Problem 1559: Detect Cycles in 2D Grid

Problem Information

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

Problem Description

Given a 2D array of characters

grid

of size

m x n

, you need to find if there exists any cycle consisting of the

same value

in

grid

.

A cycle is a path of

length 4 or more

in the grid that starts and ends at the same cell. From a given cell, you can move to one of the cells adjacent to it - in one of the four directions (up, down, left, or right), if it has the

same value

of the current cell.

Also, you cannot move to the cell that you visited in your last move. For example, the cycle

(1, 1) -> (1, 2) -> (1, 1)

is invalid because from

(1, 2)

we visited

(1, 1)

which was the last visited cell.

Return

true

if any cycle of the same value exists in

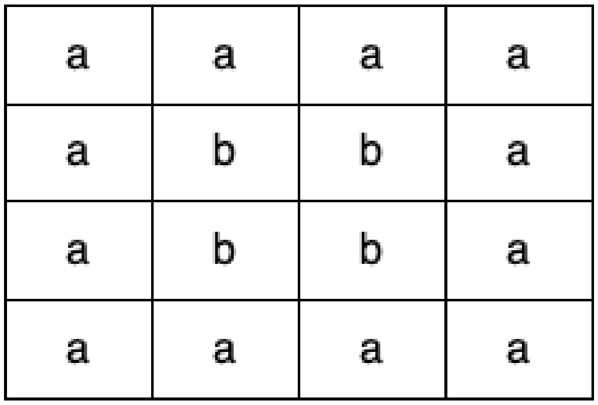
grid

, otherwise, return

false

.

Example 1:



a	a	a	a
a	b	b	a
a	b	b	a
a	a	a	a

Input:

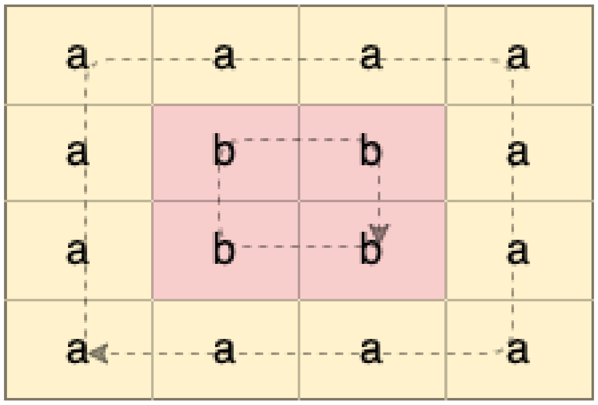
grid = [["a","a","a","a"],["a","b","b","a"],["a","b","b","a"],["a","a","a","a"]]

Output:

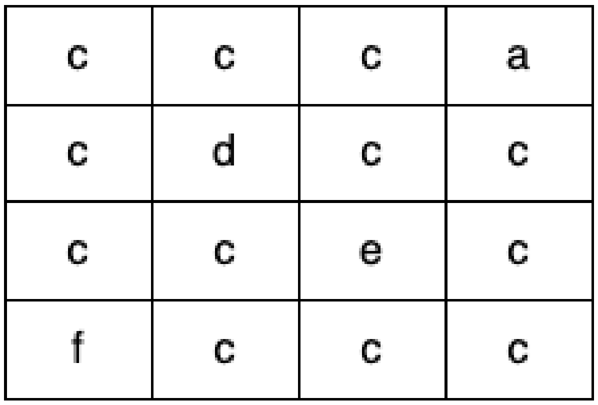
true

Explanation:

There are two valid cycles shown in different colors in the image below:



Example 2:



c	c	c	a
c	d	c	c
c	c	e	c
f	c	c	c

Input:

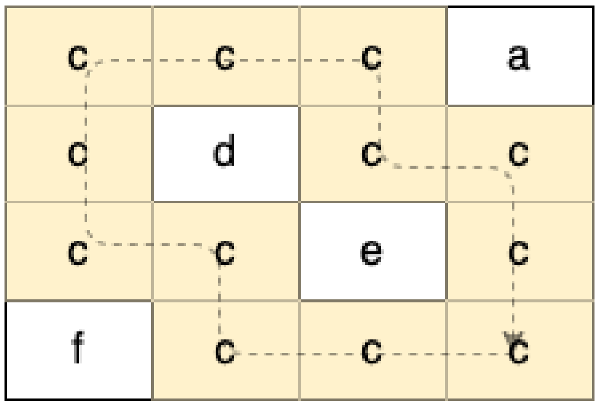
grid = [["c","c","c","a"],["c","d","c","c"],["c","c","e","c"],["f","c","c","c"]]

Output:

true

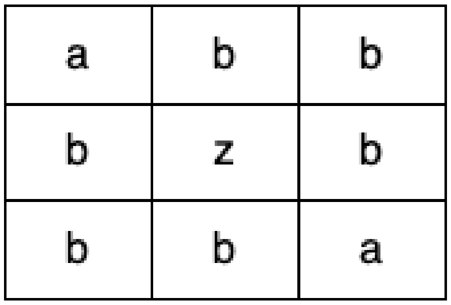
Explanation:

There is only one valid cycle highlighted in the image below:



c	c	c	a
c	d	c	c
c	c	e	c
f	c	c	c

Example 3:



a	b	b
b	z	b
b	b	a

Input:

grid = [["a","b","b"],["b","z","b"],["b","b","a"]]

Output:

false

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 500

grid

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
bool containsCycle(vector<vector<char>>& grid) {

}
};
```

Java:

```
class Solution {
public boolean containsCycle(char[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function containsCycle(grid: string[][]): boolean {

};
```

C#:

```
public class Solution {
public bool ContainsCycle(char[][] grid) {


}
}
```

C:

```
bool containsCycle(char** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func containsCycle(grid [][]byte) bool {

}
```

Kotlin:

```
class Solution {
fun containsCycle(grid: Array<CharArray>): Boolean {

}
}
```

Swift:

```
class Solution {
func containsCycle(_ grid: [[Character]]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn contains_cycle(grid: Vec<Vec<char>>) -> bool {

}
}
```

Ruby:

```
# @param {Character[][]} grid
# @return {Boolean}
def contains_cycle(grid)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $grid
* @return Boolean
*/
function containsCycle($grid) {

}
}
```

Dart:

```
class Solution {
bool containsCycle(List<List<String>> grid) {

}
}
```

Scala:

```
object Solution {
def containsCycle(grid: Array[Array[Char]]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec contains_cycle(grid :: [[char]]) :: boolean
def contains_cycle(grid) do

end
end
```

Erlang:

```
-spec contains_cycle(Grid :: [[char()]]) -> boolean().
contains_cycle(Grid) ->
.
```

Racket:

```
(define/contract (contains-cycle grid)
(-> (listof (listof char?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Detect Cycles in 2D Grid
* Difficulty: Medium
* Tags: array, graph, 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:
bool containsCycle(vector<vector<char>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Detect Cycles in 2D Grid
* Difficulty: Medium
* Tags: array, graph, 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 boolean containsCycle(char[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Detect Cycles in 2D Grid
Difficulty: Medium
Tags: array, graph, 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 containsCycle(self, grid: List[List[str]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Detect Cycles in 2D Grid
* Difficulty: Medium
* Tags: array, graph, 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 {character[][]} grid
* @return {boolean}
*/
var containsCycle = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Detect Cycles in 2D Grid
* Difficulty: Medium
* Tags: array, graph, 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 containsCycle(grid: string[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Detect Cycles in 2D Grid
* Difficulty: Medium
* Tags: array, graph, 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 bool ContainsCycle(char[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Detect Cycles in 2D Grid
* Difficulty: Medium
* Tags: array, graph, 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
*/

bool containsCycle(char** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Detect Cycles in 2D Grid
// Difficulty: Medium
// Tags: array, graph, 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 containsCycle(grid [][]byte) bool {

}
```

Kotlin Solution:

```
class Solution {
fun containsCycle(grid: Array<CharArray>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func containsCycle(_ grid: [[Character]]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Detect Cycles in 2D Grid
// Difficulty: Medium
// Tags: array, graph, 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 contains_cycle(grid: Vec<Vec<char>>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Character[][]} grid
# @return {Boolean}
def contains_cycle(grid)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String[][] $grid
* @return Boolean
*/
function containsCycle($grid) {

}
}
```

Dart Solution:

```
class Solution {
bool containsCycle(List<List<String>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def containsCycle(grid: Array[Array[Char]]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec contains_cycle(grid :: [[char]]) :: boolean
def contains_cycle(grid) do

end
end
```

Erlang Solution:

```
-spec contains_cycle(Grid :: [[char()]]) -> boolean().
contains_cycle(Grid) ->
.
```

Racket Solution:

```
(define/contract (contains-cycle grid)
(-> (listof (listof char?)) boolean?)
)
```