Universidad de Alcalá

Escuela Politécnica Superior

Universidad de Alcalá

Algoritmia y Complejidad

Laboratorio – Sesión 5

Backtracking

Laboratorio Jueves 12:00 – 14:00

Grado en Ingeniería Informática – Curso 2018/2019

Eduardo Graván Serrano – 03212337L

Marcos Barranquero Fernández – 51129104N

Ejercicio 4: Ajedrez con el caballo

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 Mij y Mpq 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

```
def caballo(tablero, posx, posy, num_movimientos):
    # Si el numero de movimientos es mayor o igual que el numero
total de casillas, es que hemos visitado todas las casillas,
    # por lo que hemos encontrado una solución.
    if num_movimientos >= filas * columnas:
        print("Solución encontrada para los parámetros:")
        print("\tTamaño del tablero (Filas x Columnas):
",filas,"x",columnas)
        print("\tPosición inicial: ", pos_inicial)
        return True

    for fila in range(filas):
        for columna in range(columnas):
            # Si el movimiento es válido y la casilla no ha sido
visitada aún
            if movimiento_valido(posx, posy, fila, columna) and
tablero[fila][columna] == 0:

                # Marco la casilla como visitada
                tablero[fila][columna] = 1

                # Llamo recursivamente a las posibles soluciones
derivadas de este paso
                exito = caballo(tablero, fila, columna,
num_movimientos+1)

                # Marco la casilla como no visitada
                tablero[fila][columna] = 0

                if exito:
                    return True

def movimiento_valido(n_fila_ini, n_columna_ini, n_fila_fin,
n_columna_fin):
```

```
    """ Devuelve true si el movimiento es correcto. Si no, False.
"""
    return (abs(n_fila_fin - n_fila_ini) == 1 and abs(n_columna_fin
- n_columna_ini) == 2) or (abs(n_fila_fin - n_fila_ini) == 2 and
abs(n_columna_fin - n_columna_ini) == 1)

# Programa principal
filas = 6
columnas = 6
pos_inicial = [0, 0] # fila, columna
tablero = [[0 for columna in range(columnas)] for fila in
range(filas)]
# Se marca la posicion inicial como visitada
tablero[pos_inicial[0]][pos_inicial[1]] = 1
# Si la funcion ha terminado y no ha retornado nada, es que no hay
solución
if caballo(tablero, pos_inicial[0], pos_inicial[1], 1) is None:
    print("No hay ninguna solución para los parámetros
introducidos.")
```

Explicación

El programa empieza definiendo el tamaño del tablero en filas y columnas, y escogiendo una posición inicial.

Se construye una matriz de tamaño Filas x Columnas que representa el tablero que tiene que recorrer el caballo. Este tablero va a almacenar las posiciones que ya se han visitado, si en una determinada posición hay un 0, significa que aún no se ha visitado esa posición, mientras que, si tiene un 1, significa que esa posición ya ha sido visitada por el caballo. Esto quiere decir que, si se hiciese un print de la matriz al finalizar la ejecución del programa, si se ha encontrado una solución, se debería devolver una matriz de tamaño F x C y cuyo único valor fuese el 1.

Una vez construida dicha matriz, se marca la posición inicial como visitada y se llama a la función codificada, pasando como parámetro la matriz que representa el tablero, las coordenadas x e y iniciales, y el numero de movimientos que hemos hecho, empezando en 1 para contar la posición inicial.

La función caballo es la función principal y la que será llamada de forma recursiva. La función empieza gestionando el caso en el que hayamos llegado a la recursión más profunda posible, que será el caso en el que hayamos encontrado una solución. Este es el caso en el que el número de movimientos que hayamos hecho sea igual (o mayor) que el número total de casillas. Si esto sucede con las condiciones que pone el enunciado y que se implementan más adelante, quiere decir que el caballo ya ha visitado todas las casillas, y por lo tanto se ha encontrado una solución, por lo que sacaremos por pantalla que se ha encontrado una solución para esos parámetros en concreto y se devolverá True para volver en la recursión.

En el caso de que no hayamos encontrado una solución aún, la función entra en un doble bucle anidado que examina toda la matriz. El primer bucle analiza por filas, y el bucle anidado analiza por columnas, de forma que se examina toda la matriz en el caso de que no haya una solución.

Dentro de este segundo bucle se hacen dos comprobaciones, si ambas devuelven true se seguirá examinando ese movimiento en concreto que viene dado por la posición de las filas y las columnas (la i y la j en los bucles) y la posición en la que nos encontramos en ese determinado momento (valores que entran como parámetro en la función).

La primera de estas comprobaciones se ha implementado como una función llamada "movimiento_valido", la cual devuelve un boolean. Entran como parámetro a la función los valores de filas y columnas iniciales (antes de efectuar el movimiento), y los valores de fila y columna a los que el caballo esta evaluando si es posible moverse. Devolverá True si el movimiento es un movimiento válido para el caballo, es decir, si se mueve un total de 2 casillas en horizontal y 1 en vertical o viceversa. Si es cualquier otro valor, la función devolverá False, ya que no sería un movimiento válido para un caballo de ajedrez.

La segunda comprobación es si esa posición que se está evaluando aún no ha sido visitada previamente. Si lo ha sido, es decir, si esa posición tiene un 1 en la matriz tablero, ese movimiento tampoco será valido ya que es una de las especificaciones del enunciado. Si, por el contrario, esa posición no se ha visitado aún, es decir, es 0 en la matriz tablero, se podrá visitar.

Si ambas condiciones son True, se marcará la casilla a evaluar como que ha sido visitada y se hará una llamada a la misma función caballo de forma que se inicia la recursión. Se pasará como parámetro el tablero actualizado con el nuevo movimiento, las fila y columna actuales como posiciones iniciales para el siguiente nivel de recursión, y el número de movimientos incrementado en 1. La llamada a esta función se hace esperando un boolean de vuelta. Si en los niveles inferiores de recursión no se encuentra una solución, esta variable tomará valor None. Si por el contrario se encuentra una solución, el valor devuelto será True, y este será el valor almacenado en esta variable.

La siguiente línea se encarga de desmarcar la posición como visitada en el tablero por si acaso no se ha encontrado una solución válida siguiendo por este camino.

Por último, se evalúa la variable que nos ha devuelto la llamada recursiva, si es True, se devolverá True al nivel de recursión anterior, propagándose hasta el nivel más alto. De esta forma conseguimos salir de la recursión, ya que solo nos interesa saber si hay una solución válida para los parámetros introducidos, no encontrar todas las posibles soluciones.

La llamada inicial a la función caballo se hace desde una sentencia if, para que en el caso de que no haya una solución y nunca se devuelva nada, el programa notifique por pantalla que no hay ninguna solución posible para esos parámetros introducidos.

Ejercicio 6: Sustitución de caracteres

Enunciado

Se tiene la tabla de sustitución que aparece a continuación que se usa de la manera siguiente:

	a	b	c	d
a	b	b	a	d
b	c	a	d	a
c	b	a	c	c
d	d	c	d	b

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): acabada ◊ acacda ◊ abcda ◊ abcd ◊ bcd ◊ bc ◊ d.

Código

```
# Contador del numero de soluciones/formas distintas de reducir la cadena hasta un único carácter
contador = 0

def modificarCadena(cadena, long_cadena):
    # Si la longitud de la cadena es 1, hemos encontrado una solución, la sacamos por pantalla junto con su número de contador
    if long_cadena == 1:
        global contador
        print("Cadena resultante","(", contador,"): ", cadena)
        contador+=1

    # Bucle que examina todos los elementos de la cadena
    for i in range(long_cadena-1):
        # Se busca en el diccionario definido a ver que indice dentro de la tabla corresponde a cada letra de la cadena en sus respectivas posiciones.
        indice1 = indice_tabla.get(cadena[i])
        indice2 = indice_tabla.get(cadena[i+1])

        # Si las letras se encuentran en el diccionario, se puede intercambiar y se concatenan las letras a reemplazar con
        # el resto de la cadena a partir de las 2 letras sustituidas.
        if (indice1 is not None) and (indice2 is not None):
            nueva_cadena = "".join([cadena[:i], tabla[indice1][indice2], cadena[i+2:]])
            modificarCadena(nueva_cadena, len(nueva_cadena))
```

```
        # Si alguna de las letras no se encuenta en el diccionario, no se puede reemplazar y por lo tanto no se podrá encontrar una solución de 1 solo carácter.
        else:
            print("No se podrá reducir a 1 solo carácter.")
            break


# Se inicializa la tabla, se ha cogido la tabla que viene en el enunciado.
def inicializarTabla():
    tabla[0][0] = "b"
    tabla[0][1] = "b"
    tabla[0][2] = "a"
    tabla[0][3] = "d"

    tabla[1][0] = "c"
    tabla[1][1] = "a"
    tabla[1][2] = "d"
    tabla[1][3] = "a"

    tabla[2][0] = "b"
    tabla[2][1] = "a"
    tabla[2][2] = "c"
    tabla[2][3] = "c"

    tabla[3][0] = "d"
    tabla[3][1] = "c"
    tabla[3][2] = "d"
    tabla[3][3] = "b"

# Tabla de referencias de a que posicion corresponde cada letra en la tabla generada
indice_tabla = {
    "a":0,
    "b":1,
    "c":2,
    "d":3
}

# Programa principal
tabla = [["" for _ in range (4)] for _ in range(4)]
inicializarTabla()

texto = "acabada"
n = len(texto)
modificarCadena(texto, n)
```

Explicación

Para este ejercicio se ha hecho uso de una variable global para hacer un seguimiento del número de soluciones diferentes del problema planteado.

Se ha utilizado el ejemplo que venía definido en el enunciado por motivos de simplicidad, para que funcionase con otros caracteres diferentes sólo habría que modificar el diccionario y la tabla.

La tabla mencionada en el párrafo anterior es una matriz, en la cual tenemos una correspondencia entre combinaciones de letras. La [i] hace referencia a la primera letra que tenemos para reemplazar, mientras que la [j] hace referencia a la segunda letra que queremos reemplazar.

A parte de esta tabla se ha utilizado un diccionario, para hacer la transformación entre el carácter que se esta leyendo de la cadena y su índice correspondiente en la tabla.

Una vez hemos definido estas estructuras, empezamos con la explicación del código en sí. Para empezar, se va a crear la tabla con strings vacios simplemente para inicializarla y se va a llamar a una función auxiliar “inicializarTabla()”, la cual no toma parámetros. Esta función se encarga de recrear la tabla que viene como ejemplo en el enunciado del problema. Va a asignar a cada combinación de posiciones [i][j] (primera y segunda letra) una letra para poder reemplazar por las dos letras iniciales.

Se define la cadena que queremos estudiar y se calcula su longitud inicial. Una vez hecho esto se hace la llamada a la función principal “modificarCadena”, esta será la función en la cual se hagan las llamadas recursivas.

La función “modificarCadena” toma como parámetros la cadena de texto y la longitud de la misma. Las primeras líneas de la función se encargan de gestionar el nivel de recursión mas profundo posible, que es aquel en el que la longitud de la cadena sea 1, es decir, ya se haya reducido al máximo la cadena. Si se alcanza esta condición, se saca por pantalla el carácter resultante de la reducción de la cadena junto con el número de solución correspondiente y se incrementa en 1 el contador de soluciones.

En caso de que no hayamos llegado a este punto aún, entraremos en un bucle que llegue hasta la penúltima posición de la cadena. Dentro de este bucle vamos a analizar las posiciones [i] e [i+1] de la cadena. Se va a definir una variable que actúa como índice para estos dos caracteres, de forma que se mira en el diccionario definido a que valor numérico corresponde cada uno de estos dos caracteres que estamos evaluando en ese momento.

Si los caracteres que estamos evaluando no están en el diccionario, estos índices tomarán valor None, y por lo tanto no se podrá encontrar una solución, ya que no se han definido reglas de sustitución para esos caracteres, por lo que no se intentará seguir reduciendo la cadena.

En caso contrario, si los índices son válidos, se va a crear una nueva cadena. Esta nueva cadena se forma uniendo el principio de la cadena hasta el primer carácter que estamos evaluando sin contarlo, el carácter que corresponde sacado de la tabla a partir de los dos índices creados anteriormente, y el resto de la cadena que aún no se ha evaluado. Una vez se tiene la nueva cadena resultante, se llama a la misma función de forma recursiva pasando como parámetro la nueva cadena y su longitud.