Transform to Chessboard

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.

Solution 1

An observation is that, in a valid ChessBoard, *any rectangle* inside the board with corner `NW, NE, SW, SE` (NW here means north-west) must satisfy `(NW xor NE) == (SW xor SE)`, where `xor` is exclusive or. Here we call it **validness property**.

Since the swap process does not break this property, for a given board to be valid, this property must hold. A corollary is that **given a row, any other row must be identical to it or be its inverse**. For example if there is a row `01010011` in the board, any other row must be either `01010011` or `10101100`.

With this observation, we **only need to consider the first column when we're swapping rows** to match the ChessBoard condition. That is, it suffices to find the minimum *row swap* to make the first column be `010101...^T` or `101010...^T`. (here `^T` means transpose.)

Similarly, it suffices to consider the first row when swapping columns.

Now the problem becomes solvable, with the following steps:

1. Check if the given board satisfy the validness property defined above.
2. Find minimum row swap to make the first column valid. If not possible, return -1.
3. Find minimum column swap to make the first row valid. If not possible, return -1.
4. Return the sum of step 2 and 3.

written by hiiwave original link here

Solution 2

Two conditions to help solve this problem:

1. In a valid chess board, there are 2 and only 2 kinds of rows and one is inverse to the other.
   For example if there is a row 01010011 in the board, any other row must be either 01010011 or 10101100.
   The same for columns
   A corollary is that, any rectangle inside the board with corners top left, top right, bottom left, bottom right must be 4 zeros or 2 ones 2 zeros or 4 zeros.

2. Another important property is that every row and column has half ones.
   Assume the board is `N * N`:
   If `N = 2*K`, every row and every column has K ones and K zeros.
   If `N = 2*K + 1`, every row and every column has K ones and K + 1 zeros or K + 1 ones and K zeros.

Since the swap process does not break this property, for a given board to be valid, this property must hold.
These two conditions are necessary and sufficient condition for a valid chessboard.

Once we know it is a valid cheese board, we start to count swaps.
Basic on the property above, when we arange the first row, we are actually moving all columns.

I try to arrange one row into `01010` and `10101` and I count the number of swaps.

1. In case of N even, I take the minimum swaps, because both are possible.
2. In case of N odd, I take the even swaps.
   Because when we make a swap, we move 2 columns or 2 rows at the same time.
   So col swaps and row swaps should be even here.

C++:

```
class Solution {
public:
    int movesToChessboard(vector<vector<int>>& b) {
        int N = b.size(), rowSum = 0, colSum = 0, rowSwap = 0, colSwap = 0;
        for (int i = 0; i < N; ++i) for (int j = 0; j < N; ++j)
                if (b[0][0]^b[i][0]^b[0][j]^b[i][j]) return -1;
        for (int i = 0; i < N; ++i) {
            rowSum += b[0][i];
            colSum += b[i][0];
            rowSwap += b[i][0] == i % 2;
            colSwap += b[0][i] == i % 2;
        }
        if (N / 2 > rowSum || rowSum > N / 2 + N % 2) return -1;
        if (N / 2 > colSum || colSum > N / 2 + N % 2) return -1;
        if (N % 2) {
            if (colSwap % 2) colSwap = N - colSwap;
            if (rowSwap % 2) rowSwap = N - rowSwap;
        }
        else {
            colSwap = min(N - colSwap, colSwap);
            rowSwap = min(N - rowSwap, rowSwap);
        }
        return (colSwap + rowSwap) / 2;
    }
};
```

Java:

```
public int movesToChessboard(int[][] b) {
        int N = b.length,rowSum=0,colSum=0,rowSwap=0, colSwap=0;
        for(int i=0; i<N;++i) for (int j=0; j<N;++j)
                if ((b[0][0]+b[i][0]+b[0][j]+b[i][j])%2 == 1) return -1;
        for(int i=0; i<N;++i) {
            rowSum += b[0][i];
            colSum += b[i][0];
            if (b[i][0] == i % 2) rowSwap ++;
            if (b[0][i] == i % 2) colSwap ++ ;
        }
        if (N/2 > rowSum || rowSum > N/2+N%2) return -1;
        if (N/2 > colSum || colSum > N/2+N%2) return -1;
        if (N % 2 == 1) {
            if (colSwap % 2 == 1) colSwap = N - colSwap;
            if (rowSwap % 2 == 1) rowSwap = N - rowSwap;
        }
        else {
            colSwap = Math.min(N - colSwap, colSwap);
            rowSwap = Math.min(N - rowSwap, rowSwap);
        }
        return (colSwap + rowSwap) / 2;
    }
```

Python:

```python
def movesToChessboard(self, b):
        N = len(b)
        if any(b[0][0]^b[i][0]^b[0][j]^b[i][j] for i in range(N) for j in range(N))
: return -1
        if not N/2 <= sum(b[0]) <= N/2+N%2: return -1
        if not N/2 <= sum(b[i][0] for i in range(N)) <= N/2+N%2: return -1

        col = sum(b[0][i] == i % 2 for i in range(N))
        row = sum(b[i][0] == i % 2 for i in range(N))
        if N % 2:
            if col % 2: col = N - col
            if row % 2: row = N - row
        else:
            col = min(N - col, col)
            row = min(N - row, row)
        return (col + row) / 2
```

written by lee215 original link here

Solution 3

The algorithm is based on counting. A solvable board has each row/column either the same as the first row/column or exactly cell-by-cell reversed color of the first row/column.

In the loop we count for **rs** and **cs**, the number of rows/columns being the same as the first row/column, and **rm** and **cm**, the number of misplaced rows/columns in the view of the first row/column. If any row/column is found to be neither the same nor reversed color then returns -1 immediately.

Then, for even number **n** there are two final forms of the first row/column. We compute the minimum swaps of the two cases. For odd number **n** there is only one final form of the board so we compute the swaps based on the fact that whether the first row/column is in the less or the greater half.

```
int movesToChessboard(vector<vector<int>>& b) {
    int n = b.size();
    int rs = 0, cs = 0, rm = 0, cm = 0;

    for (int i = 0; i < n; i++) {
        bool rf = b[0][0] == b[i][0], cf = b[0][0] == b[0][i];
        rs += rf, cs += cf;
        rm += rf ^ !(i & 1), cm += cf ^ !(i & 1);
        for (int j = 0; j < n; j++)
            if ((b[0][j] == b[i][j]) ^ rf || (b[j][0] == b[j][i]) ^ cf)
                return -1;
    }

    if (n % 2 == 0) {
        if (rs == n / 2 && cs == n / 2)
            return min(rm, n - rm) / 2 + min(cm, n - cm) / 2;
        return -1;
    }

    int res = 0;
    if (rs == n / 2)
        res += (n - rm) / 2;
    else if (rs == n / 2 + 1)
        res += rm / 2;
    else
        return -1;

    if (cs == n / 2)
        res += (n - cm) / 2;
    else if (cs == n / 2 + 1)
        res += cm / 2;
    else
        return -1;

    return res;
}
```

written by mzchen original link here