782. Transform to Chessboard

DescriptionHintsSubmissionsDiscussSolution

An N x N `board` contains only `0`s and `1`s. In each move, you can swap any 2 rows with each other, or any 2 columns with each other.

What is the minimum number of moves to transform the board into a "chessboard" - a board where no `0`s and no `1`s are 4-directionally adjacent? If the task is impossible, return -1.

```
Examples:
Input: board = [[0,1,1,0],[0,1,1,0],[1,0,0,1],[1,0,0,1]]
Output: 2
Explanation:
One potential sequence of moves is shown below, from left to right:

0110     1010     1010
0110 --> 1010 --> 0101
1001     0101     1010
1001     0101     0101

The first move swaps the first and second column.
The second move swaps the second and third row.


Input: board = [[0, 1], [1, 0]]
Output: 0
Explanation:
Also note that the board with 0 in the top left corner,
01
10

is also a valid chessboard.

Input: board = [[1, 0], [1, 0]]
Output: -1
Explanation:
No matter what sequence of moves you make, you cannot end with a valid chessboard.
```

Note:

- `board` will have the same number of rows and columns, a number in the range `[2, 30]`.
- `board[i][j]` will be only `0`s or `1`s.

We see that we can swap column and rows independently. So we can make sure first row and first column are in the correct order and then just verify whether the rest of the board is valid. While doing this we can count minimum possible swaps in rows and cols to achieve this state.
The only thing that can vary is the definition of black which can either be `0` or `1`.

```
void swap_cols(vector<vector<int>> &board, int c1, int c2){
        int n = board.size();
        for(int i=0;i<n;i++){
            swap(board[i][c1], board[i][c2]);
        }
    }

    void swap_rows(vector<vector<int>> &board, int r1, int r2){
        int n = board.size();
        for(int i=0;i<n;i++){
            swap(board[r1][i], board[r2][i]);
        }
    }

    bool verify(vector<vector<int>> &board){
        int n = board.size();
        int b = board[0][0];

        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if((i+j)%2 == 0 && board[i][j] != b) return false;
                if((i+j)%2 != 0 && board[i][j] == b) return false;
            }
        }

        return true;
    }

    int can_cols_swap(vector<vector<int>> board, int black){
        int n = board.size();
        vector<int> blks, whites;
        int moves = 0;
        for(int i=0;i<n;i++){
            if(board[0][i] == black && i%2 != 0) blks.push_back(i);
            if(board[0][i] != black && i%2 == 0) whites.push_back(i);
        }

        if(blks.size() == whites.size()){
            moves += blks.size();

            for(int i=0;i<blks.size();i++){
                swap_cols(board, blks[i], whites[i]);
            }
            if(!verify(board)) moves = INT_MAX;
        }else moves = INT_MAX;

        return moves;
    }

    int can_rows_swap(vector<vector<int>> board, int black){
        int moves = 0, n=board.size();
        vector<int> blks, whites;
        for(int i=0;i<n;i++){
            if(board[i][0]==black && i%2 != 0) blks.push_back(i);
            if(board[i][0] != black && i%2 == 0) whites.push_back(i);
        }

        if(blks.size() == whites.size()){
            moves += blks.size();

            for(int i=0;i<blks.size();i++){
                swap_rows(board, blks[i], whites[i]);
            }

            int col_moves = min(can_cols_swap(board, 0),
can_cols_swap(board, 1));
            if(col_moves == INT_MAX) moves = INT_MAX;
            else moves += col_moves;
        }else moves = INT_MAX;

        return moves;
    }

    int movesToChessboard(vector<vector<int>>& board) {
        int ans = min(can_rows_swap(board, 0), can_rows_swap(board, 
1));
        return ans == INT_MAX ? -1 : ans;
    }
```