Algoritmos recursivos

Clase 18

IIC 1253

Prof. Diego Bustamante

Outline

Tercer Acto: Aplicaciones

Algoritmos, grafos y números

Notación asintótica

Las funciones más usadas para los órdenes de notación asintótica tienen nombres típicos:

Notación	Nombre
$\Theta(1)$	Constante
$\Theta(\log n)$	Logarítmico
$\Theta(n)$	Lineal
$\Theta(n \log n)$	$n \log n$
$\Theta(n^2)$	Cuadrático
$\Theta(n^3)$	Cúbico
$\Theta(n^k)$	Polinomial
$\Theta(m^n)$	Exponencial
$\Theta(n!)$	Factorial

con $k \geq 0, m \geq 2$.

Algoritmos recursivos: un ejemplo

input : Arreglo ordenado $A[0, \ldots, n-1]$, elemento x, índices i, f
output: Índice $m \in \{0, \ldots, n-1\}$ o -1

```
BinarySearch(A, x, i, f):
    if i > f then
        return −1
    else if i = f then
        if A[i] = a then
            return i
        else
            return −1
    else
        m ← ⌊(i + f)/2⌋
        if A[m] < x then
            return BinarySearch(A, x, m + 1, f)
        else if A[m] > x then
            return BinarySearch(A, x, i, m − 1)
        else
            if A[m] = x then return m
```

Algoritmos recursivos: un ejemplo

- ¿Qué operaciones contamos?
- ¿Cuál es el peor caso?

Ejercicio

Encuentre una función $T(n)$ para la cantidad de comparaciones que realiza el algoritmo `BinarySearch` en el peor caso, en función del tamaño del arreglo.

Respuesta:

$$T(n) = \begin{cases} 3 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + 4 & n > 1 \end{cases}$$

Esta es una **ecuación de recurrencia**.

¿Cómo obtenemos una fórmula explícita?

Algoritmos recursivos: un ejemplo

Ejercicio

Encuentre una función $T(n)$ para la cantidad de comparaciones que realiza el algoritmo `BinarySearch` en el peor caso, en función del tamaño del arreglo.

Contaremos las comparaciones. Dividiremos el análisis del peor caso:

- Si el arreglo tiene largo 1, entramos en la instrucción 3 y luego hay una comparación $\Rightarrow T(n) = 3$, con $n = 1$.
- Si el arreglo tiene largo mayor a 1, el peor caso es entrar en el else de 8 y luego en la segunda llamada recursiva. En tal caso, se hacen las comparaciones de las líneas $1, 3, 10, 12$ a lo que sumamos las comparaciones que haga la llamada recursiva, que serán $T(\lfloor \frac{n}{2} \rfloor)$.

Entonces, nuestra función $T(n)$ será:

$$T(n) = \begin{cases} 3 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + 4 & n > 1 \end{cases}$$

Objetivos de la clase

- Resolver recurrencias con substituciones.
- Deducir complejidad sin substituciones.
- Conocer el teorema maestro de algoritmos.

Outline

Algoritmos recursivos: ecuaciones de recurrencia

Necesitamos *resolver* esta ecuación de recurrencia.

- Es decir, encontrar una expresión que no dependa de T, sólo de n.
- Técnica básica: sustitución de variables.

¿Cuál sustitución para n nos serviría en el caso anterior?

$$T(n) = \begin{cases} 3 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + 4 & n > 1 \end{cases}$$

Ejercicio

Resuelva la ecuación ocupando la sustitución $n = 2^k$.

Respuesta: $T(n) = 4 \cdot \log_2(n) + 3$, con n potencia de 2.

Algoritmos recursivos: ecuaciones de recurrencia

Ejercicio

Resuelva la ecuación ocupando la sustitución $n = 2^k$.

$$T(2^k) = \begin{cases} 3 & k = 0 \\ T(2^{k-1}) + 4 & k > 0 \end{cases}$$

Expandiendo el caso recursivo:

$$\begin{aligned} T(2^k) &= T(2^{k-1}) + 4 \\ &= (T(2^{k-2}) + 4) + 4 \\ &= T(2^{k-2}) + 8 \\ &= (T(2^{k-3}) + 4) + 8 \\ &= T(2^{k-3}) + 12 \\ &\vdots \end{aligned}$$

Ejercicio

Resuelva la ecuación ocupando la sustitución $n = 2^k$.

Deducimos una expresión general para $k - i \geq 0$:

$$T(2^k) = T(2^{k-i}) + 4i$$

Tomamos $i = k$:

$$T(2^k) = T(1) + 4k = 3 + 4k$$

Como $k = \log_2(n)$:

$$T(n) = 4 \cdot \log_2(n) + 3, \text{ con } n \text{ potencia de } 2$$

Problema: esto solo es válido cuando $n = 2^k$.

Notación asintótica condicional

Sea $P \subseteq \mathbb{N}$.

Definición

$$O(f \mid P) = \{g : \mathbb{N} \to \mathbb{R}^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0) \\ (n \in P \to g(n) \leq c \cdot f(n))\}$$

Las notaciones $\Omega(f \mid P)$ y $\Theta(f \mid P)$ se definen análogamente.

Estamos restringiendo a un tipo de n particular.

Volviendo al ejemplo...

Tenemos que $T(n) = 4 \cdot \log_2(n) + 3$, con n potencia de 2. ¿Qué podemos decir sobre la complejidad de T?

Sea $POTENCIA_2 = \{2^i \mid i \in \mathbb{N}\}$. Entonces:

$$T \in \Theta(\log_2(n) \mid POTENCIA_2)$$

Pero, queremos concluir que $T \in \Theta(\log_2(n))$...

Usaremos inducción.

Generalización de soluciones usando inducción

Para el ejemplo anterior:

Ejercicio

Demuestre que si $T \in O(\log_2(n) \mid POTENCIA_2)$, entonces $T \in O(\log n)$.

Algunas observaciones:

- Demostraremos que $(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(T(n) \leq c \cdot \log_2(n))$.
- Primero, debemos estimar n_0 y c (expandiendo T por ejemplo).
- ¿Cuál principio de inducción usamos?

Generalización de soluciones usando inducción

Ejercicio

Demuestre que si $T \in O(\log_2(n) \mid POTENCIA_2)$, entonces $T \in O(\log n)$.

Veamos los primeros valores de $T(n)$ para estimar c y n_0:

$$\begin{aligned} T(1) &= 3 \\ T(2) &= T(1) + 4 = 7 \\ T(3) &= T(1) + 4 = 7 \\ T(4) &= T(2) + 4 = 11 \end{aligned}$$

Podríamos tomar $c = 7$ y $n_0 = 2$, pues con $n = 1$:

$$T(1) = 3 \nleq 7 \cdot \log_2(1) = 0$$

y con $n = 2$

$$T(2) = 7 \leq 7 \cdot \log_2(2) = 7$$

La intuición nos dice $n_0 = 2$ y $c = 7$... lo demostraremos

Generalización de soluciones usando inducción

Ejercicio

Demuestre que si $T \in O(\log_2(n) \mid POTENCIA_2)$, entonces $T \in O(\log n)$.

PD: $\forall n \geq 2,\ T(n) \leq 7 \cdot \log_2(n)$. Por inducción fuerte:

BI: Además de $n = 2$, debemos mostrar la base para $n = 3$, puesto que depende de $T(1)$ que no está incluido en el resultado que estamos mostrando.

$$T(2) = 7 = 7 \cdot \log_2(2)$$
$$T(3) = 7 < 7 \cdot \log_2(3) \text{ pues el logaritmo es creciente}$$

HI: Supongamos que con $n \geq 4$, $\forall k \in \{2, \cdots, n-1\}$ se cumple que $T(k) \leq 7 \cdot \log_2(k)$.

Generalización de soluciones usando inducción

Ejercicio

Demuestre que si $T \in O(\log_2(n) \mid POTENCIA_2)$, entonces $T \in O(\log n)$.

TI: Como $n \geq 4$:

$$
\begin{aligned}
T(n) &= T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 4 && \text{/ HI} \\
&\leq 7 \cdot \log_2\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 4 && \text{/ log es creciente, sacamos el piso} \\
&\leq 7 \cdot \log_2\left(\frac{n}{2}\right) + 4 && \text{/ log de división} \\
&= 7(\log_2(n) - \log_2(2)) + 4 \\
&= 7 \cdot \log_2(n) - 7 + 4 \\
&= 7 \cdot \log_2(n) - 3 \\
&< 7 \cdot \log_2(n)_{\square}
\end{aligned}
$$

Outline

Ecuaciones de recurrencia: otra técnica

Definición

Una función $f : \mathbb{N} \to \mathbb{R}^+$ es **asintóticamente no decreciente** si:

$$(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(f(n) \leq f(n+1))$$

Ejemplos

Las funciones $\log_2(n)$, n, n^k y 2^n son asintóticamente no decrecientes.

Ecuaciones de recurrencia: otra técnica

Definición

Dado un natural $b > 0$, una función $f : \mathbb{N} \rightarrow \mathbb{R}^+$ es ***b*-armónica** si $f(b \cdot n) \in O(f)$.

Ejemplos

Las funciones $\log_2(n)$, n y n^k son b-armónicas para cualquier b.
La función 2^n no es 2-armónica (porque $2^{2n} \notin O(2^n)$).

Por contradicción, si $2^{2n} \in O(2^n)$ esto significa que

$$\begin{aligned} \exists n_0, \exists c, \forall n \geq n_0 \quad 2^{2n} &\leq c \cdot 2^n \\ 2^{n+n} &\leq c \cdot 2^n \\ 2^n \cdot 2^n &\leq c \cdot 2^n \\ 2^n &\leq c \end{aligned}$$

Como c es una constante, no importa qué tan grande sea, siempre podemos tomar un n lo suficientemente grande.

Ecuaciones de recurrencia: otra técnica

Sean $f, g : \mathbb{N} \to \mathbb{R}^+$, un natural $b > 1$ y

$$POTENCIA_b = \{b^i \mid i \in \mathbb{N}\}$$

Teorema

Si f, g son asintóticamente no decrecientes, g es b-armónica y $f \in O(g \mid POTENCIA_b)$, entonces $f \in O(g)$.

Este teorema nos permite deducir lo que queríamos,
sin tener que deducir n_0 y c.

Ejercicio

Demuestre el teorema.

Ecuaciones de recurrencia: otra técnica

Teorema

Si f, g son asintóticamente no decrecientes, g es b-armónica y $f \in O(g \mid POTENCIA_b)$, entonces $f \in O(g)$.

Como f es asintóticamente no decreciente:

$$(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(f(n) \leq f(n+1)) \tag{1}$$

Como g es asintóticamente no decreciente:

$$(\exists n_1 \in \mathbb{N})(\forall n \geq n_1)(g(n) \leq g(n+1)) \tag{2}$$

Como $f \in O(g \mid POTENCIA_b)$:

$$(\exists c \in \mathbb{R}^+)(\exists n_2 \in \mathbb{N})(\forall n \geq n_2)(n \in POTENCIA_b \rightarrow f(n) \leq c \cdot g(n)) \tag{3}$$

Ecuaciones de recurrencia: otra técnica

Teorema

Si f, g son asintóticamente no decrecientes, g es b-armónica y $f \in O(g \mid POTENCIA_b)$, entonces $f \in O(g)$.

Como g es b-armónica: $g(b \cdot n) \in O(g(n))$:

$$(\exists d \in \mathbb{R}^+)(\exists n_3 \in \mathbb{N})(\forall n \geq n_3)(g(b \cdot n) \leq d \cdot g(n)) \tag{4}$$

Tomamos $n_4 = max\{1, n_0, n_1, n_2, n_3\}$. Sea $n \geq n_4$ (para poder utilizar todas las inecuaciones). La idea es sustituir por potencias de b, para poder usar todas las ecuaciones. Para esto, vamos a acotar n entre potencias de b. Como $n \geq 1$, existe $k \geq 0$ tal que

$$b^k \leq n < b^{k+1} \tag{5}$$

Ecuaciones de recurrencia: otra técnica

Teorema

Si f, g son asintóticamente no decrecientes, g es b-armónica y $f \in O(g \mid POTENCIA_b)$, entonces $f \in O(g)$.

Como $n < b^{k+1}$, de (1): $f(n) \leq f(b^{k+1})$

De (3): $f(b^{k+1}) \leq c \cdot g(b^{k+1})$

De (5) multiplicando por b: $b^{k+1} \leq b \cdot n$

De (2): $g(b^{k+1}) \leq g(b \cdot n)$

De (4): $g(b \cdot n) \leq d \cdot g(n)$

Ecuaciones de recurrencia: otra técnica

Teorema

Si f, g son asintóticamente no decrecientes, g es b-armónica y $f \in O(g \mid POTENCIA_b)$, entonces $f \in O(g)$.

Combinando todo lo anterior:

$$f(n) \leq f(b^{k+1}) \leq c \cdot g(b^{k+1}) \leq c \cdot g(b \cdot n) \leq c \cdot d \cdot g(n)$$

Por lo tanto:

$$\forall n \geq n_4, f(n) \leq (c \cdot d) \cdot g(n)$$

y entonces $f \in O(g)$.

Ecuaciones de recurrencia: otra técnica

Volviendo al ejemplo de `BinarySearch`...

Ejercicio

Demuestre que $T \in O(\log n)$ usando el teorema anterior.

Algunas observaciones:

- Ya sabemos que $T \in O(\log_2(n) \mid POTENCIA_2)$.
- Ya sabemos que $\log_2(n)$ es asintóticamente no decreciente y 2-armónica.

Nos falta demostrar que T es asintóticamente no decreciente.
Usaremos inducción.

Ecuaciones de recurrencia: otra técnica

Ejercicio

Demuestre que

$$T(n) = \begin{cases} 3 & n = 1 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 4 & n > 1 \end{cases}$$

es asintóticamente no decreciente.

PD: $(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(T(n) \leq T(n+1))$

Tomamos $n_0 = 2$. PD: $(\forall n \geq 2)(T(n) \leq T(n+1))$

Para facilitar la demostración vamos a demostrar un resultado más fuerte:
PD: $(\forall n \geq 2)(T(m) \leq T(n))$, con $2 \leq m \leq n$

De este resultado se deduce que $T(n)$ es asintóticamente no decreciente (en lugar de demostrarlo para el antecesor, lo hacemos para todos los anteriores).

Ecuaciones de recurrencia: otra técnica

Ejercicio

Demuestre que

$$T(n) = \begin{cases} 3 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + 4 & n > 1 \end{cases}$$

es asintóticamente no decreciente.

PD: $(\forall n \geq 2)(T(m) \leq T(n))$, con $2 \leq m \leq n$

Demostraremos esto por inducción fuerte sobre n:

BI: Para $n = 2$, el único m que debemos mostrar es el mismo 2, y en este caso la propiedad se cumple trivialmente.

HI: Supongamos que $\forall k \in \{2, \ldots, n-1\}$ se cumple que

$$T(m) \leq T(k) \text{ , con } 2 \leq m \leq k$$

Ecuaciones de recurrencia: otra técnica

TI: PD: $(\forall m \leq n)(T(m) \leq T(n))$, con $n, m \geq 2$.

Como $2 \leq m \leq n$ se cumple que $1 \leq \left\lfloor \frac{m}{2} \right\rfloor \leq \left\lfloor \frac{n}{2} \right\rfloor$. Además, tenemos que $\left\lfloor \frac{n}{2} \right\rfloor < n$, y entonces podemos aplicar la HI:

$$\begin{array}{rcl} T\left(\left\lfloor \frac{m}{2} \right\rfloor\right) & \leq & T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \\ T\left(\left\lfloor \frac{m}{2} \right\rfloor\right) + 4 & \leq & T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 4 \\ T(m) & \leq & T(n) \end{array}$$

$\square$

Outline

Dividir para conquistar

- Muchos algoritmos conocidos y usados en la práctica se basan en dividir el input en instancias más pequeñas para resolverlas recursivamente.
- Típicamente, existe un umbral n_0 desde el cual se resuelve recursivamente el problema (es decir, para inputs de tamaño $n \geq n_0$).
- Se divide el input por una constante b y se aproxima a un entero (usando $\lfloor \rfloor$ o $\lceil \rceil$), haciendo a_1 y a_2 llamadas recursivas para cada caso.
- Además, en general se hace un procesamiento adicional antes o después de las llamadas recursivas, que llamaremos $f(n)$.

Dividir para conquistar: un ejemplo

Ejercicio

¿Cómo ordenamos dos listas ya ordenadas en una?

$$L_1 = \{4, 7, 17, 23\}$$

$$L_2 = \{1, 9, 10, 15\}$$

¿Cómo podemos ocupar esta técnica para ordenar una lista?

¿Cuál es la complejidad de este algoritmo?

Dividir para conquistar: un ejemplo

Ejercicio

¿Cómo ordenamos dos listas ya ordenadas en una?
¿Cuál es la complejidad de este algoritmo?

Recorremos ambas, comparando el primer elemento. En cada paso ponemos el menor de ellos en una nueva lista y avanzamos. Si alguna de las listas se acaba, ponemos lo que quede de la otra al final.

En el peor caso, recorremos ambas listas comparando uno por uno sus elementos, con lo que hacemos $n - 1$ comparaciones.

Dividir para conquistar: un ejemplo

Ejercicio

¿Cómo podemos ocupar esta técnica para ordenar una lista?

Suponiendo que tenemos un método `Combinar` que implementa el procedimiento visto anteriormente:

input : Arreglo $A[0, \ldots, n-1]$, largo n

output: Arreglo ordenado

`MergeSort`(A, n)**:**

- **if** $n \leq 1$ **then**
  - **return** A
- **else**
  - $m \leftarrow \lfloor n/2 \rfloor$
  - $A_1 \leftarrow \texttt{MergeSort}(A[0 \ldots m-1], m)$
  - $A_2 \leftarrow \texttt{MergeSort}(A[m \ldots n-1], n-m)$
  - **return** $\texttt{Combinar}(A_1, A_2)$

¿Cómo obtenemos la complejidad? ¿Habrá algún método adicional?

Teorema Maestro

Teorema

Si $a_1, a_2, b, c, c_0, d \in \mathbb{R}^+$ y $b > 1$, entonces para una recurrencia de la forma:

$$T(n) = \begin{cases} c_0 & 0 \leq n < n_0 \\ a_1 \cdot T\left(\lceil \frac{n}{b} \rceil\right) + a_2 \cdot T\left(\lfloor \frac{n}{b} \rfloor\right) + c \cdot n^d & n \geq n_0 \end{cases}$$

se cumple que:

$$T(n) \in \begin{cases} \Theta(n^d) & a_1 + a_2 < b^d \\ \Theta(n^d \cdot log(n)) & a_1 + a_2 = b^d \\ \Theta(n^{log_b(a_1 + a_2)}) & a_1 + a_2 > b^d \end{cases}.$$

¿Qué valor debe tomar n_0?

Teorema Maestro

La idea es que las llamadas recursivas deben ser más pequeñas que la original, para que el algoritmo termine. Esto significa que

$$\left\lfloor \frac{n}{b} \right\rfloor < n \quad \text{y} \quad \left\lceil \frac{n}{b} \right\rceil < n$$

Como $\left\lfloor \frac{n}{b} \right\rfloor \leq \left\lceil \frac{n}{b} \right\rceil$, basta con que $\left\lceil \frac{n}{b} \right\rceil < n$.

Como $\left\lceil \frac{n}{b} \right\rceil < \frac{n}{b} + 1$, basta con que

$$\begin{aligned} \frac{n}{b} + 1 &\leq n \\ n + b &\leq nb \\ b &\leq nb - n \\ \frac{b}{b-1} &\leq n \end{aligned}$$

Es decir, n_0 solo depende de b

Teorema Maestro

Teorema

Si $a_1, a_2, b, c, c_0, d \in \mathbb{R}^+$ y $b > 1$, entonces para una recurrencia de la forma:

$$T(n) = \begin{cases} c_0 & 0 \leq n < \frac{b}{b-1} \\ a_1 \cdot T\left(\lceil \frac{n}{b} \rceil\right) + a_2 \cdot T\left(\lfloor \frac{n}{b} \rfloor\right) + c \cdot n^d & n \geq \frac{b}{b-1} \end{cases}$$

se cumple que:

$$T(n) \in \begin{cases} \Theta(n^d) & a_1 + a_2 < b^d \\ \Theta(n^d \cdot log(n)) & a_1 + a_2 = b^d \\ \Theta(n^{log_b(a_1+a_2)}) & a_1 + a_2 > b^d \end{cases}.$$

Ejercicio

Demuestre el teorema.

Teorema Maestro

Ejercicio

¿Cuál es la complejidad de `MergeSort`?

Como vimos antes, el peor caso es que `Combinar` tenga que comparar todos los elementos. En tal caso, se hacen $n-1$ comparaciones, a la que sumamos la comparación que se hace para verificar el tamaño de la lista. Entonces, la ecuación de recurrencia para `MergeSort` es:

$$T(n) = \begin{cases} 1 & n < 2 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n & n \geq 2 \end{cases}$$

Aplicamos el teorema maestro:

$a_1 = 1, a_2 = 1, b = 2, c = 1, d = 1, c_0 = 1$

$a_1 + a_2 = 2, b^d = 2^1 = 2 \rightarrow$ Entramos en el segundo caso: $a_1 + a_2 = b^d$

Por lo tanto, $T(n) \in \Theta(n \cdot \log(n))$.

Outline

Objetivos de la clase

- Resolver recurrencias con substituciones.
- Deducir complejidad sin substituciones.
- Conocer el teorema maestro de algoritmos.