Problem 1728: Cat and Mouse II

Problem Information

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

Problem Description

A game is played by a cat and a mouse named Cat and Mouse.

The environment is represented by a

grid

of size

rows x cols

, where each element is a wall, floor, player (Cat, Mouse), or food.

Players are represented by the characters

'C'

(Cat)

,'M'

(Mouse).

Floors are represented by the character

'.'

and can be walked on.

Walls are represented by the character

'#'

and cannot be walked on.

Food is represented by the character

'F'

and can be walked on.

There is only one of each character

'C'

,

'M'

, and

'F'

in

grid

.

Mouse and Cat play according to the following rules:

Mouse

moves first

, then they take turns to move.

During each turn, Cat and Mouse can jump in one of the four directions (left, right, up, down). They cannot jump over the wall nor outside of the

grid

.

catJump, mouseJump

are the maximum lengths Cat and Mouse can jump at a time, respectively. Cat and Mouse can jump less than the maximum length.

Staying in the same position is allowed.

Mouse can jump over Cat.

The game can end in 4 ways:

If Cat occupies the same position as Mouse, Cat wins.

If Cat reaches the food first, Cat wins.

If Mouse reaches the food first, Mouse wins.

If Mouse cannot get to the food within 1000 turns, Cat wins.

Given a

rows x cols

matrix

grid

and two integers

catJump

and

mouseJump

, return

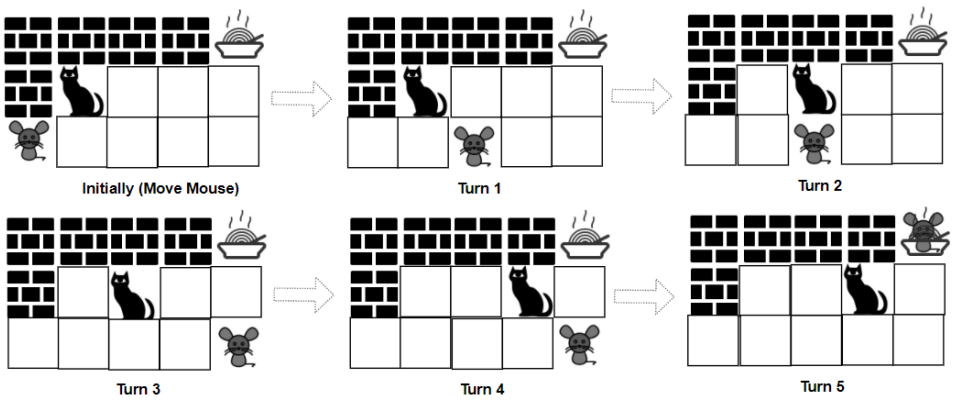
true

if Mouse can win the game if both Cat and Mouse play optimally, otherwise return

false

.

Example 1:



Input:

grid = ["####F","#C...","M...."], catJump = 1, mouseJump = 2

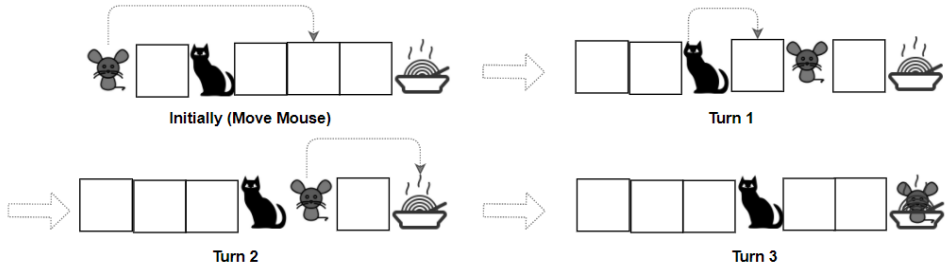
Output:

true

Explanation:

Cat cannot catch Mouse on its turn nor can it get the food before Mouse.

Example 2:



Input:

grid = ["M.C...F"], catJump = 1, mouseJump = 4

Output:

true

Example 3:

Input:

grid = ["M.C...F"], catJump = 1, mouseJump = 3

Output:

false

Constraints:

rows == grid.length

cols = grid[i].length

1 <= rows, cols <= 8

grid[i][j]

consist only of characters

'C'

,

'M'

,

'F'

,

'.'

, and

'#'

.

There is only one of each character

'C'

,

'M'

, and

'F'

in

grid

.

1 <= catJump, mouseJump <= 8

Code Snippets

C++:

```
class Solution {
public:
bool canMouseWin(vector<string>& grid, int catJump, int mouseJump) {

}
};
```

Java:

```
class Solution {
public boolean canMouseWin(String[] grid, int catJump, int mouseJump) {

}
}
```

Python3:

```
class Solution:
def canMouseWin(self, grid: List[str], catJump: int, mouseJump: int) -> bool:
```

Python:

```
class Solution(object):
def canMouseWin(self, grid, catJump, mouseJump):
"""
:type grid: List[str]
:type catJump: int
:type mouseJump: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string[]} grid
* @param {number} catJump
* @param {number} mouseJump
* @return {boolean}
*/
```

```
var canMouseWin = function(grid, catJump, mouseJump) {

};
```

TypeScript:

```
function canMouseWin(grid: string[], catJump: number, mouseJump: number):
boolean {

};
```

C#:

```
public class Solution {
public bool CanMouseWin(string[] grid, int catJump, int mouseJump) {

}
}
```

C:

```
bool canMouseWin(char** grid, int gridSize, int catJump, int mouseJump) {

}
```

Go:

```
func canMouseWin(grid []string, catJump int, mouseJump int) bool {

}
```

Kotlin:

```
class Solution {
fun canMouseWin(grid: Array<String>, catJump: Int, mouseJump: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func canMouseWin(_ grid: [String], _ catJump: Int, _ mouseJump: Int) -> Bool
{

}
}
```

Rust:

```
impl Solution {
pub fn can_mouse_win(grid: Vec<String>, cat_jump: i32, mouse_jump: i32) ->
bool {

}
}
```

Ruby:

```
# @param {String[]} grid
# @param {Integer} cat_jump
# @param {Integer} mouse_jump
# @return {Boolean}
def can_mouse_win(grid, cat_jump, mouse_jump)

end
```

PHP:

```
class Solution {

/**
* @param String[] $grid
* @param Integer $catJump
* @param Integer $mouseJump
* @return Boolean
*/
function canMouseWin($grid, $catJump, $mouseJump) {

}
}
```

Dart:

```
class Solution {
bool canMouseWin(List<String> grid, int catJump, int mouseJump) {

}
}
```

Scala:

```
object Solution {
def canMouseWin(grid: Array[String], catJump: Int, mouseJump: Int): Boolean =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec can_mouse_win(grid :: [String.t], cat_jump :: integer, mouse_jump ::
integer) :: boolean
def can_mouse_win(grid, cat_jump, mouse_jump) do

end
end
```

Erlang:

```
-spec can_mouse_win(Grid :: [unicode:unicode_binary()], CatJump :: integer(),
MouseJump :: integer()) -> boolean().
can_mouse_win(Grid, CatJump, MouseJump) ->
.
```

Racket:

```
(define/contract (can-mouse-win grid catJump mouseJump)
(-> (listof string?) exact-integer? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Cat and Mouse II
* Difficulty: Hard
* Tags: array, graph, dp, math, sort
*
* 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 canMouseWin(vector<string>& grid, int catJump, int mouseJump) {

}
};
```

Java Solution:

```
/**
* Problem: Cat and Mouse II
* Difficulty: Hard
* Tags: array, graph, dp, math, sort
*
* 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 canMouseWin(String[] grid, int catJump, int mouseJump) {

}
}
```

Python3 Solution:

```
"""
Problem: Cat and Mouse II
Difficulty: Hard
Tags: array, graph, dp, math, sort
```

```
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 canMouseWin(self, grid: List[str], catJump: int, mouseJump: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canMouseWin(self, grid, catJump, mouseJump):
"""
:type grid: List[str]
:type catJump: int
:type mouseJump: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Cat and Mouse II
* Difficulty: Hard
* Tags: array, graph, dp, math, sort
*
* 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 {string[]} grid
* @param {number} catJump
* @param {number} mouseJump
* @return {boolean}
*/
var canMouseWin = function(grid, catJump, mouseJump) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Cat and Mouse II
* Difficulty: Hard
* Tags: array, graph, dp, math, sort
*
* 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 canMouseWin(grid: string[], catJump: number, mouseJump: number):
boolean {

};
```

C# Solution:

```
/*
* Problem: Cat and Mouse II
* Difficulty: Hard
* Tags: array, graph, dp, math, sort
*
* 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 CanMouseWin(string[] grid, int catJump, int mouseJump) {

}
}
```

C Solution:

```
/*
* Problem: Cat and Mouse II
```

```
* Difficulty: Hard
* Tags: array, graph, dp, math, sort
*
* 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 canMouseWin(char** grid, int gridSize, int catJump, int mouseJump) {

}
```

Go Solution:

```
// Problem: Cat and Mouse II
// Difficulty: Hard
// Tags: array, graph, dp, math, sort
//
// 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 canMouseWin(grid []string, catJump int, mouseJump int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canMouseWin(grid: Array<String>, catJump: Int, mouseJump: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canMouseWin(_ grid: [String], _ catJump: Int, _ mouseJump: Int) -> Bool
{

}
```

```
}
```

Rust Solution:

```
// Problem: Cat and Mouse II
// Difficulty: Hard
// Tags: array, graph, dp, math, sort
//
// 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 can_mouse_win(grid: Vec<String>, cat_jump: i32, mouse_jump: i32) ->
bool {

}
}
```

Ruby Solution:

```
# @param {String[]} grid
# @param {Integer} cat_jump
# @param {Integer} mouse_jump
# @return {Boolean}
def can_mouse_win(grid, cat_jump, mouse_jump)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $grid
* @param Integer $catJump
* @param Integer $mouseJump
* @return Boolean
*/
function canMouseWin($grid, $catJump, $mouseJump) {
```

```
}
}
```

Dart Solution:

```
class Solution {
bool canMouseWin(List<String> grid, int catJump, int mouseJump) {

}
}
```

Scala Solution:

```
object Solution {
def canMouseWin(grid: Array[String], catJump: Int, mouseJump: Int): Boolean =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_mouse_win(grid :: [String.t], cat_jump :: integer, mouse_jump ::
integer) :: boolean
def can_mouse_win(grid, cat_jump, mouse_jump) do

end
end
```

Erlang Solution:

```
-spec can_mouse_win(Grid :: [unicode:unicode_binary()], CatJump :: integer(),
MouseJump :: integer()) -> boolean().
can_mouse_win(Grid, CatJump, MouseJump) ->
.
```

Racket Solution:

```
(define/contract (can-mouse-win grid catJump mouseJump)
(-> (listof string?) exact-integer? exact-integer? boolean?)
```

```
)
```