IFT2015 H10 — Examen Final

Miklós Csűrös

19 avril 2010

The English translation starts on page 4.

Aucune documentation n'est permise. L'examen vaut 150 points. L'exercice 7 vous permet de récupérer jusqu'à 20 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 Vrai ou faux (20 points)

►Répondez "vrai" ou "faux" aux énoncés ci-dessous — il n'est pas nécessaire de justifier vos réponses.

1. Si $f(n) = O(n^2)$ alors $f(n) = O(8^{\log_2 n})$.
2. Un tableau de n clés peut être transformé en un tas binaire dans $O(n)$ temps.
3. La recherche fructueuse dans un tableau de hachage avec une facteur de remplissage $\alpha < 1$ prend toujours $O(1)$ temps.
4. Il est possible d'implanter le type abstrait de file à priorités par une structure permettant insertion en $O(1)$.
5. Le parcours préfixe visite les éléments d'un arbre ordonné en tas selon l'ordre croissant des clés.
6. Pour un n assez grand, le tri par fusion est toujours plus rapide que le tri par insertion sur un même tableau de taille n.
7. Pour un n assez grand, le tri rapide est toujours plus rapide que le tri par tas sur un même tableau de taille n.
8. La recherche infructueuse prend $\Omega(n)$ temps sur une liste chaînée de longueur n dans le meilleur cas.
9. Dans le pire cas, insertion prend $O(\log n)$ temps dans un arbre *splay* (déployé) à n nœuds.
10. Dans le pire cas, insertion prend $O(\log n)$ dans un tas binaire de taille n.

2 Petit o (10 points)

Soit $f\colon \{0, 1, 2, \dots\} \mapsto [0, \infty)$. ►Donnez une définition très précise de l'ensemble de fonctions dénoté par $o(f)$.

3 Table de symboles $3 \times 3 \times 3$ (39 points)

(a) Implémentations (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 adressage ouvert (hachage double), dont la facteur de remplissage $\alpha < 0.75$. 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. (Donc, cela fait 27 résultats de complexité temporelle au total.) Il ne faut pas justifier vos réponses (sauf trois, voir (b) ci-dessous).

(b) Justifications (12 points) ▶Élaborez *trois* de vos réponses (votre choix lesquels) en **(a)** : expliquez comment la structure assure un tel temps de calcul.

4 Rangs dynamiques (30 points)

On veut modifier une structure d'arbre rouge-et-noir pour retrouver le *rang* des éléments. Ici, le $i$ème rang d'un ensemble à n éléments, où $i \in \{1, 2, \ldots, n\}$ est simplement l'élément ayant la $i$ème plus petite clé (les clés sont uniques). Dans la structure modifiée, chaque nœud possède un champs taille à l'addition aux champs habituels cle, parent, gauche, droit et couleur(x). Pour chaque nœud x, taille(x) est le nombre de nœuds non-null dans le sous-arbre enraciné à x. L'utilité de taille est illustrée par la procédure récursive suivante qui permet de retrouver le $i$ème élément.

```
    RÉCUPERER-RANG(x, i)                          // x est un nœud, i est le rang recherché
R1  si gauche(x) = null alors r ← 1
R2  sinon r ← taille(gauche(x)) + 1
R3  si i = r alors retourner x                    // r est le rang de x
R4  si i < r alors retourner RÉCUPERER-RANG(gauche(x), i)
R5  sinon retourner RÉCUPERER-RANG(droit(x), i − r)
```

a. Efficacité (10 points) ▶Quel est le temps de calcul de RÉCUPERER-RANG ? Donnez votre réponse, avec justification, en notation asymptotique, en adressant le pire, le meilleur, et le moyen cas.

b. Déterminer le rang (10 points) ▶Donnez un algorithme qui détermine le rang d'un nœud x, et analysez son temps de calcul.

c. Mise à jour dynamique (10 points) Lors de l'insertion ou suppression d'un élément, il faut mettre à jour les champs taille sur le chemin entre la racine et le nœud affecté. ▶Expliquez comment et à quels nœuds les champs taille doivent être modifiés lors d'une rotation gauche ou droite. ▶Quel est le temps de calcul pour l'insértion et la suppression dans la structure modifiée ?

5 Éléments distincts (30 points)

On veux compter le nombre d'éléments distincts dans un tableau $A[1..n]$ de nombres entiers. On n'a pas le droit de changer l'ordre des éléments dans le tableau.

a. Hachage (15 points) ►Montrez comment on peut compter en temps $O(n)$ au moyen, à l'aide d'un espace mémoire additionnel pour $2n + O(1)$ nombres entiers.

b. Tri (15 points) ►Montrez comment on peut compter en temps $O(n \log n)$ au pire cas, à l'aide d'un espace mémoire additionnel pour $n + O(1)$ nombres entiers.

6 Structure couteau suisse (20 points)

Soit $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ un ensemble de n points, avec x_i et y_j tous différents. ►Démontrez qu'il existe exactement un arbre binaire pour ces points qui est en même temps un arbre binaire de recherche selon les x_i, et est ordonné en tas selon les y_i.

7 Dictionnaire avec une cache (20 points de boni)

On veut implanter un dictionnaire en utilisant un arbre rouge-et-noir T et une cache. La cache est une liste triée d'éléments récemment insérés. Lors d'une insertion, on place l'élément sur cette liste. Si la taille de la liste atteint un seuil prédeterminé m, il faut enlever les éléments sur la liste un à un, et les insérer dans l'arbre.

```
   insert(x)
I1 cache.insert(x)
I2 si la taille de cache est m alors
I3     insérer tous les éléments de cache dans T
I4     vider cache ← ∅
```

►Donnez le code pour l'opération search(k) qui recherche une clé k, et retourne un élément avec cette clé, ou bien null si la clé n'existe pas dans le dictionnaire. ►Analysez le coût amorti des opérations insert et search (la suppression n'est pas permise). ►Comment doit-in choisir m et quel est l'avantage de cette structure combinée ?

BONNE CHANCE !

English translation

No documentation is allowed. The examen is worth 150 points. You can get up to 20 bonus points for solving Problem 7.

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 True or false (20 points)

►Answer "true" or "false" to the statements below — you do not need to justify your answers.

1. If $f(n) = O(n^2)$ ythen $f(n) = O(8^{\log_2 n})$.
2. A table with n keys can be transformed into a binary heap in $O(n)$ time.
3. A successful search always takes $O(1)$ time in a hashtable with load factor $\alpha < 1$.
4. It is possible to implement the dictionary abstract data type in such a way that insertion takes $O(1)$ time.
5. The prefix traversal of a tree in (min-)heap order visits the elements in an increasing order of keys.
6. For n large enough, mergesort of a table of length n is always faster than insertion sort of the same table.
7. For n large enough, quicksort of a table of length n is always faster than heapsort of the same table.
8. Unsuccessful search on a linked list of length n takes $\Omega(n)$ in the best case.
9. In the worst case, insertion takes $O(\log n)$ time in a splay tree of n nodes.
10. In the worst case, insertion takes $O(\log n)$ time in a binary heap of size n.

2 Little-o (10 points)

Let $f : \{0, 1, 2, \dots\} \mapsto [0, \infty)$. ►Give a very precise definition of the function set denoted by $o(f)$.

3 Symbol table $3 \times 3 \times 3$ (39 points)

(a) Implementations (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 time 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 open adressing (double hashing) with $\alpha < 0.75$ load factor. 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 (except three of them, see (b) below).

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

4 Dynamic ranks (30 points)

We want to modify a red-black tree structure in order to work with the *ranks* of the elements. For a set of n elements with distinct keys, rank $i \in \{1, 2, \ldots, n\}$ means simply the i-th smallest element. In the modified data structure, each node x is equipped with a size field that stores the number of non-null nodes in the subtree rooted at x, in addition to the usual fields key, parent, left, right et color(x). The recursive procedure below shows the utility of size in finding the i-th smallest element.

```
    FIND-RANK(x, i)                                   // x is a node, i is the rank searched for
R1  if left(x) = null then r ← 1
R2  else r ← size(left(x)) + 1
R3  if i = r then return x                                          // r is the root's rank
R4  if i < r then return FIND-RANK(left(x), i)
R5  else return FIND-RANK(right(x), i − r)
```

a. Efficiency (10 points) ►What is the running time of FIND-RANK ? Give your answer, with justification, in asymptotic notation, considering the best, worst and average cases.

b. Determining the rank (10 points) ►Give an algorithm that determines the rank for a node x, and analyze the running time.

c. Dynamic updates (10 points) On insertion and deletion, one needs to update the size fields at nodes along the path between the root and the affected node. ►Explain where and how size needs to be modified after a rotation (left or right). ►What is the running time for insertion and deletion in the modified structure ?

5 Distinct elements (30 points)

We would like to count the number of distinct elements in a table $A[1..n]$ of integers. The elements' order must not be changed.

a. Hashing (15 points) ►Show how the counting can be done in $O(n)$ time on average by using additional memory for $2n + O(1)$ integers.

b. Sorting (15 points) ►Show how the counting can be done in $O(n \log n)$ worst-case time by using additional memory for $n + O(1)$ integers.

6 Swiss Army structure (20 points)

Let $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a set of n points, with all different x_i and y_j. ►Prove that there exists exactly one binary tree over the points that is at the same time a binary search tree for the x_i, and has a heap order for the y_i.

7 Dictionary with a cache (20 points de boni)

We would like to implement a dictionary by using a red-black tree T and a cache. The cache is a sorted list of recently inserted elements. upon insertion into the dictionary. the new element is placed in the cache list first. When the cache size attains a predetermined size m, all elements are removed from the cache, and properly inserted into the tree.

```
   insert(x)
I1 cache.insert(x)
I2 if cache contains m elements then
I3     insert every element of cache into T
I4     clear cache ← ∅
```

►Give the algorithm for the search(k) operation that searches for a key k, and returns an element with key k, or else null if the key does not exist in the dictionary. ►Analyze the amortized cost of the insert and search operations (deletion is not allowed). ►How should one choose m and what is the advantage of this data structure ?

GOOD LUCK !