CS 2742 (Logic in Computer Science) – Fall 2008
Lecture 28

Antonina Kolokolova

November 28, 2008

8.1 Grows of functions

What do we mean when we say that one algorithm. is faster than another? In computer science, the standard definition of one algorithm being faster than the other is that on *large enough* inputs the faster algorithm makes less operations *in the worst case*. What it means to say "in the worst case" is that out of all executions of algorithm for an input of a give size, we look at the longest execution time. For example, if we are searching for an element in the array, it is always possible that we get lucky and hit this element on the very first step. However, it is more realistic to consider the running time of this algorithm in the case when the element is the last one we look at, or not in the array at all: we know then that the algorithm can't do worse than that, this is a guarantee on its running time.

Here we will review the growth rate of some functions. Think of these functions as describing the worst-case running time of algorithms, so a function with a larger increase in its value will correspond to a slower algorithm.

As many of you know by now, $f \in O(g)$ if "f is at least as fast as g" is defined as follows: $\exists n_0, c\ foralln > n_0 |f(n)| \leq c|g(n)|$

In particular, a $f(n) = \log n$ function grows slower (=better algorithm) than linear $f(n) = n$ (so $\log n \in O(n)$), which is in turn is better than quadratic $f(n) = n^2$, and all of them are much better than the exponential $f(n) = 2^n$. As an example, a binary search has logarithmic time complexity, searching in an array is linear, mergesort is $O(n \log n)$ which is better than the bubble sort with time $O(n^2)$.

Example 1. Here is an example of proving that one function is in $O()$ of another. We will show that $3n \in O(2^n)$. Set $n_o = 2$ and $c = 3$. Now $\forall n > 3$, $3n < 3 * 2^n$.

But there are functions, recursively defined, that grow so fast that we can't even describe

their growth. An example of such a function is the Ackermann's function, recursively defined as follows:

$$\forall m, n > 0, A(0, n) = n + 1, A(m, 0) = A(m - 1, 1), A(m, n) = A(m - 1, A(m, n - 1))$$

This function is well-defined, but $A(x, x)$ grows very fast: $A(4, 4)$ is already $2^{2^{2^{655536}}}$.

There are some recursive function that is not well-defined: $G(n) = \begin{cases} 1 & \text{if } n = 1 \\ 1 + G(n/2) & \text{if } n \text{ is even} \\ G(3n + 1) & \text{if } n > 1 \text{ and is odd} \end{cases}$

Not well-defined because $G(5) = G(14) = 1 + G(7) = 1 + G(20) = 1 + (1 + G(10)) = 1 + 1 + 1 + G(5) = 3 + G(5)$. So $G(5) = 3 + G(5)$, subtracting $G(5)$ from both sides get $0 = 3$!

The "3n+1" problem is: $T(n) = \begin{cases} n/2 & \text{if n is even} \\ 3n + 1 & \text{if n is odd} \end{cases}$ It is still open if it eventually produces 1 for any starting number n.

9 Correctness of algorithms

In the next lecture notes, we will show how to use logic to analyze algorithm correctness rather than the running time (since we started the next lecture by repeating most of the material from the end of this one, I will skip putting it in this set of notes).