Fibonacci Problem:

A farmer raises rabbits. Each rabbit gives birth when 2 months old and then each month there after.

Question:

How many rabbits will the farmer have after n *months?*

Try it on small numbers.

F_n ... the number of rabbits after n month.

Each rabbit alive the n-1th month remains nth month.

Each rabbit alive the $n-2$th month adds one more rabbit to nth month.

$F_1 = 1$

$F_2 = 1$

$F_{n+1} = F_n + F_{n-1}$ if $n \geq 2$

This formula is an example of a **recurrence formula** (as opposite to an **explicit formula**).

Recurrence formulas are often easy to obtain (and easy to code).

Often we extend the sequence by starting $F_0 = 0$

$F_1 = 0$

$F_2 = 1$

$F_{n+1} = F_n + F_{n-1}$ if $n \geq 1$

Sequence of Fibonacci numbers

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, \ldots$

Fibonacci numbers occur often in analysis of algorithms.

For that reason, Fibonacci numbers have been studied in detail.

Some basic identities:

$$F_0 + F_1 + F_2 + \cdots + F_n = F_{n+2} - 1$$

$$F_1 + F_3 + F_5 + \cdots + F_{2n-1} = F_{2n}$$

$$F_0 - F_1 + F_2 - F_3 + \cdots - F_{2n-1} + F_{2n} = F_{2n-1} - 1$$

$$F_{n-1}F_{n+1} - F_n^2 = (-1)^n$$

A Formula for the Fibonacci Numbers

Theorem 10 *The Fibonacci numbers are given by the formula*

$$F_n = \frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n)$$

$$F_n \approx \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n$$

Proof is technical and thus omitted.

Recurrence Relations

Some problems are easily solved by recursive algorithms or definitions (e.g. Fibonacci sequence).

Recursive Algorithm: an algorithm which finds solution of a larger problem in terms of solutions of the same problem of smaller size.

Typical example:

Compute the nth power of an integer. We can use the facts that:

$x^0 = 1$ and
$x^n = x \cdot x^{n-1}$ if $n > 0$

To find the efficiency of a recursive algorithms, we need to solve a related recurrence relation expressing the number of operation needed by the algorithm:

---

integer power(x, n)

{ if($n == 0$) return 1;

else return $x * power(x, n - 1)$;

}

Let p_n be the number of multiplications needed:

$p_0 = 0$;

$p_n = p_{n-1} + 1$

A sorting method: To sort n elements we can:

Sort the first half of elements;

Sort the second half of elements;

Merge the two partial solutions together.

Let s_n be the number data comparisons needed:

$$s_n = 2s_{n/2} + n$$

$$s_1 = 0$$

(Assuming that the merge of two sorted segments of size $n/2$ can be done by comparing n data elements).

Problem to consider: given a **recurrence formula**, find an **explicit formula** for it.

We divide the formulas into two groups:

Linear Recurrence formula: the value of x_n is defined using element(s) preceding x_n by a **constant** number of positions in the sequence. $x_n = ax_{n-5} + c$

Divide and Conquer Recurrence formula: the value of x_n is defined using an element(s) preceding x_n by a **fraction** of n in the sequence. $x_n = ax_{n/3} + c$

Solving Linear Recurrence formulas:

we have a **linear** recurrence formula

$x_n = bx_{n-i} + c$ recurrence relation

$x_0 = d_0,\ x_1 = d_1, \ldots,\ x_{i-1} = d_{i-1}$ initial conditions

Find an explicit formula for x_n

The number of initial conditions depends on the recurrence relation.

Initial conditions are needed to get a unique solution.

Iterative solution (Back-substitution Method)

Use the recurrence relation repeatedly to obtain the pattern. Then substitute there the initial conditions.

$$x_n = 2x_{n-1} + 3, \quad x_1 = 1$$

$$\begin{aligned}
x_n &= 2x_{n-1} + 3 \\
&= 2(2x_{n-2} + 3) + 3 = 2^2 x_{n-2} + 2 \cdot 3 + 3 \\
&= 2^2(2x_{n-3} + 3) + 2 \cdot 3 + 3 = 2^3 x_{n-3} + 2^2 \cdot 32 \cdot 3 + 3 \\
&= \ldots \\
&= 2^k x_{n-k} + 2^{k-1} \cdot 3 + 2^{k-2} \cdot 3 + \cdots 2 \cdot 3 + 3 \\
&= 2^{n-1} x_1 + 2^{n-2} \cdot 3 + 2^{k-2} \cdot 3 + \cdots 2 \cdot 3 + 3 \\
&= 2^{n-1} + 3(2^{n-2} + 2^{k-2} + \cdots 2 + 1) \\
&= 2^{n-1} + 3(2^{n-1} - 1) \\
&= 2^{n+1} - 3
\end{aligned}$$

In general, we should expect that:

if the recurrence formula is of shape

$$x_n = ax_{n-k} + b, \qquad x_1 = 1$$

then x_n is proportional $a^{n/k}$

(it means that $x_n \approx ca^{n/k}$ for some constant c).

If the recurrence formula is of shape

$$x_n = x_{n-1} + n + b, \qquad x_1 = 1$$

then x_n is proportional to n^2

Divide and Conquer Recurrences

Different algorithm for the power of an integer:

---

integer power(x, n)

if ($n == 0$) return 1;

$y = power(x, n/2)$;

if ($even(n)$) return $y * y$;

else return $y * y * x$;

Let q_n be the number of multiplications needed:

$q_n = q_{n/2} + 1$ if n is even
$q_n = q_{n/2} + 2$; if n is odd
$q_0 = 0$

A recurrence relation of type

$$x_n = ax_{n/b} + c$$
$$x_0 = d$$

is a **divide and conquer recurrence relation**

This type of relation is obtained when analyzing **divide and conquer algorithms**.

The idea of solving a problem by dividing it into several subproblems of a fractional size often gives very efficient algorithms.

To calculate the value of q_n, we have to do calculations separately for the two cases of parity:

Case 1: $n, n/2, n/4, \ldots$ are all even, (except the last)

Case 2: $n, n/2, n/4, \ldots$ are all odd.

Use the iteration method:

Case 1:

$$q_n = q_{n/2} + 1 = q_{n/2^2} + 2 = q_{n/2^3} + 3 = \ldots = q_{n/2^k} + k$$

when $n = 2^i$ which means $i = log_2\, n$ we take $k = i$ to get to q_1 and we have

$$q_n = q_1 + log_2\, n = log_2\, n + 2$$

Case 2:

$$q_n = q_{n/2} + 2 = q_{n/2^2} + 2 \cdot 2 = q_{n/2^3} + 3 \cdot 2 = \ldots =$$
$$q_{n/2^k} + k \cdot 2$$

when $2^i - 1 = n$ which means $i = log_2(n+1)$

we take $k = i - 1$ times to get to q_1.

$$q_n = q_1 + 2(\cdot log_2(n+1) - 1) = 2 \cdot log_2(n+1)$$

We can conclude:

$$log_2\, n + 2 \leq q_n \leq 2 \cdot log_2(n+1)$$

This algorithm is much better than the first one!

Consider the recurrence formula of the sorting algorithm for the number of data comparisons needed:

$$s_n = 2s_{n/2} + n$$

$$s_1 = 0$$

$$\begin{aligned} s_n &= 2s_{n/2} + n \\ &= 2(2s_{n/2^2} + n/2) + n = 2^2 s_{n/2^2} + 2n \\ &= 2^2(2s_{n/2^3} + n/2^2) + 2n = 2^3 s_{n/2^3} + 3n \\ &= \ldots \\ &= 2^k s_{n/2^k} + kn \end{aligned}$$

If $2^k = n$ which means that $k \leq log_2\, n$ we have

$$s_n = ns_1 + n\log_2\ n = n\log_2\ n$$

Recurrence formulas occur in quite many situations.

We have seen them in the analysis of algorithms.

Some nontrivial counting problems can be solved easily by recurrence formulas. (Fibonacci sequence)

We should understand, given a recurrence formula for a sequence of numbers, what are the implications of the different parameters of the formula on the size of the elements of the sequence.

Recurrence formulas are also used to solve some counting problems:

Problem: How many binary strings of length 7 contain two consecutive 0? 1101001, 1110001

It is not obvious how to apply a known counting formula in this case.

We can instead establish a recurrence formula for it:

$$x_n = x_{n-1} + x_{n-2} + 2^{n-2}$$
$$x_0 = x_1 = 0$$

We get $x_2 = 1$, $x_3 = 3$, $x_4 = 8$, $x_5 = 19$, $x_6 = 43$, $x_7 = 94, \ldots$

Integers, Divisors, Primes (some of it a review)

Let a and b be two integers.

We say that a **divides** b , or

a is a **divisor** of b,

or b is a **multiple** of a

if there exists an integer m such that $b = am$

These phrases above mean the same thing.

Notation: $a|b$.

If a is not a divisor of b, then we write $a \nmid b$.

If $a \neq 0$, then this means that the ratio b/a is an integer.

If $a \nmid b$ and $a > 0$, then we can still **divide** b **by** a **with remainder**.

The remainder r of the division $b \div a$ is an integer that satisfies $0 \leq r < a$.

If the quotient of the division with remainder is q, then we have

$$b = aq + r.$$

Primes

An integer $p > 1$ is called a **prime** if it is not divisible by any integer other than $1, -1, p$ and $-p$.

Thus $p > 1$ is a prime if it cannot be written as the product of two smaller positive integers.

An integer $n > 1$ that is not a prime is called **composite** (the number 1 is considered neither prime, nor composite).

$2, 3, 5, 7, 11$ are examples of primes

$4 = 2 \cdot 2,\ 8 = 2 \cdot 4,\ 18 = 3 \cdot 6$ are not primes.

Table below contains the primes smaller than 1000.

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103,
107, 109, 113, 127, 131, 137, 139, 149, 151, 157,
163, 167, 173, 179, 181, 191, 193, 197, 199, 211,
223, 227, 229, 233, 239, 241, 251, 257, 263, 269,
271, 277, 281, 283, 293, 307, 311, 313, 317, 331,
337, 347, 349, 353, 359, 367, 373, 379, 383, 389,
397, 401, 409, 419, 421, 431, 433, 439, 443, 449,
457, 461, 463, 467, 479, 487, 491, 499, 503, 509,
521, 523, 541, 547, 557, 563, 569, 571, 577, 587,
593, 599, 601, 607, 613, 617, 619, 631, 641, 643,
647, 653, 659, 661, 673, 677, 683, 691, 701, 709,
719, 727, 733, 739, 743, 751, 757, 761, 769, 773,
787, 797, 809, 811, 821, 823, 827, 829, 839, 853,
857, 859, 863, 877, 881, 883, 887, 907, 911, 919,
929, 937, 941, 947, 953, 967, 971, 977, 983, 991,
997

Primes have fascinated people ever since ancient times.

Their sequence seems very irregular,

yet it seems to carry a lot of hidden structure.

The ancient Greeks already knew that there are **infinitely many primes**. (They proved it!)

It was not easy to prove many further facts about primes.

Their sequence has holes and dense spots.

Theorem 11 *Every positive integer can be written as a products of primes and the factorization is unique (up to the order of primes).*

Theorem 12 *Let* $\pi(n)$ *denotes the number of primes among* $1, 2, 3, \ldots, n$.

$$\pi(n) \approx \frac{n}{\ln n}$$

This can be used to find how many primes are there having say certain number of digits

Theorem 13 Fermat's little theorem

If p is a prime and a is an integer, then $p \mid a^p - a$

It can be stated in several equivalent ways, an alternative:

If p is a prime and a is an integer, then $a^p \equiv a (\mathrm{mod}\ p)$

Euclidean Algorithm

For two positive integers a and b we define $gcd(a, b)$ to be **largest** positive integer **dividing both** a and b.

Euclidean Algorithm is an efficient way to find $gcd(a, b)$.

(Factorizations of a and b would give the answer, but it would be very inefficient.)

Euclidean Algorithm:

Input: Positive integers a, b

Output: $gcd(a, b)$

While $(a > 0)$ do
  {
    $r = b \bmod a$;
    $b = a$;
    $a = r$;
  }
print b; // the GCD

For an algorithm, we must establish:

Termination (done in class)

Correctness (done in class)

Efficiency:

Lemma 1 *During the execution of the Euclidean Algorithm, if* $a < b$ *then the value of the product of* ab *drops by a factor of at least* 2 *in each iteration.*

Theorem 14 *The number of steps of the Euclidean algorithm applied to two positive integers* a *and* b *is at most* $\log_2 a + \log_2 b$.

Thus this algorithm is very efficient.

Theorem 15 $gcd(a, b) = am + bn$ *for some integers* m, n.

(Obviously, one of m, n is negative.)

To find $gcd(a, b)$ as a linear combination of a, b we can use the intermediate steps of the Euclidean algorithm:

$$gcd(300, 18) = gcd(18, 12) = gcd(12, 6) = 6$$

$$300 = 16 \cdot 18 + 12$$

$$18 = 1 \cdot 12 + 6$$

$$12 = 2 \cdot 6$$

So $6 = 18 - 1 \cdot 12 = 18 - 1 \cdot (300 - 16 \cdot 18) = 17 \cdot 18 - 1 \cdot 300$

The Eucledian algorithm can be used to solve some linear equations when the required solutions must be integers.

Example: Find integer values x and y such that

$$7x + 13y = 1$$

Since $GCD(7, 13) = 1$ the Euclidean algorithm can be used to find x and y such that $1 = GCD(13, 7) = 7x + 13y$

Some other equations can be solved in this manner.

Congruences

How to run several application, say $A_1, A_2, \ldots, A_k$ on a single computer:

We select a small time unit, say d milliseconds and the time is divided into infinite sequence of time slots $t_1, t_2, \ldots, t_j, \ldots$ of length d.

During time slot t_j the computer runs application A_i if $j \bmod k = i + 1$.

Since human actions and perception is slow, we have impression that all applications are running at the same time.

We could say that time j_1, j_2 are assigned to the same application when $j_1 \bmod k = j_2 \bmod k$.

Since this case occurs often, we have a special notation for it, called **congruence**.

$$a \equiv b \,(\mathrm{mod} m)$$

if $a \bmod m = b \bmod m$. We call m the modulus of the congruence relation.

Congruence $\bmod m$ is an equivalence relation on N.

$a \equiv a \pmod{m}$

$a \equiv b \pmod{m} \Longrightarrow b \equiv a \pmod{m}$

$a \equiv b \pmod{m}$ and $b \equiv c \pmod{m} \Longrightarrow$

$a \equiv c \pmod{m}$

$\equiv \pmod{m}$ is an equivalence relation.

If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$ then:

$a + c \equiv b + d \pmod{m}$
$a - c \equiv b - d \pmod{m}$
$ac \equiv bd \pmod{m}$
$ak \equiv bk \pmod{m}$ for every integer k.

In many computer applications we need to use **random numbers**, something which can be seen as a toss of a coin or a throw of a die.

Pseudorandom Numbers: numbers that look random.

A linear congruential method for generating pseudorandom numbers :

$$x_{i+1} = (ax_i + c) \bmod m$$

If a, c, m are well-chosen, this generates m distinct integers in a row in no apparent order.

A possible choice:

$m = 2^i, gcd(c, m) = 1, a - 1$ is a multiple of 4.,

$x_{i+1} = (9x_i + 307) \bmod 1024$

The value of x_0 is called the **seed**.

See below an example for seed 0

0, 307, 1022, 289, 860, 879, 26, 541, 56, 811, 438, 153, 660, 103, 210, 149, 624, 803, 366, 529, 972, 906, 269, 680, 283, 806, 393, 772, 87, 66, 901, 224, 275, 734, 769, 60, 847, 762, 1021, 280

If we need random numbers between 1 and k inclusive, where $k < m$, we take $(x_i \bmod k) + 1$.

Example: $k = 36$:

$1, 11, 23, 30, 9, 28, 26, 23, 19, 34, 31, 5, \ldots$

If we need random numbers to be real numbers between 0 and 1 we take x_i/m.

Often it is convenient to work with arithmetic operation in mod manner and use $=$ instead of $\equiv$.

If we work in (mod11) then

$3+4 \equiv 7 (\text{mod } 11),\ 3+8 \equiv 0 (\text{mod } 11)$

When this is done very often we can use a bar to denote that we work in mod manner to simplify the notation. We must clearly identify the modulus or understand which one is used.

$\overline{3}+\overline{4}=\overline{7},\ \overline{3}+\overline{8}=\overline{0},\ \overline{3}*\overline{5}=\overline{4}$

How do we define a division by a in (mod11)?

We do it by defining an **inverse** $\overline{\frac{1}{a}}$ of a in mod 11 arithmetics.

Then $\overline{c}/\overline{a} = \overline{c} \cdot \overline{(\frac{1}{a})}$.

This can be done when we work mod p where p is a prime number.

Division in modular arithmetic

We define that b is a **reciprocal** or **inverse** of $a \pmod{p}$ if

$$ab \equiv 1 \pmod{p}$$

Then we define $\overline{c}/\overline{a} \pmod{p} = \overline{c}\overline{b} \pmod{p}$

Important p must be a prime number.

We can use the Euclidean algorithm to find $\overline{1}/\overline{a}$.

If p is a prime then $gcd(a, p) = 1$.

Euclidean algorithm can be used to find $gcd(a, p)$ in form $ua + vp$ where $u.\ v$ are integers.

We get $ua + vp = 1$ which implies that

$ua \equiv 1 (\text{mod } p)$

Since u could be greater than p,

(and we want the inverse to be between 0 and $p - 1$)

take $x = u \bmod p$.

Then $ax \equiv au \equiv 1 (\text{mod } p)$.

Thus, x is the inverse of a.

Important: If p is not a prime then there are integers a such that for every integer b

$$ab \not\equiv 1 (\text{mod } p)$$

Take $p = 6$ and $a = 2$

$2 \cdot 0 \equiv 0 \ (\text{mod } 6)$, $\quad 2 \cdot 1 \equiv 2 \ (\text{mod } 6)$

$2 \cdot 2 \equiv 4 \ (\text{mod } 6)$ $\quad 2 \cdot 3 \equiv 0 \ (\text{mod } 6)$

$2 \cdot 4 \equiv 2 \ (\text{mod } 6)$ $\quad 2 \cdot 5 \equiv 4 \ (\text{mod } 6)$

$2 \cdot 6 \equiv 0 \ (\text{mod } 6)$, and so on.

Thus, there is no reciprocal to 2 in (mod 6)

Once we have all arithmetic operations mod p defined, we can solve **equations involving congruences** as easily as ordinary equations.

Consider equation $7x + 3 \equiv 0 \pmod{47}$

$7x \equiv -3 \pmod{47}$

Find reciprocal of 7 (mod 47) which is 27.

So we get $27 \cdot 7x \equiv 27 \cdot -3 \pmod{47}$

$x \equiv -81 \pmod{47}$

Solution: $x = 13$.

Similarly we can solve a system of two equations if both use the **same modulus**.

$$12x + 31y \equiv 2 \pmod{127}$$
$$2x + 89y \equiv 23 \pmod{127}$$

Eliminate one of the variables:

$$12x + 31y \equiv 2 \pmod{127}$$
$$12x + 6 \cdot 89y \equiv 6 \cdot 23 \pmod{127}$$

Subtract the equations:

$-503y \equiv -136 \pmod{127}$ which is the same as
$5y \equiv 118 \pmod{127}$

Find the reciprocal of $5 \pmod{127}$ which is 51.

So $y \equiv 51 \cdot 118 \pmod{127}$ or $y \equiv 49 \pmod{127}$

Substitute for y above,

$2x + 89 \cdot 49 \equiv 23 \pmod{127}$

$2x \equiv 107 \pmod{127}$

Find the reciprocal of $2 \pmod{127}$ which is 64.

$x \equiv 64 \cdot 107 \pmod{127}$ or $x \equiv 117 \pmod{127}$

Notice:

The above allow us to solve equations involving the **same prime modulus**.

One can also solve equations in which **different prime moduli** are used, as

$$7x + 3y \equiv 1 \pmod{13}$$

$$2x + 5y \equiv 2 \pmod{17}$$

but it needs a different technique. One cannot solve the above equations using the reciprocal alone.

It needs the so called **Chinese remainder theorem**, which is not covered in this course.