Centro de
Investigación en
Computación

Divide and Conquer

Divide y Vencerás

Fundamento matemático

Tema 5

Course

Analysis and design of algorithms

Instructor

Acosta Bermejo Raúl et al.

Lecture notes

Instituto
Politécnico
Nacional

Table of contents (outline)

Tabla de contenido

Mucha
teoría y ejemplos
Slides 14-73 = 59
38%

Ejemplos

Ejemplos
extra

5.2 Recurrence relations and the master method

Técnica para resolverlas

Solving recurrences

Resolviendo recurrencias

3 basic methods + 1 mechanical method

1. **Substitution method**. We make a guess for the solution and then we use mathematical induction to prove the the guess is correct or incorrect.
2. **Iterative method**

   In the iteration method we iteratively "unfold" the recurrence until we "see the pattern".
3. **Recurrence Tree Method**. In this method, we draw a recurrence tree and calculate the time taken by every level of tree. Finally, we sum the work done at all levels. To draw the recurrence tree, we start from the given recurrence and keep drawing till we find a pattern among levels. The pattern is typically a arithmetic or geometric series.

Solving recurrences

Resolviendo recurrencias

4. **Master Method**. It is a direct way to get the solution. The master method works only for following type of recurrences or for recurrences that can be transformed to the given formula.
5. **Akra-Bazzi method**. ”On the solutions of linear recurrences equations”, 1996.

Recurrence relations and the master method

Relaciones de recurrencia y el método maestro

Recurrence relation

- It is an equation that:
  - Recursively defines a sequence or multidimensional array of values.
  - Once one or more initial terms are given: each further term of the sequence or array is defined as a function of the preceding terms.
- Example:

$$n! = n * (n-1)!$$
$$0! = 1$$

- Linear recurrence relation: first-degree polinomial.
- Homogeneous recurrence relation: All the terms have the same exponent.

Recurrence relations and the master method

Relaciones de recurrencia y el método maestro

Examples of recurrence relation

- Fibonacci numbers

  Linear recurrence relation, homogeneous recurrence relation with constant coefficients.

$$F_n = F_{n-1} + F_{n-2}\ where F_0 = 0, F_1 = 1$$
$$F(n) = F(n-1) + F(n-2)$$

- Multidimensional recurrence relation

  The binomial coefficients:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

These numbers also arise in combinatorics, where the formula gives the number of different combinations of b elements that can be chosen from an n-element set.

Recurrence relations

Metodología

Examples

Determine if the following recurrence relations are linear homogeneous recurrence relations with constant coefficients.

1. $P_n = (1.11)P_{n-1}$ — A linear homogeneous recurrence relation of degree one.
2. $a_n = a_{n-1} + a^2_{n-2}$ — Not linear
3. $f_n = f_{n-1} + f_{n-2}$ — A linear homogeneous recurrence relation of degree two.
4. $H_n = 2H_{n-1}+1$ — Not homogeneous
5. $a_n = a_{n-6}$ — A linear homogeneous recurrence relation of degree six.
6. $B_n = nB_{n-1}$ — Does not have constant coefficient.

Recurrence relations

Metodología

Secuencia de pasos

1. Deducir la ecuación de recurrencia a partir del algoritmo.
   - Distinguir (diferencia sutil) entre la recurrencia (cálculo), y
   - La ecuación de recurrencia que representa la complejidad.
2. Elegir un método de solución.
   - Substitution method
   - Iterative method
   - Recurrence Tree Method
   - Master Method
3. Verificar
   - Con más de un método.
   - Las excepciones (si hay) del método. “Extremos”

Solving recurrences

Resolviendo recurrencias

Deducción de recurrencia

A partir del algoritmo:

$$T(n) = \begin{cases} complejidad\ de\ los\ casos\ base & \text{Si n corresponde a los casos base} \\ complejidad\ de\ los\ casos\ recursivos & \text{Si n corresponde a los casos recursivos} \end{cases}$$

- Una recurrencia es una ecuación o una desigualdad que describe una función en términos de su propio valor sobre entradas más pequeñas.
- Una recurrencia es lineal, si cada llamada recursiva genera cuando mucho otra llamada recursiva, en caso contrario es no-lineal.
- Por ejemplo el factorial: $n! = n * (n-1)!$

$$f(n) = \begin{cases} 1, & n = 0 \\ nf(n-1), & n > 0 \end{cases}$$

Solving recurrences

Resolviendo recurrencias

Implementación

Factorial n!

Versión Recursiva

```
factorial(unsigned n){
  if( n==1 )
     return 1;
  else
     return n*factorial(n - 1);
}
```

Versión Iterativa

```
int res = 1, i;
for (i = 2; i <= n; i++)
   res *= i;
return res;
```

Ver/recordar

El uso de la pila/estado
Código ensamblador

https://www.sciencedirect.com/topics/computer-science/factorial-function
https://www.geeksforgeeks.org/program-for-factorial-of-a-number/

Solving recurrences

Resolviendo recurrencias

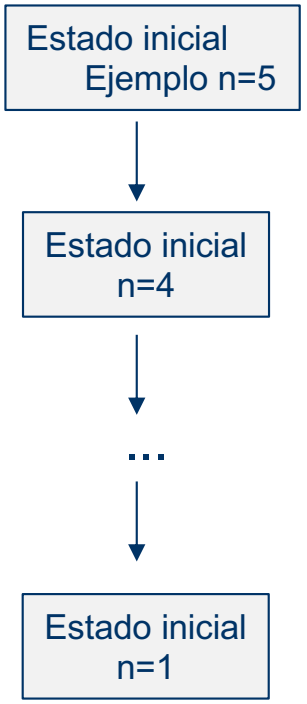
Estado
- Parámetros
- Variables locales

Implementación

Factorial n!

Versión Recursiva

```
factorial(unsigned n){
  if( n==1 )
     return 1;
  else
     return n*factorial(n - 1);
}
```



Versión Iterativa con pila

```
Stack pila;
```

Tarea optativa: programar este algoritmo

Solving recurrences

Resolviendo recurrencias

Deducción de recurrencia

Factorial $n! = n * (n-1)!$

$$f(n) = \begin{cases} 1, & n = 0 \\ n \cdot f(n-1), & n > 0 \end{cases}$$

De tal forma que:

$f(4) = 4 \cdot f(3)$

$f(3) = 3 \cdot f(2)$

$f(2) = 2 \cdot f(1)$

$f(1) = 1 \cdot f(0)$

$f(0) = 1$

- Es una recurrencia lineal.
- La complejidad está dada por las multiplicaciones que se realizan.
- Así que su complejidad está dada por:

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

Si desglosamos, por ej., $t(4)$= 1+$t(3)$=1+1+ $t(2)$=… se observa que $t(4)$ hace 4 multiplicaciones y al generalizar (inducción) obtenemos $O(n)$.

Recurrence relations and the master method

Relaciones de recurrencia y el método maestro

1. Substitution method

- Idea: Make a guess for the form of the solution and prove by induction.
- Can be used to prove both upper bounds O() and lower bounds Ω().

Example 1

Solve $T(n) = 2T(\frac{n}{2}) + n$

Guess $T(n) \leq cn \log n$ for some constant c (that is, $T(n) = O(n \log n)$).

Proof:

Base case: we need to show that our guess holds for some base case (not necessarily n = 1, some small n is ok).

Ok, since function constant for small constant n.

Recurrence relations and the master method

Relaciones de recurrencia y el método maestro

Substitution method

Example 1

Solve $T(n) = 2T(\frac{n}{2}) + n$

Proof:

(Question: Why not n − 1?)

Assume holds for n/2: $T(\frac{n}{2}) \leq c\,\frac{n}{2}\log\frac{n}{2}$

Prove for n: $T(n) \leq cn \log n$

$T(n) = 2T(\frac{n}{2}) + n$

$\leq 2(c\,\frac{n}{2}\log\frac{n}{2}) + n$ Aqui se sustituye lo que se asume como cierto.

$= cn \log\frac{n}{2} + n$

$= cn \log n - cn \log 2 + n$

$= cn \log n - cn + n$

So ok if $c \geq 1$

Recurrence relations and the master method

Relaciones de recurrencia y el método maestro

Substitution method

Similarly it can be shown that:

- $T(n) = \Omega(n \log n)$
- $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n$ is $\Theta(n \lg n)$.

Do it as exercise!

- The hard part of the substitution method is often to make a good guess.
- How do we make a good (i.e. tight) guess? Unfortunately, there's no "recipe" for this one.
- Try iteratively $O(n^3)$, $\Omega(n^3)$, $O(n^2)$, $\Omega(n^2)$ and so on.
- Try solving by iteration to get a feeling of the growth.

Recurrence relations and the master method

Relaciones de recurrencia y el método maestro

Substitution method

Exercise:

T(1) = 1

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

Solution:

- https://www.cs.cornell.edu/courses/cs3110/2014sp/recitations/24/using-the-substitution-and-master-method.html

Recurrence relations and the master method

Relaciones de recurrencia y el método maestro

Substitution method

Exercise:

Compute the n-th Fibonacci number.

Solution:

- http://homepages.math.uic.edu/~jan/mcs360f10/index.html

Varios ejemplos

- https://brilliant.org/wiki/the-substitution-method-for-solving-recurrences/

Solving recurrences

Resolviendo recurrencias

2. Iterative method

- Este método convierte la función de recurrencia en una sumatoria que posteriormente se evalúa para obtener un límite de la función.
- De manera general el método iterativo consiste en:
  i. Expansión de la recurrencia para identificar patrones de regularidad y expresarlos como sumas de términos dependientes sólo de *n* y de las condiciones iniciales.

     "*Unfold" the recurrence until we "see the pattern*"

  i. Evaluación de la sumatoria.

Links

- http://www.bowdoin.edu/~ltoma/teaching/cs231/fall05/Lectures/reccurences.pdf

Solving recurrences

Resolviendo recurrencias

Iterative method

1. Determinar una función $T(n)$ que represente el número de operaciones básicas efectuadas por el algoritmo para el tamaño base y para los casos definidos por recursividad fuera de la base.
2. Evaluar la función $T(n)$ del caso progresivo con un conjunto pequeño de valores consecutivos de n.
3. Sustituir los resultados del punto anterior en cada expresión antecesora.
4. Identificar un patrón con el cual pueda generalizarse la representación de los términos del desarrollo.
5. Expresar en notación de sumatorias el patrón identificado, destacando dos partes en la expresión: un conjunto de términos **inductivos** y un término llamado de **base** o de paro, referenciado por m.
6. Determinar un valor de m para el cual se satisfaga o al menos se aproxime a la igualdad.
7. Sustituir m en la expresión de sumatoria obtenida en el paso 5.
8. Resolver la sumatoria con una expresión equivalente o aproximada cuya única variable independiente es n.

Solving recurrences

Resolviendo recurrencias

Example 1 : Iterative method

Solve $T(n) = 8T(\frac{n}{2}) + n^2$ $(T(1) = 1)$

$T(n) = n^2 + 8T(\frac{n}{2})$

$= n^2 + 8(8T(\frac{n}{2^2}) + (\frac{n}{2})^2)$

$= n^2 + 8^2T(\frac{n}{2^2}) + 8(\frac{n^2}{4})$

$= n^2 + 2n^2 + 8^2T(\frac{n}{2^2})$

$= n^2 + 2n^2 + 8^2 (8T(\frac{n}{2^3}) + (\frac{n}{2^2})^2)$

$= n^2 + 2n^2 + 8^3 T(\frac{n}{2^3}) + 8^2 (\frac{n^2}{4^2})$

$= n^2 + 2n^2 + 2^2 n^2 + 8^3 T(\frac{n}{2^3})$

$= \ldots$

$= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \ldots$

De acuerdo a la metodología aquí el primer paso ya se hizo, es decir, determinar T(n).

Si observamos la serie está descrita por 2 partes:

1. T(algo)
2. 2^{val}

Solving recurrences

Resolviendo recurrencias

Example 1

Recursion depth: How long (how many iterations) it takes until the subproblem has constant size?

i times where $\frac{n}{2^i} = 1 \Rightarrow i = \log n$.

What is the last term? $8^i T(1) = 8^{\log n}$

$T(n) = n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \ldots + 2^{\log n - 1}n^2 + 8^{\log n}$

$= \sum_{k=0}^{\log n - 1} 2^k \, n^2 + 8^{\log n}$

$= n^2 \sum_{k=0}^{\log n - 1} 2^k + (2^3)^{\log n}$

$= n^2 \ \Theta(n) + (2^3)^{\log n}$

1

Now $\sum_{k=0}^{\log n - 1} 2^k$ is a geometric sum so we have:

$$\sum_{k=0}^{\log n - 1} 2^k = \Theta(2^{\log n - 1}) = \Theta(n)$$

2

Solving recurrences

Resolviendo recurrencias

Example 1

2

The last term

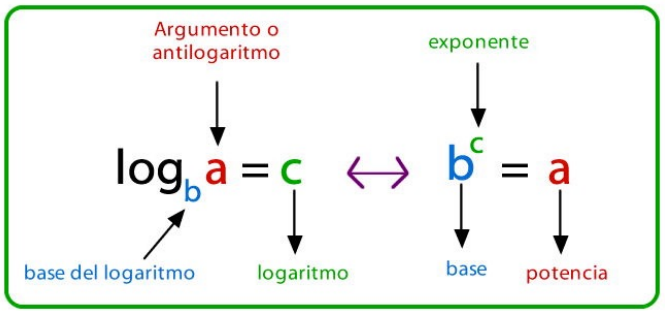
$(2^3)^{\log n} = (2^{\log n})^3 = n^3$

Ley de Exp: $(X^m)^n = X^{mn}$

Log propiedades: $2^{\log n}$

Finally we have:

Log base 2



$T(n) = n^2\ \Theta(n) + n^3$

$= \Theta(n^3)$

Solving recurrences

Resolviendo recurrencias

Ejercicios: Iterative method

Resolver:

$$T(n) = \begin{cases} 1, & n = 1 \\ n + 2t(n/2), & n > 1 \end{cases}$$

Example 2

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ n + 3t(\lfloor n/4 \rfloor), & n > 1 \end{cases}$$

Example 3

Solución en las notas (ppt) o en el link.

Solving recurrences

Resolviendo recurrencias

$$T(n) = \begin{cases} 1, & n = 1 \\ n + 2t(n/2), & n > 1 \end{cases}$$

Example 2

Solución

1. Se generan varias recurrencias (4):

$$T(n) = n + 2t(n/2)$$
$$T(n/2) = n/2 + 2t(n/4)$$
$$T(n/4) = n/4 + 2t(n/8)$$
$$T(n/8) = n/8 + 2t(n/16)$$

2. Se remplazan en la original

$$T(n) = n + \frac{2n}{2} + 2 * 2t(\frac{n}{4})$$
$$T(n) = n + \frac{2n}{2} + 2 * 2 * \frac{n}{4} + 2 * 2 * 2T(\frac{n}{8})$$
$$T(n) = n + \frac{2n}{2} + 2 * 2 * \frac{n}{4} + 2 * 2 * 2 * \frac{n}{8} + 2 * 2 * 2 * 2 * T(\frac{n}{16})$$
$$T(n) = n + 21\frac{n}{2^1} + 2^2 * \frac{n}{2^2} + 2^3\frac{n}{2^3} + 2^4 * T(\frac{n}{2^4})$$

3. No se conocen cuantos términos tiene la seria para alcanzar el caso base.

Usaremos el índice m.

$$T(n) = [n + 21\frac{n}{2^1} + 2^2 * \frac{n}{2^2} + 2^3\frac{n}{2^3} + \ldots + \text{Término}_{\text{m-1}}] + [\text{Término}_{\text{m}}]$$

Solving recurrences

Resolviendo recurrencias

Example 2

Solución

3. Como sumatoria

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

4. La sumatoria se simplifica al considerar que el último termino corresponde al caso base con

n=1 y $T(1)$=1.

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

$$T(n) = \sum_{i=0}^{m-1} n + 2^m$$

5. Se encuentra el valor de m igualando las dos condiciones de paro.

$$T\left(\frac{n}{2^m}\right) = T(1)$$

$$\frac{n}{2^m} = 1, 2^m = n, \log(2^m) = \log(n) \rightarrow m = \log_2(n)$$

Solving recurrences

Resolviendo recurrencias

Example 2

Solución

6. Remplazando m en la ultima ec. de 4 que es:

$$T(n) = \sum_{i=0}^{m-1} n + 2^m$$

$$T(n) = \sum_{i=0}^{\log(n)-1} n + 2^{log\ n}$$

n $\sum$ 1 + se anula $2^{\log_2}$ y queda n

7. Simplificando:

$T(n) = n\ log\ n + n$

O(n log n)

Example 3

Ejercicio extra clase.

Solving recurrences

Resolviendo recurrencias

3. Recurrence Tree Method

1. Draw a recurrence tree.
2. Calculate the time taken by every level of tree.
3. Sum the work done at all levels.

Example 1

Now we use this method with:

$$T(n) = \begin{cases} 0, & n = 1 \\ n^2 + 2T(n/2), & n > 1 \end{cases}$$

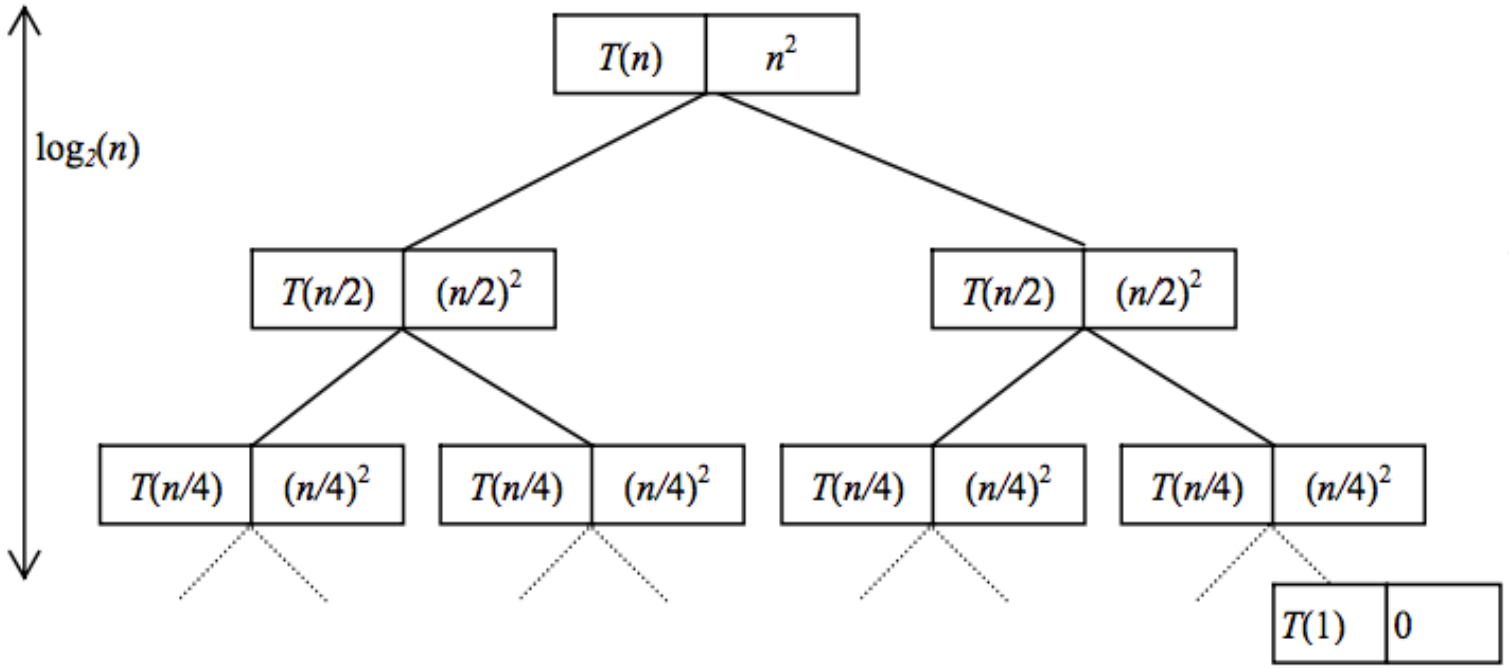
Solving recurrences

Recurrence Tree Method

Example 1

Step 1. Draw a recurrence tree

$$T(n) = \begin{cases} 0, & n = 1 \\ n^2 + 2T(n/2), & n > 1 \end{cases}$$



$T_{nivel\ 0} = n^2$

$T_{nivel\ 1} = 2(n/2)^2 = n^2/2$

$T_{nivel\ 2} = 4(n/4)^2 = n^2/4$

$T_{nivel\ m} = 0$

Solving recurrences

Resolviendo recurrencias

Example 1

Step 2. Calculate the time (every level)

Un análisis visual del árbol revela que en cada nivel:

- El número de subproblemas aumenta en potencias de dos.

  $2^0 \rightarrow 2^1 \rightarrow 2^2 \rightarrow 2^3 \rightarrow \ldots \rightarrow 2^m$

- El tamaño de los problemas disminuye en potencias de dos.

  $(\frac{1}{2^0}\,n) \rightarrow (\frac{1}{2^1}\,n) \rightarrow (\frac{1}{2^2}\,n) \rightarrow (\frac{1}{2^3}\,n) \rightarrow \ldots \rightarrow (\frac{1}{2^m}\,n)$

Es importante observar que en el último nivel se tienen 2^m nodos y cada nodo es de tamaño $\frac{1}{2^m}\,n$.

Solving recurrences

Resolviendo recurrencias

Example 1

Step 3. Sum the work done at all levels.

$$T(n) = (n^2 + \frac{1}{2} n^2 + \frac{1}{4} n^2 + \ldots + \frac{1}{2^{m-1}} n^2) + 2^m T(\frac{1}{2^m} n)$$

Usando algebra para simplificar:

$$\frac{1}{2^m} n = 1$$
$$n = 2^m$$
$$\log_2 n = \log_2 2^m$$
$$\log_2 n = m$$

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

El valor de *m* se determina igualando las dos expresiones del caso base (*), quedando m=log n.

Solving recurrences

Resolviendo recurrencias

Example 1

Step 3. Sum the work done at all levels.

Para la solución de $T(n)$ se usa la siguiente serie geométrica con $r = ½$:

$$\sum_{i=0}^{k} ar^i = a\left(\frac{r^{k+1}-1}{r-1}\right) \qquad \sum_{i=0}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k}$$

Hacer el algebra en estos 2 lugares.

$$T(n) = n^2 \sum_{i=0}^{(\log n)-1} \frac{1}{2^i} = n^2\left[2 - \left(\frac{1}{2^{(\log n)-1}}\right)\right] = 2n^2 - 2n$$

Finalmente nos da:

$T(n) = O(n^2)$

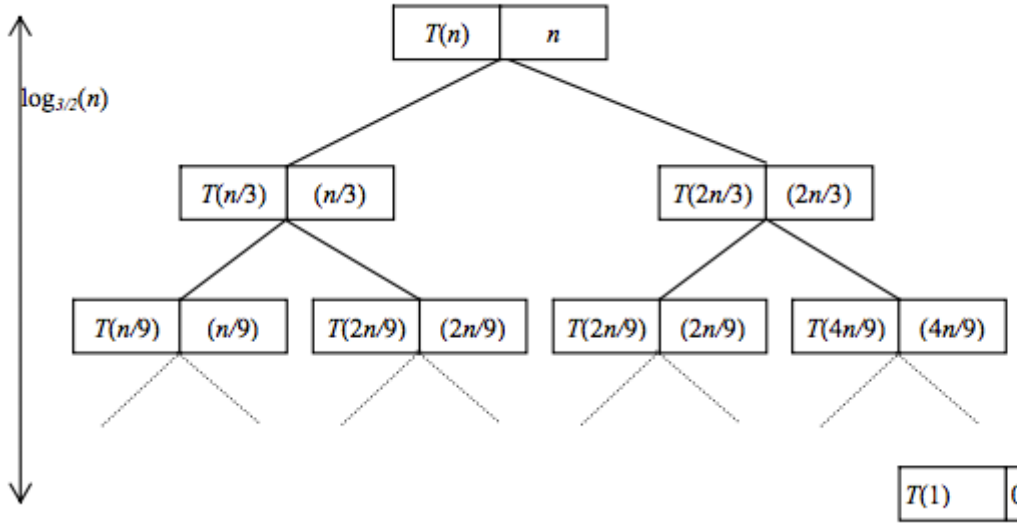
Solving recurrences

Resolviendo recurrencias

Example 2

Step 1. Draw a recurrence tree

$$T(n) = \begin{cases} 0, & n = 1 \\ T(\frac{n}{3}) + T\left(\frac{2n}{3}\right) + n, & n > 1 \end{cases}$$



$T_{nivel\ 0} = n$

$T_{nivel\ 1} = (n/3) + (2n/3) = n$

$T_{nivel\ 2} = (n/9) + \ldots + (4n/9) = n$

$T_{nivel\ m} = 0$

Solving recurrences

Resolviendo recurrencias

$$T(n) = \begin{cases} 0, & n = 1 \\ T(\frac{n}{3}) + T\left(\frac{2n}{3}\right) + n, & n > 1 \end{cases}$$

Example 2

Step 2. Calculate the time (every level)

Un análisis visual del árbol debe revelar el comportamiento de cada nivel:

- Como es el número de subproblemas? Cómo aumenta?
- Como es el tamaño de los problemas? como disminuye?
- Que se tiene en el último nivel?

Solving recurrences

Resolviendo recurrencias

Example 2

Step 2. Calculate the time (every level)

Un análisis visual del árbol revela que en cada nivel:

- El número de subproblemas aumenta en potencias de dos.

  $2^0 \rightarrow 2^1 \rightarrow 2^2 \rightarrow 2^3 \rightarrow \ldots \rightarrow 2^m$

- El tamaño más grande de problema disminuye por un factor de $(2/3)^i$.

$$\left(\frac{2}{3}\right)^0 n \rightarrow \left(\frac{2}{3}\right)^1 n \rightarrow \left(\frac{2}{3}\right)^2 n \rightarrow \ldots \rightarrow \left(\frac{2}{3}\right)^{m-1} n \rightarrow \left(\frac{2}{3}\right)^m n$$

Solving recurrences

Resolviendo recurrencias

Example 2

Step 3. Sum the work done at all levels.

- En cada nivel la complejidad algorítmica permanece constante:

$$n \to n \to \ n \to \ldots \to n \to 2^m \, T(\left(\frac{2}{3}\right)^m n)$$

- Usando la condición de terminación y expresando la recurrencia como sumatoria:

$$\mathrm{T(n)} = \sum_{i=0}^{m-1} n + \ 0$$

Solving recurrences

Resolviendo recurrencias

Example 2

Step 3. Sum the work done at all levels.

- Calculando m igualando las expresiones de terminación:

$$T(\left(\frac{2}{3}\right)^{m} n) = T(1)$$

$$\left(\frac{2}{3}\right)^{m} n = 1 \quad \rightarrow m = \log_{3/2} n$$

- Ahora se soluciona $T(n)$ y en lugar de una serie se usa la desigualdad de O ($\leq$).

Solving recurrences

Resolviendo recurrencias

Example 2

Step 3. Sum the work done at all levels.

- Ahora se soluciona $T(n)$ y en lugar de una serie se usa la desigualdad de O ($\leq$).

$$T(n) \leq \sum_{i=0}^{m-1} n \leq \sum_{i=0}^{\log_{3/2}(n)-1} n \leq n \sum_{i=0}^{\log_{3/2}(n)-1} 1 \leq n(\log_{3/2}(n) - 1 + 1)$$

$$T(n) \leq n \log_{\frac{3}{2}}(n) = O(n \log n)$$

Que se usa aqui?

Master Theorem

Teorema Maestro

Forma simplificada del cálculo
De la complejidad

Teoría y Ejemplos
26 slides

Master theorem

Teorema maestro

- Fue popularizado por el libro Introducción a los algoritmos por **C**ormen, **L**eiserson, **R**ivest, y **S**tein (**CLRS**), en el cual fue tanto introducido como probado formalmente.
- Proporciona una solución sencilla en términos asintóticos (usando una Cota superior asintótica) para ecuaciones de recurrencia.
- Muchos algoritmos recursivos surgen en el Algoritmo divide y vencerás.
- No todas las ecuaciones de recurrencia pueden ser resueltas con el uso del Teorema maestro.

Master theorem

Teorema maestro

- La **recurrencia general** para estrategias D&C puede ser definida por:

$$T(n) = \begin{cases} d, & n = 1 \\ aT(\frac{n}{b}) + f(n), & n > 1 \end{cases}$$

$$f(n) = n^c$$

- En la función de recurrencia:
  - $T(n)$ está definida en los enteros no negativos.
  - *n* es el tamaño del problema principal.
  - *a* es la constante que representa los sub-problemas (1<a< n).
  - Cada sub-problema es de tamaño *n/b* (1<b< n).
  - Para dividir el problema en sub-problemas y combinar las soluciones de los sub-problemas existe cierto costo no recursivo que se expresan en la función $f(n)$.

Master theorem

Teorema maestro

Let a≥ 1 and $b > 1$ be constants
Let f(n) be a function, and
Let T(n) defined on the nonnegqative integers.

where we interpret n/b to mean either
$\lfloor n/b \rfloor$ or $\lceil n/b \rceil$

- *T(n)* puede ser acotado asintóticamente por 3 casos:

Case 1: If $f(n)=O(n^{\log_b a-\varepsilon})$ for some constant ε > 0,
Then $T(n)=\Theta(n^{\log_b a})$.

Case 2: If $f(n)=\Theta(n^{\log_b a})$,
Then $T(n)=\Theta(n^{\log_b a}\log n)$.

Case 3: If $f(n)=\Omega(n^{\log_b a+\varepsilon})$ for some constant ε > 0 and
if $af(n/b) \leq cf(n)$ for some constant c < 1 and
all sufficiently large n,
Then $T(n)=\Theta(f(n))$.

Regularity condition

Master theorem

Teorema maestro

- When analyzing algorithms, recall that we only care about the asymptotic behavior.
- Recursive algorithms are no different. Rather than solve exactly the recurrence relation associated with the cost of an algorithm, it is enough to give an asymptotic characterization.
- You cannot use the Master Theorem if:
  - T(n) is not monotone, example: T(n) = sin(n)
  - f(n) is not a polynomial, example: $T(n) = 2T(n/2) + 2^n$
  - b cannot be expressed as a constant, example:

    $T(n) = T(\sqrt{n})$

Master theorem

Teorema maestro

Las **complicaciones** de su aplicación son:

- f(n) no sólo tiene que ser menor que $n^{log_b a}$, debe ser polinomialmente menor. Caso 1.
- f(n) no sólo tiene que ser mayor que $n^{log_b a}$, debe ser polinomialmente mayor y además satistacer la condición de regularidad. Caso 3.
- Si las dos funciones son del mismo orden, se multiplica por un factor logarítmico. Caso 2.

Los casos anteriores no cubren todos los casos para f(n):

- Hay un hueco entre los casos 1 y 2 cuando f(n) es menor pero no polinomialmente menor que $n^{log_b a}$.
- Hay un hueco entre los casos 2 y 3 cuando f(n) es mayor pero no polinomialmente mayor que $n^{log_b a}$.

Master theorem

Teorema maestro

Una **serie geométrica** es la suma de un número infinito de términos que tiene una razón constante entre sus términos sucesivos.

$$a + ar + ar^2 + ar^3 + \ldots$$

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots$$

$$T(n) = a\,T(n/b) + f(n)$$

General Format

Case 1: If $f(n) = O(n^{\log_b a - \varepsilon})$ for cte. $\varepsilon > 0$,
Then $T(n) = \Theta(n^{\log_b a})$

La complejidad es
- Serie geométrica Creciente

Case 2: If $f(n) = \Theta(n^{\log_b a})$,
Then $T(n) = \Theta(n^{\log_b a} \lg n)$

- Constante en cada nivel

Case 3: If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for cte. $\varepsilon > 0$ and f(n) satisfies the regularity condition,
Then $T(n) = \Theta(f(n))$.

- Serie geométrica Decreciente

Regularity condition

if $af(n/b) \leq cf(n)$ for some constant c < 1 and all sufficiently large n.
La complejidad del nivel 1 disminuye o se mantiene con respecto a la del nivel 0.

Master theorem

Teorema maestro

Para el caso general complicado

- La superioridad de una función $h(n)$ sobre otra $g(n)$ se dice que es polinomial, si es significativa.
- Esto es, existe una constante $\varepsilon>0$ tal que $g(n)$ es asintóticamente más pequeña que $h(n)$ por un factor n^{ε}.

$$n^{\varepsilon}\, g(n) \leq h(n)$$

Caso 1

$$h(n) = n^{\log_b a},\ y\ g(n) = f(n) \ \therefore f(n) \leq n^{\log_b a-\epsilon}$$

Caso 3

$$h(n) = f(n),\ y\ g(n) = n^{\log_b a} \ \therefore f(n) \geq n^{\log_b a+\epsilon}$$

- Las recurrencias en las que $f(n)$ es superior o inferior por un factor que no es polinomial, no deben ser resueltas con el método maestro.

Master theorem

Teorema maestro

Para el caso general (que es complicado hay que probar)

- Entonces lo que se suele hacer para verificar los casos es calcular:

$$\lim_{x\to\infty}\left(\frac{f(n)}{g(n)}\right)$$

Caso 1: El denominador crece más rápido y el límite tiende a **disminuir** (valores de 1 o 0 según la formula).

$$\lim_{x\to\infty}\left(\frac{f(n)}{n^{\log_b a-\epsilon}}\right)$$

Caso 3: El denominador crece más lento y el límite tiende a **aumentar** (infinito).

$$\lim_{x\to\infty}\left(\frac{f(n)}{n^{\log_b a+\epsilon}}\right)$$

Master theorem

Teorema maestro

Ejemplo 1: Sea T(n) = 9T(n/3) + n

- Si a = 9, b = 3, f(n) = n, g(n) = $n^{\log_3 9-\epsilon} = n^{2-\epsilon}$
- Verificando los límites con valores diferentes de ϵ (ya se ve el caso 1) y dividiendo entre n:

$$\lim_{x\to\infty}\left(\frac{f(n)}{g(n)}\right) \qquad \lim_{x\to\infty}\left(\frac{n}{n^{2-\epsilon}}\right) = \frac{1}{n^{1-\epsilon}}$$

- Los rangos de ϵ (que debe ser ϵ >0) dan:
  - 0> ϵ <1 limite de 1, y Verificar!
  - ϵ = 1 límite de 0 Significa que el denominador crece más rápido
- Por lo tanto si estamos en el Caso 1, entonces:

  T(n) = Θ($n^{\log_3 9}$) = Θ(n^2)

Master theorem

Teorema maestro

Ejemplo 2: Sea T(n) = T(2n/3) + 1

- Si a = 1, b = 3/2, f(n) = 1, g(n) = $n^{\log_{3/2} 1} = n^0 = 1$
- Verificando los límites con valores diferentes de ϵ y dividiendo entre n:

$$\lim_{x\to\infty}\left(\frac{f(n)}{g(n)}\right) \qquad \lim_{x\to\infty}\left(\frac{1}{1}\right) = 1$$

- Como ϵ no juega ningún rol estamos en el Caso 2 y entonces:

  T(n) = $\Theta(n^{\log_{3/2} 1} \log n) = \Theta(\log n)$

Master theorem

Teorema maestro

Ejemplo 3: Sea T(n) = 3T(n/4) + n lg n

- Si a = 3, b = 4, f(n) = n lg n, g(n) = $n^{\log_4 3} = n^{0.793}$
  and f(n) = $\Omega\left(n^{\log_4 3+\epsilon}\right)$.
- Donde si $\epsilon \approx 0.2$ aplica el Caso 3 si se puede demostrar la condición de regularidad:

T(n) = $\Theta(n \log n)$

$$\lim_{n\to\infty} \frac{f(n)}{n^{\log_b a+\epsilon}} = \lim_{n\to\infty} \frac{n \log n}{n^{0.793\ +\epsilon}}$$

Tiende a infinito?

Master theorem

Teorema maestro

$T(n) = a\,T(n/b) + n^c$ if $n>1$ (=d if $n=1$)

$\Theta(n^c)$

Caso 1: if $c < \log_b a$, then

$T(n) = \Theta(n^{\log_b a})$

Variante del TM
f(n)

Caso 2: if $c = \log_b a$, then

$T(n) = \Theta(n^c \log n)$

Caso 3: if $c > \log_b a$, then

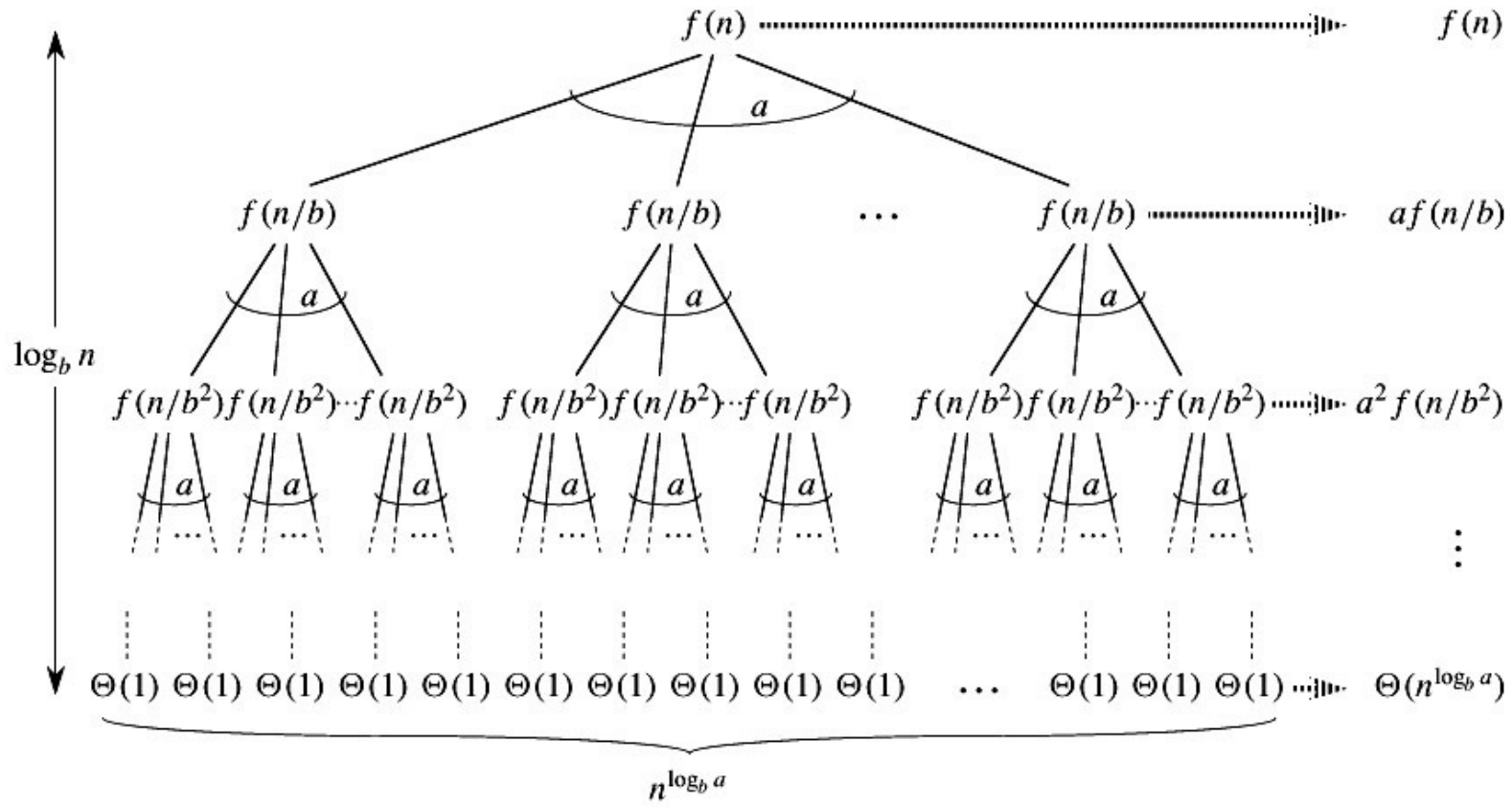
$T(n) = \Theta(n^c)$

Master theorem

Teorema maestro

Work done at root is f(n) and work done at all leaves is $\Theta(n^c)$.

Master method is mainly derived from **Recurrence Tree Method**:



Master theorem

Teorema maestro

In recurrence tree method, we calculate total work done:

- If the work done at leaves is polynomially more, then leaves are the dominant part, and our result becomes the work done at leaves.

  Case 1.
- If work done at root is asymptotically more, then our result becomes work done at root.

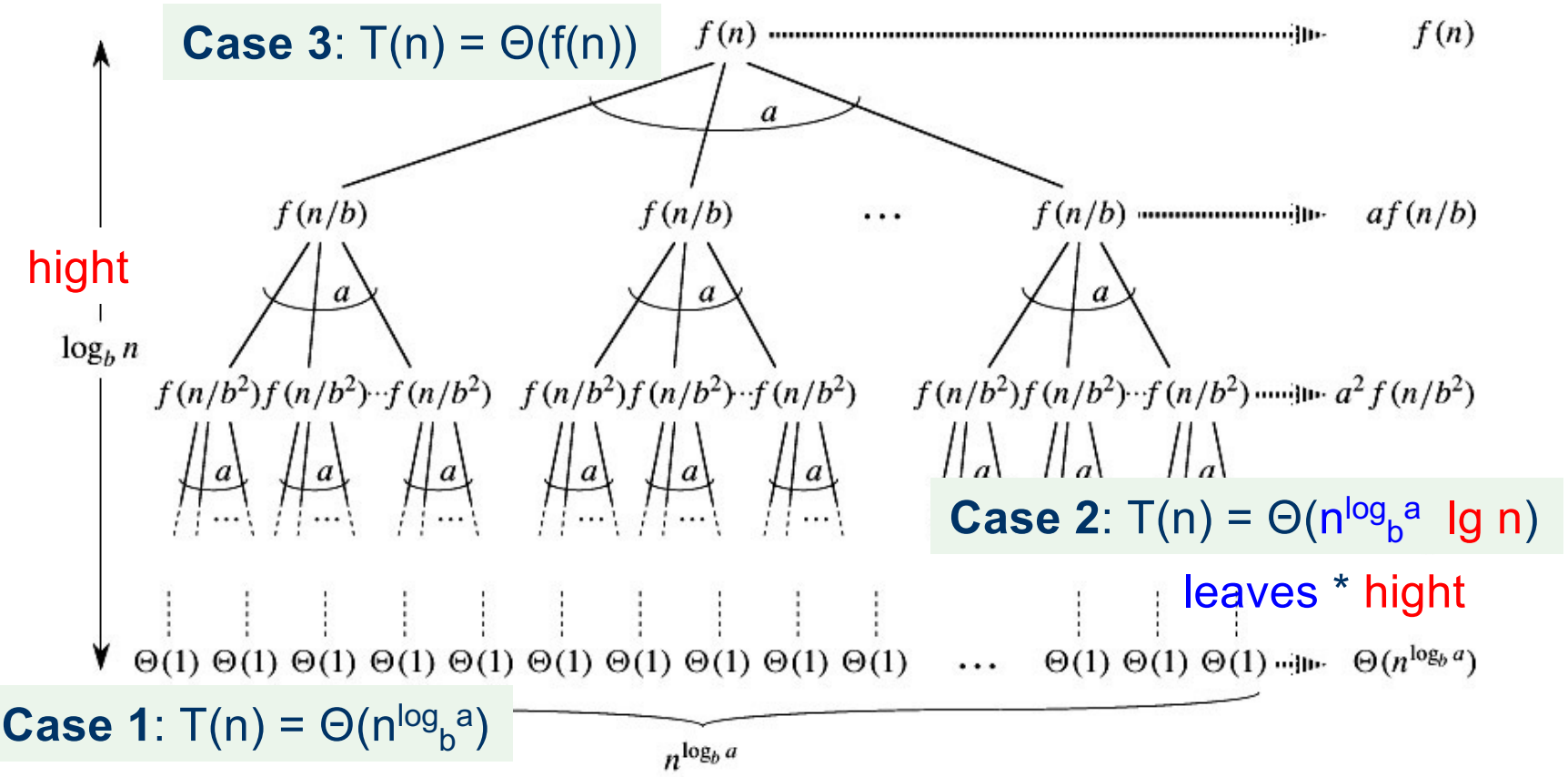
  Case 3.
- If work done at leaves and root is asymptotically same, then our result becomes height multiplied by work done at any level.

  Case 2.

Master theorem

Teorema maestro

Recurrence Tree Method



Master theorem

Teorema maestro

Extended ***Case 2***:

if $f(n) = \Theta(n^c)$ where $c = \log_b a$, then

$T(n) = \Theta(n^c \log n)$

Caso 4: if $f(n) = \Theta(n^c \log^k n)$ for $k >= 0$ and $c = \log_b a$, then

$T(n) = \Theta(n^c \log^{k+1} n)$.

Master theorem

Teorema maestro

$$T(n) = a\,T(n/b) + cn^k \quad n>b$$

Case 1: if $a > b^k$ then

$$T(n) = \Theta(n^{\log_b a})$$

2nd format easier

Case 2: if $a < b^k$ then

$$T(n) = \Theta(n^k)$$

Case 3: if $a = b^k$ then

$$T(n) = \Theta(n^k \log n)$$

It works even if we are using [n] instead of n.

Master theorem

Teorema maestro

Ahora veamos **ejemplos**:

- 5 para el formato sencillo y luego
- N para caso complicado.

Master theorem

Ejemplos del teorema maestro

Ejemplo 1: Sea $T(n) = 3T(n/2) + cn$

Polynomial Multiplication

Si $a = 3$, $b = 2$, $f(n) = cn$

Comparando a vs b^k tenemos $3>2^1$

Que cae en el Caso 1.

Entonces $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$

Master theorem

Ejemplos del teorema maestro

Ejemplo 2: Sea $T(n) = T(n/ (4\log_2 n)) + 2n \quad (n>1, T(1)=1)$

$4 \log_2 n > 4 \Rightarrow T(n/ (4\log_2 n)) <= T(n/4)$

$\Rightarrow T(n) <= T(n/4) + 2n$

Si $a = 1$, $b = 4$, $f(n) = 2n$, $k=1$

Comparando a vs b^k tenemos $1<4^1$

Que cae en el Caso 2.

Entonces $T(n) = \Theta(n^k) = \Theta(n^1) = \Theta(n)$

Master theorem

Teorema maestro

Ejemplo 3: Sea $T(n) = T(2n/3) + 1$

Si $a = 1$, $b = 3/2$, $f(n) = 1$, $k=0$

Comparando a vs b^k tenemos $1 = (3/2)^0 = 1$
Que cae en el Caso 3.

Entonces $T(n) = \Theta(n^k \log n) = \Theta(n^0 \log n) = \Theta(\log n)$

Master theorem

Teorema maestro

Ejemplo 4: Sea $T(n) = 7T(n/2) + cn^2$

Matrix Multiplication

Si $a = 7$, $b = 2$, $f(n) = cn$, $k=2$

Comparando a vs b^k tenemos $7 > 2^2$

Que cae en el Caso 1.

Entonces $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 7})$

Master theorem

Teorema maestro

Ejemplo 5: Sea T(n) = 9T(n/3) + n

Si a = 9, b = 3, f(n) = n, k=1

Comparando a vs b^k tenemos $9 > 3^1$

Que cae en el Caso 1.

$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_3 9}) = \Theta(n^2)$

Master theorem

Teorema maestro

Algoritmos basados en recorta-y-vencerás

- El problema principal de tamaño *n* se puede recortar a *b* subproblemas (*1<b< n*), cada uno de tamaño *n-c* (*1<c< n*), con costo no recursivo *f(n)* debido a la tarea de recortar y combinar.

$$T(n) = b\ T(n\text{-}c) + f(n)$$

- Si los subproblemas son de distinto tamaño se pueden obtener límites asintóticos que acotan el desempeño:
  - Cota superior expresada con:

    O(n): utilizando $T(n) \leq b\ T(n - cmax) + f(n)$
  - Cota inferior expresada con:

    Ω(n): utilizando $T(n) \geq b\ T(n - cmin) + f(n)$.

Master theorem

Teorema maestro

MacMahon Master Theorem (MMT)

- It is a result in enumerative combinatorics and linear algebra. It was discovered by Percy MacMahon.
- It is an area of combinatorics that deals with the number of ways that certain patterns can be formed.
- Two examples of this type of problem are:
  - Counting combinations, and
  - Counting permutations.

Master theorem

Teorema maestro

En general su aplicacion es “fácil” :

- Identificar que casos no aplica.
- Identificar cual de los 3 Casos.
- Identificar literales y aplicar la fórmula.

The end

Contacto

Raúl Acosta Bermejo

http:www.cic.ipn.mx
http://www.ciseg.cic.ipn.mx/

racostab@ipn.mx
racosta@cic.ipn.mx

57-29-60-00
Ext. 56652