a

m x n

matrix

mat

and an integer

k

, return

a matrix

answer

where each

answer[i][j]

is the sum of all elements

mat[r][c]

for

:

i - k <= r <= i + k,

j - k <= c <= j + k

, and

(r, c)

is a valid position in the matrix.

Example 1:

Input:

mat = [[1,2,3],[4,5,6],[7,8,9]], k = 1

Output:

[[12,21,16],[27,45,33],[24,39,28]]

Example 2:

Input:

mat = [[1,2,3],[4,5,6],[7,8,9]], k = 2

Output:

[[45,45,45],[45,45,45],[45,45,45]]

Constraints:

m == mat.length

n == mat[i].length

1 <= m, n, k <= 100

1 <= mat[i][j] <= 100

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> matrixBlockSum(vector<vector<int>>& mat, int k) {

}
};
```

Java:

```
class Solution {
public int[][] matrixBlockSum(int[][] mat, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function matrixBlockSum(mat: number[][], k: number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] MatrixBlockSum(int[][] mat, 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** matrixBlockSum(int** mat, int matSize, int* matColSize, int k, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func matrixBlockSum(mat [][]int, k int) [][]int {

}
```

Kotlin:

```
class Solution {
fun matrixBlockSum(mat: Array<IntArray>, k: Int): Array<IntArray> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[][]} mat
# @param {Integer} k
# @return {Integer[][]}
def matrix_block_sum(mat, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $mat
* @param Integer $k
* @return Integer[][]
*/
function matrixBlockSum($mat, $k) {

}
```

```
}
```

Dart:

```
class Solution {
List<List<int>> matrixBlockSum(List<List<int>> mat, int k) {

}
}
```

Scala:

```
object Solution {
def matrixBlockSum(mat: Array[Array[Int]], k: Int): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec matrix_block_sum(mat :: [[integer]], k :: integer) :: [[integer]]
def matrix_block_sum(mat, k) do

end
end
```

Erlang:

```
-spec matrix_block_sum(Mat :: [[integer()]], K :: integer()) ->
[[integer()]].
matrix_block_sum(Mat, K) ->
.
```

Racket:

```
(define/contract (matrix-block-sum mat k)
(-> (listof (listof exact-integer?)) exact-integer? (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Matrix Block Sum
* Difficulty: Medium
* Tags: array
*
* 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>> matrixBlockSum(vector<vector<int>>& mat, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Matrix Block Sum
* Difficulty: Medium
* Tags: array
*
* 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[][] matrixBlockSum(int[][] mat, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Matrix Block Sum
```

```
Difficulty: Medium
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Matrix Block Sum
* Difficulty: Medium
* Tags: array
*
* 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
* @param {number} k
* @return {number[][]}
*/
var matrixBlockSum = function(mat, k) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Matrix Block Sum
* Difficulty: Medium
* Tags: array
*
* 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 matrixBlockSum(mat: number[][], k: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Matrix Block Sum
* Difficulty: Medium
* Tags: array
*
* 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[][] MatrixBlockSum(int[][] mat, int k) {

}
}
```

C Solution:

```
/*
* Problem: Matrix Block Sum
```

```
* Difficulty: Medium
* Tags: array
*
* 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** matrixBlockSum(int** mat, int matSize, int* matColSize, int k, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Matrix Block Sum
// Difficulty: Medium
// Tags: array
//
// 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 matrixBlockSum(mat [][]int, k int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun matrixBlockSum(mat: Array<IntArray>, k: Int): Array<IntArray> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Matrix Block Sum
// Difficulty: Medium
// Tags: array
//
// 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 matrix_block_sum(mat: Vec<Vec<i32>>, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} mat
# @param {Integer} k
# @return {Integer[][]}
def matrix_block_sum(mat, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $mat
* @param Integer $k
* @return Integer[][]
*/
```

```
function matrixBlockSum($mat, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> matrixBlockSum(List<List<int>> mat, int k) {

}
}
```

Scala Solution:

```
object Solution {
def matrixBlockSum(mat: Array[Array[Int]], k: Int): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec matrix_block_sum(mat :: [[integer]], k :: integer) :: [[integer]]
def matrix_block_sum(mat, k) do

end
end
```

Erlang Solution:

```
-spec matrix_block_sum(Mat :: [[integer()]], K :: integer()) ->
[[integer()]].
matrix_block_sum(Mat, K) ->
.
```

Racket Solution:

```
(define/contract (matrix-block-sum mat k)
(-> (listof (listof exact-integer?)) exact-integer? (listof (listof
```

```
exact-integer?)))
)
```