Examen d'Algorithmique avancée (3 heures)

ENSIMAG Alternants 2A

Seuls documents autorisés : 1 page A4 recto-verso manuscrite.

1 Divide and Conquer – 40mns (5 pts)

Un tableau possède un élément majoritaire si plus de la moitié (strictement) de ses entrées sont identiques. Etant donné un tableau A à n éléments, **on se propose de concevoir un algorithme efficace qui permet de déterminer s'il possède un élément majoritaire, et dans l'affirmative de l'identifier.** Les éléments du tableau ne sont pas forcément comparables comme des entiers ou des caractères alphabétiques et il n'est pas possible d'effectuer des comparaisons de la forme A[i] $>$ A[j] (on peut penser par exemple à des images). Par contre, on peut répondre à des questions de la forme « A[i] = A[j]? » en temps constant.

Question 1.1. Proposez un algorithme simple pour trouver l'élément majoritaire dans un tableau de dimension n et évaluer son coût (on suppose que la mémoire est limitée).

Une autre approche consiste à utiliser le paradigme "diviser pour régner" : couper le tableau A en deux tableaux A1 et A2 de même taille (Dans un premier temps, on pourra supposer que la taille du tableau initial est une puissance de 2). On pourra analyser cet algorithme en distinguant les cas suivants :

1. A1 et A2 ont chacun un élément majoritaire
2. aucun des deux n'en possède
3. un seul des deux a un élément majoritaire

Question 1.2. Ecrire l'algorithme détaillé en pseudo code.
Question 1.3. Quel est son coût ? (On donnera l'équation de récurrence et on explicitera comment la résoudre).

On suppose toujours que la taille de A est une puissance de 2.

On se propose maintenant d'**améliorer la solution précédente** par un algorithme qui utilise toujours le paradigme *divide-and-conquer*. Soit cet algorithme garantit que le tableau, A ne contient pas d'élément majoritaire, soit il renvoie un élément a et un entier $c_a > n/2$ tel que a apparaisse au

plus c_a fois dans le tableau et tout autre élément apparaisse moins de $n - c_a$ fois.

Question 1.4. Donner à partir de l'algorithme précédent un algorithme complet qui vérifie si un tableau A contient un élément majoritaire et calculer son coût (en pire cas).

2 Sac-à-dos – 15mns (3 pts)

Un algorithme glouton raisonnable consiste à prendre un à un les objets dans l'ordre des densités décroissantes. Cependant, comme nous l'avons montré en cours, cette politique n'est pas bonne.
Question 2.1. Expliciter la construction d'une instance arbitrairement mauvaise.

Nous allons montrer maintenant comment l'adapter pour obtenir un algorithme avec une garantie de performance. On considère le premier objet qui ne rentre pas dans le sac en appliquant la politique *qualité-prix*. On note j cet objet.

Question 2.2. Expliciter l'argument pour prouver la propriété suivante : $OPT \leq \Sigma_{i<j} b_i + b_j$

On en déduit l'algorithme suivant :

- Sélectionner les objets du sac-à-dos par la politique qualité-prix.
- Remplir le sac en considérant les objets dans cet ordre, prenant la plus grande valeur entre le bénéfice obtenu de cette façon et le bénéfice de l'objet suivant j.

Question 2.3. Montrer que cette politique est une $\frac{1}{2}$-approximation.

3 Variante de 2Partition – 15mns (3 pts)

On se propose ici d'étudier la complexité de la variante suivante de 2Partition :

PP (Parfaite Partition)
Input : Etant donnés $2n$ entiers s_i.
Question : Existe-t-il une partition *parfaite* en deux parties égales A_1 et A_2, c'est-à-dire tels que $\Sigma_{i \in A_1} s_i = \Sigma_{i \in A_2} s_i$ où $|A_1| = |A_1| = n$?

Question 3.1. démontrer que ce problème est NP-complet.

4 BinPacking – 50mns (6 pts)

On revient ici sur le sujet qui nous a occupé pendant le mini-projet.

On considère un ensemble de n objets de tailles s_i pour $1 \leq i \leq n$ et un grand nombre de "boites" de hauteur H. On veut ranger les objets en les empilant dans un nombre minimal de boites. Bien sûr, on ne pourra pas dépasser la capacité H de chaque boite.

Pour une instance donnée on notera N_A le nombre de boites obtenu en appliquant l'algorithme A. N_{opt} désigne le nombre optimal de boites. Ce problème est le *BinPacking*, il s'exprime sous la forme suivante :

BinPacking
Instance. : n objets (de tailles entières s_i), un entier H (la hauteur maximal des boites).
Question. Déterminer un *empilement* des objets dans un nombre minimum de boites, sans dépasser la capacité de chaque boite.

Question 4.1. Formuler ce problème sous forme de décision.

Question 4.2. Montrer que BinPacking/décision est NP-complet.

On se propose maintenant d'analyser l'heuristique *FF* (First Fit) suivante : on range les objets l'un après l'autre, si l'objet ne rentre dans aucune boite, on ouvre une nouvelle boite.

Question 4.3. Montrez rapidement que l'on peut se ramener à un problème où les boites sont de tailles unitaires avec des objets rationnels. Est-ce que ceci change la complexité du problème ?

Question 4.4. Montrez que *FF* est une 2-approximation.

En supposant que *FF* utilise m boites, on pourra utiliser l'argument (évident) qu'au moins $m-1$ boites sont remplies à plus de la moitié...

Nous allons aller un peu plus loin dans l'analyse de l'approximation en raffinant l'heuristique *FF* en triant au préalable les objets par tailles décroissantes (on place d'abord les plus grands).

On cherche maintenant à établir que cette heuristique (appelée *FFD* pour First Fit Decreasing) a un rapport d'approximation meilleur inférieur à $\frac{3}{2}$.

Pour cela, nous allons découper l'instance en 4 paquets selon la taille des objets :

$A_1 = \{s_i > \frac{2}{3}\}$
$A_2 = \{\frac{2}{3} \geq s_i > \frac{1}{2}\}$
$A_3 = \{\frac{1}{2} \geq s_i > \frac{1}{3}\}$
$A_4 = \{\frac{1}{3} \geq s_i\}$

Question 4.5. (difficile) Démontrer le rapport d'approximation de $\frac{3}{2}$ pour FFD en distinguant le cas où il y a au moins une boite avec des objets de la catégorie A_4 et celui où il n'y a pas de telle boite.