Algorithmique, révision rapide

14 septembre 2011

Table des matières

1 Instructions

Les instructions qui suivent sont, avec la notion de fonction et la récursivité, à la base de l'algorithmique.

1.1 l'affectation

On la note en MAPLE :

```
> X := <expression>;
```

exemple échange des valeurs contenues dans les variables X et Y.
Que penser des deux séquences qui suivent ? Quels sont les contenus des variables après leur exécution si on suppose qu'au départ, X contient a, Y contient b?

```
> X:= Y;
> Y:= X;
```

ou bien

```
> Z:= X;
> X:= Y;
> Y:= Z;
```

1.2 la boucle for

Suites récurrentes

```
> u:= u0;
> for i from 1 to n do  u:= f(u); od;
```

Sommes, produits, listes, sequences...

```
>s:=0;
>for i from 1 to n do s:=s+ a[i] od;

>p:=1;
>for i from 1 to n do p:=p * a[i] od;

>s:= [ ];
>for i from 1 to n do s:= [ op(s),  a[i]] od;

>s:= NULL;
>for i from 1 to n do s:= s, a[i] od;
```

Exercice 1.1
Écrire un programme ou une fonction qui prend comme argument une liste L d'objets MAPLE, un terme a et retourne le nombre d'occurrences de a dans la liste L.
corrigé en 1

Exercice 1.2

1. Construire une matrice de taille n =7 telle que $M_{i,j} = i + j + 1$, en programmant explicitement, en utilisant les fonctions préprogrammées de MAPLE.
2. Construire une matrice triangulaire supérieure générique, en programmant explicitement, en utilisant les fonctions préprogrammées de MAPLE..

corrigé en 2

Exercice 1.3 *calculs de complexité*

1. Calculer le nombre d'appels à la fonction **traiter** pour les exemples 1 et 2 du tableau de la page suivante.
2. Calculer le nombre de multiplications, d'additions effectuées dans le calcul du produit de 2 matrices tel qu'il a été programmé dans ce même tableau.

Exemples de boucles imbriquées

exemple 1 :
Traitement des données rangées dans un tableau matriciel A :

```
for i from 1 to rowdim(A) do
   for j from 1 to coldim(A) do
       traiter(A,i,j);
   od;
od;
```

exemple 2 :
Traitement des seules données *au dessus de la diagonale* dans un tableau matriciel A :

```
 for i from 1 to rowdim(A)-1 do
   for j from i+1 to coldim(A) do
       traiter(A,i,j);
   od;
od;
```

exemple 3 :
Pour calculer le produit $C = AB$, de deux matrices A et B ayant respectivement n lignes et m colonnes, m lignes et p colonnes, on traduit la formule

$$\forall i \in [1,n],\ \forall j \in [1,p],\ C_{i,j} = \sum_{k=1}^{m} a_{i,k} b_{k,j}.$$

```
C:=matrix(n, p):

for i from 1 to n do
   for j from 1 to p do
       C[i,j]:= 0;
       for k from 1 to m do
           C[i,j] := C[i,j] + A[i,k]*B[k,j]
       od;
   od;
od;
```

1.3 la boucle while

En MAPLE, la syntaxe est :

```
> while < cond  >     do
        < instr >
> od;
```

Où *cond* (pour condition) est une expression **booléenne** et *inst* (pour instructions) est une suite d'instructions.
La condition est évaluée au moins une fois, en début de boucle :
– si elle prend la valeur **faux** (ou **false**), les instructions (instr) ne sont pas exécutées, le programme poursuit après la boucle ;
– si elle prend la valeur **vrai** (ou **true**), les instructions sont exécutées et le programme reprend en début de boucle (où la condition est réévaluée) ;
Elle est de la forme $C(x_1, x_2, ..., x_n)$ et dépend du contenu de variables présentes dans le programme ; elle est donc susceptible d'être modifiées par les instructions du **corps** de la boucle.

Théorème 1 *condition d'arrêt dans une boucle while.*
Considérons une suite d'instructions contenant une boucle **while** *de condition* $C(x_1, x_2, ..., x_n)$.
– *A la sortie de la boucle la condition est toujours fausse (la condition d'arrêt est la négation de* $C(x_1, x_2, ..., x_n)$*) ;*
– *Si* $C(x_1, x_2, ..., x_n)$ *est vraie à l'entrée de la boucle, et si les instructions qui suivent ne changent pas sa valeur, alors la boucle ne termine pas (on dit que le programme boucle indéfiniment) ;*

Exercice 1.4
Soit $(S_n)_n$ définie par

$$S_n = \sum_{k=1}^{n} \frac{1}{k}.$$

1. Montrer que $(S_n)_n$ a pour limite $+\infty$.
2. Déterminer un programme (ou une fonction) qui détermine le plus petit entier n, tel que $S_n \geq 5$.
3. Calculer le nombre d'opérations (additions, divisions) des trois programmes suivants :

```
S:=1;
n:=1;
while S < 5 do
   n:=n+1;
   S:=S+1/n;
od;
n;
```

```
S:=1;
 n:=1;
 while S < 5 do
    n:=n+1;
    S:=0;
    for i from 1 to n do
       S:=S+1/i;
    od;
 od;
  n;
```

```
S:=n->sum(1/k,k=1..n);
k:=1;
while S(k) < 5 do
   k:=k+1;
od;
k;
```

Exercice 1.5 *division euclidienne des entiers*
Pour calculer les entiers q et r tels que $a = bq + r$ avec $0 \leq r < b$ vous n'imaginez tout de même pas que votre processeur utilise des multiplications ? Voilà ce qu'il fait : on part de $a = 243$ on retranche $b = 13$ jusqu'à obtenir un nombre $r < b$.

243	230	217	204	191	178	165	152	139	126	113	100	87	74	61	48	35	22	9

Programmez une fonction DE :=proc(a,b) qui retourne q et r lorsque a et b sont des entiers naturels avec b>0.
corrigé en 3

1.4 les instructions conditionnelles

```
if  <relation>  then <suite d'instructions>  fi;

if  <relation>  then   <suite d'instructions>
                else  <suite d'instructions>
fi;

if    <relation> then  <suite d'instructions>
                  elif   <relation>   then   <suite d'instructions>
                  elif   <relation>   then   <suite d'instructions>
                     ...
                  else  <suite d'instructions>
 fi;
```

2 les fonctions et procédures

2.1 Paramètres, variables locales et globales

La syntaxe pour les fonctions de MAPLE est :

```
>  <nom> = proc(arg1,arg2,...)
                # séquence d'arguments ou paramètres formels...
>   local var1,var2,...;
                # déclaration des variables locales
>   ...
>   ...         # corps de la procédure
>   ...
>  end:
```

ou encore :

```
f:= (arg1, arg2, ...) -> <expression>;
```

et c'est la dernière expression évaluée qui est retournée.

2.2 Récursivité

Une fonction récursive est une fonction qui figure dans sa propre définition. Observez lecomportement des fonctions données en exemples à l'aide de l'**option trace**.

• **suite récurrente** $u_{n+1} = f(u_n)$
On examinera la **trace** de l'appel de la fonction récursive **U(f,n,a)** qui retourne le terme u_n de la suite telle que $u_0 = a$ et $u_{n+1} = f(u_n)$.

```
U:=proc(f,n,u0)
   option trace;
   if n=0 then u0
          else f(U(f,n-1,u0));
   fi;
end:
```

• **division euclidienne des entiers :** on remarque que si $a - b = bq + r$, alors $a = b(q + 1) + r$... ce qui donne un moyen de calculer le quotient et le reste de façon récursive :

```
restart;
Div:=proc(a:: integer, b::integer)
local D;
option trace;
if b = 0 then ERROR(‘division par zéro‘)
```

```
              elif a < b then [0, a]
              else  D:=Div(a-b, b);
                    subsop( 1=D[1]+1,D)
fi;
end:
```

On doit avant tout s'assurer qu'un appel avec des paramètres effectifs "termine". Ici, la condition d'arrêt est $a < b$, cette condition sera réalisée dans tous les cas où $b > 0$.

3 Exercices : révisions d'algorithmique

Exercice 3.1 *recherche d'un minimum et tri séquentiel*

1. **Stuctures de données :** Nous proposons de trier des éléments rangés dans un tableau (**array**) : regarder ce que donnent les fonctions **array** et **print** pour visualiser les tableaux.
2. Écrire une procédure
   Echange :=proc(t : :array(integer), i : :integer, j : :integer),
   qui prend un tableau, **t**, et deux entiers **i** et **j** en arguments, modifie le tableau **t** par permutation des termes d'indices **i** et **j**, et retourne **t** ainsi modifié (vérifiez). Peut on faire la même chose sous MAPLE avec une **liste** à la place de **t** ?
3. Ecrire une fonction
   RechercheMinimum := proc(t : :array(integer), d : :integer, n : :integer)
   qui retourne la position du plus petit terme d'indice compris entre d et n (longueur du tableau). Utiliser une boucle **for** (pour passer au crible les éléments du tableau) et un test pour sélectionner un élément supérieur aux précédents.
4. **Tri sélectif.** Ecrire une procédure
   TriSelection := proc(t : :array(integer), n : :integer)
   qui prend en arguments un tableau, sa longueur et trie ce tableau.

Exercice 3.2 *recherche dichotomique, cas discret et continu*

1. **Recherche dans une liste**
   (a) Construire un procédure qui prend comme argument une **liste triée,** un élément n et retourne la place de n s'il appartient à la liste, la séquence NULL sinon.
   (b) Majorer le nombre d'itérations effectuées en fonction de n.
2. **Racine d'une fonction continue**
   (a) Construire un procédure qui prend comme argument une fonction continue, un intervalle $I = [a, b]$ tel que$f(a)f(b) < 0$, un réel ε, et retourne une valeur approchée à ε prés d'une racine de f.
   (b) Evaluer le nombre d'itérations effectuées en fonction de la précision voulue et de la longueur de l'intervalle.

Exercice 3.3 *division euclidienne et itération conditionnelle*

1. **Division euclidienne des entiers**
   (a) Ecrire une procédure itérative qui prend comme arguments deux entiers a et b et retourne le quotient et le reste de la division euclidienne de a par b (supposé non nul) les seules opérations étant la soustraction et l'addition.

(b) La même récursive.

(c) Calculer le nombre d'itérations en fonction des données.

2. Idem avec des polynômes

Exercice 3.4 *Exponentiation rapide, récursivité*
Partant de l'observation :

$$a^n = (a^2)^{Ent(n/2)} * a^{Oou1},$$

construire une fonction récursive qui prend un entier n et une expression a en arguments et retourne a^n. Combien de multiplications pour calculer a^{1024}?
voir corrigé en 4

Exercice 3.5 *programmation de l'algorithme du pivot*

Nous décrivons la méthode de Gauss pour un système de n équations à p inconnues : $Mx = b$, dans lequel :

$$M \in \mathcal{M}_{n,p}(\mathbb{K}),\ x \in \mathbb{K}^p,\ b \in \mathbb{K}^n.$$

Pour résoudre le système linéaire $Mx = b$, ci-dessus par la méthode de Gauss, on commence par "compléter" la matrice M en lui ajoutant la colonne b, on obtient ainsi une novelle matrice $A \in \mathcal{M}_{n,p+1}(\mathbb{K})$. On procède de la façon décrite par l'algorithme ci-dessous pour obtenir une matrice en échelons comportant des 0 ou des 1 sur la diagonale. La discussion et la résolution du système obtenu son alors immédiates.

Rappelons l'algorithme :

Définition 1 *Soit $A \in \mathcal{M}_{n,p}(\mathbb{K})$. On appelle* **coefficient principal** *de la ligne i le premier coefficient non nul de cette ligne. On dit que A est une* **matrice en échelons** *si*

- *les coefficients principaux des lignes 1,2,...,n, sont rangés dans des colonnes d'ordres strictement croissants ;*
- *si une ligne est nulle, il en va de même pour les lignes suivantes.*

Méthode de Gauss, description :

pour chaque indice j **variant de 1 à** p, **faire :**

- calculer l'indice p_j ($j^{ième}$ **pivot**) de la ligne d'indice supérieur ou égal à j tel que
$$|a_{p_j,j}| = \sup_{i \geq j} |a_{i,j}|.$$
  - **si** $|a_{p_j,j}| > 0$ **alors, faire :**
    - $L_{p_j} \leftrightarrow L_j$;
    - $L_j \leftarrow -\dfrac{1}{a_{j,j}} L_j$;
    - **pour chaque indice** i **variant de** $j+1$ **à** p, **faire :**
$$L_i \leftarrow L_i - a_{i,j} L_j;$$
    fin faire
  fin si

fin faire

pour chaque indice j **variant de** p **à 2, faire :**

pour chaque indice i **variant de** $j-1$ **à 1, faire :**

$$L_i \leftarrow L_i - a_{i,j} L_j;$$

fin faire

fin faire

Méthode de Gauss, programmation en MAPLE :

Vous utiliserz librement les fonctions **augment**, **addrow**, **mulrow** et **swaprow** avec lesquelles vous commencerez par vous familiariser.

1. Ecrire une fonction **recherche_pivot_max(M,k)** qui prend en arguments une matrice M, un numéro de colonne k, et retourne une séquence (p, m) dans laquelle p est un entier supérieur ou égal à k et tel que $|M(p, k)|$ soit maximal.
2. Ecrire une fonction **traiter_ col_sous_diag(M,k)** qui prend en arguments une matrice M, un numéro de colonne k, et retourne la matrice déduite de M par transformations élémentaires, dont les termes sous la diagonale dans la colonne k sont nuls, le terme de la diagonale étant égal à 1 (ou peut-être à 0, si M n'est pas inversible).
3. Ecrire une fonction **pivot_1** qui prend comme arguments une matrice M, et retourne une matrice déduite de M par transformations élémentaires, dont les termes sous la diagonale sont nuls.
4. Ecrire une fonction **traiter_col_sur_diag** qui prend comme arguments une matrice M triangulaire supérieure dont les termes diagonaux sont égaux à 1 ou 0, un n° de colonne k, et retourne une matrice triangulaire supérieure déduite de M par transformation élémentaires, dont les termes au dessus la diagonale dans la colonne k sont nuls si $M_{k,k} = 1$ et inchangés sinon.
5. Ecrire **pivot_2** qui prend une matrice de la forme $[T, M']$ (T triangulaire supérieure et inversible) retournée par pivot_1 et la transforme en la matrice $[I[n], M^{-1}]$

Applications :

1. Écrire une fonction retournant l'inverse d'une matrice carrée inversible ;
2. Compléter la partie libre formée des deux vecteurs de $\mathbb{R}^6$,

$$u = [1, 2, 3 - 3, 4, 5], v = [0, 4, 1, 2, 4, -1],$$

en une base de $\mathbb{R}^6$.

4 Corrigés de certains exercices

Corrigé n° 1 indications ou corrigé *1.1*
Nombres d'occurences de a dans une liste L.

On écrit d'abord le code dans la feuille de travail, ce qui permet de vérifier la syntaxe.

```
restart;
L:=[a,b,c,a,d,e,1,a,a,a,a,21,34,d,d]:
L[3];
nops(L);
```

15

```
c:=0;
for i from 1 to nops(L) do
  if  L[i]= a then c:=c+1; fi;
od;
c;
```

6

```
occur:=proc(L,t)
 local c, i;
 c:=0;
 for i from 1 to nops(L) do
    if  L[i]= t then c:=c+1; fi;
 od;
 c;
end:
occur(L,a);
```

6

Corrigé n° 2 indications ou corrigé *1.2*

Trois façons de faire. La première avec deux boucles imbriquées.

```
restart;
  with(linalg):

  M:=matrix(7,7);
  for i from 1 to rowdim(M) do
    for j from 1 to coldim(M)do
        M[i,j]:=i+j+1;
    od;
  od;
  evalm(M);
```

$$\begin{bmatrix} 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

La deuxième avec deux appels à la fonction **seq**. On commence par regarder ce que donne la fonction **seq** avec deux paramètres dont l'un varie ; on imbrique cette instruction dans un deuxième appel en faisant varier l'autre paramètre. On obtient une liste de listes que **matrix** reconnaît comme liste de lignes :

```
restart;
[seq(u+v+1,v=1..7)];
[seq(%,u=1..7)];
```

$$[u+2, u+3, u+4, u+5, u+6, u+7, u+8]$$

$$[[3,4,5,6,7,8,9],[4,5,6,7,8,9,10],[5,6,7,8,9,10,11],[6,7,8,9,10,11,12],$$

$$[7,8,9,10,11,12,13],[8,9,10,11,12,13,14],[9,10,11,12,13,14,15]]$$

```
matrix([seq([seq(u+v+1,v=1..7)],u=1..7)]);
```

$$\begin{bmatrix} 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

C'est bien sûr tout prêt dans MAPLE...

```
f:=(i,j)->i+j+1;
M:=matrix(7,7,f);
```

$$\begin{bmatrix} 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

Question 2.b

Le chargement de **linalg** avec **with(linalg) ;** est nécessaire pour que **rowdim** et **coldim** soient reconnues.

```
with(linalg);
t:=matrix(7,7);
evalm(t);
for i from 2 to rowdim(t) do
  for j from 1 to i-1 do
     t[i,j]:=0;
  od;
od;
evalm(t);
```

$$\begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} & t_{1,4} & t_{1,5} & t_{1,6} & t_{1,7} \\ 0 & t_{2,2} & t_{2,3} & t_{2,4} & t_{2,5} & t_{2,6} & t_{2,7} \\ 0 & 0 & t_{3,3} & t_{3,4} & t_{3,5} & t_{3,6} & t_{3,7} \\ 0 & 0 & 0 & t_{4,4} & t_{4,5} & t_{4,6} & t_{4,7} \\ 0 & 0 & 0 & 0 & t_{5,5} & t_{5,6} & t_{5,7} \\ 0 & 0 & 0 & 0 & 0 & t_{6,6} & t_{6,7} \\ 0 & 0 & 0 & 0 & 0 & 0 & t_{7,7} \end{bmatrix}$$

Corrigé n° 3 Indications ou corrigé *1.5*

Version itérative et boucle **while** (version récursive dans le paragraphe **récursivité**).

```
DE:=proc(a,b)
  local q,r;
  if a < b
    then
      0, a;
    else
      q:=0;
      r:=a;
      while r  = q do
         r:=r-b;
         q:=q+1;
      od;
      q,r;
   fi;
  end:

 DE(534,24);
 DE(21,24);
```

$22, 6$

$0, 21$

Corrigé n° 4 Indications ou corrigé *3.4*

La procédure récursive avec une trace qui met en évidence la suite des appels et les entrées sorties...

```
ExpoRapide:=proc(a,n)
  local p;
  option trace;
  if n= 0 then 1
     elif n = 1 then a
     elif n mod 2 = 0 then
          p:=ExpoRapide(a,n/2);
          p*p;
     else p:=ExpoRapide(a,(n-1)/2);
          p*p*a;
  fi;
  end:
  ExpoRapide(a,5);
```

{–> enter ExpoRapide, args = a, 5
{–> enter ExpoRapide, args = a, 2
{–> enter ExpoRapide, args = a, 1
...
<– exit ExpoRapide (now in ExpoRapide) = a}
<– exit ExpoRapide (now in ExpoRapide) = a^2}
<– exit ExpoRapide (now in ExpoRapide) = a^5}

Index