Master theorem

In the analysis of algorithms, the **master theorem** provides a cookbook solution in asymptotic terms (using Big O notation) for recurrence relations of types that occur in the analysis of many divide and conquer algorithms. It was popularized by the canonical algorithms textbook *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein, in which it is both introduced and proved. Nevertheless, not all recurrence relations can be solved with the use of the master theorem; its generalizations include the Akra–Bazzi method.

Introduction

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

```
procedure T( n : size of problem ) defined as:
  if n < 1 then exit

  Do work of amount f(n)

  T(n/b)
  T(n/b)
  ...repeat for a total of a times...
  T(n/b)
end procedure
```

In the above algorithm we are dividing the problem into a number of subproblems recursively, each subproblem being of size *n/b*. This can be visualized as building a call tree with each node of the tree as an instance of one recursive call and its child nodes being instances of subsequent calls. In the above example, each node would have *a* number of child nodes. 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 tree is the sum of the work performed by all the nodes in the tree.

Algorithms such as above can be represented as a recurrence relation $T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$. This recursive relation can be successively substituted into itself and expanded to obtain expression for total amount of work done.[1]
The original Master theorem allows to easily calculate the running time of such a recursive algorithm in Θ-notation without doing expansion of above recursive relation.

Generic form

The master theorem concerns recurrence relations of the form:

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{where } a \geq 1,\, b > 1$$

In the application to the analysis of a recursive algorithm, the constants and function take on the following significance:

- *n* is the size of the problem.
- *a* is the number of subproblems in the recursion.
- *n/b* is the size of each subproblem. (Here it is assumed that all subproblems are essentially the same size.)
- *f* (*n*) is the cost of the work done outside the recursive calls, which includes the cost of dividing the problem and the cost of merging the solutions to the subproblems.

It is possible to determine an asymptotic tight bound in these three cases:

Case 1

Generic form

If $f(n) = O\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$(using Big O notation)
it follows that:

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

Example

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

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

$$a = 8,\ b = 2,\ f(n) = 1000n^2,\ \log_b a = \log_2 8 = 3$$

Now we have to check that the following equation holds:

$$f(n) = O\left(n^{\log_b a-\epsilon}\right)$$
$$1000n^2 = O\left(n^{3-\epsilon}\right)$$

If we choose ϵ = 1, we get:

$$1000n^2 = O\left(n^{3-1}\right) = O\left(n^2\right)$$

Since this equation holds, the first case of the master theorem applies to the given recurrence relation, thus resulting in the conclusion:

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

If we insert the values from above, we finally get:

$$T(n) = \Theta\left(n^3\right)$$

Thus the given recurrence relation $T(n)$ was in $\Theta(n^3)$.

(This result is confirmed by the exact solution of the recurrence relation, which is $T(n) = 1001n^3 - 1000n^2$, assuming $T(1) = 1$).

Case 2

Generic form

If it is true, for some constant $k \geq 0$, that:

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

it follows that:

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

Example

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

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

$$a = 2,\ b = 2,\ k = 0,\ f(n) = 10n,\ \log_b a = \log_2 2 = 1$$

Now we have to check that the following equation holds (in this case k=0):

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

If we insert the values from above, we get:

$$10n = \Theta\left(n^1\right) = \Theta\left(n\right)$$

Since this equation holds, the second case of the master theorem applies to the given recurrence relation, thus resulting in the conclusion:

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

If we insert the values from above, we finally get:

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

Thus the given recurrence relation *T*(*n*) was in Θ(*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

Generic form

If it is true that:

$f(n) = \Omega\left(n^{\log_b(a)+\epsilon}\right)$ for some constant $\epsilon > 0$

and if it is also true that:

$af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and sufficiently large *n*

it follows that:

$$T\left(n\right) = \Theta\left(f\left(n\right)\right)$$

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:

$$a = 2,\ b = 2,\ f(n) = n^2,\ \log_b a = \log_2 2 = 1$$

Now we have to check that the following equation holds:

$$f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$$

If we insert the values from above, and choose $\epsilon = 1$, we get:

$$n^2 = \Omega\left(n^{1+1}\right) = \Omega\left(n^2\right)$$

Since this equation holds, we have to check the second condition, namely if it is true that:

$$af\left(\frac{n}{b}\right) \leq cf(n)$$

If we insert once more the values from above, we get the number :

$$2\left(\frac{n}{2}\right)^2 \leq cn^2 \Leftrightarrow \frac{1}{2}n^2 \leq cn^2$$

If we choose $c = \frac{1}{2}$, it is true that:

$$\frac{1}{2}n^2 \leq \frac{1}{2}n^2 \ \forall n \geq 1$$

So it follows:

$$T(n) = \Theta(f(n)).$$

If we insert once more the necessary values, we get:

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

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:[2]

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

  a is not a constant
- $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

  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) 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 $\frac{f(n)}{n^{\log_b a}} = \frac{\frac{n}{\log n}}{n^{log_2 2}} = \frac{n}{n \log n} = \frac{1}{\log n}$. 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 where $f(n) = n^c$ [3]
Binary tree traversal	$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$	$O(n)$	Apply Master theorem where $f(n) = n^c$ [3]
Optimal Sorted Matrix Search	$T(n) = 2T\left(\frac{n}{2}\right) + O(\log(n))$	$O(n)$	Apply Akra-Bazzi theorem for $p = 1$ and $g(u) = \log(u)$ to get $\Theta(2n - \log(n))$
Merge Sort	$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$	$O(n \log(n))$	

Notes

[1] Duke University, "Big-Oh for Recursive Functions: Recurrence Relations", http://www.cs.duke.edu/~ola/ap/recurrence.html

[2] Massachusetts Institute of Technology (MIT), "Master Theorem: Practice Problems and Solutions", http://www.csail.mit.edu/~thies/6.046-web/master.pdf

[3] 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.

Article Sources and Contributors

Master theorem *Source*: http://en.wikipedia.org/w/index.php?oldid=484266272 *Contributors*: Alberto da Calvairate, AySz88, BeteNoir, Charles Matthews, Chris-gore, Cybercobra, David Eppstein, Dcoetzee, Dead3y3, Diegusjaimes, Dionyziz, Eric119, FauxFaux, Gavinbeatty, Geometry guy, Giftlite, Hairy Dude, Hariva, Igorpak, Jleedev, Jogloran, Justin W Smith, Kainaw, Kaisenl, Karl Dickman, Kenyon, Krappie, LOL, Magioladitis, Mathiasl26, Michael Hardy, Mikechen, Mobius, Mpercy, Njsg, Oliphaunt, Qutezuce, Ratfox, Regnaron, Rspeer, Seftembr, Sligocki, Snowolf, Svick, Sytelus, Tachikoma's All Memory, TakuyaMurata, Tbhotch, TedPavlic, Unreal128, Vedant, Wazow, WillNess, Xvani, Yuval madar, Zahd, Zeroin23a, 133 anonymous edits

License