Programmation avancée

Examen écrit, 11 janvier 2016

Livres fermés. Durée : 3h30.

Remarques

— Répondez à chaque question sur une feuille **séparée**, sur laquelle figurent votre nom et votre section.
— 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).

Question 1 : Vrai ou faux

Pour chacune des affirmations suivantes, déterminez si elle est vraie ou fausse et justifiez brièvement votre choix :

1. On peut déterminer si un arbre binaire respecte la propriété d'arbre binaire de recherche sur base uniquement de l'affichage de ses clés lors d'un parcours infixe.
2. Si $f(n) \in O(g(n))$ et que $g(n) \in \Omega(h(n))$ alors $f(n) \in O(h(n))$.
3. Lorsqu'une table de hachage a un facteur de charge de $\frac{1}{2}$, il y a une chance sur deux d'avoir une collision.
4. Une file double permet d'implémenter une pile avec des opérations en $\Theta(1)$.
5. Un arbre est H-équilibré si sa profondeur est $\Theta(\log n)$.

Question 2 : Tri

1. Décrivez l'algorithme de tri rapide dans le formalisme de votre choix. Donnez sa complexité dans le meilleur et le pire cas.
2. Cet algorithme est-il stable ? Est-il en place ? Justifiez vos réponses.
3. Construisez l'arbre de décision correspondant à cet algorithme sur un tableau de taille trois.
   Remarque : veuillez à ce que votre description du tri rapide soit suffisamment précise pour que l'arbre de décision soit non ambigu.

Question 3 : Analyse de complexité

1. Enoncez le master theorem
2. Supposons que vous disposiez des trois algorithmes suivants pour résoudre un problème de taille n :
   — L'algorithme A qui divise le problème en 4 sous-problèmes de taille $n/2$, résoud récursivement chacun des sous-problèmes, puis combine leurs solutions en utilisant n opérations exactement.
   — L'algorithme B qui divise le problème en 3 sous-problèmes de tailles $n/2$, résoud récursivement chacun des sous-problèmes, puis combine leurs solutions en utilisant n^2 opérations exactement.
   — L'algorithme C qui divise le problème en 9 sous-problèmes de tailles $n/3$, résoud récursivement chacun des sous-problèmes, puis combine leurs solutions en utilisant n^2 opérations exactement.

   Lequel de ces trois algorithmes est le plus efficace pour des grandes valeurs de n ? Justifiez le plus précisément possible votre réponse. Ne résolvez exactement les récurrences que si c'est nécessaire pour répondre à la question mais fournissez au minimum la complexité en notation asymptotique des trois algorithmes. Pour les trois récurrences, vous pouvez supposer que le traitement du cas de base ($n = 1$) et la division du problème en sous-problèmes ne nécessitent aucune opération et que la valeur de n se divise toujours parfaitement par 2 ou 3.

Question 4 : Arbres

1. Soit un arbre binaire T contenant $N = 10$ nœuds étiquetés et dont le parcours

   infixe est $[I, C, J, B, K, D, A, F, G, E]$

   postfixe est $[I, J, C, K, D, B, G, F, E, A]$

   Dessinez un arbre T correspondant à ces ordres. Cet arbre est-il unique ?
2. Donnez le pseudo-code d'une fonction BuildBackTree(I, P) permettant de reconstruire un arbre binaire de taille N à partir de ses séquences infixe et postfixe. Les deux arguments sont :

   I Un tableau de taille N qui contient les éléments de l'arbre en ordre infixe.

   P Un tableau de taille N qui contient les éléments de l'arbre en ordre postfixe.
3. Analysez les complexités de votre algorithme au pire et meilleur cas.

Question 5 : Résolution de problème

Vous êtes engagé comme stagiaire dans une banque et votre superviseur vous demande d'écrire un programme permettant de calculer combien d'argent il aurait pu gagner s'il avait investi de manière optimale dans le passé.

Plus formellement, sur base des informations suivantes :

— Une somme d'argent V disponible au temps t_1,

— Le prix de n actions au temps t_1 sous la forme d'un tableau $v_1[1\,..\,n]$,

— Le prix des n mêmes actions au temps $t_2 > t_1$ sous la forme d'un deuxième tableau $v_2[1\,..\,n]$,

on vous demande de calculer le montant maximum que vous auriez pu obtenir en t_2 en investissant de manière optimale le montant V au temps t_1, ainsi que la quantité de chacun des actions que vous auriez du acheter pour obtenir ce montant maximum. On supposera que toutes les valeurs d'actions et le montant V à investir prennent des valeurs positives entières. L'entièreté de la somme V ne doit évidemment pas nécessairement être totalement investie. L'argent non investi en t_1 fera partie du montant disponible en t_2.

1. Soit les prix des actions données dans la table ci-dessous. Donnez la solution optimale dans le cas où $V = 20$ et dans le cas où $V = 120$.

v_1	12	10	18	15
v_2	39	13	47	45

2. Expliquez les similarités et différences entre ce problème et le problème du sac à dos vu au cours.
3. Proposez une solution gloutonne plausible au problème et montrez par un contre-exemple qu'elle n'est pas correcte.
4. En vous inspirant de la solution au problème du sac à dos, on vous demande d'écrire un algorithme efficace basé sur la programmation dynamique pour résoudre ce problème. Suivez les étapes suivantes :
   (a) Précisez la fonction de coût et les équations de récurrence (y compris le/les cas de base) correspondant à votre solution.
   (b) Ecrivez le pseudo-code d'un algorithme efficace pour calculer cette fonction de coût.
   (c) Modifiez ce pseudo-code pour récupérer la solution optimale.
   (d) Analysez la complexité de votre solution dans le meilleur et dans le pire cas en fonction du nombre d'actions n et du montant V investi.

   Remarque : Vous pouvez ne donner qu'un seul pseudo-code pour les sous-questions (b) et (c) si vous mettez bien en évidence les parties relatives à la récupération de la solution optimale.