Algorithmique
Examen Partiel
vendredi 2 novembre 2018 13h30–15h30

Documentation autorisée : une feuille A4 recto verso manuscrite. Tout autre document ou appareil est interdit. **Les téléphones portables doivent être éteints et rangés dans votre sac, et vous devez poser sacs et 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.

Exercice 1 : (5 points)

Le problème $k-$SUM est le suivant. Soit k un entier fixé (penser par exemple à $k = 4$).

En entrée : une liste d'entiers positifs $X = x_1, \ldots, x_n$, une valeur cible T.

En sortie : `Vrai` s'il existe k éléments de X (d'indices distincts) de somme T. `Faux` sinon.

1. Donner une récurrence structurelle qui exprime la solution à k-SUM en fonction d'instances plus petites. Indication : donner une récurrence pour $S(X, T, l)$ où l est le nombre de valeurs qui restent à trouver (initialement $l = k$).
2. Donner un algorithme de type backtracking pour le problème $k-$SUM.
3. Donner la complexité de votre algorithme en fonction de n et k.

Exercice 2 : (5 points)

L'algorithme suivant parcourt un arbre binaire qui contient des valeurs au niveau de ses feuilles, et affiche, à chaque nœud interne, la médiane des valeurs dans le sous-arbre enraciné à ce noeud.

```
def parcours(arbre):
    if not arbre: return []
    if arbre.estFeuille(): return [arbre.getValeur()]
    Lgauche = parcours(arbre.filsGauche())
    Ldroite = parcours(arbre.filsDroit())
    L = Lgauche + Ldroite # Concaténation des deux listes
    print(arbre.getID(), mediane(L))
    return L
```

On suppose que les méthodes `getValeur()`, `filsGauche()`, `filsDroit()`, `getID()` ainsi que la concaténation de listes prennent un temps constant. On suppose que l'arbre est complet, c'est-à-dire que toutes les feuilles sont au même niveau et que chaque nœud interne a exactement deux fils. On note $M(n)$ la complexité de `mediane(L)` où n est la longueur de la liste L.

1. Donner une récurrence générale pour la complexité $P(N)$ de cet algorithme, en fonction du nombre de feuilles N de l'arbre, sans la résoudre et sans faire d'hypothèse sur $M(N)$.
2. Supposons que la fonction médiane soit implémentée avec l'algorithme suivant. (1) Trier L. (2) Retourner le $(\lceil |L|/2 \rceil)$-ième élément. Donner une récurrence pour $P(N)$ en précisant $M(N)$. En déduire un ordre de grandeur pour $P(N)$.
3. Proposer un algorithme plus efficace pour implémenter le calcul de la médiane. (Donner uniquement sa complexité sans le décrire en détail.) Donner une récurrence pour $P(N)$ en précisant $M(N)$. En déduire un ordre de grandeur pour $P(N)$.

Exercice 3 : (5 points)

Résoudre les récurrences suivantes en donnant l'ordre de grandeur. Dans tous les cas, $T(N) = 1$ lorsque $N \leqslant 2$. (Donner uniquement la réponse.)

1. $T(N) = 6\,T(N/2) + N^3$
2. $T(N) = 4\,T(N/2) + N^2$
3. $T(N) = 4\,T(N/5) + N$
4. $T(N) = 3\,T(N/2) + \sqrt{N}$
5. $T(N) = T(N-2) + T(N-3)$

Exercice 4 : (5 points)

On définit le problème PARTITION comme suit :

En entrée : Un ensemble de valeurs positives $L = x_1, \ldots, x_n$.

En sortie : `Vrai` s'il existe une partition de L en deux sous-ensembles L_1, L_2 telle que $\sum_{x \in L_1} x = \sum_{y \in L_2} y$, Faux sinon.

On rappelle que SUBSETSUM défini ci-après est $\mathcal{NP}$-complet.

En entrée : Un ensemble de valeurs positives $L = x_1, \ldots, x_n$ et une valeur cible T.

En sortie : `Vrai` s'il existe un sous ensemble de valeurs dans L de somme T, `Faux` sinon.

On définit les deux fonctions f, g suivantes :

$$f(L) = (L, \frac{1}{2}\sum_{x \in L} x)$$

$$g(L, V) = L \cup \{\sum_{x \in L} x + V, 2\sum_{x \in L} x - V\}$$

1. Montrer que PARTITION $\leqslant$ SUBSETSUM.
2. Montrer que SUBSETSUM $\leqslant$ PARTITION.
3. Montrer que PARTITION EST $\mathcal{NP}$-COMPLET.

Annexe

Identités sur les logarithmes

- $\log_u(v) = \log_2(v)/\log_2(u)$
- $\log_a(b) < \log_a(c) \iff b < c$
- $\log_a(b) < \log_c(b) \iff a > c$

Master Theorem

Théorème. *Soit T une fonction vérifiant une relation de récurrence de la forme*

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

Alors (version du cours) :

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

Ou (version du TD) :

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

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