A SOLUTION TO LIGHTS OUT GAME

Bhupathi Shwejan Raj[1], Bolisetty Sujith[2], Gummadi Pavani[3], Janagama Vamshi Krishna[4], Suravarapu Ankith[5]

[1,2,3,4,5] Department of Computer Science Engineering
Amrita Vishwa Vidyapeetham
amenu4aie20017@am.students.amrita.edu; amenu4aie20018@am.students.amrita.edu ; amenu4aie20033@am.students.amrita.edu;amenu4aie20036@am.students.amrita.edu; amenu4aie20070@am.students.amrita.edu

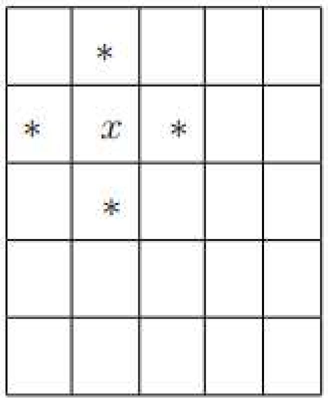
Abstract:

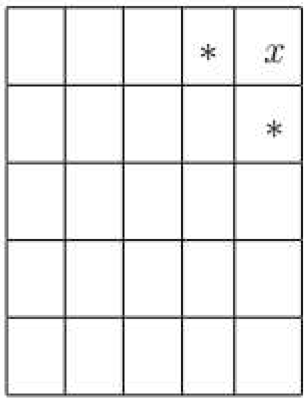
Lights Out is an electronic game delivered by Tiger Electronics in 1995. It contains of n*n blocks. This Lights Out puzzle is a straightforward yet difficult practice in rationale. This is an incredible riddle for building focus and intellectual process. At beginning in n*n blocks the lights will sparkle in an irregular position. The fundamental objective of the riddle is to turn every one of the lights out, ideally in as hardly any button presses as could really be expected. In this paper we have found out a solution to how to solve this game.We have used MATLAB for our implementation.
Keywords: Lights out, blocks, MATLAB

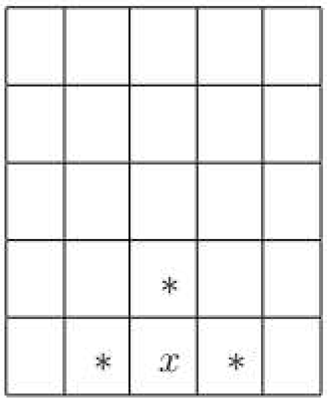
Introduction:

The game of Lights Out is a simple game with straightforward rules that is a stimulating puzzle for anyone up to the challenge of a logic game. The most intriguing aspects of this game are hidden within the mathematical structure describing the game that allows for the use of techniques from various fields of mathematics.The traditional game of Lights Out is played on a grid of n×n lights. The game starts out with an initial condition, where some combination of lights are on and the rest are off. The goal of the player is to press the correct sequence of buttons in order to turn all of the lights out. Each light on the grid is a button, and pressing a light changes the state of the light pressed, as well as the lights that are adjacent in a + shape around it. Regardless of the version of the game you play, there are a few underlying components to the Lights Out game.If a light is on, it must be toggled an odd number of times to be turned off. If a light is off, it must be toggled an even number of times (including none at all) for it to remain off.

If each of the boxes on this grid represent a light, consider what happens when light x is pressed. Under the rules of the classic game of Lights Out, the light and those connected in a + pattern around it will change state







If button x is pressed, then x as well as the lights labeled with $*$'s will change state. Here are a few possibilities

The first important component of the Lights Out Game is the graph it is played on. In the traditional versions of the game, it is played on a square grid of some size, commonly a 3×3 grid for Merlin or 5×5 for Lights Out. Newer versions have extended the playing board to grids of all sizes, including rectangular grids, and three dimensional figures such as a $3 \times 3 \times 3$ cube for the Lights Out Cube and a Dodecahedron for Orbix. The next component is the possible states of each light. Most commonly, lights can be in one of two states, either on or off.

Games such as Lights Out 2000 have colored lights, where each time a button is pressed, the light and its neighbors cycle between red, green or off. The version called Alien Tiles increases the difficulty even more by adding four different colors the lights can be, red, green, blue and yellow.

There are many possibilities for how a button press will affect neighboring lights. In the traditional Lights Out game, pressing a light changes the state of the light pressed, as well as any other lights in a + pattern around it. Some versions include a wrap-around effect, where the grid acts as if it were a torus. Sometimes the neighbors of the light will include all lights in a square pattern around the light pressed, all the lights in a × shape or even all the lights that could be reached by a knight chess piece from the button pressed. Alien Tiles changes all the lights in the same row and column as the button pressed. In any version, a button press could change its own state as well as its neighbors, or only change its neighbors. In increasingly difficult versions, the player is restricted to only pressing certain lights. There are versions where the player can only press lit buttons, or toggle between lit and unlit.

Algorithm design methodology

- An application that solves Lights Out puzzle problems. Lights Out is a 1995 puzzle game in which the player is provided with a rectangular grid of lights, some of which are illuminated. The object of the game is to switch off every light. To do so, the player can toggle the state of each of the (up to four) surrounding lights by pressing the lights (which are actually buttons). Given the grid, for example

```
. . . * .
. . . * *
. . . . .
```

- Pressing the fourth light on the second row would change the state of the grid to be

```
. . . . .
. . * . .
. . . * .
```

- This problem can be well described and solved effectively using a creative mixture of linear algebra and finite fields; this problem has been well-studied in the mathematical community. The most important thing to remember is that the state of the board may be represented as a matrix of 0s and 1s, with each 0 representing "light off" and each 1 representing "light on." The above board, for example, would be modeled as

```
0 0 0 0 0
0 0 1 0 0
0 0 0 1 0
```

- Given this, we want to change a set of lights whenever the player presses a button, replacing 0s with 1s and vice versa. Toggling a light can be thought of as adding 1 to a number and then doing modulo 2 because we wish to map 0 to 1 and 1 to 0. Considering the above grid If we press 1 in the center. When we add one to the central light and the lights around it, we obtain

```
0 0 1 0 0
0 1 2 1 0
0 0 1 1 0
```

- which, modulo 2, looks like

$$\begin{matrix} \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{matrix}$$

- This is the correct configuration of the lights at this point.
- Since pushing each button is equivalent to adding 1 (modulo 2) to the numbers around that button and leaving the rest of the lights unchanged, we can think of each button as being associated with a special matrix indicating what values to add to each of the lights in the grid. For example, the matrix for the center button is

$$\begin{matrix} \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \end{matrix}$$

- Because pressing the middle button toggles the five specified lights (by adding one modulo two) while leaving the others unchanged (by adding zero modulo two). We can then simulate pressing a button by simply adding the button's toggle matrix to the state of the world (modulo two, of course). From previous result configuration, pressing the center button a second time would result in

$$\begin{matrix} \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{matrix} + \begin{matrix} \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \end{matrix} = \begin{matrix} \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \end{matrix} = \begin{matrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{matrix}$$

- We are using a mathematical model for solving the problem. Suppose that our game grid is an m x n matrix whose initial configuration is G. Let the toggle matrix for a button at (i, j) be T(i, j). Given this, we want to find a set of matrices such that

G + T(i1 j1) + T(i2, j2) + ... + T(ik, jk) = 0 (modulo 2)

- That is, we want a set of matrices that, summed together with G (modulo 2),produce the zero matrix, in which all of the lights are turned off.

- A slightly different way of thinking about this is to assign to each toggle matrix T(i, j) a coefficient a(i, j) that is either 0 or 1.0 means that we do not push the button, and 1 means that we do. In that case, we are trying to find a set of coefficients a(i, j) such that

$$\mathbf{G + \sum_{i,j} a(i, j)\, T(i, j) = 0 \ (modulo\ 2)}$$

- Let's attempt to make things a little clearer now. Right now, doing this mathematical model is a little complicated because everything is done modulo 2 and we have additional constraints on the values of the coefficients a(i, j) (namely, that they are either zero or one). All of the values we're working with are always zero or one, and addition and multiplication are defined modulo 2. One way to think about this is to imagine that we're dealing with GF(2) elements rather than integers. The Galois field of order 2 (GF(2)) is a mathematical structure that gives a formal formulation of arithmetic modulo two. In particular, it defines arithmetic over zero and one by defining
- How to add numbers modulo two
- How to multiply numbers modulo two
- How to obtain the arithmetic inverse of a number modulo two
- How to obtain the multiplicative inverse of a nonzero number modulo two

- These definitions are as follows:

$0 + 0 = 0$	$0 * 0 = 0$
$0 + 1 = 1$	$0 * 1 = 0$
$1 + 0 = 1$	$1 * 0 = 0$
$1 + 1 = 0$	$1 * 1 = 1$
$-0 = 0$	$1/1 = 1$
$-1 = 1$	

- The rules for addition and multiplication of numbers modulo two probably make sense, but the rules for inverses are a bit tricky. It should be pretty clear why -0 = 0, but the rule -1 = 1 is not necessarily obvious. The reason that it is defined this way is that we want to have that

$$1 + (-1) = (-1) + 1 = 0$$

- Looking at the above table for arithmetic, we see that this is only possible if -1 = 1.
- Here the truth tables for addition and multiplication are the same as the boolean XOR and AND truth tables. This means we can merely use XOR and AND for arithmetic when working with boolean values modulo 2. Also, because the arithmetic inverse of a number (as well as the multiplicative inverse) is always that number, we can invert numbers by doing nothing to them.

- Another important property of these definitions of arithmetic is that multiplication distributes over addition, that is, a(b + c) = ab + ac. The reason that this is important is that it means that all of the usual properties of arithmetic of real numbers apply to arithmetic modulo two. In particular, if we have a linear equation over those values, we can multiply both sides of the equation by some value, add the same value to both sides,etc.
- Simplifying our original formula

$$\mathbf{G + \sum_{i,j} a(i, j)\, T(i, j) = 0 \ (modulo\ 2)}$$

- If we assume that everything here is drawn from GF(2), then we can drop the "modulo 2" to get

$$\mathbf{G + \sum_{i,j} a(i, j)\, T(i, j) = 0}$$

- And we can now add -G to both sides of the equation to get

$$\mathbf{\sum_{i,j} a(i, j)\, T(i, j) = -G}$$

- But remember that -x = x for all x in GF(2), so this is equivalent to

$$\mathbf{\sum_{i,j} a(i, j)\, T(i, j) = G}$$

- Now we need to find out a(i, j). To do this we need to change our notation.Right now, G and the T(i, j)'s are matrices. We can easily convert these matrices into column vectors by just looking at the values in row-major order.
  For example we write T(1,2) :

0 0 1 0 0
0 1 1 1 0 -> (0 0 1 0 0 0 1 1 1 0 0 0 1 0 0)^T
0 0 1 0 0

$$\mathbf{\sum_{i,j} a(i, j)\, T(i, j) = G}$$

- It is just a linear system of equations over a set of vectors. Let's define the matrix V to be the matrix whose columns are the T(i, j) in some order and the vector a to be the vector whose elements are the a(i, j) in that same order. If we do this, the above formula can be rewritten as

$$\mathbf{V\ a = G}$$

- The goal is simple to solve the system of linear equations, each of whose values are drawn from GF(2).
- Because GF(2) is a field, we can just use standard Gaussian elimination to solve for a value of a.

Algorithm

- The algorithm is simple and straightforward.
- The algorithm contains 5 main steps.
- Take the game configuration as input.
- Calculate the dimensions.
- Convert the game configuration matrix into a column vector(G).
- Construct a matrix whose columns are the vector form of toggle matrices(V).
- Append the column vector(G) to the matrix(V)
- Perform gauss elimination.
- Return the last column of the resultant matrix.
- Convert the column vector into a matrix and print the solution.

Analysis

Time Complexity

- There are 2 main steps which consume very much time i.e., are calculation of the toggle matrices and performing Gauss-elimination.
- Creating a 1 toggle matrix takes O(n^2) time.
- There are n^2 elements for given game configuration so we need to calculate n^2 toggle matrices. These takes O(n^2 * n^2) i.e., O(n^4).
- For calculating Gauss-elimination, at each step, we need to search the matrix for a column containing a non-zero value to use it as pivot.
- This takes O(n^2) per column and there are n^2 columns.
- So it takes O(n^4) time.
- when we find the column we then need to clear each other column.
- This may require looking at all n^4 elements of the matrix once for each of the n^2 columns. so making its runtime O(n^6).
- Therefore the total time complexity is O(n^4)+O(n^6) which is equal to O(n^6).

Space Complexity

- Storing the game configuration in a vector takes O(n^2) space.
- For strong a matrix with toggle matrices as columns takes O(n^2 * n^2) space, making it to O(n^4)
- Therefore the total space that will be used is O(n^2)+O(n^4) which is equal to O(n^4).

Implementation

- The code has been done in MATLAB.
- For testing the algorithm an app has been made in flutter.
- The solution module contains mainly 3 files 'rrefgf', 'createToggle', 'printSolution'.
- rrefgf takes a matrix as input and performs gauss elimination on it and returns the resulting matrix.
- createToggle is a function that takes the size of a matrix as input and returns a matrix whose columns are the vector form of toggle matrices.
-
- when we call printSolution by giving the configuration and size of matrix.
- It uses the services of other mentioned functions, it first calls createToggle function and gets the matrix(V).
- It then appends the configuration vector to matrix (V).
- Then printSolution calls rrefgf function and gives it the resultant 'V' matrix.
- Then the printSolution takes the last column of the returned matrix which contains a series of 0's and 1's which is the solution.
- printSolution then prints the solution in n by n format so that it can be easily used to solve the game.

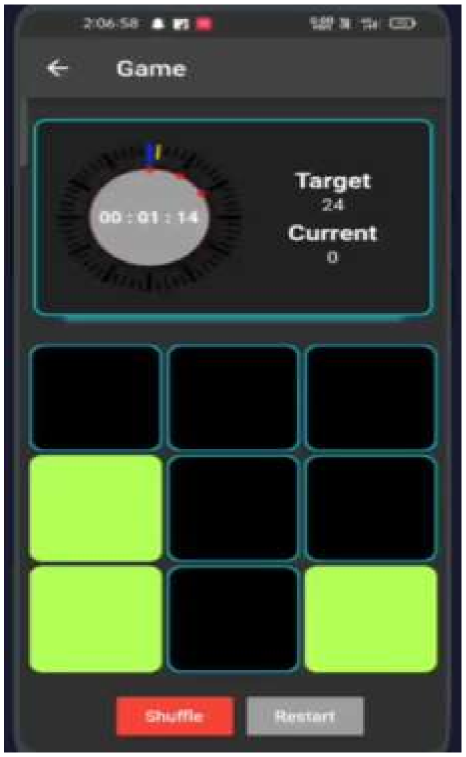
For a given game configuration if a solution exists then the solution produced will take atmost n^2 buttons toggles.

LIMITATIONS

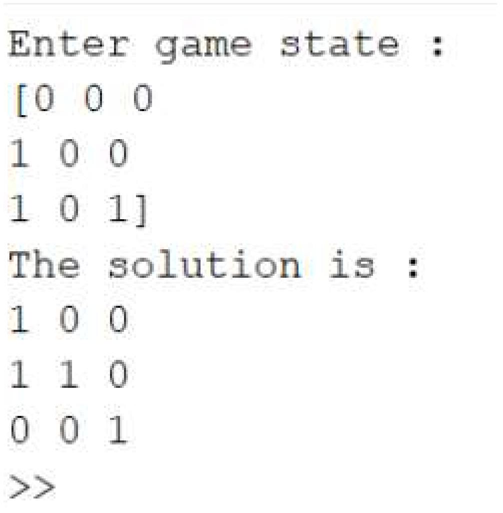
- Like any other game, the lights out game too has some limitations.
- Not all configurations are solvable.
- The matrix 'V' is not of full rank.
- V(3) is of rank 9, so every game with 3 by 3 will have a solution.
- V(5) is of rank 23 instead of 25 so there will be some combinations which can't be solved.

Input and Output

1. **For a 3 by 3 Lights-out game**



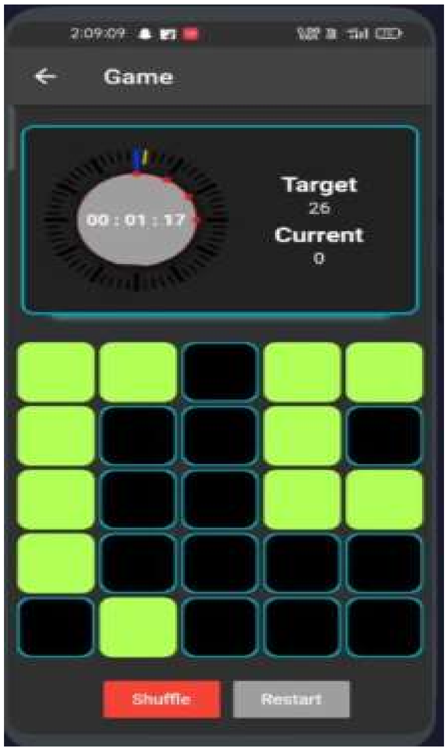
As the image appears we have entered the game state in the form of 1s and 0s as in the first row no light is on so we have taken it as three 0s followed by 1 0 0 as one light is on and then 1 0 1 as 2 lights are on in the last row.



```
Enter game state :
[0 0 0
1 0 0
1 0 1]
The solution is :
1 0 0
1 1 0
0 0 1
>>
```

After entering the game state we get the solution of the game then we need to tap on the box which results in 1 in the solution to solve the game.

2. **For a 5 by 5 Lights-out game**



Here we take the input as 5 rows and 5 columns and we find the solution for the given situation of the game .

```
Enter game state :
[1 1 0 1 1
1 0 0 1 0
1 0 0 1 1
1 0 0 0 0
0 1 0 0 0]
The solution is :
0 0 0 1 0
1 1 1 0 0
1 1 0 1 0
0 1 1 0 0
1 1 0 0 0
```

After entering the game state we get the solution of the game then we need to tap on the box which results in 1 in the solution to solve the game.

Reference:

[1] Dodis, Yevgeniy and Peter Winkler. ”Universal Configurations in Light-Flipping Games.” Society for Industrial and Applied Mathematics (2001):n.pag.Print.

[2] Eriksson, H. ”Note on the Lamp Lighting Problem.” Advances in Applied Mathematics 27.2-3 (2001):357-66. Print.

[3] Joyner, David. Adventures in Group Theory: Rubik’s Cube, Merlin’s Machine, and Other Mathematical Toys. Baltimore: Johns Hopkins UP, 2002. Print.

[4] ”Lights Out (game).” Wikipedia. N.p., 05 Mar.2013. Web. May 2013.

[5] Mulholland, Jamie. ”Lecture 24: Lights Out Puzzle.” SFU Department of Mathematics. N.p., Mar. 2011. Web. May 2013.

[6] Nowakowski, Richard J. More Games of No Chance. Cambridge: Cambridge UP, 2002. Print.

[7] Pelletier, Don. ”Merlin’s Magic Square,” (1987):n. pag. Print.

[8] Madsen, M. A. (2010): Lights Out: Solutions Using Linear Algebra. Summation, 3. 36-40, 2010

[9] Martin-Sanchez, O. and Pareja-Flores, C.: Two Reflected Analyses of Lights Out. Mathematics Magazine,74(4)

[10]Bange, D.W., Barkauskas, A.E., Slater, P.J.: Efficient dominating sets in graphs. In: Ringeisen, R.D., Roberts, F.S. (eds.) Applications of Discrete Mathematics, pp. 189–199. Society of Industrial and Applied Mathematics, Philadelphia (1988)