Problem 2556: Disconnect Path in a Binary Matrix by at Most One Flip

Problem Information

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

Problem Description

You are given a

0-indexed

m x n

binary

matrix

grid

. You can move from a cell

(row, col)

to any of the cells

(row + 1, col)

or

(row, col + 1)

that has the value

1

. The matrix is

disconnected

if there is no path from

(0, 0)

to

(m - 1, n - 1)

.

You can flip the value of

at most one

(possibly none) cell. You

cannot flip

the cells

(0, 0)

and

(m - 1, n - 1)

.

Return

true

if it is possible to make the matrix disconnect or

false

otherwise

.

Note

that flipping a cell changes its value from

0

to

1

or from

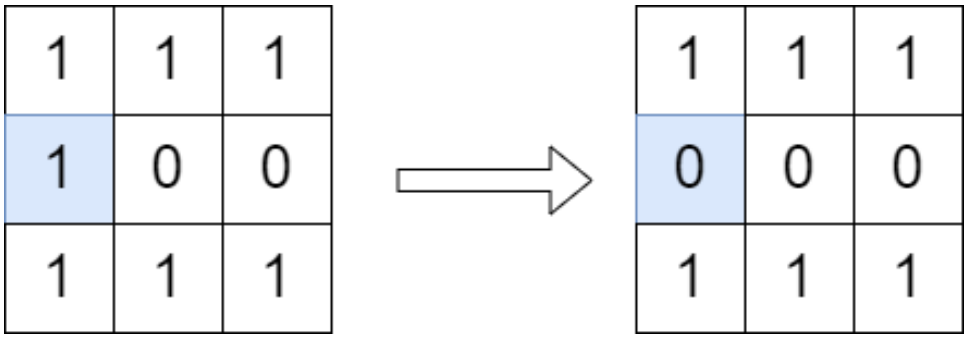
1

to

0

.

Example 1:



Input:

grid = [[1,1,1],[1,0,0],[1,1,1]]

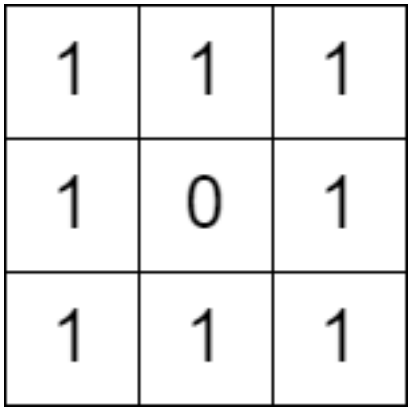
Output:

true

Explanation:

We can change the cell shown in the diagram above. There is no path from (0, 0) to (2, 2) in the resulting grid.

Example 2:



Input:

grid = [[1,1,1],[1,0,1],[1,1,1]]

Output:

false

Explanation:

It is not possible to change at most one cell such that there is not path from (0, 0) to (2, 2).

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 1000

1 <= m * n <= 10

5

grid[i][j]

is either

0

or

1

.

grid[0][0] == grid[m - 1][n - 1] == 1

Code Snippets

C++:

```
class Solution {
public:
bool isPossibleToCutPath(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public boolean isPossibleToCutPath(int[][] grid) {

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function isPossibleToCutPath(grid: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool IsPossibleToCutPath(int[][] grid) {

}
}
```

C:

```
bool isPossibleToCutPath(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func isPossibleToCutPath(grid [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun isPossibleToCutPath(grid: Array<IntArray>): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn is_possible_to_cut_path(grid: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool isPossibleToCutPath(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def isPossibleToCutPath(grid: Array[Array[Int]]): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-possible-to-cut-path grid)
(-> (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Disconnect Path in a Binary Matrix by at Most One Flip
* 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:
bool isPossibleToCutPath(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Disconnect Path in a Binary Matrix by at Most One Flip
* 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 boolean isPossibleToCutPath(int[][] grid) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Disconnect Path in a Binary Matrix by at Most One Flip
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 isPossibleToCutPath(self, grid: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Disconnect Path in a Binary Matrix by at Most One Flip
* 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[][]} grid
* @return {boolean}
*/
var isPossibleToCutPath = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Disconnect Path in a Binary Matrix by at Most One Flip
* 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 isPossibleToCutPath(grid: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Disconnect Path in a Binary Matrix by at Most One Flip
* 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 bool IsPossibleToCutPath(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Disconnect Path in a Binary Matrix by at Most One Flip
* 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
*/

bool isPossibleToCutPath(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Disconnect Path in a Binary Matrix by at Most One Flip
// 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 isPossibleToCutPath(grid [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isPossibleToCutPath(grid: Array<IntArray>): Boolean {

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Disconnect Path in a Binary Matrix by at Most One Flip
// 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 is_possible_to_cut_path(grid: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool isPossibleToCutPath(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def isPossibleToCutPath(grid: Array[Array[Int]]): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-possible-to-cut-path grid)
(-> (listof (listof exact-integer?)) boolean?)
)
```