Notes on the Master Theorem

These notes refer to the Master Theorem as presented in Sections 4.3 and 4.4 of

- [CLR] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms* MIT Press/ McGraw-Hill, 1990

and of

- [CLRS] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein, *Introduction to Algorithms (Second Edition)* MIT Press/ McGraw-Hill, 2001.

1 The recurrence and the recursion tree

A hypothetical divide-and-conquer algorithm divides a problem of size n (greater than 1) into a subproblems of size n/b, solves these subproblems recursively, and then combines their solutions into a solution of the original problem; problems of size 1 are solved directly, without any recursive calls. Naturally, a is a positive integer; in order for the size n/b of the subproblems, the size n/b^2 of the sub-subproblems, etc. to become smaller and smaller, we assume that $b > 1$; in order for these sizes to remain integral, we assume that b is an integer and that

$$n = b^m$$

for some nonnegative integer m. With $f(b^k)$ standing for the amount of work required first to divide a problem of size b^k into a subproblems of size b^{k-1} and later to combine solutions of these subproblems into a solution of the problem of size b^k, the total amount $T(n)$ of work required to solve the original problem satisfies the recurrence equation

$$T(b^k) = aT(b^{k-1}) + f(b^k)$$

whenever $k > 0$.

In the corresponding *recursion tree*, the root represents the original problem of size n, its children represent the subproblems, their children represent the sub-sub problems, etc. Each internal node has precisely a children, and so there are precisely a^j nodes on level j (the root being on level 0, its children on level 1, their children on level 2, etc.); each of these a^j nodes represents a problem of size b^{m-j}; in particular, each of the a^m nodes on level m represents a problem of size 1, and so it is a leaf of the tree. If $0 \leq j < m$, then the amount of work performed at a particular node on level j (not counting any of the work involved in the a recursive calls spawned from this node) is $f(b^{m-j})$; the amount of work performed at a particular leaf is $T(1)$; it follows that

$$T(n) = \sum_{j=0}^{m-1} a^j f(b^{m-j}) \; + \; a^m T(1). \tag{1}$$

2 Very special cases of the Master Theorem

2.1 Starting point.

Formula (1) is particularly easy to simplify when the total work done on a level of the recursion tree is independent of the depth of the level:

$$f(n) \; = \; af(b^{m-1}) \; = \; a^2 f(b^{m-2}) \; = \; \ldots \; = \; a^{m-1} f(b) \; = \; a^m T(1).$$

This is the case if and only if

$$f(b^k) \; = \; a^k T(1) \quad \text{for all } k = 1, 2, \ldots, m; \tag{2}$$

under this assumption, $T(n) = (m+1)f(n)$; now, since

$$m = \frac{\lg n}{\lg b},$$

it follows that

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

For future reference, note that the assumption (2) may be recorded as

$$f(x) \; = \; T(1)x^t \quad \text{for all } x = b, b^2, \ldots, n \tag{3}$$

with

$$t = \frac{\lg a}{\lg b}.$$

2.2 Continuation.

Next, we are going to consider the more general class of functions defined by

$$f(x) \ = \ dx^s$$

with arbitrary positive constants d and s. Here,

$$T(n) = \sum_{j=0}^{m-1} a^j \cdot d(n/b^j)^s \ + \ a^m T(1) = dn^s \sum_{j=0}^{m-1} (a/b^s)^j \ + \ T(1)n^t,$$

and so, as long as $a/b^s \neq 1$ (which is equivalent to $s \neq t$),

$$T(n) = dn^s \frac{(a/b^s)^m - 1}{(a/b^s) - 1} \ + \ T(1)n^t = \frac{db^s}{a - b^s}(n^t - n^s) \ + \ T(1)n^t. \tag{4}$$

If $s < t$, then $f(n)$ grows more slowly than the benchmark (3), and so the higher levels of the recursion tree contribute relatively less to the total work $T(n)$ and the lower levels contribute relatively more. In fact, the amount of work done in the leaves alone is representative of the grand total $T(n)$: since $a - b^s > 0$, formula (4) implies that $T(n) = \Theta(n^t)$.

If $s > t$, then $f(n)$ grows faster than the benchmark (3), and so the lower levels of the recursion tree contribute relatively less to the total work $T(n)$ and the higher levels contribute relatively more. In fact, the amount of work done in the root alone is representative of the grand total $T(n)$: since $a - b^s < 0$, formula (4) implies that $T(n) = \Theta(n^s) = \Theta(f(n))$.

2.3 Conclusion.

Theorem 1 *Let a be a positive integer, let b be an integer greater than 1, and let d and s be positive real numbers. For all perfect powers n of b, define $T(n)$ by the recurrence*

$$T(n) = aT(n/b) + dn^s$$

with a nonnegative initial value $T(1)$; write

$$t = \frac{\lg a}{\lg b}.$$

- *If $s < t$, then $T(n) = \Theta(n^t)$.*
- *If $s = t$, then $T(n) = \Theta(f(n) \log n)$.*
- *If $s > t$, then $T(n) = \Theta(f(n))$.*

3 Less special cases of the Master Theorem

Theorem 1 generalizes as follows:

Theorem 2 *Let a be a positive integer, let b be an integer greater than 1, and let f be a real-valued function defined on perfect powers of b. For all perfect powers n of b, define $T(n)$ by the recurrence*

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

with a nonnegative initial value $T(1)$; write

$$t = \frac{\lg a}{\lg b}.$$

- *If $f(n) = O(n^s)$ with $s < t$, then $T(n) = \Theta(n^t)$.*
- *If $f(n) = \Theta(n^t)$, then $T(n) = \Theta(f(n) \log n)$.*
- *If $f(n) = \Omega(n^s)$ with $s > t$ and if*

$$\exists c\ (\, c < 1 \wedge (\exists k_0 \forall k\ (\, k \geq k_0 \Rightarrow af(b^{k-1}) \leq cf(b^k)\,))\,), \tag{5}$$

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

In response to a challenge that I proposed in class, Antonyi Ganchev constructed an example showing that the conclusion of the third case of Theorem 2 may be false if the "regularity condition" (5) is dropped. Here is his example with a slight modification: If

$$f(n) = \begin{cases} n^3 & \text{when } \lg n \text{ is an even integer,} \\ n^2 & \text{otherwise,} \end{cases}$$

and

$$T(n) = \begin{cases} (16n^3 + 10n^2 - 11n)/15 & \text{when } \lg n \text{ is an even integer,} \\ (4n^3 + 20n^2 - 11n)/15 & \text{when } \lg n \text{ is an odd integer,} \end{cases}$$

then

$$T(n) = 2T(n/2) + f(n)$$

whenever n is a power of 2 greater than 1.