|  |
| --- |
| Magnan cedric  Carteron Alexandre |
| RAPPORT TP2 |
| IGI-2002 |
|  |
| **Esiee Paris** |

|  |
| --- |
|  |

1. Exercice 1

Le but de l’exercice est d’écrire le programme qui réalise le calcul de la fonction de Fibonacci en temps logarithmique (voir TD2 exercice 4.b).

**Declaration des structures M et V**

struct V{ // déclaration de la structure "vect" de 1 ligne 2 colonnes

int t[2];

};

struct M{ // déclaration de la structure "mat" de 2 lignes 2 colonnes

int t[2][2];

};

**Fonction f(n) : retourne Fibo(n)**

int f(int n) {

struct V F1={{0,1}};

struct M M1={{{0,1},{1,1}}} ;

struct M Mn1;

Mn1= puissmat(M1,n-1);

struct V Fn;

Fn=prodmat(Mn1,F1);

return Fn.t[1];

}

**Fonction puissmat(A,n) : retourne An (où A est une matrice 2\*2)**

Propriété:

I(P,A,n) : P\*An=M1n où P,A et M1 sont des matrices 2x2

Init :

P=

Arrêt :

n=0

Progression :

* I(P,A,n) **et** n≠0 **et** n1 pair **=>**  I(P, A2,n/2)
* I(P,A,n) **et** n≠0 **et** n1 impair **=>**  I(P\*A,A2,n/2)

struct M puissmat(struct M A, int n){

struct M P={{{1,0},{0,1}}} ;

while(n!=0){

if((n%2)!=0)

P=prodmat(P,A);

A=prodmat(A,A);

n=n/2;

}

return P;

}

**Fonction prodmat(A,B) : retourne A\*B (où A et B sont des matrices 2x2)**

struct M prodmat(struct M A,struct M B){ //A\*B

struct M C;

int i,j,k;

for(i=0; i<2 ;i++){

for(j=0; j<2 ;j++){

C.t[i][j]=0; // Mise à zéro des cases de la structure C

for(k=0; k<2 ;k++){

C.t[i][j]+= A.t[i][k]\*B.t[k][j];

}

}

}

return C;

}

**Fonction prodmatvect(A,V1) : retourne A\*V1 (où A est une matrice 2x2 et V1**

**une matrice 1x2)**

struct V prodmatvect(struct M A, struct V V1){ // A\*V1

int i,j;

struct V W;

for(i=0; i<2; i++){

W.t[i]=0; // Mise à zéro des cases de la structure W

for(j=0; j<2; j++)

W.t[i] += A.t[i][j] \* V1.t[j];

}

return W;

}

**Fonction affichant f(n)**

int main() {

int n;

printf("n=");

scanf("%d",&n);

printf("F(%d)=%d\n",n,f(n));

return 0;

}

**Exemple d’exécution**

F(0) = 0

F(1) = 1

F(2) = 1

F(3) = 2

F(4) = 3

F(5) = 5

F(10) = 55

F(15) = 610

F(20) = 6765

F(30) = 832040

F(40) = 102334155

**Graphique time(f(n)) : temps d’exécution de f(n) de 0 à 40**

Remarque : Pour connaître les coordonnées des points de la courbe, on écrit dans le terminal

« time ./NomDuFichierExecutable » tout en supprimant la fonction scanf() et en indiquant les différentes valeurs de n dans la fonction main(). On a ainsi les temps d’exécution de notre programme pour différentes valeurs de n.

On obtient le graphique suivant :

1. Exercice 2

Le but de l’exercice est de tester le programme qui réalise le calcul de la fonction de Fibonacci par récursivité.

* **Fonction f2(n) : retourne Fibo(n)**

**#include <stdio.h>**

**int f2(int n){**

**if(n==0) return 0;**

**if(n==1) return 1;**

**return fib2(n-2) + fib2(n-1);**

**}**

**int main() {**

**int n;**

**printf("n=");**

**scanf("%d",&n);**

**printf("F2(%d)=%d\n",n,fib2(n));**

**return 0;**

**}**

* **Exemples d’exécutions**

F2(0) = 0

F2(1) = 1

F2(2) = 1

F2(3) = 2

F2(4) = 3

F2(5) = 5

F2(10) = 55

F2(15) = 610

F2(20) = 6765

F2(30) = 832040

F2(40) = 102334155

F2(50) = 12586269025

Remarque : on notera tout de suite que fibo(50) prend extrêmement longtemps à s’exécuter, cette fonction est donc largement moins performante que celle du premier exercice.

**Graphique t(f(n)) : temps d’exécution de f(n) en millisecondes pour n allant de 0 à 50**

Au vue de ce graphique, ce programme s’exécute en temps exponentielle.

1. Exercice 3

Le but de l’exercice est d’écrire un programme qui retourne le nombre de couples tels que T[i]+T[j]=v (v étant donné) en temps linéaire.

* **Fonction exo3(T,v) : retourne x=le nombre de couple (i,j)** **∈ [0;n-1]x[0;n-1] tels que T[i]+T[j]=v**

Propriété :

I(i’,j’,x) : x est le nombre de couples (i,j) ∈ [0;i’]x[j’;n-1] tels que T[i]+T[j]=v

Init :

i’=0

j’=n-1

x=0

Arrêt :

i’=n **et** j’=-1

Progression :

* I(i’,j’,x) **et** T[i’]+T[j’]>v **et** i’≠n **et** j’≠-1 **** I(i’,j’-1,x)
* I(i’,j’,x) **et** T[i’]+T[j’]=v **et** i’≠n **et** j’≠-1 **** I(i’+1,j’-1,x+1)
* I(i’,j’,x) **et** T[i’]+T[j’]<v **et** i’≠n **et** j’≠-1 **** I(i’+1,j’,x)

Programme :

private static int exo3(int[] T, int v) {

int n=T.length;

int ip=0,jp=n-1,x=0; // I(I’,j’,x)

while( (ip!=n)&&(jp!=-1) ) { // I(I’,j’,x) et i’≠n et j’≠-1

if (T[ip]+T[jp] < v) // T[i’]+T[j’]<v  I(i’+1,j’,x)

ip++; // I(i’,j’,x)

else if (T[ip]+T[jp] > v) // T[i’]+T[j’]>v  I(i’,j’-1,x)

jp--; // I(i’,j’,x)

else if (T[ip]+T[jp] == v) { // T[i’]+T[j’]=v  I(i’+1,j’+1,x+1)

ip++; // I(i’,j’-1,x+1)

jp--; // I(i’,j’,x+1)

x++; // I(i’,j’,x)

}

} // I(I’,j’,x) et i’=n et j’=-1 ≡ S

return x;

}

* **Complexité**

Initialisation :

TINIT = cte = α

Temps de boucle : majoré par le cas où

TBOUCLE < cte = βn

*Remarque : Dans ce cas-ci, on a le nombre minimum d’exécutions de boucles mais comme à chaque boucle, il y a plus (ou autant) d’instructions à effectuer que dans les autres cas, on a le temps d’exécution de la boucle maximum.*

Conclusion :

Texo3 = α+βn  temps linéaire

* **Exemples d’exécutions**

T([0...13]= [-4,-2,-1,0,2,3,5,6,8,9,11,13,14,15] et v=10

(i,j)=(0,12) : -4+14=10

(i,j)=(2,10) : -1+11=10

(i,j)=(4,8) : 2+8=10

(i,j)=(6,6) : 5+5=10

(i,j)=(8,4) : 8+2=10

(i,j)=(10,2) : 11+-1=10

(i,j)=(12,0) : 14+-4=10

x=7

T([0...3]= [0,1,2,3] et v=7

x=0

T([0...10]= [0,1,2,3,5,8,13,21,34,55,89] (suite de Fibonacci) et v=34

(i,j)=(0,8) : 0+34=34

(i,j)=(6,7) : 13+21=34

(i,j)=(7,6) : 21+13=34

(i,j)=(8,0) : 34+0=34

x=4

1. Exercice 4

Le but de l’exercice est d’écrire un programme qui retourne le nombre de couples tels qu’il existe vérifiant en temps quadratique.

* **Fonction exo4(T) : retourne c=le nombre de couple (i,j)** **∈ [0;n-1]x[0;n-1] tels qu’il existe k∈ [0;n-1] vérifiant T[i]+T[j]=T[k]**

Propriété :

I(Vi,c) : avec x(k) le nombre de couples (i,j) ∈ [0;n-1]x[0;n-1] tels que T[i]+T[j]=T[k]

Init :

Vi=0

c=0

Arrêt :

Vi=n

Progression :

I(Vi,c) **et** Vi≠n **** I(Vi+1,c+x(Vi))

Programme :

public static int exo4(int[] T){

int n=T.length;

int x=0,c=0,Vi=0; // I(Vi,c)

while (Vi!=n) { // I(I’,j’,x) etVi≠n  I(Vi+1,c+x(Vi))

x=exo3(T,T[Vi]);

c+=x; // I(Vi+1,c)

Vi++; // I(Vi,c)

} // I(Vi,c) et Vi=n ≡ S

return c;

}

* **Complexité**

Initialisation :

TINIT = constante = γ

Temps de boucle :

TBOUCLE = n(Tx=exo3(T,T[Vi])+constante) = n(αn+β) = αn²+βn

Conclusion :

Texo4 = αn²+βn+γ  temps quadratique

* **Exemples d’exécutions**

T([0...6]= [-4,-1,0,2,5,7,15]

v=-4 :

(i,j)=(0,2) : -4+0=-4

(i,j)=(2,0) : 0+-4=-4

==>x(-4)=2

v=-1 :

(i,j)=(1,2) : -1+0=-1

(i,j)=(2,1) : 0+-1=-1

==>x(-1)=2

v=0 :

(i,j)=(2,2) : 0+0=0

==>x(0)=1

v=2 :

(i,j)=(2,3) : 0+2=2

(i,j)=(3,2) : 2+0=2

==>x(2)=2

v=5 :

(i,j)=(2,4) : 0+5=5

(i,j)=(4,2) : 5+0=5

==>x(5)=2

v=7 :

(i,j)=(2,5) : 0+7=7

(i,j)=(3,4) : 2+5=7

(i,j)=(4,3) : 5+2=7

(i,j)=(5,2) : 7+0=7

==>x(7)=4

v=15 :

(i,j)=(2,6) : 0+15=15

(i,j)=(6,2) : 15+0=15

==>x(15)=2

c=15 (=somme des x(v))