Master Theorem

The **master theorem** provides a solution to recurrence relations of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

for constants $a \geq 1$ and $b > 1$ with f asymptotically positive. Such recurrences occur frequently in the runtime analysis of
commonly encountered algorithms.

Contents

Introduction

Many algorithms have a runtime of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

where n is the size of the input and $a\,(\geq 1)$ and $b\,(> 1)$ are constants with f asymptotically positive. For instance, one ca
that runtime of the *merge sort* algorithm satisfies

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

Similarly, traversing a binary tree takes time

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1).$$

By comparing $\log_b a$ to the asymptotic behavior of $f(n)$, the **master theorem** provides a solution to many frequently seen recurrences.

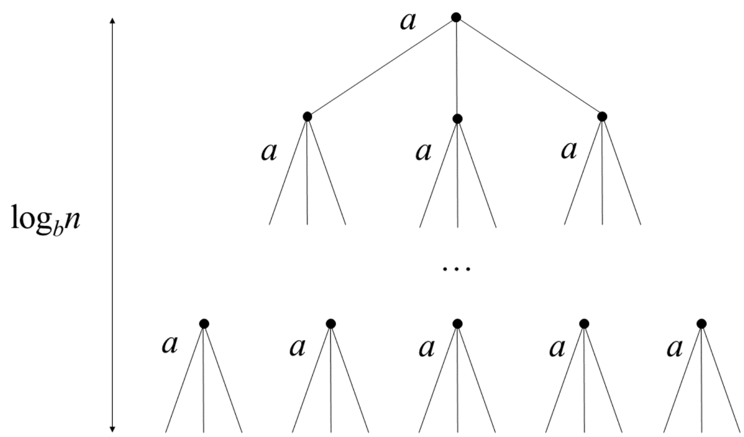
Statement of the Master Theorem

First, consider an algorithm with a recurrence of the form

$$T(n) = aT\left(\frac{n}{b}\right),$$

where a represents the number of children each node has, and the runtime of each of the three initial nodes is the runtime o
$T\left(\frac{n}{b}\right)$.

The tree has a depth of $\log_b n$ and depth i contains a^i nodes. So there are $a^{\log_b n} = n^{\log_b a}$ leaves, and hence the runtime
$\Theta\left(n^{\log_b a}\right)$.



Intuitively, the master theorem argues that if an asymptotically positive function f is added to the recurrence so that one ins
has

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

it is possible to determine the asymptotic form of T based on a relative comparison between f and $n^{\log_b a}$.

THEOREM

Master Theorem

Given a recurrence of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

for constants $a\,(\geq 1)$) and $b\,(> 1)$ with f asymptotically positive, the following statements are true:

- **Case 1.** If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.
- **Case 2.** If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a} \log n\right)$.
- **Case 3.** If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some $\epsilon > 0$ $\big($and $af\left(\frac{n}{b}\right) \leq cf(n)$ for some $c < 1$ for all n sufficiently large$\big)$, th
$T(n) = \Theta\big(f(n)\big)$.

Simply put, if $f(n)$ is polynomially smaller than $n^{\log_b a}$, then $n^{\log_b a}$ dominates, and the runtime is $\Theta\left(n^{\log_b a}\right)$. If $f(n)$ is ins
polynomially larger than $n^{\log_b a}$, then $f(n)$ dominates, and the runtime is $\Theta\big(f(n)\big)$. Finally, if $f(n)$ and $n^{\log_b a}$ are asympto
the same, then $T(n) = \Theta\left(n^{\log_b a} \log n\right)$.

Note that the master theorem does not provide a solution for all f. In particular, if f is smaller or larger than $n^{\log_b a}$ by less t
polynomial factor, then none of the three cases are satisfied. For instance, consider the recurrence

$$T(n) = 3T\left(\frac{n}{3}\right) + n \log n.$$

In this case, $n^{\log_b a} = n$. While f is asymptotically larger than n, it is larger only by a logarithmic factor; it is not the case tha
$f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some $\epsilon > 0$. Therefore, the master theorem makes no claim about the solution to this recurrence.

Examples

As mentioned in the introduction, the mergesort algorithm has runtime

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

$n^{\log_b a} = n$ and $f(n) = n$, so case 2 of the master theorem gives $T(n) = \Theta\left(n^{\log_b a} \log n\right) = \Theta(n \log n)$.

Similarly, as mentioned before, traversing a binary tree takes time

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1).$$

$n^{\log_b a} = n$, which is asymptotically larger than a constant factor, so case 1 of the master theorem gives $T(n) = \Theta\left(n^{\log_b a}\right) = \Theta(n)$.

EXAMPLE

Consider the recurrence

$$T(n) = 9T\left(\frac{n}{3}\right) + n.$$

In this case, $n^{\log_b a} = n^2$ and $f(n) = n$. Since $f(n)$ is polynomially smaller than $n^{\log_b a}$, case 1 of the master theorem ir
that $T(n) = \Theta\left(n^2\right)$.

EXAMPLE

Consider the recurrence

$$T(n) = 27T\left(\frac{n}{3}\right) + n^3.$$

In this case, $n^{\log_b a} = n^3$ and $f(n) = n^3$. Since $f(n)$ is asymptotically the same as $n^{\log_b a}$, case 2 of the master theorer
implies that $T(n) = \Theta\left(n^3 \log n\right)$.

EXAMPLE

Consider the recurrence

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2.$$

In this case, $n^{\log_b a} = n$ and $f(n) = n^2$. Since $f(n)$ is asymptotically larger than $n^{\log_b a}$, case 3 of the master theorem a
us to check whether $af\left(\frac{n}{b}\right) \leq cf(n)$ for some $c < 1$ and all n sufficiently large. This is indeed the case, so
$T(n) = \Theta\left(f(n)\right) = \Theta\left(n^2\right)$.

EXAMPLE

Consider the recurrence

$$T(n) = 8T\left(\frac{n}{2}\right) + \frac{n^3}{\log n}.$$

In this case, $n^{\log_b a} = n^3$ and $f(n) = \frac{n^3}{\log n}$. $f(n)$ is smaller than $n^{\log_b a}$ but by less than a polynomial factor. Therefore, t
master theorem makes no claim about the solution to the recurrence.

See Also

- Merge Sort
- Binary Tree

References

[1] Cormen, T.H., *et al*. *Introduction to Algorithms*. MIT Press, 2009.

Cite as: Master Theorem. *Brilliant.org*. Retrieved 19:49, November 4, 2018, from https://brilliant.org/wiki/master-theorem/