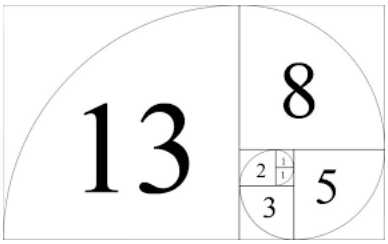
La suite de Fibonacci

$\pi\lambda$

23 Novembre 2017

Résumé

L'objectif de cette séance de travaux-pratiques est de mettre en évidence quelques observations faites en cours sur le calcul des nombres de la suite de Fibonacci. Une occasion de présenter les outils `bc`, `gcc`, `gdb`, `objdump` et `gprof`.



1 Arithmétique modulaire sur 64 bits

Soit n un module. L'ordre multiplicatif modulaire d'un entier x premier avec n désigne le plus petit entier f tel que $x^f \equiv 1 \pmod{n}$. Un tel entier est bien défini à condition que x soit inversible modulo n. En effet, l'entier k variant, la suite $x^k \pmod{n}$ prenant ses valeurs dans un ensemble fini, il existe $j > i$ tel que :

$$x^j = x^i \pmod{n} \Longrightarrow x^{j-i} = 1 \pmod{n},$$

on peut alors montrer que f divise $j - i$, et c'est bien le plus petit entier qui vérifie $x^k = 1 \pmod{n}$. Aprés compilation de la source `arithmos.c`, on obtient :

```
mesure> gcc -Wall arithmos.c -o arithmos.exe
mesure> x=2147483648
mesure> n=658812288653553079
mesure> ./arithmos.exe $x $n
ordre de 2147483648 modx 658812288653553079 > 255
mesure> bc <<< "( $x ^ 3 ) % $n"
1
```

1. Il y a un soucis, pourquoi ?

```c
#include <stdio.h>
#include <stdlib.h>
/*
 * test l'ordre multiplicatif d'un entier modulo un autre
 * status : bug
*/

typedef unsigned long long nombre;

int ordre(nombre x, nombre m)
{
    int res = 1;
    nombre y = x;
    while (y != 1 && res < 255) {
        y = (x * y) % m;
        res++;
    }
    return res;
}

int main(int argc, char *argv[])
{
    nombre x = atoll(argv[1]);
    nombre m = atoll(argv[2]);
    int f = ordre( x, m );
    if ( f < 255 )
        printf ("ordre de %Ld modx %Ld = %d\n", x, m, f);
    else
        printf ("ordre de %Ld modx %Ld > 255\n", x, m);
    return 0;
}
```

arithmos.c

2. Faire un diagnostic !
3. Décrire sommairement une solution pour sortir de ce mauvais pas.

Indication : utiliser `bc -l` pour calculer le logarithme à base 2 de x et n.

2 Fibonacci

On rappelle que la suite de Fibonacci est définie par les règles suivantes :

$$F_0 = 0, \quad F_1 = 1, \qquad \forall n > 1, \quad F_n = F_{n-1} + F_{n-2}.$$

Il s'agit mettre en oeuvre le calcul des nombres de Fibonacci dans des programmes en langage C. Les implantations doivent contenir une fonction `ullong fib(int n)` où `ullong` est un type entier de 64 bits. Les commandes associées prendront un entier n en argument pour écrire la valeur de F_n sur la sortie standard.

2.1 méthode itérative

1. Modifier la commande pour mettre en évidence la limite de $\frac{F_{n+1}}{F_n}$.
2. Proposer un équivalent de F_n en fonction de n et du nombre d'or.
3. Déterminer empiriquement le domaine de validité de la commande en affichant les premiers termes des suites F_n et F_{n+1}/F_n.
4. Utiliser l'option $-S$ compilateur pour mettre en évidence le nombre d'opérations machines générées par le compilateur pour la partie instruction de la boucle de `fib`.
5. Retrouver les résultats avec les commandes `gdb` et `objdump`.
6. Ecrire un script basé sur la commande `bc` pour calculer le n-ième nombre de Fibonacci.
7. Quel est le nombre de chiffre décinaux de F_{1000} ?
8. Retrouver ce résultat à partir de l'équivalent (4).

2.2 méthode récursive

On note $T(n)$ le temps de calcul de la version récursive.

1. Pour quelle valeur de n peut-on calculer F_n en moins d'une minute.
2. Utiliser la commande `gnuplot` pour tracer le graphe de $\log T(n)$.
3. Proposer un équivalent de $T(n)$ en fonction de F_n.
4. Utiliser le profileur `gprof` pour déterminer le nombre d'appels résursif pour calculer F_{20}.