IFT2010 H06 — Examen Intra

Miklós Csűrös

21 février 2006

Aucune documentation n'est permise à l'exception d'une feuille de $8\frac{1}{2}'' \times 11''$. L'examen vaut 100 points. Le problème 6 est optionnel pour 15 points de boni. Il y a d'autre questions où vous pouvez obtenir 6 points de boni en total.

Répondez à toutes les questions dans les cahiers d'examen.

0 Votre nom (1 point)

Écrivez votre nom et code permanent sur tous les cahiers soumis.

1 Oh (12 points)

Remplissez la dernière colonne du tableau suivant. Mettez Θ pour dénoter «$f(n) \in \theta(g(n))$», Ω pour «$f(n) \in \Omega(g(n))$», O pour «$f(n) \in O(g(n))$», o pour «$f(n) \in o(g(n))$» et ω pour «$g(n) \in o(f(n))$». Chaque ligne vaut 2 points : 2 points pour la meilleure réponse ou 0 sinon (donc 0 points pour un O quand on peut mettre Θ). Vous ne devez donner aucune justification à vos réponses ici. (Comme d'habitude, $\lg n = \log_2 n$.)

	$f(n)$	$g(n)$	relation
a	$2n^2 - \lg n$	$(n+2)^2$	
b	$n+1$	$n-1$	
c	2^n	3^n	
d	$0.01\sqrt{n}$	$12(\lg n)^2$	
e	$3n^3$	$8^{\lg n}$	
f	$4^{\sqrt{n}}$	3^n	

2 Pile $\leftrightarrow$ file (15 points)

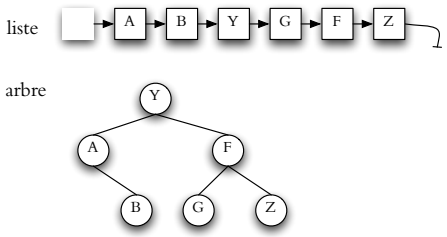
On a montré lors d'un exercice qu'on peut implanter une queue en utilisant deux piles. Donc il ne serait pas surprenant si on pouvait implanter une pile en utilisant deux queues. Mais peut-on implanter une pile avec *une* queue ? (Oui !) Montrez comment le faire. Les opérations permises sur la queue sont : enqueue, dequeue et isEmpty. Vous devez montrer l'implantation de pop, push et isEmpty pour la pile. L'implantation peut utiliser une variable simple additionnelle de n'importe quel type (si nécessaire) et une queue.

Analysez le temps de calcul des opérations quand la pile contient n éléments en supposant que toutes les opérations de la queue sont de $O(1)$.

3 Parcours (7 points)

Donnez un algorithme récursif pour l'énumération des élements dans un arbre binaire de recherche dans l'ordre *décroissant*. Comme d'habitude, vous pouvez utiliser les opérations $\mathsf{left}(x)$, $\mathsf{right}(x)$ et $\mathsf{parent}(x)$ pour trouver l'enfant gauche, enfant droit et le parent du nœud x.

4 Liste $\leftrightarrow$ arbre (35 points)



On veut implanter un TAD liste en se reliant à un arbre binaire balancé comme l'arbre AVL ou l'arbre rouge et noir. Comme d'habitude, la valeur stockée à un nœud x est dénotée par $\mathsf{val}(x)$, mais les nœuds ne sont pas organisés par $\mathsf{val}(x)$ mais plutôt par leur ordre dans la liste. Le nœud x qui correspond au k-ème élément sur la liste est le k-ème dans le parcours infixe. Pour ceci, on maintient une variable $\mathsf{taille}(x)$ associée à chaque nœud x qui est le nombre de nœuds dans le sous-arbre enraciné à x (y incluant x). Dans ce qui suit, vous devez développer les algorithmes pour l'implantation de la liste TAD. Votre code peut utiliser les opérations usuelles de $\mathsf{left}(x)$, $\mathsf{right}(x)$ et $\mathsf{parent}(x)$ pour naviguer dans l'arbre. Un enfant ou parent manquant est dénoté par null. Pour simplifier la code, on pose $\mathsf{taille}(\mathsf{null}) = 0$.

a (5 points) Donnez un algorithme récursif qui calcule taille() pour tous les nœuds dans le sous-arbre de x.

```
CALCULER-TAILLE(x) // x est un nœud
G1  if x est une feuille then taille(x) ← 1 ;
G2  else // ici vient votre code pour le cas où x n'est pas une feuille
G3  return taille(x)
```

b (5 points) Donnez un algorithme qui retourne le k-ème nœud dans le parcours infixe, en utilisant les champs taille. L'idée est d'implanter une fonction NŒUD-K(x, k) qui trouve le k-ème élément dans le parcours infixe du sous-arbre de x. (1 point boni pour un algorithme non-récursif — le patron ici est pour une solution récursive.)

```
NŒUD-K(x, k) // x est un nœud, k est un entier
K1  if k ≤ 0 ou k > taille(x) then erreur !
K2  if k = ☐ then return x
K3  if k < ☐ then return NŒUD-K(left(x), ☐)
K4  if k > ☐ then return NŒUD-K(right(x), ☐)
```

c (20 points) Expliquez comment on peut mettre à jour les valeurs taille après insertion ou deletion d'un nœud. Pour ceci, notez que quel que soit l'arbre qu'on utilise (AVL ou RN) une telle opération est executée en deux phases : insertion/suppression d'un nœud sans maintenir l'équilibre, suivi par des tests et des rotations. Montrez ce qui ce passe dans la première phase (insertion ou deletion dans un arbre binaire de recherche), et comment les tailles des nœuds doivent changer. Ensuite, montrez comment et à quels nœuds taille doit changer lors d'une rotation gauche ou droite. Démontrez que les opérations suivantes de l'ADT liste peuvent être implantées avec un temps d'exécution de $O(\log n)$ (sur n éléments) : Kth(k) [retourne le k-ème élément de la liste] insert(k, x) [insère x en position k de la liste] et delete(k) [supprime le k-ème élément de la liste].

d (5 points) Est-ce qu'il y a des avantages à cette implantation par rapport à d'autres solutions qu'on a vues (tableau et liste chaînée avec pointeurs) ? (Justifiez votre réponse.)

5 Arbre rouge-rouge-noir (30 points)

5.1 Coloriage et rrang (15 points)

Un *arbre rouge-rouge-noir* (RRN) est un arbre binaire de recherce avec des feuilles null comme les arbres rouge et noir, dans lequel les nœuds sont équipés d'un rrang. (Ne pas confondre avec le rang des arbres RN.) Les rrangs des nœuds satisfont les propriétés suivantes.

1. Pour chaque nœud x excepté la racine,

$$\mathsf{rrang}(x) \leq \mathsf{rrang}\big(\mathsf{parent}(x)\big) \leq \mathsf{rrang}(x) + 1.$$

2. Pour chaque nœud x avec un arrière-grand-parent $y = \mathsf{parent}(\mathsf{parent}(\mathsf{parent}(x)))$,

$$\mathsf{rrang}(x) < \mathsf{rrang}(y).$$

3. Pour chaque feuille x on a $\mathsf{rrang}(x) = 0$ et $\mathsf{rrang}\big(\mathsf{parent}(x)\big) = 1$.

On colorie les nœuds soit rouge soit noir : le nœud x est rouge si et seulement s'il n'est pas la racine et $\mathsf{rrang}(x) = \mathsf{rrang}\big(\mathsf{parent}(x)\big)$. Tous les autres nœuds sont noirs.

Complétez l'énoncé du théorème suivant et démontrez qu'il est vrai.

Théorème 1. *Le coloriage d'un arbre rouge-rouge-noir est tel que*

(i) chaque feuille est colorié par ☐.

(ii) si le parent d'un nœud ☐ *est* ☐*, alors son grand-parent (s'il existe) est* ☐.

(iii) chaque chemin reliant un nœud à une feuille dans son sous-arbre contient le même nombre de nœuds ☐.

5.2 La hauteur d'un arbre rouge-rouge-noir (15 points)

On veut démontrer que la hauteur d'un arbre rouge-rouge-noir avec n nœuds internes est $O(\log n)$. Il n'est pas trop difficile de démontrer le théorème suivant.

Théorème 2. *Pour tout nœud x d'un arbe rouge-rouge-noir, le nombre de nœuds internes dans le sous-arbre enraciné à x est au moins $2^{\mathsf{rrang}(x)} - 1$.*

Vous pouvez utiliser ce théorème sans preuve en **b** ici. Pour 5 points de boni, donnez une preuve formelle au théorème.

a (7 points) Démontrez que

$$\mathsf{h}(x) \leq 3 \cdot \mathsf{rrang}(x) \qquad (\star)$$

pour chaque nœud x. (Vous avez besoin du théorème 1 dans la preuve.)

b (8 points) Utilisez le théorème 2 et l'équation ($\star$) pour démontrer que la hauteur d'un arbre RRN avec n nœuds internes est borné par $c\lfloor \lg(n+1) \rfloor$ — remplacez c par un nombre aussi petit que possible. Comparez ce résultat avec les bornes sur la hauteur des arbres AVL et RN.

6 Et maintenant, quelque chose de complètement différent (15 points de boni)

Les chaînes Fibonacci f_n sont des mots sur l'alphabet $\{\mathtt{1}, \mathtt{0}\}$ définis par induction de la manière suivante. Pour $n = 0$, $f_0 = \mathtt{1}$. Pour $n > 0$, f_n est dérivée de f_{n-1} en remplaçant chaque `1` par `0` et chaque `0` par `01`. (On remplace tous les caractères de f_{n-1} en même temps.) On a donc $f_0 = \mathtt{1}$, $f_1 = \mathtt{0}$, $f_2 = \mathtt{01}$, $f_3 = \mathtt{010}$, $f_4 = \mathtt{01001}$, $f_5 = \mathtt{01001010}$, $f_6 = \mathtt{0100101001001}$, ...

Démontrez que $f_n = f_{n-1} \cdot f_{n-2}$ pour chaque $n > 1$ (le symbol $\cdot$ dénote la concaténation).