Problem 1572: Matrix Diagonal Sum

Problem Information

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

Problem Description

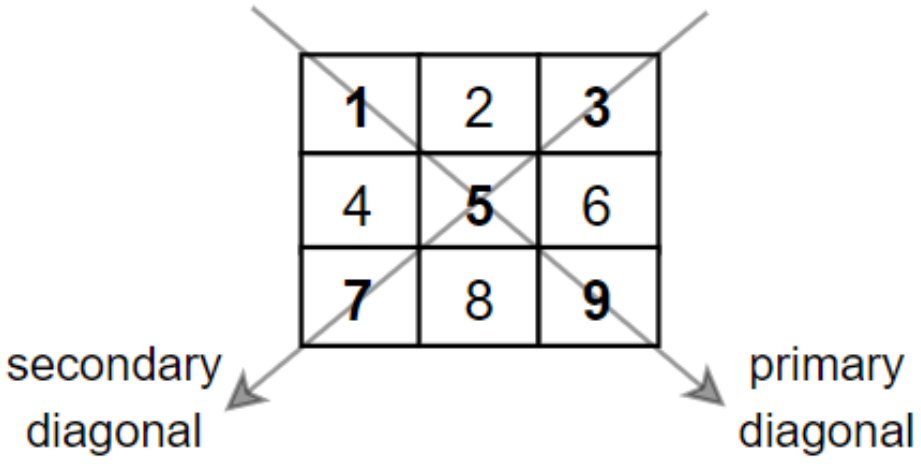
Given a square matrix

mat

, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

Example 1:



Input:

mat = [[

1

,2,

3

], [4,

5

,6], [

7

,8,

9

]]

Output:

25

Explanation:

Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25 Notice that element mat[1][1] = 5 is counted only once.

Example 2:

Input:

mat = [[

1

,1,1,

1

], [1,

1

,

1

,1], [1,

1

,

1

,1], [

1

,1,1,

1

]]

Output:

8

Example 3:

Input:

mat = [[

5

]]

Output:

5

Constraints:

n == mat.length == mat[i].length

1 <= n <= 100

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

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int diagonalSum(int** mat, int matSize, int* matColSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (diagonal-sum mat)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Matrix Diagonal Sum
* Difficulty: Easy
* 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 diagonalSum(vector<vector<int>>& mat) {

}
};
```

Java Solution:

```
/**
* Problem: Matrix Diagonal Sum
* Difficulty: Easy
* 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 diagonalSum(int[][] mat) {

}
}
```

Python3 Solution:

```
"""
Problem: Matrix Diagonal Sum
Difficulty: Easy
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 diagonalSum(self, mat: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Matrix Diagonal Sum
* Difficulty: Easy
* 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
* @return {number}
*/
var diagonalSum = function(mat) {

};
```

TypeScript Solution:

```
/**
* Problem: Matrix Diagonal Sum
* Difficulty: Easy
* 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 diagonalSum(mat: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Matrix Diagonal Sum
* Difficulty: Easy
* 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 DiagonalSum(int[][] mat) {

}
}
```

C Solution:

```
/*
* Problem: Matrix Diagonal Sum
* Difficulty: Easy
* 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
```

```
*/

int diagonalSum(int** mat, int matSize, int* matColSize) {

}
```

Go Solution:

```
// Problem: Matrix Diagonal Sum
// Difficulty: Easy
// 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 diagonalSum(mat [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Matrix Diagonal Sum
// Difficulty: Easy
// 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 diagonal_sum(mat: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (diagonal-sum mat)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```