IFT2015 H11 — Examen Intra

Miklós Csűrös

14 février 2011

Aucune documentation n'est permise. L'examen vaut 100 points. Vous pouvez avoir 10 points de boni additionnels.

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

F0 Votre nom (1 point)

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

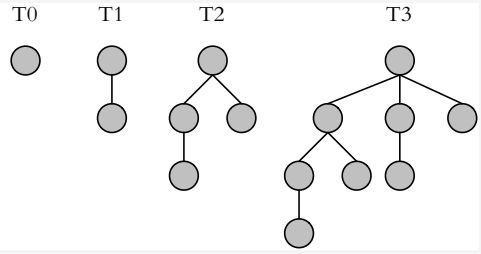
F1 Types abstraits (15 points)

Œuf de coucou (3 points) Sur la liste suivante, il y a trois types abstraits : ► identifiez lesquels.

> pile, liste chaînée, tas ternaire, queue (file FIFO), file de priorité, arbre binaire, Justin Bieber.

Interface (12 points) Les trois types abstraits (TA) identifiés sont des files généralisées avec des opérations élémentaires pour insérer et supprimer des éléments. ► Spécifiez les deux opérations pour chacun des TAs et expliquez comment elles changent l'ensemble d'éléments dans la file.

F2 Arbre binomial (20+10 points)



On définit les **arbres binomiaux** $T_0, T_1, T_2, \ldots$ par récursion :

- ⋆ T_0 contient un seul nœud (sa racine).
- ⋆ T_k contient un nœud, appellé la racine, avec les enfants $T_{k-1}, T_{k-2}, \ldots, T_0$, dans cet ordre de gauche à droit.

a. (10 points) ▶ Démontrez formellement que T_k contient exactement 2^k nœuds.

b. (10 points) ▶ Démontrez formellement qu'il y a exactement $\binom{k}{d}$ nœuds au niveau d dans T_k, où $\binom{k}{d} = \frac{k!}{d!(k-d)!}$ dénote le coefficient binomial. (D'où le nom de ces arbres.)

c. (10 points de boni) Supposons qu'on a deux arbres T_k avec des éléments stockés aux nœuds dans l'ordre de tas. ▶ Montrez comment fusionner les deux arbres pour obtenir T_{k+1}, en préservant l'ordre de tas, dans un temps de $O(k)$.

Indice : utilisez de l'induction. En **b.**, servez-vous des identitiés $\binom{k+1}{d+1} = \binom{k}{d} + \binom{k}{d+1} = \sum_{j=d}^{k} \binom{k}{j}$.

F3 Taux de croissance (20 points)

▶ Remplissez le tableau suivant : chaque réponse vaut 2 point. Pour chaque paire f, g, écrivez "=" si $\Theta(f) = \Theta(g)$, "$\ll$" si $f = o(g)$, "$\gg$" si $g = o(f)$, et "???" si aucun des trois cas n'applique. Il n'est pas nécessaire de justifier vos réponses. $\lg n$ dénote le logarithme binaire de n.

	$f(n)$	$g(n)$
a	$f(n) = 4n^2 \lg n$	$g(n) = 1 + 3n + 3n^2 + n^3$
b	$f(n) = n^{2015}$	$g(n) = n!$
c	$f(n) = \sum_{i=0}^{n} 2^i$	$g(n) = 2^n$
d	$f(n) = \sum_{i=1}^{n} 1/i$	$g(n) = \lg n$
e	$f(n) = 2.015^n$	$g(n) = 2^n$
f	$f(n) = 2.015^{\lg n}$	$g(n) = 2^{\lg n}$
g	$f(n) = n \lg n$	$g(n) = \lg(n!)$
h	$f(n) = n \lg n$	$g(n) = \begin{cases} 1 & \{n < 2\} \\ g(\lceil n/2 \rceil) + g(\lfloor n/2 \rfloor) + O(1) & \{n \geq 2\} \end{cases}$
i	$f(n) = \log_3(\log_4 n)$	$g(n) = \log_4(\log_3 n)$
j	$f(n) = \sqrt{n}$	$g(n) = \begin{cases} 1 & \{n < 2\} \\ g(n-2) + O(1) & \{n \geq 2\} \end{cases}$

F4 Q2D (29 points)

Proposez une structure pour stocker un ensemble de points (en 2D), qui permet les opérations add, deleteMinX et deleteMinY.

L'opération add(x, y) ajoute le point (x, y) à l'ensemble. L'opération deleteMinX supprime et retourne le point avec coordonnée X minimale («le plus gauche»).

L'opération deleteMinY supprime et retourne le point avec coordonnée Y minimale («le plus bas»). Par exemple, après add$(3,8)$, add$(1,12)$, add$(5,2)$, les opérations deleteMinX(), deleteMinY() retournent $(1,12)$ et $(5,2)$. Toutes les opérations doivent s'exécuter en $O(\log n)$ au pire quand l'ensemble contient n points. ▶ Montrez les détails des algorithmes principaux dans l'implantation. Il suffit de montrer l'implantation détaillée pour juste une des procédures symétriques (comme deleteMinX – deleteMinY).

Indice. Utilisez deux tas binaires sur les points avec des références aux indices entre eux. Par exemple, si un point est stocké à l'indice i dans le tas H avec les champs $H[i]$.x, $H[i]$.y et $H[i]$.indice, ce dernier donne l'indice du même point dans l'autre tas. Lors d'un swim ou sink (nager ou couler), on doit mettre à jour l'indice dans l'autre tas H' : $j \leftarrow H[i]$.indice; $H'[j]$.indice $\leftarrow i$ à chaque i visité. Une deleteMin dans un tas doit être accompagnée par la suppression de l'élément à l'indice $j = H[1]$.indice dans l'autre tas : généralisez la technique de deleteMin pour un indice quelconque j. Pour simplifier un peu, supposez que les tableaux sous-jacents ont une très grande capacité : il faut juste assurer que $H[i].\mathsf{x} = H[i].\mathsf{y} = \infty$ quand $i > n$.

compte comme 15 points boni dans l'évaluation finale; l'examen était évalué sur 100%=85 points.

F5 ~~Arbre semi-équilibré (15 points)~~

Soit $h(x)$ la hauteur d'un nœud x dans un arbre binaire :

$$h(x) = \begin{cases} 0 & \text{si } x \text{ est externe ;} \\ 1 + \max_{y \in \text{enfants de } x} h(y) & \text{si } x \text{ est interne} \end{cases}.$$

D'une façon similaire, on définit

$$r(x) = \begin{cases} 0 & \text{si } x \text{ est externe ;} \\ 1 + \min_{y \in \text{enfants de } x} r(y) & \text{si } x \text{ est interne} \end{cases}.$$

Pour un nœud externe, on a $h(x) = r(x) = 0$, même si on le représente par null et donc $h(\mathsf{null}) = r(\mathsf{null}) = 0$. Un arbre est **semi-équilibré** si $h(x) \leq 2 \cdot r(x)$ à tout x ▶ Donnez un algorithme qui vérifie si un arbre binaire est semi-équilibré. (Notez que l'algorithme doit calculer h et r.) L'algorithme doit s'exécuter en $O(n)$ sur un arbre de n nœuds. **Indice :** faire un parcours de l'arbre pour calculer $h(x)$ et $r(x)$ en parallèle.

BONNE CHANCE !

English translation

No documentation is allowed. The examen is worth 100 points, and you can collect an additonal 10 bonus points.

Answer each question in the exam booklet.

E0 Your name (1 point)

► Write your name and *code permanent* on each booklet that you submit.

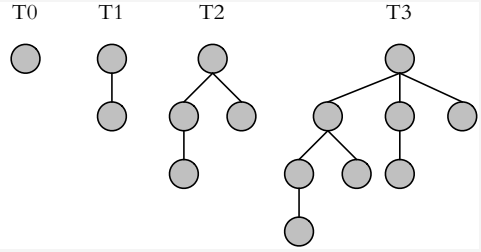
E1 Abstract types (15 points)

Cuckoo's Egg (3 points) There are three abstract data types in the following list : ► identify which ones.

> stack, linked list, ternary heap, queue, priority queue, binary tree, Justin Bieber.

Interface (12 points) The three identified abstract data types (ADTs) are generalized queues with elementary operations for insertion and removal of elements. ► Specify the two fundamental operations for each ADT, and explain how they change the set of elements in the queue.

E2 Binomial tree (20+10 points)



The **binomial trees** $T_0, T_1, T_2, \ldots$ are defined by recursion :

- ⋆ T_0 contains a single node (its root).
- ⋆ T_k contains a node, called its root, and the children $T_{k-1}, T_{k-2}, \ldots, T_0$, in this order from left to right.

a. (10 points) ► Show formally that T_k has exactly k nodes.

b. (10 points) ► Show formally that there are exactly $\binom{k}{d}$ nodes at depth d in T_k, where $\binom{k}{d} = \frac{k!}{d!(k-d)!}$ denotes the binomial coeficient (hence the name).

c. (10 bonus points) Suppose that we have two trees T_k in which elements are stored at the nodes in heap order. ► Show how to merge the two trees into T_{k+1}, while preserving the heap order, in $O(k)$ time.

Hint : use induction. In **b.**, use the identities $\binom{k+1}{d+1} = \binom{k}{d} + \binom{k}{d+1} = \sum_{j=d}^{k} \binom{k}{j}$.

E3 Growth rates (20 points)

▶ Fill out the following table : every answer is worth 2 points. For each pair f, g, write "=" if $\Theta(f) = \Theta(g)$, "$\ll$" if $f = o(g)$, "$\gg$" if $g = o(f)$, and "???" if neither of the three applies. You do not need to justify the answers. $\lg n$ denotes the binary logarithm of n.

	$f(n)$	$g(n)$
a	$f(n) = 4n^2 \lg n$	$g(n) = 1 + 3n + 3n^2 + n^3$
b	$f(n) = n^{2015}$	$g(n) = n!$
c	$f(n) = \sum_{i=0}^{n} 2^i$	$g(n) = 2^n$
d	$f(n) = \sum_{i=1}^{n} 1/i$	$g(n) = \lg n$
e	$f(n) = 2.015^n$	$g(n) = 2^n$
f	$f(n) = 2.015^{\lg n}$	$g(n) = 2^{\lg n}$
g	$f(n) = n \lg n$	$g(n) = \lg(n!)$
h	$f(n) = n \lg n$	$g(n) = \begin{cases} 1 & \{n < 2\} \\ g(\lceil n/2 \rceil) + g(\lfloor n/2 \rfloor) + O(1) & \{n \geq 2\} \end{cases}$
i	$f(n) = \log_3(\log_4 n)$	$g(n) = \log_4(\log_3 n)$
j	$f(n) = \sqrt{n}$	$g(n) = \begin{cases} 1 & \{n < 2\} \\ g(n-2) + O(1) & \{n \geq 2\} \end{cases}$

E4 Q2D (29 points)

Propose a data structure for storing a set of points (in 2D), implementing the operations add, deleteMinX and deleteMinY.

The add(x, y) operation adds the point (x, y). deleteMinX deletes and returns the point with minimal X coordinate. deleteMinY deletes and returns the point with minimal Y coordinate. For instance, after add$(3, 8)$, add$(1, 12)$, add$(5, 2)$, the operations deleteMinX(), deleteMinY() return $(1, 12)$ et $(5, 2)$. Each operation must take $O(\log n)$ time at worst when the queue has n points. ▶ Show the details of the principal algorithms in the implementation. It is enough to show the details for just one of two symmetrical procedures (like deleteMinX − deleteMinY).

Hint. Use two min-heaps, and store references to indices between them. For instance, a point can be stored at the index i in heap H with fields $H[i]$.x, $H[i]$.y and $H[i]$.indice, where the last one gives the index to the same point's position in the other heap. During swim and sink, one needs to maintain the correct indexes in the other heap $H' : j \leftarrow H[i]$.indice; $H'[j]$.indice $\leftarrow i$ at every index i visited in heap H. A deleteMin in one of the heaps must be coupled with deleting the element at index $j = H[1]$.indice on the other heap : generalize the technique of deleteMin for an arbitrary index j. In order to simplify the discussion, you can assume that

the allocated tables are large enough : just make sure that $H[i].\mathsf{x} = H[i].\mathsf{y} = \infty$ for $i > n$.

E5 Half-balanced tree (15 points)

Let $h(x)$ be the height of node x in a binary tree :

$$h(x) = \begin{cases} 0 & \text{if } x \text{ is external}\,; \\ 1 + \max_{y \in \text{children of } x} h(y) & \text{if } x \text{ is internal} \end{cases}.$$

Similarly, define

$$r(x) = \begin{cases} 0 & \text{if } x \text{ is external}\,; \\ 1 + \min_{y \in \text{children of } x} r(y) & \text{if } x \text{ is internal} \end{cases}.$$

For an external node, $h(x) = r(x) = 0$, even if it is represented by null, so $h(\mathsf{null}) = r(\mathsf{null}) = 0$. A binary tree is **half-balanced** if $h(x) \leq 2r(x)$ at every node x. ▶ Give an algorithm that checks if a binary tree is half-balanced. (Note that the algorithm has to compute h and r.) The algorithm must run in $O(n)$ on a tree with n nodes. **Hint :** do a tree traversal and compute $h(x)$ and $r(x)$ in parallel.

GODSPEED !