IFT2016 :: A16 Miklós Csűrös 19 décembre 2016

Examen final[1]

[1] The English translation follows.

L'EXAMEN vaut 150 points, et vous pouvez avoir jusqu'à 30 points de boni additionnels.

- ⋆ Aucune documentation n'est permise.
- ⋆ Décrivez vos algorithmes en pseudocode ou en Java(-esque).
- ⋆ Vous avez le droit de répondre en français ou en anglais.
- ⋆ Répondez à toutes les questions dans les cahiers d'examen.

Exo	points	boni
F0	1	
F1	$3 \times 3 \times 3 + 3 \times 4 = 39$	
F2	$5 \times 3 = 15$	
F3	$30 + 20 = 50$	
F4	$15 + 30 = 45$	
F5		30
$\sum$	150	30

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 : (1) insertion, (2) recherche fructueuse, et (3) recherche infructueuse.

▶ Donnez le temps de calcul des trois opérations avec les trois structures de données suivantes : (A) tableau non-trié, (B) arbre rouge-et-noir, et (C) tableau de hachage avec chaînage séparée dont la facteur de remplissage $\alpha < 3$. Spécifiez le temps de calcul comme une fonction du nombre des éléments n. en utilisant la notation asymptotique (en omettant les facteurs constantes), dans trois cas : (i) le pire cas, (ii) le meilleur cas, et (iii) 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.

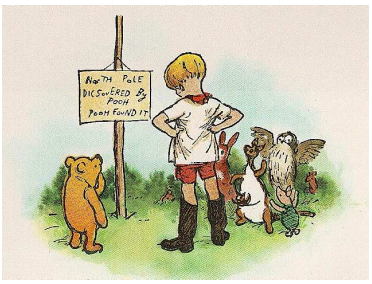
F2 Tris (15 points)

▶ Pour chacun des algorithmes de tri suivants, donnez le temps de calcul en moyen et pire cas, ainsi que l'espace de travail[2] 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, omettant les facteurs constantes. Il n'est pas nécessaire de justifier vos réponses.

[2] «espace de travail» : mémoire utilisé sans compter l'entrée de n éléments

tri	temps		mémoire
	moyen	pire	
rapide			
tas			
fusion			
insertion			
sélection			

F3 Recherche fructueuse (50 points)



Supposons qu'on quête un tableau de hachage $T[0..M-1]$ rempli en employant sondage linéaire et une fonction de hachage h :

```
Algo search(x)                                   // recherche de clé x
S1  i ← h(x)                                     // valeur de hachage i ∈ {0,1,...,M−1}
S2  while T[i] ≠ null && T[i].key ≠ x do i ← (i+1) mod M
S3  return T[i]                                  // null si clé n'existe pas
```

Dans cet exercice, on veut étudier le *coût moyen de recherche fructueuse*, ou le nombre de cases examinées ($T[i]$ dans la Ligne S2) en moyenne quand on cherche une clé existante. Dans d'autres mots, si $c(i)$ dénote le nombre d'itérations quand on cherche la clé dans cellule i, on définit

$$\bar{c} = \frac{1}{n} \sum_{i:\, T[i]\neq\text{null}} c(i) \qquad \text{(F1)}$$

comme le coût moyen (n est le nombre des éléments). Fig. 1 montre un exemple.

x	$h(x)$	x	$h(x)$	x	$h(x)$
A	10	L	10	G	0
O	5	R	0	I	4
T	7	H	2	M	1

	0	1	2	3	4	5	6	7	8	9	10
T	L	G	R	H	I	O	M	T			A
c	2	2	3	2	1	1	6	1			1

FIG. 1: Tableau de hachage sur 9 clés $x =$ A,L,...,M, construit avec la fonction de hachage h montrée en haut. Sous chaque case avec clé x, c est le nombre de cellules examinées dans la recherche pour x. Le coût moyen est alors $\bar{c} = (2+2+3+2+1+1+6+1+1)/9 = 2\frac{1}{9}$..

a. Démarrage à froid (30 points) ▶ Donnez un algorithme pour calculer $\bar{c}$ sur un tableau T fourni à l'entrée. (L'algorithme connaît la fonction de hachage $h(x)$, et il n'a pas le droit de changer T.) L'algorithme doit prendre $O(M)$ temps, et utiliser $O(1)$ espace de travail.

b. Entretien (20 points) On veut récuperer $\bar{c}$ à tout temps pendant la vie du tableau.

▶ Expliquez comment amender[3] la structure, et modifier les opérations d'insertion et de suppression pour supporter le calcul de $\bar{c}$ en $O(1)$ à tout temps. Le temps d'exécution des autres opérations ne peut être affecté que par une facteur constante. Vous avez le droit d'introduire $O(1)$ nouvelles variables.

[3] ***Indice:*** introduire des variables auxiliaires suggérées par Equation (F1), et montrer comment les mettre à jour lors d'une insertion ou suppression.

F4 Arbre colorié (45 points)

Dans cet exercice, on travaille avec un arbre binaire de recherche colorié.

a. Arbre rouge-et-noir (15 points) Dans un arbre rouge-et-noir classique, les nœuds externes sont noirs, et les nœuds internes sont soit noir, soit rouge.

▶ Récitez les règles standardes de coloriage pour les nœuds internes.

b. Arbre ougandais (30 points) Dans un arbre *ougandais*, les nœuds externes sont noirs, et les nœuds internes sont soit noir, soit rouge, soit jaune avec les contraintes suivantes :

(i) le parent d'un nœud jaune doit être rouge, le parent d'un nœud rouge doit être noir, et le parent d'un nœud noir peut être de toutes les trois couleurs.

(ii) pour tout nœud interne x, tout chemin descendant de x à un nœud externe contient le même nombre de nœuds noirs.

▶ Démontrez[4] que la hauteur $h(n)$ d'un arbre ougandais avec n nœuds internes est bornée comme

$$\lg(n+1) \leq h(n) \leq 3\lg(n+1). \tag{F2}$$

FIG. 2: Le drapeau de l'Ouganda se compose de six bandes horizontales égales alternées noire (tout en haut), jaune, rouge, noire, jaune et rouge (tout en bas), et est orné d'un cercle blanc portant une grue royale [wikipédia]

[4] ***Indice:*** définir la hauteur noire (ou *rang*) suggérée par Propriété (ii), et compléter la preuve en deux étapes : borne sur la hauteur en fonction de la hauteur noire, et borne sur le nombre de nœuds en fonction de la hauteur noire.

F5 Calcul de somme (30 points boni)

Supposons qu'on veut calculer la somme de n nombres flottants positifs. Cela se traduit au calcul de $(n-1)$ additions. À cause de la précision finie, l'ordre des additions détermine l'erreur numérique du résultat. Par exemple, avec 3 chiffres de précision, $(15.0 + 0.04) + 0.04 = 15.0$ mais $(0.04 + 0.04) + 15.0 = 15.1$, et ce dernier est une meilleure approximation de la valeur exacte. Le mieux est de calculer les sommes partielles à partir des plus petites vers les plus grandes. L'algorithme esquissé ici minimise l'erreur numérique de sommation des éléments d'un ensemble $\mathcal{X}$, à l'aide de la fonction $\mathsf{min2}$ qui retourne les deux éléments plus petits :

```
    Algo ssum(X)                          // somme des éléments de X
Z1  while |X| > 1                         // tandis qu'il en reste au moins 2
Z2      (x,y) ← min2(X)                   // les deux éléments plus petits x,y ∈ X
Z3      X ← X \ {x,y} ∪ {x+y}             // remplacer par leur somme x+y
Z4  retourner le seul élément de X
```

▶ Développez[5] un algorithme $\mathsf{asum}(T)$ pour sommer les éléments du tableau $T[0..n-1]$ de nombres flottants en $O(n \log n)$ temps, avec une erreur numérique minimale. L'algorithme a le droit de détruire T, mais peut utiliser seulement $O(1)$ espace de travail additionnel. Montrez tous les détails de la solution.

[5] ***Indice:*** employer une structure de données adéquate.

Final examination

THE EXAMEN is worth 150 points, and you can collect an additional 30 bonus points.

- ⋆ No documentation is allowed.
- ⋆ Write your algorithms in pseudocode or Java.
- ⋆ You may write your answers in English or in French.
- ⋆ Answer each question in the exam booklet.

Exercise	points	bonus
E0	1	
E1	$3 \times 3 \times 3 + 3 \times 4 = 39$	
E2	$5 \times 3 = 15$	
E3	$30 + 20 = 50$	
E4	$15 + 30 = 45$	
E5		30
$\sum$	150	30

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 : (1) insertion, (2) successful search and (3) unsuccessful search. ▶ Give the running times for the three operations in the following three data structures : (A) unsorted array, (B) red-black tree, and (C) hash table with separate chaining and a load factor of $\alpha < 3$. Give the (i) worst-case, (ii) best-case and (iii) average-case running times in asymptotic notation (omitting constant factors) 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 (15 points)

▶ Give the running time in the worst and average case, as well as the 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, omitting constant factors. You do not need to justify your answers.

method	time		memory
	average	worst	
quicksort			
heapsort			
mergesort			
insertion sort			
selection sort			

E3 Successful search (50 points)

Suppose that we are querying a hashtable $T[0..M-1]$ that is populated using a hash function h and linear probing :

```
Algo search(x)                                   // search for key x
S1  i ← h(x)                                     // hash value i ∈ {0,1,...,M−1}
S2  while T[i] ≠ null && T[i].key ≠ x do i ← (i+1) mod M
S3  return T[i]                                  // null if x does not exist
```

In this exercise, we study the *average cost of successful search*, or the number of cells examined ($T[i]$ in Line S2) on average when searching for an existing key. In other words, if $c(i)$ denotes the number of iterations when one searches with the key of cell i, then the average cost is defined as (with n as the number of elements)

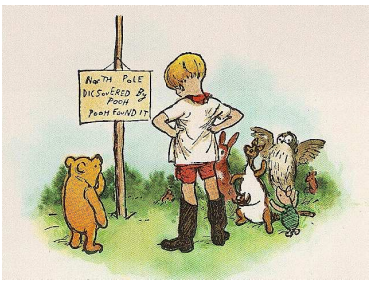
$$\bar{c} = \frac{1}{n} \sum_{i:\, T[i] \neq \text{null}} c(i). \qquad \text{(E1)}$$

See Fig. 3 for an exemple.

a. Cold start (30 points) ▶ Give an algorithm that computes $\bar{c}$ over a table T given as input. (The algorithm knows the hash function $h(x)$, and must not change T.) The algorithm must take $O(M)$ time, and use $O(1)$ work space.

b. Maintenance (20 points) We would like to retrieve $\bar{c}$ at ease, any time.

▶ Explain how you would amend[6] the basic data structure, and modify the insertion and deletion operations, in order to support computing $\bar{c}$ in $O(1)$ time always. (The running time of the operations should not be affected by more than a constant factor.) Your solution may introduce only $O(1)$ new variables.



x	$h(x)$	x	$h(x)$	x	$h(x)$
A	10	L	10	G	0
O	5	R	0	I	4
T	7	H	2	M	1

	0	1	2	3	4	5	6	7	8	9	10
T	L	G	R	H	I	O	M	T			A
c	2	2	3	2	1	1	6	1			1

FIG. 3: Hashtable with 9 keys x = A, L, ..., M, built using the hash function h shown on top. Under every cell with a key x, c is the number of cells acessed when searching for x. The average cost is thus $\bar{c} = (2+2+3+2+1+1+6+1+1)/9 = 2\frac{1}{9}$.

[6] ***Hint:*** introduce auxiliary variables suggested by Equation (E1), and show how they are updated on insertion and deletion.

E4 Colored tree (45 points)

In this exercise, we work with colored binary search trees.

a. Red-black tree (15 points) In a classic red-black tree, the external nodes are black, and the internal nodes are either red or black.

▶ Recite the standard rules for coloring internal nodes.

b. Ugandan tree (30 points) In a *Ugandan* tree, the external nodes are black, and the internal nodes are either black, or red, or yellow, complying with the following rules :

(i) a yellow node's parent is red, a red node's parent is black, and a black node's parent can be of any color.

(ii) for every internal node x, every path from x to an external node in its subtree contains the same number of black nodes.

▶ Show[7] that the height $h(n)$ of a Ugandan tree with n internal nodes is bounded as

$$\lg(n+1) \le h(n) \le 3\lg(n+1). \tag{E2}$$

FIG. 4: The flag of Uganda onsists of six equal horizontal bands of black (top), yellow, red, black, yellow, and red (bottom) ; a white disc is superimposed at the centre and depicts the national symbol, a grey crowned crane [wikipedia]

[7] ***Hint:*** define black height (or *rank*) as suggested by Property (ii), and complete the proof in two steps : bounds on the height in function of the black height, and bounds on tree size in function of black height.

E5 Summing up (30 bonus points)

Suppose that we want to sum n positive floating-point numbers, which translates into $(n-1)$ pairwise additions. As the operations are carried out at finite precision, the order of the additions influences the numerical error. For instance, with 3-digit precision, $(15.0 + 0.04) + 0.04 = 15.0$ but $(0.04 + 0.04) + 15.0 = 15.1$, and it is this latter that approximates the exact value better. The best is to compute the partial sums from the smaller to the larger ones. The algorithm sketched below minimizes the numerical error in computing the sum of all elements in a set $\mathcal{X}$, calling on the function min2 that returns the two smallest elements :

```
    Algo ssum(X)                                  // sum of elements of X
Z1  while |X| > 1                                 // while there are at least 2
Z2      (x,y) ← min2(X)                           // the two smallest elements x,y ∈ X
Z3      X ← X \ {x,y} ∪ {x+y}                     // replace by the sum x+y
Z4  return the single element in X
```

▶ Develop[8] an algorithm, called asum(T), which calculates the sum of the elements in the table $T[0..n-1]$ with minimal numerical error, in $O(n \log n)$ time. The algorithm is allowed to destroy T but can use only $O(1)$ additional work space. Show all the details of your solution.

[8] ***Hint:*** use an adequate data structure.