ICPC: Turn all the lights off

Simon Hanrath
Eberhard Karls University Tübingen
Tübingen, Germany
simon.hanrath@student.uni-tuebingen.de

1 ABSTRACT

The ICPC problem "Turn off all lights" is introduced and a solution is presented. With the help of two observations, the problem can be represented in matrix-vector notation. This allows the complex problem to be reduced to a linear system of equations. For this system of equations, it must be decided whether a solution exists. This is done using Gauss-Jordan elimination in space modulo 2. Finally, the method used is evaluated.

2 INTRODUCTION

The ICPC, formerly called ACM-ICPC (Association for Computing Machinery - International Collegiate Programming), is a competition that can be considered the championship or Olympics for programming competitions around the world. It is the largest programming event in the world. The contestants have to solve several programming tasks in a set amount of time. One of these tasks is called "Turn all the lights off" and in this paper, I will introduce the task and present an approach to solving it. Usually, the programming tasks in the ICPC are introduced by small background stories. For the task "Turn all the lights off" we should imagine that we are working in an office building and since we are the last ones to leave the building, we should turn off all the lights. But before we go and turn off the lights we have to decide if it is even possible to turn off all the lights. This seems trivial at first glance, but it is a relevant question when we consider the wiring of the light switches in this particular office building. Flipping light switches in one room also affects lights in other rooms.

3 PROBLEM STATEMENT

We are given the representation of an office building. Every office contains only square rooms, with exactly one light and an associated switch per room. The problem statement specifies that all office buildings are square and have a maximum of 1600 rooms (i.e. edge length of 40 rooms). The lights are connected in such a way that flipping a light switch in one room also flips the light switches in the four directly adjacent rooms. We can think of the office building as a 2-dimensional grid. Each of the squares represents a room and has four neighbors in the top, bottom, left, and right direction and the flipping of a light switch results in the flipping of the four adjacent light switches. The electrical wiring is such that the sides of the grid wrap around i.e., the leftmost and the rightmost columns are neighbors. Similarly, the top and bottom rows are neighbors. This ensures that each of the squares in the grid has exactly four neighbors. In the following, office buildings are shown as squares consisting of smaller squares colored either black (light off) or yellow (light on) (See figure 1).

We also introduce a coordinate system on the grid that starts with [0,0] in the upper left corner. In this way, [0,1] would refer to the room directly below the room at the very top left and [1,0] to the space directly right to it.

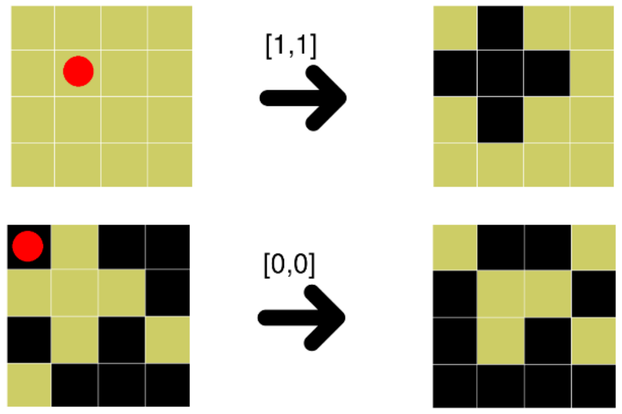

Figure 1: Flipping the switch at the marked position and the effect on the light configuration.

In summary, we get a 2D office building with rooms where the lights are either on or off as input. Our task now is to say whether all lights can be switched off, while taking into consideration that the light switches also affect the directly adjacent rooms.

4 EXAMPLE

An example of a solvable configuration and its solution path can be seen in figure 2. It becomes clear that due to the light switch situation, solutions are not immediately obvious, even for small 3×3 office buildings. In some cases, the number of switched-on lights must be increased before a solution can be found.

Non-solvable configurations are also not easy to find. One might ask whether there are any configurations at all that are non- solvable. This question is not trivial. For all we know the only way to find a non-solvable configuration is to test every possible solution path and those paths may be arbitrary long. Luckily the Task not only

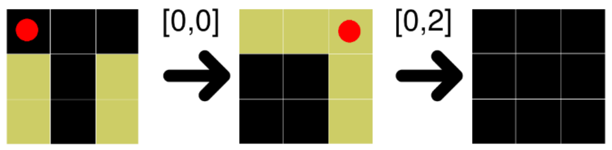

Figure 2: Solution path of a 3×3 office building.

tells us that non-solvable configurations exist, but also provides us with an example of a non-solvable configuration (See Figure 3).

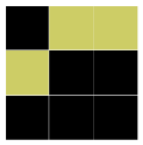

Figure 3: Example of a configuration that can not be solved.

5 SOLUTION

As we have already seen, the solution path may be arbitrarily long, and we have no obvious way of telling which switches need to be flipped. However, if we make two crucial observations, we can not only reduce the space of possible solution paths but also formulate the problem mathematically to determine analytically whether a configuration is solvable.

5.1 WE ARE IN SPACE MODULO 2

The first observation one can make is that any even number of flips of the same switch will restore the previous state. So we do not have to flip a light switch more than once in a row. We either flip the switch or we do not. Flipping a switch two or three times would have the same effect on the current configuration as not flipping it or flipping it once (See figure 4). This observation arises from the binary nature of light. It can be either on or off, and each flip swaps its current state.

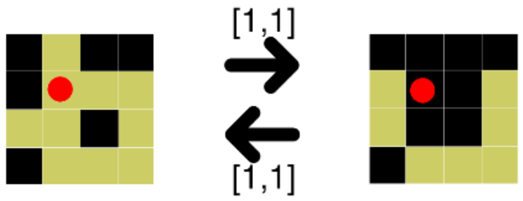

Figure 4: Flipping the same switch more than once is unnecessary.

5.2 ORDER DOES NOT MATTER

The second observation one can make is that every configuration depends only on which switches have been flipped, no matter in what order (See figure 5). The flipping of light switches is therefore commutative. This means that to reach the solution we only have to figure out which switches to flip and not in what order. It also follows that we never need to flip more than n^2 switches for a $n \times n$ office building, since each switch either needs to be flipped or not.

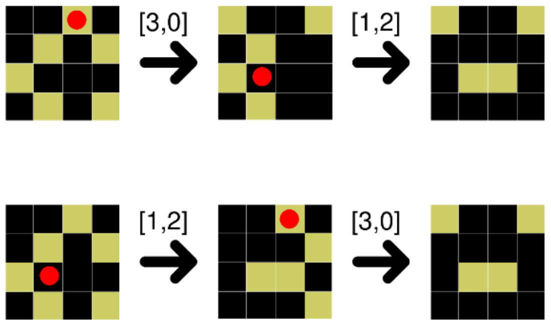

Figure 5: The order in which the switches are flipped is irrelevant for the final configuration.

5.3 SPACE OF POSSIBLE SOLUTION PATHS

The two observations allow us, as already mentioned, to limit the length of a solution path to n^2 steps for a $n \times n$ office building. In each solution path, we have to decide for each light switch whether it should be activated or not. So to test all possible combinations we have to try 2^{n^2} solution paths. For an office building of size $n = 4$, this would be $2^{16} = 65536$ possible paths. However, it can be quickly seen that for larger n, the systematic trying of possible paths becomes unrealistic. For example, for n=40, there would be $2^{1600} = 4.446241648 \cdot 10^{482}$ possible solution paths.

5.4 REPRESENTING THE PROBLEM AS LINEAR SYSTEM

Every configuration (i.e. possible office) can be represented as a matrix $C \in \mathbb{Z}_2^{n\times n}$, with ones representing lit spaces and zeros representing unlit spaces. This follows intuitively from the fact that we are in space modulo 2. Each of the switches can be flipped and each flip forces the 4 adjacent switches to flip as well. Every switch flipping action can also be represented as a $n \times n$ matrix. The matrix $A_{i,j} \in \mathbb{Z}_2^{n\times n}$ is filled with zeros except for the switches that will be flipped, when flipping the switch in the room [i,j], which will be filled with ones.

$$A_{1,1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad A_{2,0} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Now we can mathematically describe the flipping of a switch and its effect on the current configuration as a matrix addition. If we are in the configuration C we can simulate the flipping of the switch at position [i,j] by adding the action matrix $A_{i,j}$ to C and obtain the new configuration C' which represents C after flipping the switch at [i,j]. Our goal is to find a combination of switches, that when flipped lead to a solution, a matrix with only zero entries. Mathematically, a winning combination can then be expressed as

$$C + \sum_{i,j} (b_{i,j} \cdot A_{i,j}) = 0 \tag{1}$$

.
Here 0 denotes the matrix with only zero entries and $b_{i,j}$ a scalar which is either 0 or 1, thereby determining if the switch in room [i,j] should be flipped or not. We can also simplify this equation into

$$\sum_{i,j}(b_{i,j} \cdot A_{i,j}) = C \tag{2}$$

To further simplify this equation we can rewrite the action matrices $A_{i,j}$ as well as the start configuration matrix C as vectors. The order in which we write the values of the matrices into the vectors does not matter as long as it is the same for all transformations. I have chosen to simply write the matrix rows into the vector from top to bottom (see Figure 6).

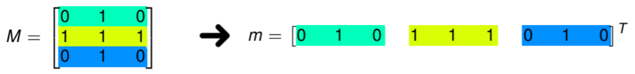

Figure 6: Transformation rule applied to C and all A$_{i,j}$

We still want to solve equation (2), but since all $a_{i,j}$ and c are vectors, we can rewrite (2) by combining the $a_{i,j}$ vectors into a single action matrix A, and similarly combining all $b_{i,j}$ into a single vector b. Thus we obtain the linear system

$$A \cdot b = c \tag{3}$$

with

$$A = \begin{bmatrix} a_{0,0} & | & a_{1,0} & | & \ldots & | & a_{n-1,n-1} \end{bmatrix}$$
$$\text{b} = \begin{bmatrix} b_{0,0}, b_{1,0}, \ldots, b_{n-1,n-1} \end{bmatrix}^T$$

To determine whether a solution exists for a configuration C, we only need to find out whether a solution b exists such that (3) is satisfied. There are several ways to check whether such a solution b exists, but a relatively simple way would be to apply Gauss-Jordan elimination with partial pivoting to A and c.

6 GAUSS-JORDAN ELIMINATION

In mathematics, Gauss-Jordan elimination is an algorithm for solving systems of linear equations. It is a sequence of operations performed on the corresponding matrix of coefficients to transform it into reduced row echelon form. In this reduced row echelon form, we can easily decide whether our system of equations has a solution. Since we are operating in the space modulo 2 we can further simplify the process of Gauss-Jordan elimination. Normally there are three elementary row operations used to achieve reduced row echelon form: Switching two rows, multiplying a row by any non-zero constant, and adding a scalar multiple of one row to any other row. In our case, we only need to switch rows and add rows together. We never need to multiply a row by a non-zero constant, because the only other constant would be one.

6.1 AUGMENTED MATRIX

In order to solve our linear system (3) with Gauss-Jordan elimination, we have to convert it into an alternative representation, which we call an augmented matrix. The augmented matrix is created by joining the matrix A and the vector c and dividing them by a vertical bar as shown below.

$$\left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{array}\right]$$

6.2 REDUCTION

We start from the top left of the augmented matrix of A and c. If the entry at this position is zero, we look at the rows below and if one has a one at in the same column we swap the two rows. If we have a one in the position either after the swap or from the beginning, we add the row to all rows in which there is also a one in the same column. After this, there is only a one at the main diagonal entry of this column. This is then repeated for all diagonal entries and leads to the reduced row echelon form. For our augmented matrix example, the following steps would be required.

$$\left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{array}\right]$$

Add the second to the third row:

$$\left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{array}\right]$$

Swap third and fourth row:

$$\left[\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Add the third to the first row:

$$\left[\begin{array}{cccc|c} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Since the last diagonal entry of the matrix is zero and there are no more ones in a row below it, we stop.

6.3 EVALUATING THE REDUCED ROW ECHELON FORM

After reducing the augmented matrix of A and c we can separate it again. The left part will be called A' and the right part c'. We can distinguish between three cases.

Case one:
All $A'_{k,k} = 1 \implies$ exactly one solution exists.

Case two:
$\exists k(A'_{k,k} = 0$ and $c'_k = 1) \implies$ no solution exists.

Case three:
$\neg\exists k(A'_{k,k} = 0$ and $c'_k = 1) \implies$ multiple solutions exist.

Since we only want to know whether a solution exists, we only need to check whether the second condition is true.

6.4 RUNTIME ANALYSES

The number of operations required for a $n \times n$ matrix is of the order n^3. The creation of the action matrix A and the vector c is negligible for the computational complexity of larger matrices. The reduction as part of the Gauss-Jordan Elimination however is not. Potentially we have to check all n diagonal entries of a given matrix and add all following entries in the row of the diagonal entry to all other rows of the matrix. Thus leading to cubic computational complexity. This is further supported by the empirical run time analyses of a Python implementation (see Figure 7)(see appendix A). Interestingly, even though the curve appears to rise globally in a cubic fashion, it does not rise monotonically. For example, an input of size 31 takes more time than inputs of size 32. Since the action matrix for a given input size is always the same, it is probably the case that the action matrix for input matrices of size 32 is easier to reduce than that for size 31. Despite their difference in size.
While the solution may be found even faster by fine tuning the program or even switching to a faster language, this implementation still solves even the largest expected inputs in less than nine seconds.

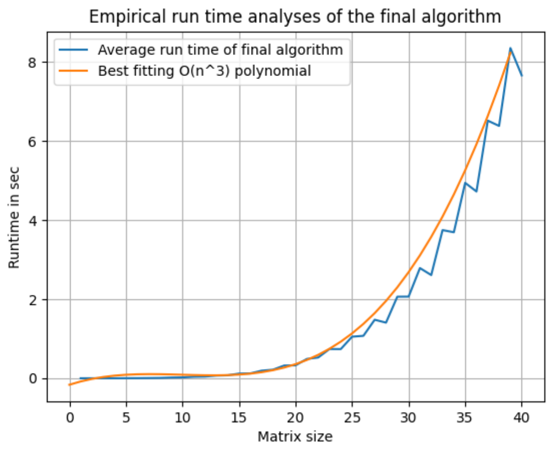

Figure 7: Average run time of the final algorithm in a python implementation. For each input size the run time was averaged over 100 samples.

7 CONCLUSION

The presented solution procedure, can handle all possible inputs of the problem in acceptable time. The solution is based on two important observations. Firstly, we move through the binary nature of the lights in space modulo 2 and secondly, the order of the actions (i.e. flipping the light switches) does not matter for the final result. These assumptions allow us to define the solution space as a linear system of equations and check for a solution with Gauss-Jordan elimination in the space modulo 2. I am sure that more effective ways of solving the system of linear equations exist. For example, one could imagine a method that exploits the sparseness of the action matrix, such as FOM or GMRES [1]. However, this would go beyond the scope of this document.
Although the example of turning off lights seems quite contrived, the solution can be generalised to many other problems. Whenever we want to decide whether a set of discrete actions can lead to a target state and the order of the actions does not matter, the described method (or a version of it) is applicable.

A PYTHON IMPLEMENTATION

An implementation of the final algorithm and a manually usable version of the office building can be found at:
https://github.com/SimonHanrath/ICPC_turn-all-the-Lights-off

REFERENCES

[1] Yousef Saad. 2003. *Iterative Methods for Sparse Linear Systems* (2nd. ed.). Society for Industrial and Applied Mathematics.