Algorithmic Problem Solving
Le 12 – Number Theory

Fredrik Heintz
Dept of Computer and Information Science
Linköping University

Outline

- Modular arithmetic (Lab 3.5)
- Chinese reminder theorem (Lab 3.6-3.7)
- Primes and Prime testing (Lab 3.8)

Modular Arithmetic (Z_n)

Definition

$a \equiv b \pmod{n} \Leftrightarrow n | (b - a)$, alternatively $a = qn + b$

Zn for an integer n is an equivalence relation

Definition (An equivalence class mod n)

$[a] = \{x \mid x \equiv a \pmod{n}\} = \{a + qn \mid q \in \mathbb{Z}\}$

Arithmetic can be done with these equivalence classes (Lab 3.5)

Modular Inverse

- What does it mean to calculate x / y mod n?
- Reformulate as $x \cdot y^{-1} \bmod n$
- That is, we are looking for y^{-1}, such that $y \cdot y^{-1} \bmod n = 1$ holds
- Recall Euclid's algorithm for greatest common divisor:

```
ull gcd(ull a, ull b) {
    ull t;
    while (b) t = a, a = b,b = t%b;
    return a;
}
```

- And the extended Euclidean algorithm, that finds x, y such that $ax + by = gcd(a,b)$:

```
void exeuclid(ll a, ll b, ll *x ,ll *y) {
    if (!b) *x = 1, *y = 0;
    else exeuclid(b, a%b, y, x),*y -=*x * (a/b);
}
```

Chinese Remainder Theorem (Lab 3.6-3.7)

Theorem 2.9: (Chinese Remainder Theorem) Let $m_1, m_2, \ldots, m_n$ be pairwise relatively prime positive integers and let $b_1, b_2, \ldots, b_n$ be any integers. Then the system of linear congruences in one variable given by

$$x \equiv b_1 \bmod m_1$$

$$x \equiv b_2 \bmod m_2$$

$$\vdots$$

$$x \equiv b_n \bmod m_n$$

has a unique solution modulo $m_1 m_2 \cdots m_n$.

Proof: We first construct a solution to the given system of linear congruences in one variable. Let $M = m_1 m_2 \cdots m_n$ and, for $i = 1, 2, \ldots, n$, let $M_i = M/m_i$. Now $(M_i, m_i) = 1$ for each i. (Why?) So $M_i x_i \equiv 1 \bmod m_i$ has a solution for each i by Corollary 2.8. Form

$$x = b_1 M_1 x_1 + b_2 M_2 x_2 + \cdots + b_n M_n x_n$$

Note that x is a solution of the desired system since, for $i = 1, 2, \ldots, n$,

$$\begin{aligned} x &= b_1 M_1 x_1 + b_2 M_2 x_2 + \cdots + b_i M_i x_i + \cdots + b_n M_n x_n \\ &\equiv 0 + 0 + \cdots + b_i + \cdots + 0 \bmod m_i \\ &\equiv b_i \bmod m_i \end{aligned}$$

It remains to show the uniqueness of the solution modulo M. Let x' be another solution to the given system of linear congruences in one variable. Then, for all i, we have that $x' \equiv b_i \bmod m_i$; since $x \equiv b_i \bmod m_i$ for all i, we have that $x \equiv x' \bmod m_i$ for all i, or, equivalently, $m_i \mid x - x'$ for all i. Then $M \mid x - x'$ (why?), from which $x \equiv x' \bmod M$. The proof is complete. ■

Note that the proof of the Chinese Remainder Theorem shows the existence and uniqueness of the claimed solution modulo M by actually *constructing* this

Primes

- First prime and the only even prime: 2
  - First 10 primes: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
- Primes in range:
  - 1 to 100: 25 primes
  - 1 to 1,000: 168 primes
  - 1 to 7,919: 1,000 primes
  - 1 to 10,000: 1,229 primes
- Largest prime in signed 32-bit int = 2,147,483,647

Prime Testing (Lab 3.8)

- Algorithms for testing if N is prime: isPrime(N)
  - First try: check if N is divisible by i ∈ [2, …, N-1]?
  - O(N)
- Improved 1: Is N divisible by i ∈ [2, …, sqrt(N)]?
  - O(sqrt(N))
- Improved 2: Is N divisible by i ∈ [3, 5, …, sqrt(N)]
  - One test for i=2, no need to test other even numbers
  - O(sqrt(N)/2)=O(sqrt(N))
- Improved 3: Is N divisible by i primes ≤ sqrt(N)
  - O(π(sqrt(N)) = O(sqrt(N)/log(sqrt(N)))
  - π(M) = number of primes up to M
  - For this, we need smaller primes beforehand

Prime Generation

- Generate primes between [0, …, N]:
  - Use bitset of size N, set all true except index 0 and 1
  - Start from i=2 until k*I > N
    - If bitset at index i is on, cross all multiples of i (i.e. turn off bit at index I
  - Finally, whatever not crossed are primes
- Example:
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, 77, …
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, 77, …
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, 77, …
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, 77, …
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, 77, …

Prime Testing and Generation


```
#include <bitset>          // compact STL for Sieve, better than vector<bool>!
ll _sieve_size;                  // ll is defined as: typedef long long ll;
bitset<10000010> bs;                  // 10^7 should be enough for most cases
vi primes;                  // compact list of primes in form of vector<int>

void sieve(ll upperbound) {         // create list of primes in [0..upperbound]
  _sieve_size = upperbound + 1;                // add 1 to include upperbound
  bs.set();                                                // set all bits to 1
  bs[0] = bs[1] = 0;                                   // except index 0 and 1
  for (ll i = 2; i <= _sieve_size; i++) if (bs[i]) {
    // cross out multiples of i starting from i * i!
    for (ll j = i * i; j <= _sieve_size; j += i) bs[j] = 0;
    primes.push_back((int)i);         // add this prime to the list of primes
} }                                              // call this method in main method

bool isPrime(ll N) {                   // a good enough deterministic prime tester
  if (N <= _sieve_size) return bs[N];                     // O(1) for small primes
  for (int i = 0; i < (int)primes.size(); i++)
    if (N % primes[i] == 0) return false;
  return true;                      // it takes longer time if N is a large prime!
}                     // note: only work for N <= (last prime in vi "primes")^2
```

Outline

- Modular arithmetic (Lab 3.5)
- Chinese reminder theorem (Lab 3.6-3.7)
- Primes and Prime testing (Lab 3.8)