CSE 347 Analysis of Algorithms August 28, 2017

Homework 0

Instructor: Jeremy Buhler/Brendan Juba *Due: September 6, 2017, 11:59 PM*

Reminder: You may work in groups and use outside sources. But, you must write up solutions in your own words and properly reference your sources for each problem. This includes listing your collaborators and properly citing any sources you use. Solutions to each problem must be electronically typeset and submitted online via Blackboard. Instructions appear in the E-Homework Guide: `http://classes.engineering.wustl.edu/cse347/ehomework/`

1. (a) We will say that two positive integers, n_i and n_j, are *relatively prime* if the greatest common divisor of n_i and n_j is 1. Suppose that we are given a system of equations:

$$\begin{aligned} a_1 &= x \bmod n_1 \\ a_2 &= x \bmod n_2 \\ &\vdots \\ a_m &= x \bmod n_m \end{aligned}$$

where each n_i and n_j are relatively prime for $i \neq j$. For $N = \prod_{i=1}^{m} n_i$, give an efficient algorithm to compute a solution to this system of equations x that is between 0 and $N - 1$.

(Note: by "efficient algorithm," we mean one that runs in time that is asymptotically bounded by a polynomial function of the total number of digits appearing in $n_1, n_2, \ldots, n_m$. Here and everywhere else in the course, this means that you should (i) show that your algorithm is correct and (ii) show that it always finishes in such a bounded amount of time.)

(b) CPUs use registers of a bounded size (currently, 64 bits) to represent numbers. "Floating point" representation can represent somewhat larger numbers, but at the cost of many complications. For example, we cannot use floating point to reliably represent a counter, since eventually the effect of incrementing the counter gets "rounded off"—there are simply not enough bits in the floating point representation to represent large numbers to such high precision. Consider the following alternative scheme for computing with large numbers: we choose m large, prime numbers $n_1, \ldots, n_m$, and represent each number x using m registers, in which the ith register contains a_i, the remainder of x divided by n_i. Now we can do arithmetic "component-wise": to add, subtract, or multiply x and y, represented respectively by $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$, we simply perform the respective operation to each a_i and b_i, and compute the result modulo n_i. Furthermore, we can test whether or not the result is equal to any value z we wish (for example, zero) by pre-computing the sequence of remainders $(c_1, \ldots, c_m)$ for z and checking whether or not each ith component is equal to c_i. There is just one concern: how do we know that this is correct? Could it be that there are two different large numbers, w and z, such that $w = z \bmod n_i$ for all i?

Show that $x \in \{0, 1, \ldots, N-1\}$ **is uniquely determined by** $a_1 \in \{0, 1, \ldots, n_1 - 1\}$**,** $a_2 \in \{0, 1, \ldots, n_2 - 1\}$**,**$\ldots$**,** $a_m \in \{0, 1, \ldots, n_m - 1\}$**.** That is that for each such sequence of remainders, there is a unique x having those remainders.

Since it is furthermore clear that for each x (and fixed $n_1, n_2, \ldots, n_m$) there is a unique sequence of remainders, our strategy for representing large numbers using m registers represents each number in the range $\{0, 1, \ldots, N-1\}$ in a unique way and our proposed equality test will give the correct answer. This representation does not suffer from the many defects arising from the lack of precision in floating point representation.