Algorithmic Problem Solving

Le 9 Number Theory

Herman Appelgren
Dept of Computer and Information Science
Linköping University

Outline

- ~~Exercise 9: Strings II~~
  - ~~A: Suffix Array Re-construction~~
  - ~~B: Code Theft~~
  - ~~B: Messages from Outer Space~~
  - ~~C: Life Forms~~
- Number Theory
  - Primes (Lab 3.8)
  - Greatest Common Divisor and Least Common Multiple (Lab 3.4)
  - Modular Arithmetic (Lab 3.5)
  - Chinese Remainder Theorem (Lab 3.6-3.7)

Primes

- **Definition:** $x > 1$ is a prime if it isn't divisible by any integer other than 1 and itself.
  - Example: The first 8 primes are 2, 3, 5, 7, 11, 13, 17, 19.
  - Example: 2,147,483,647 is the largest prime that fits in an int32.
- **Fundamental theorem of arithmetic:** Each $x > 1$ can be uniquely represented as a product of primes.
  - This representation is called the *prime factorization* of x.
  - Example: 35 = 5*7, 54 = 2*3^3, 7 = 7
- Algorithms for testing if N is prime.
  - Naïve $O(N)$ algorithm: Try to divide N by 2, 3, ..., N-1.
  - Improve to $O(\sqrt{N})$ by noting that if N isn't prime, at least one factor must be $\sqrt{N}$ or less.
  - Improve to $O(\frac{\sqrt{N}}{\log(N)})$ by only dividing by primes. Requires that all primes less than $\leq \sqrt{N}$ are known.

Sieve of Eratosthenes (Lab 3.8)

- Efficient algorithm for finding all primes ≤ N.
  - Initialize a bitset is_prime to true for all integers except 0 and 1.
  - For each integer i, if is_prime[i], add it to our list of primes and set is_prime[ik] = false for all k.
- Time complexity $O(N \log N)$
  - If is_prime[p] = true, we have to mark $\frac{N}{p}$ integers as not prime.
  - The number of iterations of the inner loop is
    $$\frac{N}{2} + \frac{N}{3} + \frac{N}{5} + \cdots < N\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N}\right)$$
  - We recognize the sum of the harmonic series, which is in $O(\log N)$, so the time complexity is in $O(N + N \log N) = O(N \log N)$.
- Multiple optimizations possible
  - If we don't need the primes list, we only have to iterate $i = 2, 3, \ldots, \sqrt{N}$.
  - We can start marking with $k = i$, since all other multiples will already be marked.

GCD and LCM (Lab 3.4)

- Greatest Common Divider
  - $\gcd(a, b)$ is the largest value g such that $a = mg$ and $b = ng$ for some $m, n \in \mathbb{Z}$.
  - Note than $\gcd(m, n) = 1$, otherwise $g * \gcd(m, n)$ would be a greater common divider.
  - Example: $\gcd(6, 9) = 3$, $\gcd(33, 121) = 11$, $\gcd(7, 15) = 1$.
  - If $\gcd(a, b) = 1$ we say that a and b are *coprime* (sometimes *relatively prime* or *mutually prime*), since they don't share any prime factors.
- Least Common Multiple
  - $\text{lcm}(a, b)$ is the smallest value l such that $l = ma$ and $l = nb$ for some $m, n \in \mathbb{Z}$.
  - As a direct consequence of the fundamental theorem of arithmetic, $\text{lcm}(a, b) = ab / \gcd(a, b)$
  - Example: $\text{lcm}(12, 9) = 36$, $\text{lcm}(2, 3) = 6$, $\text{lcm}(5, 10) = 10$
- Both are useful when implementing rational arithmetic in Lab 3.4.

Euclidean Algorithm

- **Theorem:** $\gcd(a, b) = \gcd(a - kb, b)$ for all $k \in \mathbb{Z}$.
  - Proof: Let c be a common divisor of a and b. Then $a = mc$ and $b = nc$, so $a - kb = mc - knc = (m - kn)c$ and c is a divisor of $a - kb$. Similarly, if d is a common divisor of $a - kb$ and b, then it must also divide a. Consequently a, b and $a - kb$ have the same common divisors, and in particular the same gcd.
- This observation is at the core of the *Euclidean Algorithm*:
  - Without loss of generality, assume $a \geq b$.
  - If $b = 0$ return a
  - Otherwise $\gcd(a, b) = \gcd(kb + a\%b, b) = \gcd(b, a\%b)$
- Example:
  - $\gcd(175, 145) = \gcd(1 * 145 + 30, 145) =$
    $\gcd(145, 30) = \gcd(4 * 30 + 25, 30) =$
    $\gcd(30, 25) = \gcd(1 * 25 + 5, 30) =$
    $\gcd(25, 5) = \gcd(5 * 5 + 0, 5) =$
    $\gcd(5, 0) = 5$

Modular Arithmetic (Lab 3.5)

- **Definition:** a is *congruent* with b modulo m if $a + km = b$ for some $k \in \mathbb{Z}$.
  - Equivalently: m divides $a - b$, i.e. $km = a - b$ for some $k \in \mathbb{Z}$.
  - Symbolically: $a \equiv_m b$ or $a \equiv b \ (mod\ m)$.
- Arises naturally when modelling cyclic behavior, but is also central in cryptography and other fields.
- The remainder operator %
  - $a\%m$ is the remainder of a divided by m, i.e. the unique number $0 \leq r < m$ such that $a \equiv_m r$.
  - Note that this definition is different from most programming languages, where $-m \leq a\%m < 0$ if $a < 0$.
  - $a = a \,//\, m * a + a\%\text{m}$ where // is integer division.
- **Theorem:** If $a \equiv_m b$, then $a\%m = b\%m$.

Modular Arithmetic – Add/Sub

- **Theorem:** If $x \equiv_m a$ and $y \equiv_m b$, then $x + y \equiv_m a + b$.
  - Proof: $x + km = a$ and $y + jm = b$, so $x + y + (k + j)m = a + b$.
  - Similarly for subtraction.
- Thus $(a \pm b)\%m = (a\%m \pm b\%m)\%m$.
  - By computing the remainder of the operands, we avoid intermediary values larger than $2m$, which reduces the risk for overflow.

Modular Arithmetic – Mult

- **Theorem:** If $x \equiv_m a$ and y $\equiv_m b$ then $xy \equiv_m ab$.
  - Proof: $(x + km)(y + jm) = xy + (ky + jx + kjm)m = ab$.
- We can thus use the same trick for multiplication, but the intermediary value is now on the order of m^2. Can we do better?

Modular Arithmetic – Mult

- Idea: Use that $ab = a(2k + d) = 2ak + ad$ where $k \in \mathbb{Z}$ and $d \in \{0, 1\}$.
- Example: What is $(64 * 25)\%37$?
  - $64 * 25 \equiv_{37} 64\%37 * 25 = 27 * 25$
    This reduces the product, but it would still e.g. overflow an int8, even though the final remainder will fit nicely.
  - We recursively partition 25:
    $$27 * 25 = 27 * (2 * 12 + 1) = 2 * 27 * 12 + 27$$
    $$27 * 12 = 27 * (2 * 6 + 0) = 2 * 27 * 6$$
    $$27 * 6 = 27 * (2 * 3 + 0) = 2 * 27 * 3$$
    $$27 * 3 = 27 * (2 * 1 + 1) = 2 * 27 + 27$$
    $$2 * 27 = 54 \equiv_{37} 17$$
  - Now we backtrack and compute remainder along the way
    $27 * 3 = 2 * 27 + 27 \equiv_{37} 17 + 27 = 44 \equiv_{37} 7$
    $27 * 6 = 2 * 27 * 3 \equiv_{37} 2 * 7 = 14$
    $27 * 12 = 2 * 27 * 6 \equiv_{37} 2 * 14 = 28$
    $27 * 25 = 2 * 27 * 12 + 27 \equiv_{37} 2 * 28 + 27 = 56 + 27 \equiv_{37} 19 + 27$
    $= 46 \equiv_{37} 9$
- Using this algorithm, no intermediary values are larger than $2m$, i.e. same as for addition/subtraction.

Modular Arithmetic – Exp

- Modular exponentiation $a^b \% m$ isn't included in the lab, but is still a useful algorithm.
- First attempt: Apply modular multiplication b times.
  - Pro: No intermediary values above $2m$.
  - Con: $O(b)$ time complexity.
- Improvement: Binary Exponentiation
  - We use that $b = b_0 2^0 + b_1 2^1 + \cdots + b_n 2^n$ where $b_i \in \{0, 1\}$ and $n = \lfloor \log_2 b \rfloor$.
  - This gives us $a^b = a^{b_0 2^0 + \cdots + b_n 2^n} = a^{b_0 2^0} * a^{b_1 2^1} * \cdots * a^{b_n 2^n}$. In other words, a^b is the product of n factors, each being the square of the one before, where b_i tells us if factor i should be included.
  - We thus have $\log_2 b$ factors, each taking $\log_2 m$ time to compute using modular multiplication, and similar to multiply them with each other.
  - Since we only use modular multiplication repeatedly, we still don't need intermediary values larger than $2m$.

Modular Arithmetic – Inverse

- How about division?
  - Straight-forward definition doesn't work, since $\frac{1}{a}$ is not an integer unless $a = 1$.
  - Instead, we define $\frac{1}{a} = a^{-1}$ such that $a * a^{-1} \equiv_m 1$.
  - Note that a^{-1} depends on both a and m!
- The definition means that $a * a^{-1} + km = 1$ for some $k \in \mathbb{Z}$.
  - This is a *diofantine equation* with unknowns a^{-1} and k.

Extended Euclidean Algorithm

- Diofantine equations are of the form $ax + by = c$ where a, b and c are constants and x and y are unknown. All values involved are integers.
- There are either no solutions or an infinite number of solutions.
  - No solutions if $\gcd(a, b)$ doesn't divide c, since both sides must be divisible by the same numbers.
  - If x_0, y_0 is a solution, then $x_0 + bk/\gcd(a,b)$, $y_0 - ak/\gcd(a,b)$ are also solutions for all $k \in \mathbb{Z}$, since $a(x_0 + bk/\gcd(a,b)) + b(y_0 - ak/\gcd(a,b)) = ax_0 + \frac{abk - abk}{\gcd(a,b)} + by_0 = ax_0 + by_0 = c$.
  - If we additionally require either x or y fall within a range (typically $0 \le x < b/\gcd(a,b)$), then the solution is unique.
- The *Extended Euclidean Algorithm* solves diofantine equations.
  - Without loss of generality, assume $a \ge b$.
  - if $b = 0$
    if a divides c return $x = c\ //\ a, y = 0$
    else return Impossible
    else
    $x_0, y_0 = extended\_euclid(b, a\%b)$
    return $x = y_0, y = x_0 - y_0 * a\ //\ b$

Extended Euclidean Algorithm

- Example: $175x + 145y = 15$
  - Call recursively
    extended_euclid(175, 145)
    extended_euclid(145, 30)
    extended_euclid(30, 25)
    extended_euclid(25, 5)
    extended_euclid(5, 0): 5 divides 15, so $x = 3, y = 0 \Rightarrow 5 * 3 + 0 * 0 = 15$
  - Reconstruct solution when backtracking
    extended_euclid(25, 5): $x = 0, y = 3 - 0 = 3 \Rightarrow 25 * 0 + 5 * 3 = 15$
    extended_euclid(30, 25): $x = 3, y = 0 - 3 * 30 \,//\, 25 = -3 \Rightarrow$
    $\Rightarrow 30 * 3 + 25 * (-3) = 90 - 75 = 15$
    extended_euclid(145, 30): $x = -3, y = 3 - (-3) * 29 \,//\, 6 = 15 \Rightarrow$
    $\Rightarrow 145 * (-3) + 30 * 15 = -435 + 450 = 15$
    extended_euclid(175, 145): $x = 15, y = -3 - 15 * 175 \,//\, 145 = -18 \Rightarrow$
    $\Rightarrow 175 * 15 + 145 * (-18) = 2625 - 2610 = 15$
- All solutions are given by $x = 15 + 29k, y = -18 - 35k$ for $k \in \mathbb{Z}$.
  - E.g. $k = 1 \Rightarrow x = 44, y = -53 \Rightarrow$
    $\Rightarrow 175x + 145y = 175 * 44 + 145 * (-53) = 7{,}700 - 7{,}685 = 15$

Systems of Congruences (Lab 3.7)

- **Problem:** Find all x that satisfy the two congruences $x \equiv a \ (mod\ m)$ and $x \equiv b \ (mod\ n)$.
  - Equivalently, $x + jm = a$ and $x + kn = b$ for some $j, k \in \mathbb{Z}$.
  - Subtract them to get $jm - kn = a - b$.
  - This is a diofantine equation with j, k unknown. It is solvable if $\gcd(m, n)$ divides $a - b$, otherwise no solution exists.
  - Find j, k using Extended Euclidean, then $x = a - jm$.
  - j is unique modulo $\frac{n}{\gcd(m,n)}$, so x is unique modulo $\frac{mn}{\gcd(m,n)} = \operatorname{lcm}(m, n)$. The two congruences taken together are therefore equivalent to $x \equiv a - jm \ (mod\ \operatorname{lcm}(m, n))$.
- For more than two congruences, solve them pairwise and “compress” them using the solution above.

Chinese Remainder Theorem (Lab 3.6)

- **The Chinese Remainder Theorem:** Given a system of congruences $x \equiv a_i \ (mod\ m_i), i = 1 \ldots n$, where $\gcd(m_i, m_j) = 1$ when $i \neq j$. Then one solution is given by $x = \sum a_i M_i y_i$ where $M = \prod m_i$, $M_i = M/m_i$ and $y_i = M_i^{-1}(mod\ m_i)$.
  - Proof: All except the ith term of x contains a factor m_i, and $M_i y_i = 1 \ (mod\ m_i)$. Consequently $x \equiv 0 + \cdots + a_i M_i y_i + 0 + \cdots + 0 \equiv a_i * 1 \equiv a_i \ (mod\ m_i)$.
  - The solution is unique modulo M (proof omitted).
  - This is an important special case of the general approach presented before. Since $\gcd(m_i, m_j) = 1$ it is always solvable and

$$\text{lcm}(m_i, m_j) = m_i m_j$$

Chinese Remainder Theorem

- Example: $x \equiv_2 0$, $x \equiv_3 2$, $x \equiv_5 0$ and $x \equiv_7 3$.
  - $M = 2 * 3 * 5 * 7 = 210$

$$y_1 = (105)^{-1}\ (mod\ 2) = 1$$
$$y_2 = (70)^{-1}\ (mod\ 3) = 1$$
$$y_3 = (42)^{-1}\ (mod\ 5) = 3$$
$$y_4 = (30)^{-1}\ (mod\ 7) = 4$$

  - $x = 0 * \frac{210}{2} * 1 + 2 * \frac{210}{3} * 1 + 0 * \frac{210}{5} * 3 + 3 * \frac{210}{7} * 4 =$

$$= 0 + 140 + 0 + 360 = 500 \equiv_M 80$$

  - Verify: $80 = 2 * 40$, $80 = 3 * 78 + 2$, $80 = 16 * 5$, $80 = 11 * 7 + 3$

Summary

- Exercise 9: Strings II
  - A: Suffix Array Re-construction
  - B: Code Theft
  - B: Messages from Outer Space
  - C: Life Forms
- Number Theory
  - Primes (Lab 3.8)
  - Greatest Common Divisor and Least Common Multiple (Lab 3.4)
  - Modular Arithmetic (Lab 3.5)
  - Chinese Remainder Theorem (Lab 3.6-3.7)