ÁLGEBRA APLICADA

Prof: Maglis Mujica

Autores: Juan Nocetti - 5.390.966-5
Renzo Giacometti - 5.590.459-8
Ionas Josponis - 5.242.903-0

10 de Noviembre de 2023

Índice

Introducción

Este proyecto se centra en el juego "Lights Out" (traducido como "Luces Apagadas"), un desafío que implica la modificación de una matriz (cuyo tamaño puede ser definido por el usuario) con el objetivo de transformarla en una matriz de ceros. El desafío radica en que al cambiar el estado de una celda, las celdas adyacentes también se ven afectadas. En este contexto, hemos desarrollado un programa en Python que genera una matriz con valores 0 y 1, representando las luces apagadas y encendidas, respectivamente. Además, hemos diseñado un método que proporciona un vector de 0s y 1s como solución para resolver el juego utilizando técnicas de álgebra lineal, en particular, la eliminación gaussiana reducida.

Objetivos

1- Desarrollar un programa en Python que permita resolver el juego "Lights Out".
2- Generar una matriz de luces con dimensiones definidas por el usuario, donde los valores 0 y 1 representan las luces apagadas y prendidas, respectivamente.
3- Implementar un algoritmo que pueda cambiar el estado de las luces y sus adyacentes de acuerdo con las reglas del juego.
4- Desarrollar un método que genere un sistema de ecuaciones lineales a partir de la matriz de luces.

5- Crear una función para aplicar la eliminación gaussiana reducida al sistema de ecuaciones y obtener la solución en forma de vector binario (0s y 1s).
6- Verificar que el programa asegure una solución adecuada y correcta.
7- Representar el juego con una interfaz gráfica, quizás no es necesario pero pensamos que es un buen aspecto para considerar.

Marco Teórico

Sistemas de ecuaciones:

Los sistemas de ecuaciones lineales representan un conjunto de ecuaciones algebraicas interrelacionadas que comparten variables comunes. La resolución de estos sistemas implica encontrar los valores específicos de las variables que cumplen simultáneamente todas las ecuaciones. Estos tienen muchas formas de ser resueltos, uno de ellos es el método de Gauss.

Método de Gauss:

Se trata de emplear operaciones básicas en las ecuaciones para convertir un sistema de ecuaciones lineales n × n en una matriz triangular reducida. Este sistema triangular conserva la equivalencia con el original, lo que significa que comparten las mismas soluciones.

Por ejemplo, si nuestro sistema tiene tres ecuaciones y tres incógnitas, su forma triangular será:

$$\begin{aligned} ax + by + cz &= A \\ b'y + c'z &= B \\ c''z &= C \end{aligned}$$

Expresado como matriz quedaría:

$$\begin{vmatrix} a & b & c & A \\ 0 & b' & c' & B \\ 0 & 0 & c'' & C \end{vmatrix}$$

Para resolver estos sistemas lo único que tendríamos que hacer sería realizar una sustitución hacia adelante, en la cual sustituimos el valor final que nos da c (tomado de la ultima educación) en las otras dos ecuaciones, y luego resolvemos la segunda, tomado el valor de b y sustituyendo en la primera, para por fin resolver la primer ecuación.

Matriz triangular superior: Una matriz cuadrada es triangular superior si y sólo si todos los elementos por debajo de la diagonal principal son cero.

$$\begin{bmatrix} 1 & 4 & 9 \\ 0 & 3 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Matriz aumentada: La llamamos matriz aumentada a una matriz que se obtuvo al concatenar dos matrices distintas.

Eliminación hacia delante o eliminación gaussiana: Consiste en simplificar la matriz a una matriz triangular superior, con ceros debajo de la diagonal, usando operaciones entre filas.

Operación XOR: Es una puerta lógica digital que implementa el O exclusivo; es decir, una salida verdadera resulta si una, y solo una de las entradas a la puerta es verdadera. Si ambas entradas son falsas o ambas son verdaderas, resulta en una salida falsa.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Herramientas

- IDE: Es un entorno de desarrollo para poder programar, en nuestra aplicación usamos PyCharm de JetBrains y Visual Studio Code.

- **Python:** Lenguaje de desarrollo.

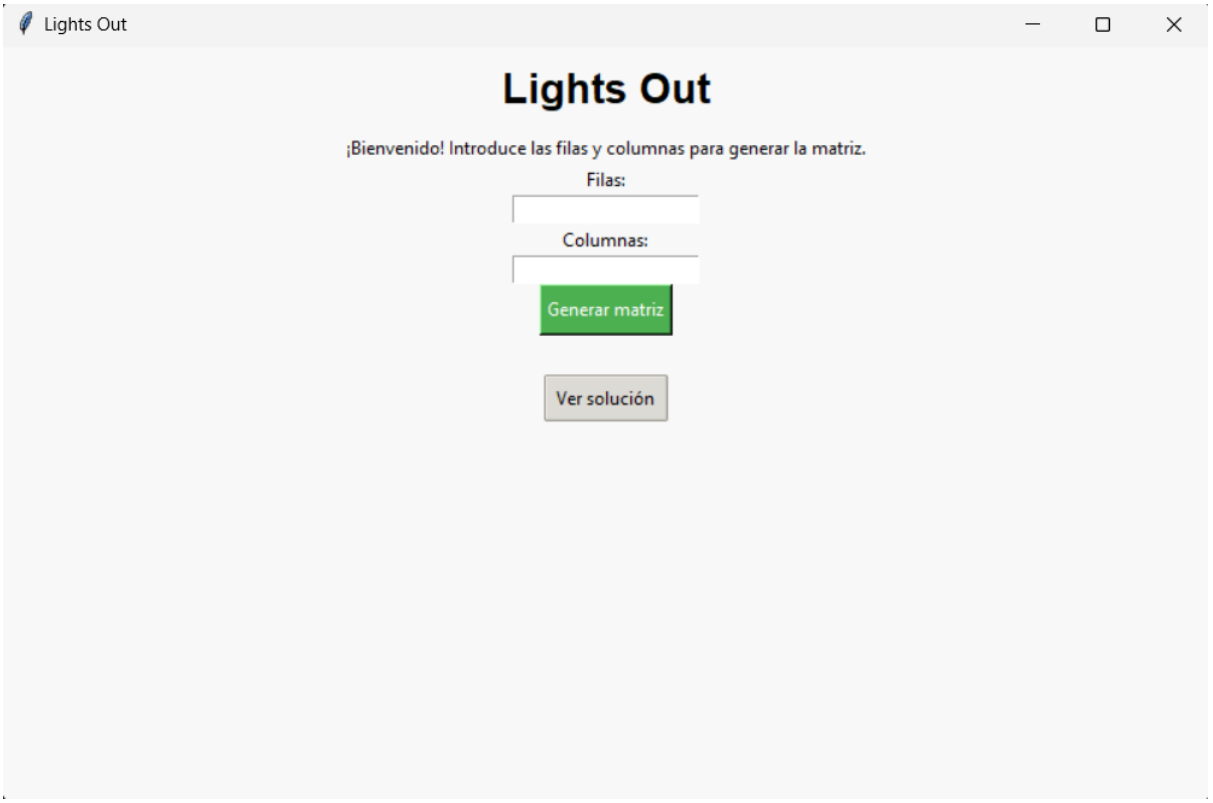
-GitHub: Es una herramienta para desarrollar software colectivamente.

-**Numpy:** Librería de Python que nos facilita algunas operaciones entre vectores y matrices.

Desarrollo

Para desarrollar este programa creamos dos clases principales, la clase utils y main. Dentro de la clase utils se encuentra toda la lógica para poder realizar las diferentes partes del proyecto. Mientras tanto, la clase main conecta la interfaz gráfica con la lógica del programa.

Este programa cuenta con una interfaz gráfica que permite ingresar la cantidad de columnas y filas que se quieren y jugar el juego dentro del mismo, así como también generar una vector de soluciones, en la cual un 1 representa una celda que se debe clickear para ganar, y un 0 una celda en la cual no se debe hacer click.



A continuación se muestran las funciones utilizadas en Python en la clase utils (responsables de la lógica detrás del juego y su solución):

crearMatriz(filas, columnas): Crea la matriz inicial del juego.

```
def crearMatriz(filas, columnas):
    return np.random.randint(2, size=(filas, columnas))
```

cambiarLuces(ai, aj, matriz): Esta función recibe como parámetros la matriz de estado actual del juego, así como las coordenadas i, j de la posición que se acaba de clickear. Retorna la matriz con el valor de la coordenada cambiado, así como también los de las celdas adyacentes.

```
def cambiarLuces(ai, aj, matriz):
    if (not existePosicion(ai, aj, matriz)):
        print("La posición no existe")
        return matriz
    else:
        if matriz[ai][aj] == 0:
            matriz[ai][aj] = 1

        else:
            matriz[ai][aj] = 0

    # Cambiar luz de arriba del seleccionado(ai, aj)
    if (existePosicion(ai, aj - 1, matriz)):
        if matriz[ai][aj - 1] == 0:
            matriz[ai][aj - 1] = 1
        else:
            matriz[ai][aj - 1] = 0
    # Cambiar luz de abajo del seleccionado(ai, aj)
    if (existePosicion(ai, aj + 1, matriz)):
        if matriz[ai][aj + 1] == 0:
            matriz[ai][aj + 1] = 1
        else:
            matriz[ai][aj + 1] = 0

    # Cambiar luz de la izquierda del seleccionado(ai, aj)
    if (existePosicion(ai - 1, aj, matriz)):
        if matriz[ai - 1][aj] == 0:
            matriz[ai - 1][aj] = 1
        else:
            matriz[ai - 1][aj] = 0

    # Cambiar luz de la derecha del seleccionado(ai, aj)
    if (existePosicion(ai + 1 , aj, matriz)):
        if matriz[ai + 1][aj] == 0:
            matriz[ai + 1][aj] = 1
        else:
            matriz[ai + 1][aj] = 0
    return matriz
```

obtenerCambios(ai, aj, matriz): Esta función devuelve qué posiciones de la matriz se modifican en caso de que se haga clic en una posición ai, aj de la matriz que ingresamos como parámetro. Retorna solamente los posibles cambios, no cambia los valores.

```
def obtenerCambios(ai, aj, matriz):
    num_a_cambiar = []
    num_a_cambiar.append([ai, aj])
    # Cambiar luz de arriba del seleccionado(ai, aj)
    if (existePosicion(ai, aj - 1, matriz)):
        num_a_cambiar.append([ai, aj - 1])

    # Cambiar luz de abajo del seleccionado(ai, aj)
    if (existePosicion(ai, aj + 1, matriz)):
        num_a_cambiar.append([ai, aj + 1])

    # Cambiar luz de la izquierda del seleccionado(ai, aj)
    if (existePosicion(ai - 1, aj, matriz)):
        num_a_cambiar.append([ai - 1, aj])

    # Cambiar luz de la derecha del seleccionado(ai, aj)
    if (existePosicion(ai + 1 , aj, matriz)):
        num_a_cambiar.append([ai + 1, aj])
    return num_a_cambiar
```

existePosicion(ai, aj, matriz): Devuelve verdadero en caso de la posición ingresada exista en la matriz, falso en caso de que no exista.

```
def existePosicion(ai, aj, matriz):
    if (ai < matriz.shape[0] and ai >= 0 and aj < matriz.shape[1] and aj >= 0):
        return True
    else:
        return False
```

convertirAX(matriz): Dada una matriz, este método devuelve una nueva matriz con el mismo tamaño pero cambiando los valores (ceros y unos) por x1, x2,.. xn. Por ejemplo:
Dada la siguiente matriz:

⌈0 0 ⌉
⌊0 1⌋

Este método retorna la siguiente matriz:

⌈x1 x2 ⌉
⌊x3 x4⌋

```
def convertirAX(matriz):
    matrizAX = np.full((matriz.shape[0], matriz.shape[1]), '    ')
    i = 0
    j = 0
    cont = 1
    for i in range(matriz.shape[0]):
        for j in range(matriz.shape[1]):
            agregar = str('x' + str(cont))
            matrizAX[i][j] = agregar
            cont = cont + 1
    return matrizAX
```

devolverPosX(i, j, matrizAX): Este método devuelve la posición x en la matrizAX a partir de una fila y columna dada.

```
def devolver_posicion_i_j(x, matriz):
    vector = []
    cont = 1
    for elementos in range(len(matriz[0]) * len(matriz[0])):
        vector.append('x' + str(cont))
        cont = cont + 1

    for i in range(len(vector)):
      if vector[i] == x:
        return i
```

crearEcuaciones(matriz): Esta función devuelve un sistema de ecuaciones a partir de la matriz original en una lista. Sirve para obtener las ecuaciones para usar posteriormente en el método de Gauss.

```
def crearEcuaciones(matriz):

    matrizEcuaciones = []
    i = 0
    j = 0

    matrizAX = convertirAX(matriz)

    for i in range(matriz.shape[0]):
        for j in range(matriz.shape[1]):
            ecuacion = []
            strEcuacion = ''
            cambios = obtenerCambios(i, j,matriz)
            cantidad_cambios = len(cambios)
            for k in range(cantidad_cambios):
                c = cambios[k]
                pos1 = c[0]  # Guardar el primer valor en pos1
                pos2 = c[1]  # Guardar el segundo valor en pos2
                if(k + 1 < cantidad_cambios):
                    strEcuacion += devolverPosX(pos1, pos2, matrizAX) + ' + '
                else:
                    strEcuacion += devolverPosX(pos1, pos2, matrizAX)
            strEcuacion += ' = ' + str(matriz[i][j])
            ecuacion = strEcuacion
            matrizEcuaciones.append(ecuacion)
    # for num in range(len(matrizEcuaciones)):
    #     print(matrizEcuaciones[num])
    return matrizEcuaciones
```

devolver_posicion_i_j(x): Este método dado un x devuelve la posición en el vector posicion.

```
def devolver_posicion_i_j(x):
    vector = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9']
    for i in range(len(vector)):
      if vector[i] == x:
        return i
```

crearA_B(matriz, sistema_ecuaciones): Este método crea las matriz A y el vector B a partir del sistemas ecuaciones para poder luego usarla en la solución del juego.

```
def crearA_B(matriz, sistema_ecuaciones):
    A = np.zeros((len(sistema_ecuaciones), len(sistema_ecuaciones)))
    B = np.zeros((len(sistema_ecuaciones) , 1))



    for n_ecuacion in range(len(sistema_ecuaciones)):

        ecuacion = sistema_ecuaciones[n_ecuacion]
        #ecuacion = ecuacion.replace(" ", "")
        ecuacion =ecuacion.replace("+", "")
        parteIzquierda = ecuacion.split('=')
        cantidad_variables = len(parteIzquierda[0])
        cant_splits = cantidad_variables / 2
        cant_splits = int(cant_splits)
        partes_x = parteIzquierda[0].split(' ')
        #for parte in partes_x:
            #if parte:
        partes_x = [parte for parte in parteIzquierda[0].split(' ') if parte]

        for i in range(len(partes_x)):
            x = partes_x[i]
            pos_i = n_ecuacion
            pos_j = devolver_posicion_i_j(x)
            A[pos_i][pos_j] = 1

        parteIzq = parteIzquierda[1].replace(" ", "")
        B[n_ecuacion] = int(parteIzq)

    return A, B
```

resolverJuego(matriz, sistema_ecuaciones): Esta función genera una matriz aumentada AB para luego obtener el vector solución tras seguir una serie de pasos.

```
def resolverJuego(matriz, sistema_ecuaciones):
    A, B = crearA_B(matriz, sistema_ecuaciones)

    # Matriz aumentada
    AB  = np.concatenate((A,B),axis=1)
    AB0 = np.copy(AB)

    tamano = np.shape(AB)
    n = tamano[0]
    m = tamano[1]
    AB = pivoteo_parcial(AB, n, m)
    AB1 = np.copy(AB)

    # eliminación hacia adelante (por filas)
    cont = 1
    for i in range(0,n-1,1):
        AB = pivoteo_parcial(AB,n, m)
        pivote   = AB[i,i]
        adelante = i + 1
        for k in range(adelante,n,1):  # k inicia de adelante hasta la últim fila avanzando fila por fila
            if(pivote!= 0):
                factor  = AB[k,i]/pivote
                AB = AB.astype(int)
                print(cont)
                cont = cont + 1
                AB[k,:] = (AB[k,:]) ^ (AB[i,:]* int(factor))

    # sustitución hacia atrás
    ultfila = n-1
    ultcolumna = m-1
    X = np.zeros(n,dtype=float)

    for i in range(ultfila,0-1,-1):  # Para incluir el primera fila valor (0-1) en pasos de -1
        suma = 0
        for j in range(i+1,ultcolumna,1):
            suma = suma + AB[i,j]*X[j]
        b = AB[i,ultcolumna] #valor de la matriz B en la fila i
        #X[i] = (b-suma)/AB[i,i]
        X[i] = (b - suma) % 2

    X = np.transpose([X])

    # SALIDA
    print(X)
    return X
```

pivoteo_parcial(AB, n, m): Este método devuelve la matriz AB pero en su respectiva diagonal estará el mayor elemento de cada fila(En nuestro caso, el mayor elemento siempre será 1)

```
def pivoteo_parcial(AB, n, m):
    # Pivoteo parcial por filas

    # Para cada fila en AB
    for i in range(0,n-1,1):
        # columna desde diagonal i en adelante
        columna  = abs(AB[i:,i])  #todas las filas desde i hasta el final de la matriz y solo la columna i.
        dondemax = np.argmax(columna) #identifica la posición del elemento más grande en columna.

        # dondemax no está en diagonal
        if (dondemax !=0):
            # intercambia filas
            temporal = np.copy(AB[i,:])
            AB[i,:] = AB[dondemax+i,:]
            AB[dondemax+i,:] = temporal
    return AB
```

Respuestas a interrogantes planteadas:

1. *¿Es necesario que alguna de las luces sea presionada más de una vez para ganar el juego?*

No, ya que si se presionara una luz una cantidad par de veces, el resultado sería el mismo a no haberla presionado.

2. *¿El orden en que se presionan las luces en una solución es relevante?*

No, el orden no es relevante ya que no importa en qué orden se apliquen las transformaciones, la cantidad total de cambios de luz de cada celda serán los mismos.

3. *En base a las respuestas de las preguntas anteriores, propondremos como modelo matemático para las soluciones del juego un vector* $v = (x1, x2, \ldots, xn^2)$ *de tamaño n 2 . Dado* $i \in \{1, 2, \ldots, n^2\}$ *usamos la división entera para escribir* $i - 1 = n \cdot q + r$ *con* $0 \leq r < n$. *Luego definimos xi = 1 si la luz en la entrada a(q+1)(r+1) debe ser presionada en la solución y xi = 0 en caso contrario.*
Justifique por qué el modelo matemático anterior es correcto para modelar las soluciones del juego.

El modelo matemático es correcto ya que el vector v, al tener tamaño n^2 contiene a todos los elementos de la matriz n x n, luego i se define como el subíndice de cualquiera de estas posiciones, por lo que también puede tomar cualquier valor de 1 a n^2, por último, la división entera hace referencia a la división de (i - 1) / n, la cual nos deja con que q y r solo pueden valer un número menor a n hasta el 0 inclusive, haciendo que la entrada a(q+1)(r+1) siempre se encuentre en el rango de 1 a n. El definir xi = 1 para encender una luz funciona ya que solo hay dos estados posibles de acción para tomar en una celda, o la apretamos, o no la apretamos.

4. *Justifique detalladamente la construcción de las otras ecuaciones del sistema en este ejemplo.*

$x1 + x2 + x4 = 0$

$x1 + x2 + x3 + x5 = 1$

$x2 + x3 + x6 = 0$

$x1 + x4 + x5 + x7 = 1$

$x2 + x4 + x5 + x6 + x8 = 1$

$x3 + x5 + x6 + x9 = 0$

$x4 + x7 + x8 = 0$

$x5 + x7 + x8 + x9 = 0$

$x6 + x8 + x9 = 1$

Aquí las otras ecuaciones se construyen siguiendo el ejemplo de la primera, seleccionando el siguiente elemento de $a_{i,j}$ generando una ecuación en base los valores del x relacionado a ese elemento y a los adyacentes de la lista (a+1 , j), (a-1 , j), (a , j+1), (a , j-1). Luego el resultado final de la ecuación es la suma de todos sus valores, teniendo en cuenta que en nuestro caso, 1+1 = 0.

Viendo este sistema de ecuaciones, podemos ver que los valores de x = 1 son x5 y x9. Esto se puede observar viendo que las únicas ecuaciones con un valor igual a 1 son aquellas donde el 5 o el 9 se encuentran de forma individual, y en las que están juntas la ecuación es igual a 0. Podemos ver que no hay más valores ya que no aparece la influencia de más valores en ninguna otra ecuación.

5. Verifique que la solución encontrada anteriormente donde $x5 = x9 = 1$ y $xi = 0$ para $i \in \{1, 2, 3, 4, 6, 7, 8\}$ es solución del sistema de ecuaciones.

Para verificar la solución haremos una sustitución en todo el sistema con los valores respectivos, nos quedarían estas cuentas:

1) $0 + 0 + 0 = 0$
2) $0 + 0 + 0 + 1 = 1$

3) $0 + 0 + 0 = 0$
4) $0 + 0 + 1 + 0 = 1$
5) $0 + 0 + 1 + 0 + 0 = 1$
6) $0 + 1 + 0 + 1 = 0$
7) $0 + 0 + 0 = 0$
8) $1 + 0 + 0 + 1 = 0$
9) $0 + 0 + 1 = 1$

Las soluciones son todas correctas, la mayoría de estas cuentas son bastante usuales y solo debemos estar atentos a los casos 6 y 8 donde debemos recordar que $1 + 1 = 0$.

Conclusiones

1- En el transcurso de este proyecto, hemos alcanzado con éxito los objetivos que nos propusimos desde el inicio.
2- Desarrollamos un programa capaz de presentarnos el juego Lights Outs, y decidir al usuario de qué dimensiones será su matriz, visualizarla y obtener el vector solución para dicha matriz.
3- Hemos implementado una solución que se representa a través de un vector binario, lo que permite a los usuarios comprender rápidamente el estado de las luces y cómo deben cambiar para ganar el juego.
4- Permitimos que el usuario compruebe la solución dentro de nuestro software.
5- Presentamos el juego con una interfaz gráfica, esto proporciona una experiencia más amigable para el usuario y facilita la interacción con el juego.

Pasos a futuro

Como pasos futuros pensamos que podríamos mejorar el diseño de la interfaz gráfica, haciéndola más bonita visualmente y agregarle la cantidad de movimientos realizados por el usuario.

A largo plazo también podríamos haber buscado otras formas de crear el vector binario de solución a través de otros métodos matemáticos para resolver el problema con

matrices que no presentan una clara diagonal (aunque con matrices 2x2 no tenemos problema).

Bibliografía

Presentaciones de clase de Google Colabs.
https://es.wikipedia.org/wiki/Puerta_XOR