Challenge May 2023
Grid of bulbs
A solution with python DFS by Julien Pradier pradier.j@fr.ibm.com

It's a simple DFS. The double heuristic to speed up things are:

1. Memorized combination of already switches (just avoid wasting your time)
2. Make sure you switch only lights that are off (well that was the rule) but also that the ones that sit in the intersection of a row and column that haven even number of 1s (because if don't want to introduce more unbalance)

Lastly, using int for matrix manipulation is saving a lot of memory and compute. The 24x24 answer come back in less than 2s on my laptop.

Special thanks to ChatGPT for code generation and rock!! who gave me a lot of ideas: https://leetcode.com/problems/minimum-number-of-flips-to-convert-binary-matrix-to-zero-matrix/solutions/446342/java-python-3-convert-matrix-to-int-bfs-and-dfs-codes-w-explanation-comments-and-analysis/

```
from typing import List
import math

def iterative_dfs(mat: List[List[int]]):
    R, C = len(mat), len(mat[0])       # (R) rows, (C) columns
    N = R*C                            # (N) number of elements in mat
    start = int(''.join(map(str, [elem for row in mat for elem in row])), 2)
    end = int("1" * N, 2)              # End matrix with all 1s

    rows_as_numbers = []
    for r in range(R):
        rows_as_numbers.append(int("1" * C, 2) << R*(R-r-1))

    cols_as_numbers = []
    for c in range(C):
        cols_as_numbers.append(sum(1 << (r*R + (C-c-1)) for r in range(R)))

    visited = set()
    stack = [(start, 0, [(0,0)])]
    while stack:
        node, count, path = stack.pop()
        if node == end:
            return (count, path)

        visited.add(node)

        for r in range(R):
            for c in range(C):
                # test1 - should not try to switch 1s
                test1 = not (node & (1 << ((R-r-1) * R + (C-c-1))))
                # test2 - always switch a light which is on a row and column with even number of 1's
                test2 = (bin(node & rows_as_numbers[r]).count('1') == bin(node & cols_as_numbers[c]).count('1'))
                if test1 and test2:
```

```
                next = node ^ rows_as_numbers[r]
                next ^= cols_as_numbers[c]
                next ^= 1 << ((R-r-1) * R + (C-c-1))
                if next not in visited: # avoid infinite loop
                    stack.append((next, count+1, path + [(r, c)])) # ISSUE r,c have wrong value
    return (-1, [])

def transform_to_coord(r,c,R,C):
    return (c+1, R-r)

def get_matrix_from_binary_string(binary_string):
    R = int(math.sqrt(len(binary_string)))
    matrix = [[0] * R for _ in range(R)]

    for i in range(len(binary_string)):
        row = i // R
        col = i % R
        if binary_string[i] == '1':
            matrix[row][col] = 1

    return matrix


matrix = [[0, 0, 1, 1],
          [1, 1, 0, 1],
          [0, 1, 1, 0],
          [0, 0, 0, 1]]
binary_string =
"0000010000000000011100111101000101101010000100110111011100000011010011100001101110001101011011001011010110100100111010101110001001011101001010001100010111000000000001011000000100011000000000100001100100101101101010010111011010111110111000000110001010101110111111001000111100100001001001110001110100101000100011100110010100010011111101011100010000101110001000000000010110010100010100111100101001001001111011010000011000111111101101011010000000111001001100101010001101110011001000001100000000100111001001011111110110111111000101010100000000011010101110000011000111110000001100111 1"
binary_strings = [
    '1100010000001001011100110010 00',
    '0101000011000111010101111001 10',
    '0001100110100111111000101000 10',
    '1111011101100111011010011000 1',
    '0001100010001000110011011000 10',
    '1011110011101100101111010011 11',
    '0011100001011010011010000011 01',
    '1110011100001010111111111101 00',
    '1100000000001101111110011001 00',
    '1110011101001111100011101110 11',
    '1110101000100101000000011011 00',
    '0101111100110011111101000010 01',
    '0101001110110000011000000110 10',
    '0100010101101111001001110011 01',
    '1111110100010111001011001101 10',
```

```
    '1010001101100101111111011001001',
    '1110110001001011111101001100010',
    '1010011000110101000100000100001',
    '1111111001111111110010111110010',
    '0100000100000110010010100100011',
    '1110101100110111111101100110110',
    '0111001100011010011000000000110',
    '1111101001010100001000110110101',
    '1111000111110000111100010011111',
    '1110001111111010111111011100100',
    '1010110000110011101010110000011',
    '0011010111010000011001011010001',
    '0100101000000110111001010100001',
    '0101111010011101000101100100101',
    '1100000110101111101001100000010'
]
#binary_string = "".join(binary_strings)
#print(len(binary_string))
matrix = get_matrix_from_binary_string(binary_string)
R, C = len(matrix), len(matrix[0])       # (R) rows, (C) columns

result = iterative_dfs(matrix)
count, paths = result
print("Solved with number of steps = ", count)
is_first_iteration = True
coords = []
for step in paths:
    r, c  = step
    if is_first_iteration:
        is_first_iteration = False
    else:
        coord = transform_to_coord(r,c,R,C)
        coords.append(coord)

print("RESULT: ", coords)
```