IFT2015 automne 2011 — Examen Final

Miklós Csűrös

9 décembre 2011

Aucune documentation n'est permise. L'examen vaut 150 points. Il comprend 7 exercices réguliers (F0–F6) et un exercice de bonus. Vous pouvez avoir 30 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 Table de symboles (39 points)

(a) 3 structures, 3 opérations, 3 performances (27 points) On a vu plusieurs structures de données qui peuvent implanter le type abstrait de la table de symboles. Ici vous devez comparer le temps de calcul pour trois opérations : insertion, recherche fructueuse, et recherche infructueuse. ► Donnez le temps de calcul des trois opérations avec les trois structures de données suivantes : une liste chaînée d'éléments non-triés, un arbre rouge-et-noir, et un tableau de hachage avec sondage linéaire (facteur de remplissage $\alpha < 3/4$). Spécifiez le temps de calcul en notation asymptotique comme une fonction du nombre des éléments n, dans trois cas : le pire cas, le meilleur cas, et en moyenne. Il ne faut pas justifier vos réponses.

(b) Justifications (12 points) ► Élaborez *3* de vos réponses (votre choix lesquels parmi les 27) en **(a)** : expliquez comment la structure assure un tel temps de calcul.

F2 Types abstraits (20 points)

(a) Définitions (4 points) ► Donnez les définitions de *type* [de données] et de *type abstrait* [de données].

(b) Types fameux (16 points) ► Décrivez les opérations principales pour les types abstraits suivants, sans discuter l'implantation : (1) pile, (2) file FIFO (queue), (3) file de priorité, (4) table de symboles.

F3 Tris (20 points)

► Pour chacun des tris suivants, donnez le temps de calcul en meilleur, moyen et pire cas, ainsi que l'espace de travail utilisé au pire : tri rapide, tri par tas, tri par fusion, tri par insertion, tri par sélection. Donnez vos réponses en notation asymptotique pour un tableau de taille n. (Notez que l'espace de travail inclut toute les variables locales à part du tableau à l'entrée, et dépend de la profondeur de la pile d'exécution quand le tri utilise de la récursion.) Il n'est pas nécessaire de justifier vos réponses.

F4 Sondage dubieux (20 points)

Soit un tableau de hachage $H[0..m-1]$ initialement vide ($\forall i\colon H[i] = \mathsf{null}$), utilisant l'adressage ouvert avec la fonction de hachage h (connue). Après n insertions ($0 \le n \le m$), H contient n éléments non-nulls. Un tableau est *valide* si les éléments ont été insérés par le code suivant :

```
INSERT(x)                                   // insertion de l'élément x par sondage linéaire
I1  j ← h(x)
I2  for k ← j, j+1, ..., m−1 do if H[k] = null then H[k] ← x ; return
I3  for k ← 0, 1, ..., j−1 do if H[k] = null then H[k] ← x ; return
I4  erreur de débordement : trop d'éléments dans le tableau
```

► Donnez un algorithme VALID($H[0..m-1]$) qui vérifie si le tableau est valide.

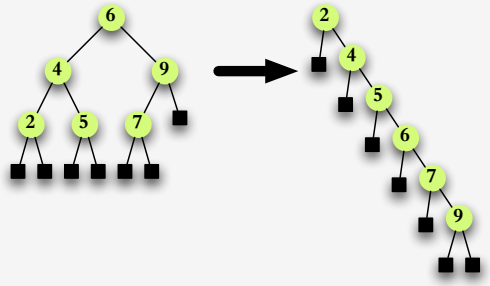
Indice : il faut parcourir les indices pour vérifier si $h(H[i])$ est possible pour tout i avec $H[i] \neq \mathsf{null}$. Faites attention aux cas où $H[m-1] \neq \mathsf{null}$ ou $n = m$.

F5 Rotations (30+10 points)

Notation pour arbres binaires de recherche : à chaque nœud interne N, la variable $N.\mathsf{key}$ donne la clé, $N.\mathsf{left}$ et $N.\mathsf{right}$ donnent les enfants gauche et droit (null si enfant externe), $N.\mathsf{parent}$ donne le parent (null si N est la racine).

(a) Une rotation (10 points) ► Montrez le code de la procédure ROTR(x) qui performe une rotation droite à nœud x (impliquant son enfant gauche qui devient le parent de x après la rotation).

(b) Arbre → liste (20 points)



► Donnez un algorithme pour transformer un arbre binaire de recherche (spécifié par sa racine r) en une chaîne orientée à droite : c'est à dire un ABR équivalent dans lequel aucun enfant gauche n'est interne. **Indice :** utilisez des rotations droites.
► Analysez le temps de calcul de votre algorithme.

(c$^\heartsuit$) Arbre → arbre (10 points boni) ► Montrez qu'un arbre binaire de recherche arbitraire peut être transformé en n'importe quel autre arbre binaire de recherche sur le même ensemble de n clés, à l'aide de $O(n)$ rotations.

F6 Les Immortels (20 points)

Les Immortels doivent se battre entre eux en duels. Chaque Immortel x possède une *classe* $c[x]$ et une *puissance* $p[x]$ qui déterminent la démarche des duels selon les règles suivantes :

(i) Pour tout Immortel x, $c[x] = 0$ et $p[x] = 1$ au début.

(ii) Lorsque le vainqueur (x) d'un duel élimine[1] son adversaire (y), il absorbe la puissance libérée de l'autre[2] : $p[x] \leftarrow p[x] + p[y]$.

(iii) Dans un duel de classes inégales, celui de classe inférieure perd (même si sa puissance est supérieure).

(iv) Un duel entre deux Immortels de même classe peut mener au victoire de l'un ou de l'autre, en augmentant le classement du vainqueur (x) : $c[x] \leftarrow c[x] + 1$.

Notez qu'en (iv), le duel finit avec le victoire de l'un ou de l'autre, changeant la puissance et la classe. En (iii), la puissance change, mais le classement ne change pas. Exemples :

adversaires avant	adversaires après
Ramirez : $c = 4, p = 17$; Kurgan : $c = 4, p = 47$	Ramirez : mort ; Kurgan : $c = 5, p = 64$
Kastagir : $c = 3, p = 12$; Kurgan : $c = 5, p = 64$	Kastagir : mort ; Kurgan : $c = 5, p = 76$
MacLeod : $c = 5, p = 34$; Kurgan : $c = 5, p = 76$	MacLeod : $c = 6, p = 110$; Kurgan : mort

► Démontrez que la puissance d'un Immortel de classe c est au moins 2^c : $p[x] \geq 2^{c[x]}$ pour tout x.

Indice : utilisez de l'induction en duels.

F7♡ Recherche optimale (20 points boni)

L'algorithme suivant retourne le nœud avec clé x dans un ABR (notation comme en F5) :

```
    SEARCH(x, N)                                  // recherche dans le sous-arbre du nœud N
S1  if N = null then return null                  // recherche infructueuse
S2  if x < N.key then return SEARCH(x, N.left)
S3  if x = N.key then return N                    // recherche fructueuse
S4  else return SEARCH(x, N.right)                // {x > N.key}
```

(a♡) Définition (5 points) Soit $t[N]$ le nombre de comparaisons arithmétiques entre x et des clés lors d'une recherche fructueuse finissant avec N (dernière comparaison : $x = N.\mathsf{key}$). ► Donnez une définition récursive de $t[N]$. **Indice :** Il y a une comparaison à compter en Ligne S2 et une autre en Ligne S3. Notez que t croît différemment vers la gauche et vers la droite.

(b♡) Isobathes (5 points) Rapel : $F(k)$ dénote le k-ème nombre Fibonacci : $F(0) = 0$, $F(1) = 1$ et $F(k) = F(k-1) + F(k-2)$ pour tout $k > 1$. ► Démontrez que pour tout $k = 2, 3, \ldots$, il y a tout au plus $F(k-1)$ nœuds internes avec $t[N] = k$.

(c♡) Arbre optimal (5 points) Soit $t_{\max}$ le maximum de $t[N]$ dans l'arbre. ► Caractérisez les arbres avec le maximum nombre de nœuds pour un $t_{\max}$ donné.

(d♡) Le vrai nombre de comparaisons (5 points) Soit $\phi = \frac{1+\sqrt{5}}{2}$. ► Démontrez que dans un arbre avec n nœuds internes, $t_{\max} \geq \log_\phi(n+1) + c$ avec une constante c qui ne dépend pas de n.

[1] par décapitation

[2] dans un phénomène formidable appelé *quickening*

English translation

No documentation is allowed. The examen is worth 150 points. You can collect an additonal 30 bonus points. The examen includes 6 regular exercises (E0–E6) and a bonus exercise (E7).

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 Symbol table (39 points)

(a) 3 structures, 3 operations, 3 performance guarantees (27 points) We have seen a number of data structures that can be used to implement the abstract data type of symbol table. Not all implementations have the same performance. You need to compare the running times for three fundamental operations : insertion, successful search and unsuccessful search. ► Give the running times for the tree operations in the three following data structures : unsorted linked list, red-black tree, hash table with linear probing and a load factor of $\alpha < 3/4$. Give the worst-case, best-case and average-case running times in asymptotic notation as a function of the number n of elements. (That is, 27 statements about time complexity.) You do not need to justify your answers.

(b) Justifications (12 points) ► Develop *3* of your answers (of your choice among the 27) from **(a)**. Explain how the structure allows for such performance.

E2 Abstract types (20 points)

(a) Definitions (4 points) ► Give the definitions for [data] *type* and *abstract data type*.

(b) Famous types (16 points) ► Describe the fundamental operations for each of the following abstract data types, without discussing implementation : (1) stack, (2) queue, (3) priority queue, (4) symbol table.

E3 Sorting (20 points)

► Give the running time in the best, worst, and average case, as well as the worst-case work space requirements for the following sorting algorithms : quicksort, heapsort, mergesort, insertion sort, and selection sort. Give your answers in asymptotic notation for a table of n elements. (Note that the work space includes all allocated local variables aside from the input table, and depends on the depth of the call stack when recursion is used.) You do not need to justify your answers.

E4 Dubious probes (20 points)

Let $H[0..m-1]$ be an initially empty hash table ($\forall i\colon H[i] = \mathsf{null}$) that uses open addressing with a known hash function h. After n insertions ($0 \leq n \leq m$), H has n non-null entries. A table is *valid* if its elements were inserted using the following code :

```
INSERT(x)                                                    // insertion of x using linear probing
I1  j ← h(x)
I2  for k ← j, j+1, ..., m−1 do if H[k] = null then H[k] ← x ; return
I3  for k ← 0, 1, ..., j−1 do if H[k] = null then H[k] ← x ; return
I4  overflow error : too many elements in the table
```

► Give an algorithm VALID($H[0..m-1]$) that verifies whether the table H is valid.

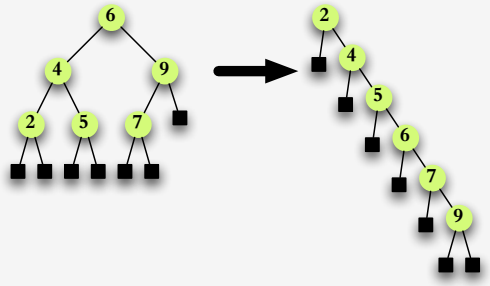
Hint : one needs to check at each index i with $H[i] \neq \mathsf{null}$ whether $h(H[i])$ is viable. Pay attention to the cases when $H[m-1] \neq \mathsf{null}$ or $n = m$.

E5 Rotations (30+10 points)

Notation for binary search trees : at every internal node N, the variable $N.\mathsf{key}$ gives the key, $N.\mathsf{left}$ et $N.\mathsf{right}$ give the left and right children (null if external child), $N.\mathsf{parent}$ gives the parent (null if N is the root).

(a) One rotation (10 points) ► Show the code for ROTR(x) that performs a right rotation at node x (involving its left child, which becomes the parent of x after the rotation).

(b) Tree → list (20 points)



► Give an algorithm to transform a binary search tree (specified by its root r) into a right-leaning chain (i.e., an equivalent binary search tree where every left child is external). **Hint :** use a series of right rotations.

► Analyze your algorithm's running time.

(c$^{\heartsuit}$) Tree → tree (10 bonus points) ► Show that an arbitrary binary search tree can be transformed into another binary search tree over the same set of n keys using $O(n)$ rotations.

E6 There can be only one (20 points)

The Immortals have to fight between each other in duels. Every Immortal x has a *class* $c[x]$ and a *power* $p[x]$, determining the duel's outcome by the following combat rules :

(i) Initially, $c[x] = 0$ and $p[x] = 1$ for each Immortal x.

(ii) A duel's victor (x) eliminates[3] his adversary (y), and absorbs[4] the released power of the other : $p[x] \leftarrow p[x] + p[y]$.

(iii) In a duel between unequal classes, the one with the lower class loses (even if he has more power).

(iv) In a duel between Immortals (x, y) of the same class, either of them can win, increasing the victor's class : $c[x] \leftarrow c[x] + 1$.

Note that in (iv), either x or y wins, boosting the victor's class and power. In (iii), only the power changes, but the class stays the same. Examples :

adversaries before	adversaries after
Ramirez : $c = 4, p = 17$; Kurgan : $c = 4, p = 47$	Ramirez : dead ; Kurgan : $c = 5, p = 64$
Kastagir : $c = 3, p = 12$; Kurgan : $c = 5, p = 64$	Kastagir : dead ; Kurgan : $c = 5, p = 76$
MacLeod : $c = 5, p = 34$; Kurgan : $c = 5, p = 76$	MacLeod : $c = 6, p = 110$; Kurgan : dead

► Show that an Immortal in class c has power at least 2^c : $p[x] \geq 2^{c[x]}$ for all x.

Hint : use induction by duels.

E7$^\heartsuit$ Optimal search (20 bonus points)

The next algorithm finds the node with key x in a binary search tree (notation as in E5) :

```
    SEARCH(x, N)                                   // search in the subtree of node N
S1  if N = null then return null                   // unsuccessful search
S2  if x < N.key then return SEARCH(x, N.left)
S3  if x = N.key then return N                     // successful search
S4  else return SEARCH(x, N.right)                 // {x > N.key}
```

(a$^\heartsuit$) Definition (5 points) Let $t[N]$ be the number of arithmetic comparisons ($a = b$ and $a < b$) involving x and some node key during a successful search ending with N (last comparison : $x = N.\mathsf{key}$). ► Give a recursive definition for $t[N]$. **Hint :** there is one comparison to count in Line S2 and another in Line S3. Note that t grows asymmetrically towards left and right.

(b$^\heartsuit$) Isobaths (5 points) Recall that $F(k)$ denotes the k-th Fibonacci number : $F(0) = 0$, $F(1) = 1$ and $F(k) = F(k-1) + F(k-2)$ for all $k > 1$. ► Show that for all $k = 2, 3, \ldots$, there are at most $F(k-1)$ internal nodes with $t[N] = k$.

(c$^\heartsuit$) Optimal tree (5 points) Let $t_{\max}$ be the maximum of $t[N]$ over the tree's internal nodes. ► Characterize the trees with maximum number of nodes for a given $t_{\max}$.

(d$^\heartsuit$) The true number of comparisons (5 points) Let $\phi = \frac{1+\sqrt{5}}{2}$. ► Show that in a tree of n internal nodes, $t_{\max} \geq \log_\phi(n+1) + c$ with some constant c that does not depend on n.

[3] by decapitation

[4] in an impresssive phenomenon called *quickening*