Problem 348: Design Tic-Tac-Toe

Problem Information

Difficulty: Medium
Acceptance Rate: 58.70%
Paid Only: Yes
Tags: Array, Hash Table, Design, Matrix, Simulation

Problem Description

Assume the following rules are for the tic-tac-toe game on an `n x n` board between two players:

1. A move is guaranteed to be valid and is placed on an empty block. 2. Once a winning condition is reached, no more moves are allowed. 3. A player who succeeds in placing `n` of their marks in a horizontal, vertical, or diagonal row wins the game.

Implement the `TicTacToe` class:

* `TicTacToe(int n)` Initializes the object the size of the board `n`. * `int move(int row, int col, int player)` Indicates that the player with id `player` plays at the cell `(row, col)` of the board. The move is guaranteed to be a valid move, and the two players alternate in making moves. Return * `0` if there is **no winner** after the move, * `1` if **player 1** is the winner after the move, or * `2` if **player 2** is the winner after the move.

Example 1:

Input ["TicTacToe", "move", "move", "move", "move", "move", "move", "move"] [[3], [0, 0, 1], [0, 2, 2], [2, 2, 1], [1, 1, 2], [2, 0, 1], [1, 0, 2], [2, 1, 1]] **Output** [null, 0, 0, 0, 0, 0, 0, 1] **Explanation** TicTacToe ticTacToe = new TicTacToe(3); Assume that player 1 is "X" and player 2 is "O" in the board. ticTacToe.move(0, 0, 1); // return 0 (no one wins) |X| | | | | | | // Player 1 makes a move at (0, 0). | | | | ticTacToe.move(0, 2, 2); // return 0 (no one wins) |X| |O| | | | | // Player 2 makes a move at (0, 2). | | | | ticTacToe.move(2, 2, 1); // return 0 (no one wins) |X| |O| | | | | // Player 1 makes a move at (2, 2). | | |X| ticTacToe.move(1, 1, 2); // return 0 (no one wins) |X| |O| | |O| | // Player 2 makes a move at (1, 1). | | |X| ticTacToe.move(2, 0, 1); // return 0 (no one wins) |X| |O| | |O| | // Player 1 makes a move at (2, 0). |X| |X| ticTacToe.move(1, 0, 2); // return 0 (no one wins) |X| |O| |O|O| | // Player 2 makes a move at

(1, 0). |X| |X| ticTacToe.move(2, 1, 1); // return 1 (player 1 wins) |X| |O| |O|O| | // Player 1 makes a move at (2, 1). |X|X|X|

Constraints:

* `2 <= n <= 100` * player is `1` or `2`. * `0 <= row, col < n` * `(row, col)` are **unique** for each different call to `move`. * At most `n2` calls will be made to `move`.

Follow-up: Could you do better than `O(n2)` per `move()` operation?

Code Snippets

C++:

```
class TicTacToe {
public:
TicTacToe(int n) {

}

int move(int row, int col, int player) {

}
};

/**
* Your TicTacToe object will be instantiated and called as such:
* TicTacToe* obj = new TicTacToe(n);
* int param_1 = obj->move(row,col,player);
*/
```

Java:

```
class TicTacToe {

public TicTacToe(int n) {

}

public int move(int row, int col, int player) {
}
}

/**
* Your TicTacToe object will be instantiated and called as such:
* TicTacToe obj = new TicTacToe(n);
* int param_1 = obj.move(row,col,player);
*/
```

Python3:

```
class TicTacToe:

def __init__(self, n: int):



def move(self, row: int, col: int, player: int) -> int:




# Your TicTacToe object will be instantiated and called as such:
# obj = TicTacToe(n)
# param_1 = obj.move(row,col,player)
```