IFT2015 H11 — Examen Final

Miklós Csűrös

18 avril 2011

Aucune documentation n'est permise. L'examen vaut 150 points. Vous pouvez avoir 20 points de boni additionnels. Vous pouvez décrire vos algorithmes en pseudocode ou en Java.

► **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)

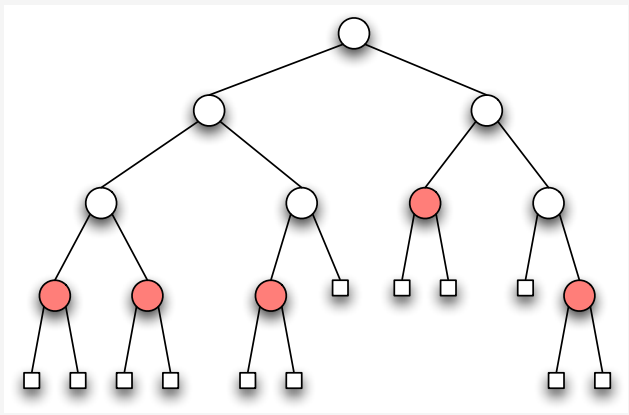
(a) 3 structures, 3 opérations, 3 performances (27 points) On a vu plusieurs structures de données qui peuvent servir à implémenter le type abstrait de la table de symboles. L'efficacité des implémentations n'est pas la même : ici vous devez comparer le temps de calcul pour trois opérations fondamentales : 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 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) 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 Les feuilles d'un arbre (20+20 points)



Une *feuille* dans un arbre binaire est un nœud interne avec exactement deux enfants externes.

a. Parcours (20 points) ► Donnez un algorithme récursif pour calculer le nombre de feuilles dans un arbre binaire.

La variable r stocke la racine de l'arbre ; chaque nœud interne N possède les variables N.parent, N.left, N.right, pour le parent, et les enfants gauche et droit. Les nœuds externes sont null.

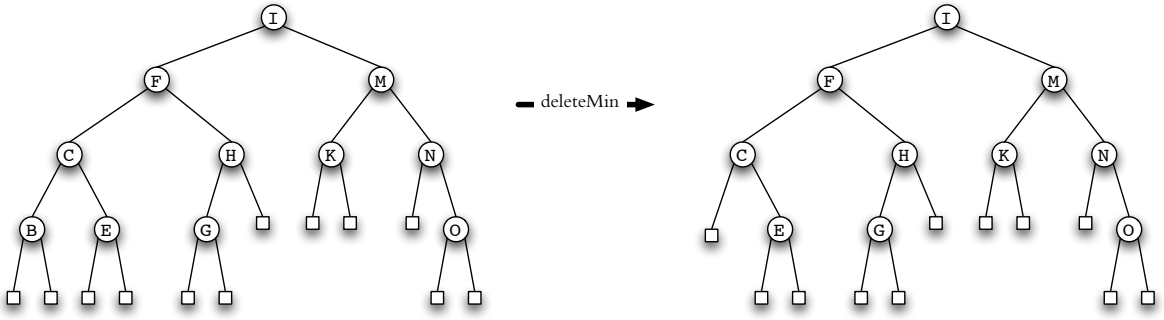
b. Moyenne (20 points boni) ► Calculez le nombre moyen de feuilles dans un arbre binaire de recherche aléatoire, résultant de l'insertion successive des clés $1, 2, \ldots n$ selon une permutation choisie uniformement au hasard. **Indice :** calculez la probabilité p_i que clé i est à une feuille : le nombre de feuilles est $\sum_{i=1}^{n} p_i$ en espérance. Le nœud avec la clé $1 < i < n$ devient une feuille si et seulement si $(i-1)$ et $(i+1)$ sont insérés plus tôt que lui.

F4 File de priorité avec ABR (25 points)

On veut implanter une file de priorité à l'aide d'un arbre binaire de recherche (ABR). ► Montrez comment implanter l'opération deleteMin() (qui supprime et retourne la clé minimale) dans l'ABR.

► Arguez que votre algorithme est correct, et montrez tous les détails de l'implantation. (En particulier, vous n'avez pas le droit de faire appel à d'autres opérations comme delete sans les définir.) Faites attention au cas où le minimum se trouve à la racine. **Indice :** recherchez le nœud avec clé minimale, et supprimez-le — la suppression n'est pas trop compliquée . . . [pourquoi ?]

La variable r stocke la racine de l'arbre ; chaque nœud interne N possède les variables N.left, N.right, N.parent, et N.key pour enfants gauche et droit, le parent, et la clé. Les nœuds externes sont null.



F5 Suppression avec adressage ouvert (45 points)

Les algorithmes suivants implantent des opérations principales dans un tableau de hachage $H[0..M-1]$, à l'aide d'une fonction de hachage $h\colon \mathcal{X} \mapsto \{0, 1, \ldots, M-1\}$.

```
insert(x) // insertion de la clé x
I1  i ← h(x)
I2  tandis que H[i] ≠ null faire
I3      i ← (i + 1) mod M
I4  H[i] ← x
```

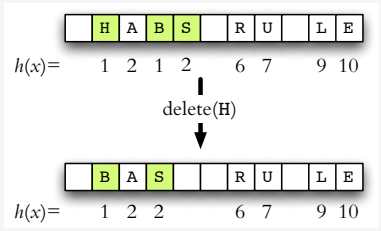
```
search(x) // recherche de la clé x
S1  i ← h(x)
S2  tandis que H[i] ≠ null et H[i] ≠ x faire
S3      i ← (i + 1) mod M
S4  retourner H[i]
```

Dans les problèmes suivants, vous pouvez assumer que la facteur de remplissage reste toujours $\alpha = n/M < 2/3$.

a. Qu'est-ce que c'est ? (2 points) ▶ Identifiez le type abstrait implanté par la structure. ▶ Identifiez la politique d'adressage ouvert employée.

b. Suppression paresseuse (18 points) ▶ Donnez l'implantation de l'opération delete(x) qui supprime la clé x dans le tableau en utilisant l'approche paresseuse à l'aide d'une sentinelle DELETED. ▶ Montrez comment modifier le code de search et insert pour accommoder cette solution.

c. Suppression impatiente (25 points) On veut une solution altérnative basée sur une *approche impatiente*, sans changer les implantations de search et d'insert.



Dans une approche impatiente, il faut remplir le «trou» créé lors de suppression. Pour cela, il faut décaler quelques éléments dans le tableau.

▶ Montrez l'implantation de l'opération delete(x) dans l'approche impatiente. **Indice :** utilisez une boucle pour examiner les éléments à $j = i+1, i+2, \ldots, M-1, 0, 1, \ldots$ après la case $H[i] = x$, d'où x est enlevé. Déterminez la condition sous laquelle $H[j]$ peut être déplacé à $H[i]$, et continuez avec la recherche d'un remplacement pour $H[j]$ (si nécessaire).

BONNE CHANCE !

English translation

No documentation is allowed. The examen is worth 150 points, and you can collect an additonal 20 bonus points. Describe your algorithms in pseudocode or Java.

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)

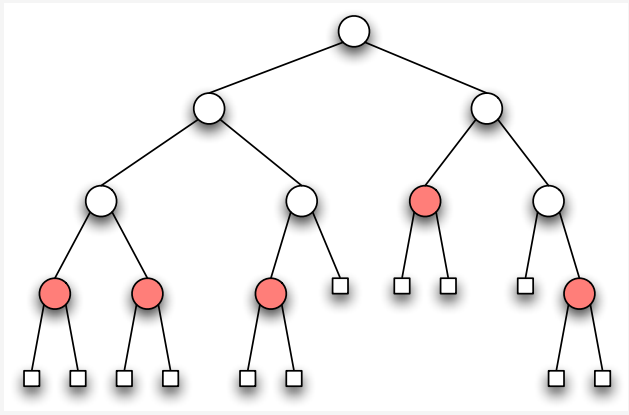
(a) 3 structures, 3 operations, 3 performance guarantees (27 points) We have seen a number of data structures that can be used to implement the abstract data type of symbol table. Not all implementations have the same performance. You need to compare the running times for three fundamental operations : insertion, successful search and unsuccessful search. ► Give the running times for the tree operations in the three following data structures : unsorted linked list, 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) 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 Leaves of a tree (20+20 points)



A *leaf* in a binary tree is an internal node with exactly two external children.

a. Traversal (20 points) ► Give a recursive algorithm that computes the number of leaves in a binary tree.

Variable r stores the tree root ; every node N has the fields N.parent, N.left, and N.right. External nodes are null.

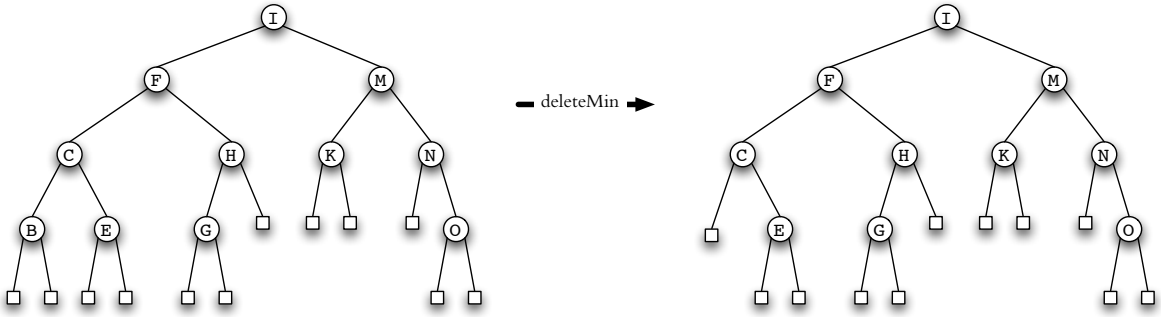
b. Average (20 bonus points) ► Compute the average number of leaves in a random binary search tree, constructed by inserting the keys $1, 2, \ldots, n$ in a random uniform permutation. **Hint :** compute the probability p_i that key i is at a leaf : the number of leaves is $\sum_{i=1}^{n} p_i$ in expectation. Key $1 < i < n$ is stored at a leaf if and only if $(i-1)$ and $(i+1)$ are inserted earlier.

E4 Priority queue with a BST (25 points)

One can implement a priority queue using a binary search tree (BST). ► Show how to implement the deleteMin() operation in a BST, which deletes and returns the minimal key.

Argue the correctness of your algorithm, and show all the implementation details. (In particular, you must not rely on other operations such as delete without showing how to implement them.) Pay attention to the case when the minimum is at the root. **Hint :** search for the node with minimal key, and delete it — deletion is not too complicated . . . [why ?]

Variable r stores the tree root ; every internal node N has the fields N.parent, N.left, N.right, and N.key. External nodes are null.



E5 Deletion with open addressing (45 points)

The following algorithms implement the principal operations in a hash table $H[0..M-1]$, using a hash function $h\colon \mathcal{X} \mapsto \{0, 1, \ldots, M-1\}$.

```
    insert(x) // insertion of key x
I1  i ← h(x)
I2  while H[i] ≠ null do
I3      i ← (i + 1) mod M
I4  H[i] ← x
```

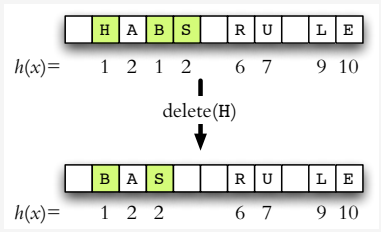
```
    search(x) // search for key x
S1  i ← h(x)
S2  while H[i] ≠ null et H[i] ≠ x do
S3      i ← (i + 1) mod M
S4  return H[i]
```

In the following problems, you can assume that the load factor always stays $\alpha = n/M < 2/3$.

a. What is this ? (2 points) ▶ Identify the abstract data type implemented by this structure. ▶ Identify the employed policy of open addressing.

b. Lazy deletion (18 points) ▶ Give an implementation for the delete(x) operation that deletes key x from the table, using a lazy approach with a sentinel element DELETED. ▶ Show also how to modify search and insert to accommodate this solution.

c. Eager deletion (25 points) In an alternative implementation, one can use on an eager approach, and there is no need to modify search and insert.



In eager deletion, one needs to fill the "hole" in place of the deleted element by shifting some subsequent elements in the table.

▶ Show an eager implementation of delete(x). **Hint :** use a loop to examine the entries at $j = i+1, i+2, \ldots, M-1, 0, 1, \ldots$ after the entry $H[i] = x$ that needs to be deleted. Determine the condition for moving $H[j]$ to $H[i]$, and continue with finding a replacement for $H[j]$ (if necessary).

GOOD LUCK !