# Proyecto ADA Entrega 1

Josue Peña Atencio - 8935601
Octubre 11, 2019

## Especificación del problema

**Entrada**: Dos enteros $K$ y $M$ ( $2 \leq K \leq 10^6$ , $0 \leq M \leq 10^4$ ) representando la cantidad de luces $L[1], L[2], \ldots, L[K]$, y la cantidad de segundos a considerar, respectivamente.
El estado inicial de las $K$ luces es dado como un número en hexadecimal, describiendo el estado de encendido o apagado de las luces si el número se escribe en representación binaria. Luego, por cada segundo $i \in 1..M$ a considerar, se tienen dos números $a_i$ y $b_i$ representando el rango $L[a_i..b_i]$ de luces que deben ser tratadas en el problema (con sus respectivas condiciones).

**Salida**: Configuración final de las $K$ luces en el segundo $M$ (es decir, después de realizar todas las modificaciones) dada como un número en hexadecimal.

## Algoritmos y estructuras de datos

Como estructura de datos se hará uso del estado inicial de las luces dado en hexadecimal como una máscara de bits, para poder realizar *switches* en los rangos de luces de forma constante. La búsqueda del primer bit encendido a la izquierda ( $l_i$ ) o derecha ( $r_i$ ) de los rangos $L[a_i..b_i]$ cuando el bit $a_i$ o $b_i$ está apagado (respectivamente) se hará de forma lineal recorriendo la máscara de bits usando corrimiento binario y el operador AND.

## Estrategia de solución

Como la configuración inicial de luces está dado en representación hexadecimal y en una cadena, podemos hacer uso de la función $int()$ en Python con el parámetro '16' para obtener la representación en número entero del estado inicial. Note que no es necesario 'llenar' con ceros a la izquierda si el número necesita menos de $K$ bits para ser representado, ya que las operaciones de $AND$ y $XOR$ en Python funcionan aunque los operandos no tengan la misma cantidad de bits (es decir, que los ceros a la izquierda ya son proporcionados de forma implícita por el lenguaje)

La estrategia consta a los siguientes pasos:

1. Verificar si $a_i$ o $b_i$ están apagados. Si lo están, realizar la búsqueda del primer bit encendido a la izquierda o derecha recorriendo la máscara de bits posición a posición con corrimiento binario ( $1 << j$ ) a partir de $a_i$ y/o $b_i$ , verificando si se ha encontrado el bit encendido con $mask\ AND\ (1 << j) \geq 0$ . De encontrar tal primer bit encendido en la máscara, se guarda el valor de $j$ en $l_i$ y/o $r_i$ .

2. Una vez que se tengan todos los rangos que deben ser modificados ( $[l_i..a_i - 1]$ , $[a_i..b_i]$ , $[b_i + 1..r_i]$ ) (si estos existen), se procede a hacer el *switch*.
3. Se calcula la longitud total del rango de luces para hacer el *switch* ( $L[l_i..r_i]$ , o $L[a_i..r_i]$ , o $L[a_i..b_i]$ ) restando ambos límites del intervalo respectivo y sumando 1.
4. Suponiendo que la longitud del rango en 3) es $s$ , se calcula una nueva máscara de bits $p$ que tenga $s$ cantidad de 1's. Esto se calcula con $p = (1 << s) - 1$
5. Luego, se procede a alinear la máscara $p$ de modo que $mask\ XOR\ p$ dé como resultado la misma máscara bits con los bits invertidos en el rango calculado en el paso 2). Para tal razón, se hace corrimiento binario de la máscara de 1's auxiliar $p << (K - r)$ . $r$ representa la posición final del intervalo calculado en 2).
6. Finalmente realizamos el *switch* de los bits con $mask = mask\ XOR\ p$
7. Se toman los siguientes par de números $a_i$ y $b_i$ y se realiza el mismo proceso desde 1) con la nueva máscara generada en el paso 6).

## Análisis de complejidad temporal y espacial

Como se mencionó anteriormente, el *switch* de los bits toma tiempo constante, porque solo se necesitan hacer operaciones aritméticas y de corrimiento binario que toman tiempo constante.
Sin embargo, la búsqueda del primer bit a la izquierda o derecha es $O(n)$ con respecto a la cantidad $K$ de luces. Eso significa que si se tienen $M$ segundos y $K$ luces, la complejidad temporal total sería $O(MK)$ . En términos de complejidad espacial, solo se hace uso de una cantidad fija de variables que almacenan números enteros en cualquier momento. Luego la complejidad espacial total es $O(1)$ .

## Bibliografía


- Bitwise operation (https://en.wikipedia.org/wiki/Bitwise_operation)
- Algorithm to generate bit mask (https://stackoverflow.com/a/1392065)
- Convert hex string to int in Python (https://stackoverflow.com/a/209550)
- XOR gate (https://en.wikipedia.org/wiki/XOR_gate)