Description | Solution | Discuss (541) | Submissions

Java · Autocomplete

348. Design Tic-Tac-Toe

Medium 1422 83 Add to List Share

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.

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 n^2 calls will be made to `move`.

Follow-up: Could you do better than $O(n^2)$ per `move()` operation?

Accepted 156,594 | Submissions 276,026

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months | 6 months ~ 1 year | 1 year ~ 2 years

Amazon 12 | Facebook 10 | Microsoft 9 | Apple 3 | Goldman Sachs 2

Related Topics

Array | Hash Table | Design | Matrix

Similar Questions

Valid Tic-Tac-Toe State — Medium

Hide Hint 1

Could you trade extra space such that move() operation can be done in O(1)?

Hide Hint 2

You need two arrays: int rows[n], int cols[n], plus two variables: diagonal, anti_diagonal.

```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);
 */āpublic class TicTacToe {
    int[] rows;
    int[] cols;
    int diagonal;
    int antiDiagonal;

    public TicTacToe(int n) {
        rows = new int[n];
        cols = new int[n];
    }

    public int move(int row, int col, int player) {
        int currentPlayer = (player == 1) ? 1 : -1;
        // update currentPlayer in rows and cols arrays
        rows[row] += currentPlayer;
        cols[col] += currentPlayer;
        // update diagonal
        if (row == col) {
            diagonal += currentPlayer;
        }
        //update anti diagonal
        if (col == (cols.length - row - 1)) {
            antiDiagonal += currentPlayer;
        }
        int n = rows.length;
        // check if the current player wins
        if (Math.abs(rows[row]) == n ||
                Math.abs(cols[col]) == n ||
                Math.abs(diagonal) == n ||
                Math.abs(antiDiagonal) == n) {
            return player;
        }
        // No one wins
        return 0;
    }
}
```