Algoritmia y Complejidad
Entrega ejercicios 4

Grado en Ingeniería Informática

Raúl López Llana
Grupo Laboratorio: Miércoles 12.00 – 14.00 h.

Índice

Problema 4

Enunciado:
Alí Babá ha conseguido entrar en la cueva de los ciento un mil ladrones, y ha llevado consigo su camello junto con dos grandes alforjas; el problema es que se encuentra con tanto tesoro que no sabe ni qué llevarse. Los tesoros son joyas talladas, obras de arte, cerámica... es decir, son objetos únicos que no pueden partirse ya que entonces su valor se reduciría a cero.
Afortunadamente los ladrones tienen todo muy bien organizado y se encuentra con una lista de todos los tesoros que hay en la cueva, donde se refleja el peso de cada pieza y su valor en el mercado de Damasco. Por su parte, Alí sabe la capacidad de peso que tiene cada una de las alforjas.
Diseñar un algoritmo que, teniendo como datos los pesos y valor de las piezas, y la capacidad de las dos alforjas, permita obtener el máximo beneficio que podrá sacar Alí Babá de la cueva de las maravillas

A partir de un ejemplo, se procederá a la explicación:

Partimos del siguiente ejemplo:

La intención es meter dentro de cada alforja la mejor combinación posible de objetos para poder llevarnos el valor más alto en total.

Para ello primero creamos una tabla con las combinaciones posibles con el valor que obtendríamos (programación dinámica).

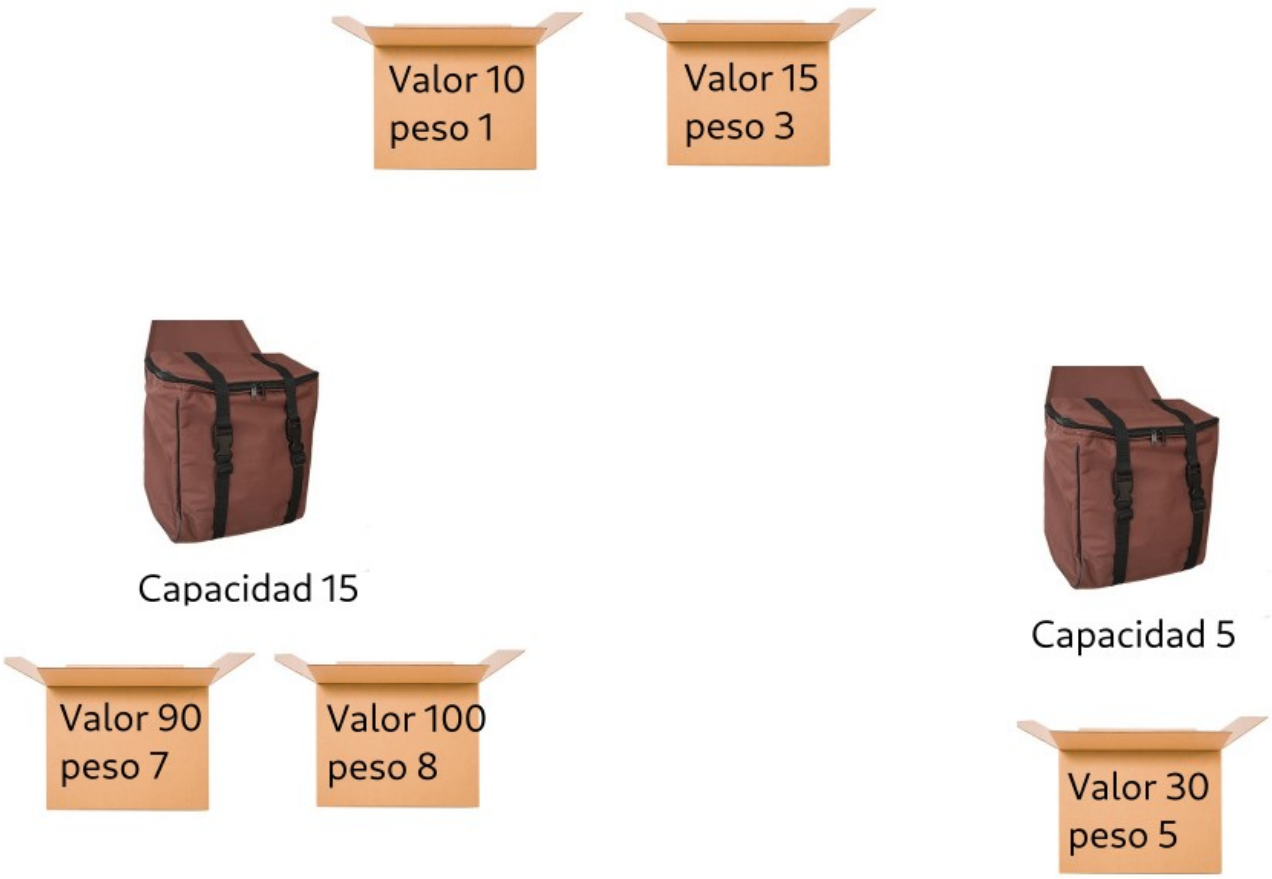
```
[0, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10]
[0, 10, 10, 15, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25]
[0, 10, 10, 15, 25, 30, 40, 40, 45, 55, 55, 55, 55, 55, 55, 55]
[0, 10, 10, 15, 25, 30, 40, 90, 100, 100, 105, 115, 120, 130, 130, 135]
[0, 10, 10, 15, 25, 30, 40, 90, 100, 110, 110, 115, 125, 130, 140, 190]
```

Tras tener rellena la tabla con las combinaciones posibles maximizando el resultado, podemos recorrerla de forma inversa a como ha sido rellenada para poder ir viendo que objetos me tengo que llevar.

```
[0, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10]
[0, 10, 10, 15, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25]
[0, 10, 10, 15, 25, 30, 40, 40, 45, 55, 55, 55, 55, 55, 55]
[0, 10, 10, 15, 25, 30, 40, 90, 100, 100, 105, 115, 120, 130, 130, 135]
[0, 10, 10, 15, 25, 30, 40, 90, 100, 110, 110, 115, 125, 130, 140, 190]
```

Al hacer esto ya sabemos que objetos tenemos que meter en la primera alforja.

Ahora tendremos que realizar los mismos pasos pero con la otra alforja y los objetos restantes y nos quedará el siguiente resultado.



Código:

```
'''Defino una clase objeto que contendrá el valor y el peso de cada objeto'''
import math


class objeto:

    def __init__(self, valor, peso):
        self.valor = valor
        self.peso = peso

    def setValor(self, valor):
        self.valor = valor

    def setPeso(self, peso):
        self.pesp = peso

    def getValor(self):
        return self.valor

    def getPeso(self):
        return self.peso
```

```
'''Funcion que rellenará las alforjas'''


def llenarAlforja(capacidadAlforja, objetos):
    '''Primero me debo crear la tabla dinámica'''
    tabla = []
    objetosSelecionadas = []
    '''tantas filas como objetos'''
    for i in range(len(objetos)):
        aux = []
        '''tnatas columnas como el peso máximo que pueda exa alforja'''
        for a in range(capacidadAlforja):
            aux.append(0)
        tabla.append(aux)
    '''Teniendo ya la tabla creada y con todo a 0
    Recorro de nuevo la tabla para rellenarla'''

    for fila in range(len(objetos)):
        for columna in range(capacidadAlforja):
            #print("---- FILA: ",fila," COLUMNA: ",columna," -------")


            '''Primera columna es que puedo cargar un peso de 0'''
            if (columna == 0):
                tabla[fila][columna] = 0
            else:
                ''' Tengo que comparar si el objeto de esta fila entra en la mochila
                Tengo que mirar si es más rentable que lo que teniamos antes.
                 Al tener los objetos ordenados por peso ascendiente, solo  comparo
con el primer objeto'''

                if (objetos[fila].getPeso() <= columna+1):

                    # print("objeto peso: ",objetos[fila].getPeso())
                    # print("objeto valor: ",objetos[fila].getValor())
                    '''El objeto podria entrar dentro de la mochila'''
                    '''Miro si es rentable'''
                    if (fila == 0):
                        tabla[fila][columna] = objetos[fila].valor
                    else:
                        anterior = tabla[fila - 1][columna]
                        propuesta = objetos[fila].valor + tabla[fila - 1][columna -
objetos[fila].getPeso()]
                        if (anterior < propuesta):
                            '''Es mejor el nuevo objeto y lo inserto en al alforja'''
                            tabla[fila][columna] = propuesta
                        else:
                            '''Es mejor quedarme con lo que ya tenia'''
```

```
                    tabla[fila][columna] = tabla[fila - 1][columna]
            else:
                '''No puedo meter el nuevo objeto en la alforja
                sigo con lo que tuviese antes puesto'''
                tabla[fila][columna] = tabla[fila - 1][columna]

    '''Una vez rellena la tabla,selecciono el conjunto de objetos que me
maximice el valor'''
    '''Para ello voy a recorrer la tabla de de abajo a la derecha hacia arriba a la
izquierda (descendiente)'''
    capacidadParcial = capacidadAlforja
    '''Ultima posicion de la tabla'''
    ultimo = tabla[len(objetos) - 1][capacidadAlforja - 1]
    for i in range(len(objetos) - 1, -1, -1):
        '''Desde el total de objetos hasta 0 con un paso de -1'''

        '''si llego a 0 antes de terminar de recorrer la tabla, paro'''
        if (capacidadParcial <= 0):
            break
        if (i != 0):
            if (ultimo == tabla[i - 1][capacidadParcial - 1]):
                continue
            else:
                '''Los valores no son iguales'''
                elemento = objetos.__getitem__(i)
                objetosSelecionadas.append(elemento)
                ultimo = ultimo - elemento.getValor()
                capacidadParcial = capacidadParcial - elemento.getPeso()

        else:
            '''Los valores no son iguales'''
            elemento = objetos.__getitem__(i)
            objetosSelecionadas.append(elemento)
            ultimo = ultimo - elemento.getValor()
            capacidadParcial = capacidadParcial - elemento.getPeso()
    return objetosSelecionadas


def quitarSeleccionados(objetos, seleccionados):
    '''Selecionados contendrá el valor de los objetos ya introducidos en una
alforja'''
    for va in seleccionados:
        if (objetos.__contains__(va)):
            objetos.remove(va)
    return objetos


def getValores(objetos):
    total = 0
    for ob in objetos:
```

```
        total += ob.getValor()
    return total


def listarObjetos(lista):
    print("-----------------------------")
    for a in lista:
        print("Peso: ", a.getPeso(), " Valor: ", a.getValor())
    print("-----------------------------")


def rellenarAlforjas(objetos, alforja1, alforja2):
    capacidadMaxAlforja = max(alforja1, alforja2)
    '''primero relleno una alforja, la que tenga mayor capacidad'''
    seleccionados = llenarAlforja(capacidadMaxAlforja, objetos)
    '''quito los elementos que ya he seleccionado'''

    print("seleccionados alforja con mayor capacidad")
    listarObjetos(seleccionados)

    objetos2 = quitarSeleccionados(objetos, seleccionados)
    '''relleno la otra alforja'''

    capacidadMinAlforja = min(alforja1, alforja2)
    print("capacidad de la segunda alforja: ", capacidadMinAlforja)
    seleccionados2 = llenarAlforja(capacidadMinAlforja, objetos2)

    print("seleccionados alforja con menor capacidad")
    listarObjetos(seleccionados2)

    objetos3 = quitarSeleccionados(objetos2, seleccionados2)

    print("Objetos no seleccionados")
    listarObjetos(objetos3)

    '''devuelvo el valor que me llevo en cada alforja'''
    return getValores(seleccionados), getValores(seleccionados2)


'''Defino objetos'''
objetos = []
'''El peso debe estar ordenado de menor a mayor'''
valor = [200, 200, 10, 10]
peso = [1,1,2,2]

'''valor = [200, 150, 10, 5]
peso = [3,1,1,2]'''
for i in range(0, len(peso)):
    objetos.append(objeto(valor[i], peso[i]))
```

```
objetosFinales = []

'''Ordeno por peso'''
while (objetos):
    pesoMax = math.inf
    valorMax=0
    posicion = 0
    sel = 0
    for a in objetos:
        if (a.getPeso() < pesoMax):
            pesoMax = a.getPeso()
            valorMax=a.getValor()
            menor = a
            sel = posicion
        if (a.getPeso()== pesoMax):
            if(a.getValor()< valorMax):
                pesoMax = a.getPeso()
                valorMax = a.getValor()
                menor = a
                sel = posicion
        posicion = posicion + 1
    objetosFinales.append(menor)
    objetos.pop(sel)

'''peso límite alforjas'''
alforja1 = 3
alforja2 = 3

max, min = rellenarAlforjas(objetosFinales, alforja1, alforja2)

print("Una alforja lleva un valor de ", max)
print("Otra alforja lleva un valor de ", min)
```

Explicación:

Definimos un array con los objetos dado su valor y su peso. Creo las 2 alforjas con su capacidad máxima.

Llamo a la función “rellenarAlforjas” en la que primero miro cual es la alforja que tiene más capacidad y sobre esa llamo a “llenarAlforja” pasandole la capacidad de dicha alforja y la lista de objetos disponibles.
Dentro de esta función primero me creo la tabla dinámica rellena de 0 y posteriormente la vulvo a recorrer pero esta vez con tantas iteraciones como objetos tenga. Por cada iteración rellenaré una de las filas de la tabla con el valor máximo que me podría llevar en ese instante con el objeto actual y los anteriores.

Una vez rellena la tabla, miro desde la ultima posición hacia la primera y comprueba si se ha seleccionado el objeto de esa fila o no. En caso de haberse seleccionado, me almaceno el objeto y me desplazo a una fila inferior restando el peso del objeto seleccionado. En caso de que no se haya seleccionado ese objeto (la misma posición en una fila inferior tiene el mismo valor), desciendo una fila.

Al llegar a capacidad restante de la alforja a 0 o al final de tabla tendré almacenados los objetos que meteré en esa alforja y devuelvo un array con esos objetos.

Borro del array de objetos candidatos los que ya he seleccionado y vuelvo a llamar a la función para que ahora la haga con los objetos restantes y con la otra alforja

Por último, muestro por pantalla el valor que se lleva en cada alforja.

Ejemplo de ejecución:

Para este ejemplo hemos creado los siguientes objetos :

```
valor = [10, 15, 30, 90, 100]
peso =  [1,  3,  5,  7,   8]
```

Un objeto está compuesto por un peso y un valor asociado. En este programa está definido para que dada una posición se encuentre su valor y su peso en dicha posición de su array. Por ejemplo, El objeto 3 está compuesto por un peso de 5 y un valor de 30.

También para este ejemplo hemos definido 2 alforjas con las siguientes capacidades de peso:

```
alforja1 = 15
alforja2 = 5
```

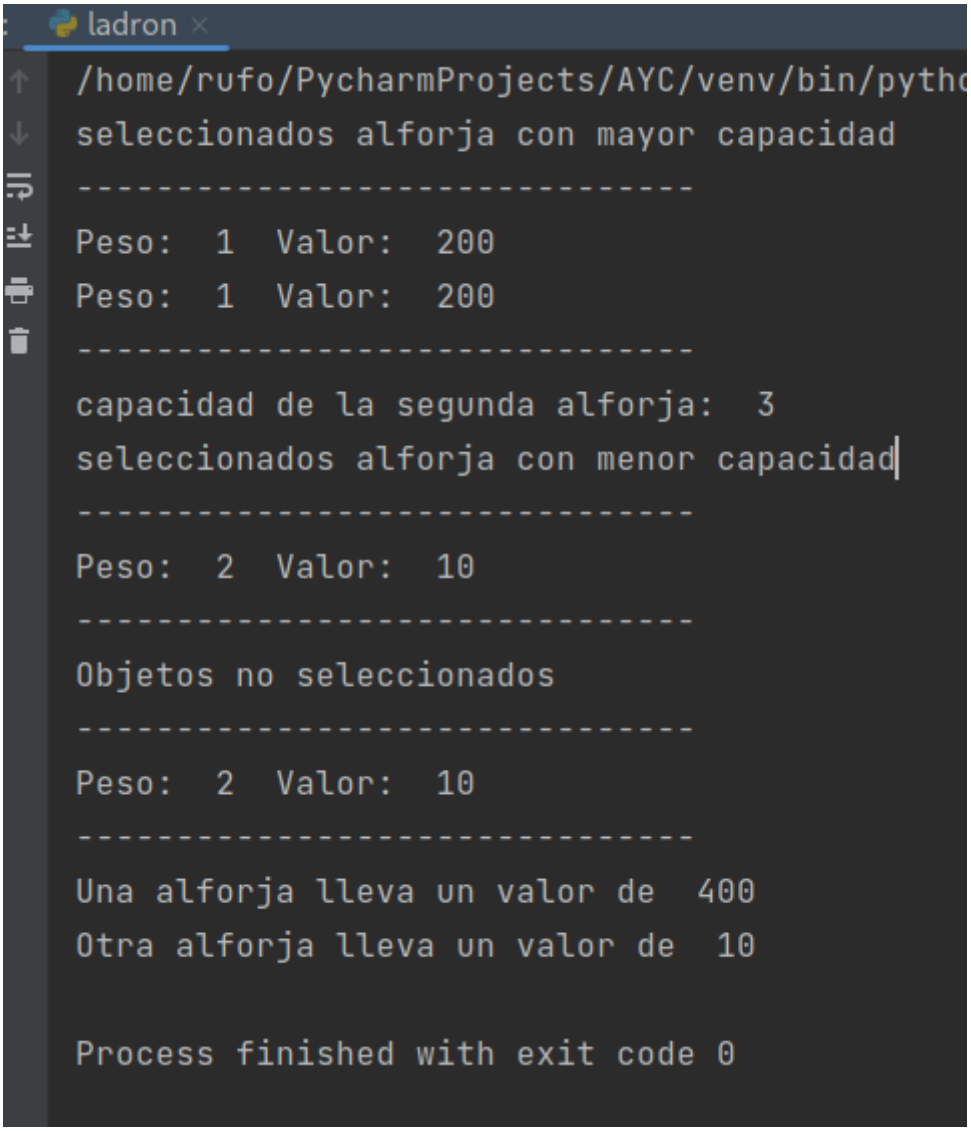
Sabiendo esto, el resultado que obtenemos es el siguiente:

```
seleccionados alforja con mayor capacidad
------------------------------
Peso:  8  Valor:  100
Peso:  7  Valor:  90
------------------------------
seleccionados alforja con menor capacidad
------------------------------
Peso:  5  Valor:  30
------------------------------
Objetos no seleccionados
------------------------------
Peso:  1  Valor:  10
Peso:  3  Valor:  15
------------------------------
Una alforja lleva un valor de  190
Otra alforja lleva un valor de  30
```

Caso de prueba propuestos

Primer caso.

- **Número de tesoros:** 4
  - **Valores:** 200, 200, 10, 10
  - **Pesos:** 1, 1, 2, 2
- **Capacidad peso 1ª alforja:** 3
- **Capacidad peso 2ª alforja:** 3



Segundo caso.

- **Número de tesoros:** 4
  - **Valores:** 200, 150, 10, 5
  - **Pesos:** 3, 1, 1, 2
- **Capacidad peso 1ª alforja:** 3
- **Capacidad peso 2ª alforja:** 3

```
seleccionados alforja con mayor capacidad
------------------------------
Peso:  3  Valor:  200
------------------------------
capacidad de la segunda alforja:  3
seleccionados alforja con menor capacidad
------------------------------
Peso:  1  Valor:  150
Peso:  1  Valor:  10
------------------------------
Objetos no seleccionados
------------------------------
Peso:  2  Valor:  5
------------------------------
Una alforja lleva un valor de  200
Otra alforja lleva un valor de  160
```

Problema 7

Enunciado:
Se define una secuencia de bits A como una sucesión A = {a1, a2, ..., a n} donde cada ai puede tomar el valor 0 o el valor 1, y n es la longitud de la secuencia A. A partir de una secuencia se define una subsecuencia X de A como X = {x1, x2, ..., x k}, siendo k ≤ n, de forma que X puede obtenerse eliminando algún elemento de A pero respetando el orden en que aparecen los bits; por ejemplo, si A = {1, 0, 1, 1, 0, 0, 1} podríamos obtener como subsecuencias {1, 1, 1, 0, 1}, {1, 0, 1} o {1, 1, 0, 0} entre otras, pero nunca se podría conseguir la subsecuencia {1, 0, 0, 1, 1}.
Dadas dos secuencias A y B, se denomina a X una subsecuencia común de A y B cuando X es subsecuencia de A y además es subsecuencia de B. (aunque puede que se hayan obtenido quitando distintos elementos en A que B, e incluso distinta cantidad).
Suponiendo las secuencias A = {0, 1, 1, 0, 1, 0, 1, 0} y B = {1, 0, 1, 0, 0, 1, 0, 0, 1}, una subsecuencia común sería X = {1, 1, 0, 1}, pero no podría serlo X = {0, 1, 1, 1, 0}.
Se desea determinar la subsecuencia común de dos secuencias A y B que tenga la longitud máxima, para lo que se pide

- explicar con detalle la forma de resolver el problema, y
- hacer un algoritmo de Programación Dinámica que obtenga la longitud máxima posible y una secuencia común de dicha longitud.

Ejemplo:
Para este ejercicio no voy a poner un ejemplo de cadenas ya que en el propio enuncia viene uno.

Código:

```
import math


def con(propuesta, cadena):
    '''Esta es la función principal, mira si una cadena es subcadena de otra'''
    posicion_aux = 0
    resul = True
    '''Recorro hasta ver que entra toda la propuesta en la cadena
    En caso contrario, salta una excepcion y me indica que no entra todo ya que
hemos llegado al final'''
    try:
        for i in propuesta:
            '''Recorro los bits'''
            while (i != cadena[posicion_aux]):
                posicion_aux = posicion_aux + 1
            posicion_aux = posicion_aux + 1
    except:
        resul = False
    return resul


def esSubcadena(propuesta, CadenaA, CadenaB):
    '''comparo la propuesta por separado con cada una de las cadenas'''
    result = False
    if (con(propuesta, CadenaA)):
```

```
        if (con(propuesta, CadenaB)):
            result = True
    return result


'''Esta funcion convierte un numero decimal a binario'''


def DecimalToBinary(n):
    return bin(n).replace("0b", "")


def crearTabla(cadenaA, cadenaB):
    '''Primero me debo crear la tabla dinámica'''
    tabla = []
    '''tantas filas como objetos'''
    for i in range(len(cadenaA)):
        aux = []
        '''tantas columnas como el peso máximo que pueda esa alforja'''
        for a in range(len(cadenaB)):
            aux.append(0)
        tabla.append(aux)
    '''Teniendo ya la tabla creada y con todo a 0
    Recorro de nuevo la tabla para rellenarla'''

    for fila in range(len(cadenaA)):
        for columna in range(len(cadenaB)):
            if (fila == 0):
                '''Si el caracter de la cadenaA se encuentra en alguna posicion de lo
que llevo de cadenaB'''
                if (cadenaA[0] in cadenaB[0:(columna + 1)]):
                    tabla[fila][columna] = 1
                else:
                    tabla[fila][columna] = 0
            elif (columna == 0):
                '''Si el caracter de la cadenaA se encuentra en alguna posicion de lo
que llevo de cadenaB'''
                if (cadenaB[0] in cadenaA[0:(fila + 1)]):
                    tabla[fila][columna] = 1
                else:
                    tabla[fila][columna] = 0
    return tabla


def mayor(cadenaA, cadenaB):
    '''Primero saco caul es la cadena con mayor longitud usando una tabla
dinamica'''
    mayor = crearTabla(cadenaA, cadenaB)

    '''Una vez creada la tabla, recorro todas las posiciones'''
```

```
    for fila in range(len(cadenaA)):
        for columna in range(len(cadenaB)):
            '''Si no estoy en el caso de que las dos listas estén vacias'''
            if (fila != 0 and columna != 0):
                '''Si el valor antorior es distinto a la fila + 1 y
                si el valor el bit de la primera cadena coincide con el bit de la
segunda
                '''
                if (mayor[fila][columna - 1] != (1 + fila) and cadenaA[fila] ==
cadenaB[columna]):
                    '''El valor anterior más 1'''
                    mayor[fila][columna] = mayor[fila][columna - 1] + 1

                else:
                    '''No se ha producido una mejora y me quedo con el valor
anterior'''
                    mayor[fila][columna] = mayor[fila][columna - 1]
    '''Muestro la tabla por consola'''
    print("tabla:")
    for a in mayor:
        print(a)
    '''El valor más alto se encuentra en la ultima posición '''
    mayor = mayor[len(cadenaA) - 1][len(cadenaB) - 1]
    posibilidades = []
    '''Sabiendo esto, ahora ya podemos crear las distintas posibilidades de
cadenas (binario)'''
    for posible in range(int(math.pow(2, mayor))):
        '''Añado los numero binarios'''
        posibilidades.append(DecimalToBinary(posible))
    resul = []
    '''Recorro todas las posibilidades mirando si son subcadenas de las 2 dadas'''
    for a in posibilidades:
        if (esSubcadena(a, cadenaA, cadenaB)):
            resul.append(a)
    '''Ahora ya tenemos almacenadas las cadenas que son subcadenas de las 2
dadas'''
    '''Solo queda devolver la última ya que será la más grande'''
    return resul[-1]


CadenaA = "01101010"
CadenaB = "101001001"
propuesta = "11011"
print("Cadena A: ", CadenaA)
print("Cadena B: ", CadenaB)

'''Primero miramos si la propuesta es subcadena de A y B'''
if (esSubcadena(propuesta, CadenaA, CadenaB)):
    print(propuesta, " es subcadena")
else:
```

```
    print(propuesta, " NO es subcadena")

print()
'''Ahora, en vamos a ver cual es la secuencia más grande posible'''
print("La cadena más grande posible para ser subcadena de las otras 2 es: ",
mayor(CadenaA, CadenaB))
```

Explicación:

Antes de nada, defino las dos cadenas principales y la cadena propuesta de ser subcadena.
Divido el ejercicio en 2 partes. La primera mira si la cadena propuesta es desencadena de las otras dos. La segunda parte mira cual es la cadena de máxima longitud posible que sea subcadena.

Para la primera parte, llamo a la función “esSubcadena” la cual compara por separado la una de las cadenas dadas con la propuesta y si para las dos la propuesta es desencadena, retorna True. Para ello lo que hago es mirar bit por bit desde la izquierda hacia la derecha de la cadena de referencia si se encuentran los bits de la propuesta. En caso de que tenga que mirar fuera del indice de la cadena de referencia, saltaría una excepción y me indica que no es desencadena.

Para la segunda parte me creo un array con todos los binarios posibles que tenga una longitud menor a la del mayor de las 2 cadenas de referencia. A continuación voy pasando cada una de estas cadenas propuestas a la función que hemos utilizado previamente para saber si es sub cadena de las 2 de referencia y me iré almacenando las que lo sean. Por último solo me quedará quedarme con la última añadida ya que será la de mayor longitud.

<u>Ejemplo de ejecución:</u>

Dada la cadena A “11101” y la cadena B “010101”, vemos que la cadena “11011” no puede ser subcadena de las otras 2.
Y posteriormente, calculo cual es la cadena de longitud máxima que puede ser subcadena de las 2 dadas

```
Cadena A:  11101
Cadena B:  010101
11011  NO es subcadena
La cadena más grande posible para ser subcadena de las otras 2 es:  1101
```

Casos de prueba:

Dada las cadenas “011” y “1010”

```
Cadena A:  011
Cadena B:  1010
11  es subcadena

tabla:
[0, 1, 1, 1]
[1, 1, 2, 2]
[1, 1, 2, 2]
La cadena más grande posible para ser subcadena de las otras 2 es:  11
```

Dada las cadenas “01101010” y “101001001”

```
Cadena A:  01101010
Cadena B:  101001001
11011  es subcadena

tabla:
[0, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 2, 2, 2, 2, 2, 2, 2]
[1, 1, 2, 2, 2, 3, 3, 3, 3]
[1, 2, 2, 3, 4, 4, 4, 4, 4]
[1, 1, 2, 2, 2, 3, 3, 3, 4]
[1, 2, 2, 3, 4, 4, 5, 6, 6]
[1, 1, 2, 2, 2, 3, 3, 3, 4]
[1, 2, 2, 3, 4, 4, 5, 6, 6]
La cadena más grande posible para ser subcadena de las otras 2 es:  110101
```