Análisis de algoritmos recursivos

- La estimación de la complejidad de un algoritmo iterativo es en muchas ocasiones directa, gracias al uso de series aritméticas o geométricas.

- En el caso de los algoritmos recursivos, la propia estimación de la complejidad es en general recursiva, dando lugar a una ecuación de recurrencia.

- Estrategia:
  - Decidir el parámetro/s que determinan el tamaño de la entrada.
  - Identificar la operación u operaciones básicas del algoritmo.
  - Comprobar si el número de veces que se ejecuta la operación básica depende únicamente del tamaño de la entrada, o hay otras condiciones adicionales que influyen. En este caso, habría que distinguir entre los casos peor, medio y mejor.
  - Establecer la ecuación de recurrencia, con las condiciones iniciales.
  - Resolver la ecuación de recurrencia, o al menos, establecer el orden de crecimiento de su solución (se verá más adelante).

Análisis de Algoritmos Recursivos Factorial

```
/**
 * @param n número natural
 * @return n! = 1*...*n
 */
public static int factorial(int n){
   if (n<0) throw
     new IllegalArgumentException("número no válido");
   if (n == 0) return 1;
   else return n*factorial(n-1);
}
```

- Tamaño de la entrada: el valor del número natural
- Operación básica: la multiplicación de números
- El cálculo de la complejidad nos lleva de forma natural a una ecuación de recurrencia:

$$T(n) = \begin{cases} 0 & n = 0 \\ 1 + T(n-1) & n > 0 \end{cases}$$

Caso base (condiciones iniciales)

Producto de n y factorial(n-1)

Multiplicaciones para calcular factorial(n-1)

Resolución de recurrencias
Sustitución Hacia Atrás

- Hay un gran arsenal matemático disponible para convertir una recurrencia en una ecuación en forma cerrada. El método simple es la sustitución hacia atrás.

- Se calcula la recurrencia para n-1, n-2,…

$$
\begin{aligned}
T(n) &= 1 + T(n-1) = 1 + (1 + T(n-2)) = 2 + T(n-2) \\
&= 2 + (1 + T(n-3)) = 3 + T(n-3)) = \ldots \\
&= i + T(n-i)
\end{aligned}
$$

- La sustitución sigue hasta que se llega al caso base $T(0) = 0$. En ese caso, $0 = n-i \Rightarrow i = n$.

- $T(n) = n + T(0) = n$;

Recurrencias Lineales Homogéneas (I)

- El método del polinomio característico puede utilizarse cuando la recurrencia tiene la forma

$$T(n) = a_{k-1}T(n-1) + a_{k-2}T(n-2) + \cdots + a_0T(n-k) + f(n)$$

donde cada a_i es una cierta constante. Necesitamos en este caso k condiciones iniciales $T(1), \cdots, T(k)$.

- Si $f(n) = 0$ tenemos una recurrencia lineal homogénea de orden k,

$$T(n) - a_{k-1}T(n-1) - a_{k-2}T(n-2) - \cdots - a_0T(n-k) = 0$$

cuya ecuación característica es

$$r^k - a_{k-1}r^{k-1} - a_{k-2}r^{k-2} - \cdots - a_0 = 0$$

- La obtención de las raíces de la ecuación característica nos va a permitir resolver la recurrencia.

Recurrencias Lineales Homogéneas (II)

- Si las raíces $\lambda_1, \cdot \cdot \cdot, \lambda_k$ son todas diferentes, la solución a la recurrencia homogénea es

$$T(n) = \alpha_1 \lambda_1^n + \alpha_2 \lambda_2^n + \cdots + \alpha_k \lambda_k^n$$

- Los valores de α_i dependen de las condiciones iniciales. Para obtenerlos se resuelve el sistema de ecuaciones que resulta de igualar la ecuación anterior a las condiciones iniciales (n= 1,2,..,k):

$$\begin{aligned} T(1) &= \alpha_1 \lambda_1 + \alpha_2 \lambda_2 + \cdots + \alpha_k \lambda_k \\ T(2) &= \alpha_1 \lambda_1^2 + \alpha_2 \lambda_2^2 + \cdots + \alpha_k \lambda_k^2 \\ &\dots \\ T(k) &= \alpha_1 \lambda_1^k + \alpha_2 \lambda_2^k + \cdots + \alpha_k \lambda_k^k \end{aligned}$$

Ejemplo. Resolver Recurrencia Lineal Homogénea

Sucesión de Fibonacci: $T(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ T(n-1) + T(n-2) & n > 2 \end{cases}$

La ecuación característica del caso general *T(n) – T(n-1) – T(n-2) = 0* es

$r^2 - r - 1 = 0$ cuyas raíces son $r = \frac{1 \pm \sqrt{5}}{2}$

La ecuación cerrada será: $T(n) = \alpha_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + \alpha_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$

Sustituyendo los casos iniciales obtenemos el sistema de ecuaciones

$$T(1) = 1 = \alpha_1 \left(\frac{1+\sqrt{5}}{2}\right) + \alpha_2 \left(\frac{1-\sqrt{5}}{2}\right)$$

$$T(2) = 1 = \alpha_1 \left(\frac{1+\sqrt{5}}{2}\right)^2 + \alpha_2 \left(\frac{1-\sqrt{5}}{2}\right)^2$$

Al resolverlo se obtienen los valores de las constantes: $\alpha_1 = 1/\sqrt{5}$, $\alpha_2 = -1/\sqrt{5}$

Recurrencias Lineales Homogéneas(III)

- Si alguna raíz λ_i es múltiple, el término correspondiente es $p(n) \cdot \lambda_i^n$, donde μ es la multiplicidad de la raíz, y $p(n)$ es un polinomio de grado $\mu - 1$.

- Ejemplo: Resolver $T(n) = 4T(n-1) - 4T(n-2)$, con $T(1) = 1$ y $T(2) = 4$.

  La ecuación característica es $r^2 - 4r + 4 = 0$ cuya raíz doble es $r = 2$.

  Obtenemos entonces $T(n) = (\alpha_1 + \alpha_2 n) \cdot 2^n$.

  Sustituyendo los casos iniciales y resolviendo el sistema de ecuaciones

$$T(1) = 1 = (\alpha_1 + \alpha_2)2$$
$$T(2) = 4 = (\alpha_1 + 2\alpha_2)2^2$$

  resulta $\alpha_1 = 0$ y $\alpha_2 = 1/2$, por lo que $T(n) = 0.5\, n \cdot 2^n$

Recurrencias Lineales No Homogéneas (I)

- Si la recurrencia lineal $T(n) = a_{k-1}T(n-1) + a_{k-2}T(n-2) + \cdots + a_0T(n-k) + f(n)$ es no homogénea, $f(n) \neq 0$, podemos aplicar transformaciones matemáticas y conseguir una recurrencia lineal homogénea.

```java
/**
 * @param n número natural
 * @return el n-ésimo elemento de fibonacci
 * Implmentación ineficiente
 */
public static int fibonacci(int n){
    //n >=0
    if (n==0) return 0;
    if (n==1) return 1;
    else return fibonacci(n-1)+fibonacci(n-2);
}
```

- Tamaño de la entrada: valor de n
- Operación básica: suma
- La complejidad viene determinada por una recurrencia lineal no homogénea
- $T(n) = T(n-1) + T(n-2) + 1$

Resolver T(n) = T(n-1) + T(n-2) + 1

1. Calculamos la recurrencia para entrada n-1:

   $T(n-1) = T((n-1)-1) + T((n-1)-2) + 1;$

   $T(n-1) - T(n-2) - T(n-3) = 1$

2. Restamos ambas expresiones:

$$\begin{array}{lr}
T(n) - T(n-1) - T(n-2) & = 1 \\
\quad - T(n-1) + T(n-2) + T(n-3) & = -1 \\
\hline
T(n) - 2T(n-1) + 0 \quad + T(n-3) & = 0
\end{array}$$

3. Resolvemos la recurrencia lineal homogénea resultante.

   El polinomio característico es $x^3 - 2x^2 + (0 \cdot x) + 1 = 0$

   con raíces $x_1 = 1, x_2 = \frac{1+\sqrt{5}}{2}$ y $x_3 = \frac{1-\sqrt{5}}{2}$

   La ecuación resultante es

$$T(n) = \alpha_1 1^n + \alpha_2 \left(\frac{1+\sqrt{5}}{2}\right)^n + \alpha_3 \left(\frac{1-\sqrt{5}}{2}\right)^n =$$

$$\alpha_1 + \alpha_2 \left(\frac{1+\sqrt{5}}{2}\right)^n + \alpha_3 \left(\frac{1-\sqrt{5}}{2}\right)^n$$

Ejercicio: Finalizar el análisis de factorial

```java
/**
 * @param n número natural
 * @return n! = 1*...*n
 */
public static int factorial(int n){
   if (n<0) throw
     new IllegalArgumentException("número no válido");
   if (n == 0) return 1;
   else return n*factorial(n-1);
}
```

$$T(n) = \begin{cases} 0 & n = 0 \\ 1 + T(n-1) & n > 0 \end{cases}$$

Recurrencias Lineales No Homogéneas (II)

Para ecuaciones de recurrencia lineal no homogéneas del tipo:

$$T(n) = a_{k-1}T(n-1) + a_{k-2}T(n-2) + \cdot\cdot\cdot + a_0T(n-k) + \mathbf{b^n p(n)}$$

con $p(n)$ un polinomio de grado μ, $b \in \mathbb{R}$ y $\forall 1 \leq i \leq k: a_i \in \mathbb{R}$

el polinomio característico que nos permite encontrar la forma de la solución es

$$(x^k - a_{k-1}x^{k-1} - \ldots - a_1x^1 - a_0)\,\mathbf{(x-b)^{\mu+1}} = 0$$

Una vez obtenidas las raíces se procede de la forma descrita para el caso en que la ecuación sea homogénea.

Ejemplo

- Resolver la ecuación $T(n) = 2T(n-1) + 2^n$

$$T(n) - 2\,T(n-1) = \underbrace{2}_{b}{}^{n} \cdot \underbrace{1}_{p(n),\ \mu = 0}$$

- El polinomio característico es

$$(x-2)\,(x-2)^{0+1} = (x-2)^2 = 0$$

- Es decir, tenemos una única raíz, $x = 2$, con grado de multiplicidad 2.

- Por lo que la solución es de la forma:

$$T(n) = (\alpha_1 + \alpha_2 \cdot n)\cdot 2^n = \alpha_1 \cdot 2^n + \alpha_2 \cdot n \cdot 2^n$$

Recurrencias Lineales No Homogéneas (III)

En general, dada una recurrencia del tipo

$$T(n) = a_{k-1}T(n-1) + \cdots + a_0T(n-k) + \boldsymbol{p_1(n)b_1^n} + \cdots + \boldsymbol{p_m(n)b_m^n}$$

donde $\forall 1 \leq j \leq m\colon p_j(n)$ es un polinomio de grado μ_j y $b_j \in \mathbb{R}$, $\forall 1 \leq i \leq k\colon a_i \in \mathbb{R}$ y $\forall 1 \leq i,j \leq m, i \neq j\colon b_i \neq b_j$

Entonces el polinomio característico que nos permite encontrar la forma de la solución es

$$\left(x^k - a_{k-1}x^{k-1} - \ldots - a_1x^1 - a_0\right)\boldsymbol{(x-b_1)^{\mu_1+1} \cdot \ldots \cdot (x-b_m)^{\mu_k+1}} = 0$$

Una vez obtenidas las raíces se procede de la forma descrita para el caso en que la ecuación sea homogénea.

Ejemplo

- Resolver la ecuación $T(n) = 5 \cdot T(n-1) + 2^n + n \cdot 3^n$

$$T(n) - 5 \cdot T(n-1) = 2^n \cdot 1 + n \cdot 3^n$$

b_2

b_1

$p_1(n),\ \mu_1 = 0$

$p_2(n),\ \mu_2 = 1$

- El polinomio característico es

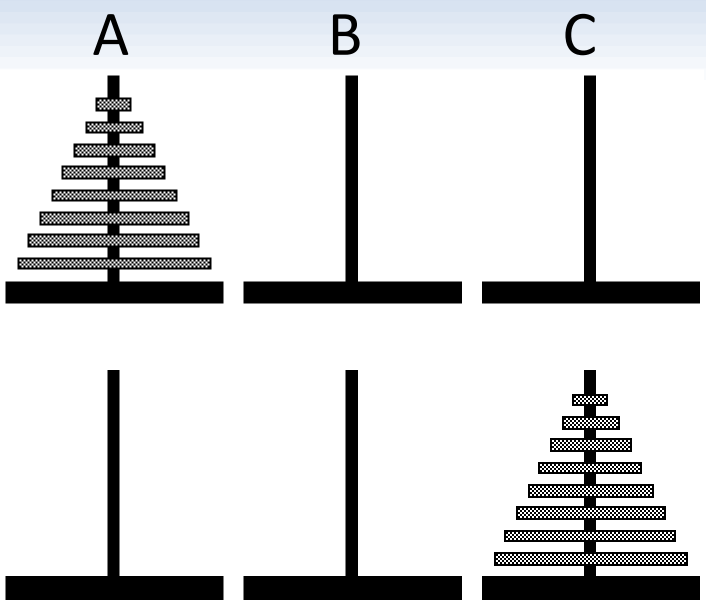
$$(x-5)\,(x-2)^{0+1}\,(x-3)^{1+1} = (x-5)\,(x-2)\,(x-3)^2 = 0$$

- Es decir, tenemos dos raíces simples, x = 5 y x = 2, y una raíz múltiple, x = 3, de grado 2.

- Por lo que la solución es de la forma:

$$T(n) = \alpha_1 \cdot 5^n + \alpha_2 \cdot 2^n + \alpha_3 \cdot 3^n + \alpha_4 \cdot n \cdot 3^n$$

Ejercicio: Analizar el Algoritmo Hanoi



- Supón tres varillas *A, B y C*, y *n* discos de distintos diámetros dispuestos inicialmente como se indica en la fila superior.
- El objetivo del juego es pasar todos los discos de la varilla *A* a la *C* (usando *B* como auxiliar) de manera que nunca haya un disco sobre otro de diámetro más pequeño.
- Sólo se puede pasar un disco cada vez entre varillas.

```
/**
 * @param n numero de discos
 * @param a, b, c varillas origen, intermedia y destino
*/

public static void hanoi(int n, char a,char b,char c){
  // n >=1
  if (n>0){
    hanoi(n-1,a,c,b);
    System.out.println("Pasar un disco de "+a+" a "+c);
    hanoi(n-1,b,a,c);
  }
}
```

Ejercicio: Analizar el Algoritmo Hanoi

```java
/**
 * @param n numero de discos
 * @param a, b, c varillas origen, intermedia y destino
*/

public static void hanoi(int n, char a,char b,char c){
  // n >=1
  if (n>0){
    hanoi(n-1,a,c,b);
    System.out.println("Pasar un disco de "+a+" a "+c);
    hanoi(n-1,b,a,c);
  }
}
```

- Tamaño de la entrada: número de discos de la torre.
- Operación primitiva: pasar un disco de una varilla a otra
- Función de complejidad:

  $T(n) = T(n-1) + 1 + T(n-1) = 2T(n-1) + 1$ si n>0,

  T(0) = 0.
- Resolver la recurrencia lineal no homogénea por sustitución hacia atrás y mediante la técnica del polinomio característico.

Recurrencias No Lineales

- Se aplican transformaciones y cambios de variable para conseguir una recurrencia lineal.

```java
/**
 * @param l lista con n > 0 elementos (min <= max)
 * @param min y max, índices mínimo y máximo a sumar
 * @return Suma de los elementos de la lista
*/
public double suma(List<Double> l,int min,int max){
      if (max == min) return l.get(min);
      else {
         double s1 = suma(l, min,(min+max)/2);
         double s2 = suma(l, (min+max)/2+1,max);
         return s1+s2;
      }
}
```

- Tamaño de la entrada, n, es el número de elementos en la lista l
- Operaciones básicas: suma de números reales (Double) y accesos a la lista (l.get()).
- Complejidad:

$$T(n)=2T(n/2)+1,\ n > 1$$
$$T(1) = 1$$

1. Se hace un cambio de variable. En este caso, **$n=2^k$**

   $T(2^k) = 2\ T(2^k/2)+1 = 2\ T(2^{k-1}) + 1$

2. Se renombra la función compuesta $T(2^k)$ como $F(k)$

   $F(k) = 2F(k-1) + 1$

3. Se resuelve la recurrencia lineal.

   $\mathrm{F(k)} = \alpha_1 2^{\mathrm{k}} + \alpha_2$

4. Hay que deshacer los cambios, en orden inverso al que se aplicaron.

   $T(2^k) = \alpha_1 2^k + \alpha_2;$

   $T(n) = \alpha_1 \cdot n + \alpha_2;$

5. Calculamos las constantes.

   $\mathrm{T(n)} = 2\mathrm{n} - 1$

Ejercicio: Resolver $T(n) = n\, T^2(n/2)$

Algunas fórmulas útiles

$$\sum_{i=min}^{max} c \cdot a_i = c \sum_{i=min}^{max} a_i$$

$$\sum_{i=min}^{max} a_i \pm b_i = \sum_{i=min}^{max} a_i \pm \sum_{i=min}^{max} b_i$$

$$\sum_{i=min}^{max} c = c \cdot (max - min + 1)$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=min}^{max} a_i = \frac{(a_{min} + a_{max})}{2}(max - min + 1)$$

$$\forall \mathrm{i} \geq 0 \; a_{i+1} - a_i = d, \qquad d \in \mathbb{N}$$

$$log_a(x) = y \Leftrightarrow a^y = x$$

$$log_a(x * y) = log_a(x) + log_a(y)$$

$$log_a(x/y) = log_a(x) - log_a(y)$$

$$log_a(x^y) = y * log_a(x)$$

$$log_a(x) = \frac{log_b(x)}{log_b(a)}$$

$$a^{log_b(c)} = c^{log_b(a)}$$