22 janvier 2020

Programmation avancée

Examen écrit, session 1 (210 minutes)

- Aides électroniques, documents, livres et notes interdits.
- Répondez à chaque question sur une feuille **séparée**, sur laquelle figurent vos noms, prénoms et matricules.
- Merci d'indiquer clairement le numéro de question avant votre réponse.
- Soyez bref et concis, mais précis.
- Sauf mention explicite, toutes les complexités sont à décrire par rapport au temps d'exécution des opérations concernées. Soyez toujours le plus précis possible dans le choix de votre notation (Ω, O ou Θ).
- Donnez une spécification claire et concise pour chaque fonction auxiliaire que vous définissez.

1. Questions courtes diverses

Aucune des questions ci-dessous ne nécessite de réponse de plus de trois lignes.

1. Soit une table hash de taille 10 (indicée de 0 à 9) avec la fonction hash $h(k) = k \bmod 10$, adressage ouvert, et sondage linéaire. Donner la table hash après ajout de 24, 54, 39, 49, 77, 88, 47, en supposant qu'il n'y a pas de rehashage. La table après insertion des deux premiers éléments est ci-dessous.

				24	54				

   Solution: La table après insertion des 7 éléments est:

49	47			24	54		77	88	39

2. Pour l'implémentation d'un compilateur, un étudiant écrit, pour les identifiants de variables, une fonction hash qui étend la chaîne de caractères représentant l'identifiant à un multiple de 4 bytes en ajoutant des bytes 0 si nécessaire, coupe ensuite la chaîne en mots de 4 bytes (le premier caractère étant le byte de poids fort), réalise un "ou" exclusif de tous ces mots, la clé hash retournée étant le résultat modulo la taille de la table, taille initialisée à 2048 entrées. Pourquoi est-ce une mauvaise idée? Indice: beaucoup de variables locales ont des noms courts `i`, `j`, `k`, `Ai`,...

   Solution: Toutes les chaînes de caractères de 2 bytes auront 0 comme clé hash.

3. Est-il possible de développer un algorithme pour transformer un tas max en arbre binaire de recherche qui serait $\Theta(n)$ dans le pire cas?

   Solution: Non. La mise en tas max est linéaire (voir cours), et l'obtention d'un tableau trié contenant toutes les étiquettes d'un arbre binaire de recherche est aussi linéaire. Si un tel algorithme existait, le problème du tri serait $O(n)$.

4. Donner la complexité la plus précise possible pour chaque extrait de code C suivant:

   A
   ```
   int count = 0;
   for (int i = 0; i < N; i++)
     for (int j = 0; j < i; j++)
       count++;
   ```

   B
   ```
   int count = 0;
   for (int i = 1; i + 1 < N; i++)
     for (int j = i - 1; j < i + 1; j++)
       count++;
   ```

   C
   ```
   int count = 0;
   for (int i = N; i > 0; i /= 2)
     for (int j = 0; j < i; j++)
       count++;
   ```

   D
   ```
   int count = 0;
   for (int i = N; i > 0; i--)
     for (int j = 0; j < N; j++)
       for (int k = 0; k < N; k++)
         count++;
   ```

   Solution:

   A. $\sum_{i=0}^{N} i \in \Theta(N^2)$

   B. $\sum_{i=0}^{N} 3 \in \Theta(N)$

C. $\sum_{i=1}^{\log_2 N} 2^i = 2^{\log N} - 2 \in \Theta(N)$

D. $\sum_{i=1}^{N} \sum_{j=0}^{N-1} N \in \Theta(N^3)$

5. Écrire un invariant de boucle pour la boucle extérieure (ligne 1) de l'algorithme suivant. Pour rappel, un invariant de boucle est une formule préservée par chaque itération de la boucle, et qui permet de prouver que la propriété voulue à la fin de la boucle est vérifiée (ici, que le tableau est trié).

```
MySort(A)
1  for n = A.length to 1
2      for i = 1 to A.length − 1
3          if A[i] > A[i + 1]
4              swap(A[i], A[i + 1])
```

Solution: Les propriétés suivantes sont préservées par la boucle

- Le tableau $A[n+1..A.length]$ est trié;
- Tous les éléments de $A[n+1..A.length]$ sont supérieurs ou égaux aux éléments de $A[1..n]$;
- Facultatif: A est une permutation du tableau original.

2. Analyse d'algorithmes

Soit la fonction suivante:

```
Avg(A, p, q)
 1  if p > q
 2      return 0
 3  n = q − p + 1
 4  d = ⌊n/3⌋
 5  s = p + d
 6  r = q − d
 7  x_l = Avg(A, p, s − 1)
 8  x_r = Avg(A, r + 1, q)
 9  x_c = 0
10  for i = s to r
11      x_c = x_c + A[i]
12  return (d/n) x_l + (1/n) x_c + (d/n) x_r
```

qui renvoie la moyenne des éléments du (sous-)tableau de nombres $A[p\,..\,q]$. Appelons n la taille de A et $T(n)$ la complexité en temps de la fonction par rapport à n.

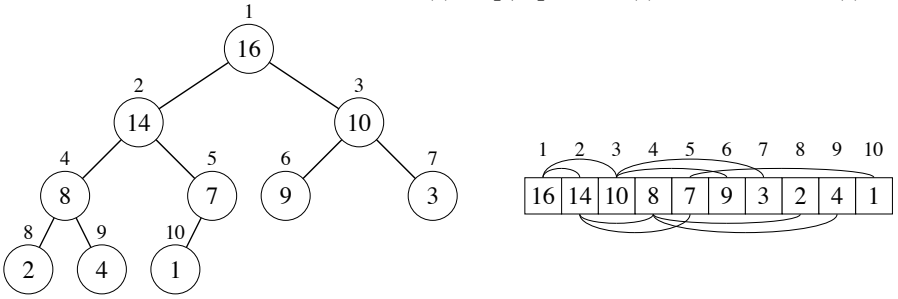
1. Justifiez que $T(n) \in \Omega(n)$.

2. Donnez une formulation récursive de $T(n)$ et justifiez celle-ci.

3. Sur base de cette formulation récursive, dérivez une borne serrée pour $T(n)$. Vous pouvez supposer que n est une puissance de 3.
   Remarque: la méthode à utiliser est libre, le raisonnement est évalué.

3. Tri

Pour rappel un tas est un arbre binaire complet, tel que la clé de chaque noeud est supérieure ou égale à celle de ses fils. Un tas peut être représenté de manière compacte à l'aide d'un tableau indicé à partir de 1. La racine de l'arbre est le premier élément du tableau, $\text{Parent}(i) = \lfloor i/2 \rfloor$, $\text{Left}(i) = 2i$ et $\text{Right}(i) = 2i + 1$.



Une fonction fondamentale de manipulation des tas est la fonction MAX-HEAPIFY(A, i). Considérons l'arbre dont la racine est i. Si c'est un arbre binaire complet, et si les sous-arbres gauche et droit sont des tas, un appel à MAX-HEAPIFY(A, i) réarrange l'arbre en tas contenant les éléments initiaux.

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  largest = i
 4  if l ≤ A.heap-size ∧ A[l] > A[largest]
 5       largest = l
 6  if r ≤ A.heap-size ∧ A[r] > A[largest]
 7       largest = r
 8  if largest ≠ i
 9       swap(A[i], A[largest])
10       MAX-HEAPIFY(A, largest)
```

1. Réécrire cette fonction de façon itérative. **Solution:**

   Au plus proche de la fonction originale:

```
MAX-HEAPIFY(A, i)
 1  repeat
 2       l = LEFT(i)
 3       r = RIGHT(i)
 4       largest = i
 5       if l ≤ A.heap-size ∧ A[l] > A[largest]
 6            largest = l
 7       if r ≤ A.heap-size ∧ A[r] > A[largest]
 8            largest = r
 9       if largest == i
10            return
11       swap(A[i], A[largest])
12       i = largest
13  until FALSE
```

   Plus succinct

```
MAX-HEAPIFY(A, i)
 1  repeat
 2       l = LEFT(i)
 3       r = RIGHT(i)
 4       j = i
 5       if l ≤ A.heap-size ∧ A[l] > A[i]
 6            i = l
 7       if r ≤ A.heap-size ∧ A[r] > A[i]
 8            i = r
 9       swap(A[i], A[j])
10  until i == j
```

2. Écrire un algorithme de tri basé sur les tas. Vous pouvez utiliser cette fonction. L'algorithme prend en argument un tableau non trié A, et à l'issue de l'appel à HEAP-SORT(A), le tableau est trié.

   Solution: Voir cours:

```
HEAP-SORT(A)
1  BUILD-MAX-HEAP(A)
2  for i = A.length downto 2
3       swap(A[i], A[1])
4       A.heap-size = A.heap-size − 1
5       MAX-HEAPIFY(A, 1)
```

```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3       MAX-HEAPIFY(A, i)
```

3. Donner la complexité en pire cas de cet algorithme et expliquer brièvement.

   Solution: Deux fois $O(n)$ appels à MAX-HEAPIFY, chacun étant $O(\log n)$. La complexité est $O(n \log n)$. Comme c'est aussi la complexité du problème du tri, l'algorithme est donc $\Theta(n \log n)$.

4. Structures de données

1. Qu'est-ce qu'un arbre binaire de recherche ?

2. Supposons qu'un arbre binaire de recherche contienne des clés entières comprises entre 1 et 1000 et qu'on recherche dans cet arbre la clé 363. Quelle séquence ci-dessous ne peut pas être la séquence des clés examinées lors de la recherche ? Justifiez votre réponse.

   (i) 4 924 278 347 621 299 392 358 363

   (ii) 2 252 401 398 330 363

   (iii) 3 923 220 911 244 898 258 362 363

3. Ecrivez une fonction BST-LEAST-DIFFERENCE qui renvoie en $O(n)$ la plus petite différence (en valeur absolue) entre deux éléments d'un arbre binaire de recherche. Par exemple, si l'arbre T contient les clés $\{4, 2, 7, 11, 8\}$, BST-LEAST-DIFFERENCE(T) doit renvoyer 1 (la plus petite différence étant $|7 - 8| = 1$).

5. Résolution de problèmes

Soit un tableau A de n valeurs. Afin de résumer A en k parties on cherche $I = [i_1, \ldots, i_{k-1}]$ tel que

$$err(A, I) = \sum_{l=1}^{k} V_A(i_{l-1}, i_l) \tag{1}$$

$$= \sum_{l=1}^{k} \frac{1}{i_l - i_{l-1}} \sum_{i=i_{l-1}+1}^{i_l} \left(A[i] - \overline{A[i_{l-1}+1 \,..\, i_l]} \right)^2 \tag{2}$$

soit minimum, avec $i_0 = 0$ et $i_k = n$ et où $\overline{A[i \,..\, j]}$ est la valeur moyenne du sous-tableau $A[i \,..\, j]$.
Informellement, on souhaite découper le tableau en k sections contiguës homogènes, peu importe leur longueur. Par exemple, si A est le tableau

$$[1, 2, 3, 2, 7, 6, 5, 2, 0] \tag{3}$$

on peut résumé A en $k = 3$ parties en choisissant les indices $i_1 = 4$, $i_2 = 7$. Ainsi, les sous-tableaux $A[1 \,..\, 4] = [1, 2, 3, 2]$, $A[5 \,..\, 7] = [7, 6, 5]$ et $A[8 \,..\, 10] = [2, 0]$ forment trois parties homogènes. Le résumé, à proprement parlé, est le tableau des moyennes: $[2, 6, 1]$.
Dans ce qui suit, vous pouvez supposer disposer de la fonction V_A et de son homologue en pseudo-code VARIANCE(A, p, q), où A est le tableau, $p + 1$ est l'indice de début du sous-tableau et q est l'indice de fin (inclus).

1. Discutez de la complexité (en k et n) d'une approche exhaustive pour résoudre ce problème.

2. Soit $E(j, k)$ l'erreur minimum au sens de (1) sur le tableau $A[1 \,..\, j]$ lorsqu'on résume A en k parties. Formulez récursivement $E(j, k)$.

3. Donnez un pseudo-code d'une fonction ERROR(A, j, k) permettant de calculer *efficacement* $E(j, k)$ pour un tableau A.

4. Analysez la complexité de votre algorithme en temps et en espace (en k et n).