Année 2006-2007 | Session de Janvier 2007

GU : Licence Sciences Technologies (semestre 5) | **UE :** INF209
Épreuve de : Algorithmique Avancée
Date : 11 janvier 2007, 14h | **Durée :** 3h
Documents autorisés
Épreuve de Monsieur Robson

1 [6 points]

Des arbres binaires peuvent être representés par des structures de noeuds, chaque noeud possédant un champ `valeur:entier` et deux champs `gauche` et `droit` qui seront des pointeurs vers les sous-arbres gauche et droit (ou `nul`).
On peut définir un ordre sur les arbres binaires tel que l'arbre vide précède tout autre arbre et, entre deux arbres non vides, l'un précède l'autre si sa valeur est inférieur, ou (dans le cas de deux valeurs égales) son sous-arbre gauche précède celui de l'autre, ou (dans les cas de deux sous-arbres gauches égaux) son sous-arbre droit précède celui de l'autre.
Ecrire en exalgo

i. une fonction entière avec deux paramètres qui seront des pointeurs vers les racines de deux arbres (éventuellement vides) et qui retourne -1 si le premier arbre précède l'autre dans cet ordre, 0 si les deux sont identiques et +1 dans le dernier cas où le deuxième précède le premier.
Réponse

```
fonction precede(val X,Y : arbreptr): entier;
var compg: entier;
debut
si X=nul alors
   si Y=nul alors retourne 0
   sinon retourne -1
   finsi
finsi;
si Y=nul retourne 1 finsi;
si X^.valeur != Y^.valeur alors
   si X^.valeur<Y^.valeur alors
      retourne -1
   sinon retourne 1
   finsi
finsi;
compg:=precede(X^.gauche,Y^.gauche);
si compg=0 alors
   retourne precede(X^.droit,Y^.droit)
sinon retourne compg
finsi
finfonction;
```

ii. une procédure avec un paramètre, un tableaux de pointeurs vers la racine d'un arbre, et qui trie les pointeurs selon l'ordre de leur arbre, en utilisant la fonction de la partie précédente. (On n'exige pas que l'algorithme de tri soit efficace.)
Réponse

```
procedure TRI(ref T:ptrtableau)
```

```
var ptr: arbreptr;
debut
pour i allant de tailletableau a 2 par pas de -1 faire
   pour j allant de 1 a i-1 par pas de 1 faire
      si precede(T[j],T[i])=1 alors
         ptr:=T[i];
         T[i]:=T[j];
         T[j]:=ptr
      finsi
   finpour
finpour
finprocedure
```

2 [5 points]

On veut calculer la probabilité qu'un arbre binaire de recherche aléatoire de 1000 noeuds ait son hauteur strictement inférieure à 50.

i. Donner une récurrence pour cette probabilité dans le cas général d'un arbre de n noeuds et hauteur inférieure à h.
Vous devrez utiliser le fait que le sous-arbre gauche de cet arbre a un nombre g de noeuds avec probabilité $1/n$ pour tout g entre 0 et $n-1$.
Réponse Soit $P_{n,h}$ la probabilité demandée. $P_{0,h} = 1$ pour tout h et $P_{n,0} = 0$ pour tout $n > 0$. Dans les autres cas, l'arbre aura hauteur inférieure à h si ses deux sous-arbres (de taille g et $n-1-g$) ont hauteur inférieure à $h-1$, ce qui donne la récurrence
$P_{n,h} \;=\; somme(g = 0 \ldots n-1) P_{g,h-1} \times P_{n-1-g,h-1}/n$

ii. Donner une procédure pour calculer la probabilité demandée en utilisant la programmation dynamique.
Réponse

```
var PTAB: /* tableau de 2 dimensions */
procedure Proba()
debut
pour h allant de 0 a 50 par pas de 1 faire
   PTAB[0][h]:=1
finpour;
pour n allant de 1 a 1000 par pas de 1 faire
   PTAB[n][0]:=0;
   pour h allant de 1 a 50 par pas de 1 faire
      somme:=0;
      pour g allant de 0 a n-1 par pas de 1 faire
         somme:=somme+PTAB[g][h-1]*PTAB[n-1-g][h-1]
      finpour;
      PTAB[n][h]:=somme/n
   finpour
finpour
finprocedure
/* PTAB[1000][50] sera la probabilité souhaitée */
```

3 [4 points]

i. Donner des affectations d'initialisation et une boucle `tant que` qui calculent le plus grand entier i tel que $i! + (2i)!$ soit inférieure à un entier positif donné m.
Réponse

```
i:=0; facti:=1; fact2i:=1;
tantque fact1+fact2<m faire
```

```
    i:=i+1;
    facti:=facti*i;
    fact2i:=fact2i*(2*i-1)*2*i
fintantque;
/* i-1 est l'entier cherché */
```

ii. Donner un invariant pour votre boucle et expliquer (sans démonstration) comment cet invariant peut être utilisé pour prouver que la boucle soit correcte.
Réponse
$facti = i!$ *et* $fact2i = (2i)!$ *et* $(i-1)! + (2(i-1))! \; < \; m$
(où on interprète $n!$ comme 0 pour $n < 0$)
Les trois faits suivants suffisent pour démontrer que la boucle est (partiellement) correcte :
– L'invariant est une conséquence de la précondition $i = 0$ *et* $facti = 1$ *et* $fact2i = 1$ *et* $m > 0$
– si l'invariant est vrai et $fact1 + fact2 < m$ avant l'exécution de `i:=i+1; facti:=facti*i; fact2i:=fact2i*(2*i-1)*2*i` l'invariant reste vrai après
– La postcondition souhaitée (i-1 est l'entier cherché) est une conséquence de l'invariant et $fact1 + fact2 \geq m$.

4 5 [points]

Déterminer (en fonction de n à $O(\ )$ près) le temps de calcul des boucles (i), (ii), (iii), (iv) et (v) (`P()` est une procédure dont le temps de calcul est constant.)

i. `pour i allant de 1 a n*n par pas de 1 faire P(); finpour;`
Réponse $O(n^2)$

ii.
```
pour i allant de 1 a n par pas de 1 faire
    pour j allant de 1 a i par pas de 1 faire
        pour k allant de j a i par pas de 1 faire P()
        finpour
    finpour;
finpour;
```
Réponse $O(n^3)$

iii.
```
i:=n;
tant que i>1 faire
    si i mod 2 = 0 alors
        i:=i/2
    sinon
        i:=i-1
    finsi
fintantque;
```
Réponse $O(\log n)$

iv.
```
pour i allant de 1 a n par pas de 1 faire
    j:=1;
    tant que i+j<n faire j:=j*2;
    fintantque;
finpour;
```
Réponse $O(n \log n)$

v.
```
i:=1;
tant que i<n*n faire
    i:=i+1+i/n /* arrondi vers le bas */
fintantque
```
Réponse i est incrémenté de 1 à n par pas de 1, ensuite de n à $2n$ par pas de 2, et de $2n$ à $3n$ par pas de 3 etc. Donc le temps est $\approx n + n/2 + n/3 + \cdots + n/n$, c'est-à-dire $n \times (1 + 1/2 + 1/3 + \cdots 1/n)$
$O(n \log n)$