Problem 1034: Coloring A Border

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

grid

, and three integers

row

,

col

, and

color

. Each value in the grid represents the color of the grid square at that location.

Two squares are called

adjacent

if they are next to each other in any of the 4 directions.

Two squares belong to the same

connected component

if they have the same color and they are adjacent.

The

border of a connected component

is all the squares in the connected component that are either adjacent to (at least) a square not in the component, or on the boundary of the grid (the first or last row or column).

You should color the

border

of the

connected component

that contains the square

grid[row][col]

with

color

.

Return

the final grid

.

Example 1:

Input:

grid = [[1,1],[1,2]], row = 0, col = 0, color = 3

Output:

[[3,3],[3,2]]

Example 2:

Input:

grid = [[1,2,2],[2,3,2]], row = 0, col = 1, color = 3

Output:

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

Example 3:

Input:

grid = [[1,1,1],[1,1,1],[1,1,1]], row = 1, col = 1, color = 2

Output:

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

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 50

1 <= grid[i][j], color <= 1000

0 <= row < m

0 <= col < n

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> colorBorder(vector<vector<int>>& grid, int row, int col,
int color) {

}
};
```

Java:

```
class Solution {
public int[][] colorBorder(int[][] grid, int row, int col, int color) {

}
}
```

Python3:

```
class Solution:
def colorBorder(self, grid: List[List[int]], row: int, col: int, color: int)
-> List[List[int]]:
```

Python:

```
class Solution(object):
def colorBorder(self, grid, row, col, color):
"""
:type grid: List[List[int]]
:type row: int
:type col: int
:type color: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @param {number} row
* @param {number} col
* @param {number} color
* @return {number[][]}
*/
var colorBorder = function(grid, row, col, color) {

};
```

TypeScript:

```
function colorBorder(grid: number[][], row: number, col: number, color:
number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] ColorBorder(int[][] grid, int row, int col, int color) {

}
}
```

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** colorBorder(int** grid, int gridSize, int* gridColSize, int row, int
col, int color, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func colorBorder(grid [][]int, row int, col int, color int) [][]int {

}
```

Kotlin:

```
class Solution {
fun colorBorder(grid: Array<IntArray>, row: Int, col: Int, color: Int):
Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func colorBorder(_ grid: [[Int]], _ row: Int, _ col: Int, _ color: Int) ->
[[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn color_border(grid: Vec<Vec<i32>>, row: i32, col: i32, color: i32) ->
Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer} row
# @param {Integer} col
# @param {Integer} color
# @return {Integer[][]}
def color_border(grid, row, col, color)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $row
* @param Integer $col
* @param Integer $color
* @return Integer[][]
*/
function colorBorder($grid, $row, $col, $color) {

}
}
```

Dart:

```
class Solution {
List<List<int>> colorBorder(List<List<int>> grid, int row, int col, int
color) {

}
}
```

Scala:

```
object Solution {
def colorBorder(grid: Array[Array[Int]], row: Int, col: Int, color: Int):
Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec color_border(grid :: [[integer]], row :: integer, col :: integer, color
:: integer) :: [[integer]]
def color_border(grid, row, col, color) do

end
end
```

Erlang:

```
-spec color_border(Grid :: [[integer()]], Row :: integer(), Col :: integer(),
Color :: integer()) -> [[integer()]].
color_border(Grid, Row, Col, Color) ->
.
```

Racket:

```
(define/contract (color-border grid row col color)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Coloring A Border
* Difficulty: Medium
* Tags: array, search
*
* 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>> colorBorder(vector<vector<int>>& grid, int row, int col,
int color) {

}
};
```

Java Solution:

```
/**
* Problem: Coloring A Border
* Difficulty: Medium
* Tags: array, search
*
```

```
* 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[][] colorBorder(int[][] grid, int row, int col, int color) {

}
}
```

Python3 Solution:

```
"""
Problem: Coloring A Border
Difficulty: Medium
Tags: array, search

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 colorBorder(self, grid: List[List[int]], row: int, col: int, color: int)
-> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def colorBorder(self, grid, row, col, color):
"""
:type grid: List[List[int]]
:type row: int
:type col: int
:type color: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Coloring A Border
* Difficulty: Medium
* Tags: array, search
*
* 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} row
* @param {number} col
* @param {number} color
* @return {number[][]}
*/
var colorBorder = function(grid, row, col, color) {

};
```

TypeScript Solution:

```
/**
* Problem: Coloring A Border
* Difficulty: Medium
* Tags: array, search
*
* 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 colorBorder(grid: number[][], row: number, col: number, color:
number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Coloring A Border
* Difficulty: Medium
* Tags: array, search
*
* 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[][] ColorBorder(int[][] grid, int row, int col, int color) {

}
}
```

C Solution:

```
/*
* Problem: Coloring A Border
* Difficulty: Medium
* Tags: array, search
*
* 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** colorBorder(int** grid, int gridSize, int* gridColSize, int row, int
col, int color, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Coloring A Border
// Difficulty: Medium
// Tags: array, search
//
// 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 colorBorder(grid [][]int, row int, col int, color int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun colorBorder(grid: Array<IntArray>, row: Int, col: Int, color: Int):
Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func colorBorder(_ grid: [[Int]], _ row: Int, _ col: Int, _ color: Int) ->
[[Int]] {

}
}
```

Rust Solution:

```
// Problem: Coloring A Border
// Difficulty: Medium
// Tags: array, search
//
// 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 color_border(grid: Vec<Vec<i32>>, row: i32, col: i32, color: i32) ->
```

```
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer} row
# @param {Integer} col
# @param {Integer} color
# @return {Integer[][]}
def color_border(grid, row, col, color)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $row
* @param Integer $col
* @param Integer $color
* @return Integer[][]
*/
function colorBorder($grid, $row, $col, $color) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> colorBorder(List<List<int>> grid, int row, int col, int
color) {

}
}
```

Scala Solution:

```
object Solution {
def colorBorder(grid: Array[Array[Int]], row: Int, col: Int, color: Int):
Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec color_border(grid :: [[integer]], row :: integer, col :: integer, color
:: integer) :: [[integer]]
def color_border(grid, row, col, color) do

end
end
```

Erlang Solution:

```
-spec color_border(Grid :: [[integer()]], Row :: integer(), Col :: integer(),
Color :: integer()) -> [[integer()]].
color_border(Grid, Row, Col, Color) ->
.
```

Racket Solution:

```
(define/contract (color-border grid row col color)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer? (listof (listof exact-integer?)))
)
```