Preuves et Analyses des Algorithmes Séances de Travaux-Pratiques

Ph. Langevin

Année Universitaire 2008

Contents

1 Préambule

L'objectif de ces séances de travaux-pratiques est de faire des expériences numériques sur des implantations de quelques algorithmes qui ont été vus et étudiés en cours.

Vous prendrez soin d'organiser votre travail en créant un répertoire de travail principal (`tp31`, par exemple), dans lequel vous placerez plus tard des répertoires secondaires correspondants aux différents sujets qui seront abordés.

Vous utiliserez de façon systématique la commande `make` de sorte à automatiser la construction et la compilation de vos fichiers, les options de compilation `-Wall` et `-g` sont obligatoires. Vous éviterez de faire des programmes et commandes interactives, il s'agit de prendre rapidement l'habitude de gérer les options de la ligne de commande avec la fonction `getopt` !

La section (8) contient quelques exemples de sources pour vous aider à démarrer.

Enfin, je vous suggère d'utiliser un editeur de texte tel que `nedit`, `emacs`, et `vi`.

2 Suite de Fibonacci

La suite de Fibonacci F_n est la suite d'entiers positifs définie par la relation de récurrence d'ordre deux :

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

Nous avons vu en cours que F_n s'exprime à partir des deux racines l'équation $X^2 = X + 1$ qui sont $\phi \approx 1.618$ et $\hat{\phi} \approx -0.6$, par :

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

En particulier, F_n croît exponentiellement, plus précisément, F_n est équivalent à $\frac{1}{\sqrt{5}}\phi^n$ au voisinage de l'infini. Par exemple $F_{10} = 55$,

$$F_{100} = 354224848179261915075$$

et F_{1000} est un nombre de 289 chiffres décimaux : 43 466 557 686 937 456 435 688 527 675 040 625 802 564 660 517 371 780 402 481 729 089 536 555 417 949 051 890 403 879 840 079 255 169 295 922 593 080 322 634 775 209 689 623 239 873 322 471 161 642 996 440 906 533 187 938 298 969 649 928 516 003 704 476 137 795 166 849 228 875

L'objectif de cette séance de travaux-pratiques est de comparer les implantations itérative et récursive qui ont étée vues et analysées en cours. Les calculs seront faits en utilisant des entiers non signés de 64 bits.

2.1 Fibonacci récursif

Commencez par implanter une fonction récursive `ullong fibrec(int n)` pour calculer le n-eme terme de la suite de Fibonacci. Ecrire une commande `fib.exe` qui prend un entier n en argument sur la ligne de commande puis calcule et affiche la valeur de F_n. Assurez-vous du bon fonctionnement de votre fonction.

On note $T_r(n)$ le temps de calcul de cette commande.

1. Utiliser la commande `/usr/bin/time` pour faire des mesures de temps de calcul.
2. Faire une représentation graphique avec la commande `gnuplot`.
3. Déterminer les meilleures constantes A et B tel que $T_r(n) = AB^n$.

2.2 Fibonacci itératif

Implanter une fonction itérative `ullong fibiter(int n)` pour calculer le n-eme terme de la suite de Fibonacci. Modifier la commande `fib.exe` pour choisir le mode de calcul (itératif ou récursif) par un jeu d'options approprié (`-r` ou `-i`).

1. Utiliser la commande `/usr/bin/time` pour faire des mesures de temps de calcul.
2. Faire une représentation graphique avec la commande `gnuplot`.
3. Déterminer une formule du temps de calcul de la version itérative.
4. Déterminer le domaine de validité de la fonction `fibiter( )`.

2.3 Chronométrage

Pour chronométrer les temps de calcul, vous utiliserez la commande `/usr/sbin/time` qui envoie ses résultats sur la sortie d'erreur standard. Le chronométrage de la commande `sleep 5` est obtenu par :

```
/usr/bin/time --format=" duree=%E cpu=%U" sleep 5
```

qui affichera :

```
duree=0:05.01 cpu=0.00
```

L'information de chronométrage passe par le canal standard d'erreur. Pour récupérer ce résultat dans un fichier, il faut *rediriger* ce canal vers la sortie standard :

```
/usr/bin/time --format=" duree=%E cpu=%U" sleep 5 2>&1 | grep duree > tempo.txt
```

3 Algorithmes Additifs

L'objectif de cette séance de travaux pratiques est de réaliser une commande `fib -n: [-b:]` pour calculer le n-ième terme de la suite de fibonacci dans la base spécifiée.

3.1 Registre

Un nombre est représenté par un tableau de chiffres dans une certaine base. Pour des raisons de commodités, dans les programme que vous réaliserez, tous les nombres seront représentés dans la même base, et ils auront la même taille; c'est l'approche *registre*. La base et la taille seront définis par l'utilisateur. Ci-dessous un exemple de programme qui initialise un nombre à une certaine valeur, puis affiche ce nombre.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int TAILLE = 10;
int  BASE  =  2;

typedef    int    CHIFFRE;
typedef CHIFFRE * NOMBRE;

NOMBRE init( int v )
{ NOMBRE res;
  int  i = 0;
  res = ( NOMBRE ) calloc( TAILLE, sizeof( CHIFFRE ) );
  while ( v ) {
    res[ i ] = v % BASE;
    v = v / BASE;
    i = i + 1;
  }
  return res;
}
void print( NOMBRE z )
{ int err=0;
  int  i = TAILLE - 1;
  while ( 0 == z[ i ] && i >= 0 )
    i = i - 1;
  while ( i >= 0 ) {
    if ( z[ i ] >= BASE ) err++;
```

```
    if ( z[ i ] < 0      ) err++;
    printf(".%d", z[i] );
    i--;
  }
  if ( err ) {
    printf("\npanic : %d erreurs", err );
    exit( 1 );
  }
}

int main( int argc, char *argv[] )
{
  int opt;
  char *optlist ="b:t:v:";
  int valeur;
  NOMBRE z;
  while ( ( opt = getopt( argc, argv , optlist) ) > 0  )
          switch( opt ){
          case 'b': BASE    = atoi( optarg ); break;
          case 't': TAILLE  = atoi( optarg ); break;
          case 'v': valeur  = atoi( optarg ); break;
          default : printf("\nusage %s -> %s", argv[0], optlist);
                    exit ( 1 );
          }

  z = init( valeur );
  print( z );
  free( z );
  printf("\n");
  return 0;
}
```

3.2 algorithmes additifs

Implanter les algorithmes additifs de base du cours. Les implantations devront détecter les *dépassements de capacité* (overflows).

1. `Inc( NOMBRE z )` qui réalise $z \leftarrow z + 1$.
2. `Add( NOMBRE y, x )` qui réalise $y \leftarrow x + y$.
3. `Dec( NOMBRE z )` qui réalise $z \leftarrow z + 1$.

3.3 nombres de Fibonacci

1. Implanter la fonction `int taille(int b, n)` qui détermine un majorant de la taille de F_n en base b.

2. Implanter une fonction `NOMBRE fib( int n)`

3. Véfifier le bon fonctionnement de votre programme en calculant F_{1000} en base 10.

4. Déterminer une expression du temps de calcul en fonction de n en base 10.

5. Pour quelles valeurs de n pourrait-on calculer F_n en moins de 15 minutes ?

4 Algorithmes Multiplicatif

L'objectif de cette séance de travaux pratiques est de réaliser une commande `fact -n: [-b:]` pour calculer le $n!$ dans la base spécifiée.

4.1 Algorithmes multiplicatifs

Implanter les algorithmes additifs de base du cours. Les implantations devront détecter les *dépassements de capacité* (overflows).

1. `cmul( NOMBRE z , CHIFFRE c)` qui réalise $z \leftarrow c * z$.
2. `prd( NOMBRE y, x )` qui réalise $y \leftarrow x * y$. Cette fonction sera utilisée dans le prochain sujet. L'implantation peut être différée.

4.2 Factorielle

1. Implanter la fonction `int taille(int b, n)` qui détermine un majorant de la taille de $n!$ en base b.
2. Implanter une fonction `NOMBRE fact( int n)`
3. Véfifier le bon fonctionnement de votre programme en calculant 100! en base 10.
4. Déterminer une expression du temps de calcul en fonction de n en base 10.
5. Pour quelles valeurs de n pourrait-on calculer $n!$ en moins de 15 minutes ?

5 Dichotomie

L'objectif de cette séance de travaux pratiques est de réaliser une commande `racine -n: [-b:]` pour calculer la racine carrée entière du nombre n dans la base spécifiée.

5.1 Algorithmes de base

Implanter les algorithmes de base du cours. Les implantations devront détecter les *dépassements de capacité* (overflows).

1. `int cmp( NOMBRE y, x)` pour comparer x et y.
2. `cdiv( NOMBRE z, CHIFFRE c)` qui réalise $z \leftarrow z/c$.
3. `prd( NOMBRE y, x )` qui réalise $y \leftarrow x * y$.

5.2 Racine carrée

Implanter le calcul de la racine carrée d'un nombre par une approche dichotomique.

1. `void racine( NOMBRE z )` qui réalise $z \leftarrow \sqrt{z}$.

6 Algorithme d'Euclide

L'objectif de cette séance de travaux pratiques est de vérifier expérimentalement le comportement du temps de calcul de l'algorithme d'Euclide. Etant donnés deux entiers positifs a et b, on note $I(a,b)$ le nombre d'itération de l'algorithme d'Euclide pour déterminer $\text{PGCD}(a,b)$.

6.1 Nombre d'itération

1. Implanter `ullong pgcd(ullong a,ullong b)` en utilisant l'algorithme itératif. Vérifier le bon fonctionnement de votre fonction.

2. Implanter `ullong iter(ullong a,ullong b)` qui retourne le nombre d'itérations de l'algorithme d'Euclide.

3. Utiliser `gnuplot` pour représenter le graphe de la fonction

$$a \mapsto \sup_{0 \leq b \leq a} I(a,b).$$

6.2 Euclide étendu

1. Implanter une version récursive `void bbrec(ullong a, ...*u, ullong b, ...*v)` de l'algorithme d'Euclide.

2. Implanter une version itérative `void bbitr(ullong a, ...*u, ullong b, ...*v)` de l'algorithme d'Euclide.

3. Vérifier le bon fonctionnement de ces deux implantations.

4. Faire des versions "verbeuses" qui tracent l'évolution des variables au cours des étapes intermédiaires.

7 Exponentiation

8 Exemples

8.1 Source

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  void decomp( int n, int b)
6  { int t;
7    if ( n == 0 ) return;
8    t = n % b;
9    decomp( n / b, b );
10   if ( t > 9 )
11       printf("%c", 55 + t );
12     else
13       printf("%d",   t);
14   }
15
16
17 int main( int argc, char *argv[] )
18 {
19   int opt;
20   char *optlist ="n:b:";
21
22   int base = 2, n = 0;
23
24   while ( ( opt = getopt( argc, argv , optlist) ) > 0  )
25           switch( opt ){
26           case 'n': n    = atoi( optarg ); break;
27           case 'b': base = atoi( optarg ); break;
28           default : printf("\nusage %s -> %s", argv[0], optlist);
29                     exit ( 1 );
30           }
31
32   printf("Decomposition de %d en base %d :", n, base);
33   decomp( n , base);
34
35   return 0;
36 }
```

8.2 makefile

```
all : decomp.exe

val = 123456
decomp.exe : decomp.c
	gcc -Wall -g decomp.c -o decomp.exe

demo:
	echo
	./decomp.exe -n $(val)
	echo
	./decomp.exe -n $(val) -b5
	echo
	./decomp.exe -n $(val) -b16


data.txt : decomp.exe
	./script.sh 1000 5

loop :
	for (( n = 1; n <= 10 ; n = $$n + 1)); do  \
	./decomp.exe -n $$n; \
	done


exp :
	gnuplot plot.com

data.fig:  data.txt
	gnuplot plot.cmd

data.pdf: data.fig
	fig2dev -Lpdf data.fig data.pdf

doc.pdf : data.pdf decomp.c doc.tex
	pdflatex doc
	pdflatex doc
```

8.3 script

```
max=$1
pas=$2
rm -f data.txt
touch data.txt
for (( n = 1; n <= $max ; n = $n + $pas)); do
    echo
    ./decomp.exe -n $n
    echo -n $n" " >> data.txt
    ./decomp.exe -n $n | sed 's/.*://' | wc -c >> data.txt
done
```

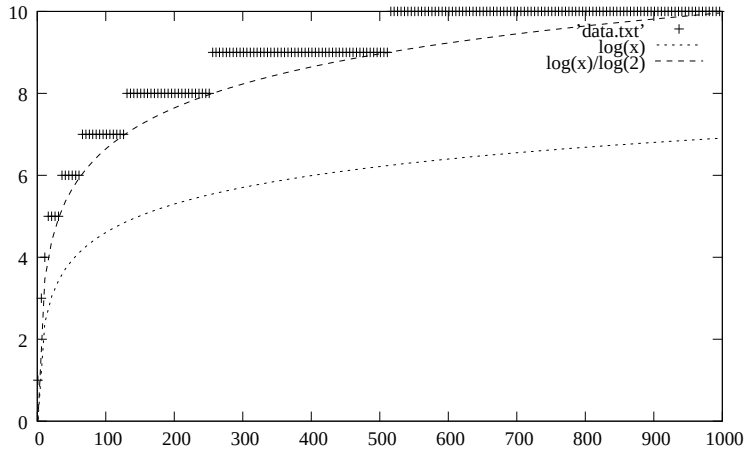
8.4 gnuplot

Un exemple de fichier pour tester,

```
plot 'data.txt', log(x) , log(x)/log(2)
pause -1;
```

Un fichier pour construire une image au format `fig` pour la commande `xfig` ou l'outil de conversion d'image `fig2dev`.

```
set terminal fig
set output 'data.fig'
plot 'data.txt', log(x) , log(x)/log(2)
```



8.5 latex

Un exemple de source `latex`

```
\documentclass[a4paper, 12pt]{article}
\usepackage{amsmath, amssymb}




\usepackage{ifpdf}

\ifpdf
  \usepackage[pdftex]{color,graphicx}
  \usepackage[pdftex ,colorlinks=true,
        urlcolor=blue,
        citecolor=blue,
        pdfstartview=FitH]{hyperref}
\else
  \usepackage[dvips]{color}
  \usepackage[pagebackref,
              colorlinks=true,urlcolor=blue,citecolor=blue]{hyperref}
\fi



\usepackage{listings}

\lstset{
    basicstyle=\small,
    keywordstyle=\color{black}\bfseries,

    identifierstyle=,
    commentstyle=\color{white},
    stringstyle=\ttfamily,
    emphstyle=\color{red},
    showstringspaces=false}



\def\gcc#1{\texttt{\textcolor{blue}{#1}}}
\def\shell#1{\texttt{\textcolor{red}{#1}}}
\def\tty#1{\texttt{\textcolor{magenta}{#1}}}
```

```
\title{D\'ecomposition d'un entier}
\date{Novembre 2008}
\author{Pavle Michko}



\begin{document}

\maketitle

\tableofcontents

\section{Source}

\lstinputlisting[emph={getopt}, numbers=left , language={C}]{decomp.c}


\section{R\'esultat num\'erique}

\begin{center}
\rotatebox{-90}{\includegraphics[scale=0.6]{data}}
\end{center}

\end{document}
```