Theoretical foundations

Introduction to computational number theory - Part 7

Jean-Sébastien Coron

Université du Luxembourg

November 8, 2009

Summary

- C programming
  - Functions
- Algorithmic number theory
  - Euclidean division

Functions

- ```
  double max(double a,double b)
  {
    double m;
    if(a>b)
    {
      m=a;
    }
    else
    {
      m=b;
    }
    return m;
  }
  ```
  ```

Using functions

- For function
  `double max(double a,double b)`
  - Let `x,y,z` be variables of type `double`.
  - Then instruction

    ```
    z=max(x,y);
    ```

    applies function `max` to variables `x` and `y`.
  - and stores the result in `z`.

Call by value

- When function is called, the value of variables given as argument are copied in the parameter variables of the function.
  - `double max(double a,double b)`
  - `z=max(x,y);`
  - The content of variables `x` and `y` is copied into `a` and `b`.
- Call by value
  - If the content of variables `a` or `b` is modified inside the function, this does not affect variables `x` and `y`.

Call by reference

- We would like to modify the value of variables given as argument.
  - We would like a function `swap(u,v)` that swaps the variables.

```
void swap(int a,int b) {
  int m=a;  a=b;  b=m;
}
int main()
{
  int u=1; int v=2;
  swap(u,v);
  printf("u=%d v=%d\n",u,v); // u=1 v=2
}
```

Analysis

- The previous example does not work.
  - The function `swap` only swap the values of variables `a,b`, not the values of `u,v`.
- Solution: use pointers:
  - We give to `swap` the address of variables `u,v`.
  - The function `swap` will exchange the values at these two adresses.
  - One call `swap(&u,&v);`

- Address of a variable d'une variable=pointer
  - The function `swap` takes as input two pointers.

```
void swap(int *a,int *b) {
  int m=*a;
  *a=*b;  *b=m;
}
int main()
{
  int u=1;  int v=2;
  swap(&u,&v);
  printf("u=%d v=%d\n",u,v); // u=2 v=1
}
```

Conclusion

- When do we use call by reference ?
  - When we want to modify the value of a variable given as argument.
  - Otherwise, it is better to use call by value.

```
void addition(int a,int b,int *c) {
  *c=a+b;
}
int main()
{
  int u=1;  int v=2; int w;
  addition(u,v,&w);
  printf("w=%d\n",w); // w=3
}
```

Algorithmic number theory

- Goal: modular computation with large integers.
  - Addition, multiplication, inversion modulo n.
- Euclidean division:
  - Given a, b, find q, r such that

$$a = b \cdot q + r$$

    where a, b are big integers.

Substraction

- Computing $c = a - b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k \geq \ell \geq 1$.
    Let $c = (c_k c_{k-1} \dots c_0)$
    $carry \leftarrow 0$
    for $i = 0$ to $\ell - 1$ do
    $\quad tmp \leftarrow a_i - b_i + carry$
    $\quad carry \leftarrow tmp/B$; $c_i \leftarrow tmp \mod B$
    for $i = \ell$ to $k - 1$ do
    $\quad tmp \leftarrow a_i + carry$
    $\quad carry \leftarrow tmp/B$; $c_i \leftarrow tmp \mod B$
    $c_k \leftarrow carry$
  - If $a \geq b$ then $c_k = 0$, otherwise $c_k = -1$.
  - If $c_k = -1$, compute $c' = b - a$ and let $c := -c'$.

Division with remainder

- Let $a = (a_{k-1} \ldots a_0)_B$ and $b = (b_{\ell-1} \ldots b_0)_B$ with $a > b > 0$ and $b_{\ell-1} \neq 0$.
  - Compute q and r such that $a = b \cdot q + r$ and $0 \leq r < b$.
  - $q = (q_{m-1} \ldots q_0)_B$, with $m := k - \ell + 1$.
- Algorithm overview:

$r \leftarrow a$
for $i = m - 1$ downto 0 do
  $q_i \leftarrow r/(B^i b)$
  $r \leftarrow r - B^i \cdot q_i \cdot b$
output r

Division with remainder

- For all i, $0 \leq r < B^i \cdot b$ after step i
  - Therefore, $0 \leq r < b$ eventually.
- How to compute $q_i = r/(B^i \cdot b)$
  - Test all possible values of $0 \leq q_i < B$
  - Not efficient, except if B is small (*e.g.* $B = 10$).
  - Possible to do much better

Division with remainder

- Complete algorithm (for small B)

$r \leftarrow a$
for $i = m - 1$ downto 0 do
  $q_i \leftarrow 0$
  while $r >= 0$
    $r \leftarrow r - B^i \cdot b$
    $q_i \leftarrow q_i + 1$
  $q_i \leftarrow q_i - 1$
  $r \leftarrow r + B^i \cdot b$
output r

Computing in $\mathbb{Z}_n$

- Computing $c = a + b$ in $\mathbb{Z}_n$
  - Let $c \leftarrow a + b$ in $\mathbb{Z}$
  - Let $c \leftarrow c \mod n$.
  - Complexity: $\mathcal{O}(\log n)$
- Computing $c = a \cdot b$ in $\mathbb{Z}_n$
  - Let $c \leftarrow a + b$ in $\mathbb{Z}$
  - Let $c \leftarrow c \mod n$.
  - Complexity: $\mathcal{O}(\log^2 n)$.