Discrete Mathematics

	topic
week 1	***Sets***
week 2	***Functions***
week 3	***Counting***
week 4	***Relations***
week 5	***Sequences***
week 6	***Modular Arithmetic***
week 7	***Asymptotic Notation***
week 8	***Orders***

Jonathan Barrett
jonathan.barrett@cs.ox.ac.uk
Material by Andrew Ker
University of Oxford
Department of Computer Science

Discrete Mathematics

Jonathan Barrett
jonathan.barrett@cs.ox.ac.uk
Material by Andrew Ker
University of Oxford
Department of Computer Science

Chapter 6: Modular Arithmetic

Definition

Fix a positive integer n. We can define an equivalence relation $\equiv$ on $\mathbb{Z}$ by

$$x \equiv y \pmod{n} \quad \text{if} \quad n \mid (x - y)$$

$x \equiv y \pmod{n}$ ***can be understood as***
"x and y have the same remainder when divided by n."

Definition

Fix a positive integer n. We can define an equivalence relation $\equiv$ on $\mathbb{Z}$ by

$$x \equiv y \pmod{n} \quad \text{if} \quad n \,|\, (x - y)$$

$x \equiv y \pmod{n}$ can be understood as
"x and y have the same remainder when divided by n."

The equivalence classes are of the form

$$[x] = \{\ldots,\ x - 2n,\ x - n,\ x,\ x + n,\ x + 2n,\ \ldots\}$$

So $\mathbb{Z}$ is partitioned into the equivalence classes

$$[0],\ [1],\ [2],\ \ldots,\ [n-1]$$

Addition and Multiplication

Claim If $x_1 \equiv x_2 \pmod{n}$ and $y_1 \equiv y_2 \pmod{n}$ then

- $x_1 + y_1 \equiv x_2 + y_2 \pmod{n}$ and
- $x_1 y_1 \equiv x_2 y_2 \pmod{n}$.

Addition and Multiplication

Claim If $x_1 \equiv x_2 \pmod{n}$ and $y_1 \equiv y_2 \pmod{n}$ then

- $x_1 + y_1 \equiv x_2 + y_2 \pmod{n}$ and
- $x_1 y_1 \equiv x_2 y_2 \pmod{n}$.

It means that these operations on the equivalence classes:

$$[x] + [y] = [x + y] \qquad\qquad [x] \cdot [y] = [x \cdot y]$$

are well defined.

This allows us to compute "keeping only remainders", reducing modulo n at each stage of a computation.

Addition and Multiplication

Claim If $x_1 \equiv x_2 \pmod{n}$ and $y_1 \equiv y_2 \pmod{n}$ then

- $x_1 + y_1 \equiv x_2 + y_2 \pmod{n}$ and
- $x_1 y_1 \equiv x_2 y_2 \pmod{n}$

It means that these operations on the equivalence classes:

$$[x] + [y] = [x + y] \qquad [x] \cdot [y] = [x \cdot y]$$

are well defined.

This allows us to compute "keeping only remainders", reducing modulo n at each stage of a computation.

Also, every integer x has an **additive inverse** (mod n), $(-x)$, with

$$x + (-x) \equiv 0 \pmod{n},$$

so subtraction of equivalence classes is also well defined.

Exponentiation

Claim If $x_1 \equiv x_2 \pmod{n}$ and $y \in \mathbb{N}$ then

- $(x_1)^y \equiv (x_2)^y \pmod{n}$

NB For $x_1 \equiv x_2$ (mod n), we do not in general have $y^{x1} \equiv y^{x2}$ (mod n).

There are ways to manipulate the exponent:

if p and q are different prime numbers, *and x_1, x_2 positive integers,*

$y^{x_1} \equiv y^{x_2} \pmod{p}$ ***as long as*** $x_1 \equiv x_2 \pmod{p-1}$

$y^{x_1} \equiv y^{x_2} \pmod{pq}$ ***as long as*** $x_1 \equiv x_2 \pmod{(p-1)(q-1)}$

etc.

Method of Repeated Squaring

An efficient method of computing $x^y \pmod{n}$ is

$$x^y = \begin{cases} 1 & \text{if } y = 0 \\ x \cdot (x^2)^{\frac{y-1}{2}} & \text{if } y \text{ is odd} \\ (x^2)^{\frac{y}{2}} & \text{if } y > 0 \text{ is even} \end{cases}$$

which is true for all x and non-negative integers y.

Method of Repeated Squaring

An efficient method of computing $x^y \pmod{n}$ is

$$x^y = \begin{cases} 1 & \text{if } y = 0 \\ x \cdot (x^2)^{\frac{y-1}{2}} & \text{if } y \text{ is odd} \\ (x^2)^{\frac{y}{2}} & \text{if } y > 0 \text{ is even} \end{cases}$$

which is true for all x and non-negative integers y.

Computing x^y by multiplying x by itself y times would take $y - 1$ multiplications, but the method of repeated squaring takes only about $\log_2 y$.

NB: exponentiating (mod n) is something of great practical importance in cryptography.

MOD and DIV

Two operations on integers, provided as primitives in many programming languages, are

$$\text{MOD} : \mathbb{N} \times \mathbb{N}_+ \to \mathbb{N} \qquad\qquad \text{DIV} : \mathbb{N} \times \mathbb{N}_+ \to \mathbb{N}$$

$$\begin{aligned} m \text{ DIV } n &= \lfloor m/n \rfloor \\ m \text{ MOD } n &= \text{the remainder when } m \text{ is divided by } n \end{aligned}$$

MOD and DIV

Two operations on integers, provided as primitives in many programming languages, are

$$\text{MOD} : \mathbb{N} \times \mathbb{N}_+ \to \mathbb{N} \qquad\qquad \text{DIV} : \mathbb{N} \times \mathbb{N}_+ \to \mathbb{N}$$

$$\begin{array}{rcl} m \text{ DIV } n & = & \lfloor m/n \rfloor \\ m \text{ MOD } n & = & \text{the remainder when } m \text{ is divided by } n \end{array}$$

We have the equation

$$m = n \cdot (m \text{ DIV } n) + m \text{ MOD } n$$

expressing m as an integer multiple of n plus a nonnegative remainder.

Many languages attempt to extend the domain to all integers. It is not at all clear what the "right" answers are when the modulus is negative and not all languages make the same choice. Beware!

Greatest Common Divisor

The **greatest common divisor** $\gcd(m, n)$ is the largest integer dividing both m and n.

(Equivalently: $g = \gcd(m, n)$ if
1. $g \mid m$
2. $g \mid n$
3. $l \mid m$ and $l \mid n \Rightarrow l \mid g$)

The greatest common divisor of a <u>set</u> of integers is the largest integer which divides them all.

<u>Fact</u>

gcd(m,n) = gcd(m,m-n)

Euclid's Algorithm

A method for computing $\gcd(m, n)$:

```
0. a := m; b := n;

1. q := a DIV b;
   r := a MOD b;

2. If r==0 return(b);
      else a := b;
           b := r;
           goto 1;
```

Euclid's Algorithm

A method for computing $\gcd(m, n)$:

```
0. a := m; b := n;

1. q := a DIV b;
   r := a MOD b;

2. If r==0 return(b);
      else a := b;
           b := r;
           goto 1;
```

On paper we write a sequence of equations: $a = qb + r$ with $0 \leq r < b$ ***until the remainder is zero.***

Example Find $\gcd(1029, 273)$.

Euclid's Algorithm

A method for computing $\gcd(m, n)$:

```
0. a := m; b := n;

1. q := a DIV b;
   r := a MOD b;

2. If r==0 return(b);
      else a := b;
           b := r;
           goto 1;
```

On paper we write a sequence of equations: $a = qb + r$ with $0 \leq r < b$ ***until the remainder is zero.***

Example Find $\gcd(1029, 273)$.

Answer 21

Euclid's Extended Algorithm

A method for computing $\gcd(m, n)$:

```
0. a := m; b := n;    x := 0; y := 1; x′ := 1; y′ := 0;

1. q := a DIV b;          x := x′ - (q * x);  x′ := x;
   r := a MOD b;      y := y′ - (q * y);  y′ := y;

2. If r==0 return(b, x′, y′);
      else a := b;
           b := r;
           goto 1;
```

We find integers x, y such that

$$\gcd(m, n) = mx + ny.$$

Multiplicative Inverses

Fix a modulus n and let x be an integer. A **multiplicative inverse** for x is an integer y satisfying

$$xy \equiv 1 \pmod{n}$$

and we write $y \equiv x^{-1} \pmod{n}$.

Multiplicative Inverses

Fix a modulus n and let x be an integer. A **multiplicative inverse** for x is an integer y satisfying

$$xy \equiv 1 \pmod{n}$$

and we write $y \equiv x^{-1} \pmod{n}$.

Claim If p is prime, every x with $x \not\equiv 0 \pmod{p}$ has a multiplicative inverse (mod p).

Not only does the extended Euclid Algorithm tell us that an inverse exists, it gives an efficient way to find one.

The Pigeonhole Principle

"You cannot put more than n pigeons into n pigeonholes without having at least two pigeons in one hole."

This is a powerful technique for tricky proofs, especially "nonconstructive" proofs.

The Pigeonhole Principle

"You cannot put more than n pigeons into n pigeonholes without having at least two pigeons in one hole."

This is a powerful technique for tricky proofs, especially "nonconstructive" proofs.

Claim In any group of at least 2 people, there must be a pair with the same number of friends within the group.

(Assuming friendship is symmetric!)

The Pigeonhole Principle

"You cannot put more than n pigeons into n pigeonholes without having at least two pigeons in one hole."

This is a powerful technique for tricky proofs, especially "nonconstructive" proofs.

A **lossless compression algorithm** takes arbitrary computer files as inputs and outputs a compressed version, from which the original can be perfectly reconstructed.

Claim If a lossless compression algorithm reduces the size of any files, it must also increase the size of some files.

The Pigeonhole Principle

"You cannot put more than n pigeons into n pigeonholes without having at least two pigeons in one hole."

This is a powerful technique for tricky proofs, especially "nonconstructive" proofs.

A **lossless compression algorithm** takes arbitrary computer files as inputs and outputs a compressed version, from which the original can be perfectly reconstructed.

Claim If a lossless compression algorithm reduces the size of any files, it must also increase the size of some files.

In practice, a useful compression method is one that reduces the size of files we usually want to store on a computer.

Applications

Claim Take any set A of $n+1$ distinct positive integers less than or equal to $2n$, where $n \geq 1$. Then

- at least one element of A divides another, and
- at least two elements of A are coprime.

Applications

Claim Take any set A of $n+1$ distinct positive integers less than or equal to $2n$, where $n \geq 1$. Then

- at least one element of A divides another, and
- at least two elements of A are coprime.

"You cannot put more than n pigeons into $2n$ different pigeonholes without having at least two pigeons in consecutive holes."

Applications

Claim Take any set A of $n+1$ distinct positive integers less than or equal to $2n$, where $n \geq 1$. Then

- at least one element of A divides another, and
- at least two elements of A are coprime.

Claim For any prime p there is a solution (in x and y) of $x^2 + y^2 \equiv -1 \pmod{p}$.

Diversion: Modular Square Roots of -1

Claim If p is prime, then $x^2 \equiv -1 \pmod{p}$ has a solution if and only if $p = 2$ or $p \equiv 1 \pmod{4}$.

Discrete Mathematics

Jonathan Barrett
jonathan.barrett@cs.ox.ac.uk
Material by Andrew Ker
University of Oxford
Department of Computer Science

End of Chapter 6