Number Theory

Zachary Friggstad

Programming Club Meeting

Outline

- Factoring
- Sieve
- Multiplicative Functions
- Greatest Common Divisors
- Applications
- Chinese Remainder Theorem

Throughout, **problems to try** are highlighted. Some are just routine implementations of the algorithm recently discussed, some require a bit of thought.

Advice

Number theory is a fundamental and very beautiful topic! But it is hard to visualize things, especially in the context of algorithms (unlike graphs and geometry).

Even some standard things like running-time bounds are hard to establish.

Trace some algorithms by hand on small examples to see what they are doing! This helped me a lot.

Factoring

Theorem (Fundamental Theorem of Arithmetic)

Every integer $n \geq 2$ can be uniquely expressed in the form $p_1^{a_1} \cdot \ldots \cdot p_k^{a_k}$ where $p_1 \leq \ldots \leq p_k$ are primes and $a_i \geq 1$ are integers.

We usually just try *trial division* to factor an integer n:

- Find the smallest integer $p \geq 2$ dividing n.
- Divide it out (it must be a prime) and repeat.

Speedup: just try values $p \leq \sqrt{n}$, if anything remains it is a prime since n cannot have two prime divisors $> \sqrt{n}$.

```
map<int, int> primes;

for (int p = 2; p*p <= n; ++p)
  while (n%p == 0) {
    ++primes[p];
    n /= p;
  }
if (n > 1) ++primes[n];

for (auto& x : primes) {
  //x.first is a prime dividing n
  //x.second is the number of times it divides n
}
```

Running Time: $O(\sqrt{n})$

Problem to try:
UVa 583 - Prime Factors

Sieve: Find all primes $\leq n$.

- Write all numbers from 2 to n.
- Find the smallest number p not highlighted.
- Highlight it and cross off larger multiples.

$$2, 3, 4, 5, 6, 7, 8, 9, 10$$

Sieve: Find all primes $\leq n$.

- Write all numbers from 2 to n.
- Find the smallest number p not highlighted.
- Highlight it and cross off larger multiples.

$$2, 3, 4, 5, 6, 7, 8, 9, 10$$

$$\mathbf{2}, 3, \not{4}, 5, \not{6}, 7, \not{8}, 9, \not{1}0$$

Sieve: Find all primes $\leq n$.

- Write all numbers from 2 to n.
- Find the smallest number p not highlighted.
- Highlight it and cross off larger multiples.

$$2, 3, 4, 5, 6, 7, 8, 9, 10$$

$$\mathbf{2}, 3, \not{4}, 5, \not{6}, 7, \not{8}, 9, \not{1}0$$

$$\mathbf{2}, \mathbf{3}, \not{4}, 5, \not{6}, 7, \not{8}, \not{9}, \not{1}0$$

Sieve: Find all primes $\leq n$.

- Write all numbers from 2 to n.
- Find the smallest number p not highlighted.
- Highlight it and cross off larger multiples.

$$2, 3, 4, 5, 6, 7, 8, 9, 10$$

$$\mathbf{2}, 3, \not{4}, 5, \not{6}, 7, \not{8}, 9, \not{10}$$

$$\mathbf{2}, \mathbf{3}, \not{4}, 5, \not{6}, 7, \not{8}, \not{9}, \not{10}$$

$$\mathbf{2}, \mathbf{3}, \not{4}, \mathbf{5}, \not{6}, 7, \not{8}, \not{9}, \not{10}$$

Sieve: Find all primes $\leq n$.

- Write all numbers from 2 to n.
- Find the smallest number p not highlighted.
- Highlight it and cross off larger multiples.

$$2, 3, 4, 5, 6, 7, 8, 9, 10$$

$$\mathbf{2}, 3, \not 4, 5, \not 6, 7, \not 8, 9, \not 10$$

$$\mathbf{2}, \mathbf{3}, \not 4, 5, \not 6, 7, \not 8, \not 9, \not 10$$

$$\mathbf{2}, \mathbf{3}, \not 4, \mathbf{5}, \not 6, 7, \not 8, \not 9, \not 10$$

$$\mathbf{2}, \mathbf{3}, \not 4, \mathbf{5}, \not 6, \mathbf{7}, \not 8, \not 9, \not 10$$

Sieve: Find all primes $\leq n$.

- Write all numbers from 2 to n.
- Find the smallest number p not highlighted.
- Highlight it and cross off larger multiples.

$$2, 3, 4, 5, 6, 7, 8, 9, 10$$

$$\mathbf{2}, 3, \not{4}, 5, \not{6}, 7, \not{8}, 9, \not{10}$$

$$\mathbf{2}, \mathbf{3}, \not{4}, 5, \not{6}, 7, \not{8}, \not{9}, \not{10}$$

$$\mathbf{2}, \mathbf{3}, \not{4}, \mathbf{5}, \not{6}, 7, \not{8}, \not{9}, \not{10}$$

$$\mathbf{2}, \mathbf{3}, \not{4}, \mathbf{5}, \not{6}, \mathbf{7}, \not{8}, \not{9}, \not{10}$$

Crossed out numbers are multiples of smaller numbers: **not prime**.

Sieve: Find all primes $\leq n$.

- Write all numbers from 2 to n.
- Find the smallest number p not highlighted.
- Highlight it and cross off larger multiples.

$$2, 3, 4, 5, 6, 7, 8, 9, 10$$

$$\mathbf{2}, 3, \not 4, 5, \not 6, 7, \not 8, 9, \not 10$$

$$\mathbf{2}, \mathbf{3}, \not 4, 5, \not 6, 7, \not 8, \not 9, \not 10$$

$$\mathbf{2}, \mathbf{3}, \not 4, \mathbf{5}, \not 6, 7, \not 8, \not 9, \not 10$$

$$\mathbf{2}, \mathbf{3}, \not 4, \mathbf{5}, \not 6, \mathbf{7}, \not 8, \not 9, \not 10$$

Crossed out numbers are multiples of smaller numbers: **not prime**.

Highlighted numbers are not multiples of smaller numbers: **prime**.

Speedup

1) Only go up to $\sqrt{n}$, anything not highlighted or crossed out must be a prime since any composite number is divisible by a prime $\leq \sqrt{n}$.

Note: This is only a practical speedup, not an asymptotic speedup.

Speedup

1) Only go up to $\sqrt{n}$, anything not highlighted or crossed out must be a prime since any composite number is divisible by a prime $\leq \sqrt{n}$.

Note: This is only a practical speedup, not an asymptotic speedup.

```
vector<int> primes(n+1);
for (int i = 2; i <= n; ++i) primes[i] = i;

for (int p = 2; p*p <= n; ++p)
   if (primes[p] == p) //if p is not crossed off yet
      //then cross off multiples of p
      for (int q = 2*p; q <= n; q += p)
         primes[q] = p;
//now p is a prime if and only if primes[p] == p
//if p is composite, then primes[p] is a prime divisor of p
```

Speedup

1) Only go up to $\sqrt{n}$, anything not highlighted or crossed out must be a prime since any composite number is divisible by a prime $\leq \sqrt{n}$.

Note: This is only a practical speedup, not an asymptotic speedup.

```
vector<int> primes(n+1);
for (int i = 2; i <= n; ++i) primes[i] = i;

for (int p = 2; p*p <= n; ++p)
  if (primes[p] == p) //if p is not crossed off yet
    //then cross off multiples of p
    for (int q = 2*p; q <= n; q += p)
      primes[q] = p;
//now p is a prime if and only if primes[p] == p
//if p is composite, then primes[p] is a prime divisor of p
```

Running Time

The inner loop iterates $\sum_{p \leq n \text{ prime}} \frac{n}{p} = O(n \log\log n)$ times.

Multiplicative Functions

Definition

A multiplicative function is a function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ satisfying $f(a \cdot b) = f(a) \cdot f(b)$ whenever $\gcd(a, b) = 1$.

Examples:

- $\phi(n) =$ number of integers $1 \leq k \leq n$ with $\gcd(n, k) = 1$.
- $\tau(n) =$ number of distinct positive divisors of n
- $\sigma(n) =$ sum of all positive divisors of n
- $\mu(n) = 0$ if $p^2 | n$ for some p, otherwise is $(-1)^k$ where k is the number of distinct prime divisors of n.

Let f be multiplicative. If you can easily compute $f(p^a)$ for primes p and $a \geq 1$, then you can compute $f(n)$ for all n by factoring:

$$\text{if } n = p_1^{a_1} \cdot \ldots \cdot p_k^{a_k} \text{ then } f(n) = f(p_1^{a_1}) \cdot \ldots \cdot f(p_k^{a_k}).$$

Let f be multiplicative. If you can easily compute $f(p^a)$ for primes p and $a \geq 1$, then you can compute $f(n)$ for all n by factoring:

$$\text{if } n = p_1^{a_1} \cdot \ldots \cdot p_k^{a_k} \;\text{ then } f(n) = f(p_1^{a_1}) \cdot \ldots \cdot f(p_k^{a_k}).$$

Examples:

- $\phi(p^a) = p^{a-1} \cdot (p-1)$
- $\tau(p^a) = a + 1$
- $\sigma(p^a) = 1 + p + p^2 + \ldots + p^a = \frac{p^{a+1}-1}{p-1}$
- $\mu(p^a) = \begin{cases} -1 & \text{if } a = 1 \\ 0 & \text{if } a \geq 2 \end{cases}$

Can conveniently compute $f(n)$ for all values up to n with a sieve.

Idea

- For each $2 \leq k \leq n$, compute a prime divisor of k with a sieve.
- Initialize $f(1) = 1$.
- For some $k \geq 2$, let $p | k$ with multiplicity a.
- Compute $f(n) = f(n/p^a) \cdot f(p^a)$.

Can conveniently compute $f(n)$ for all values up to n with a sieve.

Idea

- For each $2 \leq k \leq n$, compute a prime divisor of k with a sieve.
- Initialize $f(1) = 1$.
- For some $k \geq 2$, let $p|k$ with multiplicity a.
- Compute $f(n) = f(n/p^a) \cdot f(p^a)$.

Example

```
vector<int> primes(n+1);
//suppose we sieved so primes[p] is a prime divisor of p
vector<int> sigma(n+1);
sigma[1] = 1;
for (int k = 2; k <= n; ++k) {
   int s = 1, p = primes[k], m = k;
   while (m % p == 0) {
      m /= p;
      s = s*p + 1;
   } //invariant: s = sigma(p^i) after i iterations
   sigma[k] = s*sigma[m]
}
```

Problems to try:
UVa 10042 - Smith Numbers
UVa 10738 - Riemann vs. Mertens
UVa 294 - Divisors

One more thought:
Can factor in $O(\sqrt{n/\log n})$ time with the following trick.

- Sieve all primes up to $\sqrt{n}$ in $O(\sqrt{n}\log\log n)$ time.
- Do trial division up to $\sqrt{n}$ to factor n, but iterate only over the primes you sieved.

I've never needed this improvement, but it's good to keep in mind.

Greatest Common Divisors

$\gcd(a, b)$ for integers $a, b \geq 0$ is the largest integer d such that $d|a$ and $d|b$.

Note $\gcd(a, 0) = a$ if $a \geq 1$.
Standard convention: $\gcd(0, 0) = 0$.

Greatest Common Divisors

$\gcd(a, b)$ for integers $a, b \geq 0$ is the largest integer d such that $d|a$ and $d|b$.

Note $\gcd(a, 0) = a$ if $a \geq 1$.
Standard convention: $\gcd(0, 0) = 0$.

Observation
$\gcd(a, b) = \gcd(a - b, b)$ if $a \geq b$
Because anything that divides a and b also divides $a \pm b$.

Greatest Common Divisors

$\gcd(a, b)$ for integers $a, b \geq 0$ is the largest integer d such that $d|a$ and $d|b$.

Note $\gcd(a, 0) = a$ if $a \geq 1$.
Standard convention: $\gcd(0, 0) = 0$.

Observation
$\gcd(a, b) = \gcd(a - b, b)$ if $a \geq b$
Because anything that divides a and b also divides $a \pm b$.

Accelerated Subtraction
$\gcd(a, b) = \gcd(a \bmod b, b)$ (even if $a < b$)
Because $a \bmod b$ is obtained by repeatedly subtracting b from a.

Euclid's algorithm to compute $\gcd(a, b)$.

- If $b = 0$ then the answer is a.
- Otherwise, the answer is $\gcd(b, a \bmod b)$ (even if $a \leq b$).

```
int gcd(int a, int b) { return b ? gcd(b, a%b) : a; }
```

Euclid's algorithm to compute gcd(a, b).

- If $b = 0$ then the answer is a.
- Otherwise, the answer is gcd$(b, a \bmod b)$ (even if $a \leq b$).

```
int gcd(int a, int b) { return b ? gcd(b, a%b) : a; }
```

Running time: $O(\log a + \log b)$ because $a \bmod b \leq a/2$ if $a \geq b$.

Euclid's algorithm to compute $\gcd(a, b)$.

- If $b = 0$ then the answer is a.
- Otherwise, the answer is $\gcd(b, a \bmod b)$ (even if $a \leq b$).

```
int gcd(int a, int b) { return b ? gcd(b, a%b) : a; }
```

Running time: $O(\log a + \log b)$ because $a \bmod b \leq a/2$ if $a \geq b$.

Quick Proof: Obvious if $b \leq a/2$ since $a \bmod b < b$.
Otherwise $a \bmod b = a - b \leq a/2$.

Euclid's algorithm to compute $\gcd(a, b)$.

- If $b = 0$ then the answer is a.
- Otherwise, the answer is $\gcd(b, a \bmod b)$ (even if $a \leq b$).

```
int gcd(int a, int b) { return b ? gcd(b, a%b) : a; }
```

Running time: $O(\log a + \log b)$ because $a \bmod b \leq a/2$ if $a \geq b$.

Quick Proof: Obvious if $b \leq a/2$ since $a \bmod b < b$.
Otherwise $a \bmod b = a - b \leq a/2$.

Least Common Multiple

Find the smallest integer m that is a common multiple of positive integers a, b.

Euclid's algorithm to compute $\gcd(a, b)$.

- If $b = 0$ then the answer is a.
- Otherwise, the answer is $\gcd(b, a \bmod b)$ (even if $a \leq b$).

```
int gcd(int a, int b) { return b ? gcd(b, a%b) : a; }
```

Running time: $O(\log a + \log b)$ because $a \bmod b \leq a/2$ if $a \geq b$.

Quick Proof: Obvious if $b \leq a/2$ since $a \bmod b < b$.
Otherwise $a \bmod b = a - b \leq a/2$.

Least Common Multiple

Find the smallest integer m that is a common multiple of positive integers a, b.

Simply put: $\mathrm{lcm}(a, b) = \frac{a \cdot b}{\gcd(a,b)}$.

```
int lcm(int a, int b) { return a/gcd(a, b)*b; }
//division before multiplication may avoid overflow
```

Extended Euclidean Algorithm

Given integers $a, b \geq 0$, for any other integers c, d we have that $\gcd(a, b)$ divides $ac + bd$.

Question
Can we find integers c, d such that $ac + bd = \gcd(a, b)$.

Answer
Yes, and the **Extended Euclidean Algorithm** finds them.

Define a sequence of tuples (r_i, s_i, t_i) for $0 \leq i$ inductively as follows.

- $r_0 = a, r_1 = b$
- $s_0 = 1, s_1 = 0$
- $t_0 = 0, t_1 = 1$

Invariant, for any i will maintain $a \cdot s_i + b \cdot t_i = r_i$. True for $i = 0, 1$.

Define a sequence of tuples (r_i, s_i, t_i) for $0 \leq i$ inductively as follows.

- $r_0 = a, r_1 = b$
- $s_0 = 1, s_1 = 0$
- $t_0 = 0, t_1 = 1$

Invariant, for any i will maintain $a \cdot s_i + b \cdot t_i = r_i$. True for $i = 0, 1$.

Inductively for $i \geq 2$

- $q_i = \lfloor r_{i-2}/r_{i-1} \rfloor$ (**quotient**)
- $r_i = r_{i-2} - q_i \cdot r_{i-1}$ (**remainder**) *same as* $r_i = r_{i-2} \bmod r_{i-1}$
- $s_i = s_{i-2} - q_i \cdot s_{i-1}$
- $t_i = t_{i-2} - q_i \cdot t_{i-1}$

Define a sequence of tuples (r_i, s_i, t_i) for $0 \le i$ inductively as follows.

- $r_0 = a, r_1 = b$
- $s_0 = 1, s_1 = 0$
- $t_0 = 0, t_1 = 1$

Invariant, for any i will maintain $a \cdot s_i + b \cdot t_i = r_i$. True for $i = 0, 1$.

Inductively for $i \ge 2$

- $q_i = \lfloor r_{i-2}/r_{i-1} \rfloor$ (**quotient**)
- $r_i = r_{i-2} - q_i \cdot r_{i-1}$ (**remainder**) *same as* $r_i = r_{i-2} \bmod r_{i-1}$
- $s_i = s_{i-2} - q_i \cdot s_{i-1}$
- $t_i = t_{i-2} - q_i \cdot t_{i-1}$

The $r_0, r_1, r_2, \ldots$ sequence is just following Euclid's gcd algorithm.

Consequence of the Invariants

Let j be the first index where $r_j = 0$. Then

$$\gcd(a, b) = r_{j-1} = s_{j-1} \cdot a + t_{j-1} \cdot b.$$

Example

Find x, y such that $21x + 27y = \gcd(21, 27) = 3$.

i	q_i	r_i	s_i	t_i
0	–	21	1	0
1	–	27	0	1
2	0	21	1	0
3	1	6	-1	1
4	3	3	4	-3
5	2	0	-10	13

Therefore $3 = \gcd(21, 27) = 21 \cdot 4 + 27 \cdot (-3)$.
i.e. $x = 4, y = -3$

```
typedef pair<int, int> pii; //#include utility

void update(pii& p, int q) {
  p = pii(p.second, p.first - q*p.second);
}

//returns gcd(r.first, r.second) and p is set so
//gcd(r.first, r.second) = p.first*r.first+p.second*r.second
int gcdex(pii r, pii& p) {
  pii s(1,0), t(0,1);
  while (r.second) {
    int q = r.first/r.second;
    update(r, q);
    update(s, q);
    update(t, q);
  }
  p = pii(s.first, t.first);
  return r.first;
} //can prove |p.first| <= r.second, |p.second| <= r.first

pii p;
int g = gcdex(pii(a,b), p);
//now g = gcd(a,b) = a*p.first + b*p.second
```

Problem to try:
UVa 10104 - Euclide Problem

Neat fact: the coefficients in the pair `p` you get from the Extended Euclidean algorithm discussed earlier will work, no modification necessary.

Applications

Modular Inverses

Recall $a \equiv b \mod m$ means $m|(a-b)$.

Given $a \in \mathbb{Z}$ and $m > 0$ find b such that $a \cdot b \equiv 1 \mod m$.

Applications

Modular Inverses
Recall $a \equiv b \mod m$ means $m|(a-b)$.

Given $a \in \mathbb{Z}$ and $m > 0$ find b such that $a \cdot b \equiv 1 \mod m$.

Cleanup: We usually like to think of $0 \leq a < m$. If $a \geq m$, then just compute $a \bmod m$. If $a < m$, then we have to be more careful.

Applications

Modular Inverses
Recall $a \equiv b \mod m$ means $m|(a-b)$.

Given $a \in \mathbb{Z}$ and $m > 0$ find b such that $a \cdot b \equiv 1 \mod m$.

Cleanup: We usually like to think of $0 \leq a < m$. If $a \geq m$, then just compute $a \bmod m$. If $a < m$, then we have to be more careful.

ISO Standard: In `c++`, if $a < 0$ and $m > 0$, then `a % m` is the "negative reminder closest to 0".

Example: `-17 % 5 == -2`.

Applications

Modular Inverses

Recall $a \equiv b \mod m$ means $m|(a-b)$.

Given $a \in \mathbb{Z}$ and $m > 0$ find b such that $a \cdot b \equiv 1 \mod m$.

Cleanup: We usually like to think of $0 \leq a < m$. If $a \geq m$, then just compute $a \bmod m$. If $a < m$, then we have to be more careful.

ISO Standard: In c++, if $a < 0$ and $m > 0$, then `a % m` is the "negative reminder closest to 0".

Example: `-17 % 5 == -2`.

```
//assumes m > 0, returns the residue of a mod m in [0, m-1]
int safe_mod(int a, int m) { return (a%m + m)%m; }
```

Recall, we are finding b such that $a \cdot b \equiv 1 \bmod m$ where $m > 0$.

If $\gcd(a, m) > 1$, impossible.

Recall, we are finding b such that $a \cdot b \equiv 1 \mod m$ where $m > 0$.

If $\gcd(a, m) > 1$, impossible.

Otherwise, use Euclid's extended algorithm to find c, d such that

$$a \cdot c + m \cdot d = \gcd(a, m) = 1.$$

So $a \cdot c \equiv 1 \mod m$.

Recall, we are finding b such that $a \cdot b \equiv 1 \mod m$ where $m > 0$.

If $\gcd(a, m) > 1$, impossible.

Otherwise, use Euclid's extended algorithm to find c, d such that

$$a \cdot c + m \cdot d = \gcd(a, m) = 1.$$

So $a \cdot c \equiv 1 \mod m$.

```
//assumes m > 0, returns an integer b in [1, m-1] such that
// a * b equiv 1 mod m
int modinv(int a, int m) {
  a = safe_mod(a, m); //ensure a >= 0
  pii p;
  assert(gcdex(pii(a,m), p) == 1);
  return safe_mod(p.first, m);
}
```

Problem to try:
UVa 11174 - Stand in a Line

Neat Fact:
If $p \not| a$ then a^{p-2} is the inverse of a modulo p, so we can compute a^{-1} with fast exponentiation as well!. This is by Fermat's little Theorem, briefly discussed near the end of these slides.

But I encourage you to solve the above problem using the gcd approach to get practice with it.

Linear Diophantine Equations

Given integers a, b, d, find integers x, y such that $ax + by = d$.

Linear Diophantine Equations

Given integers a, b, d, find integers x, y such that $ax + by = d$.

Idea: Find x', y' with $ax' + by' = \gcd(a, d)$. Scale x', y' by $d/\gcd(a, b)$. Some fussing to handle negatives.

Linear Diophantine Equations

Given integers a, b, d, find integers x, y such that $ax + by = d$.

Idea: Find x', y' with $ax' + by' = \gcd(a, d)$. Scale x', y' by $d / \gcd(a, b)$. Some fussing to handle negatives.

```
pii lin_diop(int a, int b, int d) {
   pii p;
   int g;
   g = gcdex(pii(abs(a), abs(b)), p);
   assert(d % g == 0); //impossible if d%g != 0

   //now abs(a)*p.first + abs(b)*p.second == g
   if (a < 0) p.first = -p.first;
   if (b < 0) p.second = -p.second;
   p.first *= d/g;
   p.second *= d/g;

   return p;
}
```

Problem to try:
UVa 10090 - Marbles

Note:
The function `lin_diop` only finds some solution x, y to $a \cdot x + b \cdot y = d$.
However, given one such solution and letting $g = \gcd(a, b)$,

$$\begin{aligned} x(t) &:= x + t \cdot \frac{b}{g} \\ y(t) &:= y - t \cdot \frac{a}{g} \end{aligned}$$

parameterizes all solutions as t ranges over integers. Use this parameterization to find the “min-cost” solution.

Chinese Remaindering

If a is an integer such that $a \equiv 4 \mod 15$ then we know $a \equiv 1 \mod 3$ and $a \equiv 4 \mod 5$.

That is, $15|(a-4)$ means surely $3|(a-1)$ and $5|(a-4)$.

Chinese Remaindering

If a is an integer such that $a \equiv 4 \bmod 15$ then we know $a \equiv 1 \bmod 3$ and $a \equiv 4 \bmod 5$.

That is, $15|(a-4)$ means surely $3|(a-1)$ and $5|(a-4)$.

What about the other way around? Given "target remainders" x, y modulo 3 and 5 respectively, is there some integer a such that $a \equiv x \bmod 3$ and $a \equiv y \bmod 5$?

Chinese Remaindering

If a is an integer such that $a \equiv 4 \mod 15$ then we know $a \equiv 1 \mod 3$ and $a \equiv 4 \mod 5$.

That is, $15 | (a - 4)$ means surely $3 | (a - 1)$ and $5 | (a - 4)$.

What about the other way around? Given "target remainders" x, y modulo 3 and 5 respectively, is there some integer a such that $a \equiv x \mod 3$ and $a \equiv y \mod 5$?

More generally

Theorem

Let m, n be such that $\gcd(m, n) = 1$. Then for any integers x, y there exists an integer a such that $a \equiv x \mod m$ and $a \equiv y \mod n$.

Idea: let m, n be the moduli and x, y the target remainders.

As $\gcd(m, n) = 1$, compute integers m', n' such that $m \cdot m' \equiv 1 \mod n$ and $n \cdot n' \equiv 1 \mod m$.

The answer is just $x \cdot m \cdot m' + y \cdot n \cdot n'$ (try reducing mod m and n to see why).

```
//assumes m,n > 0
//returns 0 <= a < m*n congruent to x mod m and y mod n
int chrem(int x, m, int y, int n) {
   int mi = modinv(m, n), ni = modinv(n, m);
   return safe_mod(x*m*mi + y*n*ni, m*n);
}
```

More generally, given moduli $m_1, \ldots, m_n > 0$ and target remainders $x_1, \ldots, x_n$ find an integer a such that $a \equiv x_i \mod m_i$ for each i.

Assumption: $\gcd(m_i, m_j) = 1$ for any $i \neq j$.

More generally, given moduli $m_1, \ldots, m_n > 0$ and target remainders $x_1, \ldots, x_n$ find an integer a such that $a \equiv x_i \mod m_i$ for each i.

Assumption: $\gcd(m_i, m_j) = 1$ for any $i \neq j$.

Base Case: if $n = 1$ just return $a = x$.

More generally, given moduli $m_1, \ldots, m_n > 0$ and target remainders $x_1, \ldots, x_n$ find an integer a such that $a \equiv x_i \mod m_i$ for each i.

Assumption: $\gcd(m_i, m_j) = 1$ for any $i \neq j$.

Base Case: if $n = 1$ just return $a = x$.
Inductive Step

- Inductively construct b congruent to $x_i \mod m_i$ for $i \leq j$.
- Solve the case $n = 2$ to find a congruent to $b \mod \prod_{i=1}^{j} m_i$ and congruent to $x_{j+1} \mod m_{j+1}$.

More generally, given moduli $m_1, \ldots, m_n > 0$ and target remainders $x_1, \ldots, x_n$ find an integer a such that $a \equiv x_i \mod m_i$ for each i.

Assumption: $\gcd(m_i, m_j) = 1$ for any $i \neq j$.

Base Case: if $n = 1$ just return $a = x$.

Inductive Step

- Inductively construct b congruent to $x_i \mod m_i$ for $i \leq j$.
- Solve the case $n = 2$ to find a congruent to $b \mod \prod_{i=1}^{j} m_i$ and congruent to $x_{j+1} \mod m_{j+1}$.

```
int chrem_multi(int *x, int *m, int n) {
  int a = x[0], mm = m[0];
  for (int j = 0; j+1 < n; ++j) {
    a = chrem(a, mm, x[j+1], m[j+1]);
    mm *= m[j+1];
  }
  return a;
}
```

Problems to Try:

[Open Kattis - chineseremainder]

[Open Kattis - generalchineseremainder]

We didn't discuss how to solve the latter. There is a solution iff $\gcd(m, n) | (a - b)$.

- Can you see why this is necessary?
- Now generalize the "equation" we wrote to solve the relatively-prime modulus case.

Quadratic Residues

Say a is a quadratic residue mod m if $b^2 \equiv a \bmod m$ for some b.

Quadratic Residues

Say a is a quadratic residue mod m if $b^2 \equiv a \bmod m$ for some b.

In general, this is hard to determine for a given a, m (would yield efficient factoring algorithms).

Quadratic Residues

Say a is a quadratic residue mod m if $b^2 \equiv a$ mod m for some b.

In general, this is hard to determine for a given a, m (would yield efficient factoring algorithms).

Easier if m is a **prime**.

Theorem (Euler's Criterion)

Let a, p be integers with p prime. Then a is a quadratic residue mod m if either $p|a$ or $a^{\frac{p-1}{2}} \equiv 1$ mod p.

Quadratic Residues

Say a is a quadratic residue mod m if $b^2 \equiv a$ mod m for some b.

In general, this is hard to determine for a given a, m (would yield efficient factoring algorithms).

Easier if m is a **prime**.

Theorem (Euler's Criterion)

Let a, p be integers with p prime. Then a is a quadratic residue mod m if either $p|a$ or $a^{\frac{p-1}{2}} \equiv 1$ mod p.

Use modular fast exponentiation to determine this (next meeting).

Quadratic Residues

Say a is a quadratic residue mod m if $b^2 \equiv a \bmod m$ for some b.

In general, this is hard to determine for a given a, m (would yield efficient factoring algorithms).

Easier if m is a **prime**.

Theorem (Euler's Criterion)

Let a, p be integers with p prime. Then a is a quadratic residue mod m if either $p|a$ or $a^{\frac{p-1}{2}} \equiv 1$ mod p.

Use modular fast exponentiation to determine this (next meeting).

Very Interesting: If a is a quadratic residue mod p, find such an integer b. This can be done efficiently, but is non-trivial.

Problem to Try:
Open Kattis - quadres

The description is interesting, but a lot more complicated than it needs to be :)

Tips

Primes are **very** dense: the number of primes $\leq n$ is $\sim \frac{n}{\ln n}$.

They also do not follow any obvious pattern. This is sometimes helpful in heuristic reasoning about why an algorithm runs quickly.

Theorem (Euler's Theorem)

Let $a, m \geq 1$ *be integers with* $\gcd(a, m) = 1$. *Then* $a^{\phi(m)} \equiv 1$ *mod* m.

Special case: Fermat's little theorem, $a^{p-1} \equiv 1 \bmod p$.

Theorem (Wilson's Theorem)

For a prime p, $(p-2)! \equiv 1$ *mod* p.

Missing Topics

Chinese remaindering when moduli are not relatively prime. Either there is no solution or it is unique modulo the least-common multiple of all moduli.

Discrete logarithms, finding integer solutions for integer quadratic equations.

Finding integer solutions to a system of integer linear equations.

Next Lecture

Tricks in Combinatorics and Arithmetic.

UVa - 11064 Number Theory (link)

UVa - `11064 Number Theory` (link)

Just compute $\phi(n) - 1$, the number of values $1 < m < n$ that are relatively prime to n.

UVa - `11064 Number Theory` (link)

Just compute $\phi(n) - 1$, the number of values $1 < m < n$ that are relatively prime to n.

n can be very big, so we can't sieve.

UVa - `11064 Number Theory` (link)

Just compute $\phi(n) - 1$, the number of values $1 < m < n$ that are relatively prime to n.

n can be very big, so we can't sieve.

Factor $n = \prod_i p_i^{a_i}$ and compute $\phi(p_i^{a_i}) = p_i^{a_i - 1}(p - 1)$ explicitly.

UVa - `11064 Number Theory` (link)

Just compute $\phi(n) - 1$, the number of values $1 < m < n$ that are relatively prime to n.

n can be very big, so we can't sieve.

Factor $n = \prod_i p_i^{a_i}$ and compute $\phi(p_i^{a_i}) = p_i^{a_i - 1}(p - 1)$ explicitly.

Treat $n = 0, 1$ with care!

UVa - 543 Goldbach's Conjecture (link)

UVa - 543 Goldbach's Conjecture (link)

Well, Goldbach's conjecture isn't disproven yet so it is unlikely a correct solution will not find a solution!

UVa - 543 Goldbach's Conjecture (link)

Well, Goldbach's conjecture isn't disproven yet so it is unlikely a correct solution will not find a solution!

Heuristic Reasoning: Primes are dense (about $\frac{n}{\ln n}$ primes are $\leq n$) and don't follow any simple pattern so for many primes $p \leq n$ it should be that $n - p$ is also a prime.

UVa - 543 Goldbach's Conjecture (link)

Well, Goldbach's conjecture isn't disproven yet so it is unlikely a correct solution will not find a solution!

Heuristic Reasoning: Primes are dense (about $\frac{n}{\ln n}$ primes are $\leq n$) and don't follow any simple pattern so for many primes $p \leq n$ it should be that $n - p$ is also a prime.

So just run through all p in increasing order and check that $p, n - p$ are primes, it won't take long before you get a hit. Sieve beforehand to make this check efficient.

UVa - 543 Goldbach's Conjecture (link)

Well, Goldbach's conjecture isn't disproven yet so it is unlikely a correct solution will not find a solution!

Heuristic Reasoning: Primes are dense (about $\frac{n}{\ln n}$ primes are $\leq n$) and don't follow any simple pattern so for many primes $p \leq n$ it should be that $n - p$ is also a prime.

So just run through all p in increasing order and check that $p, n - p$ are primes, it won't take long before you get a hit. Sieve beforehand to make this check efficient.

While the worst case per input seems to be $\Theta(n)$, you will **never** see this!

UVa - 10831 Gerg's Cake (link)

UVa - `10831 Gerg's Cake` (link)

Suppose we cut the cake into $n \times n$ squares and each present party guest gets d pieces.

UVa - `10831 Gerg's Cake` (link)

Suppose we cut the cake into $n \times n$ squares and each present party guest gets d pieces.

So $n^2 = d \cdot p + a$.

UVa - `10831 Gerg's Cake` (link)

Suppose we cut the cake into $n \times n$ squares and each present party guest gets d pieces.

So $n^2 = d \cdot p + a$.

We already discussed what to do:

UVa - 10831 Gerg's Cake (link)

Suppose we cut the cake into $n \times n$ squares and each present party guest gets d pieces.

So $n^2 = d \cdot p + a$.

We already discussed what to do:
Quadratic Residues. Just use Euler's criteria.

UVa - 10831 Gerg's Cake (link)

Suppose we cut the cake into $n \times n$ squares and each present party guest gets d pieces.

So $n^2 = d \cdot p + a$.

We already discussed what to do:
Quadratic Residues. Just use Euler's criteria.

Be careful about overflow in the calculations.

UVa - 10692 Huge Mods (link)

UVa - 10692 Huge Mods (link)

We want to compute $a_1^{a_2^{a_3^{\cdot^{\cdot^{\cdot}}}}} \bmod m$.

UVa - 10692 Huge Mods (link)

We want to compute $a_1^{a_2^{a_3^{\cdot^{\cdot^{\cdot}}}}} \bmod m$.

Let $k := a_2^{a_3^{\cdot^{\cdot^{\cdot}}}} \bmod \phi(m)$.

If $\gcd(a_1, m) = 1$, it suffices to recursively compute $k \bmod \phi(m)$ and then $a^k \bmod m$.

UVa - 10692 Huge Mods (link)

We want to compute $a_1^{a_2^{a_3^{\cdot^{\cdot^{\cdot}}}}} \bmod m$.

Let $k := a_2^{a_3^{\cdot^{\cdot^{\cdot}}}} \bmod \phi(m)$.

If $\gcd(a_1, m) = 1$, it suffices to recursively compute $k \bmod \phi(m)$ and then $a^k \bmod m$.

In general, if $k \geq 14$ then $a^{[(k-14) \bmod \phi(m)]+14} \bmod m$ suffices:
Reason: because $14 \geq \log_2 m$ so any $p|(a_1, m)$ has $p^{14} \equiv 0 \bmod m$.

UVa - 10692 Huge Mods (link)

We want to compute $a_1^{a_2^{a_3^{\cdot^{\cdot^{\cdot}}}}}$ mod m.

Let $k := a_2^{a_3^{\cdot^{\cdot^{\cdot}}}}$ mod $\phi(m)$.

If $\gcd(a_1, m) = 1$, it suffices to recursively compute k mod $\phi(m)$ and then a^k mod m.

In general, if $k \geq 14$ then $a^{[(k-14) \bmod \phi(m)]+14}$ mod m suffices:
Reason: because $14 \geq \log_2 m$ so any $p|(a_1, m)$ has $p^{14} \equiv 0$ mod m.

So write an ad-hoc function to check if $k \geq 14$ or not. If not, then just explicitly calculate a^k mod m. If so, do as above.