Chapitre 1

Introduction à la complexité arithmétique

1.1 Notion de récursivité

Une procédure qui fait référence à elle même dans sa définition est une procédure récursive. La notion mathématique de définition par récurrence correspond au principe de programmation par récursivité, l'écriture de procédure récursive est souvent proche de la formule mathématique par récurrence.

Exemple 1 $n! := \begin{cases} 1 \ si \ n = 0 \\ n * (n-1)! \ si \ n \neq 0 \end{cases}$.

La procédure Maple suivante utilise un appel récursif pour calculer la valeur de $n!$

```
nfact :=proc(n)
if n=0 then 1
else n*nfact(n-1)
end;
```

Le processus de calcul déclenché par l'appel nfact(4) peut être représenté par le schéma suivant :

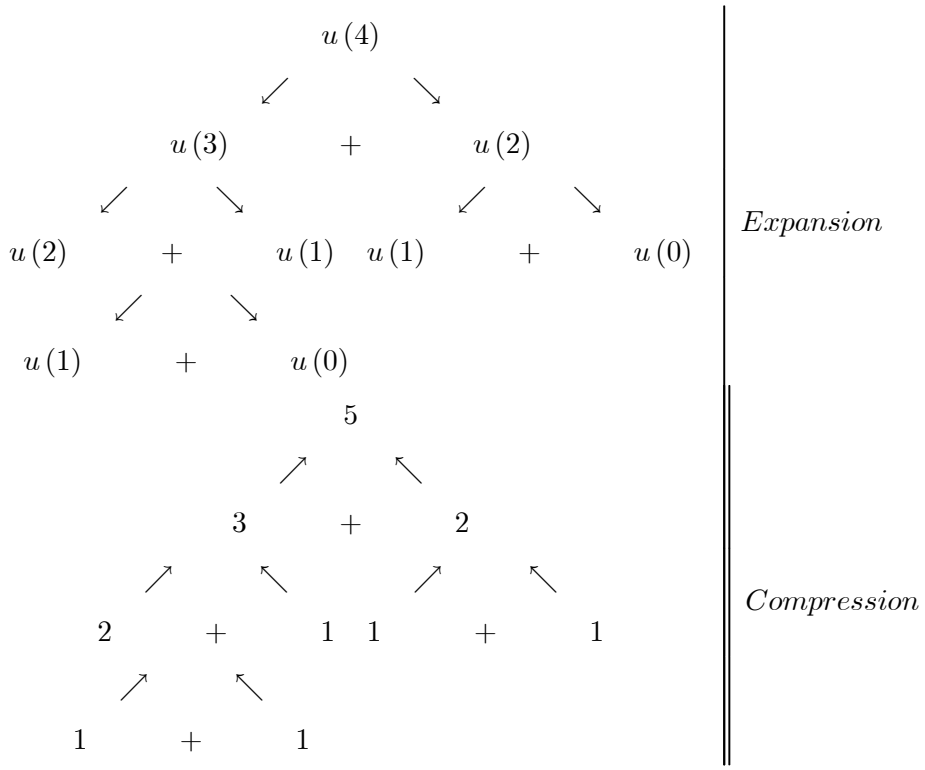
nfact(4)
*4*nfact(3)*
4(3*nfact(2))*
4(3*(2*nfact(1)))*
4(3*(2*(1*1)))*
4(3*(2*1))*
4(3*2)*
*4*6*
24

↘ *Expansion*

↙ *Compression*

Exemple 2 *La suite de Fibonnaci définie par :* $\begin{cases} u(0) = 1 \\ u(1) = 1 \\ u_n = u_{n-1} + u_{n-2} \text{ si } n \geq 2 \end{cases}$.

Le processus de calcul déclenché par l'appel u(4) peut être représenté par le schéma suivant :



Remarque 3 *Les calculs récursifs peuvent être acceléré sous Maple grace à l'option remember.*

Exemple 4

```
> restart :
  fibo := proc(n)
  if n = 0 then 1
  elif n = 1 then 1
  else fibo(n-1) + fibo(n-2);
  fi;
  end:
  t := time( ) :
  fibo(30); time( ) - t;
```

```
1346269
5.523
```

Exemple 5

```
> restart:
  fibo:=proc(n)
  option remember:
  if n=0 then 1
  elif n=1 then 1
  else fibo(n-1)+fibo(n-2);
  fi;
  end:
> t := time(): fibo(30); time()-t
```

```
1346269
0.015
```

1.2 Complexité arithmétique

Définition 6 *Etant données deux fonctions f et g de $\mathbb{N}^*$ dans $\mathbb{R}_+^*$ on dit que $g(n) = \mathcal{O}(f(n))$ s'il existe une constante réelle $c > 0$ telle que $\lim \frac{f(n)}{g(n)} = c$.*

Définition 7 *Le coût arithmétique d'une méthode de calcul est le nombre d'opérations arithmétiques nécessaires à son aboutissement, on l'appelle également complexité arithmétique. On utilise la notation $\mathcal{O}$ pour indiquer le comportement asymptotique de la complexité.*

Nous allons à présent étudier quelques algorithmes et calculer leurs complexité arithmétique.

1.2.1 Calcul de la puissance

1. **La méthode naïve.**

   On désire écrire un algorithme qui permet de calculer x^n pour x, n donné.

   Une première approche pourrait consister en l'évaluation en faisant des multiplications successives comme suit :

   $$x^n = \underbrace{x \times x \times ..... \times x}_{(n-1)\ fois}$$

   Le coût de cette méthode en opération arithmétiques est de $(n-1)$ multiplications et on note $\mathcal{O}(n)$.

2. **L'exponentiation dichotomique.**

   Il existe une autre technique d'évaluation de x^n qui consiste à décomposer le produit x^n en deux parties, on peut décrire ce procédé par la formule :

   $$x^n = \begin{cases} \left(x^{\frac{n}{2}}\right)^2 & si\ n\ est\ pair \\ x\left(x^{\frac{n-1}{2}}\right)^2 & si\ n\ est\ impair \end{cases}$$

Exemple 8 $x^{16} = \left(x^8\right)^2 = \left(\left(x^4\right)^2\right)^2 = \left(\left(\left(\left(x^2\right)\right)^2\right)^2\right)^2$ *ainsi on a besoin de 4 opérations de multiplications pour évaluer* x^{16} *contrairement à 15 opérations pour la méthode naive.*

Exemple 9 *Voici deux programmes le premier nommé naive calcule la puissance par la méthode naive :*

```
restart :
naive := proc(x, n)
local i, P;
P := 1 :
for i to n do
P := P·x :
od:
P :
end:
```

Le second nommé dicho calcule la puissance par la méthode dichotomique :

```
dicho := proc(x, n)
if n = 0 then 1 elif n mod 2 = 0 then dicho(x, n/2)^2 else x*dicho(x, (n-1)/2)^2 fi:
end:
```

Voici à présent une comparaison du temps d'exécution des deux programmes

```
st := time():
naive(9, 10^6):
time() - st;

st := time():
dicho(9, 10^6):
time() - st;
```

```
[Length of output exceeds limit of 1000000]
486.988

[Length of output exceeds limit of 1000000]
0.765
```

1.2.2 Evaluation des polynômes.

1. **Méthode naive**

   Soit $P_n(x) = a_0 + a_1x + ... + a_nx^n$ on peut évaluer ce pôlynome au point α en utilisant directement la définition du pôlynome, cette définition nous impose de calculer les puissances successives de x

   Ainsi pour évaluer un pôlynome par la méthode intuitive on devra procéder comme suit :

$$\begin{cases} calculer\ x^2, ...x^n \\ calculer\ a_1x, a_2x^2...a_nx^n \\ calculer\ a_0 + a_1x + ... + a_nx^n \end{cases}$$

   Le coût dépend de la méthode utilisée pour évaluer x^n. dans notre exemple on suppose que x^n est évalué également par la méthode naive donc nécéssitant $n-1$ opérations de

multiplications.

$$\left|\begin{array}{ll} Calculer\ x & \\ Calculer\ x^2 \quad (1\ mutliplication) & \\ ... & (1+2+...+n-1)=\frac{1}{2}\left(n^2-n\right)\ mutliplications. \\ Calculer\ x^n \quad (n-1\ mutliplications) & \end{array}\right.$$

$$\left|\begin{array}{l} Calculer\ a_1x \\ Calculer\ a_2x^2 \\ ..... \\ Calculer\ a_nx^n \end{array}\right. \quad n\ mutliplications.$$

$$Calculer\ a_0+a_1x+....+a_nx^n \qquad n\ additions$$

Donc le coût de l'opération est de $\frac{1}{2}\left(n^2-n\right)+(2n-1)+n=\frac{1}{2}n^2+\frac{5}{2}n-1$ opérations donc de l'ordre $\mathcal{O}\left(n^2\right)$.

2. **Algorithme de Horner**

L'algorithme de Horner a pour but d'éviter l'évaluation des puissances successives de x, l'idée consiste en une évaluation répétée d'un polyônme de degré 1, ainsi un polynôme P de degré n sera évalué selon la technique suivante :

$$\left|\begin{array}{c} P(x)=a_nx^n+a_{n-1}x^{n-1}....+a_3x^3+a_2x^2+a_1x+a_0 \\ =\left(a_nx^{n-1}+a_{n-1}x^{n-2}....+a_3x^2+a_2x+a_1\right)x+a_0 \\ ..... \\ =\left(\left(\left(\underbrace{\left(\underbrace{a_nx+a_{n-1}}_{P1}\right)x+a_{n-2}....+a_3}_{P2}\right)x+a_2\right)x+a_1\right)x+a_0 \end{array}\right.$$

Cette formule peut être noté par réccurence par :

$$\begin{cases} P_0=a_n \\ P_i=P_{i-1}x+a_{n-i} \qquad 1\leq i\leq n \end{cases}$$

On peut vérifier par récurrence que la formule calcule bien le polynôme défini plus haut

Le Coût de la méthode de Horner peut être évalué comme suit :

$$\left|\begin{array}{ll} P_1 = P_0 x + a_{n-1} & 1\ addition + 1\ multiplication \\ P_2 = P_1 x + a_{n-2} & 1\ addition + 1\ multiplication \\ P_3 = P_2 x + a_{n-3} & 1\ addition + 1\ multiplication \\ \ldots\ldots\ldots\ldots\ldots & 1\ addition + 1\ multiplication \\ P_n = P_{n-1} x + a_0 & 1\ addition + 1\ multiplication \end{array}\right.$$

Donc le coût en calcul de la méthode de Horner est de n multiplications + n addition ou $2n$ opérations. L'ordre est par conséquent $\mathcal{O}(n)$.

Exemple 10

```
restart:
Horner := proc(l, x)
 option remember:
 if nops(l) = 0 then 0
 elif nops(l) = 1 then l[1]
 else x·Horner([seq(l[i], i = 2..nops(l))], x) + l[1]
 fi: end:
PolyNaive := proc(l, x)
 local P, i;
 P := 0:
for i to nops(l) do
 P := P + l[i]·naive(x, i − 1);
 od: P: end:
st := time( ):
 Horner([seq(1, i = 1..10^4)], 9):
 time( ) − st;
                                                    0.

st := time( ):
 PolyNaive([seq(1, i = 1..10^4)], 9):
 time( ) − st;
                                                  9.485
```

1.2.3 Calcul matriciel

Calcul du déterminant

Le détérminant d'une matrice de dimension $(n \times n)$ peut se calculer de la façon suivante :

$$\begin{vmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nn} \end{vmatrix} = a_{11}M_{11} - a_{21}M_{21} + ... (-1)^{n+1} a_{n1}M_{n1}$$

Où M_{i1} sont appelés les mineurs de la matrice A. C'est les détérminants obtenus en suprimant la ligne i et la colonne 1 de la matrice A.

Pour l'évaluation du déterminant on doit calculer

$$n \textit{ déterminants de dimension } ((n-1) \times (n-1)) + n \textit{ multiplication} + (n-1) \textit{ additions}$$

Pour évaluer un déterminant de dimension $(n-1) \times (n-1)$ on doit calculer

$(n-1)$ *déterminants de dimension* $((n-2) \times (n-2))$+$(n-1)$ *multiplication*+$(n-2)$ *addition*

Notons $Cout_n$: le coût d'évaluation d'un déterminant de dimension n donc on a :

$$Cout_n = nCout_{n-1} + n \textit{ multiplication} + (n-1) \textit{addition}$$

$Cout_n = n\left[(n-1)\, Cout_{n-2} + (n-1) \textit{ multiplication} + (n-2) \textit{ addition}\right]$+$n$ *multiplication*+$(n-1)$ *addi*

On peut déduire que le coût d'évaluation du déterminant est supérieur à n !.

Algorithme de Strassen 1969

Soient A, B deux matrices carrés de dimension $(2^n \times 2^n)$ l'algorithme de Strassen permet de calculer à un cout plus faible le produit des matrices par rapport à la compexité en $\mathcal{O}\left(n^3\right)$ de la méthode directe.

On décompose chaque matrice en 4 blocs identiques de dimension $\left(2^{n-1} \times 2^{n-1}\right)$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

On calcule alors

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_2 &= (A_{21} + A_{22}) B_{11} \\ M_3 &= A_{11}(B_{12} - B_{22}) \\ M_4 &= A_{22}(B_{21} - B_{11}) \\ M_5 &= (A_{11} + A_{12}) B_{22} \\ M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

Et on a :

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7 \\ C_{12} &= M_3 + M_5 \\ C_{21} &= M_2 + M_4 \\ C_{22} &= M_1 - M_2 + M_3 + M_6 \end{aligned}$$

Remarque 11 *Si la matrice n'est pas de dimension 2^n on complète avec le nombre nécéssaire de 0 pour atteindre cette dimension.*

Calcul de la complexité de la méthode de Strassen

Soit $n = 2^m$ et notons CoutStrassen le cout du produit par la méthode de Strassen et notons CoutAdd le cout de l'addition de deux matrices.

On a alors

$$\begin{aligned}
CoutStrassen\,(n) &= 7 \times CoutStrassen\left(\frac{n}{2}\right) + 18 CoutAdd\left(\frac{n}{2}\right) = 7 \times CoutStrassen\left(\frac{n}{2}\right) + \frac{18}{4}n^2 \\
&= 7 \times \left(7 CoutStrassen\left(\frac{n}{4}\right) + 18 CoutAdd\left(\frac{n}{4}\right)\right) + \frac{18}{4}n^2 \\
&= 7^2 \times CoutStrassen\left(\frac{n}{4}\right) + 7 \times 18 \times \frac{1}{16}n^2 + \frac{18}{4}n^2 \\
&= 7^2 \times CoutStrassen\left(\frac{n}{4}\right) + \frac{99}{8}n^2 \\
&= .... \\
&= 7^m CoutStrassen\,(1) + \mathcal{O}\left(n^2\right)
\end{aligned}$$

On a $2^m = n \Rightarrow m = \frac{\ln(n)}{\ln(2)}$ d'où

$$7^m = 7^{\frac{\ln(n)}{\ln(2)}} = \exp\left(\ln\left(7^{\frac{\ln(n)}{\ln(2)}}\right)\right) = \exp\left(\frac{\ln(n)}{\ln(2)}\ln(7)\right) = \exp\left(\ln(n)\right)^{\frac{\ln(7)}{\ln(2)}} = n^{\frac{\ln(7)}{\ln(2)}}$$

La valeur approchée de $\frac{\ln(7)}{\ln(2)}$ est donnée par $2.807\,4$ d'où

$$CoutStrassen\,(n) = \mathcal{O}\left(n^{2.807\,4}\right) + \mathcal{O}\left(n^2\right) = \mathcal{O}\left(n^{2.807\,4}\right).$$