Solución Examen Zinobe

Kevin Sebastian Pineda Jaramillo
Kevin@gmail.com
Zinobe

Planteamiento del Problema

En el siguiente documento pretende describir los pasos para llegar a la solución de 4 problemas planteados por la empresa Zinobe, estos problemás son:

- Implementar una función en Python tal que dada una matriz M con entradas en $\{0,1\}$, encontrar la submatriz más grande donde todas sus entradas son 0. Estas función debe estar definida como:

  ```
  def maximo_rectangulo(M):
      pass
  ```

  si se evalua la matriz M en nuestra función, debe returnar la tupla (i,j,a,b), donde (i,j) son los indices de la primera entrada de la submatriz y (a,b) son los anchos y largos de la submatriz.

- Implementar una función en Python tal que dada una matriz M con entradas en $\{0,1\}$, encontrar la submatriz cuadrada más donde todas sus entradas son 0. Estas función debe estar definida como:

  ```
  def maximo_cuadrado(M):
      pass
  ```

  si se evaluá la matriz M en nuestra función, debe retornar la tupla (i,j,a), donde (i,j) son los indices de la primera entrada de la submatriz y a es la longitud de un lado de nuestro cuadrado.

- Implementar una función en Python tal que dada una matriz M con entradas en $\{0,1\}$ y una entrada (i,j), esta función retorna un iterador que genera celdas libres y este forma un camino.Esta función debe estar definida como:

  ```
  def iterador_celdas_libres(M):
      pass
  ```

- Resolver usando Python el siguiente sistema nolineal con condiciones:

$$
\begin{gathered}
-F_r + \omega_x = ma \\
N - \omega_y = 0 \\
F_r = N\mu \quad \omega_x = \omega \sin\theta \quad \omega_y = \omega\cos\theta \\
\mu = 0{,}2 \quad \omega = mg \quad a = 0{,}012\frac{m}{s^2} \quad g = 10\frac{m}{s^2} \quad m = 10kg
\end{gathered}
$$

La solución a los dos primeros items pueden ser dada gracias al algoritmo de Kadane que nos permite determinar la matriz con mayor suma y sus indices, (este algoritmo sera descrito en la siguiente sección) el tercer algoritmo se resolverá con los objetos básicos de Python y por último el sistema de ecuaciones nolineales sera resuelto con métodos de la librería Scipy.

Solución teórica

Algoritmo de Kadane

Antes de observar el algoritmo Kadane en 2D, estudiamos el algoritmo que nos permite hallar el cual encuentra el máximo subarreglo con máxima suma en una lista de elementos $a(x_1, ..., x_n)$, este algorimo esta dado como:

```
def kadane1D(array):
    sumMax = 0 , t=0
    ind= 0
    for j in range(len(array)):
        t+=array[j]
        if t>sumMax:
            sumMax=t
            ini,finish=ind,j
        if t<=0:
            t=0
            ind=j+1
    return sumMax,ini,finish
```

esto nos permite encontrar ella suma máxima en subarreglos y el arreglo $a[init : finish]$, y puede ser extendido en a 2D.

Para generalizar nuestro algoritmo, creamos un lista de ceros cuya longitud es el número más grande entre filas y columnas, esto para recolectar las sumas parciales que nos va generando el algoritmo de Kadane en 1D,y recorriendo nuestra matriz por y calculando subarreglo con suma máxima para cada fila de nuestra matriz, podemos encontrar nuestros valores. Se puede ver en el siguiente código:

```
def kadane(A,min_l,max_l):
    M = convert_matriz(A)
    sum_max = 0
    row_t ,col_t = 0,0
    row_f ,col_f = 0,0
    for g in range(min_l):
        sum_par = [0]*max_l
        for i in range(g, min_l):
            t=h=0
            for j in range(max_l):
                try:
                    sum_par[j]+= M[i][j]
                except IndexError:
                    continue
                    sum_par[j]+= M[j][i]
                t+=sum_par[j]
                if t>sum_max:
                    sum_max = t
                    row_t,col_t= h,g
                    row_f,col_f = j,i
                if t<=0:
                    t=0
                    h=j+1
    return sum_max,row_t,col_t,row_f,col_f
```

esta función nos retoma los indices de la primera y ultima entrada de nuestro submatriz con suma máxima. Sin embargo, como nuestra matriz es de ceros y uno haciendo uso de la librería Numpy en Python, podemos

convertir las entradas de la matriz que son iguales a 1 con menos el máximo entero que toma el es sistema y nuestro entradas con valores iguales a 0 las convertimos en 1, es decir,
$1 \to -9223372036854775807$
$0 \to 1$
esto puede ser visto en la siguiente función:

```
def convert_matriz(A):
    A = np.array(A, dtype=np.float)
    A = np.where(A==1,-sys.maxsize,A)
    A = np.where(A==0,1,A)
    return A
```

así la suma máxima va hacer la submatriz, donde todas sus entradas son cero.

Solución problemas

Máximo rectángulo

Para crear nuestra función *maximo_rectangulo*, nosotros creamos dos condicionales para determinar cuando las filas son mayores a las columnas o de manera dual las columnas mayores que las filas, así de esta manera me genera una lista con los datos que me proporciona el algortimo de Kadane.
Puedo calcular el ancho y el alto para que nuestra función pueda retornar estos valores, esto se puede ver en a siguiente figura: Tambien se maneja algunas excepciones, como son:

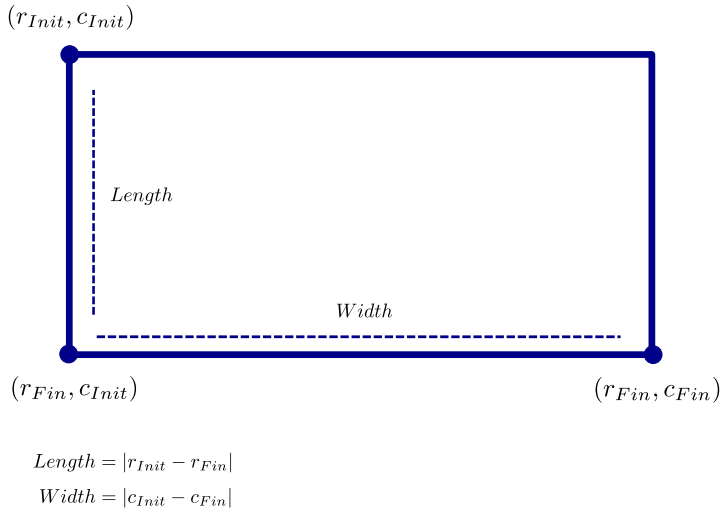


Figura 1: Rectángulo.

- Matriz con solo entradas en 1: se convierte todos los elementos de la matriz en un conjunto, como todos son iguale entonces el conjunto es igual a $\{1\}$.
- Matriz vacía: se convierte todos los elementos de la matriz en un conjunto como no hay entonces la longitud de este conjunto es igual a 0.
- Matriz con entradas diferentes a 0 y 1: se convierte la matriz en un conjunto y se verifica la longitud de este como tiene elementos diferentes a $0, 1$ entonces la longitud es mayor que 2.
- Matriz con entradas no enteras: se usa la sentencia *try-exception* y la excepcion *ValueError*, y se imprime el mensaje de que la matriz no cumple con las condiciones.

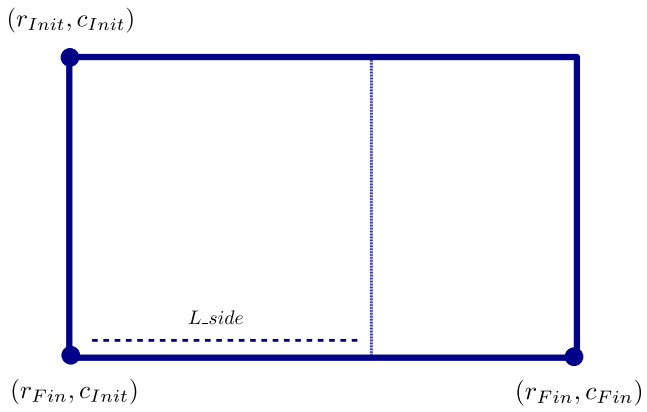
ahora si, nuestra función esta dada como :

```
def maximo_rectangulo(M):
    try:
        if len(set(chain.from_iterable(M)))>2:
            return "La matriz posee números distintos de 1 y 0"
        elif set(chain.from_iterable(M)) == {1}:return (0,0,0,0)
        elif len(set(chain.from_iterable(M))) ==0 :
            return "La matriz no posee elementos"
        len_col,len_row=len(M[0]),len(M)
        if len_col >= len_row:
            date = list(kadane(M,len_row, len_col))
            width = abs(date[1]-date[3])+1
            length= abs(date[2]-date[4])+1
            return date[2],date[1], width,length
        else:
            date = list(kadane(M,len_col, len_row))
            width = abs(date[1]-date[3])+1
            length = abs(date[2]-date[4])+1
            return date[2],date[1], width, length
    except ValueError:
        return "La matriz no cumple las condiciones"
```

al hacer esto, nuestra función nos retorna una coordenada con las siguentes entradas $(r_{Init}, c_{Init}, width, length)$.

Máximo cuadrado

Razonamos de manera similar a la función *maximo_rectangulo* pero en vez de tomar el ancho y el la longitud de un lado de nuestro cuadrado esta dado como el mínimo entre el ancho y el largo, esto lo podemos ver en la siguiente figura:



$$Length = |r_{Init} - r_{Fin}|$$
$$Width = |c_{Init} - c_{Fin}|$$
$$L\_side = \min(Lenght, Width)$$

Figura 2: Cuadrado.

nuestra función posee las mismas excepciones que la función *maximo_rectangulo*, aunque *maximo_cuadrado* retorna la siguiente coordenada $(r_{Init}, c_{Init}, \text{mín}\,(width, length))$, y su código en Python esta dado como:

```
def maximo_cuadrado(M):
    try:
        if set(chain.from_iterable(M)) == {1}:return (0,0,0)
        elif len(set(chain.from_iterable(M)))>2:
            return "La matriz posee números distintos de 1 y 0"
        elif len(set(chain.from_iterable(M))) ==0 :
            return "La matriz no posee elementos"
        len_col,len_row=len(M[0]),len(M)
        if len_col >= len_row:
            date = list(kadane(M,len_row, len_col))
            width = abs(date[1]-date[3])+1
            length = abs(date[2]-date[4])+1
            return date[2],date[1],  min(length, width)
        else:
            date = list(kadane(M,len_col, len_row))
            width = abs(date[1]-date[3])+1
            length = abs(date[2]-date[4])+1
            return date[2],date[1], min(length, width)
    except ValueError:
        return "La matriz no cumple las condiciones"
```

Camino iterable

Para esta función, primero pensé en como crear una lista que cuyas entradas son los elementos de de la vecindad de una celda, y una función que me almacenara en un conjunto todas las parejas de indices de las vecindad de una celdas, esto puede verse en las siguientes matrices:

$$\begin{pmatrix} & \vdots & \vdots & \vdots & \\ \cdots & A & B & C & \cdots \\ \cdots & D & E & F & \cdots \\ \cdots & G & H & I & \cdots \\ & \vdots & \vdots & \vdots & \end{pmatrix} \qquad \begin{pmatrix} & \vdots & \vdots & \vdots & \\ \cdots & (i-1,j-1) & (i-1,j) & (i-1,j+1) & \cdots \\ \cdots & (i,j-1) & (i,j) & (i,j+1) & \cdots \\ \cdots & (i+1,j-1) & (i+1,j) & (i+1,j+1) & \cdots \\ & \vdots & \vdots & \vdots & \end{pmatrix}$$

$$\begin{aligned} element(M,i,j) &= [A,B,C,D,E,F,G,H,I] \\ neigh &= \{(i-1,j-1),(i-1,j),(i-1,j+1),(i,j-1), \\ &\quad (i,j),(i,j+1),(i+1,j-1),(i+1,j),(i+1,j+1)\} \end{aligned}$$

estas funciones pueden ser escritas fácilmente a partir de una función lambda como sigue:

```
element = lambda M,row,col: [M[i][j] for i in range(row-1,row+2) for j in range(col-1,col+2)
                             if (0 <= i < len(M) and 0 <= j < len(M[0]))]
neigh   = lambda M,row,col: {(i,j) for i in range(row-1,row+2) for j in range(col-1,col+2)
                             if (0 <= i < len(M) and 0 <= j < len(M[0]))}
```

Ahora, nuestro función *iterador_celdas_libres* va estar dada de la siguiente manera, primero recorremos las celdas anteriores a nuestra celda dada (i,j) retrocedemos en las columnas con una sentencia **For** y retrocedemos en nuestras filas con una sentencia **For**.

De esta manera nuestro código va calculando la suma de los elementos de nuestra vecindad en la celda en la que esta actualmente, y verifica que la celda en la que esta posicionada este en nuestro conjunto de vecindades de celdas anteriores, si cumple esta posición nuestra celda sera almacenada en una lista y se actualiza nuestro conjunto de vecindades.

Pensando de manera similar se crea un sentencia **For** para que vaya creciendo nuestras filas y realice el mismo calculo, después se reescribe la misma idea pero en vez de disminuir en columnas, el aumenta en ellas. El código de nuestra función esta dado como:

```
def equation(variable):
    (N,t)= variable
    #constans
    mu, mass , gravity, acce = 0.2, 10, 10, 0.012
    w = mass*gravity
    #assignment
    F_r = N*mu
    w_x = w*sin(t)
    w_y = w*cos(t)
    #Equation system
    system_q = [-F_r+w_x-mass*acce, N-w_y]
    return system_q
solution = opt.fsolve(equation, (0,2*pi) )
```

Esta función posee algunas excepciones y son las siguientes:

- No existe caminos: si damos una entrada en nuestra matriz y la vecindad posee un 1, nuestra función nos arroja un mensaje diciendo que no existen caminos para esa celda.
- Matriz vacía: se convierte todos los elementos de la matriz en un conjunto como no hay entonces la longitud de este conjunto es igual a 0.
- Matriz con entradas diferentes a 0 y 1: se convierte la matriz en un conjunto y se verifica la longitud de este como tiene elementos diferentes a $0, 1$ entonces la longitud es mayor que 2.
- Matriz con entradas no enteras: se usa la sentencia *try-exception* y la excepcion *ValueError*, y se imprime el mensaje de que la matriz no cumple con las condiciones.

Con esto para terminar nuestra lista de elementos que son caminos de una celda seran convertidos en un iterador, para facilitar más los cálculo entonces nuestra función nos retorna un elemento iterable.

Ejercicio plus

Para resolver nuestro sistema de ecuaciones, usamos el metodo *fsolve()* del modulo de la *optimize* de nuestra libreria de Scipy de Python, entonces como nuestro metodo solo acepta funciones, se crea una función que retorne mi sistema de ecuaciones como sigue:

```
def equation(variable):
    (N,t)= variable
    #constans
    mu, mass , gravity, acce = 0.2, 10, 10, 0.012
    w = mass*gravity
    #assignment
    F_r = N*mu
    w_x = w*sin(t)
    w_y = w*cos(t)
    #Equation system
    system_q = [-F_r+w_x-mass*acce, N-w_y]
    return system_q
solution = opt.fsolve(equation, (0,2*pi) )
```

y así la solución queda guardada en la variable *solution*.

Pruebas

Se añaden una serie de pruebas, en nuestro archivo solucion de python con algunas matrices que cumplen las condiciones, y también con otras que no permiten ser evaluadas en nuestras función. Sin embargo, se crea un test unitarios para nuestras dos primeras funciones usando la libreria *unittest*. En nuestra carpeta podemos encontrar los archivos para crear nuestro archivo *data.json* y un archivo en Python que nos permite añadir as pruebas para complementar nuestro archivo *json*.