# **Algoritmia y Complejidad**
Entrega ejercicios 5

Grado en Ingeniería Informática

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

# Índice

# Problema 4

Enunciado:

Se dispone de un tablero M de tamaño FxC (F es la cantidad de filas y C la cantidad de columnas) y se pone en una casilla inicial (posx, posy) un caballo de ajedrez. El objetivo es encontrar, si es posible, la forma en la que el caballo debe moverse para recorrer todo el tablero de manera que cada casilla se utilice una única vez en el recorrido (el tablero 8x8 siempre tiene solución independientemente de dónde comience el caballo). El caballo puede terminar en cualquier posición del tablero.
Un caballo tiene ocho posibles movimientos (suponiendo, claro está, que no se sale del tablero). Un movimiento entre las casillas $M_{ij}$ y $M_{pq}$ es válido solamente si:

- (|p–i|=1)&&(|q–j|=2), o bien si
- (|p–i|=2)&&(|q–j|=1),

es decir, una coordenada cambia dos unidades y la otra una única unidad.

Código:

```
from random import randint

'''tablero de 8x8'''
maxTablero = 8


def compro(tablero, posx, posy, movimientos, recorridos):
    if movimientos >= maxTablero * maxTablero:
        print("Solución encontrada para los parámetros:")
        print("\tTamaño del tablero (Filas x Columnas): ", maxTablero, "x",
maxTablero)
        for a in recorrido:
            print(a)
        return True

    for fila in range(maxTablero):
        for columna in range(maxTablero):
            '''Compruebo si el movimiento es posible y que no hayamos pasado ya
por el'''
            if (tablero[fila][columna] == False and (
                    ((abs(fila - posx) == 1) and (abs(columna - posy) == 2)) or (
                    (abs(fila - posx) == 2) and (abs(columna - posy) == 1)))):
                '''Me muevo a esa posicion'''
                tablero[fila][columna] = True

                cadena = str("".join(("[",
                                      str(fila),
                                      ",",
```

```
                            str(columna),
                            "]")))

                recorridos.append(cadena)
                '''Llamo a la funcion'''
                resul = compro(tablero, fila, columna, movimientos + 1, recorridos)
                recorridos.remove(cadena)
                tablero[fila][columna] = False

                if (resul):
                    recorridos.append(cadena)
                    return True


'''Relleno el tablero (False si no he pasado por el, True en caso contrario)'''

tablero = [[False for col in range(maxTablero)] for fil in range(maxTablero)]

'''Posicion inicial del caballo'''
posx = randint(0, maxTablero - 1)
posy = randint(0, maxTablero - 1)
'''Marco visitada la posicion inicial'''
print("Posicion Inicial: x=", posx, " y:=", posy)
tablero[posy][posy] = True
recorrido = ['']
'''Evaluo si es posible completar el tablero'''
if (compro(tablero, posx, posy, 1, recorrido) is None):
    print("No es posible recorrer todo el tablero")
else:
    print("posible")
```

**Explicación:**

Creamos una posición aleatorio donde empezar y llamamos a la función principal.

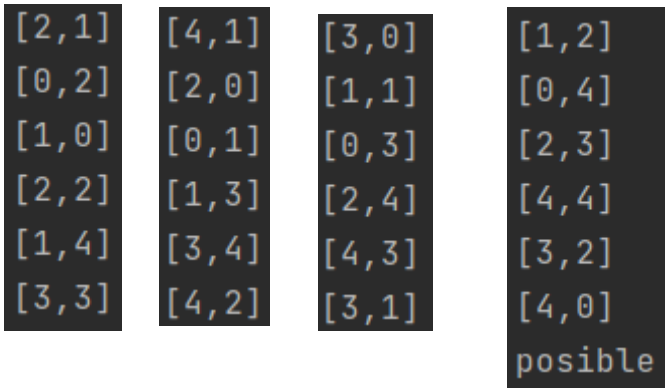
En ella primero miramos si ya he pasado por todas las casillas y entonces mostraría el recorrido que he seguido. En caso contrario creo 2 bucles, uno para las filas y otro para las columnas en las que voy mirando si desde mi posición puedo llegar a la posición que marcan los bucles, en dicho caso me muevo a la nueva posición y llamo de forma recursiva a la función.

<u>Ejemplo de ejecución:</u>

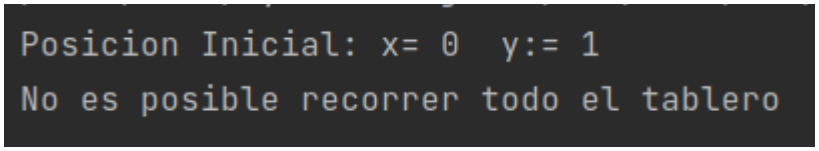
Para este ejemplo hemos creado un tablero de 5x5 para reducir el tiempo de ejecución:

```
Posicion Inicial: x= 4  y:= 0
Solución encontrada para los parámetros:
    Tamaño del tablero (Filas x Columnas):  5 x 5
```

Tras ejecutar y confirmar que es posible recorrer todo el tablero, nos indica los pasos que ha seguido.




En caso de nos ser posible, nos mostraría un mensaje informativo.



# Problema 6

Enunciado:

Se tiene la tabla de sustitución que aparece a continuación

| | a | b | c | d |
|---|---|---|---|---|
| **a** | b | b | a | d |
| **b** | c | a | d | a |
| **c** | b | a | c | c |
| **d** | d | c | d | b |

que se usa de la manera siguiente: en una cadena cualquiera, dos caracteres consecutivos se pueden sustituir por el valor que aparece en la tabla, utilizando el primer carácter como fila y el segundo carácter como columna. Por ejemplo, se puede cambiar la secuencia ca por una b, ya que M[c,a]=b.

Implementar un algoritmo Backtracking que, a partir de una cadena no vacía texto y utilizando la información almacenada en una tabla de sustitución M, sea capaz de encontrar la forma de realizar las sustituciones que permite reducir la cadena texto a un carácter final, si es posible.

Ejemplo: Con la cadena texto=acabada y el carácter final=d, una posible forma de sustitución es la siguiente (las secuencias que se sustituyen se marcan para mayor claridad): aca*ba*da → a*ca*cda → abc*da* → *ab*cd → b*cd* → *bc* → d.

Código:

```
'''Dado un caracter, miro si es una de los 4 que pertenecen a los indices de la tabla'''
def caracterValido(caracter):
    if (caracter == "a" or caracter == "b" or caracter == "c" or caracter == "d"):
        return True
    else:
        return False

'''Dado un caracter, digo a que fila o columna de indice pertenece dicho caracter'''
def indice(caracter):
    resul = -1
    '''Fila o columna 0'''
    if (caracter == "a"):
        resul = 0
```

```
    '''Fila o columna 1'''
    if (caracter == "b"):
        resul = 1
    '''Fila o columna 2'''
    if (caracter == "c"):
        resul = 2
    '''Fila o columna 3'''
    if (caracter == "d"):
        resul = 3
    return resul

'''Metodo principal del programa'''
def cambiarCadena(cadena):
    '''En caso de haber lledado a la cadena con solo un caracter, la muestro y
termino'''
    if (len(cadena) == 1):
        print("cadena con longitud 1 es: ", cadena)
        exit(1)
    '''En caso de tener la cadena con longitud mayor a 1'''
    for a in range(len(cadena) - 1):
        '''Me guardo los 2 caracteres que voy a tratar'''
        caracter1 = cadena[a]
        caracter2 = cadena[a + 1]
        '''Compruebo que sean caracteres validos para mi tabla de
correspondencias'''
        if (caracterValido(caracter1) and caracterValido(caracter2)):
            '''Cambio estos 2 caracteres por su equivalente'''
            texto = cadena.replace(caracter1 + caracter2,
tablero[indice(caracter1)][indice(caracter2)])
            print(cadena," => " , caracter1 + caracter2," =
",tablero[indice(caracter1)][indice(caracter2)] ," => ", texto)
            '''Llamada recursiva con el cambio realizado'''
            cambiarCadena(texto)
        else:
            '''Si he encontrado un caracter que no puedo convertir, paro'''
            print("No puedo reducirlo a un caracter")
            exit(1)
    return

'''Creo el tablero con las conversiones'''
tablero = ["b", "b", "a", "d"], ["c", "a", "d", "a"], ["b", "a", "c", "c"], ["d", "c", "d",
"b"]
'''Defino mi cadena que quiero convertir'''
texto = "acabada"  # final esperado D
print("tablero")
for a in tablero:
    print(a)
print("\ncadena: ",texto)
print("Comienzo\n----------")
'''Comienzo con la conversión'''
```

```
cambiarCadena(texto)
```

**Explicación:**

Definimos la tabla de conversiones y la cadena que queremos reducir.

Llamamos al método principal del programa que primero comprueba si la longitud de la cadena es 1 que seria en el momento que hemos llegado a nuestro objetivo y pararíamos.

En caso contrario, hago un bucle para recorrer toda la cadena seleccionando dos caracteres consecutivos y mirando si están permitidos para mi tabla. En caso afirmativo realizo el cambio y llamo a la función con la nueva cadena.
En caso de encontrar uno no permitido, informo y termino la ejecución.

Ejemplo de ejecución:

Dada la cadena “acabada” observamos como va realizando los cambios progresivamente.

```
tablero
['b', 'b', 'a', 'd']
['c', 'a', 'd', 'a']
['b', 'a', 'c', 'c']
['d', 'c', 'd', 'b']

cadena:  acabada
Comienzo
------------
acabada  =>  ac  =  a  =>  aabada
aabada  =>  aa  =  b  =>  bbada
bbada  =>  bb  =  a  =>  aada
aada  =>  aa  =  b  =>  bda
bda  =>  bd  =  a  =>  aa
aa  =>  aa  =  b  =>  b
cadena con longitud 1 es:  b
```

En caso de introducir “acabadas” nos informaría de que es imposible.

```
aadas  =>  aa  =  b  =>  bdas
bdas  =>  bd  =  a  =>  aas
aas  =>  aa  =  b  =>  bs
No puedo reducirlo a un caracter
```