CH-231-A

Algorithms and Data Structures

ADS

Lecture 10

Dr. Kinga Lipskoch

Spring 2020

Master Method (1)

The master method applies to recurrences of the form
$T(n) = aT(n/b) + f(n)$
where $a \geq 1, b > 1$, and f is asymptotically positive.

It distinguishes 3 common cases by comparing
$f(n)$ with $n^{log_b a}$

Master Method (2)

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

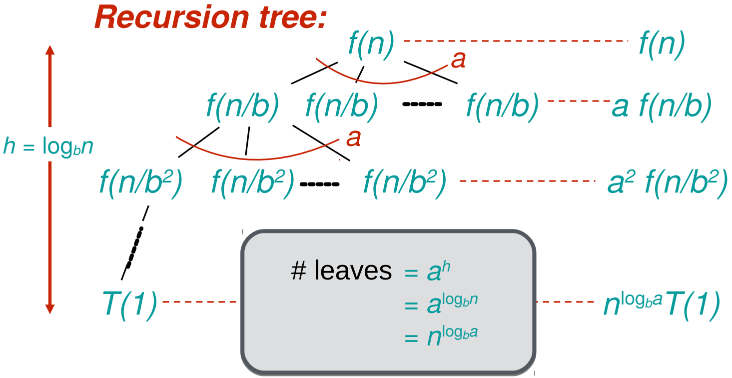
1. If $f(n) = O(n^{log_b a - \epsilon})$ for some constant $\epsilon > 0$
   – $f(n)$ is polynomially smaller than $n^{log_b a}$ –,
   then $T(n) = \Theta(n^{log_b a})$.
2. If $f(n) = \Theta(n^{log_b a})$,
   then $T(n) = \Theta(n^{log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{log_b a + \epsilon})$ for some constant $\epsilon > 0$
   – $f(n)$ is polynomially larger than $n^{log_b a}$ –
   and $af(n/b) \leq cf(n)$ – regularity condition – for some constant $c < 1$,
   then $T(n) = \Theta(f(n))$.

Master Method Not Always Applicable

- There is a gap between cases 1 and 2 when $f(n)$ is smaller than $n^{log_b a}$ but not polynomially smaller
- There is a gap between cases 2 and 3 when $f(n)$ is larger than $n^{log_b a}$ but not polynomially larger
- If the regularity condition in case 3 fails to hold or you are in one of the gaps then you cannot use the master method to solve the recurrence

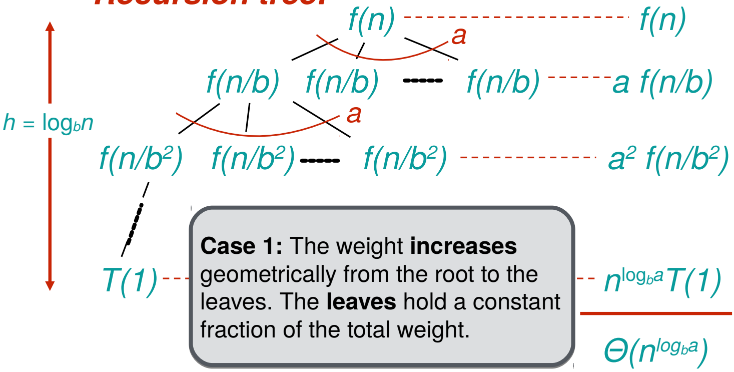
Idea of the Master Theorem (1)

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



Idea of the Master Theorem (2)

Recursion tree:

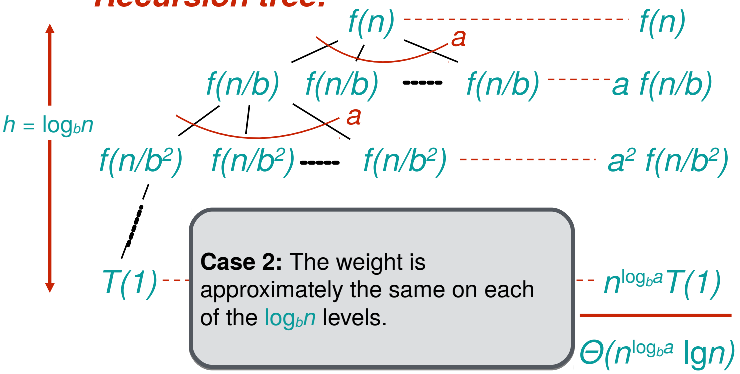


Case 1: The weight **increases** geometrically from the root to the leaves. The **leaves** hold a constant fraction of the total weight.

1

Idea of the Master Theorem (3)

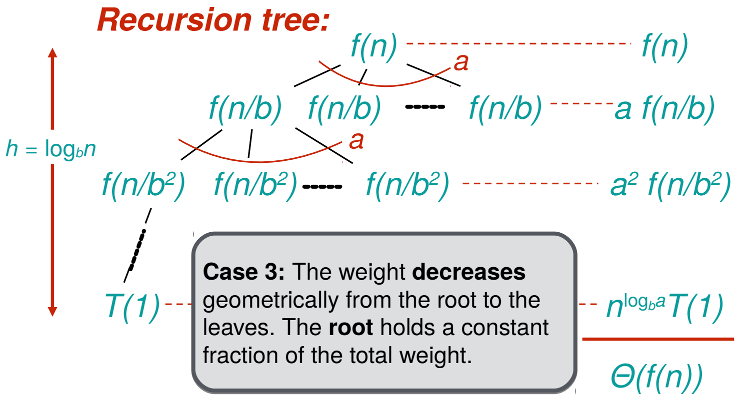
Recursion tree:



Case 2: The weight is approximately the same on each of the $\log_b n$ levels.

1

Idea of the Master Theorem (4)



Case 3: The weight **decreases** geometrically from the root to the leaves. The **root** holds a constant fraction of the total weight.

1

Example (1)

$T(n) = 4T(n/2) + n$
$a = 4, b = 2$

$n^{log_b a} = n^2$
$f(n) = n$

Case 1: $f(n) = O(n^{2-\epsilon})$ for $\epsilon = 1$
Thus, $T(n) = \Theta(n^2)$.

Example (2)

$$T(n) = 4T(n/2) + n^2$$
$$a = 4, b = 2$$

$$n^{log_b a} = n^2$$
$$f(n) = n^2$$

Case 2: $f(n) = \Theta(n^2)$,
Thus, $T(n) = \Theta(n^2 \lg n)$.

Example (3)

$T(n) = 4T(n/2) + n^3$
$a = 4, b = 2$

$n^{log_b a} = n^2$
$f(n) = n^3$

Case 3: $f(n) = \Omega(n^{2+\epsilon})$ for $\epsilon = 1$
and $4(n/2)^3 \leq cn^3$ for $c = 1/2$ (regularity condition)
Thus, $T(n) = \Theta(n^3)$.

Example (4)

$T(n) = 4T(n/2) + n^2/\lg n$
$a = 4, b = 2$

$n^{log_b a} = n^2$
$f(n) = n^2/\lg n$

Master method does not apply
(for every constant $\epsilon > 0$, we have $n^\epsilon = \omega(\lg n)$)

Recall: Divide & Conquer

Design paradigm:

1. Divide the problem (instance) into subproblems.
2. Conquer the subproblems by solving them recursively.
3. Combine subproblem solutions.

Recall: Merge Sort

1. Divide: Trivial
2. Conquer: Recursively sort 2 subarrays
3. Combine: Linear-time merge

$$T(n) = 2T(n/2) + \Theta(n)$$

#subproblems

subproblem size

work dividing
and combining

1

Master Method on Merge Sort

$T(n) = 2T(n/2) + n$
$a = 2,\ b = 2$

$n^{log_b a} = n$
$f(n) = n$

Case 2:
$f(n) = \Theta(n)$,
Thus, $T(n) = \Theta(n \lg n)$.

Power of a Number

- Problem:
  - Input: numbers $a \in \mathbb{R}$ and $n \in \mathbb{N}$.
  - Output: a^n
- Naive algorithm:
  - $T(n) = \Theta(n)$
- Divide & Conquer:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

- Recurrence:
  - $T(n) = T(n/2) + \Theta(1)$
- Solution:
  - $a = 1, b = 2, n^{log_b a} = 1, f(n) = \Theta(1) \Longrightarrow$ Case 2
  - Thus, $T(n) = \Theta(\lg n)$