IFT2015 A10 — Examen Final

Miklós Csűrös

10 décembre 2010

The English translation starts on page 5.

Aucune documentation n'est permise. L'examen vaut 150 points. En complétant tous les exercices de Section 4, et l'œuf de Pâques, vous pouvez récupérer jusqu'à 35 points de boni.

► **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 Opérations (10 points)

Décrivez les opérations de base dans les types abstraits suivants : pile, file FIFO (queue), file à priorités (min-tas).

2 Table de symboles (39 points)

(a) 3 structures, 3 opérations, 3 performances (27 points) On a vu plusieurs structures de données qui peuvent servir à implémenter le type abstrait de la table de symboles. L'efficacité des implémentations n'est pas la même : ici vous devez comparer le temps de calcul pour trois opérations fondamentales : 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 2-3-4, et un tableau de hachage avec chaînage séparé, dont la facteur de remplissage $\alpha = O(1)$. Spécifiez le temps de calcul comme une fonction du nombre des éléments n, en utilisant la notation asymptotique, dans trois cas : le pire cas, le meilleur cas, et en moyenne. Il ne faut pas justifier vos réponses.

(b) Arbres binaires de recherche (12 points) Les arbres binaires de recherche (ABR) permettent une implantation du tableau de symboles avec l'assurance de performer les opérations de base en $O(\log n)$, *en un sens quelconque*. ► Expliquez les types d'assurances de performance par les implantations d'ABRs suivantes : (1) ABR simple [opérations de base sans rotations], (2) ABR déployé (*splay*), et (3) ABR équilibré. Répondez, p.e., que «les opérations prennent $O(\log n)$ *au pire* en implantation X», avec quelques mots de justification (sans trop de détails, et surtout pas de preuves). Expliquez bien la notion du «moyen» si c'est le genre de performance mentionnée.

3 Easter egg (10 points de boni)

Si vous avez répondu à Question 2b, vous avez le droit à 10 points de boni en répondant à la question suivante : ► quel est le temps de calcul des opérations de base dans la structure union-find (appartenance-union) la plus efficace vue au début du cours ? Donnez une réponse aussi précise que possible.

4 Quatre sur cinq (100+25 points)

Vous devez compléter 4 sur les 5 exercices de §4.1–4.5. Si vous complétez un cinquième, vous pouvez avoir jusqu'à 25 points de boni.

4.1 Vérification de coloriage (25 points)

Supposons qu'on a un arbre rouge-et-noir mais on n'est pas certain si le coloriage est valide. ► Donnez un algorithme qui vérifie si le coloriage est valide : (1) chaque nœud rouge doit avoir un parent noir, (2) les nœuds avec enfant(s) null doivent être noir, et (3) on doit avoir le même nombre de nœuds noirs sur tout chemin qui départ du même nœud x et arrive à une feuille null.

Syntaxe : chaque nœud [non-null] x contient les champs x.left, x.right, x.parent et x.color pour enfants gauche et droit, le parent, et la couleur. (***Indice*** *: on peut vérifier les propriétés dans un seul parcours. En (3), imaginez que vous devez calculer la hauteur noire [ou «rang» dans mes notes de cours] de chaque nœud.*)

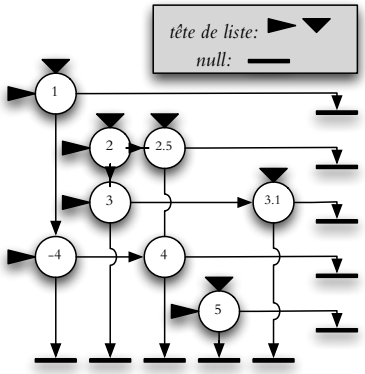
4.2 La valse (25 points)

► Démontrez (formellement) qu'un arbre binaire de recherche arbitraire à n nœuds peut être transformé en n'importe quel autre arbre binaire de recherche (avec les mêmes clés), à l'aide de $O(n)$ rotations. (***Indice*** *: commencer par montrer qu'au plus* $(n-1)$ *rotations droites suffisent à transformer l'arbre en une chaîne orientée à droite. Pour cela, le mieux est de montrer un algorithme de transformation vers la chaîne.*)

4.3 Matrices éparses (25 points)

0	1	0	0	0	0
0	0	2	2.5	0	0
0	0	3	0	0	3.1
0	-4	0	4	0	0
0	0	0	0	5	0

Une *matrice éparse* est une matrice de taille $n \times m$ dans laquelle on a $o(nm)$ cases avec une valeur non-zéro. Cet exercice vous demande d'examiner une structure de données plus économique pour des matrices éparses. L'idée est de ne stocker que les coefficients non-zéro.



Notre **structure chaînée** utilise des listes chaînées entrelacées. Un nœud u dans la structure correspond à une case $A[i,j] \neq 0$:

- ⋆ u.nextHorizontal() donne le prochain nœud (coefficient non-zéro) dans la même rangée i, où retourne null s'il n'y en a pas.
- ⋆ u.nextVertical() donne le prochain nœud (coefficient non-zéro) dans la même colonne j, où retourne null s'il n'y en a pas.

À l'addition des liens horizontaux et verticaux, un nœud contient les coordonnées i, j, et la valeur numérique du coefficient $A[i,j]$. L'avantage de la structure est son usage efficace de mémoire : l'espace est linéaire dans le nombre de valeurs non-zéro.

Typiquement, on utilise simplement un tableau 2-dimensionnel de nombre flottants. En Java, un tel tableau est alloué par `new double[n][m]`. Le produit matriciel $C = A \cdot B$ se calcule avec les tableaux en appliquant la définition $C[i,j] = \sum_k A[i,k] \cdot B[k,j]$, à l'aide de boucles imbriquées parcourant tous les i, j, k. Avec la structure chaînée, $C[i,j]$ se calcule en parcourant la rangée i de A et la colonne j de B pour trouver les termes avec $A[i,k] \cdot B[k,j] \neq 0$. ► Mon-

trez comment implanter la multiplication matricielle dans la structure chaînée. Vous devez décider les détails nécessaires de l'implantation (p.e., si on veut stocker les têtes des listes dans un tableau ou dans une liste chaînée, ou si on veut aussi stocker la queue de chaque liste). (***Indice** : inspirez-vous de la fusion de listes triées.*) ▶ Analysez (en notation asymptotique) le temps de calcul de la multiplication dans votre implantation, et comparez-le au temps de calcul de l'implantation naïve avec tableaux. Votre implantation doit multiplier des matrices éparses de taille $n \times n$ en un temps $o(n^3)$.

4.4 Partition d'un ABR (25 points)

La PARTITION(x) sur un arbre binaire de recherche T retourne une paire (T^-, T^+) d'arbres où T^- est un ABR avec les clés inférieures à x, et T^+ est un ABR avec les clés supérieures à x. Quand x est à la racine, il est facile de calculer une partition : on retourne les deux enfants de la racine. ▶ Montrez un algorithme pour PARTITION(x) où x peut être n'importe quelle clé dans l'arbre (x doit toujours être une clé, et toutes les clés sont uniques). Votre algorithme doit prendre $O(h)$ au pire sur un ABR de hauteur h : démontrez que c'est le cas.

Syntaxe : un nœud x contient les champs x.left, x.right, et x.parent pour enfants gauche et droit, et le parent. (***Indice** : notez que la structure des arbres n'est pas spécifiée — on peut les transformer comme on veut.*)

4.5 tatitatatitatitatatitatati (25 points)

Les *chaînes Fibonacci* f_n sont des mots sur l'alphabet $\{\texttt{ti}, \texttt{ta}\}$ définis par induction en utilisant la règle que la chaîne $\mathcal{R}(f)$ est calculée en remplaçant chaque `ti` par `ta` et chaque `ta` par `tati` dans la chaîne f. (On remplace tous les caractères en même temps, en parallèle.) On définit $f_0, f_1, \ldots$ par récurrence :

$$f_0 = \texttt{ti}; \qquad f_n = \mathcal{R}(f_{n-1}) \quad \{n > 0\}$$

On a donc $f_0 =$ `ti`, $f_1 =$ `ta`, $f_2 =$ `tati`, $f_3 =$ `tatita`, $f_4 =$ `tatitatati`, $f_5 =$ `tatitatatitatita`, $f_6 =$ `tatitatatitatitatatitatati`, etc. Démontrez que $f_n = f_{n-1} \cdot f_{n-2}$ pour chaque $n > 1$ (le symbol $\cdot$ dénote la concaténation), et que la longueur est exponentielle : $|f_n| = \Theta(a^n)$ avec un a quelconque. (***Indice :** utilisez une preuve par induction.*).

BONNE CHANCE !

English translation

No documentation is allowed. The examen is worth 150 points. By completing all the exercises of Question 4, and the Easter egg, you can score up to 35 bonus points.

► **Answer each question in the exam booklet.**

0 Your name (1 point)

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

1 Operations (10 points)

Describe the basic operations in the following abstract data types : stack, (FIFO) queue, pririty queue (min-heap).

2 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, 2-3-4 tree, hash table with separate chaining and a load factor of $\alpha = O(1)$. 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) Binary search trees (12 points) Binary search trees (BST) are used to implement symbol tables with a performance guarantee of $O(\log n)$ *in some sense*. ► Explain the specific types of performance guarantees offered by different BST implementations : (1) simple BST [basic operations without rotations], (2) splay tree, and (3) balanced BST. Answer, for instance that "the basic operations take $O(\log n)$ time with such-and-such implementation *in the worst case*," and briefly justify how the structure achieves that (do not go into too many details, and certainly do not give proofs). Explain the particular notion of "average" if you have to invoke it in a characterization.

3 Easter egg (10 bonus points)

If you have answered Question 2b, you have the right to answer the following question for 10 bonus points : ► what is the running time of the basic operations in the most efficient union-find data structure seen at the beginning of the class ? Make your answer as precise as possible.

4 Four out of five ain't bad (100+25 points)

You need to complete 4 out of the 5 exercises §4.1–4.5. If you complete all five, then you can have up to 25 bonus points.

4.1 Verifying the coloring (25 points)

Suppose that you have a red-black tree but you are not sure if it is colored correctly. ► Give an algorithm to verify if the nodes are properly colored : (1) every red node must have a black parent, (2) nodes with null child(ren) must be black, and (3) the same number of black nodes must be encountered on every path from the same node x to a null leaf. Syntax : every node [non-null] x contains the fields x.left, x.right, x.parent et x.color for left and right children, parent and the color. (***Hint*** *: you can verify all properties in the same tree traversal. In (3), imagine that you need to compute the black height [or "*rang*" in my course notes] of each node.*)

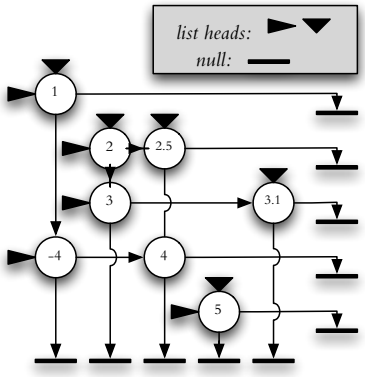
4.2 Waltz (25 points)

► Prove (formally) that a binary search tree with n nodes can be transformed into any other BST (with the same keys), using $O(n)$ rotations. (***Hint*** *: start with showing that at most* $(n-1)$ *right rotations suffice for transforming a BST into a right-leaning chain. The best is to give an explicit algorithm that does that transformation.*)

4.3 Sparse matrices (25 points)

0	1	0	0	0	0
0	0	2	2.5	0	0
0	0	3	0	0	3.1
0	-4	0	4	0	0
0	0	0	0	5	0

A *sparse matrix* is an $n \times m$ matrix in which only $o(nm)$ etries are different from zero. This exercise asks you to examine an economical data structure for sparse matrices. The basic idea is to store only the non-zero entries.



Our **chained structure** interlaces linked lists. A node u in the structure corresponds to a non-zero matrix entry $A[i, j]$:

- u.nextHorizontal() gives the next node (non-zero entry) in the same row i, or returns null when there are no more entries.
- u.nextVertical() gives the next node (non-zero entry) in the same column j, or returns null when there are no more entries. .

In addition to horizontal and vertical links, a node also contains the coordinates i, j, and the numerical value of the entry $A[i, j]$. The advantage of the chained structure is its efficient memory usage : the space is linear in the number of non-zero entries.

Typically, one uses simply a 2-dimensional array of floating-point numbers to represent matrices. In Java, such an array is allocated by `new double[n][m]`. The matrix multiplication $C = A \cdot B$ is computed in the array representation by computing the dot product $C[i, j] = \sum_k A[i, k] \cdot B[k, j]$ for each entry, using nested loops over the indices i, j, k. With the chained structure, one can calculate $C[i, j]$ by traversing row i of A and column j of B in order to identify the terms $A[i, k] \cdot B[k, j] \neq 0$. ▶ Show how to implement matrix multiplication in the chained structure. You need to decide on the implementation details, as necessary (for instance, whether you store the list heads for rows and columns in arrays or linked lists, or whether you maintain the tails of the lists). (***Hint*** *: take inspiration from the merging algorithm for sorted lists.*) ▶ Analyze (in asymptotic notation) the running time of your implementation, and compare it to the naïve, array-based implementation. Your solution should mutliply $n \times n$ matrices in $o(n^3)$ time.

4.4 Partitioning a BST (25 points)

The PARTITION(x) of a binary search tree T returns a pair of trees (T^-, T^+), where T^- is a BST with keys less than x, and T^+ is a BST with keys greater than x. When x is at the root, the partition is easy : one just takes the two child subtrees. ▶ Design an algorithm for PARTITION(x) where x can be any key in the tree (x is always present in the tree, and all the keys are different). Your algorithm should run in $O(h)$ time for a BST of height h : prove that it does.

Syntax : a node x contains the fields x.left, x.right, et x.parent for left and right children, and the parent. (***Hint*** *: notice that the structure of the trees is not specified — you can transform the BSTs at will.*)

4.5 tatitatatitatitatatitatati (25 points)

The *Fibonacci strings* f_n are words over a binary alphabet $\{\texttt{ti}, \texttt{ta}\}$ defined by induction using the rule that the string $\mathcal{R}(f)$ is obtained from f by replacing every `ti` with `ta`, and every `ta` with `tati`. The replacements are carried out at the same time, in parallel. The strings $f_0, f_1, \ldots$ are defined by the recursion :

$$f_0 = \texttt{ti}; \qquad f_n = \mathcal{R}(f_{n-1}) \quad \{n > 0\}$$

So, $f_0 = \texttt{ti}$, $f_1 = \texttt{ta}$, $f_2 = \texttt{tati}$, $f_3 = \texttt{tatita}$, $f_4 = \texttt{tatitatati}$, $f_5 = \texttt{tatitatatitatita}$, $f_6 = \texttt{tatitatatitatitatatitatati}$, etc. Prove that $f_n = f_{n-1} \cdot f_{n-2}$ for all $n > 1$ (the symbol $\cdot$ denotes concatenation), and that the length is exponential : $|f_n| = \Theta(a^n)$ with some $a > 1$. (***Hint :*** *construct an induction proof.*).