

IFT2015 H12 — Examen Final

Miklós Csűrös

18 avril 2011

Aucune documentation n'est permise. L'examen vaut 150 points. Il suffit de compléter **trois sur quatre** exercices parmi F5.a, F5.b, F6.a et F6.b. Si vous complétez tous les quatre, vous pouvez avoir jusqu'à 20 points de boni additionnels. Vous pouvez décrire vos algorithmes en pseudocode ou en Java(-esque).

► **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 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 rouge-et-noir, et un tableau de hachage avec sondage linéaire, dont la facteur de remplissage $\alpha < 0.9$. 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) 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 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.



F3 O* (15 points)

(a) O (3 points) ▶ Soit f, g des fonctions positives sur les nombres naturels ($f, g: \{0, 1, 2, \dots\} \mapsto \mathbb{R}^+$).
 ▶ Qu'est-ce qu'on veut dire exactement en écrivant $f(n) = \left(g(n)\right)^{1+o(1)}$?

(b) O* (12 points) On écrit que $f(n) = O^*(g(n))$ si et seulement si il existe une fonction polylogarithmique $\text{polylog}(n)$ telle que $f(n) = O(g(n) \cdot \text{polylog}(n))$. Ainsi, $n^2 \lg n (\lg \lg n)^3 = O^*(n^2)$. ▶ Démontrez que si $f(n) = O^*(g(n))$, alors $f(n) = \left(g(n)\right)^{1+o(1)}$.

F4 Prédécesseur-successeur (15 points)



On a un arbre binaire de recherche où les nœuds contiennent les variables $x.\text{left}$, $x.\text{right}$, $x.\text{parent}$ et $x.\text{key}$ (enfants gauche et droit, parent, et clé). L'arbre contient les clés $y_1 < y_2 < \dots < y_n$. ▶ Donnez un algorithme *récurif* pour afficher les paires (y_i, y_{i+1}) dans l'arbre (dans leur ordre naturel $i = 1, 2, \dots, n-1$). Par exemple, sur un arbre avec clés $1, 2, \dots, 6$, on doit afficher $(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)$. L'algorithme doit prendre $O(n)$ temps et n'utiliser aucune variable globale.

F5 ABR gauchiste (40 points)

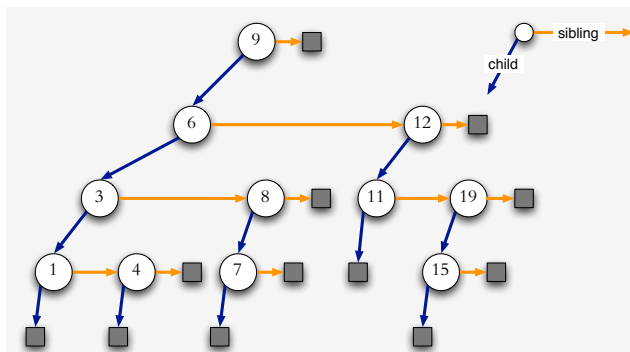


(Il suffit de compléter 3 sur 4 exercices parmi F5.a, F5.b, F6.a et F6.b.)

Un arbre binaire de recherche T est dit «gauchiste» si et seulement si à chaque nœud interne x , l'enfant gauche n'est externe que si l'enfant droit l'est aussi : si $x.\text{left} = \text{null}$ alors $x.\text{right} = \text{null}$. (Mais $x.\text{right} = \text{null}$ est permis même avec $x.\text{left} \neq \text{null}$.)

a. Gauchisation (20 points) ▶ Démontrez qu'on peut transformer n'importe quel ABR T avec n nœuds internes en un ABR gauchiste sur le même ensemble d'éléments en $O(n)$ temps. **Indice** : Donnez un algorithme qui transforme T par $O(n)$ rotations.

b. Enfant gauche-frère droit (20 points)



On peut implanter un ABR gauchiste dans la représentation «enfant gauche-frère droit» : chaque nœud interne x possède les variables $x.\text{child}$ (enfant gauche) et $x.\text{sibling}$ (frère droit), ainsi que $x.\text{parent}$ et $x.\text{key}$ (clé). L'arbre est stocké par sa racine root .

▶ Donnez un algorithme $\text{INSERT}(c)$ qui insère une nouvelle clé c dans un ABR implémenté par enfant gauche-frère droit. **Indice** : adaptez la procédure normale d'insertion dans l'ABR. Si la clé devrait être insérée comme l'enfant droit d'un nœud avec enfant gauche null, performez une rotation gauche. Faites attention à la mise à jour de parents, enfants, frères et possiblement la racine.

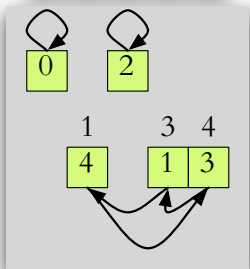
F6 Permutations (40 points)

(Il suffit de compléter 3 sur 4 exercices parmi F5.a, F5.b, F6.a et F6.b.)

On représente une permutation sur n éléments par la vecteur $\pi = \pi[0..n-1]$ telle que pour tout $j = \{0, 1, \dots, n-1\}$ il existe exactement un indice i avec $\pi[i] = j$.

a. Cycles (20 points)

i	0	1	2	3	4
π	0	4	2	1	3



Un *cycle* de la permutation est une suite maximale d'indices différents $i, \pi[i], \pi[\pi[i]], \pi[\pi[\pi[i]]], \dots$. On peut décomposer chaque permutation uniquement en un ensemble de cycles. Par exemple, la permutation $\pi = [0 \ 4 \ 2 \ 1 \ 3]$ comprend trois cycles : (0) , $(1 \ 4 \ 3)$ et (2) .

► Donnez un algorithme $\text{NUMCYCLES}(\pi)$ qui compte le nombre de cycles dans une permutation fournie comme un tableau $\pi[0..n-1]$. L'algorithme doit prendre $O(n)$ temps. (L'algorithme a le droit de changer les éléments de π si nécessaire.)

b. Permutation d'un tableau (20 points) Soit A un tableau de n éléments. La permutation de ses éléments selon π est le tableau

$$\pi \cdot A = A[\pi[0] \ \pi[1] \ \dots \ \pi[n-1]].$$

Par exemple, si $\pi = [0 \ 4 \ 2 \ 1 \ 3]$,

$$\pi \cdot [A \ B \ C \ D \ E] = [A \ E \ C \ B \ D]$$

► Donnez un algorithme $\text{PERM}(\pi, A)$ pour calculer $\pi \cdot A$ *en place* : en retour, les éléments de A sont permutés selon π . Les arguments π, A sont des tableaux de taille (connue) n . L'algorithme doit prendre $O(n)$ temps et utiliser $O(1)$ mémoire de travail à part de π et A . (L'algorithme a le droit de changer les éléments de π pendant son exécution.) **Indice** : Détectez les cycles comme en a., et décalez les éléments au long de chaque cycle $i, \pi[i], \pi[\pi[i]], \dots$.

BONNE CHANCE !

English translation

No documentation is allowed. The examen is worth 150 points. You have to complete only **three out of four** exercises among E5.a, E5.b, E6.a and E6.b. You can collect an additional 20 bonus points for completing all four. Describe your algorithms in pseudocode or Java(-esque).

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 $\alpha < 0.9$. 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 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.



E3 O* (15 points)

(a) O (3 points) ▶ Let f, g be positive functions over the natural numbers ($f, g: \{0, 1, 2, \dots\} \mapsto \mathbb{R}^+$). ▶ What does it mean exactly that $f(n) = (g(n))^{1+o(1)}$?

(b) O* (12 points) One writes $f(n) = O^*(g(n))$ if and only if there exists a polylogarithmic function $\text{polylog}(n)$ with which $f(n) = O(g(n) \cdot \text{polylog}(n))$. So, for instance, $n^2 \lg n (\lg \lg n)^3 = O^*(n^2)$. ▶ Prove that if $f(n) = O^*(g(n))$, then $f(n) = (g(n))^{1+o(1)}$.

E4 Predecessor-successor (15 points)



We have a binary search tree where the nodes are equipped with the variables $x.\text{left}$, $x.\text{right}$, $x.\text{parent}$ et $x.\text{key}$ (left and right child, parent, and key). The tree contains the keys $y_1 < y_2 < \dots < y_n$. ▶ Give a recursive algorithm to list all pairs (y_i, y_{i+1}) in the tree (in their natural order for $i = 1, 2, \dots, n - 1$). For example, in a tree with keys $1, 2, \dots, 6$, the algorithm will list $(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)$. The algorithm must take $O(n)$ time and must not use any global variable.

E5 Leftist BST (40 points)

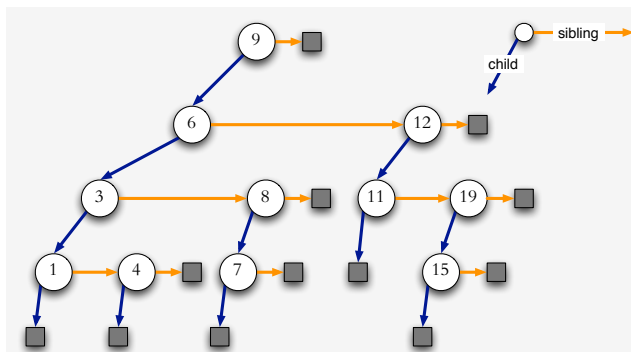


(It suffices to complete 3 out of 4 exercises among E5.a, E5.b, E6.a and E6.b.)

A binary search tree T is called *leftist* (for this exercise) if and only if at every internal node x , the left child is external only if the right child is too : if $x.\text{left} = \text{null}$ then $x.\text{right} = \text{null}$. (But $x.\text{right} = \text{null}$ is possible even with $x.\text{left} \neq \text{null}$.)

a. Leftification (20 points) ▶ Prove that it is possible to transform any BST T with n internal nodes into a leftist BST over the same set of elements in $O(n)$ time. **hint** : Give an algorithm that transforms T using $O(n)$ rotations and prove that it does exactly that.

b. Left child-right sibling (20 points)



One can implement a leftist BST in the “left child-right sibling” encoding : each internal node x contains the variables $x.\text{child}$ (left child) and $x.\text{sibling}$ (right sibling), as well as $x.\text{parent}$ and $x.\text{key}$ (key). The tree is specified by the variable root .

▶ Give an algorithm $\text{INSERT}(c)$ that inserts a new key c into a BST in the left child-right sibling encoding. **Hint** : adapt the normal procedure for insertion. If the key should be inserted as the right child of a node with no left child, then do a rotation. Pay attention to updating parents, children, singlings and possibly root.

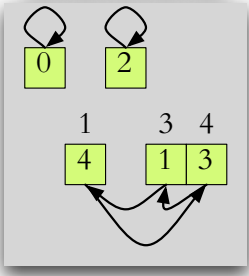
E6 Permutations (40 points)

(It suffices to complete **3 out of 4** exercises among E5.a, E5.b, E6.a and E6.b.)

A permutation of n elements is represented as the vector $\pi = \pi[0..n-1]$ such that for all $j = \{0, 1, \dots, n-1\}$, there exists exactly one index i where $\pi[i] = j$.

a. Cycles (20 points)

i	0	1	2	3	4
π	0	4	2	1	3



A *cycle* in the permutation is a maximal sequence of different indices $i, \pi[i], \pi[\pi[i]], \pi[\pi[\pi[i]]], \dots$. Every permutation has a unique cycle decomposition. For example, the permutation $\pi = [0 \ 4 \ 2 \ 1 \ 3]$ comprises three cycles : (0) , $(1 \ 4 \ 3)$ et (2) .

► Give an algorithm $\text{NUMCYCLES}(\pi)$ that counts the number of cycles in a permutation $\pi[0..n-1]$. The algorithm must take $O(n)$ time. (If necessary, you can change the values in the array π .)

b. Table permutation (20 points) Let $A[0..n-1]$ be a table of n elements. Its permutation by π is the table

$$\pi \cdot A = A[\pi[0] \ \pi[1] \ \dots \ \pi[n-1]].$$

For example, if $\pi = [0 \ 4 \ 2 \ 1 \ 3]$,

$$\pi \cdot [A \ B \ C \ D \ E] = [A \ E \ C \ B \ D]$$

► Give an algorithm $\text{PERM}(\pi, A)$ that computes $\pi \cdot A$ *in place* : after return, the elements of A are correctly permuted by π . The arguments π, A are of known size n . The algorithm must take $O(n)$ time and use $O(1)$ work space in addition to the input/output arrays π and A . (The algorithm can change the values in π .)

Indice : Detect the cycles as in a., and perform a circular shift along each cycle $i, \pi[i], \pi[\pi[i]] \dots$.

GOOD LUCK !