

## PRÁCTICA 11

### Análisis de Algoritmos

Semana del 24 al 28 de noviembre

#### 1. Objetivo

El objetivo principal de esta práctica consiste en aplicar las técnicas de análisis de algoritmos a pequeños trozos de código fuente para estimar los ordenes de función y las cotas de tiempo que van a consumir, además de contar el número de pasos que consume cada bloque. Para ello, se expondrán algunos ejemplos básicos de análisis sobre bucles del tipo `for`, y posteriormente se evaluará la práctica con un pequeño cuestionario que incluirá ejercicios muy parecidos a los realizados en la práctica.

#### 2. Código fuente necesario

Para llevar a cabo los experimentos de la siguiente sección, necesitaremos el siguiente código fuente que podemos editar como **P11.cpp**:

```
1  #include <iostream>
2
3  using namespace std;
4
5  // poner aqui los ejemplos como funciones con el siguiente formato,
6  // donde 'X' es el numero de ejemplo
7  int ejemploX(const int n)
8  { int suma = 0;
9    // codigo fuente de cada bloque
10   return suma;
11 }
12
13 int main()
14 { // pedir el valor de n
15   int n;
16   cout << "Introducir n: ";
17   cin >> n;
18
19   // invocar a cada ejemplo
20   cout << "Ejemplo X: " << ejemploX(n) << endl;
21   return 0;
22 }
```

### 3. Ejemplos de análisis de algoritmos

#### 3.1. Operaciones elementales/básicas

Hemos visto en clase que las operaciones aritméticas, de comparación, lógicas, etc, se consideran operaciones elementales/básicas y, por lo tanto, su orden es  $T(n) = c = \Theta(1)$ .

```
1 int a = 1, b = 2, c = 3;
2 // asignacion
3 a = b; //  $\Theta(1)$ 
4 // operaciones aritmeticas
5 a = b + c; //  $\Theta(1)$ 
```

#### 3.2. Bucle simple for de 1 a $n$

El bucle `for` entre 1 y  $n$  tiene un orden de complejidad  $\Theta(n)$ .

```
1 int suma = 0; //  $\Theta(1)$ 
2 for (int i = 1; i <= n; i++) //  $\Theta(n)$ 
3     suma++; //  $\Theta(1)$ 
```

#### 3.3. Dos bucles anidados for de 1 a $n$ independientes

Los bucles anidados `for` entre 1 y  $n$  siguen la Regla de Simplificación de Bucles (3b) vista en clase. El bloque de código tiene por tanto una complejidad de orden  $\Theta(n^2)$ .

```
1 int suma = 0; //  $\Theta(1)$ 
2 for (int i = 1; i <= n; i++) //  $\Theta(n)$ 
3     for (int j = 1; j <= n; j++) //  $\Theta(n)$ 
4         suma++; //  $\Theta(1)$ 
```

Por ejemplo, si  $n = 5$ , el valor final de *suma* sería igual a 25.

#### 3.4. Dos bucles anidados for dependientes

No todos los bucles anidados `for` permiten obtener la complejidad tan fácilmente como en el ejemplo 2.3.

```
1 int suma = 0; //  $\Theta(1)$ 
2 for (int i = 1; i <= n; i++) //  $\Theta(n)$ 
3     for (int j = 1; j <= i; j++) //  $\Theta(i)$ 
4         suma++; //  $\Theta(1)$ 
```

El bucle `for` de la línea 3 se ejecuta del orden de  $\Theta(i)$  pasos. Es decir, que dependiendo del valor de  $i$ , tenemos que el número de operaciones básicas que se llevan a cabo serán:

$i$	$j$	Incremento de <i>suma</i>
1	1	1
2	1, 2	2
3	1, 2, 3	3
$\vdots$	$\vdots$	$\vdots$
$n$	1, 2, 3, ..., $n$	$n$

Teniendo en cuenta que las operaciones básicas se ejecutan en tiempo  $c$  constante, y sumando todos los pasos, tenemos que:

$$T(n) = c(1 + 2 + 3 + \dots + n) = c \sum_{i=1}^n i = \frac{cn(n+1)}{2} = \frac{(cn^2 + cn)}{2} = \Theta(n^2)$$

No hace falta saber si la serie anterior es aritmética o geométrica para poder calcular la suma. Podemos hacerlo muy fácilmente usando **Wolfram Alpha** de la siguiente forma: `sum i, i=1 to n`

Aún cuando los bloques de código de los ejemplos 3 y 4 tienen el mismo orden  $\Theta(n^2)$ , el número de operaciones básicas  $T(n)$  de este último ejemplo es aproximadamente la mitad que el otro. Por ejemplo, si  $n = 10$ , el valor final de *suma* sería igual a 55, en comparación con los  $n^2 = 100$  pasos del ejemplo 3.

### 3.5. Dos bucles anidados for con incremento de potencias

No todos los bucles anidados tipo **for** tienen una complejidad  $\Theta(n^2)$ . Asumiendo que  $n$  es una potencia de 2, es decir  $n = 2^x$  con  $x$  entero, vamos a ver el siguiente ejemplo:

```
1 int suma = 0; // Θ(1)
2 for (int i = 1; i <= n; i *= 2) // Θ(log n)
3     for (int j = 1; j <= n; j++) // Θ(n)
4         suma++; // Θ(1)
```

Los valores de  $i$  en el bucle **for** de la línea 2 son:

$$i = 1, 2, 4, 8, 16, \dots, 2^{k-1} = 2^0, 2^1, 2^2, 2^3, 2^4, \dots, 2^{k-1} \leq n$$

Estos  $k$  valores (desde 0 a  $k-1$ ) corresponden al número de iteraciones que realiza el bucle en  $i$ . De este modo, el número de pasos  $k$  que se ejecuta el bucle **for** de la línea 3 será igual a:

$$2^{k-1} \leq n \rightarrow k \leq \log_2 n + 1 = \Theta(\log_2 n)$$

Como el siguiente bucle es independiente del valor  $i$ , se puede aplicar la Regla de Simplificación de Bucles (3b) vista en clase y, por lo tanto, el orden de complejidad total es de  $\Theta(n \log n)$ .

### 3.6. Dos bucles anidados for dependientes con incremento de potencias

Para finalizar los ejemplos, vamos a analizar el siguiente bloque de código, asumiendo de nuevo que  $n$  es una potencia de 2, es decir  $n = 2^x$  con  $x$  entero,:

```
1 int suma = 0; // Θ(1)
2 for (int i = 1; i <= n; i *= 2) // Θ(log n)
3     for (int j = 1; j <= i; j++) // Θ(i)
4         suma++; // Θ(1)
```

Como ya hemos visto en el ejemplo anterior, el bucle de la línea 2 es de orden  $\Theta(\log n)$ , pero el bucle de la línea 3 depende del valor de  $i$ . Si contamos el número de pasos, tenemos que:

$i$	$j$	Incremento de $\textit{suma}$
1	1	1
2	1, 2	2
4	1, 2, 3, 4	4
8	1, 2, 3, 4, 5, 6, 7, 8	8
$\vdots$	$\vdots$	$\vdots$
$2^{k-1}$	$1, 2, 3, \dots, 2^{k-1}$	$2^{k-1}$

Del ejemplo anterior, sabemos que  $k$  es como máximo  $\log_2 n + 1$ . Sumando todos los pasos tenemos que:

$$T(n) = c(1 + 2 + 4 + 8 + \dots + 2^{k-1}) = c \sum_{i=0}^{k-1} 2^i = c \sum_{i=0}^{\log_2 n} 2^i = c(2n - 1) = \Theta(n)$$

De nuevo, podemos calcular fácilmente este sumatorio usando **Wolfram Alpha** de la siguiente forma: `sum 2i, i=0 to log2(n)`. Por tanto, la complejidad de este ejemplo es  $\Theta(n)$ .

## 4. Ejercicios

Antes de realizar el cuestionario de evaluación, se recomienda practicar analizando los siguientes dos bloques de código.

### 4.1. Ejercicio 1

```
1 int suma = 0;
2 for (int i = 1; i <= n; i++)
3     for (int j = 1; j <= n; j *= 2)
4         for (int k = 1; k <= n; k++)
5             suma++;
```

### 4.2. Ejercicio 2

```
1 int suma = 0;
2 for (int i = 1; i <= n - 1; i++)
3     for (int j = i + 1; j <= n; j++)
4         suma++;
```

## 5. Evaluación

La evaluación de esta práctica consiste en la realización del cuestionario de la **Práctica 11** del aula virtual de [Computabilidad y Algoritmia](#).