HEC 922, 2001/2002 **TP 3**

9, 10 et 16 octobre 2001

Récursif VS Itératif

Laure Danthony
`http://www.ens-lyon.fr/~ldanthon/`

Généralités, Objectifs

L'objectif de ce TP est de comparer l'usage de la récursion et celui de l'itération et de voir sur des exemples les avantages en terme de coût qu'offre l'itération. Nous verrons toutefois un exemple de programme qu'il serait difficile de dérécursiver.
Comme d'habitude, les programmes utilisés (programmes et tests) sont fournis dans les fichiers `prog_x.pas`,`recursio.pas` et `hanoi.pas`. Les trois parties de ce TP sont totalement indépendantes mais de difficulté croissante. On conseille donc de les traiter dans l'ordre.

A vous !

1 Tout d'abord, un programme mystérieux

[HEC éco 1999] On rappelle que la fonction `random(i:integer)` renvoie un entier aléatoire compris entre 0 et $i-1$. L'appel à `RANDOM` permet d'initaliser cette fonction.

On donne le programme suivant :

```
var
   T        : array[1..20001] of integer;
   U,S,i,n  : integer;
   coincide : boolean;

procedure X;
begin
   randomize ; {initialisation de la fonction random}
   for i:=1 to 20001 do T[i]:=1+random(20000)
end;

begin
   X;
   i:=1;coincide:=false;
   repeat
      i:=i+1;
      S:=0;
      while (S<i-1) AND NOT coincide do
      begin
```

```
        S:=S+1;
        if T[S]=T[i] then coincide:=TRUE
      end;
   until coincide=true;
   U:=i;
   for n:=1 to i do write(T[n],' ; ');
   writeln;
   writeln('U = ',U);
   writeln('S = ',S);
   readln;
end.
```

▶ Exo :

1. A la fin du programme, que contiennent les variables S et U ?
2. Vérifiez votre hypothèse en rajoutant si nécessaire des lignes de code au fichier `prog_x.pas`.

2 Suite de Fibonacci

La suite de Fibonacci, bien connue des informaticiens, est définie de la façon suivante :

$$F_0 = F_1 = 1 \text{ et } \forall n > 0, F_{n+2} = F_{n+1} + F_n$$

▶Exo :

1. Ecrire une **fonction** récursive `fibo_rec(n:integer):longint` qui calcule le n-ième terme de la suite de Fibonacci.
2. Ecrire une **fonction** itérative `fibo_iter(n:integer):longint` qui réalise la même chose mais sans calculer deux fois le même terme. On pourra par exemple stocker dans un tableau tous les termes déjà calculés.
3. Vérifier la supériorité de la version itérative !

3 Suite de Hamming

Cette suite est constituée des entiers naturels de la forme $2^a 3^b 5^c$, $(a, b, c \in \mathbb{N})$, rangés par ordre croissant. Ses premiers termes sont donc :

$$1\ 2\ 3\ 4\ 5\ 6\ 8\ 9\ 10\ 12\ 15\ 16\ 18\ldots$$

L'objectif de ce paragraphe est de faire imprimer par PASCAL les N premiers termes de cette suite.

▶Exo :

1. Ecrire une fonction `suivant(x:integer):integer` qui étant donné l'entier x renvoie le premier terme de Hamming STRICTEMENT supérieur à x.
   Indication : On peut commencer par remarquer qu'un terme de la suite est nécessairement égal au double, au triple ou au quintuple d'un terme précédent de la suite. On montre ensuite (faire un dessin !) l'expression :

$$suivant(n) = \text{Min}(2 * suivant(\lfloor \frac{n}{2} \rfloor), 3 * suivant(\lfloor \frac{n}{3} \rfloor), 5 * suivant(\lfloor \frac{n}{5} \rfloor))$$

qui permet de calculer les termes `suivant` *par récurrence...*

2. A l'aide de la fonction précédente, écrire une procédure récursive `Hamming_rec(n:integer)` qui imprime les n premiers termes de la suite.
3. Ecrire une procédure itérative `Hamming_rec(n:integer)` qui réalise la même chose en stockant au fur et à mesure dans un tableau les termes de la suite.

4 Détente : Les tours de Hanoï

Nous allons voir dans cette section que quelquefois la récursion est indispensable à la résolution d'un problème. Dans cette exemple on voit mal comment on pourrait dérécursiver le programme !

4.1 Le problème

Le joueur dispose de trois piquets (A, B et C) et de n rondelles de taille différente (disons de 1 à n).
Au début du jeu, toutes les rondelles sont placées par ordre croissant de taille sur le piquet de gauche (piquet A). Le but du jeu est de toutes les déplacer **sans jamais poser une rondelle sur une rondelle plus petite** vers le piquet de droite (piquet C) en se servant du piquet du milieu (piquet B).
"Résoudre hanoi(n)", c'est donner les mouvements successifs d'un piquet `origine` à un piquet `destination` qui a pour résultat un déplacement "légal" des rondelles empilées du piquet A vers le piquet C.

4.2 Analyse du problème

Le programmeur qui ne sait pas (de tête) résoudre le problème `hanoi(5)` se dit qu'il va laisser faire le travail à PASCAL, sachant que, vu qu'il n'est pas totalement bête, il sait résoudre `hanoi(1)` (!).
La résolution du problème (par récursivité) est basée sur la remarque suivante : si on sait résoudre `hanoi(n-1)`, il suffit pour résoudre `hanoi(n)` de déplacer les $n-1$ rondelles les plus petites de A vers B (en se servant de C comme piquet intermédiaire vide au départ),puis de déplacer la dernière rondelle de A vers C, et enfin de déplacer les $n-1$ rondelles de B vers C (en se servant du piquet A, bien sûr !)

4.3 Le code

Le code fourni (fichier `hanoi.pas`) découle directement de l'analyse précédente.

▶ Amusez vous ! Remarquez au passage que le nombre de mouvements pour résoudre `hanoi(n)` est $2^n + 1$.