Nom :
Prénom :
Groupe:

Université de Nice-Sophia Antipolis
École Supérieure en Sciences Informatiques

2004–2005

Contrôle de Mathématiques Discrètes
du 24 Janvier 2005

Durée : 2 heures

Corrigé

1 Définition inductive d'un ensemble de mots [4 points]

On considère l'ensemble M des mots sur l'alphabet $\{a,b,c\}$ tels que toutes les suites (maximales) de c consécutifs soient de longueur paire. Ainsi, le mot *abccccbaccaa* est un mot de M , mais le mot *aaccbbcccba* n'est pas un mot de M. On considère l'ensemble E défini inductivement par :

- $\epsilon \in E$
- Si $m \in E$, alors $am \in E$
- Si $m \in E$, alors $bm \in E$
- Si $m \in E$, alors $ccm \in E$

1. A-t-on M inclus dans E? On peut le montrer par récurrence, sur la longueur des mots.
   Le mot vide qui est dans M, est bien dans E.
   Si tous les mots de M de longueur strictement inférieure à n sont dans E (hypothèse de récurrence), alors pour tout mot m de M de longueur n, on a soit :
   - $m = am'$ ou $m = bm'$, avec m' qui appartient nécessairement à M
   - $m = cm''$: auquel cas nécessairement (m ne peut pas commencer par un seul c puisque toute suite de c est de longueure paire), $m = ccm'$ avec $m' \in M$ (car $m \in M$).

   Dans ces 2 cas, l'hypothèse de récurrence implique que m' appartient à E et donc m appartient à E.
   Remarque : on peut aussi dire que M est l'ensemble des mots écrits sur l'alphabet $\{a,b,cc\}$, et qu'il est alors évident que $E = M$.
2. A-t-on E inclus dans M?
   Il est immédiat de constater que le seul élément de la base de E est dans M et que chacun des trois constructeurs appliqué à un mot de M donne un mot de M et donc que E inclus dans M.
3. Le schéma définissant E est-il libre? Étant donné un mot m de E, si c'est le mot vide, il est dans la base et ne peut être obtenu par aucun constructeur, sinon la lettre a,b ou c (et donc cc) par laquelle il commence détermine le constructeur qui a permis d'obtenir m et l'unique mot auquel ce constructeur est appliqué pour obtenir m. Donc le schéma est libre.
   On peut aussi dire qu'il est évident que tout mot de E peut être construit de manière unique par applications successives des trois constructeurs, c'est-à-dire que le shéma est libre.
4. Trouver et résoudre une relation de récurrence permettant de compter le nombre de mots de longueur n appartenant à E. En notant E_n l'ensemble des mots de longueur n de E, et $card(E_n)$ le nombre des éléments de E_n, on a :
   $\forall n > 1 : E_n = aE_{n-1} \cup bE_{n-1} \cup ccE_{n-2}$.

Comme les trois ensembles aE_{n-1}, bE_{n-1} et ccE_{n-2} sont 2 à 2 disjoints, l'égalité précédente sur les ensembles donne une égalité sur le nombre d'éléments de ces ensembles :
$\forall n > 1 : card(E_n) = card(aE_{n-1}) + card(bE_{n-1}) + card(ccE_{n-2})$.
D'où : $\forall n > 1 : card(E_n) = 2card(E_{n-1}) + card(E_{n-2})$.
Avec $card(E_0) = 1$ et $card(E_1) = 2$.
D'où : $\forall n > 1 : card(E_n) = \frac{2+\sqrt{2}}{4}(1+\sqrt{2})^n + \frac{2-\sqrt{2}}{4}(1-\sqrt{2})^n$.

2 Tours de Hanoï, avec trois piquets dont un piquet central [4 points]

On considère une variante des tours de Hanoï avec trois piquets A,B,C et n disques.

- les n disques sont de tailles différentes
- initialement les n disques sont rangés sur un des piquets X, du plus grand en bas, au plus petit en haut
- on désire amener les n disques dans le même ordre sur un autre piquet Y
- on ne peut déplacer qu'un seul disque à la fois
- on ne peut pas poser un disque sur un disque de taille inférieure
- tous les déplacements de disque doivent se faire via le piquet central (un déplacement direct entre les deux piquets extérieurs est interdit)

On rappelle que si X et Y sont les piquets extérieurs, le nombre minimum de déplacements pour transporter n disques entre X et Y est de $3^n - 1$.

On suppose que moveExterieurToCentral(n,X) (que vous pourrez noter mE2C(n,X) pour abréger) déplace de manière optimale une tour de hauteur n d'un piquet extérieur X vers le piquet central. De même, moveCentralToExterieur(n,X) (que vous pourrez noter mC2E(n,X) pour abréger) déplace de manière optimale une tour de hauteur n du piquet central vers un piquet extérieur X.

1. Déterminer, prouver et résoudre les équations de récurrence permettant de calculer la complexité de ces deux méthodes. Pour déplacer n disques du piquet extérieur X vers le piquet central, il faut necessairement déplacer le plus grand disque de X vers le piquet central, avant cela il faudra d'abord déplacer les $n-1$ plus petits disques sur le piquet extérieur Y, après le déplacement du grand disque, il faudra enfin déplacer les $n-1$ plus petits disques du piquet extérieur Y sur le piquet central.
   Donc l'algorithme suivant est optimal :
   ```
   mE2C(n, X){
     if (n>0){
           mE2E(n-1, X, Y);
           moveDisque(X, central);
           mE2C(n-1, Y);
     }
   }
   ```
   Donc : $ComplexiteE2C(n) = 3^{n-1} - 1 + 1 + ComplexiteE2C(n-1)$
   D'où : $ComplexiteE2C(n) = \frac{3^n-1}{2}$
   Pour déplacer n disques du central vers le piquet extérieur X, il faut d'abord déplacer les $n-1$ plus petits disques sur le piquet extérieur Y, puis déplacer le grand disque vers le piquet extérieur X et enfin déplacer les $n-1$ plus petits disques du piquet extérieur Y vers le piquet extérieur X.
   Donc l'algorithme suivant est optimal :
   ```
   mC2E(n, X){
     if (n>0){
           mC2E(n-1, Y);
   ```

```
        moveDisque(central, X);
        mE2E(n-1, X, Y);
  }
}
```
Ce qui donne pour la complexité de mC2E, la même équation de récurrence et donc la même complexité que mE2C.

2. En déduire que l'algorithme suivant est optimal
```
moveExterieurToExterieur(n, initial, final){
  if (n>0){
        mE2C(n-1, initial);
        mC2E(n-1, final);
        moveDisque(initial, central);
        mE2C(n-1, final);
        mC2E(n-1, initial);
        moveDisque(central, final);
        mE2C(n-1, initial);
        mC2E(n-1, final);
        }
}
```

La complexité de l'algorithme précédent est donnée par :

$$C(n) = \frac{3^{n-1}-1}{2} + \frac{3^{n-1}-1}{2} + 1 + \frac{3^{n-1}-1}{2} + \frac{3^{n-1}-1}{2} + 1 + \frac{3^{n-1}-1}{2} + \frac{3^{n-1}-1}{2}$$

Soit : $C(n) = 3^n - 1$, qui est la complexité optimale pour déplacer n disques d'un piquet extérieur à l'autre.
Il est à noter que ce mode de déplacement avec passage obligé par le piquet central, est déterministe en ce sens qu'à toute étape il y a une unique possibilité de déplacer un disque de ou vers le piquet central et qui n'est pas l'inverse du déplacement précédent. On peut donc décrire de manière très simple l'algorithme optimal pour déplacer n disques d'un piquet quelconque à un autre : faire le seul déplacement autorisé (c'est-à-dire qui respecte la contrainte de taille de deux disques superposés) qui n'est pas l'inverse du déplacement précédent ; et au départ, faire le seul déplacement autorisé.

3 Tours de Hanoï, avec quatre piquets [4 points]

On suppose connue la méthode moveHanoi(n,initial, final, intermediaire) qui déplace de manière optimale (c'est à dire avec $2^n - 1$ mouvements de disques) une tour de Hanoi de hauteur n, d'un piquet initial vers un piquet final en utilisant un seul piquet intermédiare avec comme contraintes :

- les n disques sont de tailles différentes
- initialement les n disques sont rangés sur un piquet initial, du plus grand en bas, au plus petit en haut
- on désire amener les n disques dans le même ordre sur un piquet final
- on ne peut déplacer qu'un seul disque à la fois
- on ne peut pas poser un disque sur un disque de taille inférieure.

On suppose que l'on dispose d'un quatrième piquet. Pour chacune des stratégies suivantes, évaluer la compléxité en nombre de déplacements de disques, et déterminer la meilleure de ces stratégies.

- Stratégie 1 : Déplacer les n-2 plus petits disques vers l'un des piquets intermediaires, puis déplacer les 2 grands disques restant du piquet initial vers le piquet final (en utilisant le

piquet intermediaire resté libre) , enfin déplacer les n-2 plus petits disques disques vers le piquet final.
La complexité $CS1$ de cette stratégie est donnée par l'équation de récurrence suivante :

$$CS1(n) = CS1(n-2) + 3 + CS1(n-2) = 2CS1(n-2) + 3$$

Ce qui fait que $CS1(n)$ est en $\Theta(\sqrt{2}^n)$.

– Stratégie 2 : Déplacer les n-3 plus petits disques vers l'un des piquets intermediaires, puis déplacer les 3 grands disques vers le piquet final (en utilisant le piquet intermediaire resté libre) , enfin déplacer les n-3 disques vers le piquet final. La complexité $CS2$ de cette stratégie est donnée par l'équation de récurrence suivante :

$$CS2(n) = CS2(n-3) + 7 + CS2(n-3) = 2CS2(n-3) + 7.$$

C'est une équation de récurrence linéaire non homogène d'ordre 3. Le polynôme caractérisque est $X^3 - 1$. Et donc $CS2(n)$ est en $\Theta(\sqrt[3]{2}^n)$.
Ce qui est meilleur que la complexité de la première stratégie puisque $\sqrt[3]{2} < \sqrt{2}$.

– Stratégie 3 : Déplacer les n/2 plus petits disques vers l'un des piquets intermediaires, puis déplacer les n/2 grands disques vers le piquet final (en utilisant le piquet intermédiaire resté libre), enfin déplacer les n/2 petits disques vers le piquet final. La complexité $CS3$ de cette stratégie est donnée par l'équation de récurrence suivante :

$$CS3(n) = CS3(n/2) + 2^{n/2} - 1 + CS3(n/2) = 2CS3(n/2) + 2^{n/2} - 1.$$

Donc, sans résoudre cette équation, on peut dire $CS3(n)$ est dominée par $2^{n/2} = \sqrt{2}^n$ et donc que la stratégie 3 est au mieux de même ordre que la stratégie 1.

– Meilleure stratégie parmi les trois?
Les résultats précédents montrent que le stratégie 2 est meilleure que les deux autres.

Question facultative : avez-vous une meilleure stratégie ?
On peut reprendre la stratégie 1 ou 2, avec un entier $k > 2$, la complexité sera alors en $\Theta(2^{\frac{n}{k}})$ qui décroit lorsque k croit. Mais ceci ne donne pas la complexité optimale qui n'est d'ailleurs pas connue : http://mathworld.wolfram.com/TowerofHanoi.html.

4 Listes d'entiers [4 points]

On considère des listes d'entiers L_E, définies inductivement par :

– $liste\_vide \in L_E$
– Si $l \in L_E$ et $e \in E$, alors $cons(e,l) \in L_E$

On rappelle que les fonctions $car(l)$ et $cdr(l)$ sont définies sur les listes non vides par :
Si $l = cons(e,l')$, $car(l) = e$ et $cdr(l) = l'$

On dit qu'une liste l est croissante, si elle est vide ou réduite à un seul élément ou sinon si $car(l)$ est inférieur ou égal à tous les éléments de $cdr(l)$, et $cdr(l)$ est elle même une liste croissante. Si de plus il n'y a pas deux éléments consécutifs égaux de la liste l, l est strictement croissante.

Soit LC_E le sous ensemble de L_E constitué des listes croissantes, et LSC_E le sous-ensemble de LC_E constitué des listes strictement croissantes.

Donner une définition *récursive* des fonctions suivantes et déterminer la complexité de chacune de ces fonctions, en supposant que car, cdr et $cons$ $estVide$ sont toutes en temps constant.

1. Croissante : $L_E \rightarrow \{vrai, faux\}$
Cette fonction retourne vrai si et seulement si la liste considérée est croissante

```
boolean Croissante(L_E l){
  if ( estVide(l) ){  return true;}
  if ( estVide(cdr(l)) ){  return true;}
  return ( car(l) <= car(cdr(l)) ) && Croissante(cdr(l);
}
```

Si n est la longueur de la liste, la complexité vérifie $c_n = c_{n-1} + 1$ et est linéraire en la longueur de la liste traitée (c'est-à-dire en $\Theta(l.length())$).

2. Simplifiée : $LC_E \to LSC_E$
   Cette fonction retourne une liste où les répétitions ont été éliminées (donc une liste strictement croissante)
   ```
   LSC_E Simplifiée(LC_E l){
     if ( estVide(l) ){  return liste_vide ;}
     if ( estVide(cdr(l)) ){  return cons(car(l),liste_vide) ;}
     if ( car(l) != car(cdr(l)) ) {return cons(car(l),Simplifiée(cdr(l))) ; }
     return Simplifiée(cdr(l)) ;
   }
   ```
   La complexité est linéraire en la longueur de la liste traitée.
3. Fusion : $LC_E \times LC_E \to LC_E$
   Cette fonction retourne une liste de LC_E qui est la fusion des deux listes triées.
   ```
   LC_E Fusion(LC_E l1, LC_E l2){
     if ( estVide(l1) && estVide(l2)){  return liste_vide ;}
     if ( !estVide(l1)) && estVide(l2) )
                            {return cons(car(l1),Fusion(cdr(l1),l2)) ;}\\ copie de l1
     if ( estVide(l1)) && !estVide(l2) )
                            {return cons(car(l2),Fusion(l1,cdr(l2))) ;}\\ copie de l2
     if ( car(l1) <= car(l2) ) { return cons(car(l1),Fusion(cdr(l1),l2)) ;}
     return cons(car(l2),Fusion(cdr(l2),l1)) ;
   }
   ```
   La complexité de Fusion CF est fonction de la somme n des longueurs des 2 listes traitées et est donnée par l'équation suivante :
   $CF(n) = CF(n-1) +$ Constante et $CF(0) = 1$.
   La complexité est donc linéraire en la somme des longueurs des 2 listes traitées.
4. Intersection : $LSC_E \times LSC_E \to LSC_E$
   Cette fonction retourne une liste de LSC_E contenant les éléments qui sont dans les deux listes.
   ```
   LSC_E Intersection(LSC_E l1, LSC_E l2){
     if ( estVide(l1) || estVide(l2)){  return liste\vide ;}
     if ( car(l1) < car(l2) ) { return Intersection(cdr(l1),l2) ;}
     if ( car(l2) < car(l1) ) { return Intersection(l1,cdr(l2)) ;}
     // car(l1) = car(l2)
     return cons(car(l1),Intersection(cdr(l1),cdr(l2)) ;
   }
   ```
   La complexité de Intersection CI est fonction de la somme des longueurs des 2 listes traitées et vérifie l'équation suivante :
   $CI(n) = CI(n-1) +$ Constante et $CF(0) = 1$.
   La complexité est donc linéraire en la somme des longueurs des 2 listes traitées.

5 Équations de récurrence [4 points]

Donnez l'ordre de grandeur des solutions des équations de récurrence suivantes :

1. $u_n = 6u_{n-1} - 8u_{n-2} + 3^n$ Les valeurs 2 et 4 sont racines du polynôme caractéristique $P(X) = X^2 - 6X + 8$ de l'équation homogène associée. Donc d'après (par exemple) le cours, les solutions (u_n) sont de la forme :
   $a(2^n) + b(4^n) + \frac{3^2}{P(3)}(3^n)$, pour tout réel a et b.
   Donc (u_n) est en $\Theta(4^n)$, sauf si $b = 0$ auquel cas (u_n) est en $\Theta(3^n)$.

2. $v_n = 6v_{n-1} - 8v_{n-2} + 4^n$ D'après le cours les solutions sont en $\Theta(n4^n)$.
3. $w_n = 6w_{n-1} - 8w_{n-2} + 2^n$ Les solutions (w_n) sont de la forme :
   $a(2^n) + b(4^n) + 2(n2^n)$ pour tout réel a et b.
   Donc (w_n) est en $\Theta(4^n)$, sauf si $b = 0$ auquel cas (w_n) est en $\Theta(n2^n)$.
4. $x_n = 6x_{n-1} - 9x_{n-2} + 3^n$
   Ici 3 est racine double du polynôme caractéristique $P(X) = X^2 - 6X + 9 = (X-3)^2$ de l'équation homogène associée. Donc d'après le cours, les solutions (x_n) sont en $\Theta(n^2 \times 3^n)$.