Algorithmic Problem Solving
AAPS20 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 $a = qn + b$

Z_n 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 $x / y \bmod n$?
- 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
- This can be reformulated as $y \cdot y^{-1} + n \cdot x = 1$, which can be solved using the extended Euclidean algorithm if $gcd(y,n)=1$.

```
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);
}
```

- There is only a modular inverse if y and n are relative prime
- For example y=2 mod n=4 does not have a solution

Chinese Remainder Theorem (Lab 3.6-3.7)

- Suppose that $m_1, m_2, \ldots, m_r$ are pairwise relatively prime positive integers, and let $a_1, a_2, \ldots, a_r$ be integers.

  Then the system of congruences,
  $x \equiv a_i \pmod{m_i}$ for $1 \le i \le r$,
  has a unique solution modulo $M = m_1 \times m_2 \times \ldots \times m_r$,
  which is given by:
  $x \equiv a_1 M_1 y_1 + a_2 M_2 y_2 + \ldots + a_r M_r y_r \pmod{M}$,
  where $M_i = M/m_i$ and $y_i \equiv (M_i)^{-1} \pmod{m_i}$ for $1 \le i \le r$.

CRT Example

- Find the smallest multiple of 10 which has remainder 2 when divided by 3, and remainder 3 when divided by 7.
- We are looking for a number which satisfies the congruences,
  $x \equiv 2 \bmod 3$,
  $x \equiv 3 \bmod 7$,
  $x \equiv 0 \bmod 2$ and
  $x \equiv 0 \bmod 5$.
- Since, 2, 3, 5 and 7 are all relatively prime in pairs, the Chinese Remainder Theorem tells us that there is a unique solution modulo 210 (= 2×3×5×7).
- We calculate the Mi's and yi's as follows:
  $M_2 = 210/2 = 105$; $y_2 \equiv (105)^{-1} \pmod 2 = 1$
  $M_3 = 210/3 = 70$; $y_3 \equiv (70)^{-1} \pmod 3 = 1$
  $M_5 = 210/5 = 42$; $y_5 \equiv (42)^{-1} \pmod 5 = 3$ and
  $M_7 = 210/7 = 30$; $y_7 \equiv (30)^{-1} \pmod 7 = 4$.
- So, $x \equiv 0(M_2y_2) + 2(M_3y_3) + 0(M_5y_5) + 3(M_7y_7)$
  $\equiv 0 + 2(70)(1) + 0 + 3(30)(4)$
  $\equiv 140 + 360$
  $\equiv 500 \bmod 210$
  $\equiv 80$.

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 $i \in [2, \ldots, N-1]$?
  - O(N)
- Improved 1: Is N divisible by $i \in [2, \ldots, \text{sqrt}(N)]$?
  - O(sqrt(N))
- Improved 2: Is N divisible by $i \in [3, 5, \ldots, \text{sqrt}(N)]$
  - 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)
  - $O(\pi(\text{sqrt}(N)) = O(\text{sqrt}(N)/\log(\text{sqrt}(N)))$
  - $\pi(M)$ = 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)