TÉCNICAS DE PROGRAMACIÓN AVANZADA

Práctica 1

Universidad Europea Madrid
Álvaro Farreny Boixader

Contenido

Ejercicio 1

Manteniendo los dos bucles **for**, realizar las modificaciones **mínimas** necesarias en el código de **es Neutralizada_v1** para que simplemente funcione de manera correcta.

```
private static int[][] resta (int[][] m1, int[][] m2){

    int[][] resultado = new int[m1.length][m1.length];

    for (int fila=0; fila < resultado.length; fila++)
        for (int col=0; col < resultado[fila].length; col++)
            resultado[fila][col] = m1[fila][col] - m2[fila][col];

    return resultado;
}
```

```
// CORREGIR esta funciÃ³n
public static boolean esNeutralizada_v1 (int[][] m1, int[][] m2) {
    boolean neutraliza = true;
    int[][] matrizResta = resta(m1, m2);

    for (int fila = 0; fila < m1.length; fila++) {        // No editar esta lÃ-nea
        for (int col = 0; col < m1[fila].length; col++) { // No editar esta lÃ-nea
            if(neutraliza == true)
                if (fila==col)
                    if (matrizResta[fila][col] == 1)
                        neutraliza = true;
                    else
                        neutraliza = false;
                else
                    if (matrizResta[fila][col] == 0)
                        neutraliza = true;
                    else
                        neutraliza = false;
        }//for
    }//for
    return neutraliza;
}
```

Tras los cambios anteriores, esNeutralizada_v1 será correcta, pero no será una versión eficiente. Se pide entonces implementar en **esNeutralizada_v2** una versión más eficiente en tiempo para resolver el problema.

```
public static boolean esNeutralizada_v2 (int[][] m1, int[][] m2) {
    int[][] matrizResta = resta(m1, m2);

    for (int fila = 0; fila < m1.length; fila++) {        // No editar esta lÃ-nea
        for (int col = 0; col < m1[fila].length; col++) { // No editar esta lÃ-nea
                if (fila == col) {
                    if (matrizResta[fila][col] != 1)
                        return false;
                } else {
                    if (matrizResta[fila][col] != 0)
                        return false;
                }
        }//for
    }//for

    return true; // sentencia insertara para poder compilar
}
```

Implementar en esNeutralizada_DyV una versión que siga la estrategia “Divide y Vencerás” para resolver el problema.

```
public static boolean esNeutralizada_DyV (int[][] m1, int[][] m2) {
        if (esIdentidad_DyV(m1) != esIdentidad_DyV(m2)) {
            return false;
        } else {
            return true;
        }
    }

public static boolean esIdentidad_DyV (int[][] matriz) {
    return esIdentidad_DyV2(matriz, 0, matriz.length-1);
}

private static boolean esIdentidad_DyV2 (int[][] matriz, int inicio, int fin) {
    if(inicio == fin) {
        return comprobar(matriz, inicio);
    } else {
        int mitad = (inicio + fin)/2;
        boolean x = esIdentidad_DyV2(matriz, inicio, mitad);
        boolean y = esIdentidad_DyV2(matriz, mitad+1, fin);
        if(x == false) {
            return false;
        }else if (y == false) {
            return false;
        } else {
            return true;
        }       }
}
private static boolean comprobar (int[][] matriz, int inicio){
    if(matriz[inicio][inicio] == 1) {
        for(int i=0; i<matriz.length; i++) {
            if(i!= inicio) {
                if(matriz[inicio][i] != 0) {
                    return false;
                }
            }
        }
    }else {
        return false;
    }
    return true;
}
```

No es exactamente como se debería hacer, además tiene algún error y no sale el resultado correcto pero lo que he hecho ha sido comprar si son identidad y si las dos son debería salir true y si no lo son false, pero me salen todo el rato false así que no lo se.

Calcular la complejidad de las tres funciones. Aplicar los correspondientes sumatorios y reducciones para argumentar y demostrar el resultado alcanzado.

Primero calculamos la complejidad de la función resta ya que la vamos a utilizar durante todo nuestro programa:

Resta

Función	***Resultado del calculo***
`int[][] resultado = new int[m1.length][m1.length];`	Al ser un valor que estamos calculando y ser una resta, cogemos un 1.
`for (int col=0; col < resultado[fila].length; col++)` `  resultado[fila][col] = m1[fila][col] - m2[fila][col];`	Esto es igual a: $$1+\left(\sum_{k=0}^{n-1} 1+1+1\right)+1$$ Primero, cambiamos el sumatorio y le sumamos 1 arriba y 1 abajo $$2+\left(\sum_{k=1}^{n} 3\right)$$ El sumatorio de un valor constante es igual al valor que tiene encima ese sumatorio, por tanto: $$2+(3\mathrm{n})$$

`for (int fila=0; fila < resultado.length; fila++)`
`  for (int col=0; col < resultado[fila].length; col++)`
`  resultado[fila][col] = m1[fila][col] - m2[fila][col];`
	$$1+\left(\sum_{k=0}^{n-1} 1+(2+3\mathrm{n})+1\right)+1$$
	Solucionamos primero el sumatorio y simplificamos los valores de su interior
	$$2+\left(\sum_{k=0}^{n-1} 4+3\mathrm{n}\right)$$
	Resolvemos el sumatorio sumando 1 arriba y 1 abajo
	$$2+\left(\sum_{k=1}^{n} 4+3\mathrm{n}\right)$$
	La suma de este sumatorio es lo mismo que:
	$$2+\left(\sum_{k=1}^{n} 4\right)+\left(\sum_{k=1}^{n} 3\mathrm{n}\right)$$
	$$2+4\mathrm{n}+3n^2$$

Una vez tenemos ya calculado el tiempo del for, calculamos el tiempo total de la función.

int[][] resultado = new int[m1.length][m1.length];	for (int fila=0; fila < resultado.length; fila++)	return resultado;
1	$2+4\mathrm{n}+3n^2$	1

Por tanto, sumamos los valores y obtenemos el resultado del tiempo de ejecución y el orden

$$\mathbf{T(resta) = } 3n^2 + 4n + 4$$

$$O(n^2)$$

Una vez calculada la función resta, pasamos a calcular el apartado a de nuestro problema, la función v1.

esNeutralizada_v1

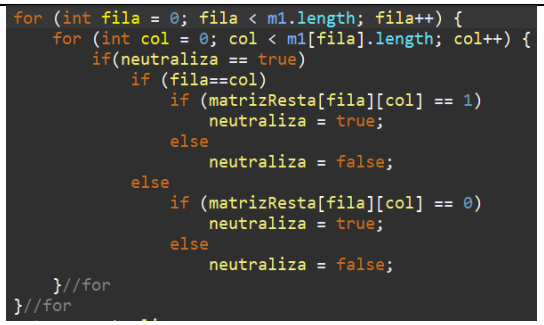
E	`if (matrizResta[fila][col] == 0)`
	`  neutraliza = true;`
	`else`
	`  neutraliza = false;`
		Los ifs se cogen siempre el valor máximo de tiempo de ejecución, es decir, el peor de los casos.
		1 + Max (1) = 1+1 = **2**

F	`if (matrizResta[fila][col] == 1)`
	`  neutraliza = true;`
	`else`
	`  neutraliza = false;`
		Los ifs se cogen siempre el valor máximo de tiempo de ejecución, es decir, el peor de los casos.
		1 + Max (1) = 1+1 = **2**

D	`if (fila==col)`
	`  if (matrizResta[fila][col] == 1)`
	`  neutraliza = true;`
	`  else`
	`  neutraliza = false;`
	`else`
	`  if (matrizResta[fila][col] == 0)`
	`  neutraliza = true;`
	`  else`
	`  neutraliza = false;`
		Como podemos ver, ahora tenemos de nuevo un if con los dos if calculados anteriormente.
		Por tanto:
		1+Max(t(e)+t(f)) = 1+Max (2,2) = 1+2= **3**

C	`if(neutraliza == true)`
	`  if (fila==col)`
	`  if (matrizResta[fila][col] == 1)`
	`  neutraliza = true;`
	`  else`
	`  neutraliza = false;`
	`  else`
	`  if (matrizResta[fila][col] == 0)`
	`  neutraliza = true;`
	`  else`
	`  neutraliza = false;`
		De nuevo, tenemos un if por tanto:
		1+Max(D) = 1 + Max (3) = 1+4 = **5**

B	`for (int col = 0; col < m1[fila].length; col++) {`
	`  if(neutraliza == true)`
	`  if (fila==col)`
	`  if (matrizResta[fila][col] == 1)`
	`  neutraliza = true;`
	`  else`
	`  neutraliza = false;`
	`  else`
	`  if (matrizResta[fila][col] == 0)`
	`  neutraliza = true;`
	`  else`
	`  neutraliza = false;`
	`}//for`
		Calculamos el for con todo el cuerpo que tiene dentro que es nuestra C.
		Por tanto, obtenemos:
		$1 + \left(\sum_{k=0}^{n-1} 1 + (5) + 1\right) + 1$
		Resolvemos el sumatorio y simplificamos su interior
		$2 + \left(\sum_{k=1}^{n} 7\right)$
		2+7n

A		Calculamos por último el ultimo for para resolver la parte central de nuestra función. $1 + \left(\sum_{k=0}^{n-1} 1 + (2 + 7\mathrm{n}) + 1\right) + 1$ $1 + \left(\sum_{k=1}^{n} 4 + 7\mathrm{n}\right) + 1$ La suma de este sumatorio es lo mismo que: $2 + \left(\sum_{k=1}^{n} 4\right) + \left(\sum_{k=1}^{n} 7\mathrm{n}\right)$ $\mathbf{2 + 4n + 7n^2}$

Una vez tenemos ya calculado el tiempo del for, calculamos el tiempo total de la función.

boolean neutraliza = true;	int[][] matrizResta = resta(m1, m2);	for (int fila = 0; fila < m1.length; fila++)	return neutraliza;
1	$2 + 4\mathrm{n} + 3n^2$	$\mathbf{2 + 4n + 7n^2}$	1

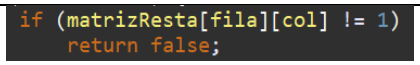
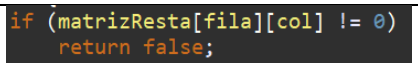
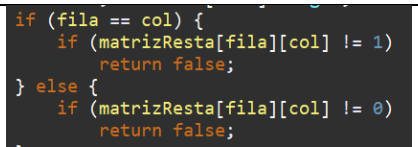
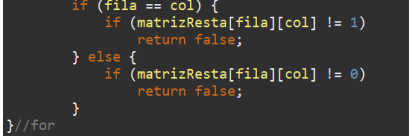
Sumamos todos los valores obtenidos:

$$\mathbf{T(esNeutraliza\_v1) = 10n^2 + 8n + 6}$$

$$\boldsymbol{O(n^2)}$$

Calculamos ahora el tiempo de ejecución y el orden de la siguiente función

esNeutralizada_v2

E		Los ifs se cogen siempre el valor máximo de tiempo de ejecución, es decir, el peor de los casos. 1 + Max (1) = 1+1 = **2**
D		Los ifs se cogen siempre el valor máximo de tiempo de ejecución, es decir, el peor de los casos. 1 + Max (1) = 1+1 = **2**
C		Los ifs se cogen siempre el valor máximo de tiempo de ejecución, es decir, el peor de los casos. 1 + Max (E, D) = 1+MAX (2, 2) = **3**
B		Calculamos el for con todo el cuerpo que tiene dentro que es nuestra C. Por tanto, obtenemos: $1 + \left(\sum_{k=0}^{n-1} 1 + (3) + 1\right) + 1$ Resolvemos el sumatorio y simplificamos su interior

		$2+\left(\sum_{k=1}^{n} 5\right)$ **2+5n**
A		Calculamos por último el ultimo for para resolver la parte central de nuestra función. $1+\left(\sum_{k=0}^{n-1} 1+(2+5\mathrm{n})+1\right)+1$ $1+\left(\sum_{k=1}^{n} 4+5\mathrm{n}\right)+1$ La suma de este sumatorio es lo mismo que: $2+\left(\sum_{k=1}^{n} 4\right)+\left(\sum_{k=1}^{n} 5\mathrm{n}\right)$ $\mathbf{2+4n+5n^2}$

Una vez tenemos ya calculado el tiempo del for, calculamos el tiempo total de la función.

int[][] matrizResta = resta(m1, m2);	for (int fila = 0; fila < m1.length; fila++)	Return true
$2+4\mathrm{n}+3n^2$	$2+4\mathrm{n}+5n^2$	1

T(esNeutraliza_v2) = $8n^2+8n+5$

$$O(n^2)$$

Comparar las complejidades obtenidas, y argumentar cuál de las tres versiones sería la mejor.

La que seria mejor en todos los casos es la de Divide y vencerás ya que gracias a ese algoritmo podemos dividir el problema en subproblemas de menor complejidad y solucionarlo todo a la vez en un menor tiempo gracias a la recursividad. Al no tener bien el ejercicio 3, no he calculado la complejidad ya que no me saldría correctamente.

Ejercicio 2

Primera Parte

Implementar una función que utilizando la estrategia de “Divide y Vencerás”, determine si dos arrays de números enteros son uno el inverso del otro.

```java
public static void main(String[] args) {
    int a1 [] = {1,2,3,4,5,6,7,8,9};
    int a2 [] = {9,8,7,6,5,4,3,2,1};

    int inicioarr1 = 0;
    int finarr1 = (a1.length-1);

    int inicioarr2 = 0;
    int finarr2 = (a2.length-1);

    System.out.println(calcular(a1, a2, inicioarr1, inicioarr2, finarr1, finarr2));

}
```

```java
public static boolean calcular(int arr1[], int arr2[], int i1, int i2, int f1, int f2) {
    if (arr1.length != arr2.length) { //comprobamos longitudes
        return false;
    } else if (i1==f1 && i2==f2){ // caso cuando el inicio del primero sea igual al final es decir tamaño 1:1
        return arr1[i1] == arr2[f2]; //hacemos la comparacion y si es true sera correcta y si no false
    } else {
        if (arr1[i1] == arr2[f2]) { //recursividad
            return calcular(arr1, arr2, i1+1, i2, f1, f2-1);
        } else {
            return false;
        }
    }
}
```

Segunda Parte

Calcular razonadamente su complejidad, detallando las reducciones o sumatorios necesarios para su desarrollo, y comparar el resultado con el que obtendríamos si aplicásemos una estrategia iterativa tradicional.

<table>
<tr><th colspan="3">CASO BASE</th></tr>
<tr><td>A</td><td rowspan="2">

```java
if (arr1.length != arr2.length) {
    return false;
} else if (i1==f1 && i2==f2){ //
    return arr1[i1] == arr2[f2];
```

</td><td>T(A) = T(cond)+T(cuerpo), por tanto, T(A) = 1 + 1 = 2</td></tr>
<tr><td>B</td><td>T(B) = T(cond)+T(cuerpo), por tanto, T(B) = 1 + 1 = 2</td></tr>
<tr><td colspan="3">De estos dos ifs, cogemos el valor máximo por tanto, MAX (A, B) = 2</td></tr>
<tr><th colspan="3">CASO GENERAL</th></tr>
<tr><td>C</td><td>

```java
} else {
    if (arr1[i1] == arr2[f2]) { //recursividad
        return calcular(arr1, arr2, i1+1, i2, f1, f2-1);
    } else {
        return false;
    }
}
```

</td><td>Al ser el caso general, tenemos que obtener el valor de a,b,k.
A = el numero de veces que llamamos a la función recursiva dentro de nuestro cuerpo, A=1
B = el numero por el q sustraes el problema, en nuestro caso, B=1
K= resolver el general como si no hubiese recursividad. Por tanto, obtenemos dos ifs.
T(c1) = 1+1 = 2
T(c2) = 1+1 = 2
MAX (c1, c2) = 2</td></tr>
</table>

		Por tanto, O (1) y como tenemos que obtener el valor de k, O (1 * n^0) --> k=0
Una vez tenemos a, b, k utilizamos la fórmula para obtener la recursividad. Como a = 1, O (n^{k+1}) por lo que **O(n)**		

Ejercicio 3

$$H(K) = H_1(K) + (c \cdot H_2(K))$$

$$\mathbf{H_1(K) = K \bmod N}$$

$$\mathbf{H_2(K) = 3 - (K \bmod 3)}$$

0	
1	27
2	30
3	
4	2
5	12
6	16

N=7

$\mathrm{H}(16) = 2 + (0 \cdot 2) = 2$

$\mathrm{H}(16) = 2 + (1 \cdot 2) = 4$

$\mathrm{H}(16) = 2 + (2 \cdot 2) = 6$

$\mathrm{H}(12) = 5 + (0 \cdot 3) = 5$

$\mathrm{H}(27) = 6 + (0 \cdot 3) = 6$

$\mathrm{H}(27) = 6 + (1 \cdot 3) = 9$

$\mathrm{H}(27) = 6 + (2 \cdot 3) = 12$

$\mathrm{H}(27) = 6 + (3 \cdot 3) = 15$

Si introducimos el siguiente valor, va a superar el factor de carga del 80%. Por tanto, redimensionamos la tabla para poder introducir más valores.

$N \cdot 2 = 7 \cdot 2 = 14$

Siguiente número primo es el 17, por lo que N=17.

Calculamos los valores y lo insertamos en la tabla:

$H(30) = 13 + (0 \cdot 3) = 13$

$H(2) = 2 + (0 \cdot 1) = 2$

$H(16) = 16 + (0 \cdot 2) = 16$

$H(12) = 12 + (0 \cdot 3) = 12$

$H(27) = 10 + (0 \cdot 3) = 10$

$H(29) = 12 + (0 \cdot 1) = 12$

$H(29) = 12 + (1 \cdot 1) = 13$

$H(29) = 12 + (2 \cdot 1) = 14$

0	
1	
2	2
3	
4	
5	
6	
7	
8	
9	
10	27
11	
12	12
13	30
14	29
15	
16	16