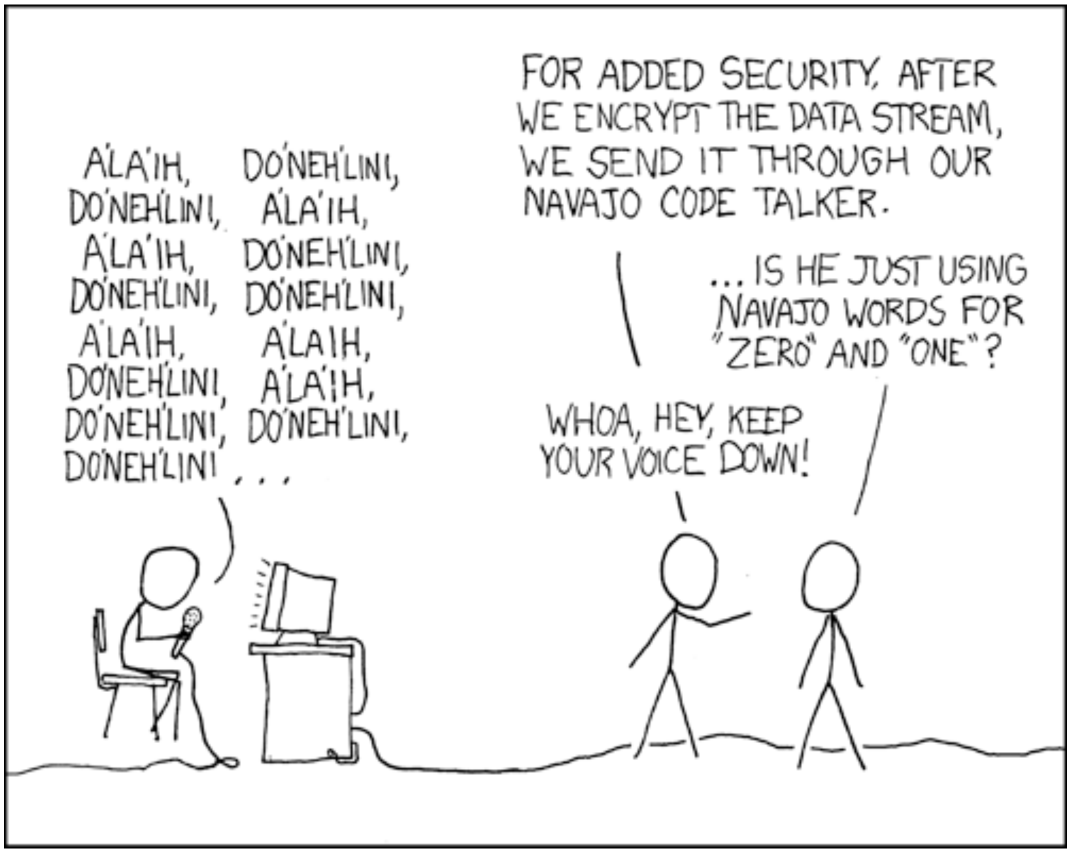
CSE 311: Foundations of Computing

Lecture 13: Primes, GCD



Recall: Familiar Properties of "="

- **If $a = b$ and $b = c$, then $a = c$.**
  - **i.e., if $a = b = c$, then $a = c$**

- **If $a = b$ and $c = d$, then $a + c = b + d$.**
  - **in particular, since $c = c$ is true, we can "$+\ c$" to both sides**

- **If $a = b$ and $c = d$, then $ac = bd$.**
  - **in particular, since $c = c$ is true, we can "$\times c$" to both sides**

These are the facts that allow us to use algebra to solve problems

Modular Arithmetic: Basic Property

Let $\boldsymbol{m}$ be a positive integer.
If $\boldsymbol{a} \equiv_{\boldsymbol{m}} \boldsymbol{b}$ and $\boldsymbol{b} \equiv_{\boldsymbol{m}} \boldsymbol{c}$, then $\boldsymbol{a} \equiv_{\boldsymbol{m}} \boldsymbol{c}$.

Modular Arithmetic: Basic Property

Let $\boldsymbol{m}$ be a positive integer.
If $\boldsymbol{a} \equiv_{\boldsymbol{m}} \boldsymbol{b}$ and $\boldsymbol{b} \equiv_{\boldsymbol{m}} \boldsymbol{c}$, then $\boldsymbol{a} \equiv_{\boldsymbol{m}} \boldsymbol{c}$.

Suppose that $a \equiv_m b$ and $b \equiv_m c$.

Modular Arithmetic: Basic Property

Let $\boldsymbol{m}$ be a positive integer.
If $\boldsymbol{a \equiv_m b}$ and $\boldsymbol{b \equiv_m c}$, then $\boldsymbol{a \equiv_m c}$.

Suppose that $a \equiv_m b$ and $b \equiv_m c$. Then, by the previous property, we have $a \bmod m = b \bmod m$ and $b \bmod m = c \bmod m$.

Putting these together, we have $a \bmod m = c \bmod m$, which says that $a \equiv_m c$, by the previous property.

Modular Arithmetic: Addition Property

Let $\boldsymbol{m}$ be a positive integer. If $\boldsymbol{a} \equiv_{\boldsymbol{m}} \boldsymbol{b}$ and $\boldsymbol{c} \equiv_{\boldsymbol{m}} \boldsymbol{d}$, then $\boldsymbol{a}+\boldsymbol{c} \equiv_{\boldsymbol{m}} \boldsymbol{b}+\boldsymbol{d}$.

Modular Arithmetic: Addition Property

Let $\boldsymbol{m}$ be a positive integer. If $\boldsymbol{a \equiv_m b}$ and $\boldsymbol{c \equiv_m d}$, then $\boldsymbol{a + c \equiv_m b + d}$.

Suppose that $a \equiv_m b$ and $c \equiv_m d$.

Modular Arithmetic: Addition Property

Let $\boldsymbol{m}$ be a positive integer. If $\boldsymbol{a \equiv_m b}$ and $\boldsymbol{c \equiv_m d}$, then $\boldsymbol{a + c \equiv_m b + d}$.

Suppose that $a \equiv_m b$ and $c \equiv_m d$. Unrolling the definitions, we can see that $a - b = km$ and $c - d = jm$ for some $k, j \in \mathbb{Z}$.

Adding the equations together gives us
$(a + c) - (b + d) = m(k + j)$.

By the definition of congruence, we have $a + c \equiv_m b + d$.

Modular Arithmetic: Multiplication Property

Let $\boldsymbol{m}$ be a positive integer. If $\boldsymbol{a \equiv_m b}$ and $\boldsymbol{c \equiv_m d}$, then $\boldsymbol{ac \equiv_m bd}$.

Modular Arithmetic: Multiplication Property

Let $\boldsymbol{m}$ be a positive integer. If $\boldsymbol{a \equiv_m b}$ and $\boldsymbol{c \equiv_m d}$, then $\boldsymbol{ac \equiv_m bd}$.

Suppose that $a \equiv_m b$ and $c \equiv_m d$.

Modular Arithmetic: Multiplication Property

Let $\boldsymbol{m}$ be a positive integer. If $\boldsymbol{a \equiv_m b}$ and $\boldsymbol{c \equiv_m d}$, then $\boldsymbol{ac \equiv_m bd}$.

Suppose that $a \equiv_m b$ and $c \equiv_m d$. Unrolling the definitions, we can see that $a - b = km$ and $c - d = jm$ for some $k, j \in \mathbb{Z}$ or equivalently, $a = km + b$ and $c = jm + d$.

Multiplying both together gives us $ac = (km + b)(jm + d) = kjm^2 + kmd + bjm + bd$.

Modular Arithmetic: Multiplication Property

Let $\boldsymbol{m}$ be a positive integer. If $\boldsymbol{a} \equiv_{\boldsymbol{m}} \boldsymbol{b}$ and $\boldsymbol{c} \equiv_{\boldsymbol{m}} \boldsymbol{d}$, then $\boldsymbol{ac} \equiv_{\boldsymbol{m}} \boldsymbol{bd}$.

Suppose that $a \equiv_m b$ and $c \equiv_m d$. Unrolling the definitions, we can see that $a - b = km$ and $c - d = jm$ for some $k, j \in \mathbb{Z}$ or equivalently, $a = km + b$ and $c = jm + d$.

Multiplying both together gives us $ac = (km + b)(jm + d) = kjm^2 + kmd + bjm + bd$. Re-arranging, this becomes $ac - bd = m(kjm + kd + bj)$.

This says $ac \equiv_m bd$ by the definition of congruence.

Modular Arithmetic: Properties

If $\boldsymbol{a \equiv_m b}$ and $\boldsymbol{b \equiv_m c}$, then $\boldsymbol{a \equiv_m c}$.

If $\boldsymbol{a \equiv_m b}$ and $\boldsymbol{c \equiv_m d}$, then $\boldsymbol{a + c \equiv_m b + d}$.

Corollary: If $\boldsymbol{a \equiv_m b}$, then $\boldsymbol{a + c \equiv_m b + c}$.

If $\boldsymbol{a \equiv_m b}$ and $\boldsymbol{c \equiv_m d}$, then $\boldsymbol{ac \equiv_m bd}$.

Corollary: If $\boldsymbol{a \equiv_m b}$, then $\boldsymbol{ac \equiv_m bc}$.

Modular Arithmetic: Properties

If $\boldsymbol{a} \equiv_m \boldsymbol{b}$ and $\boldsymbol{b} \equiv_m \boldsymbol{c}$, then $\boldsymbol{a} \equiv_m \boldsymbol{c}$.

If $\boldsymbol{a} \equiv_m \boldsymbol{b}$, then $\boldsymbol{a} + \boldsymbol{c} \equiv_m \boldsymbol{b} + \boldsymbol{c}$.

If $\boldsymbol{a} \equiv_m \boldsymbol{b}$, then $\boldsymbol{ac} \equiv_m \boldsymbol{bc}$.

"$\equiv_m$" allows us to solve problems in modular arithmetic, e.g.

- **add / subtract numbers from both sides of equations**
- **chains of "$\equiv_m$" values shows first and last are "$\equiv_m$"**
- **substitute "$\equiv_m$" values in equations (not proven yet)**

Substitution Follows From Other Properties

Given $2y + 3x \equiv_m 25$ **and** $x \equiv_m 7$,
show that $2y + 21 \equiv_m 25$. **(substituting** 7 **for** x**)**

Start from $x \equiv_m 7$

Multiply both sides $3x \equiv_m 21$

Add to both sides $2\text{y} + 3x \equiv_m 2y + 21$

Combine $\equiv_m$**'s** $2y + 21 \equiv_m 2\text{y} + 3x \equiv_m 25$

Basic Applications of mod

- Two's Complement
- Hashing
- Pseudo random number generation

n-bit Unsigned Integer Representation

- Represent integer x as sum of powers of 2:

99	= 64 + 32 + 2 + 1	= $2^6 + 2^5 + 2^1 + 2^0$
18	= 16 + 2	= $2^4 + 2^1$

If $b_{n-1}2^{n-1} + \cdots + b_1 2 + b_0$ with each $b_i \in \{0,1\}$
then binary representation is $b_{n-1} \ldots b_2\ b_1\ b_0$

- For n = 8:
  99: 0110 0011
  18: 0001 0010

Easy to implement arithmetic $\mathbf{mod}\ \mathbf{2^n}$
... just throw away bits n+1 and up

$2^n \mid 2^{n+k}$ so $b_{n+k}2^{n+k} \equiv_{2^n} 0$
for $k \geq 0$

n-bit Unsigned Integer Representation

- Largest representable number is $2^n - 1$

$$2^n = 100\ldots000 \qquad \text{(n+1 bits)}$$

$$2^n - 1 = 11\ldots111 \qquad \text{(n bits)}$$

THE WALL STREET JOURNAL.

32 bits

1 = $0.0001

$429,496.7295 max

Berkshire Hathaway Inc. (BRK-A)

NYSE - Nasdaq Real Time Price. Currency in USD

436,401.00 +679.50 (+0.16%)

At close: 4:00PM EDT

Sign-Magnitude Integer Representation

n-bit signed integers
Suppose that $-2^{n-1} < x < 2^{n-1}$
First bit as the sign, $n-1$ bits for the value

99 = 64 + 32 + 2 + 1
18 = 16 + 2

For n = 8:
99: 0110 0011
-18: 1001 0010

Problem: this has both +0 and -0 (annoying)

Two's Complement Representation

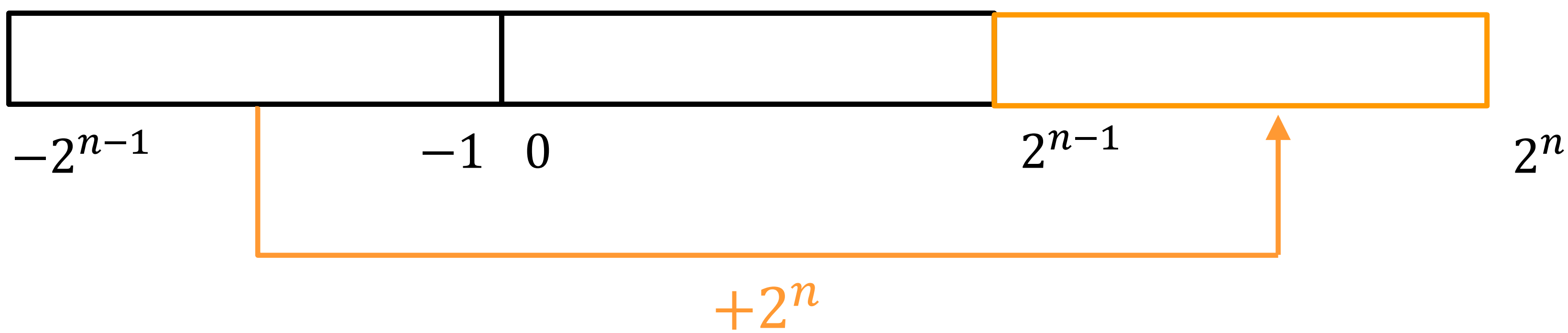
Suppose that $0 \leq x < 2^{n-1}$

   x is represented by the binary representation of x

Suppose that $-2^{n-1} \leq x < 0$

   x is represented by the binary representation of $x + 2^n$

   result is in the range $2^{n-1} \leq x < 2^n$



0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Two's Complement Representation

Suppose that $0 \leq x < 2^{n-1}$

x is represented by the binary representation of x

Suppose that $-2^{n-1} \leq x < 0$

x is represented by the binary representation of $x + 2^n$

result is in the range $2^{n-1} \leq x < 2^n$

0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

99 = 64 + 32 + 2 + 1

18 = 16 + 2

For n = 8:

99: 0110 0011

-18: 1110 1110 (-18 + 256 = 238)

Two's Complement Representation

Suppose that $0 \leq x < 2^{n-1}$

x is represented by the binary representation of x

Suppose that $-2^{n-1} \leq x < 0$

x is represented by the binary representation of $x + 2^n$

result is in the range $2^{n-1} \leq x < 2^n$

0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Key property: First bit is still the sign bit!

Key property: Twos complement representation of any number $\boldsymbol{y}$ is equivalent to $\boldsymbol{y} \bmod \mathbf{2^n}$ so arithmetic works $\bmod \mathbf{2^n}$

$$y + 2^n \equiv_{2^n} y$$

Two's Complement Representation

- For $0 < x \leq 2^{n-1}$, $-x$ is represented by the binary representation of $-x + 2^n$
  - How do we calculate –x from x?
  - E.g., what happens for "`return -x;`" in Java?

$$-x + 2^n = (2^n - 1) - \mathrm{x} + 1$$

- To compute this, flip the bits of x then add 1!
  - All 1's string is $2^n - 1$, so

    Flip the bits of x means replace x by $2^n - 1 - x$

    Then add 1 to get $-x + 2^n$

More Number Theory

Primes and GCD

Primality

An integer *p* greater than 1 is called *prime* if the only positive factors of *p* are 1 and *p*.

$$p > 1 \wedge \forall x\, ((x \mid p) \rightarrow ((x = 1) \vee (x = p)))$$

A positive integer that is greater than 1 and is not prime is called *composite*.

$$p > 1 \wedge \exists x\, ((x \mid p) \wedge (x \neq 1) \wedge (x \neq p))$$

Fundamental Theorem of Arithmetic

Every positive integer greater than 1 has a “unique” prime factorization

48 = 2 • 2 • 2 • 2 • 3
591 = 3 • 197
45,523 = 45,523
321,950 = 2 • 5 • 5 • 47 • 137
1,234,567,890 = 2 • 3 • 3 • 5 • 3,607 • 3,803

Algorithmic Problems

- **Multiplication**
  - Given primes $p_1, p_2, \ldots, p_k$, calculate their product $p_1 p_2 \ldots p_k$
- **Factoring**
  - Given an integer n, determine the prime factorization of n

Factoring

Factor the following 232 digit number [RSA768]:

123018668453011775513049495838496272077
285356959533479219732245215172640050726
365751874520219978646938995647494277406
384592519255732630345373154826850791702
612214291346167042921431160222124047927
4737794080665351419597459856902143413

1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413

=

3347807169895689878604416984821269081770479498371376856891243138898288379387800228761471165253174308773781446799949

×

3674604366679959042824463379962795263227915816434308764267603228381573966651127923337341714339681027009279873630891

Famous Algorithmic Problems

- Factoring
  - Given an integer n, determine the prime factorization of n
- Primality Testing
  - Given an integer n, determine if n is prime

- Factoring is hard
  - (on a classical computer)
- Primality Testing is easy

Greatest Common Divisor

GCD(a, b):

Largest integer d such that $d \mid a$ and $d \mid b$

- GCD(100, 125) =
- GCD(17, 49) =
- GCD(11, 66) =
- GCD(13, 0) =
- GCD(180, 252) =

d is GCD iff $(d \mid a) \wedge (d \mid b) \wedge \forall x\, (((x \mid a) \wedge (x \mid b)) \rightarrow (x \leq d))$

GCD and Factoring

$a = 2^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 = 46{,}200$

$b = 2 \cdot 3^2 \cdot 5^3 \cdot 7 \cdot 13 = 204{,}750$

$$\mathrm{GCD}(a, b) = 2^{\min(3,1)} \cdot 3^{\min(1,2)} \cdot 5^{\min(2,3)} \cdot 7^{\min(1,1)} \cdot 11^{\min(1,0)} \cdot 13^{\min(0,1)}$$

Factoring is hard!

Can we compute GCD(a,b) without factoring?

Useful GCD Fact

Let *a* and *b* be positive integers.
We have gcd(*a*,*b*) = gcd(*b*, *a* mod *b*)

Proof:
We will show that every number dividing a and b also divides b and $a \bmod b$.
I.e., $d|a$ and $d|b$ iff $d|b$ and $d|(a \bmod b)$.

Hence, their set of common divisors are the same,
which means that their greatest common divisor is the same.

Useful GCD Fact

Let *a* and *b* be positive integers.
We have gcd(*a*,*b*) = gcd(*b*, *a* mod *b*)

Proof:
By definition of mod, $a = qb + (a \bmod b)$ for some integer $q = a \operatorname{div} b$.

Suppose $d|b$ and $d|(a \bmod b)$.
Then $b = md$ and $(a \bmod b) = nd$ for some integers m and n.
Therefore $a = qb + (a \bmod b) = qmd + nd = (qm + n)d$.
So $d|a$.

Suppose $d|a$ and $d|b$.
Then $a = kd$ and $b = jd$ for some integers k and j.
Therefore $(a \bmod b) = a - qb = kd - qjd = (k - qj)d$.
So, $d|(a \bmod b)$ also.

Since they have the same common divisors, $\gcd(a, b) = \gcd(b, a \bmod b)$. ∎

Another simple GCD fact

Let a be a positive integer.
We have $\gcd(a,0) = a$.

Euclid's Algorithm

$\gcd(a, b) = \gcd(b, a \bmod b)$ $\gcd(a, 0) = a$

```
int gcd(int a, int b){ /* Assumes: a >= b, b >= 0 */
    if (b == 0) {
        return a;
    } else {
        return gcd(b, a % b);
    }
}
```

Note: gcd(b, a) = gcd(a, b)

Euclid's Algorithm

Repeatedly use $\gcd(a, b) = \gcd(b, a \bmod b)$ to reduce numbers until you get $\gcd(g, 0) = g$.

gcd(660,126) =

Euclid's Algorithm

Repeatedly use $\gcd(a, b) = \gcd(b, a \bmod b)$ to reduce numbers until you get $\gcd(g, 0) = g$.

$$
\begin{aligned}
\gcd(660,126) &= \gcd(126,\ 660 \bmod 126) &&= \gcd(126,\ 30)\\
&= \gcd(30,\ 126 \bmod 30) &&= \gcd(30,\ 6)\\
&= \gcd(6,\ 30 \bmod 6) &&= \gcd(6,\ 0)\\
&= 6
\end{aligned}
$$