Université de Limoges
Faculté des Sciences & Techniques
IREM

Jeudi 12 novembre 2009

Stage d'Algorithmique
Et si on comptait . . .

Pour l'aide en ligne : le plus simple est de saisir le mot ou l'expression dans la fenêtre de travail, de le sélectionner avec la souris et de sélectionner dans le menu déroulant en haut de fenêtre la commande help on "..." .

Exercice 1 : Petite mise en route.

On considère le programme Maple suivant :

```
mots22 := proc(a,b)
  local L,j,prof,k,x,S ;
  L[1] := [a,a] ; j :=2 ; prof := 0 ;
  while ((prof<= 1) ) do
    k :=prof ;
    for k from prof to 0 by (-1) do
      L[j] :=L[j-1] ;
      if (L[j][2-k]= a) then L[j][2-k] := b ; else L[j][2-k] := a ; fi ;
      j :=j+1 ;
    end do :
    prof := prof +1 ;
  end do :
  S :=convert(op(L),'list') ;
end proc ;
```

1- Que contient la variable S une fois cet algorithme exécuté ? De quel type est-elle ?
2- Ajouter les commentaires nécessaires à la compréhension de cet algorithme.
3- Déterminer sa spécification (c'est-à-dire préciser les "entrées" et "les sorties") .
4- Saisir ce programme dans une fenêtre de travail Maple et l'exécuter sur un jeu d'entrée.

Exercice 2 : Une généralisation

Instructions Maple suffisantes pour faire cet exercice :

seq, op, nops.

Si on voulait généraliser l'algorithme précédent à des mots à deux signes a et b mais de longueur 3, 4 ou 5 (avec les mêmes contraintes qui sont que les mots sont fabriqués un par un, chacun à partir du précédent en changeant juste un signe), le problème serait compliqué.... Mais ce qui est très simple c'est de construire les mots de longueur n (où n est un entier strictement supérieur à 1) à partir des mots de taille $n-1$.

1- Écrire un algorithme en Maple dont la spécification est la suivante :
Entrée : deux signes distincts.
Sortie : la liste des mots de longueur 1 pouvant être écrits avec ces deux signes. Chaque mot sera représenté par la liste de ses lettres.
Tester cet algorithme dans une fenêtre de travail.

```
mot12 = proc(a,b) local S ; S :=[[a],[b]] end :
```

Pour construire alors les mots de longueur 2 on ajoute devant chaque mot de longueur 1 soit le signe a soit le signe b. On obtient alors la procédure (en conservant les listes de listes comme type de données pour représenter les listes de mots) :

```
mot22 = proc(a,b)
        local S ;
         K := mot12(a,b) ;
         S :=[seq([a,op(K[j])], j = 1..2) ,seq([b,op(K[j])], j = 1..2])] ;
end : .
```

On peut travailler récursivement (pour en savoir plus : wikipédia par exemple).
Le plus célèbre des algorithmes récursifs est la factorielle : $0! = 1,\ \forall n \in \mathbb{N}^*\ ,\ n! = (n-1)!$.
Soit n un entier naturel $n > 1$. Le langage Maple permet la récursivité en écrivant :

```
nom-de-proc := proc(..., n)
local nmoinsun,.... ; nmoinsun := nom-de-proc(..., n-1) ...
endproc ;
```

2- Modifier la procédure précédente (`mot22(a,b)`), de manière à construire les mots de longueur n à partir de ceux de longueur $n-1$ et la programmer dans une fenêtre de travail.

```
recmot := proc(B,Z, n)
  local L, K ;
  if n = 1 then
    L := mot12(B, Z) ;
  else
    K := recmot(B, Z, n - 1) ;
    L := [seq([B, op(K[j])], j = 1 .. nops(K)),
    seq([Z, op(K[j])], j = 1 .. nops(K))] ;
  end if ;
  L ;
end proc ;
```

Exercice 3-1 : Dénombrement séquentiel

Maintenant que l'on a fabriqué tous les mots de longueur n que l'on peut écrire avec deux signes, on s'intéresse au dénombrement de ceux qui contiennent un nombre de fois donné, un des deux signes.
Supposons que l'on veuille écrire un algorithme dont la spécification est la suivante :
Entrée : deux signes distincts ; un entier non nul n
Sortie : La liste des mots considérés et un tableau dont la position i (pour i de 1 à $n+1$), est occupée par le nombre de mots de taille n contenant $i-1$ fois le signe choisi.

Si l'on veut écrire un tel algorithme, sans utiliser la récursivité et de manière à mettre à jour le tableau de sortie au fur et à mesure de la construction des mots, on ne peut pas utiliser ce qui à déjà été fait dans l'exercice 2.
Qu' est-ce qu'un mot ? Par quoi est-il caractérisé ? Dans l'exercice précédent on ne s'attachait qu'à le construire ....
Un mot ne contient que deux signes. Soient a et b ces deux signes. Il est alors caractérisé par la position du signe b dans son écriture. Le mot *abbbab* de longueur 6 peut être entièrement caractérisé par la liste des "positions" du signe b dans son écriture : $\{2, 3, 4, 6\}$.

Construire les mots de deux signes distincts de longueur n n'est alors rien d'autre que de fabriquer toutes les parties de l'ensemble $E = \{1, 2, 3, \ldots, n\}$. Soit l'algorithme suivant :
Entrée : n un entier non nul.
Sortie : L'ensemble des parties de $E = \{1, 2, 3, \ldots, n\}$
Initialisation : $A = \{\}$ et $P = \{A\}$.
$**$ Recherche du plus grand nombre j de E qui n'est pas dans A.
S'il existe : on enlève à A tous les nombres plus grands que j et on insère j ;
on met à jour $P = P \cup \{A\}$;
on retourne à $**$ avec le nouvel A.
S'il n'existe pas : on a fini (celà signifie que A = E).

1- Vérifier "à la main", pour quelques valeurs de n, que cet algorithme fournit bien le résultat annoncé.

2- Ci-après la procédure Maple correspondante. La saisir dans une fenêtre de travail.

```
partie := proc(n)
  local f, A, B, j, R, P ;
  A := {} ;
  B := {seq(i, i = 1 .. n)} ;
  j := n ;
  P := {A} ;
  while j <> -infinity do
    f :=x->is(x>j) ;
    A := remove(f, A) ;
   A := A union {j} ;
    P := P union {A} ;
    R := B minus A ;
    j := max(op(R)) ;
    end do ;
  return(P) ;
end proc ;
```

3- La modifier, pour qu'au fur et à mesure de la construction de l'ensemble des parties de l'ensemble E, elle construise de plus, la liste des mots et le tableau dont la position i (pour i de 1 à $n + 1$), est occupée par le nombre de mots de taille n contenant $i - 1$ fois le signe choisi.

```
motscomptés := proc(a, b, n)
  local C, f, EA, EB, ER, A, B, k, j, P, MOT, LM, v ;
  A := {} ;
  B := {seq(i, i = 1 .. n) ;}
  j := n ;
  P :={A} ;
  LM := [[seq(a, i = 1 .. n)]] ;
  C := [1, seq(0, i = 1 .. n)] ;
  while j <> -infinity do
    f :=x->is(x>j) ;
    A := remove(f, A) ;
    A := A union {j} ;
    MOT := [seq(a, i = 1 .. n)] ;
    for k from 1 to nops(A) do MOT[A[k]] := b
    end do ;
   LM := [op(LM), MOT] ;
    v := nops(A) + 1 ;
    C[v] := C[v] + 1 ;
    P := P union {A} ;
    ER := B minus A ;
    j := max(op(ER)) ;
  end do ;
  return(P, LM, C) ;
end proc ;
```

Exercice 3-2 : Dénombrement récursif

Instructions Maple suffisantes pour faire cet exercice :

numboccur, convert, seq, nops, return.

Maintenant que l'on a fabriqué tous les mots de longueur n que l'on peut écrire avec deux signes, on s'intéresse au dénombrement de ceux qui contiennent un nombre de fois donné, un des deux signes.

1. Supposons que l'on veuille écrire un algorithme dont la spécification est la suivante :
   Entrée : deux signes distincts ; n un entier non nul
   Sortie : La liste des mots et un tableau dont la position i (pour i de 1 à $n+1$), est occupée par le nombre de mots de taille n contenant $i-1$ fois le signe choisi.

   Écrire un tel algorithme, en utilisant l'algorithme de l'exercice 2 ou 3 de manière à mettre à jour le tableau de sortie une fois les mots construits. Le tester dans une fenêtre de travail avec un jeu de données.

```
comptage2 := proc(a, b, n)
  local l, B, T, k, S;
  B := recmot(a, b, n);
  T := [seq(0, i = 1 .. n + 1)];
  for k from 1 to nops(B) do l := numboccur(B[k], b);
    T[l + 1] := T[l + 1] + 1;
  end do;
  S := convert(T, list);
  return( B, S);
end proc;
```

2. Et si on utilisait la récursivité :
   On peut bien sûr utiliser la récursivité dans la même optique : un mot de longueur n contenant k fois le signe choisi peut provenir de l'ajoût de l'autre signe dans un mot de longueur $n-1$ contenant déjà k fois le signe choisi, ou de l'ajoût de ce signe dans un mot de longueur $n-1$ contenant déjà $k-1$ fois ce signe (triangle de Pascal). Et c'est tout.

   Construire une procédure récursive Maple, construisant (sans construire les mots) le tableau dont la position i (pour i de 1 à $n+1$), est occupée par le nombre de mots de taille n contenant $i-1$ fois le signe choisi.

```
compte1 := proc(a, b) local C; C := [1, 1] end proc;
reccompte := proc(a, b, n)
  local L, K;
  if n = 1 then
    L := compte1(a, b)
  else
    K := reccompte(a, b, n - 1);
    L := [1, seq(K[k] + K[k + 1], k = 1 .. n - 1), 1];
  end if;
  L;
end proc;
```