The Master Theorem

Remington Greko, Tyler Gutowski, and Spencer Hirsch

March 6, 2023

Work with your team to write a report about the *Master Theorem* for solving a recurrence.

I've usually avoided the *Master Theorem* because it has too many rules and does not offer real insight to the solution of the recurrene. However, I do include it in my handouts and the *Master Therorem* may be useful for find the solution to *Strassen's matrix multiplication recurrence.*

$$T(n) = 7T(n/2) + O(n^2)$$

There are many details beneath this recurrence. I strongly encourage you to read what is in the Corman textbook and other sources.

Submit the team's report on Canvas. Include a task matric indicating who did what.

Wikipedia Summary of the Master Theorem

In 1980, the Master Theorem was proposed as the "unifying method" for solving recurrences by Jon Bentley, Dorothea Blostein, and James B. Saxe.[1] Although the name may imply it can solve all recurrences, this is not the case, it's a generalized theory that may be helpful in solving recurrence relations. The Master Theorem utilizes a divide and conquer approach with allows it to separate processing into numerous parts. The master theorem can be expressed by adding the time taken by the top level process with the time made with the recursive calls of the algorithm.[2] The algorithm for the recurrence relation is expressed as:[3]

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

Where, n is the input size, a is the number of subproblems, and b is the factor by which the size is reduced.[4] It can be determined based on the Theorem the best case time complexity of the Master Theorem is constant time or *O(1)*.

Brilliant Article on the Master Theorem

Statement of the Master Theorem

Given the algorithm listed above, the runtime of each of the initial nodes is the runtime of T(n/b). Therefore the depth of the tree is $log_b n$ with the depth i has a^i nodes in that final level.[5] Therefore,

$$a^{log_b n} = n^{log_b n}$$

which demonstrates the the runtime of the program is $O(n^{log_b a})$.[6] Through this reasoning you can determine that the form of T is based between f and $n^{log_b a}$.[7]

[1] Wikipedia.
[2] Id.
[3] Id.
[4] Id.
[5] Brilliant
[6] Id.
[7] Id.

The Master Theorem Itself

Given the previous form listed in the Wikipedia section, with constants $a \geq 1$ and $b > 1$ with f.[8]
The follwing are true:

"$f(n) = O(n^{log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = O(n^{log_b a})$.
$f(n) = O(n^{log_b a})$, then $T(n) = O(n^{log_b a} logn)$.
$f(n) = O(n^{log_b a + \epsilon})$ for some $\epsilon > 0$, then *T(n) = O(f(n)).*" [9]

Cormen Discussion of the Master Theorem

Cormen describes the master theorem as a "cookbook" method for solving recurrences which are in the form $T(n) = aT(n/b) + f(n)$. This equation represents the running time of an algorithm which is divided into n subproblems, sized n/b, where n and b are positive constants. Each subproblem a is solved recursived in time $T(n/b)$ and the cost of dividing the problem is described by $f(n)$.

The Master Theorem

The master theorem itself is as follows:

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$. Then $T(n)$ can be bounded asymptotically as follows.

1. If $f(n) = O(n^{log_b a - \in})$ for some constant $\in > 0$, 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 $f(n) = \Omega(n^{log_b a + \in})$ for some constant $\in > 0$, and if $af(n/b)/lecf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Essentially this expresses the solution to a recurrence through comparing the function $f(n)$ with the function $n^{log_b a}$. Whichever of the two functions is larger is the solution. The first case requires that the expression $f(n)$ must be *polynomially* smaller than $n^{log_b a}$. This is the same in case three where $f(n)$ must be polynomially *larger* than $n^{log_b a}$. The three cases do not cover all possibilites for $f(n)$, there is a gap between cases 1 and 2 when $f(n)$ is smaller than $n^{log_b a}$ but not polynomially. This occurs again between 2 and 3 where $f(n)$ is not polynomially larger. In the cases where $(f(n))$ falls between these cases, the master theorem cannot be used to solve the recurrence.

[8] Id.
[9] Id.

Master Theorem Example

To use the master theorem we first determine which case applies to the given equation. For example consider:

$T(n) = 9T(n/3) + n$

In this recurrence we have $a = 9, b = 3, f(n) = n$, and thus $n^{log_b a} = n^{log_3 9} = \Theta(n^2)$. Since $f(n) = O(n^{log_3 9 - \epsilon})$, where $\epsilon = 1$, we can apply case 1 of the master theorem. This produces the solution $T(n) = \Theta(n^2)$

The binary search algorithm is a simple and efficient algorithm that allows a user to find a target value in a sorted array of values. How can we use the master algorithm to determine the time complexity of binary search?
The recurrence relation is expressed as:

$T(n) = T(n/2) + O(1)$

where n is the size of the array. $O(1)$ represents the time it takes to compare a value with the midpoint of an array.

By using master algorithm, we can identify a, b, and $f(n)$:

1. $a = 1$, since there is only one subproblem at each recurrence level. (Search left or right of the array.)
2. $b = 2$, since the size of the subproblem is halves at each recurrence level.
3. $f(n) = O(1)$, since the time taken to compare is constant.

Now we compare $f(n)$ with $nlogb(a) = nlog2(1) = 0$
Since $f(n)$ is $O(1)$, and is smaller than $nlog2(1)$, we can apply case 2 of the master theorem.
In this case, $O(n^{log}(b(a)) * log^{(k+1)(n))} = O(log(n))$

Therefore, the time complexity of binary search is $O(log(n))$, which confirms that binary search is a very efficient algorithm.

1 Works Cited

"Master Theorem (Analysis of Algorithms)." Wikipedia. Wikimedia Foundation, January 31, 2023. $https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)$.

"Master Theorem." Brilliant Math & Science Wiki. Accessed February 28, 2023. $https://brilliant.org/wiki/master$ $theorem/$.

Name	**Section**
Remington Greko	Cormen Summary
Tyler Gutowski	Usage with Binary Search
Spencer Hirsch	Read Wikipedia Article and Brilliant Article