CS 3510 - Design & Analysis of Algorithms

Co-Instructor : Prasad Tetali, office: Skiles 132, email: tetali@math.gatech.edu

Comments on some Exercises from Chapter 1, following Lecture on Feb. 7, 2018:

Problem 1.18. (It might help to know the following.)

Besides running the GCD algorithm on the given numbers, here is another way to compute the GCD of two numbers, **assuming one knows the prime factorizations of the two numbers**, which is indeed a big assumption!

First an example: Suppose $m = 2^3 \cdot 5^2 \cdot 11$ and $n = 2^2 \cdot 3^2 \cdot 5 \cdot 11^3$. Then it is easy to see that

$$GCD(m, n) = 2^2 \cdot 5 \cdot 11\,,$$

wherein we take each prime that appears in either m or n, and raise it to the smaller of the two powers to which it appears in the two numbers. Note that 3 does not appear in m, while it appears in n, so we take it as appearing to power 0 in m. And so on. The justification is that since the GCD has to divide both m and n, each prime appearing in the GCD can only appear to a power as high as in the minimum of the two appearances.

So more generally, suppose $m = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$ and $n = p_1^{b_1} p_2^{b_2} \cdots p_k^{b_k}$, where $a_i, b_i \geq 0$ – indeed, some of the a_i or b_i may be zero in m or n (bot not both). Then it is easy to see that we can write a formula:

$$GCD(m, n) = p_1^{\min\{a_1,b_1\}} p_2^{\min\{a_2,b_2\}} \cdots p_k^{\min\{a_k,b_k\}}\,.$$

Similarly, the *least common multiple* (LCM) would be:

$$LCM(m, n) = p_1^{\max\{a_1,b_1\}} p_2^{\max\{a_2,b_2\}} \cdots p_k^{\max\{a_k,b_k\}}\,.$$

Of course, the beauty of Euclid's GCD algorithm is that it finds efficiently in cubic time (in the number of bits of the larger of the two numbers involved), the GCD, without *knowing the prime-factorizations. Given that factoring is hard, the GCD algorithm is indeed very valuable!*

Problem 1.19. Recall that the basis of the GCD algorithm is that if $a = q \cdot b + r$, where $0 \leq r < b$, and $q \geq 1$, then $GCD(a, b) = GCD(b, r)$ and we repeat this. So ask yourself, what happens (at the end - what numbers you end up with) when you start with the consecutive Fibonacci numbers and execute the algorithm. What happens after one step, for example? Then you might see the pattern.

Optional Problem 1.35. Concerning, Wilson's theorem, which asserts $(N-1)! \equiv -1 \pmod N$ if and only if N is prime:

Since I brought this up in the class (and discussed Part (d)), let me answer the other parts here:

a) Let p be prime and $1 \leq x \leq p-1$. If x is its own inverse mod p, it means $x^2 \equiv 1 \pmod p$. This means $(x-1)(x+1) \equiv 0 \pmod p$, which in turns means p divides either $(x-1)$ or $(x+1)$ or both. So x is either 1 or $-1 \pmod p$; the latter is the same as $p-1$.

b) By Part(a), we can pair up every integer from 2 up to $p-2$ with its inverse that is distinct from itself. Since their product equals 1. when we consider the product, $(p-1)!$ taken (mod p), only 1 and $p-1$ remain, and they multiply to $-1 \pmod p$.

c) If N is *not prime*, suppose for a contradiction that $(N-1)! \equiv -1 \pmod N$. Then this means we have, for some integer k, $(N-1)! + 1 = kN$, which implies

$$(N-1)! - kN = 1\,.$$

Since N is not prime, it has a factor d so that $1 < d < N$. This d divides the left side of the above equation (see why?), which means it should divide the right side. But the right side is 1, so we obtain a contradiction. [Another way to complete the argument is to say that the above equation implies that 1 is the GCD of $(N-1)!$ and N (see why?), which is not true, if N is composite.]