COMPSCI 250: Introduction to Computation

Lecture #14: The Chinese Remainder Theorem
David Mix Barrington and Ghazaleh Parvini
6 October 2023

The Chinese Remainder Theorem

- Reviewing Inverses and the Inverse Theorem
- Systems of Congruences, Examples
- The Simple (Two Modulus) Version
- Proving the Simple Version
- The Full (Many Modulus) Version
- Working With Really Big Numbers

Practicality for Large Inputs

- We have a general algorithm to test whether a number is prime, but it is wholly impractical for very large inputs.
- If a number has 100 digits, we would have to check every possible prime divisor up to its square root, which would be a number of about 50 digits.
- Since a sizable fraction of all such numbers are prime, this would take us eons even if we could test a trillion per second.

Practicality For Large Inputs

- By contrast, testing *relative* primality is very practical even for very large inputs (once you have a data structure to work with numbers too big for an int or a long).

- We'll see later in the course that on inputs with n digits, the Euclidean Algorithm takes O(n) time -- on inputs of 100 digits it will take a few hundred steps at worst. The worst case is when the inputs are **Fibonacci numbers**, as in our example of 610 and 233.

Practicality for Large Inputs

- There are better ways to **test for primality**, mentioned briefly last lecture and in more detail in COMPSCI 501.
- The most practical one is **randomized**, and actually has a small chance of falsely claiming that a composite number is prime.
- But that chance can be made arbitrarily small by doing a reasonable number of independent repeated tests. An error probability of 2^{-100} is good enough for nearly any application.

Practicality for Large Inputs

- But factoring appears to be an even harder problem -- if I multiply two 100-digit primes together, there is no practical method known to get the factors back.
- The **RSA cryptosystem** is currently believed to be secure, because the only known (known to the general public, at least) way to break it is to factor the product of two very large primes.

Reviewing Inverses

- We have been working with arithmetic where the "numbers" are congruence classes modulo m.
- A class [x] (the set $\{n: n \equiv x\}$) has a **multiplicative inverse** if there is another class [y] such that $[x][y] = [1]$, or $xy \equiv 1 \pmod{m}$.
- The **Inverse Theorem** says that a number z has a multiplicative inverse modulo m if and only if z and m are relatively prime, or $\gcd(z, m) = 1$.

The Inverse Algorithm

- It's fairly clear that if z and m have a common factor $g > 1$, then a multiplicative inverse for z modulo m is impossible.
- The Euclidean Algorithm is our method to compute gcd's and thus test for relative primality.
- The **Extended Euclidean Algorithm** takes z and m as inputs and uses the arithmetic from the Euclidean Algorithm, but gets an additional result at each step.

The Inverse Algorithm

- We write each number that occurs as an integer **linear combination** of z and m.
- If z and m are relatively prime, we compute numbers a and b such that $az + bm = 1$.
- Then a is an inverse of z modulo m and b is an inverse of m modulo z.

```
119 % 65 = 54
65 % 54 = 11
54 % 11 = 10
11 % 10 = 1
10 % 1 = 0
```

```
119 = 1×65 + 54
65 = 1×54 + 11
54 = 4×11 + 10
11 = 1×10 + 1
10 = 10×1 + 0
```

```
119 = 1×119 + 0×65
65 = 0×119 + 1×65
54 = 1×119 - 1×65
11 = -1×119 + 2×65
10 = 5×119 - 9×65
1 = -6×119 + 11×65
```

Clicker Question #1

- Suppose we are using the Inverse Algorithm to find the inverse of 19, mod 27. If we have already calculated $19 = 0 \cdot 27 + 1 \cdot 19$ and $8 = 1 \cdot 27 - 1 \cdot 19$, what do we get *next*?
- (a) $27 = 1 \cdot 27 + 0 \cdot 19$, by addition
- (b) $11 = -1 \cdot 27 + 2 \cdot 19$, subtracting 8 once
- (c) $3 = -2 \cdot 27 + 3 \cdot 19$, subtracting 8 twice
- (d) $1 = -7 \cdot 27 + 10 \cdot 19$, inverse is 10

Not the Answer

Clicker Answer #1

- Suppose we are using the Inverse Algorithm to find the inverse of 19, mod 27. If we have already calculated $19 = 0 \cdot 27 + 1 \cdot 19$ and $8 = 1 \cdot 27 - 1 \cdot 19$, what do we get *next*?

backwards!

- (a) $27 = 1 \cdot 27 + 0 \cdot 19$, by addition

subtract as many as you can

- (b) $11 = -1 \cdot 27 + 2 \cdot 19$, subtracting 8 once

$19/8 = 2$

- (c) $3 = -2 \cdot 27 + 3 \cdot 19$, subtracting 8 twice

correct, but IA doesn't do it on one step

- (d) $1 = -7 \cdot 27 + 10 \cdot 19$, inverse is 10

Systems of Congruences

- Modular arithmetic was invented to deal with periodic processes. We've seen how to work with multiple congruences that have the same period -- for example, we know that if $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then $ac \equiv bd \pmod{m}$.
- But we sometimes have interacting periodic processes with different moduli. For example, days of the week have period 7. Suppose you have to take a pill every five days. How often do you take a pill on a Wednesday? Every 35 days, as it turns out.

Systems of Congruences

- A mod-5 process and a mod-7 process interact to give a mod-35 process, and something similar happens whenever the moduli are relatively prime.
- If two moduli are not relatively prime, the two congruences may not have any common solution -- consider $x \equiv 1 \pmod 4$ and $x \equiv 4 \pmod 6$.

Examples of Congruence Systems

- Suppose we have around a thousand soldiers marching along the road and we would like to know exactly how many there are.
- We tell them to line up in rows of 7 and determine how many are left over. Then we do the same for rows of 8, then again for rows of 9.
- The full form of the Chinese Remainder Theorem lets us use these three remainders to find the number of soldiers modulo $7 \times 8 \times 9 = 504$. It might say, for example, that the number is either 806 or 1310, and then we can tell which.

Examples of Congruence Systems

- The pseudoscientific (i.e. “wrong”) theory of biorhythms says that a person has three cycles started at birth, of 23, 28, and 33 days.
- According to the full form of the Chinese Remainder Theorem, a person would be at the initial position of all three cycles again exactly $23 \times 28 \times 33 = 21252$ days, or about 58.2 years, after birth.

The Simple (Two-Modulus) Version

- How can we find a common solution to the two congruences $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$?
- The **Simple Version of the Chinese Remainder Theorem** says that if m and n are relatively prime, this pair of congruences is equivalent to the single congruence $x \equiv c \pmod{mn}$, where c is a number that we can calculate from a, b, m, and n.

Clicker Question #2

- Suppose that x is a natural satisfying the congruences $x \equiv 1 \pmod{3}$ and $x \equiv 25 \pmod{55}$. What does the Chinese Remainder Theorem tell us?
- (a) $x \equiv c \pmod{165}$, for some number c.
- (b) $x = 35$, as $35 \equiv 1 \pmod{3}$ and $35 \equiv 25 \pmod{55}$.
- (c) There can be no such x, because 3 and 55 are not relatively prime.
- (d) It tells us nothing, because 3 and 55 are not relatively prime.

Not the Answer

Clicker Answer #2

- Suppose that x is a natural satisfying the congruences $x \equiv 1 \pmod{3}$ and $x \equiv 25 \pmod{55}$. What does the Chinese Remainder Theorem tell us?
- (a) $x \equiv c \pmod{165}$, for some number c.
- (b) $x = 35$, as **$35 \equiv 1 \pmod{3}$** and **$35 \equiv 25 \pmod{55}$**.
  Wrong! Wrong!
- (c) There can be no such x, because 3 and 55 are **not relatively prime**. Wrong!
- (d) It tells us nothing, because 3 and 55 are **not relatively prime**.
  Wrong!

The Simple Version

- Note first that if x is a solution to the two congruences, so is any y that satisfies $x \equiv y \pmod{mn}$.
- This is because in this case $y = x + kmn$ for some integer k. When we divide y by m, for example, we get the remainder for x plus the remainder for kmn, and the latter is 0 because m divides kmn.
- We need a c that gives us a solution to both congruences, and we must show that any solution x to both congruences must satisfy $x \equiv c \pmod{mn}$.

Proving the Simple Version

- Since m and n are assumed to be relatively prime, the Inverse Algorithm gives us integers y and z such that ym + zn = 1.
- Our number c will be bym + azn.
- Let's verify that this works. When we divide bym + azn by m, the first term gives remainder 0 and the second gives [azn] = [a][zn] = [a][1] = [a].

Proving the Simple Version

- Dividing $bym + azn$ by n, the first term gives $[b][ym] = [b][1] = [b]$, and the second term gives 0.
- A good way to think of this is that the original equation $ym + zn = 1$ tells us how to get a number whose remainders are 1 (mod m) and 1(mod n).
- To get arbitrary a and b we can adjust either term without affecting the remainder for the other modulus.

Proving the Simple Version

- Let x be any solution to $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$, and let d be x - c. Then d is divisible by both m and n.
- Use the Euclidean Algorithm to find the gcd of d and mn (or -d and mn, if d is negative) -- call this q. But q is a **common multiple** of m and n, and the least common multiple of two relatively prime numbers is their product.

The Full (Many-Modulus) Version

- More generally, as in our examples, suppose we have several congruences $x = a_1 \pmod{m_1}$, $x = a_2 \pmod{m_2}$,… $x = a_k \pmod{m_k}$, and that the moduli are **pairwise relatively prime**. (This means that any two of them are relatively prime to each other.)
- Then the Full Form of the Chinese Remainder Theorem says that this system of congruences is equivalent to a single congruence $x \equiv c \pmod{M}$, where $M = m_1 m_2 \ldots m_k$.

Clicker Question #3

- Suppose that x is a natural satisfying $x \equiv 2 \pmod{6}$, $x \equiv 1 \pmod{7}$, and $x \equiv 6 \pmod{8}$. What conclusion can we draw from the Chinese Remainder Theorem?
- (a) $x = 302$
- (b) $x \equiv 302 \pmod{6 \cdot 7 \cdot 8 = 336}$
- (c) None, because 6, 7, and 8 are not pairwise relatively prime
- (d) None, because 2, 1, and 6 are not pairwise relatively prime

Not the Answer

Clicker Answer #3

- Suppose that x is a natural satisfying $x \equiv 2 \pmod{6}$, $x \equiv 1 \pmod{7}$, and $x \equiv 6 \pmod{8}$. What conclusion can we draw from the Chinese Remainder Theorem?
- (a) $x = 302$ CRT never gives you a number without more conditions
- (b) $x \equiv 302 \pmod{6 \cdot 7 \cdot 8 = 336}$ In fact here $x = 302 \pmod{168}$.

  would be correct if CRT worked here, but it doesn't
- (c) None, because 6, 7, and 8 are not pairwise relatively prime

  CRT does not apply for this reason
- (d) None, because 2, 1, and 6 are not pairwise relatively prime

  no reason for *those* numbers have to be relatively prime

The Full Version

- Specifically, M is the product of the m_i's and c is a number that can be calculated from the a_i's and the m_i's.
- We can prove the Full Version from the Simple Version. If $k = 3$, for example, we first use the Simple Version to find a c such that the first two congruences are equivalent to $x \equiv c \pmod{m_1 m_2}$. Then we have two congruences, that and $x \equiv a_3 \pmod{m_3}$.

The Full Version

- We now just use the Simple Version again to get a common solution to these two congruences. (The pairwise relatively prime property guarantees that $m_1 m_2$ will be relatively prime to m_3.)
- This clearly extends to larger k.
- In the book, it is shown how we can calculate the single c directly.

Working With Very Big Numbers

- If I have some very very big integers, each too big to store in a single computer word, the Chinese Remainder Theorem gives me an alternate way to calculate with them.
- Say I want to multiply n of these numbers together.
- I pick a bunch of different prime numbers, so many that their product is bigger than the product of my big numbers.

Working With Very Big Numbers

- How do we know that such primes exist?
- A more sophisticated analysis shows that there are *lots* of primes that fit in a single machine word, so I can get to very very big numbers by multiplying them together.)
- I then find the remainder of each big number modulo each prime.

Working With Very Big Numbers

- If I multiply together all the remainders for a given prime p, and take the result modulo p, I have my product's remainder modulo p.
- And this can be done with calculations on reasonably-sized numbers, because I can do this in parallel for each prime.
- Then running the Chinese Remainder calculation once, I can get my product in the regular notation.