Advanced Algorithmic Problem Solving

Le 6 – Math and Search

Fredrik Heintz
Dept of Computer and Information Science
Linköping University

Outline

- Arithmetic (lab 3.1 and 3.2)
- Solving linear equation systems (lab 3.3 and 3.4)
- Chinese reminder theorem (lab 3.5 and 3.6)
- Prime numbers and factorization (lab 3.7 and 3.8)
- Heuristic Search (exercise 4)

Arithmetic

- Range of default integer data types (C++)
  - unsigned int = unsigned long: 232 (9-10 digits)
  - unsigned long long: 264 (19-20 digits)
- How to represent 777!
- Operations on Big Integer
  - Basic: add, subtract, multiply, divide, etc
  - Use “high school method”

```
 1  ← carry
218
 45
--- +
263
```

```
  218
   45
  --- x
 1090   (218*5)
 872    (218*4)*10
----- +
 9810
```

Arithmetic

- Greatest Common Divisor (Euclidean Algorithm)
  - GCD(a, 0) = a
  - GCD(a, b) = GCD(b, a mod b)
  - `int gcd(int a, int b) { return (b == 0 ? a : gcd(b, a % b)); }`
- Least Common Multiplier
  - LCM(a, b) = a*b / GCD(a, b)
  - `int lcm(int a, int b) { return (a / gcd(a, b) * b); }`
    - // Q: why we write the lcm code this way?
- GCD/LCM of more than 2 numbers:
  - GCD(a, b, c) = GCD(a, GCD(b, c))
- Find d, x, y such that d = ax + by and d = GCD(a,b) (Extended Euclidean Algorithm)
  - EGCD(a,0) = (a,1,0)
  - EGCD(a,b)
    - (d',x',y') = EGCD(b, a mod b)
    - (d,x,y) = (d',y',x' – a/b*y')

Arithmetic

- Representing rational numbers.
  - Pairs of integers a,b where GCD(a,b)=1.
- Representing rational numbers modulo m.
  - The only difficult operation is inverse, $ax = 1 \pmod{m}$, where an inverse exists if and only if a and m are co-prime (gcd(a,m)=1).
  - Can be found using the Extended Euclidean Algorithm
    $ax = 1 \pmod{m} \Rightarrow ax - 1 = qm \Rightarrow ax - qm = 1$
    $(d, x, y) = \text{EGCD}(a,m) \Rightarrow x$ is the solution iff $d = 1$.

Systems of Linear Equations

A system of linear equations can be presented in different forms

$$\left.\begin{array}{l} 2x_1 + 4x_2 - 3x_3 = 3 \\ 2.5x_1 - x_2 + 3x_3 = 5 \\ x_1 \qquad\qquad - 6x_3 = 7 \end{array}\right\} \Leftrightarrow \begin{bmatrix} 2 & 4 & -3 \\ 2.5 & -1 & 3 \\ 1 & 0 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 7 \end{bmatrix}$$

Standard form — Matrix form

Solutions of Linear Equations

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \text{is a solution to the following equations :}$$

$$x_1 + x_2 = 3$$

$$x_1 + 2x_2 = 5$$

Solutions of Linear Equations

- A set of equations is **inconsistent** if there exists no solution to the system of equations:

$$x_1 + 2x_2 = 3$$

$$2x_1 + 4x_2 = 5$$

These equations are inconsistent

Solutions of Linear Equations

- Some systems of equations may have **infinite number of solutions**

$$x_1 + 2x_2 = 3$$

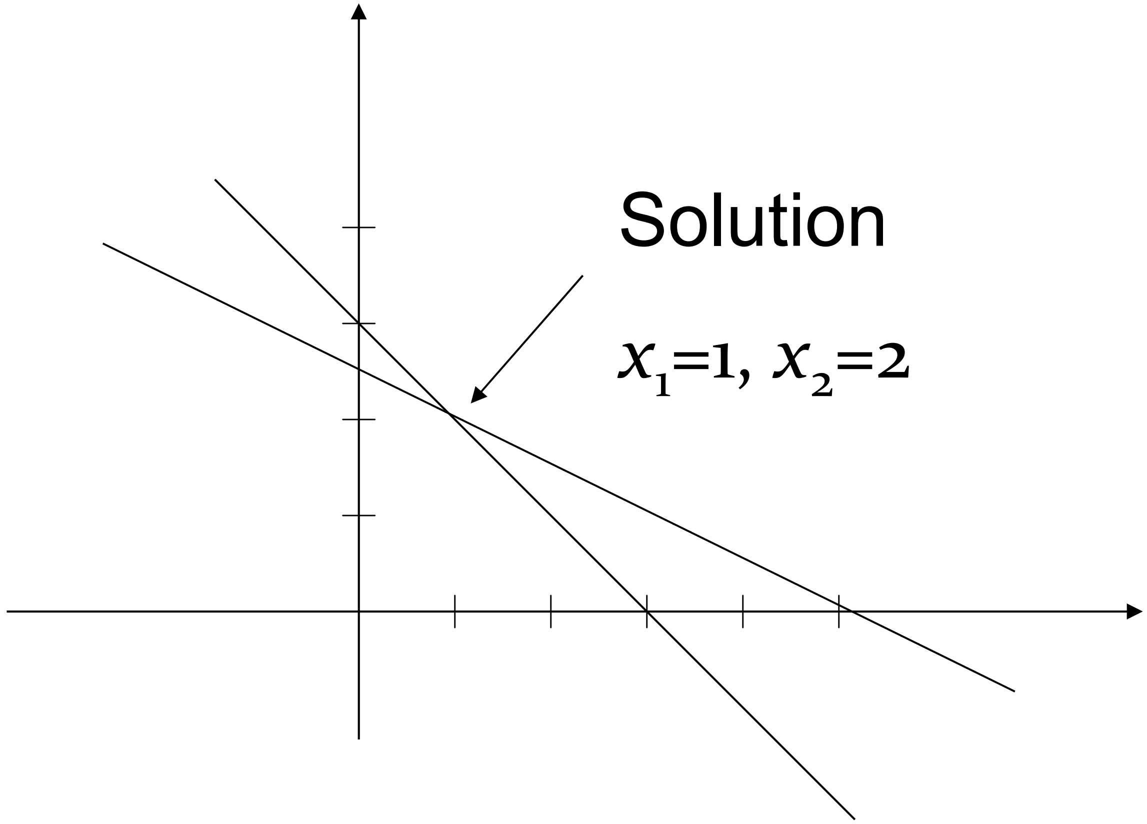
$$2x_1 + 4x_2 = 6$$

have infinite number of solutions

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a \\ 0.5(3-a) \end{bmatrix} \text{ is a solution for all } a$$

Graphical Solution of Systems of Linear Equations

$$x_1 + x_2 = 3$$
$$x_1 + 2x_2 = 5$$



Cramer's Rule is Not Practical

Cramer's Rule can be used to solve the system

$$x_1 = \frac{\begin{vmatrix} 3 & 1 \\ 5 & 2 \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ 1 & 2 \end{vmatrix}} = 1, \qquad x_2 = \frac{\begin{vmatrix} 1 & 3 \\ 1 & 5 \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ 1 & 2 \end{vmatrix}} = 2$$

Cramer's Rule is not practical for large systems.

To solve N by N system requires $(N+1)(N-1)N!$ multiplications.

To solve a 30 by 30 system, 2.38×10^{35} multiplications are needed.

It can be used if the determinants are computed in efficient way

Naive Gaussian Elimination

- The method consists of two steps:
  - **Forward Elimination**: the system is reduced to upper triangular form. A sequence of elementary operations is used.
  - **Backward Substitution**: Solve the system starting from the last variable.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \Rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}' & a_{23}' \\ 0 & 0 & a_{33}' \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2' \\ b_3' \end{bmatrix}$$

Elementary Row Operations

- Adding a multiple of one row to another
- Multiply any row by a non-zero constant

Example: Forward Elimination

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ 26 \\ -19 \\ -34 \end{bmatrix}$$

Part 1 : Forward Elimination

Step1 : Eliminate x_1 from equations 2, 3, 4

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -27 \\ -18 \end{bmatrix}$$

Example: Forward Elimination

Step2 : Eliminate x_2 from equations 3, 4

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -21 \end{bmatrix}$$

Step3 : Eliminate x_3 from equation 4

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix}$$

Example: Forward Elimination

Summary of the Forward Elimination :

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ 26 \\ -19 \\ -34 \end{bmatrix} \Rightarrow \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix}$$

Example: Backward Substitution

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix}$$

Solve for x_4, then solve for x_3,... solve for x_1

$$x_4 = \frac{-3}{-3} = 1, \qquad x_3 = \frac{-9+5}{2} = -2$$

$$x_2 = \frac{-6-2(-2)-2(1)}{-4} = 1, \qquad x_1 = \frac{16+2(1)-2(-2)-4(1)}{6} = 3$$

Forward Elimination

To eliminate x_1

$$\left.\begin{aligned} a_{ij} &\leftarrow a_{ij} - \left(\frac{a_{i1}}{a_{11}}\right) a_{1j} \qquad (1 \le j \le n) \\ b_i &\leftarrow b_i - \left(\frac{a_{i1}}{a_{11}}\right) b_1 \end{aligned}\right\} 2 \le i \le n$$

To eliminate x_2

$$\left.\begin{aligned} a_{ij} &\leftarrow a_{ij} - \left(\frac{a_{i2}}{a_{22}}\right) a_{2j} \qquad (2 \le j \le n) \\ b_i &\leftarrow b_i - \left(\frac{a_{i2}}{a_{22}}\right) b_2 \end{aligned}\right\} 3 \le i \le n$$

Forward Elimination

To eliminate x_k

$$\left.\begin{aligned} a_{ij} &\leftarrow a_{ij} - \left(\frac{a_{ik}}{a_{kk}}\right) a_{kj} \quad (k \le j \le n) \\ b_i &\leftarrow b_i - \left(\frac{a_{ik}}{a_{kk}}\right) b_k \end{aligned}\right\} k+1 \le i \le n$$

Continue until x_{n-1} is eliminated.

Backward Substitution

$$x_n = \frac{b_n}{a_{n,n}}$$

$$x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}$$

$$x_{n-2} = \frac{b_{n-2} - a_{n-2,n}x_n - a_{n-2,n-1}x_{n-1}}{a_{n-2,n-2}}$$

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i,j}x_j}{a_{i,i}}$$

Naive Gaussian Elimination

- The method consists of two steps
  - **Forward Elimination**: the system is reduced to upper triangular form. A sequence of elementary operations is used.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \Rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}' & a_{23}' \\ 0 & 0 & a_{33}' \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2' \\ b_3' \end{bmatrix}$$

  - **Backward Substitution**: Solve the system starting from the last variable. Solve for x_n ,x_{n-1},…x_1.

Example 1

Solve using Naive Gaussian Elimination:

Part 1: Forward Elimination ___ Step1: Eliminate x_1 from equations 2, 3

$$x_1 + 2x_2 + 3x_3 = 8 \qquad eq1 \ \ unchanged\ (pivot\ equation)$$

$$2x_1 + 3x_2 + 2x_3 = 10 \qquad eq2 \leftarrow eq2 - \left(\frac{2}{1}\right) eq1$$

$$3x_1 + \ \ x_2 + 2x_3 = 7 \qquad eq3 \leftarrow eq3 - \left(\frac{3}{1}\right) eq1$$

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 8 \\ -\ \ x_2 - 4x_3 &= -6 \\ -5x_2 - 7x_3 &= -17 \end{aligned}$$

Example 1

Part 1: Forward Elimination Step2: Eliminate x_2 from equation 3

$$x_1 + 2x_2 + 3x_3 = 8 \qquad eq1 \ \ unchanged$$

$$-\ x_2 - 4x_3 = -6 \qquad eq2 \ \ unchanged\ (pivot\ equation)$$

$$-5x_2 - 7x_3 = -17 \qquad eq3 \leftarrow eq3 - \left(\frac{-5}{-1}\right) eq2$$

$$\Rightarrow \begin{cases} x_1 + 2x_2 + 3x_3 = 8 \\ -\ x_2 - 4x_3 = -6 \\ 13x_3 = 13 \end{cases}$$

Example 1: Backward Substitution

$$x_3 = \frac{b_3}{a_{3,3}} = \frac{13}{13} = 1$$

$$x_2 = \frac{b_2 - a_{2,3}x_3}{a_{2,2}} = \frac{-6 + 4x_3}{-1} = 2$$

$$x_1 = \frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3}{a_{1,1}} = \frac{8 - 2x_2 - 3x_3}{a_{1,1}} = 1$$

The solution is $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$

Determinant

The elementary operations do not affect the determinant

Example:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 2 \\ 3 & 1 & 2 \end{bmatrix} \xrightarrow{\text{Elementary operations}} A' = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -4 \\ 0 & 0 & 13 \end{bmatrix}$$

$$\det(A) = \det(A') = -13$$

How Many Solutions Does a System of Equations AX=B Have?

Unique	No solution	Infinite
$\det(A) \neq 0$	$\det(A) = 0$	$\det(A) = 0$
reduced matrix has no zero rows	reduced matrix has one or more zero rows	reduced matrix has one or more zero rows
	corresponding B elements $\neq 0$	corresponding B elements $= 0$

Examples

Unique

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} X = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix} X = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

solution :

$$X = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$$

No solution

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} X = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} X = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

No solution

$$0 = -1 \ \textit{impossible!}$$

infinte # of solutions

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} X = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} X = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

Infinite # solutions

$$X = \begin{bmatrix} \alpha \\ 1 - .5\alpha \end{bmatrix}$$

Pseudo-Code: Forward Elimination


```
Do k = 1 to n-1
   Do i = k+1 to n
      factor = a(i,k) / a(k,k)
      Do j = k+1 to n
         a(i,j) = a(i,j) – factor * a(k,j)
      End Do
      b(i) = b(i) – factor * b(k)
   End Do
End Do
```

Pseudo-Code: Back Substitution

$x_n = b_n / a_{n,n}$

Do i = n-1 downto 1

    sum = b_i

    Do j = i+1 to n

        sum = sum – $a_{i,j} * x_j$

    End Do

    x_i = sum / $a_{i,i}$

End Do

Problems with Naive Gaussian Elimination

- The Naive Gaussian Elimination may fail for very simple cases. **(The pivoting element is zero).**

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

- **Very small pivoting element may result in serious computation errors**

$$\begin{bmatrix} 10^{-10} & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

How Do We Know If a Solution is Good or Not

Given AX=B

X is a solution if AX-B=0

Compute the residual vector R= AX-B

Due to rounding error, R may not be zero

$$\text{The solution is acceptable if } \max_{i} |r_i| \leq \varepsilon$$

How Good is the Solution?

$$\begin{bmatrix} 1 & -1 & 2 & 1 \\ 3 & 2 & 1 & 4 \\ 5 & -8 & 6 & 3 \\ 4 & 2 & 5 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \quad \textit{solution} \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -1.8673 \\ -0.3469 \\ 0.3980 \\ 1.7245 \end{bmatrix}$$

$$\text{Residues: } R = \begin{bmatrix} 0.005 \\ 0.002 \\ 0.003 \\ 0.001 \end{bmatrix}$$

Chinese Reminder Theorem

- ***First found in an ancient Chinese puzzle:***
  There are certain things whose number is unknown. Repeatedly divided by 3, the remainder is 2;
  by 5 the remainder is 3; and by 7 the remainder is 2.
  What will be the number?
- ***In modern notation***
  - $x = 2 \pmod 3$
  - $x = 3 \pmod 5$
  - $x = 2 \pmod 7$

Chinese Reminder Theorem

三人同行七十里

Three men walking together for seventy miles,

五树梅花二十一枝

Five plum trees with twenty one branches in flower,

七子团圆正月半

Seven disciples gathering right by the half-moon,

一白零五转回起

One hundred and five and we're back at the start

Chinese Reminder Theorem

What does the poem mean?

$$x = \boxed{a_1 \ (mod\ 3)}$$
$$x = \boxed{a_2 \ (mod\ 5)}$$
$$x = \boxed{a_3 \ (mod\ 7)}$$

三人同行七十里
五树梅花二十一枝
七子团圆正月半
一百零五转回起

$$x = \boxed{70a_1} + \boxed{21a_2} + \boxed{15a_3} \ \boxed{(mod\ 105)}$$

- Why is the solution correct?
  - x = 70 a1 + 21 a2 + 15 a3 (mod 105)
    notice that
    70 = 1 (mod 3) = 0 (mod 5) = 0 (mod 7)
    21 = 0 (mod 3) = 1 (mod 5) = 0 (mod 7)
    15 = 0 (mod 3) = 0 (mod 5) = 1 (mod 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 solution. Such a proof is said to be *constructive*; the advantage of constructive proofs is that they yield a procedure or algorithm for obtaining the desired quantity. We now use the procedure motivated in the proof of Theorem 2.9 to solve the system of linear congruences in one variable of Example 9.

Chinese Reminder Theorem

- For example, consider the problem of finding an integer x such that

$$x \equiv 2 \pmod 3$$
$$x \equiv 3 \pmod 4$$
$$x \equiv 1 \pmod 5$$

- A brute-force approach converts these congruences into sets and writes the elements out to the product of 3×4×5 = 60 (the solutions modulo 60 for each congruence):
  - x ∈ {2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 50, 53, 56, 59, ...}
  - x ∈ {3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, ...}
  - x ∈ {1, 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, ...}
- To find an x that satisfies all three congruences, intersect the three sets to get: x ∈ {11, ...}
- Which can be expressed as

$$x \equiv 11 \pmod{60}$$

Chinese Reminder Theorem

- Another way to find a solution is with basic algebra, modular arithmetic, and stepwise substitution.
- We start by translating these congruences into equations for some t, s, and u:
  - Eq 1: $x = 2 + 3t$
  - Eq 2: $x = 3 + 4s$
  - Eq 3: $x = 1 + 5u$
- Substitute x from equation 1 into congruence 2:
  - $2+3t = 3 \pmod 4) => t = 3 + 4s$
- Substitute t into equation 1: $x = 11+12s$
- Substitute this into congruence 3:
  - $11+12s = 1 \pmod 5 => s = 0 + 5u$
- Finally, $x = 11+12s = 11 + 12(5u) = 11 + 60u$

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

- Algorithms for testing if N is prime: isPrime(N)
  - First try: check if N is divisible by i ϵ [2 .. N-1]?
  - O(N)
- Improved 1: Is N divisible by i ϵ [2 .. sqrt(N)]?
  - O(sqrt(N))
- Improved 2: Is N divisible by i ϵ [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 ϵ primes ≤ sqrt(N)
  - O(π(sqrt(N))) = O(sqrt(N)/log(sqrt(N)))
  - π(M) = num 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 & 1
  - Start from i = 2 until k*i > N
    - If bitset at index i is on, cross all multiple of i (i.e. turn off bit at index i) starting from i*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, <u>2</u>, 3, <u>4</u>, 5, <u>6</u>, 7, <u>8</u>, 9, <u>10</u>, 11, …, 51, <u>52</u>, 53, <u>54</u>, 55, …, 75, <u>76</u>, 77, …
  - 0, 1, 2, <u>3</u>, 4, 5, 6, 7, 8, <u>9</u>, 10, 11, …, <u>51</u>, 52, 53, <u>54</u>, 55, …, <u>75</u>, 76, 77, …
  - 0, 1, 2, 3, 4, <u>5</u>, 6, 7, 8, 9, <u>10</u>, 11, …, 51, 52, 53, 54, <u>55</u>, …, <u>75</u>, 76, 77, …
  - 0, 1, 2, 3, 4, 5, 6, <u>7</u>, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, <u>77</u>, …

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
```

Factorization

- An integer N can be expressed as:
  - N = PF * N‘, where
    PF = a **prime factor**
    N' = another number which is N / PF
- If N' = 1, stop; otherwise, repeat
- N is reduced every time we find a divisor

```
vi primeFactors(ll N) {      // remember: vi is vector<int>, ll is long long
  vi factors;
  ll PF_idx = 0, PF = primes[PF_idx];  // PF = 2, then 3,5,7,... is also ok
  while (N != 1 && (PF * PF <= N)) {  // stop at sqrt(N); N can get smaller
    while (N % PF == 0) { N /= PF; factors.push_back(PF); }    // remove PF
    PF = primes[++PF_idx];                                 // only consider primes!
  }
  if (N != 1) factors.push_back(N);            // special case if N is a prime
  return factors;      // if N does not fit in 32-bit integer and is a prime
}                   // then `factors' will have to be changed to vector<ll>
```

Search

- When a problem is small or (almost) all possibilities have to be tried *complete search* is a candidate approach.
- To determine the feasibility of complete search estimate the number of calculations that have to be made in the worst case.
- *Iterative complete search* uses nested loops to *generate* every possible complete solution and *filter* out the valid ones.
  - Iterating over all permutations using next_permutation
  - Iterating over all subsets using bit set technique
- *Recursive complete search* extends a partial solution with one element until a complete and valid solution is found.
  - This approach is often called *recursive backtracking*.
  - *Pruning* is used to significantly improve the efficiency by removing partial solutions that can not lead to a solution as soon as possible. In the best case only valid solutions are generated.

Summary

- Arithmetic (lab 3.1 and 3.2)
- Solving linear equation systems (lab 3.3 and 3.4)
- Chinese reminder theorem (lab 3.5 and 3.6)
- Prime numbers and factorization (lab 3.7 and 3.8)
- Heuristic Search (exercise 4)