Number Theory III

Margaret M. Fleck

10 February 2010

This lecture finishes the discussion of number theory, by covering some of the material in section 3.6 and Appendix 3 of Rosen.

1 Startup

The first 20 minutes of class are consumed by the first quiz. The rest of the time will cover a short topic: the Euclidean algorithm for computing gcd.

First, let's check that we're on top of one special case for gcd. What's $\gcd(0, 37)$? Every integer k divides 0, so the largest common divisor of 0 and 37 must be 37. And, in general, $\gcd(0, k) = k$ for any non-zero integer k.

Notice that $\gcd(0, 0)$ isn't defined. All integers are common divisors of 0 and 0, so there is no greatest one.

2 An interesting fact about gcd

First, let's note one useful property of gcd.

Claim 1 *For any integers a, b, q and r, if $a = bq + r$, then* $\gcd(a, b) = \gcd(b, r)$.

> Proof: Suppose that n is some integer which divides both a and b. Then n divides bq and so n divides $a - bq$. (E.g. use various

lemmas about divides from last week.) But $a - bq$ is just r. So n divides r.

By an almost identical line of reasoning, if n divides both b and r, then n divides a.

So, the set of common divisors of a and b is exactly the same as the set of common divisors of b and r. But $\gcd(a, b)$ and $\gcd(b, r)$ are just the largest numbers in these two sets, so if the sets contain the same things, the two gcd's must be equal.

Notice that $r = a \bmod b$ satisfies the equation $a = bq + r$, so therefore

Corollary: For any positive integers a and b, then $\gcd(a, b) = \gcd(b, a \bmod b)$.

The term "corollary" means that this fact is a really easy consequence of the preceding claim.

3 Euclidean algorithm

Given this fact, we can give a fast algorithm for computing gcd, which dates back to Euclid (around 300 B.C.).

```
procedure gcd(a,b: positive integers)
x := a
y := b
while y != 0
   begin
   r := x mod y
   x := y
   y := r
   end
return x
```

Let's trace this algorithm on inputs $a = 105$ and $b = 252$. Traces should summarize the values of the most important variables.

x	y	$r = x \bmod y$
105	252	105
252	105	42
105	42	21
42	21	0
21	0	

Since x is smaller than y, the first iteration of the loop swaps x and y. After that, each iteration reduces the sizes of a and b, because $a \bmod b$ is smaller than b. In the last iteration, y has gone to zero, so we output the value of x which is 21.

To verify that this algorithm is correct, we need to convince ourselves of two things. First, it must halt, because each iteration reduces the magnitude of y. Second, by our corollary above, the value of $\gcd(x, y)$ does not change from iteration to iteration. Moreover, $\gcd(x, 0)$ is x, for any non-zero integer x. So the final output will be the gcd of the two inputs a and b.

This is a genuinely very nice algorithm. Not only is it fast, but it involves very simple calculations that can be done by hand (without a calculator). It's much easier than factoring both numbers into primes, especially as the individual prime factors get larger. Most of us can't quickly see whether a large number is divisible by, say, 17.

4 Pseudocode

Notice that this algorithm is written in *pseudocode*. Pseudocode is an abstracted type of programming language, used to highlight the important structure of an algorithm and communicate between researchers who may not use the same programming language. It borrows many control constructs (e.g. the while loop) from imperative languages such as C. But details required only for a mechanical compiler (e.g. type declarations for all variables) are omitted and equations or words are used to hide details that are easy to figure out.

Actual C or Java code is **not** acceptable pseudocode.

If you have taken a programming course, pseudocode is typically easy to read. You won't need to write much pseudocode in this class, but you'll need

to write a lot for CS 373. A common question is how much detail to use. Try to use about the same amount as in the examples shown in class.

Appendix 3 of Rosen shows the specific conventions used in the textbook. We'll try to use something similar. However, it's not necessary to adhere religiously to a precise set of conventions, because pseudocode is read by a human, not a computer.

5 A recursive version of gcd

We can also write gcd as a recursive algorithm

```
procedure gcd(a,b: positive integers)
r := a mod b
if (r = 0) return b
else return gcd(b,r)
```

This code is very simple, because this algorithm has a natural recursive structure. Our corollary allows us to express the gcd of two numbers in terms of the gcd of a smaller pair of numbers. That is to say, it allows us to reduce a larger version of the task to a smaller version of the same task.