Algorithmes et Programmation Impérative 2

Examen de janvier 2007

durée 2h - documents de cours autorisés - calculatrices non autorisées.

Exercice 1. *Complexité*

Question 1.1. $f(n)$ pour $n = 0, 1, 2, 4$

Solution 1.1. Il faut à chaque fois remplacer n par sa valeur dans le corps de la fonction f et l'exécuter :
- $n = 0$. Dans le corps de f, on passe dans la partie **then** et donc on retourne 1 comme résultat de la fonction.
- Supposons qu'on a calculé $f(1)$ et $f(2)$ et on veut calculer $f(3)$
- On remplace n par 3. On passe dans la partie **else** et on doit exécuter les instructions :

```
s := 0 ;
for i := 0 to 2 do s := s + f(i)*f(2-i);
```

ce qui donne $s = f(0)f(2) + f(1)f(1) + f(2)f(0) = 2 + 1 + 2 = 5$ et c'est le résultat retourné par f.

n	0	1	2	3	4
$f(n)$	1	1	2	5	14

Question 1.2. Relation entre $f(n+1)$ et $f(k), k \leqslant n$.

Solution 1.2.

$$f(n+1) = \sum_{i=0}^{n} f(i)f(n-i)$$

Question 1.3.

Q 1.3–1. On note $c(n)$ le nombre de multiplications pour calculer $f(n)$.

Solution 1.3. Quand $n = 0$, on retourne la valeur 1 sans aucun calcul donc $c(0) = 0$.

Pour $n \geqslant 1$, l'évaluation de $f(n)$ entraîne l'exécution de la boucle Pour dans laquelle dans chaque itération, on fait une multiplication, plus le nombre de multiplications nécessaires pour calculer $f(i)$ plus le nombre de multiplications nécessaires pour calculer $f(n-1-i)$. Donc, à chaque itération, on fait $1 + c(i) + c(n-1-i)$ multiplications.

La boucle Pour s'exécute n fois (pour i allant de 0 à $n-1$) et donc le nombre total de multiplications sera :

$$\begin{aligned} c(n) &= \sum_{i=0}^{n-1}(1 + c(i) + c(n-1-i)) \\ &= n + \sum_{i=0}^{n-1} c(i) + \sum_{i=0}^{n-1} c(n-1-i) \\ &= n + 2\sum_{i=0}^{n-1} c(i) \end{aligned}$$

Q 1.3–2. Expression de $c(n+1)$ en fonction de $c(n)$:

Solution 1.3. En utilisant le résultat de la question précédente :

$$\begin{aligned} c(n+1) &= n + 1 + 2\sum_{i=0}^{n} c(i) \\ &= \underbrace{n + 2\sum_{i=0}^{n-1} c(i)}_{c(n)} + 2c(n) + 1 \\ &= 3c(n) + 1 \end{aligned}$$

Q 1.3–3. Montrer par récurrence que $c(n) = \frac{3^n - 1}{2}$.

Solution 1.3.

1. Pour $n = 0$, $c(0) = 0 = \frac{3^0-1}{2}$.
2. En supposant que $c(n) = \frac{3^n-1}{2}$, et en utilisant le résultat de la question précédente, calculons $c(n+1)$:

$$\begin{aligned} c(n+1) &= 3c(n)+1 \\ &= 3 \cdot \frac{3^n-1}{2} + 1 \\ &= \frac{3^{n+1}-3}{2} + \frac{3}{2} \\ &= \frac{3^{n+1}-1}{2} \end{aligned}$$

Question 1.4. Algorithme quadratique.

Solution 1.4. Pour éviter de recalculer les $f(i)$, on les stocke dans un tableau en mettant $f(i)$ dans la case t[i] du tableau t.

```
function f2(n: CARDINAL): CARDINAL ;
const MAX = 10 ;
var
   i,j: INTEGER ;
   t : array[0..MAX] of CARDINAL ;
begin
   t[0] := 1 ; // f(0)
   for i := 1 to n do begin
      t[i] := 0 ;
      for j := 0 to i do t[i] := t[i]+t[j]*t[i-1-j] ;
   end; // for
   f2 := t[n] ;
end; // f2
```

Exercice 2. *Purger une pile*

Question 2.1. Structure de données ?

Solution 2.1. On utilise une pile auxiliaire dans laquelle on met tous les éléments de la pile initiale différents de e. Ce choix est justifié par
- l'aspect dynamique de la structure ;
- et la préservation de l'ordre des éléments.

Question 2.2. La procédure purger

Solution 2.2.

```
procedure purger(var p: PILE; e: ELEMENT) ;
var
   paux: PILE ;
begin
   paux := PILEVIDE ;
   while not(estPileVide(p)) do begin
      if sommet(p)=e then empiler(sommet(p),paux) ;
      depiler(p)
   end; // while
   while not(estPileVide(paux)) do begin
      empiler(sommet(paux),p) ;
      depiler(paux)
   end; // while
end ; // purger
```

Exercice 3. *Permutation circulaire*

Question 3.1. La fonction permute :

Solution 3.1. On donne la fonction copie (que l'on peut supposer connue) puis deux versions de la fonction permute. Dans la première version, on fait trois fois le parcours de la liste : une fois pour faire la copie, une fois pour le calcul de dernier et une fois pour pointer sur l'avant dernier (**while**)

```
function copie(const l: LISTE): LISTE ;
begin
   if estListeVide(l) then copie := LISTEVIDE
   else copie := ajouteEnTete(tete(l),copie(reste(l)))
```

```
end; // copie

function permute(l: LISTE) : LISTE ;
var
   l1,l2,l3 : LISTE ;
begin
   l1 := copie(l) ;
   if estListeVide(l1) or estListeVide(Reste(l1)) then
      permute := l1
   else begin
      l3 := dernier(l1) ;
      modifierReste(l3,l1) ;
      l2 := Reste(l1) ;
      while reste(l2) <> l3 do l2 := reste(l2) ;
      modifierReste(l2,LISTEVIDE) ;
      permute := l3
   end; // if
end; // permute

function permutebis(l: LISTE) : LISTE ;
var
   l1,l2 : LISTE ;
begin
   if estListeVide(l) or estListeVide(Reste(l)) then
      permutebis := l
   else begin
      l1 := LISTEVIDE ;
      l2 := l ;
      while not estListeVide(reste(l2)) do begin
         ajouterEnFin(tete(l2),l1) ;
         l2 := reste(l2) ;
      end; // while
      l1 := ajouteEnTete(tete(l2),l1) ;
      permutebis := l1
   end; // if
end; // permutebis
```

Question 3.2. La procédure permuter

Solution 3.2.

```
procedure permuter(var l: LISTE) ;
var
   l1: LISTE ;
begin
   if not(estListeVide(l) or estListeVide(Reste(l))) then begin
      l1 := l ;
      while not estListeVide(Reste(Reste(l1))) do l1 := Reste(l1) ;
      modifierReste(Reste(l1),l) ;
      l := Reste(l1) ;
      modifierReste(l1, LISTEVIDE) ;
   end; // if
end; // permuter
```

Exercice 4. *Construction dichotomique d'arbres*

Question 4.1.

Solution 4.1.

```
function tableauEnArbre(t: TABLEAU; i,j: T_INDICE_ETENDU): T_ARBRE ;
var
   m: T_INDICE_ETENDU ;
begin
   if i > j then tableauEnArbre := ARBREVIDE
   else begin
      m := milieu(i,j) ;
```

```
        tableauEnArbre := creerArbre(t[m],
                                     tableauEnArbre(t,i,m-1),
                                     tableauEnArbre(y,m+1,j));
    end ; // if
end; // tableauEnArbre
```

Question 4.2. Arbres équilibrés en hauteur ?

Solution 4.2. Oui. On démontre par récurrence sur la taille $n \geqslant 1$ du tableau que l'arbre obtenu est équilibré en hauteur et que cette hauteur vaut

$$h(n) \quad = \quad \lfloor \log_2(n) \rfloor \tag{1}$$

1. Pour $n = 1$, l'arbre est de taille 1, il est évidemment équilibré en hauteur et sa hauteur est $h(1) = 0 = \lfloor \log_2(1) \rfloor$.
2. Soit $n \geqslant 1$ et supposons que pour tout tableau de taille $\leqslant n$, les arbres obtenus sont équilibrés en hauteur et que leur hauteur est donnée par l'équation (1).
3. Considérons l'arbre obtenu pour un tableau de taille $n+1$. D'après l'hypothèse de récurrence, les deux sous-arbres obtenus par appels récursifs sont équilibrés en hauteur et leur hauteur sont respectivement égales à

$$\begin{aligned} h_g &= \left\lfloor \log_2\left(\left\lfloor \frac{n+1}{2} \right\rfloor \right) \right\rfloor \\ h_q &= \left\lfloor \log_2\left(\left\lceil \frac{n+1}{2} \right\rceil \right) \right\rfloor \end{aligned}$$

   l'égalité à 1 n'ayant lieu que lorsque $n+1$ est une puissance de 2.

   Par conséquent, l'arbre obtenu par la construction dichotomique pour un tableau de taille $n+1$ est équilibré en hauteur.

NB : les notations $\lfloor x \rfloor$ et $\lceil x \rceil$ désignent les deux entiers encadrant le nombre réel x, ces deux entiers coincidant avec x lorsque celui-ci est un entier.

Question 4.3. Arbres équilibrés en taille ?

Solution 4.3. Oui. Par récurrence sur la taille du tableau.

Exercice 5. *Parking*

Question 5.1. La propriété qui peut caractériser une voiture ?

Solution 5.1.
- Ça ne peut pas être la couleur car beaucoup de voitures peuvent avoir la même couleur
- Ça ne peut pas être la marque car beaucoup de voitures peuvent être de la même marque
- L'immatriculation est unique et donc elle peut servir à caractériser une voiture

Question 5.2. Calcul du numéro de place :

Solution 5.2. À partir de l'immatriculation, on utilise la méthode du modulo
- transformer l'immatriculation en nombre entier ;
- réduire ce nombre modulo M (nombre de places)

Question 5.3. Résolution des collisions :

Solution 5.3. On ne peut faire qu'un hachage fermé (nombre de places fixe dans le parking). On doit donc conseiller à l'automobiliste qui trouve sa place occupée, d'aller occuper la première place libre qui la suit.