IFT2015 A11 — Examen Intra

Miklós Csűrös

21 octobre 2011

The English translation is on the last pages

Aucune documentation n'est permise. L'examen vaut 100 points. Vous pouvez avoir jusqu'à 15 points de boni additionnels (pour des questions dénotées par ♡).

► **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 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	$f(n) = 2015n$	$g(n) = n^2/2015$
b	$f(n) = \sqrt{n}$	$g(n) = \sqrt[3]{n}$
c	$f(n) = \sum_{i=0}^{n} 3^i$	$g(n) = 3^n$
d	$f(n) = \sum_{i=1}^{n} n/i$	$g(n) = n \lg n$
e	$f(n) = n!$	$g(n) = (n+1)!$
f	$f(n) = 2.015^n$	$g(n) = 2.015^{n+1}$
g	$f(n) = 2.015^n$	$g(n) = 2.015^{2n}$
h	$f(n) = n \lg n$	$g(n) = \lg(n!)$
i	$f(n) = \max\{1, n + 2015 \sin n\}$	$g(n) = \begin{cases} 1 & \{n = 0\} \\ g(n-1) + \Theta(1) & \{n > 0\} \end{cases}$
j	$f(n) = n!$	$g(n) = \begin{cases} 1 & \{n = 0\} \\ 2ng(n-1) & \{n > 0\} \end{cases}$

F2 Parenthèses (20 points)

Le langage $\mathsf{Dyck}(s)$ est celui des expressions avec s types de parenthèses : $\underset{\smile}{j}$, $\widehat{j}$ pour $j = 1, 2 \ldots, s$ (comme le langage XML). Une expression valide est dérivée par les règles

$$
\begin{aligned}
E &\mapsto \varepsilon && \textit{ε dénote l'expression vide}\\
E &\mapsto E\,E && \textit{plusieurs termes}\\
E &\mapsto \underset{\smile}{1}\,E\,\widehat{1} && \textit{parenthèses de type } 1\\
&\ldots\\
E &\mapsto \underset{\smile}{s}\,E\,\widehat{s} && \textit{parenthèses de type } s
\end{aligned}
$$

Donc $\underset{\smile}{1}\,\underset{\smile}{2}\,\widehat{2}\,\underset{\smile}{3}\,\widehat{3}\,\widehat{1}\,\underset{\smile}{1}\,\widehat{1}$ est valide, mais $\underset{\smile}{1}\,\underset{\smile}{2}\,\widehat{2}\,\underset{\smile}{3}\,\widehat{1}\,\widehat{3}$ ne l'est pas. Une *expression encodée* est un tableau $x[0..n-1]$ avec éléments $x[i] \in \{\pm 1, \pm 2, \ldots, \pm s\}$, où $+j$ est le code de parenthèse ouvrant $\underset{\smile}{j}$, et $-j$ est le code de parenthèse fermant $\widehat{j}$ pour $j = 1, \ldots, s$.

► Donnez un algorithme pour vérifier si un encodage $x[0..n-1]$ est valide, en temps $O(n)$. Justifier bien que votre algorithme est correct.

F3 File de quelques priorités (20+5 points)

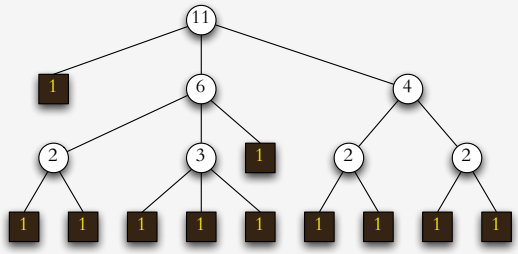
Supposons que la priorité peut prendre juste un petit nombre de valeurs. Dans un tel cas, le tas binaire n'est pas nécessairement l'implantation la plus efficace pour supporter les opérations de base de l'interface.

► Proposez une structure de données pour une file de priorités où la rangée de priorités possibles $\{0, 1, \ldots, m-1\}$ est connue. Montrez l'implantation des opérations usuelles de l'interface, ainsi que l'initialisation d'une file vide avec m fourni comme argument. La structure doit performer $\mathsf{deleteMin}()$ et $\mathsf{insert}(x)$ en $O(m)$ au pire.

♡[**5 points boni**] Donnez une solution avec $O(\log m)$ temps au pire pour $\mathsf{deleteMin}$ et insert.

La priorité du paramètre x de l'insertion est donnée par la fonction $\phi(x)$ prédéfinie. Dans la déscription de votre algorithme, donnez seulement autant de détails que nécessaire : servez-vous des interfaces de types abstraits communs, et/ou de techniques de base avec des structures discutées (p.e., suppression de nœuds sur une liste chaînée, ou nager/couler avec un tas binaire).

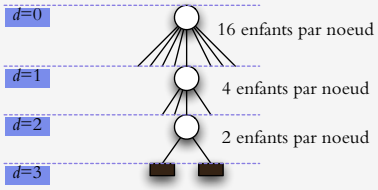
F4 Comptez les nœuds (19 points)



Dans l'arbre enraciné T, chaque nœud interne x possède des enfants dont l'ensemble est stocké par la variable x.children. Soit $n[x]$ le nombre de nœuds externes dans le sous-arbre enraciné à x (y incluant x même si c'est un nœud externe).

a. Calculer la taille (10 points)

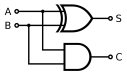
- ► Donnez une définition récursive de $n[x]$.
- ► Donnez un algorithme récursif pour déterminer $n[x]$ quand x (interne ou externe) est fourni comme argument. L'algorithme doit s'exécuter dans un temps linéaire (en $n[x]$).

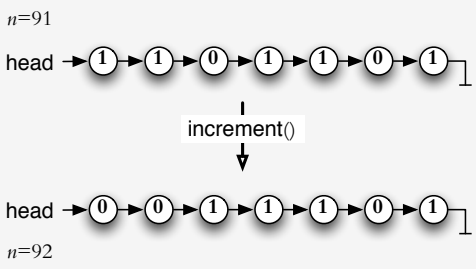


On définit *l'arbre hollandais* de hauteur h comme un arbre ordonné où les nœuds internes au chaque niveau $d = 0, 1, 2, \ldots, h-1$ sont d'arité $2^{2^{h-d-1}}$. Les nœuds externes se trouvent sur niveau h.

b. Arbre hollandais (9 points) ► Montrez que l'arité de la racine d'un arbre hollandais à n nœuds externes est $\Theta(\sqrt{n})$.

F5 Arithmétique binaire (20+10 points)





On veut une structure de données pour représenter des entiers non-négatifs de grandeur arbitraire. Pour cela, on utilise une liste chaînée de bits. Un nœud x de la liste chaînée contient donc les variables $x.\mathsf{next}$ pour le prochain nœud ($=$ null à la fin), et $x.\mathsf{bit}$ pour le bit même (qui prend la valeur 0 ou 1).

La tête de la liste (head) est le nœud pour le bit de poids faible. On représente le nombre $n = 0$ par une liste comportant un seul nœud x dont $x.\mathsf{bit} = 0$. Un nombre $n > 0$ avec $2^{k-1} \leq n < 2^k$ est représenté par une liste de longueur k : $n = \sum_{i=0}^{k-1}(x_i.\mathsf{bit}) \cdot 2^i$ où x_i est le i-ème nœud après la tête.

- ▶ Donnez une implantation de l'opération increment() qui incrémente (par un) le nombre représenté. L'algorithme doit mettre à jour la liste (au lieu de créer une autre liste). Analysez la croissance asymptotique du temps de calcul au pire cas en fonction de la valeur incrémentée n. Est-ce qu'on peut dire que votre algorithme prend un temps *linéaire* ? Justifiez votre réponse.
- ♡[**5 points boni**] Démontrez que le temps amorti de increment() est $O(1)$ dans votre implantation.
- ♡[**5 points boni**] Donnez une implantation de increment pour l'*encodage Fibonacci*. Dans cet encodage, une liste de longueur k représente le nombre $n = \sum_{i=0}^{k-1}(x_i.\mathsf{bit}) \cdot F(i+2)$, où $F(i)$ est le i-ème nombre Fibonacci. (Rappel : $F(0) = 0$, $F(1) = 1$.) Notez que la représentation Fibonacci n'est pas unique : par exemple, $19 = \overline{101001} = \overline{11111}$ car $19 = 13 + 5 + 1 = 8 + 5 + 3 + 2 + 1$. **Indice.** La clé est d'utiliser la représentation de poids minimal : c'est celle qui minimise $\sum_i x_i.\mathsf{bit}$. On peut toujours remplacer la suite $\overline{011}$ par $\overline{100}$ car $F(i) = F(i-2) + F(i-1)$.

BONNE CHANCE !

English translation

No documentation is allowed. The examen is worth 100 points, and you can collect up to 15 additional bonus points for exercises marked by ♡. You may write your answers in English or in French.

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 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	$f(n) = 2015n$	$g(n) = n^2/2015$
b	$f(n) = \sqrt{n}$	$g(n) = \sqrt[3]{n}$
c	$f(n) = \sum_{i=0}^{n} 3^i$	$g(n) = 3^n$
d	$f(n) = \sum_{i=1}^{n} n/i$	$g(n) = n \lg n$
e	$f(n) = n!$	$g(n) = (n+1)!$
f	$f(n) = 2.015^n$	$g(n) = 2.015^{n+1}$
g	$f(n) = 2.015^n$	$g(n) = 2.015^{2n}$
h	$f(n) = n \lg n$	$g(n) = \lg(n!)$
i	$f(n) = \max\{1, n + 2015 \sin n\}$	$g(n) = \begin{cases} 1 & \{n = 0\} \\ g(n-1) + \Theta(1) & \{n > 0\} \end{cases}$
j	$f(n) = n!$	$g(n) = \begin{cases} 1 & \{n = 0\} \\ 2ng(n-1) & \{n > 0\} \end{cases}$

E2 *Parenthèses* (20 points)

The formal language $\mathsf{Dyck}(s)$ is the one of well-formed expressions with s types of parentheses : $\underset{\smile}{j}, \overset{\frown}{j}$ for $j = 1, 2 \ldots, s$ (like the XML language). A valid expression is derived by the rules

$$
\begin{array}{ll}
E \mapsto \varepsilon & \textit{ε is the empty string} \\
E \mapsto E\,E & \textit{more than one term side by side} \\
E \mapsto \underset{\smile}{1}\,E\,\overset{\frown}{1} & \textit{parentheses of type } 1 \\
\ldots & \\
E \mapsto \underset{\smile}{s}\,E\,\overset{\frown}{s} & \textit{parentheses of type } s
\end{array}
$$

So, $\underset{\smile}{1}\,\underset{\smile}{2}\,\overset{\frown}{2}\,\underset{\smile}{3}\,\overset{\frown}{3}\,\overset{\frown}{1}\,\underset{\smile}{1}\,\overset{\frown}{1}$ is valid, but $\underset{\smile}{1}\,\underset{\smile}{2}\,\overset{\frown}{2}\,\underset{\smile}{3}\,\overset{\frown}{1}\,\overset{\frown}{3}$ is not. An *encoded expression* is an array $x[0..n-1]$ with elements $x[i] \in \{\pm 1, \pm 2, \ldots, \pm s\}$, where $+j$ encodes opening parenthesis $\underset{\smile}{j}$, and $-j$ encodes closing parenthesis $\overset{\frown}{j}$ for $j = 1, \ldots, s$.

► Give an algorithm to verify if an encoded expression is valid, in $O(n)$ time. Justify why your algorithm is correct.

E3 Queue with just a few priorities (20+5 points)

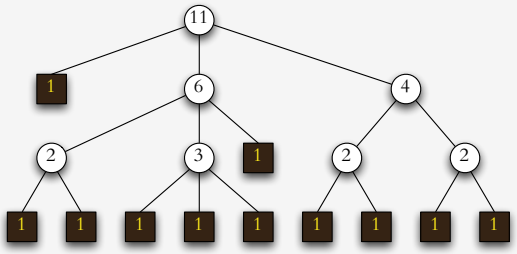
Suppose that the priorities have only a few possible values. It is then possible to design a better data structure for a priority queue than the binary heap.

► Propose a data structure implementing a priority queue for a known set of possible priorities $\{0, \ldots, m-1\}$. Show how the usual operations of the interface are implemented, and the initialization of the structure with m passed as an argument. The data structure must support $\mathsf{deleteMin}()$ and $\mathsf{insert}(x)$ in $O(m)$ worst-case time.

♡[**5 points boni**] Give a solution performing $\mathsf{deleteMin}$ and insert in $O(\log m)$ time.

In your algorithm, use the prefined function $\phi(x)$ to get the priority of x (the parameter of insert). Show only as much details as necessary. You do not need to show how common abstract data type interfaces are implemented. Rely on basic techniques on data structures discussed at the course (e.g., deleting a node on a linked list or swim/sink in a binary heap), but indicate clearly how you use them.

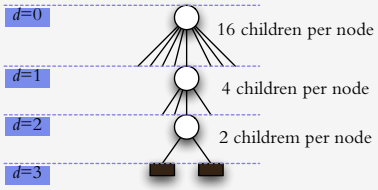
E4 Node counting (19 points)



In a rooted tree T, every internal node x stores its children in the variable x.children. Let $n[x]$ be the number of external nodes in the subtree rooted at x (including x if it is external).

a. Calculating size (10 points)

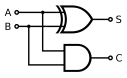
- ▶ Give a recursive definition for $n[x]$.
- ▶ Give a recursive algorithm for determining $n[x]$ for a node x (internal or external) passed as argument. The algorithm must take linear time (in the number of external nodes $n[x]$).

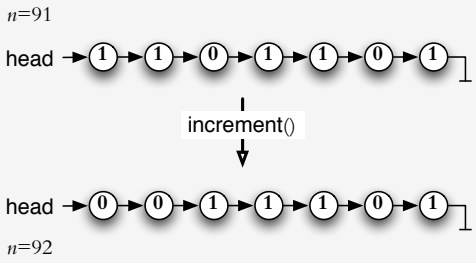


A *Dutch tree* of height h is an ordered tree where the internal nodes at each level $d = 0, 1, 2, \ldots, h-1$ have exactly $2^{2^{h-d-1}}$ children each. The external nodes are at level h.

b. Dutch tree (9 points) ▶ Show that the root of a Dutch tree with n external nodes has $\Theta(\sqrt{n})$ children.

E5 Binary arithmetics (20+10 points)





Suppose that one uses a linked list to represent arbitrarily large non-negative integers. Each list node x is equipped thus with x.next for next node (= null denotes the end), and x.bit for storing one bit in the binary representation (taking the value 0 or 1).

The list head (head variable) gives the least significant bit. The number 0 is represented by a list with a single node x where $x.\mathsf{bit} = 0$. A positive number $n > 0$ with $2^{k-1} \leq n < 2^k$ is represented by a list of length k : $n = \sum_{i=0}^{k-1}(x_i.\mathsf{bit}) \cdot 2^i$ where x_i is the i-th node after the head (head is x_0).

- ▶ Give an algorithm implementing the increment() operation which increments (by one) the number represented by the list. (The algorithm should not create a different list for the result, but rather update the list itself.) Analyze the asymptotic growth for the worst-case running time in function of the icremented value n. Would you say that our algorithm takes a *linear* time ? Justify your response.
- ♡[**5 points boni**] Show that increment takes $O(1)$ amortized time in your implementation.
- ♡[**5 points boni**] Implement increment for *Fibonacci encoding*. In that encoding, a liste with k nodes represents the number $n = \sum_{i=0}^{k-1}(x_i.\mathsf{bit}) \cdot F(i+2)$, where $F(i)$ is the i-th Fibonacci number. (Reminder : $F(0) = 0, F(1) = 1$.) Note that the Fibonacci representation is not unique : for example, $19 = \overline{101001} = \overline{11111}$ since $19 = 13 + 5 + 1 = 8 + 5 + 3 + 2 + 1$. **Hint.** The key is to use the minimum-weight encoding which is the one minimizing $\sum_i x_i.\mathsf{bit}$. For instance, one can always replace the bit sequence $\overline{011}$ by $\overline{100}$ because $F(i) = F(i-2) + F(i-1)$.