Problem 542: 01 Matrix

Problem Information

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

Problem Description

Given an

m x n

binary matrix

mat

, return

the distance of the nearest

0

for each cell

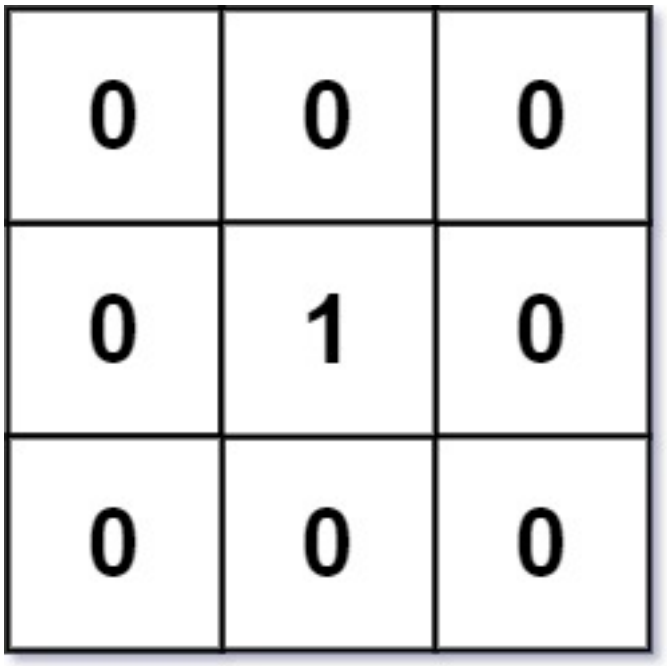
.

The distance between two cells sharing a common edge is

1

.

Example 1:



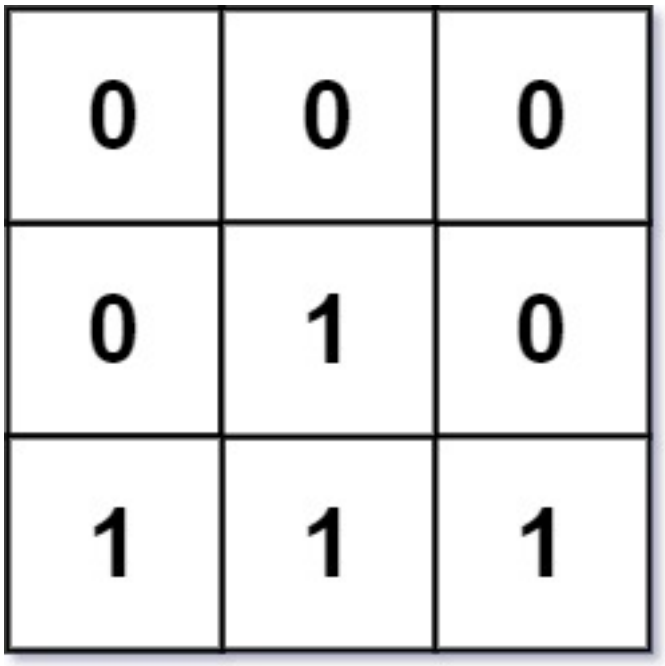
Input:

mat = [[0,0,0],[0,1,0],[0,0,0]]

Output:

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

Example 2:



Input:

mat = [[0,0,0],[0,1,0],[1,1,1]]

Output:

[[0,0,0],[0,1,0],[1,2,1]]

Constraints:

m == mat.length

n == mat[i].length

1 <= m, n <= 10

4

1 <= m * n <= 10

4

mat[i][j]

is either

0

or

1

.

There is at least one

0

in

mat

.

Note:

This question is the same as 1765:

https://leetcode.com/problems/map-of-highest-peak/

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

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** updateMatrix(int** mat, int matSize, int* matColSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (update-matrix mat)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: 01 Matrix
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: 01 Matrix
```

```
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int[][] updateMatrix(int[][] mat) {

}
}
```

Python3 Solution:

```
"""
Problem: 01 Matrix
Difficulty: Medium
Tags: array, dp, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: 01 Matrix
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: 01 Matrix
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function updateMatrix(mat: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: 01 Matrix
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int[][] UpdateMatrix(int[][] mat) {

}
}
```

C Solution:

```
/*
* Problem: 01 Matrix
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* 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** updateMatrix(int** mat, int matSize, int* matColSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: 01 Matrix
// Difficulty: Medium
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(n) or O(n * m) for DP table

func updateMatrix(mat [][]int) [][]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: 01 Matrix
// Difficulty: Medium
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (update-matrix mat)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```