Problem 2852: Sum of Remoteness of All Cells

Problem Information

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

Problem Description

You are given a

0-indexed

matrix

grid

of order

n * n

. Each cell in this matrix has a value

grid[i][j]

, which is either a

positive

integer or

-1

representing a blocked cell.

You can move from a non-blocked cell to any non-blocked cell that shares an edge.

For any cell

(i, j)

, we represent its

remoteness

as

R[i][j]

which is defined as the following:

If the cell

(i, j)

is a

non-blocked

cell,

R[i][j]

is the sum of the values

grid[x][y]

such that there is

no path

from the

non-blocked

cell

(x, y)

to the cell

(i, j)

.

For blocked cells,

R[i][j] == 0

.

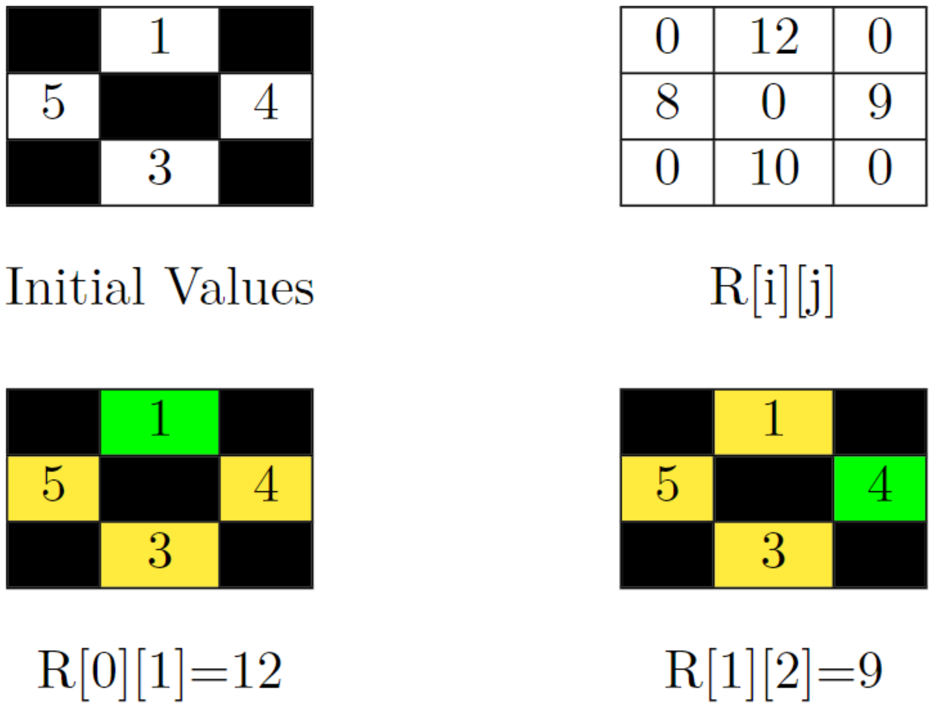
Return

the sum of

R[i][j]

over all cells.

Example 1:



Input:

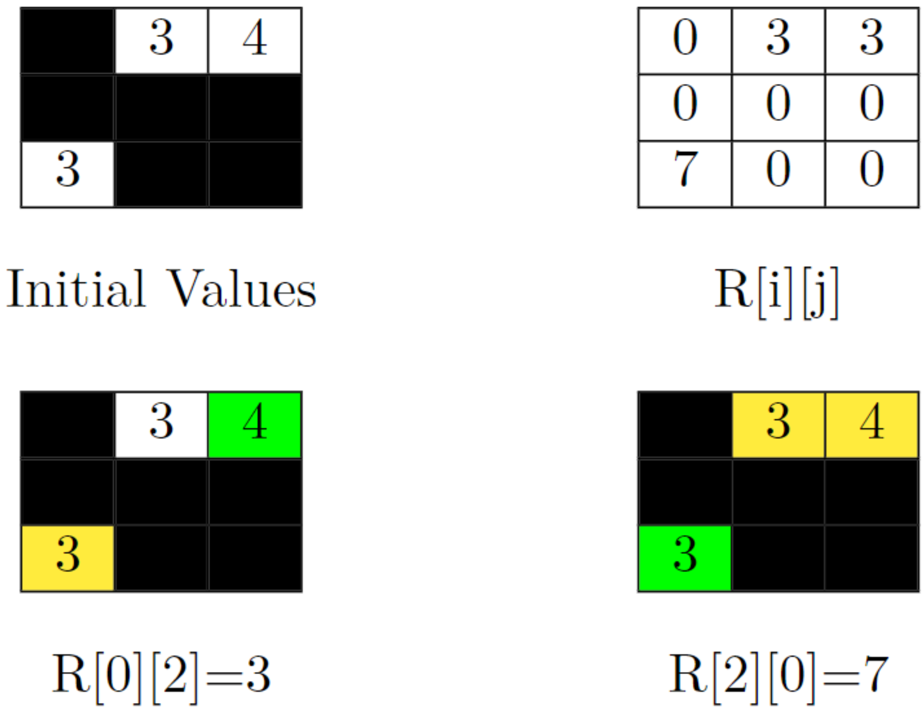
grid = [[-1,1,-1],[5,-1,4],[-1,3,-1]]

Output:

39

Explanation:

In the picture above, there are four grids. The top-left grid contains the initial values in the grid. Blocked cells are colored black, and other cells get their values as it is in the input. In the top-right grid, you can see the value of R[i][j] for all cells. So the answer would be the sum of them. That is: 0 + 12 + 0 + 8 + 0 + 9 + 0 + 10 + 0 = 39. Let's jump on the bottom-left grid in the above picture and calculate R[0][1] (the target cell is colored green). We should sum up the value of cells that can't be reached by the cell (0, 1). These cells are colored yellow in this grid. So R[0][1] = 5 + 4 + 3 = 12. Now let's jump on the bottom-right grid in the above picture and calculate R[1][2] (the target cell is colored green). We should sum up the value of cells that can't be reached by the cell (1, 2). These cells are colored yellow in this grid. So R[1][2] = 1 + 5 + 3 = 9.



Example 2:

Input:

grid = [[-1,3,4],[-1,-1,-1],[3,-1,-1]]

Output:

13

Explanation:

In the picture above, there are four grids. The top-left grid contains the initial values in the grid. Blocked cells are colored black, and other cells get their values as it is in the input. In the top-right grid, you can see the value of R[i][j] for all cells. So the answer would be the sum of them. That is: 3 + 3 + 0 + 0 + 0 + 0 + 7 + 0 + 0 = 13. Let's jump on the bottom-left grid in the above picture and calculate R[0][2] (the target cell is colored green). We should sum up the value of cells that can't be reached by the cell (0, 2). This cell is colored yellow in this grid. So R[0][2] = 3. Now let's jump on the bottom-right grid in the above picture and calculate R[2][0] (the target cell is colored green). We should sum up the value of cells that can't be reached by the cell (2, 0). These cells are colored yellow in this grid. So R[2][0] = 3 + 4 = 7.

Example 3:

Input:

grid = [[1]]

Output:

0

Explanation:

Since there are no other cells than (0, 0), R[0][0] is equal to 0. So the sum of R[i][j] over all cells would be 0.

Constraints:

1 <= n <= 300

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

6

or

grid[i][j] == -1

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long sumRemoteness(int[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long SumRemoteness(int[][] grid) {

}
}
```

C:

```
long long sumRemoteness(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func sumRemoteness(grid [][]int) int64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn sum_remoteness(grid: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sum-remoteness grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Remoteness of All Cells
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* 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:
long long sumRemoteness(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Remoteness of All Cells
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* 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 long sumRemoteness(int[][] grid) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Sum of Remoteness of All Cells
Difficulty: Medium
Tags: array, graph, hash, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Remoteness of All Cells
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* 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 sumRemoteness = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Remoteness of All Cells
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* 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 sumRemoteness(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Remoteness of All Cells
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* 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 long SumRemoteness(int[][] grid) {

}
```

```
}
```

C Solution:

```c
/*
* Problem: Sum of Remoteness of All Cells
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

long long sumRemoteness(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```go
// Problem: Sum of Remoteness of All Cells
// Difficulty: Medium
// Tags: array, graph, hash, search
//
// 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 sumRemoteness(grid [][]int) int64 {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun sumRemoteness(grid: Array<IntArray>): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Remoteness of All Cells
// Difficulty: Medium
// Tags: array, graph, hash, search
//
// 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 sum_remoteness(grid: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sum-remoteness grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```