Algorithmique
Examen de 2e session

Lundi 19 juin 2017 12h00–15h00

Documentation autorisée : deux feuilles A4 recto verso manuscrites. Toute autre document ou appareil est interdit. **Les téléphones portables doivent être éteints et déposés dans votre sac et vous devez poser vos sacs et vos vestes à l'avant de la salle avant de commencer.**

Indications : Le barème est donné à titre indicatif seulement. Sauf indication contraire, justifiez vos réponses et expliquez vos algorithmes. Les algorithmes doivent être aussi efficaces que possible et présentés de façon lisible et claire. Cela veut dire que les noms de variables doivent être bien choisis pour représenter ce qu'elles sont censées contenir et que les principales étapes doivent être accompagnées d'explications. Une partie de la note portera sur la clarté de présentation de vos réponses. Les algorithmes doivent être donnés en pseudo-code impératif (comme Python, C ou Java). Les langages fonctionnels ne sont pas admis.

Exercice 1 : (6 points)

On souhaite donner un algorithme de type diviser-pour-régner qui effectue un tri de type fusion, mais qui découpe le tableau de taille n en $k = 2^t$ parties (plutôt qu'en 2 parties comme le fait l'algorithme traditionnel). Chaque partie est triée récursivement. On suppose que t est une constante fixe, et disons, pour les fins d'explication, que $t = 2$. Dans ce cas, l'algorithme coupe la liste en $k = 2^t = 4$ listes de taille (presque) $n/4$, et procède récursivement sur les 4 sous-listes.

1. Proposer un algorithme `multifusion` de fusion efficace qui prend en entrée k listes triées (k quelconque n'est pas connu d'avance) et retourne une liste contenant tous les éléments des k listes dans l'ordre. En pratique l'algorithme prend en entrée une liste `L` composée de k listes. Les listes données en entrée peuvent avoir des longueurs différentes et peuvent être vides.
2. Donner la complexité de votre algorithme losrque l'entrée `L` est composée de k listes de taille m, en fonction de m et k. Justufier votre réponse.
3. Donner l'algorithme de tri par fusion qui utilise l'approche diviser-pour-régner décrite ci-dessus. Utiliser l'algorithme `multifusion` pour la fusion.
4. Donner la récurrence qui décrit la complexité de votre algorithme pour t fixé quelconque. (Une partie des points sera accordée si vous ne traitez que le cas particulier $t = 2$, c'est-à-dire, $k = 2^t = 4$.)
5. Résoudre la récurrence pour t fixé quelconque. (Une partie des points sera accordée si vous ne traitez que le cas particulier $t = 2$.)
6. Comparer la complexité de cet algorithme à celle du tri fusion traditionnel (avec $t = 1$).

Exercice 2 : (3 points)

Nous avons vu qu'un problème R est dans la classe $\mathcal{NP}$ s'il existe un prédicat binaire `verif` décidable en temps polynomial tel que $R(x)$ est vrai si et seulement s'il existe un y tel que `verif(x,y)` est vrai. Pour chacun des problèmes suivants, décrire un prédicat `verif` adapté, et justifier qu'il est vérifiable en temps polynomial en décrivant brièvement un algorithme dont vous donnerez la complexité en fonction de la taille n de l'entrée

SETCOVER : Étant donné une collection d'ensembles $\mathcal{S} = \{S_1, \ldots, S_m\}$ et un entier k, existe-t-il une sous-collection de $\mathcal{S}$ de cardinal k dont l'union des éléments couvre celle des éléments de $\mathcal{S}$?

HAMILTONIANCYCLE : Étant donné un graphe G, existe-t-il un cycle *hamiltonien* dans G (visitant chaque sommet exactement une fois) ?

Exercice 3 : (3 points)

Résoudre les récurrences suivantes. (Donner uniquement la réponse.)

1. $T(N) = 3T(N/7) + N^3$
2. $T(N) = 7T(N/3) + N$
3. $T(N) = 4T(N/4) + N^{1/2}$

Exercice 4 : (8 points)

Un parc d'aventure a installé des ponts suspendus dans les arbres. Les arbres sont numérotés 1, 2,... , n. À l'entrée du parc, les participants doivent acheter des jetons. À chaque pont est associé un coût en nombre de jetons pour le traverser. Le but est de commencer à l'arbre 1 et terminer le parcours à l'arbre n en dépensant le moins de jetons possible. Le tableau des coûts est donné à l'avance aux participants. Un pont reliant les arbres i et j avec $i < j$ ne peut être emprunté que de i vers j. Plusieurs ponts peuvent partir d'un arbre vers des arbres différents. Au moins un pont part de l'arbre i si $i < n$. Vous pouvez supposer que la fonction $J(i, j)$ représente le nombre de jetons pour emprunter le pont qui relie l'arbre i à l'arbre j, si ce pont existe, et ∞ sinon.

1. Un participant hardi prend une stratégie gloutonne pour traverser le parc : à chaque arbre il prend le pont le moins cher. Cette stratégie est-elle optimale ? Si oui, donner la preuve. Si non, donner un contre-exemple de coût supérieur à la valeur optimale.
2. Un participant soucieux calcule la stratégie optimale avec un algorithme de type backtracking. Donner un tel algorithme et analyser sa complexité.
3. Un participant rusé propose un algorithme de type programmation dynamique pour résoudre le problème.
   (a) Donner une récurrence $T(i, j)$ qui exprime la valeur de la solution optimale pour aller de l'arbre i vers l'arbre j pour tout $1 \leqslant i < j \leqslant n$. *Attention : Toute réponse qui ne serait pas accompagnée d'une brève explication ne sera pas prise en considération.*
   (b) Préciser les dimensions, les conditions de bord et l'ordre de remplissage du ou des tableaux. Dire à quel indice se trouve la solution minimale à la fin de l'algorithme.

(c) Donner un algorithme de programmation dynamique pour trouver la solution optimale. Votre algorithme doit retourner la liste des ponts à emprunter. Une partie des points seront donnés si vous donnez uniquement la valeur optimale.

Annexe

Master Theorem

Théorème. *(Version du cours) Soit T une fonction vérifiant une relation de récurrence de la forme*

$$T(n) = a\,T(\tfrac{n}{b}) + f(n).$$

Alors :

(cas « f petit ») $af(N/b) = cf(N),\ c > 1 \implies T(n) \in \Theta(n^{\log_b a})$

(cas « f moyen ») $af(n/b) = f(N) \implies T(n) \in \Theta(n^{\log_b a} \log n)$

(cas « f grand ») $af(N/b) = cf(N),\ c < 1 \implies T(n) \in \Theta(f(n))$

Théorème. *(Version du TD) Soit T une fonction vérifiant une relation de récurrence de la forme*

$$T(n) = a\,T(\tfrac{n}{b}) + f(n).$$

Alors :

(cas « f petit ») $f(n) \in O(n^{\log_b a - \epsilon}) \implies T(n) \in \Theta(n^{\log_b a})$

(cas « f moyen ») $f(n) \in \Theta(n^{\log_b a}) \implies T(n) \in \Theta(n^{\log_b a} \log n)$

(cas « f grand ») $\left.\begin{array}{l} f(n) \in \Omega(n^{\log_b a + \epsilon}) \\ af(\frac{n}{b}) \leqslant cf(n)\ \ (c < 1) \end{array}\right\} \implies T(n) \in \Theta(f(n))$

Rappel : Pour calculer les logarithmes, on peut utiliser la formule suivante :

$$\log_b a = \frac{\log_2 a}{\log_2 b}$$