Problem 1275: Find Winner on a Tic Tac Toe Game

Problem Information

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

Problem Description

Tic-tac-toe

is played by two players

A

and

B

on a

3 x 3

grid. The rules of Tic-Tac-Toe are:

Players take turns placing characters into empty squares

' '

.

The first player

A

always places

'X'

characters, while the second player

B

always places

'O'

characters.

'X'

and

'O'

characters are always placed into empty squares, never on filled ones.

The game ends when there are

three

of the same (non-empty) character filling any row, column, or diagonal.

The game also ends if all squares are non-empty.

No more moves can be played if the game is over.

Given a 2D integer array

moves

where

moves[i] = [row

i

, col

i

]

indicates that the

i

th

move will be played on

grid[row

i

][col

i

]

. return

the winner of the game if it exists

(

A

or

B

). In case the game ends in a draw return

"Draw"

. If there are still movements to play return

"Pending"

.

You can assume that

moves

is valid (i.e., it follows the rules of

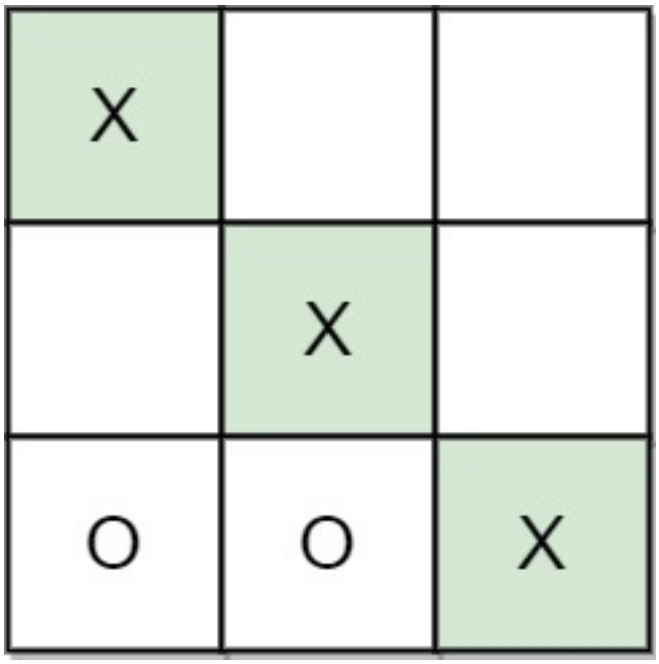
Tic-Tac-Toe

), the grid is initially empty, and

A

will play first.

Example 1:



Input:

moves = [[0,0],[2,0],[1,1],[2,1],[2,2]]

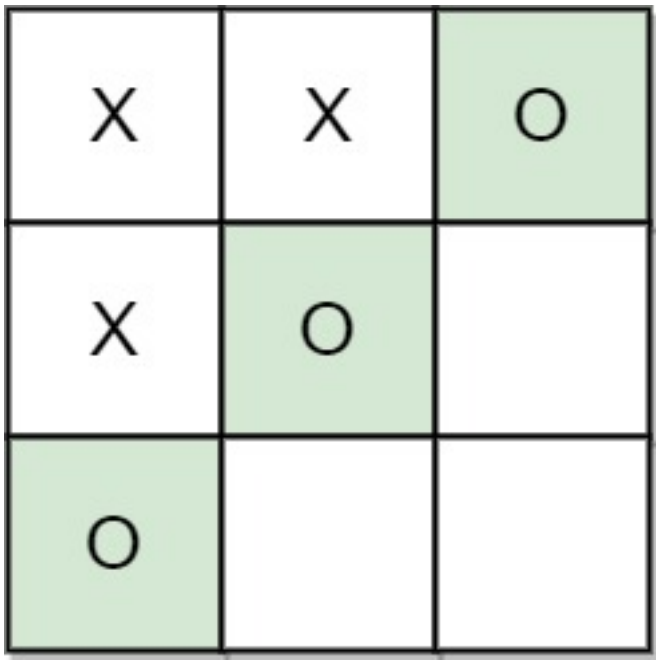
Output:

"A"

Explanation:

A wins, they always play first.

Example 2:



Input:

moves = [[0,0],[1,1],[0,1],[0,2],[1,0],[2,0]]

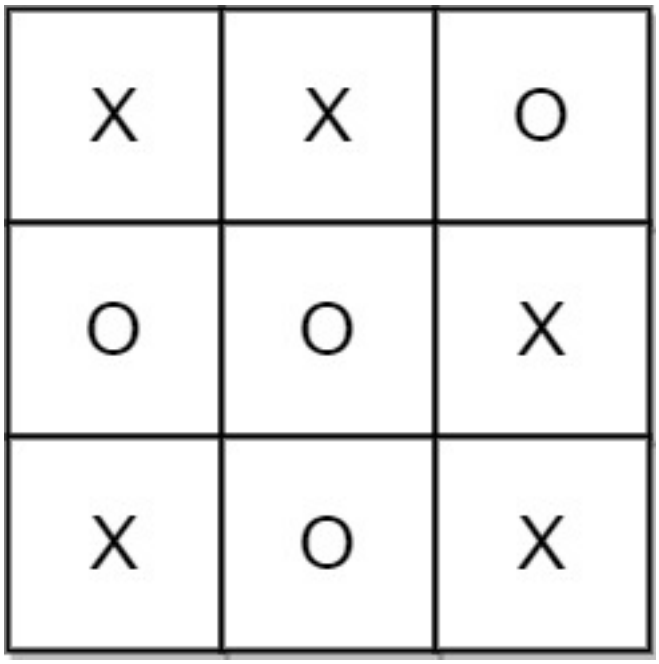
Output:

"B"

Explanation:

B wins.

Example 3:



Input:

moves = [[0,0],[1,1],[2,0],[1,0],[1,2],[2,1],[0,1],[0,2],[2,2]]

Output:

"Draw"

Explanation:

The game ends in a draw since there are no moves to make.

Constraints:

1 <= moves.length <= 9

moves[i].length == 2

0 <= row

i

, col

i

<= 2

There are no repeated elements on

moves

.

moves

follow the rules of tic tac toe.

Code Snippets

C++:

```
class Solution {
public:
string tictactoe(vector<vector<int>>& moves) {

}
};
```

Java:

```
class Solution {
public String tictactoe(int[][] moves) {

}
}
```

Python3:

```
class Solution:
def tictactoe(self, moves: List[List[int]]) -> str:
```

Python:

```
class Solution(object):
def tictactoe(self, moves):
"""
:type moves: List[List[int]]
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number[][]} moves
* @return {string}
*/
var tictactoe = function(moves) {

};
```

TypeScript:

```
function tictactoe(moves: number[][]): string {

};
```

C#:

```
public class Solution {
public string Tictactoe(int[][] moves) {

}
}
```

C:

```
char* tictactoe(int** moves, int movesSize, int* movesColSize) {

}
```

Go:

```
func tictactoe(moves [][]int) string {

}
```

Kotlin:

```
class Solution {
fun tictactoe(moves: Array<IntArray>): String {

}
}
```

Swift:

```
class Solution {
func tictactoe(_ moves: [[Int]]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn tictactoe(moves: Vec<Vec<i32>>) -> String {

}
}
```

Ruby:

```
# @param {Integer[][]} moves
# @return {String}
def tictactoe(moves)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $moves
* @return String
*/
function tictactoe($moves) {

}
```

```
}
```

Dart:

```
class Solution {
String tictactoe(List<List<int>> moves) {

}
}
```

Scala:

```
object Solution {
def tictactoe(moves: Array[Array[Int]]): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec tictactoe(moves :: [[integer]]) :: String.t
def tictactoe(moves) do

end
end
```

Erlang:

```
-spec tictactoe(Moves :: [[integer()]]) -> unicode:unicode_binary().
tictactoe(Moves) ->
.
```

Racket:

```
(define/contract (tictactoe moves)
(-> (listof (listof exact-integer?)) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Winner on a Tic Tac Toe Game
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
string tictactoe(vector<vector<int>>& moves) {

}
};
```

Java Solution:

```
/**
* Problem: Find Winner on a Tic Tac Toe Game
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String tictactoe(int[][] moves) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Winner on a Tic Tac Toe Game
Difficulty: Easy
Tags: array, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

```
class Solution(object):
def tictactoe(self, moves):
"""
:type moves: List[List[int]]
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Find Winner on a Tic Tac Toe Game
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[][]} moves
* @return {string}
*/
var tictactoe = function(moves) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Winner on a Tic Tac Toe Game
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function tictactoe(moves: number[][]): string {

};
```

C# Solution:

```
/*
* Problem: Find Winner on a Tic Tac Toe Game
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string Tictactoe(int[][] moves) {

}
}
```

C Solution:

```
/*
* Problem: Find Winner on a Tic Tac Toe Game
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

char* tictactoe(int** moves, int movesSize, int* movesColSize) {

}
```

Go Solution:

```
// Problem: Find Winner on a Tic Tac Toe Game
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func tictactoe(moves [][]int) string {

}
```

Kotlin Solution:

```
class Solution {
fun tictactoe(moves: Array<IntArray>): String {

}
}
```

Swift Solution:

```
class Solution {
func tictactoe(_ moves: [[Int]]) -> String {

}
}
```

Rust Solution:

```
// Problem: Find Winner on a Tic Tac Toe Game
// Difficulty: Easy
// Tags: array, hash
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn tictactoe(moves: Vec<Vec<i32>>) -> String {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} moves
# @return {String}
def tictactoe(moves)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $moves
* @return String
*/
function tictactoe($moves) {

}
}
```

Dart Solution:

```
class Solution {
String tictactoe(List<List<int>> moves) {

}
}
```

Scala Solution:

```
object Solution {
def tictactoe(moves: Array[Array[Int]]): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec tictactoe(moves :: [[integer]]) :: String.t
def tictactoe(moves) do

end
end
```

Erlang Solution:

```
-spec tictactoe(Moves :: [[integer()]]) -> unicode:unicode_binary().
tictactoe(Moves) ->
.
```

Racket Solution:

```
(define/contract (tictactoe moves)
(-> (listof (listof exact-integer?)) string?)
)
```