Master theorem (analysis of algorithms)

In the analysis of algorithms, the **master theorem for divide-and-conquer recurrences** provides an asymptotic analysis (using Big O notation) for recurrence relations of types that occur in the analysis of many divide and conquer algorithms. The approach was first presented by Jon Bentley, Dorothea Haken, and James B. Saxe in 1980, where it was described as a "unifying method" for solving such recurrences.[1] The name "master theorem" was popularized by the widely used algorithms textbook *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein.

Not all recurrence relations can be solved with the use of this theorem; its generalizations include the Akra–Bazzi method.

Contents

Introduction

Consider a problem that can be solved using a recursive algorithm such as the following:

```
procedure p( input x of size n ):
  if n < some constant k:
    Solve x directly without recursion
  else:
    Create a subproblems of x, each having size n/b
    Call procedure p recursively on each subproblem
    Combine the results from the subproblems
```

The above algorithm divides the problem into a number of subproblems recursively, each subproblem being of size n/b. Its call tree has a node for each recursive call, with the children of that node being the other calls made from that call. The leaves of the tree are the base cases of the recursion, the subproblems (of size less than k) that do not recurse. The above example would have a child nodes at each non-leaf node. Each node does an amount of work that corresponds to the size of the sub problem n passed to that instance of the recursive call and given by $f(n)$. The total amount of work done by the entire algorithm is the sum of the work performed by all the nodes in the tree.

The runtime of an algorithm such as the 'p' above on an input of size 'n', usually denoted $T(n)$, can be expressed by the recurrence relation

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

where $f(n)$ is the time to create the subproblems and combine their results in the above procedure. This equation can be successively substituted into itself and expanded to obtain an expression for the total amount of work done.[2] The master theorem allows many recurrence relations of this form to be converted to Θ-notation directly, without doing an expansion of the recursive relation.

Generic form

The master theorem sometimes yields asymptotically tight bounds to some recurrences from divide and conquer algorithms that partition an input into smaller subproblems of equal sizes, solve the subproblems recursively, and then combine the subproblem solutions to give a solution to the original problem. The time for such an algorithm can be expressed by adding the work that they perform at the top level of their recursion (to divide the problems into subproblems and then combine the subproblem solutions) together with the time made in the recursive calls of the algorithm. If $T(n)$ denotes the total time for the algorithm on an input of size n, and $f(n)$ denotes the amount of time taken at the top level of the recurrence then the time can be expressed by a recurrence relation that takes the form:

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

Here n is the size of an input problem, a is the number of subproblems in the recursion, and b is the factor by which the subproblem size is reduced in each recursive call. The theorem below also assumes that, as a base case for the recurrence, $T(n) = \Theta(1)$ when n is less than some bound $\kappa > 0$, the smallest input size that will lead to a recursive call.

Recurrences of this form often satisfy one of the three following regimes, based on how the work to split/recombine the problem $f(x)$ relates to the *critical exponent* $c_{\text{crit}} = \log_b a$. (The table below uses standard big O notation.)

$$c_{\text{crit}} = \log_b a = \log(\#\text{subproblems}) / \log(\text{relative subproblem size})$$

Case	Description	Condition on $f(n)$ in relation to c_{crit}, i.e. $\log_b a$	Master Theorem bound	Notational examples
1	Work to split/recombine a problem is dwarfed by subproblems. i.e. the recursion tree is leaf-heavy	When $f(n) = O(n^c)$ where $c < c_{\text{crit}}$ (upper-bounded by a lesser exponent polynomial)	... then $T(n) = \Theta\left(n^{c_{\text{crit}}}\right)$ (The splitting term does not appear; the recursive tree structure dominates.)	If $b = a^2$ and $f(x) = O(n^{1/2-\epsilon})$, then $T(n) = \Theta(n^{1/2})$.
2	Work to split/recombine a problem is comparable to subproblems.	When $f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$ for any $k \geq 0$ (rangebound by the critical-exponent polynomial, times zero or more optional logs)	... then $T(n) = \Theta\left(n^{c_{\text{crit}}} \log^{k+1} n\right)$ (The bound is the splitting term, where the log is augmented by a single power.)	If $b = a^2$ and $f(x) = \Theta(n^{1/2})$, then $T(n) = \Theta(n^{1/2} \log n)$. If $b = a^2$ and $f(x) = \Theta(n^{1/2} \log n)$, then $T(n) = \Theta(n^{1/2} \log^2 n)$.
3	Work to split/recombine a problem dominates subproblems. i.e. the recursion tree is root-heavy.	When $f(n) = \Omega(n^c)$where $c > c_{\text{crit}}$ (lower-bounded by a greater-exponent polynomial)	... this doesn't necessarily yield anything. If it is furthermore known that $af\left(\frac{n}{b}\right) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n (often called the *regularity condition*) then the total is dominated by the splitting term $f(x)$: $T(n) = \Theta(f(n))$	If $b = a^2$ and $f(x) = \Omega(n^{1/2+\epsilon})$ and the regularity condition holds, then $T(n) = \Theta(f(n))$.

Examples

Case 1 example

$$T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$$

As one can see from the formula above:

$a = 8,\ b = 2,\ f(n) = 1000n^2$, so
$f(n) = O\left(n^c\right)$, where $c = 2$

Next, we see if we satisfy the case 1 condition:

$$\log_b a = \log_2 8 = 3 > c.$$

It follows from the first case of the master theorem that

$$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^3\right)$$

(indeed, the exact solution of the recurrence relation is $T(n) = 1001n^3 - 1000n^2$, assuming $T(1) = 1$).

Case 2 example

$T(n) = 2T\left(\frac{n}{2}\right) + 10n$

As we can see in the formula above the variables get the following values:

$a = 2,\ b = 2,\ c = 1,\ f(n) = 10n$
$f(n) = \Theta\left(n^c \log^k n\right)$ where $c = 1, k = 0$

Next, we see if we satisfy the case 2 condition:

$\log_b a = \log_2 2 = 1$, and therefore, $c = \log_b a$

So it follows from the second case of the master theorem:

$$T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right) = \Theta\left(n^1 \log^1 n\right) = \Theta\left(n \log n\right)$$

Thus the given recurrence relation $T(n)$ was in $\Theta(n \log n)$.

(This result is confirmed by the exact solution of the recurrence relation, which is $T(n) = n + 10n \log_2 n$, assuming $T(1) = 1$.)

Case 3 example

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

As we can see in the formula above the variables get the following values:

$a = 2,\ b = 2,\ f(n) = n^2$
$f(n) = \Omega\left(n^c\right)$, where $c = 2$

Next, we see if we satisfy the case 3 condition:

$\log_b a = \log_2 2 = 1$, and therefore, yes, $c > \log_b a$

The regularity condition also holds:

$$2\left(\frac{n^2}{4}\right) \leq kn^2, \text{ choosing } k = 1/2$$

So it follows from the third case of the master theorem:

$$T(n) = \Theta(f(n)) = \Theta\left(n^2\right).$$

Thus the given recurrence relation $T(n)$ was in $\Theta(n^2)$, that complies with the $f(n)$ of the original formula.

(This result is confirmed by the exact solution of the recurrence relation, which is $T(n) = 2n^2 - n$, assuming $T(1) = 1$.)

Inadmissible equations

The following equations cannot be solved using the master theorem:[3]

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

  a is not a constant; the number of subproblems should be fixed

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

  non-polynomial difference between f(n) and $n^{\log_b a}$ (see below)

- $T(n) = 0.5T\left(\frac{n}{2}\right) + n$

  $a < 1$ cannot have less than one sub problem

- $T(n) = 64T\left(\frac{n}{8}\right) - n^2 \log n$

  f(n), which is the combination time, is not positive

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

  case 3 but regularity violation.

In the second inadmissible example above, the difference between $f(n)$ and $n^{\log_b a}$ can be expressed with the ratio $\frac{f(n)}{n^{\log_b a}} = \frac{n/\log n}{n^{\log_2 2}} = \frac{n}{n \log n} = \frac{1}{\log n}$. It is clear that $\frac{1}{\log n} < n^{\epsilon}$ for any constant $\epsilon > 0$. Therefore, the difference is not polynomial and the Master Theorem does not apply.

Application to common algorithms

Algorithm	Recurrence relationship	Run time	Comment
Binary search	$T(n) = T\left(\frac{n}{2}\right) + O(1)$	$O(\log n)$	Apply Master theorem case $c = \log_b a$, where $a = 1, b = 2, c = 0, k = 0$[4]
Binary tree traversal	$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$	$O(n)$	Apply Master theorem case $c < \log_b a$ where $a = 2, b = 2, c = 0$[4]
Optimal sorted matrix search	$T(n) = 2T\left(\frac{n}{2}\right) + O(\log n)$	$O(n)$	Apply the Akra–Bazzi theorem for $p = 1$ and $g(u) = \log(u)$ to get $\Theta(2n - \log n)$
Merge sort	$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$	$O(n \log n)$	Apply Master theorem case $c = \log_b a$, where $a = 2, b = 2, c = 1, k = 0$

See also

- Akra–Bazzi method

- Asymptotic complexity

Notes

1. Bentley, Jon Louis; Haken, Dorothea; Saxe, James B. (September 1980), "A general method for solving divide-and-conquer recurrences", *ACM SIGACT News*, **12** (3): 36–44, doi:10.1145/1008861.1008865 (https://doi.org/10.1145%2F1008861.1008865)
2. Duke University, "Big-Oh for Recursive Functions: Recurrence Relations", http://www.cs.duke.edu/~ola/ap/recurrence.html
3. Massachusetts Institute of Technology (MIT), "Master Theorem: Practice Problems and Solutions", http://www.csail.mit.edu/~thies/6.046-web/master.pdf
4. Dartmouth College, http://www.math.dartmouth.edu/archive/m19w03/public_html/Section5-2.pdf

References

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw–Hill, 2001. ISBN 0-262-03293-7. Sections 4.3 (The master method) and 4.4 (Proof of the master theorem), pp. 73–90.
- Michael T. Goodrich and Roberto Tamassia. *Algorithm Design: Foundation, Analysis, and Internet Examples*. Wiley, 2002. ISBN 0-471-38365-1. The master theorem (including the version of Case 2 included here, which is stronger than the one from CLRS) is on pp. 268–270.