# The cost of trying too hard: String searching

devblogs.microsoft.com/oldnewthing/20060119-11

January 19, 2006

Raymond Chen

There are many algorithms for fast string searching, but the running of a string search is inherently $O(n)$, where $n$ is the length of the string being searched: If $m$ is the length of the string being searched for (which I will call the "target string"), then any algorithm that accesses fewer than $n/m$ elements of the string being searched will have a gap of $m$ unaccessed elements, which is enough room to hide the target string. More advanced string searching algorithms can take advantage of characteristics of the target string, but in the general case, where the target string is of moderate size and is not pathological, all that the fancy search algorithms give you over the naive search algorithm is a somewhat smaller multiplicative constant. In the overwhelming majority of cases, then, a naive search algorithm is adequate. As long as you're searching for normal strings and not edge cases like "Find `aaaaaaaaaaaaaaab` in the string `aaaaaaaaaaaaaaabaaaaaaaaaaaaaaab`". If you have a self-similar target string, the running time of a naive search is $O(mn)$ where $m$ is the length of the target string. The effort in the advanced searching algorithms goes towards diminishing the effect of $m$, but pay for it by requiring preliminary analysis of the target string. If your searches are for "relatively short" "normal" target strings, then the benefit of this analysis doesn't merit the cost.

That's why nearly all library functions that do string searching use the naive algorithm. The naive algorithm is the correct algorithm over 99% of the time.

Raymond Chen

**Follow**