Algoritmos Secuenciales y Recursivos

M. Andrea Rodríguez-Tastets
Ayudante: Erick Elejalde

Universidad de Concepción,Chile
www.inf.udec.cl\ ∼andrea
andrea@udec.cl

I Semestre - 2013

Análisis secuencial

En un algoritmo sin llamadas recursivas, una vez seleccionado el tamaño del problema y la operación básica, el análisis se realiza utilizando técnicas tradicionales de conteo (sucesiones, progresiones aritméticas, sumas, etc.) y el concepto de orden de una función.

Secuencia

Sean $\{I_1, I_2\}$ una secuencia de conjunto de instrucciones I_1 e I_2, con complejidades $O(f)$ y $O(g)$ respectivamente, entonces el orden de la secuencia es $O(max(f, g))$.

Selección condicional

Sea $O(f)$ la complejidad de ejecución cuando la condición de selección es verdadera y sea $O(g)$ la complejidad cuando la condición es falsa. Entonces:

- Peor caso: $O(max(f, g))$.
- Mejor caso: $O(min(f, g))$.
- Caso promedio: se toma la probabilidad P de que la condición sea verdadera y entonces la complejidad está dada por $O(Pf) + O((1 - P)g)$.

Iteración

Si el bloque de complejidad $O(f)$ se realiza n veces, entonces la iteración es $O(nf(n))$. En muchas veces la complejidad depende del índice. Si la complejidades $O(f(i))$ para $1 \leq i \leq n$ la complejidad de la iteración es $\sum_{i=1}^{n} f(i)$.

Ejercicio

Sean A y B dos matrices de $n \times n$ de entradas reales. Calcular la matriz producto $C = AB$.

Algoritmo

```
MatrizProducto(n, A,B, var C)
begin
  for (i = 1) a n do                          I1
   for (j = 1) a n do                         I2
    C[i,j] ← 0;                               I3
    for (k = 1) a n do                        I4
     C[i,j] ← C[i,j] + A[i,k] * B[k,j];       I5
    endWhile
   endWhile
  endWhile
end
```

Ejercicio (cont.)

Tamaño del problema	n, la dimensión de las matrices.
Operación básica	La multiplicación entre las entradas de las matrices
Caso	El algoritmo hace el mismo número de operaciones en todos los casos.

Ejercicio: Análisis temporal

Se puede obtener el orden de complejidad si asignamos un orden a cada bloque de instrucciones según las estructuras de control.

1. La instrucción I_5 tiene un orden constante $O(1)$, lo que es razonable cuando los números que se multiplan tienen un tamaño fijo.
2. Como I_5 está en ciclo I_4, entonces el orden es $O(n \times 1) = O(n)$.
3. El ciclo I_2 contiene al conjunto de instrucciones I_3 e I_4, si I_3 es $O(1)$ se tienen que la secuencia tiene orden $O(n)$, pero ésta se realiza n veces, así el orden para este bloque es $O(n \times n) = O(n^2)$.
4. Por último, el ciclo I_1 contiene a la instrucción I_2 que se realiza n veces, de esta forma el orden para I_1 y en consecuencia para el algoritmo es $O(n \times n^2) = O(n^3)$.

Sistemas recurrentes

Existen ocasiones donde algún fenómeno que queremos estudiar puede ser modelado matemáticamente por funciones definidas recursivamente. Una función de este tipo se dice *ecuación recurrente*. Si además están dados uno o más valores n específicos, lo que se tiene es un sistema recurrente.

Los algoritmos al igual que las funciones también pueden ser definidos recursivamente, con entradas, salidas y reglas definidas para obtener entradas en función de las salidas.

La función de complejidad de un algoritmo recursivo queda determinada por la función recurrente que lo caracterice.

Factorial

```
     Factorial(n) ≡
     begin
1      if n = 0 then
2        Factorial ← 1;
3      else
4        Factorial ← n × Factorial(n − 1);
       endif
     end
```

Para encontrar la complejidad se debe elegir una operación tal que el orden del número de veces que se realiza sea igual al orden del total de operaciones del algoritmo, es decir, se debe realizar en cada una de las llamadas recursivas o en un número de veces que sea del mismo orden que el orden del total de llamadas.

Factorial: sistema recurrente

Considerando como operación básica la multiplicación, entonces el algoritmo para un problema de tamaño n hace una mutiplicación y resuelve un problema de tamaño $n-1$ que hace una multiplicación y resuelve un problema de tamaño $n-2$, y así sucesivamente. Asumiendo que la multiplicación es de orden constante, es decir O(1), la función de complejidad se puede expresar como un sistema recurrente de la siguiente forma.

(i) La ecuación recurrente: $T(n) = T(n-1) + 1$

(ii) Un conjunto de valores frontera o condiciones iniciales: $T(1) = 1$.

Solución de recurrecia: expansión

Asumiendo que la multiplicación es de orden constante, uno puede fácilmente expandir la recurrencia. El método de expansión de recurrencia consiste en utilizar la recurrencia misma para sustituir $T(n)$ por su valor en términos de $T(m)$, donde $m < n$, iterativamente hasta encontrar una condición de frontera $T(r)$ que tendrá un valor constante. En la ecuación recurrente se ponen todos los términos con T en el lado izquierdo de la ecuación y se realiza la expansión. Al sumar todas las ecuaciones obtenemos una expresión para $f(n)$ en término de n y algunas constantes. A esta expresión se le denomina *forma cerrada* para $T(n)$.

Factorial: expansión de recurrencia (para operación multiplicación $O(1)$)

$$\begin{aligned} T(n) - T(n-1) &= 1 \\ T(n-1) - T(n-2) &= 1 \\ T(n-2) - T(n-3) &= 1 \\ &\dots \\ T(2) - T(1) &= 1 \\ T(1) - T(0) &= 1 \\ T(0) &= 0 \end{aligned}$$

Sumando todas las ecuaciones, nos queda:
$T(n) = 1 + 1 + 1 + 1 \dots + 1 = n$.

Factorial: expansión de recurrencia (para operación multiplicación $O(1)$)

$$\begin{aligned} T(n) - T(n-1) &= 1 \\ T(n-1) - T(n-2) &= 1 \\ T(n-2) - T(n-3) &= 1 \\ &\dots \\ T(2) - T(1) &= 1 \\ T(1) - T(0) &= 1 \\ T(0) &= 0 \end{aligned}$$

Sumando todas las ecuaciones, nos queda:
$f(n) = 1 + 1 + 1 + 1 \dots + 1 = n$.

¿Qué hay de malo con esto?

Factorial: para un n arbitrariamente largo

- Cuando n es arbitrariamente largo uno no puedo pensar en que la multiplicación es constante.
- Cuando se multiplican dos número de n y m bit de representación, un algoritmo ingenuo toma $O(nm)$. Existen algoritmos más eficientes, Schönhage-Strassen toma O(n*log(n)*log(log(n))) for 2 n-bit numbers.
- Si asumimos el algoritmo ingenuo, en cada ciclo se multiplica un número que es $(k-1)!$ por k. Para representar $(k-1)!$ se usan $O(klogk)$ y para representar k se usan $O(logk)$. Luego la multiplicación ocupa $k(logk)^2$. El sistema recurrente queda expresado entonces como:

  (i) La ecuación recurrente: $T(n) = T(n-1) + n(logn)^2$
  (ii) Un conjunto de valores frontera o condiciones iniciales: $T(0) = 1$.

Luego, el costo es $\sum_{k=1}^{n} k(logk)^2 \leq (logn)^2(\sum_{k=1}^{n} k) = O(n^2(logn)^2)$.

Permutaciones: sistema recurrente

El número de permutaciones de n permutaciones se puede expresar como un sistema recurrente.

(i) La ecuación recurrente: $P(0) = 1$, ya que hay solo una permutación de 0 elementos.

(ii) $P(n) = nP(n-1)$, ya que podemos elegir un elemento de los n posibles y después cualquier secuencia que sea una permutación de los $n-1$ elementos.

Permutaciones: expansión

$$\begin{aligned} P(n) - nP(n-1) &= 0 \\ nP(n-1) - n(n-1)P(n-2) &= 0 \\ &\dots \\ n(n-1)(n-2)\dots 2 \times P(1) - n(n-1)(n-2)\dots 1 \times P(0) &= 0 \\ P(0) &= 1 \end{aligned}$$

Sustituyendo $P(0)$ y sumando ecuaciones:

$$\begin{aligned} P(n) - n(n-1)(n-2)\cdots \times 2 \times 1 &= 0 \\ P(n) &= n! \end{aligned}$$

Sistemas recurrentes: forma genérica

Los sistemas recurrentes obtenidos cuando se intenta calcular la complejidad de los algoritmos recursivos con frecuencia se ajustan a alguna forma estándar. Si conocemos la solución genérica para esa forma, nos ahorramos tener que hallar la solución cada vez que se presente un nuevo caso.

Sistemas recurrentes: forma genérica (cont.)

Consideremos el siguiente sistema recurrente:

$$T(1) = a$$
$$T(n) = cT(n-1) + b$$

Donde a, b, c son enteros no negativos y $f : \mathbb{N} \to \mathbb{R}$. Entonces expandiendo:

$$T(n) - cT(n-1) = b$$
$$cT(n-1) - c^2 T(n-2) = cb$$
$$c^2 T(n-2) - c^3 T(n-3) = c^2 b$$
$$\ldots$$
$$c^{n-2} T(2) - c^{n-1} T(n-2) = c^{n-2} b$$
$$c^{n-2} T(2) - c^{n-1} T(n-2) = c^{n-2} b$$

Sustituyendo

$$T(n) = c^{n-1} a + b + cb + c^2 b * \ldots c^{n-2} b$$
$$T(n) = c^{n-1} a + b \sum_{i=0}^{n-2} c^i$$

Teorema 7

Si a, b, c son enteros positivos y $T : \mathbb{N} \to \mathbb{R}$ tal que

$$\sum_{i=0}^{n} c^i = n + 1 \text{ si } c = 1$$

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1} - 1}{c - 1} \text{ si } c \neq 1$$

Entonces

$$T(n) = a + b(n + 1) \text{ si } c = 1$$

$$T(n) = ac^{n-1} + b\frac{c^{n-1} - 1}{c - 1} \text{ si } c \neq 1$$

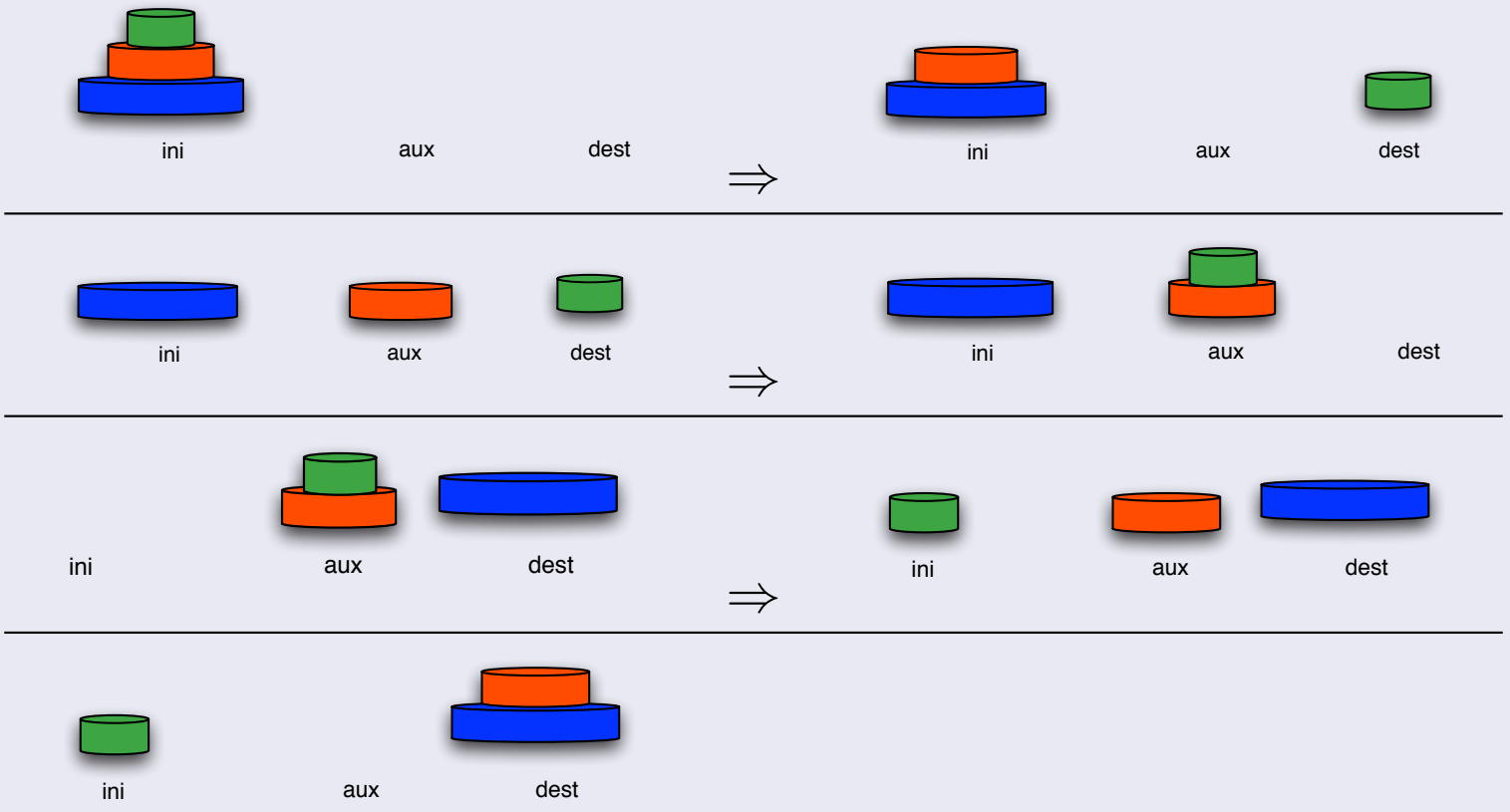
Torres de Hanoi

Se tienen n discos apilados de mayor a menor, en la torre fuente, y se tienen que mover a la torre destino utilizando una torre auxiliar; los discos se tienen que mover uno a uno y apilarlos de tal forma que nunca quede un disco bajo otro mayor que él.

La solución recursiva al problema se ilustra en el algoritmo siguiente

```
      Hanoi(n, ini, med, fin) ≡
      begin
1       if n = 1 then
2        escribe(ini, →, fin);
3       else
4         Hanoi(n − 1, ini, fin, med);
5         escribe(ini, →, fin);
6         Hanoi(n − 1, med, ini, fin);
        endif
      end
```

Solución torres de hanoi



Torres de Hanoi: sistema recurrente

(i) $T(1) = 1$.

(ii) $T(n) = 2T(n-1) + 1$.

Por Teorema 7, entonces $a = 1$, $b = 1$ y $c = 2$, entonces $T(n) = 2^{n-1} + \frac{2^{n-1}-1}{1} = 2^n - 1$, que es $O(2^n)$.

Número de hojas: sistema recurrente

Encontrar el número de hojas que tiene un árbol de altura h, donde cada nodo tiene un grado $k \geq 2$.

(i) $T(1) = k$.

(ii) $T(h) = kT(h-1)$.

Por Teorema 7, entonces $a = k$, $b = 0$ y $c = k$, entonces $f(h) = kk^{h-1} = k^h$.

Solución de recurrencia: método de sustitución

El método de sustitución utiliza dos pasos:

- Proponer una forma de solución.
- Hacer un prueba por inducción para encontrar las constantes y mostrar que la solución funciona.

Inducción matemática

- Probar que funciona para $P(1)$
- Probar que si $P(1) \ldots P(n)$ son ciertas, entonces también funciona para $P(n+1)$, con n cualquier entero positivo.

Ejemplo de sustitución

Considere el caso de una recurrencia $T(n) = 2T(\lfloor n/2 \rfloor) + n$, y asumamos una solución $T(n) = O(nlogn)$.
La idea entonces es probar que $T(n) \leq cnlgn$ para un $c > 0$.
Haciendo inducción:

- Asumimos que se cumple para $\lfloor n/2 \rfloor$, esto es $T(\lfloor n/2 \rfloor) = c\lfloor n/2 \rfloor log(\lfloor n/2 \rfloor)$
- Luego por sustitución:

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor log(\lfloor n/2 \rfloor)) + n \\ &\leq cnlog(n/2) + n \\ &\leq cnlogn - cnlog2 + n \\ &\leq cnlogn - cn + n \\ &\leq cnlogn \end{aligned}$$

con $c \geq 1$.

Ejemplo de sustitución (cont..)

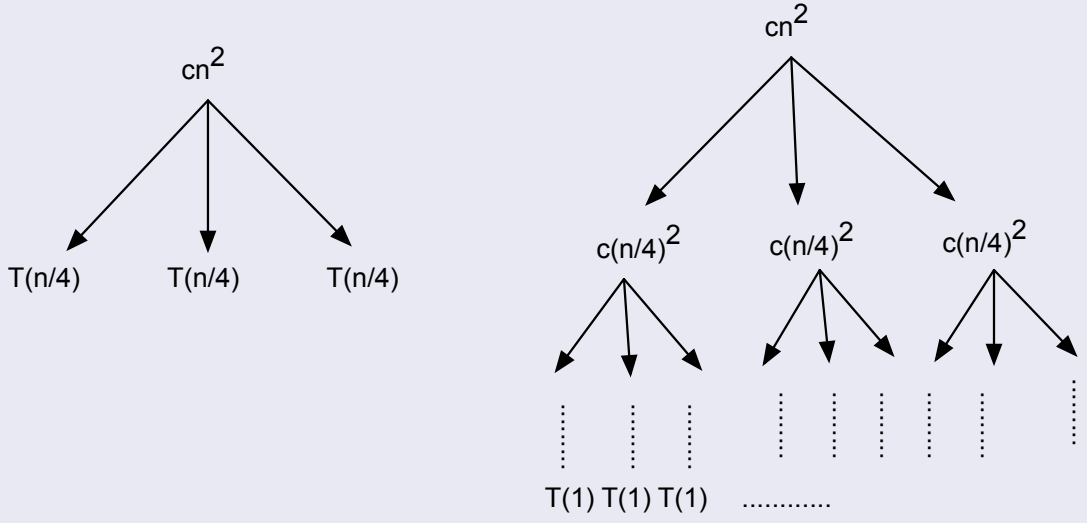
La inducción requiere que se pruebe que la solución funciona para las condicions de borde. Si asumimos que $T(1) = 1$ es la condición de borde, pero $T(n) \leq cnlogn$ nos da que $T(1) \leq c1lg1 = 0$, una contradicción. Sin embargo, podemos hacer que $T(n) \leq cnlogn$ sea verdadero para algún $n \geq n_0$. Entonces lo que debemos hacer es definir por ejemplo para un $n_0 = 2$. Así hemos extendido la condición de borde.

Ejercicios: métodos de sustitución

1. Demostrar que $T(n) = T(\lceil n/2 \rceil) + 1$ es $O(logn)$
2. Demostrar que $T(n) = 2T(\lceil n/2 \rceil) + n$ no es solo $O(nlogn)$ sino $\Omega(nlogn)$

Solución de recurrencia = árbol recursivo

El siguiente árbol representa la recurrencia $T(n) = 3T(n/4) + cn^2$.

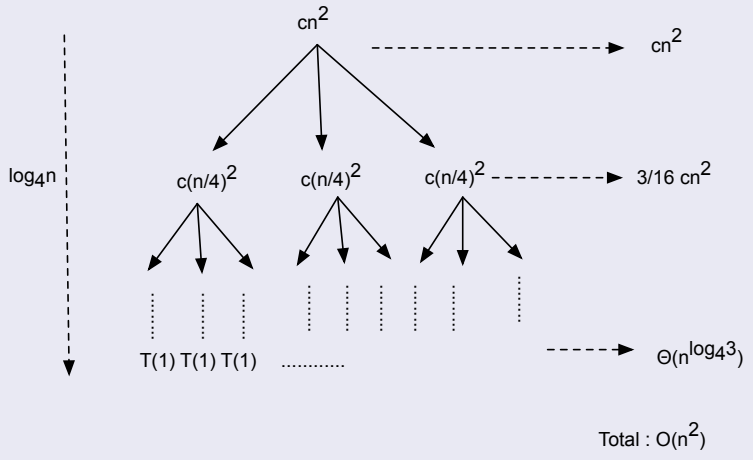


Método del árbol recursivo

En un árbol recursivo, cada nodo representa el costo de un subproblema dentro del proceso recursivo. Se suma el costo de cada nivel del árbol para obtener el costo por nivel y luego se suman los costos por niveles.

Método del árbol recursivo (cont...)

El tiempo de ejecución estimado para la recurrencia $T(n) = 3T(n/4) + cn^2$ es:



En cada iteración el problema se va reduciendo en 4. Eso implica que en la última iteración i, el problema es de tamaño 1. Generalizando tenemos $n/4^i = 1$, lo que nos queda que $n = 4^i$. Despejando el máximo número de iteraciones es $i = log_4 n$. Dado que en cada iteración se hacen 3 llamadas, el último nivel tiene 3^i nodos, con un costo de $3^i c(n/4^i)^2 = (3/16)^i cn^2$. El último nivel es tal que $i = log_4 n$, con $3^{log_4 n} = n^{log_4 3}$ nodos. Si cada uno nodo del último nivel tiene un costo $\Theta(1)$, entonces el costo al último nicel es $\Theta(n^{log_4 3})$.

Ejercicios: árbol recursivo

1. Use el método recursivo para determinar una cota superior para la recurrencia $T(n) = 3T(\lfloor n/2 \rfloor) + n$. Use el método de sustitución para verificar la respuesta.
2. Muestre que la solución de $T(n) = T(n/3) + T(2n/3) + cn$, donde c es una constante, es $\omega(nlogn)$ usando el árbol recursivo.

Teorema 8: Teorema Maestro

Sean a, b, c enteros tales que $a \geq 1, b > 1$ y $c > 0$, y $f : \mathbb{N} \to \mathbb{R}$ una función arbitraria. Defina $T(n)$ sobre enteros no negativos por la recurrencia:

$$T(n) = aT(n/b) + f(n)$$

donde n/b significa $\lfloor n/b \rfloor$ o $\lceil n/b \rceil$. Entonces $T(n)$ puede ser dominada asintóticamente como sigue

(i) Si $f(n) \in O(n^{log_b a - \epsilon})$ para alguna constante $\epsilon > 0$, entonces $T(n) \in \Theta(n^{log_b a})$.

(ii) $f(n) \in \Theta(n^{log_b a})$, entonces $T(n) \in \Theta(n^{log_b a} logn)$.

(iii) Si $f(n) \in \Omega(n^{log_b a + \epsilon})$ para alguna constante $\epsilon > 0$, y si $af(n/b) \leq cf(n)$ para alguna constante $c < 1$ y un n suficientemente grande, entonces $T(n) \in \Theta(f(n))$.

Teorema Maestro: comentarios

- En cada caso, la función $f(n)$ es comparada con $n^{log_b a}$.
- En el primer caso, la función $n^{log_b a}$ es más grande. Más aún, $f(n)$ debe ser polinomialmente menor que $n^{log_b a}$ por un factor n^{ϵ}.
- En el caso 3, la función $f(n)$ es la más grande. Más aún, $f(n)$ debe ser polinomialmente grande y satisfacer la condición regular que $af(n/b) \leq cf(n)$.
- En el caso 2, ellas son del mismo tamaño.

Teorema Maestro : Uso

(i) Si $f(n) \in O(n^{log_b a - \epsilon})$ para alguna constante $\epsilon > 0$, entonces $T(n) \in \Theta(n^{log_b a})$.

(ii) $f(n) \in \Theta(n^{log_b a})$, entonces $T(n) \in \Theta(n^{log_b a} logn)$.

(iii) Si $f(n) \in \Omega(n^{log_b a + \epsilon})$ para alguna constante $\epsilon > 0$, y si $af(n/b) \leq cf(n)$ para alguna constante $c < 1$ y un n suficientemente grande, entonces $T(n) \in \Theta(f(n))$.

Considere el caso de $T(n) = 3T(n/4) + nlogn$. Entonces, $a = 3$, $b = 4$, $f(n) = nlogn$ y $n^{log_4 3} = O(n^{0,793})$. Ya que $f(n) = \Omega(n^{log_4 3 + \epsilon})$, con $\epsilon \sim 0{,}2$, entonces se puede aplicar el caso 3, siempre que se cumpla la regularidad para $f(n)$. Para valores suficientemente grandes de n, $af(n/b) = 3(n/4)log(n/4) \leq (3/4)nlogn = cf(n)$ para $c = 3/4$.

Teorema Maestro : Demostración (1/7)

- No se requiere entender la demostración para aplicar el teorema maestro. Se verá en clases la forma intuitiva de la demostración, la que se encuentra en forma extensa en el libro guía.
- La demostración se divide en dos partes: Primero se muestra que el teorema funciona para $T(n)$ que se aplican sobre potencias exactas de b, es decir, para $n = 1, b, b^2, ...$ Esta parte usa 3 lemas. Luego se extiende para n que son enteros positivos.

Teorema Maestro : Demostración (2/7)

- La primera parte se reduce a tres lemas. El primer lema reduce el problema maestro a evaluar una expresión que contiene una sumatoria. Esto se basa en el árbol recursivo.
- El segundo lema encuentra cotas a la sumatoria.
- El tercer lema combina el primero con el segundo para probar que el problema maestro se cumple para el caso de que b sea potencias de n.

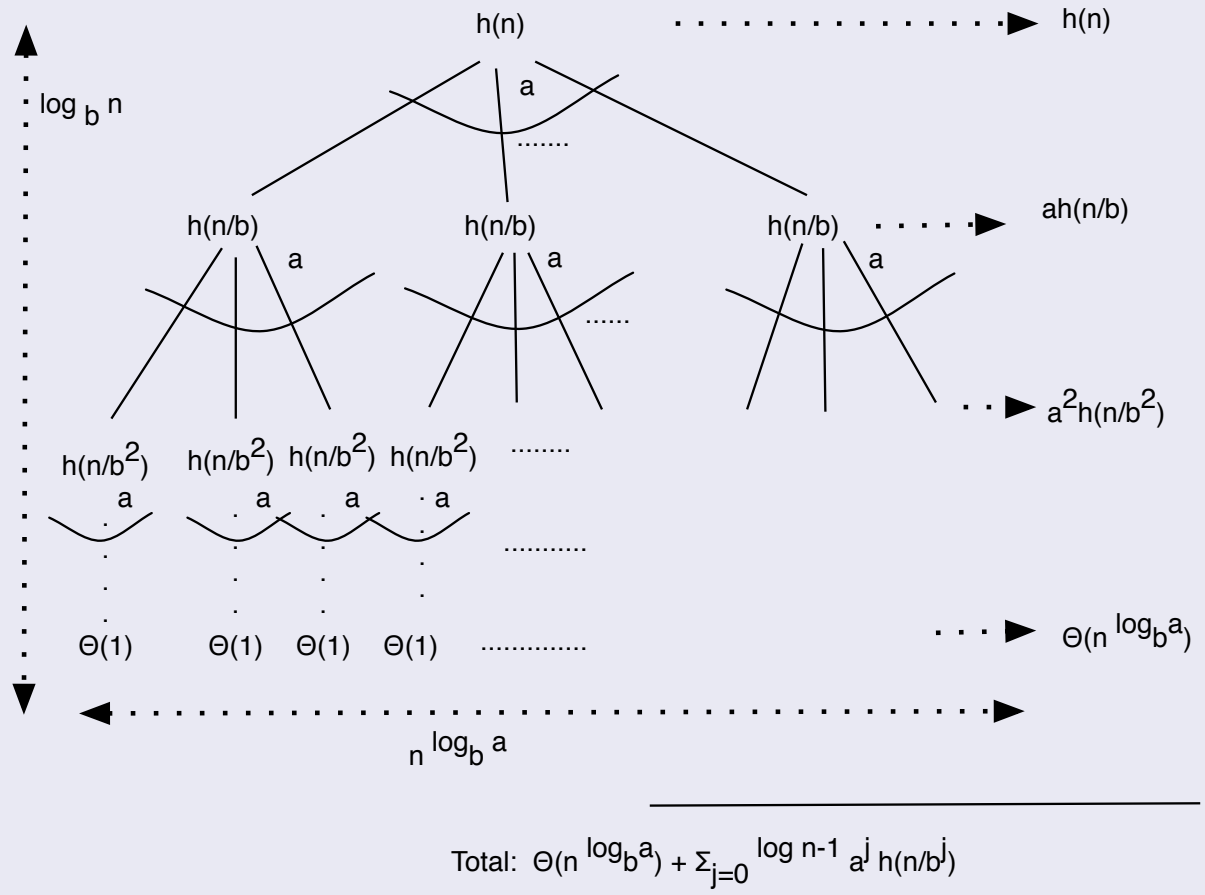
Lema 1: Demostración (3/7)

Sean a, b, c enteros tales que $a \geq 1, b > 1$ y $c > 0$, y sea $f : \mathbb{N} \to \mathbb{R}$ una función definida para potencias de b. Defina $T(n)$ sobre potencias exactas de b por la recurrencia:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ aT(n/b) + f(n) & \text{if } n = b^i \end{cases}$$

donde i es un entero positivo. Entonces:
$T(n) = \Theta(n^{log_b a}) + \sum_{j=0}^{log_b n - 1} a^j f(n/b^j)$

Lema 1: Demostración (4/7)



Ojo: en esta figura considere $h(n)$ la función $f(n)$ del teorema.

Lema 2: Demostración (5/7)

Sean a, b, c enteros tales que $a \geq 1, b > 1$ y $c > 0$, y sea $f : \mathbb{N} \to \mathbb{R}$ una función definida para potencias de b. Una función $g(n)$ defina sobre potencias exactas de b por:

$$g(n) = \sum_{j=0}^{log_b n - 1} a^j f(n/b^j)$$

puede ser dominada asintóticamente por potencias exactas de b como sigue:
(1) Si $f(n) = O(n^{log_b a - \epsilon})$ para algún $\epsilon > 0$, entonces $g(n) = O(n^{log_b a})$.
(2) Si $f(n) = \Theta(n^{log_b a})$, entonces $g(n) = \Theta(n^{log_b a log n})$.
(3) Si $af(n/b) \leq cf(n)$ para algún $c < 1$ y todo $n \geq b$, entonces $g(n) = \Theta(f(n))$.

Lema 3: Demostración (6/7)

Sean a, b, c enteros tales que $a \geq 1, b > 1$ y $c > 0$, y sea $f : \mathbb{N} \to \mathbb{R}$ una función definida para potencias de b. La función $T(n)$ sobre potencias exactas de b queda definida por la recurrencia:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ aT(n/b) + f(n) & \text{if } n = b^i \end{cases}$$

donde i es positivo. Entonces $T(n)$ puede ser dominada asintóticamente por potencias exactas de b como sigue

(i) Si $f(n) \in O(n^{log_b a - \epsilon})$ para alguna constante $\epsilon > 0$, entonces $T(n) \in \Theta(n^{log_b a})$.

(ii) $f(n) \in \Theta(n^{log_b a})$, entonces $T(n) \in \Theta(n^{log_b a} logn)$.

(iii) Si $f(n) \in \Omega(n^{log_b a + \epsilon})$ para alguna constante $\epsilon > 0$, y si $af(n/b) \leq cf(n)$ para alguna constante $c < 1$ y un n suficientemente grande, entonces $T(n) \in \Theta(f(n))$.

Para esta demo se usa el Lema 2 para evaluar la expresión en el Lema 1.

Teorema Maestro : Demostración (7/7)

- Para completar la prueba del teorema maestro hay que extender la primera parte para $T(n)$ con n cualquier entero positivo. Esto es, se extendie el análisis para situaciones en las cuales $\lfloor \rfloor$ y $\lceil \rceil$ son usados en la recurrencia de maestro, de manera que se define para todo entero n.

Teorema Maestro : Demostración (7/7)

- Para completar la prueba del teorema maestro hay que extender la primera parte para $T(n)$ con n cualquier entero positivo. Esto es, se extendie el análisis para situaciones en las cuales $\lfloor \rfloor$ y $\lceil \rceil$ son usados en la recurrencia de maestro, de manera que se define para todo entero n.

Ejercicios:

- Considere la recurrencia $T(n) = 16T(n/4) + n^2 logn$ e indique cómo aplicaría el Teorema Maestro.
- Escriba un algoritmo recursivo para determinar la altura de un árbol binario, muestre que es correcto. Determine el sistema recurrente y en función de eso, escriba una expresión general que resuelva la recurrencia.