Algorithms for Numbers and Public-Key Cryptography

Jean-Sébastien Coron

Université du Luxembourg

March 11, 2015

Course Summary

- Algorithms for numbers
  - Describe basic algorithms for dealing with numbers
  - Implement them on a computer
- Public-key cryptography
  - Describe the basic public-key algorithms
  - Implement them on a computer

Course 1

- Basics of C programming
  - This is to ensure that everybody has the same minimal background in programming.
  - However you can choose any other language.
    - Python, with the Sage Library.
- Number theory.
  - GCD
  - Euclid's algorithm
  - Euclid's extended algorithm
  - Modular arithmetic.

C programming

- Bases of C programming.
  - Structure of a C program.
  - Variables and types.
  - Printing.
  - Control structures: if and while.
  - For loop.
  - Arrays
  - `argc` and `argv`

Structure of a C program

```
#include <stdio.h>
#define A 10
int main()
{
  printf("Hello world \n");
  printf("A=%d\n",A);
}
```

- #include: include libraries
- #define: definition of constants.
- int main(): definition of main function.

Variables

- A program can store variables in memory.
- One must declare a variable before using it.
  - `int a;` declaration of variable `a` as integer.
- Integer variables
  - `short`: 16 bits ± 32767.
  - `int`: 16 or 32 bits ± 32767 or $\pm 2.10^9$.
  - `long`: 32 bits $\pm 2.10^9$.
- `unsigned short`, `unsigned int`, `unsigned long` $\rightarrow$ non-negative integers.

Floats

- Encoding
  - Mantissa: m.
  - Exponent: e.
  - $m * 2^e$.
- `float`: 24+8 bits.
  - $< 10^{38}$.
- `double`: 53+11 bits.
  - $< 10^{308}$.
- `long double`: 64+16 bits.
  - $< 10^{4932}$.

Operations on variables

- Assignation:
  - `a=b;`
  - the content of variable `b` is copied in variable `a`.
- Arithmetic operations:
  - $a + b$: addition.
  - $a - b$: subtraction.
  - $a * b$: multiplication.
  - a/b: division.
    - Euclidean division for integers :
  - $a\%b$: remainder.

Example

- Incrementation of a variable :
  - `i=i+1;`
- Circumference of a circle with radius in variable `float r` :
  - ```
    float c;
    c=2*3.14*r;
    ```
- Average of variables `x` and `y`:
  - ```
    float x,y,m;
    m=(x+y)/2;
    ```

Initialization of variables

- When a variable has been declared, its value is arbitrary.
- One can initialize it simultaneously :

```
#include <stdio.h>
int u=3;
int main()
{
  int a=2;
  printf("a=%d,u=%d\n",a,u);
}
```

Printing variables

- `printf` can print text and variable value on the standard output.
  - %d for an `int` or `long`.
  - %f for an `float` or `double`.

```
float a=2.3;
int b=4;
printf("a=%f,b=%d\n",a,b);
```

Input

- `scanf` enables to read a variable value from keyboard.

```
float a;
int b;
printf("Give a float:");
scanf("%f",&a);
printf("Give an integer:");
scanf("%d",&b);
```

Example

- Computing the circumference and area of a disk :

```
#include <stdio.h>
int main()
{
  float x;
  scanf("%f",&x);
  float pi=3.1415926;
  float c=2*pi*x;
  float a=pi*x*x;
  printf("circonference=%f\n",c);
  printf("aire=%f\n",a);
}
```

Conditions

- `if then else`

```
if (test)
{
    instructions if true
}
else
{
    instructions if false
}
```

- `else {...}` is optional.

Tests

- Possibles tests :
  - Equality: `a==b`
  - Non-equality: `a!=b`
  - Comparison: `a<b`
  - Comparison: `a<=b`
- Operations on tests :
  - Negation: `!(test)`.
  - And: `((test1) && (test2))`
  - Or: `((test1) || (test2))`

Example

- Ask for two integers and print them in increasing order :

```
#include <stdio.h>
int main()
{
  int a,b;
  printf("entrez deux entiers:\n");
  scanf("%d%d",&a,&b);
  if(a<b)
  {
    printf("%d %d\n",a,b);
  }
  else
  {
    printf("%d %d\n",b,a);
  }
}
```

While

- Repeat `instruction` while `test` is true.

  `while` (*test*)
  `{`
  instruction
  `}`

- Example: determine the bit-size of a :

```
unsigned int a; int t=0;
while(a>0)
{
  a=a/2;
  t=t+1;
}
```

For loop

- Repeat the same instruction many times with a counter.
- Syntax: `for(< init >;< test >;< counter >)`
- Example: print integers from 1 to 10.
  - `for (i=1;i<=10;i++) printf("%d\n",i);`
- `< init >`: initialize counter.
- `< test >`: test counter.
- `< counter >`: increment counter.

Example

- Compute 2^n given n :

```
int c=1;
int i;
for(i=0;i<n;i++)
{
  c=c*2;
}
// c contains 2^n.
```

Arrays in C

- Arrays can store a group of variables of the same type.
  - For example:

```
int notes[5]; // array of 5 integers
notes[0]=15; // first entry
notes[1]=8;
notes[2]=16;
notes[3]=17;
notes[4]=9;// 5th entry
```

Arrays type

- Arrays type:
  - `float tabf[5]`: array of 5 `float`.
  - `double tabd[10]`: array of 10 `double`.
  - `int tabi[7]`: array of 7 `int`.
- Index:
  - An array of n elements is indexed from 0 to $n-1$:
  - `int tabi[7]`.
    - From `tab[0]` to `tab[6]`.

Constant size

- An array must be of constant size.
  - This size must be written in the program, for example `int tab[10]`
  - `#define`:

```
#include <stdio.h>
#define N 10    // one defines N=10
int main()
{
  int tab[N];
  int autretab[5];
}
```

Characters

- Stored in a byte (8 bits).
  - ASCII encoding:
  - `'A'` $\rightarrow$ 65, `'B'` $\rightarrow$ 66,...
  - `'0'` $\rightarrow$ 48,...
- Printing a character:

```
char x;
x='A';
printf("%c",x);
```

Strings

- A string is an array of characters.
  - `char ch[10]="hello";` creates an array of characters such that :
  - `ch[0]='h'`, `ch[1]='e'`, `ch[2]='l'`, `ch[3]='l'`, `ch[4]='o'`
  - `ch[5]='\0'` is the last character.
  - The others elements are not initialized.
- Printing a string :
  - `printf("%s",ch);`

Initialization of an array

- Using `for`:

```
#define N 10
int main()
{
  int tab[N];
  int i;
  for(i=0;i<N;i++)
  {
    tab[i]=0;
  }
}
```

Example

- Factorial using array :
  - $n! = n \cdot (n-1) \cdots 2 \cdot 1$

```
#define N 10
int main()
{
  int fac[N];
  int i;
  fac[0]=1;
  for(i=1;i<N;i++)
  {
    fac[i]=fac[i-1]*i;
  }
}
```

2-dimensional arrays

- One can declare arrays with two dimensions or more :
  - `int tab[4][3];` declares an array of size 4*3.
- Initialization :

```
#define M 10
#define N 5
int main()
{
  int tab[M][N];
  int i,j;
  for(i=0;i<M;i++)
    for(j=0;j<N;j++)
      tab[i][j]=0;
}
```

Command-line arguments

- Obtaining command-line arguments :
  - One would like to be able to write :

    ```
    $ fact 5
    120
    ```
- Advantage :
  - No need to write `int n=5` in the code (then code needs to be recompiled each time `n` is changed).
  - Avoid a `scanf`.

argc and argv

- Command-line arguments are stored in array `argv`.
- `argc` contains the number of arguments (size of `argv`).

```
#include <stdio.h>
int main(int argc,char *argv[])
{
  int i;
  for(i=0;i<argc;i++)
  {
    printf("%s\n",argv[i]);
       // print each argv[i] word
  }
}
```

Using `argc` and `argv`

- If the previous program is named `affiche`, then :
  - ```
    $ affiche hello world 2
    affiche
    hello
    world
    2
    ```
- Here `argc=4`.
    ```

Converting a string to an integer

- `int atoi()` enables to convert a string to an integer.
  - Example : print the square of an integer.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc,char *argv[])
{
  int a=atoi(argv[1]); // conversion
  printf("%d\n",a*a);
}
```

- `$ carre 3`
  `9`

Pointers

- A pointer is a memory address.
  - When a variable is declared, some memory is allocated to it.
  - The address is obtained using `&`

```
// allocated memory for a
int a;

// prints the address of a
// (for ex: 2678673).
printf("%d\n",&a);
```

Pointers

- Pointer declaration:
  - Integer pointer: `int *p;`
  - Char pointer: `char *pc;`
  - Float pointer: `float *pf;`
- Access to content:
  - `*p` is the value at address p.

Example

```
int a; // allocate memory for a
a=2;

int *p; //
p=&a; // p is now a pointer to a

printf("%d\n",*p);
// prints the content at address p
// 2.

*p=3; // now a=3
```

Memory allocation

- Pointer declaration:
  - `int *p;`
  - Does not allocate memory at address `p`.
  - `*p=2;` can give an error.
- `p` can become a pointer to an existing variable:
  - `int a; int *p; p=&a;`
- Or one can allocate memory for `p`.
  - Using `malloc`.
  - `int *p;`
    `p=malloc(sizeof(int));`

Dynamic arrays

- Dynamic array of size `n`:
  - ```
    int *t;
    t=malloc(n*sizeof(int));
    ```
  - `t[0]` to `t[n-1]`
- Dynamic size.
  - Not necessarily known at compilation time.
  - Known at execution time.
  - As opposed to
    ```
    int t[10];
    ```
    ```

Example

```
#include <stdio.h>
int main()
{
  int n;
  n=2*10;
  // n is known only at execution time

  int *p;
  p=malloc(n*sizeof(int));

  int i;
  for(i=0;i<n;i++) p[i]=0;
}
```

Freeing memory

- Function `free`.
  - `int *t=malloc(n*sizeof(int)); free(t);`

Functions

- Syntax :
  - ```
    rtype fname(para1,para2,...)
    {
     localvariables
     functioncode
    }
    ```
- Example :
  - ```
    double max(double a,double b)
    {
      double m;
      if(a>b) m=a; else m=b;
      return m;
    }
    ```

Using the function

- ```
  #include <stdio.h>
  double max(double a,double b)
  {
    double m;
    if(a>b) m=a; else m=b;
    return m;
  }
  int main()
  {
    double x=3.5;
    double y=3.2;
    double z=max(x,y);
  }
  ```
  ```

`void` function

- A `void` function is a function that returns nothing.

```
#include <stdio.h>
void affiche(int a)
{
  printf("La valeur est:%d\n",a);
}

int main()
{
  int u=3;
  affiche(u);
}
```

Printing an array

```
#include <stdio.h>
void affiche(int tab[],int n)
{
  int i;
  for(i=0;i<n;i++) printf("%d ",tab[i]);
  printf("\n");
}

int main()
{
  int t[5]={1,3,6,5,1}
  affiche(t,5);
}
```

GCD

- Common divisor :
  - Let a, b be two integers. A common divisor of a and b is an integer m such that $m|a$ and $m|b$.
- GCD.
  - GCD of two integers a and b is the greatest common divisor of a and b.
  - If $d = \mathrm{GCD}(a, b)$, then for all m such that $m|a$ and $m|b$, we have $m|d$.
- Example
  - $\mathrm{GCD}(9, 6) = 3$
  - $\mathrm{GCD}(7, 5) = 1$.

Euclid's algorithm

- Euclid's algorithm :
  - Input: a, b.
  - Let $r_0 = a$ and $r_1 = b$.
  - For $i \geq 0$, one defines the sequence (r_i) and (q_i) such that :

    $$r_i = q_i \cdot r_{i+1} + r_{i+2}$$

    where q_i and r_{i+2} are the quotient and remainder of the division of r_i by r_{i+1}
  - The exists $k > 0$ such that $r_k = 0$.
  - Then $\text{GCD}(a, b) = r_{k-1}$.

Proof

- Let $a > 0$ and $b \geq 0$.
  - If $b = 0$, then $\text{GCD}(a, b) = \text{GCD}(a, 0) = a$
  - Otherwise, let $a = b \cdot q + r$ with $0 \leq r < b$.
  - Then $\text{GCD}(a, b) = \text{GCD}(b, r)$.
  - (b, r) is less than (a, b).
- $\text{GCD}(a, b) = \text{GCD}(b, r)$
  - If $d|a$ and $d|b$, then $d|r$, and then $d|\text{GCD}(b, r)$. Then $\text{GCD}(a, b)|\text{GCD}(b, r)$.
  - If $d'|b$ and $d'|r$, then $d'|a$, and then $d'|\text{GCD}(a, b)$. Then $\text{GCD}(b, r)|\text{GCD}(a, b)$.
  - Then $\text{GCD}(a, b) = \text{GCD}(b, r)$.

Basic Properties of Integers

Theorem (Fundamental theorem of arithmetic)

Every non-zero integer n can be expressed as

$$n = \pm p_i^{e_1} \cdots p_r^{e_r}$$

where the p_i's are distinct primes and the e_i are positive integers. Moreover the decomposition is unique, up to reordering of the primes.

- Proof: existence is easy by recursion; unicity: see any standard textbook.

Theorem (Division with remainder property)

For $a, b \in \mathbb{Z}$ with $b > 0$, there exist unique $q, r \in \mathbb{Z}$ such that $a = bq + r$ and $0 \leq r < b$.

Congruence

- Definition
  - Let $n > 0$, and $a, b \in \mathbb{Z}$.
  - a is *congruent* to b if $n|(a - b)$.
  - $a \equiv b \pmod{n}$.
  - n is called the *modulus*.
  - Should not be confused with the *mod* of Euclidean division.

Theorem

Let $n > 0$. For any integer a, there exists a unique integer b such that $a \equiv b \pmod{n}$ and $0 \leq b < n$, namely $b := a \bmod n$.

Examples and properties

- Examples :
  - $2 \equiv 8 \mod 3$ since $3|(8-2)$.
  - $12 \equiv 2 \mod 5$ since $5|(12-2)$.
- Properties :
  - $a \equiv b \mod n \Leftrightarrow \exists k \in \mathbb{Z}, a = b + k \cdot n$.
  - $a \equiv a \mod n$
  - $a \equiv b \mod n \Rightarrow b \equiv a \mod n$
  - $a \equiv b \mod n$ and $b \equiv c \mod n$ implies $a \equiv c \mod n$

Properties

- Addition and multiplication
  - If $a \equiv a' \mod n$ and $b \equiv b' \mod n$, then
  - $a + b \equiv a' + b' \mod n$ and $a \cdot b \equiv a' \cdot b' \mod n$.
- When computing modulo n, one can substitute to x a value x' congruent to x modulo n.
  - Computing a with $0 \leq a < 8$ such that $a \equiv 83 \cdot 72 \mod 7$.
  - First solution: $83 \cdot 72 = 5976$
    $a = 5976 \mod 7 = 5$.
  - Second solution: $83 \equiv 6 \mod 7$, $72 \equiv 2 \mod 7$,
    $83 \cdot 72 \equiv 6 \cdot 2 \equiv 12 \equiv 5 \mod 7$.

Multiplicative inverse

- Multiplicative inverse :
  - Let $n > 0$ and $a \in \mathbb{Z}$. An integer a' is a *multiplicative inverse* of a modulo n if $a \cdot a' \equiv 1 \mod n$.
- Theorem :
  - Let $n, a \in \mathbb{Z}$ with $n > 0$. Then a has a multiplicatif inverse modulo n iff $\text{PGCD}(a, n) = 1$.
  - Proof ($\Rightarrow$)
    - If a' is a multiplicative inverse of a modulo n, then $a \cdot a' \equiv 1 \mod n$.
    - Let $k \in \mathbb{Z}$ such that $a \cdot a' = 1 + k \cdot n$.
    - If $d|a$ and $d|n$, then $d|1$. Therefore $\text{PGCD}(a, n) = 1$.

Example

- A multiplicative inverse of 5 modulo 7 is 3 because

$$3 \cdot 5 \equiv 15 \equiv 1 \mod 7$$

- 2 has no multiplicative inverse modulo 6 :
  - $2 \cdot 1 \equiv 2 \mod 6$
  - $2 \cdot 2 \equiv 4 \mod 6$
  - $2 \cdot 3 \equiv 0 \mod 6$
  - $2 \cdot 4 \equiv 2 \mod 6$
  - $2 \cdot 5 \equiv 4 \mod 6$

Euclid's extended algorithm

- Euclid's extended algorithm
  - Let $a, b \in \mathbb{Z}$ and $d = \mathrm{PGCD}(a, b)$.
  - Computes $s, t \in \mathbb{Z}$ such that $a \cdot s + b \cdot t = d$.
- Multiplicative inverse.
  - Let a, n with $n > 0$ and $\mathrm{PGCD}(a, n) = 1$.
  - With Euclid's extended algorithm, one computes s, t such that
$$a \cdot s + n \cdot t = 1$$
  - Then $a \cdot s \equiv 1 \mod n$
  - s is one multiplicative inverse of a modulo n.

Euclid's extended algorithm

- Euclid's extended algorithm, for $a > 0$ and $b \geq 0$.
  - Two additional sequences u_i and v_i.
  - $r_0 = a$ and $r_1 = b$.
  - For $i \geq 0$, let $r_i = q_i \cdot r_{i+1} + r_{i+2}$
  - $u_0 := 1,\ v_0 := 0,\ u_1 := 0,\ v_1 := 1$ and for $i \geq 2$, one defines $u_i = u_{i-2} - q_{i-2} \cdot u_{i-1}$ and $v_i = v_{i-2} - q_{i-2} \cdot v_{i-1}$.
- There exists $k > 0$ such that $r_k = 0$.
  - Then $\text{PGCD}(a, b) = r_{k-1} = u_{k-1} \cdot a + v_{k-1} \cdot b$.

Proof

- We always have $r_i = u_i \cdot a + v_i \cdot b$.
  - True for $r_0 = a = 1 \cdot a + 0 \cdot b$.
  - True for $r_1 = b = 0 \cdot a + 1 \cdot b$.
  - If $r_{i-2} = u_{i-2} \cdot a + v_{i-2} \cdot b$ and $r_{i-1} = u_{i-1} \cdot a + v_{i-1} \cdot b$, then :

$$\begin{aligned} u_i \cdot a + v_i \cdot b &= (u_{i-2} - q_{i-2} \cdot u_{i-1}) \cdot a + \\ &\quad (v_{i-2} - q_{i-2} \cdot v_{i-1}) \cdot b \\ &= r_{i-2} - q_{i-2} \cdot r_{i-1} \\ &= r_i \end{aligned}$$

Modular arithmetic

- Let an integer $n > 1$ called the modulus.
- Modular reduction
  - $r := a \mod n$, remainder of the division of a by n.
  - $0 \leq r < n$
  - Ex: $11 \mod 8 = 3$, $15 \mod 5 = 0$.
- Congruence:
  - $a \equiv b \mod n$ if $n|(a - b)$.
  - $a \equiv b \mod n$ iif a and b have same remainder modulo n.
  - Ex: $11 \equiv 19 \mod 8$.
  - If $r := a \mod n$, then $r \equiv a \mod n$.

Modular arithmetic

- If $a_0 \equiv b_0 \mod n$ and $a_1 \equiv b_1 \mod n$
  - $a_0 + a_1 \equiv b_0 + b_1 \mod n$
  - $a_0 - a_1 \equiv b_0 - b_1 \mod n$
  - $a_0 \cdot a_1 \equiv b_0 \cdot b_1 \mod n$
- Integers modulo n
  - Integers modulo n are $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$
  - Addition, subtraction or multiplication in $\mathbb{Z}_n$ is done by first doing it in $\mathbb{Z}$ and then reducing the result modulo n.
  - For example in $\mathbb{Z}_7$:
    - $6 + 4 = 3$, $3 - 4 = 6$, $3 \cdot 6 = 4$.