400099-2011-M1
1er semestre

R1 IF M1 AA 11

Université Lille1
UFR IEEA

Examen AAC 2ème session
Documents autorisés

28 février 2011
M1 Informatique

Exercice 1 : Compétences de base

Remarque: pour chaque question de cet exercice, des points seront retirés si le nombre de réponses incongrues est déraisonnable.

Q 1. Trier les fonctions $n^4 + \sqrt{n}, n + \log n, n^3 + 2n^2, 4n^2 + 4n, n \log n, 2^n, \log n + \sqrt{n}, 3^n$ suivant leur ordre de grandeur asymptotique: f sera "avant" g si $f \in O(g)$. Par exemple, $n, 1, n^2, \log n$ serait trié en $1, \log n, n, n^2$.

Q 2. Pour chacun des algorithmes suivants, dire si sa complexité (temporelle) dans le pire des cas est en $\Theta(\log n)$, $\Theta(n)$ ou $\Theta(2^n)$, en justifiant très brièvement:

```
int T1(int n){
  if (n <= 0) return 1;
  else return 3*T1(n-3);}
```

```
int T2(int n){
  int c=0;
  for (int i=1;5*i<n;i++) c++;
  return c;}
```

```
int T3(int n){
  if (n <= 0) return 1;
  else return 1+ 4*T3 (n/3);}
```

```
int T4(int n){
  if (n <= 0) return 1;
  else return T4(n-1)+T4(n-1);}
```

Q 3. Répondre en justifiant brièvement aux questions suivantes:

Q 3.1. Toute propriété P est-elle NP?

Q 3.2. Toute propriété NP peut-elle être décidée par un algorithme exponentiel?

Q 3.3. Votre collègue prétend avoir trouver un algorithme polynomial pour décider une propriété NP-dure. Qu'en pensez-vous?

Exercice 2 : La bonne tranche

Soit un tableau T de n entiers triés dans l'ordre croissant et a et b deux entiers: on recherche dans le tableau T le nombre d'éléments compris entre deux valeurs a et b.

Entrée : n entier, T un tableau de n entiers, a et b deux entiers ($a \leq b$).

Sortie : Le nombre de i tel que $a \leq T[i] \leq b$

Par exemple, si $n = 6$ et $T[0] = 1, T[1] = 6, T[2] = 12, T[3] = 12, T[4] = 19, T[5] = 24, T[6] = 27$, alors pour $a = 6$ et $b = 15$, on retournera 3, pour $a = 13$ et $b = 15$, on retournera 0.

Proposer un algorithme en $O(log\ n)$ pour le problème. Justifier qu'il est correct et que sa complexité est bien en $O(log\ n)$.

Exercice 3 : Le plus grand facteur

Le problème: On cherche à trouver le(un) plus long facteur commun à deux mots. w est facteur de u si u peut s'écrire dwf, avec d et f deux mots éventuellement vides. Par exemple, les facteurs du mot abc sont donc $\epsilon, a, b, c, ab, bc, abc$. Le plus long facteur commun aux mots *photographe* et *grappe* est *grap*. Le plus long facteur commun aux mots *algorithme* et *tango* est *go*. Les plus longs facteurs communs à *algorithme* et *complexe* sont tous de longueur 1: *l*,*o*,*m*,*e*.

On pourra supposer que les mots sont des objets JAVA de type STRING et qu'on dispose des fonctionnalités de base du type. On notera Σ l'alphabet.

Q 1. Dans un premier temps, on pourra se contenter de chercher la longueur:

Entrée: deux mots u et v sur Σ

Sortie: La longueur d'un plus long facteur commun.

Proposer un algorithme en $O(|u| * |v|)$ qui résout le problème:

Aide: On pourra utiliser la programmation dynamique avec une table L, où $L(i, j)$ représentera la longueur maximale d'un facteur DROIT commun au mot formé des i premières lettres de u et à celui formé des j premières lettres de v. Par exemple, pour les mots *photographe* et *grappe*:

		p	*h*	*o*	*t*	*o*	*g*	*r*	*a*	*p*	*h*	*e*
	0	0	0	0	0	0	1	0	0	0	0	0
g	0	0	0	0	0	0	1	0	0	0	0	0
r	0	0	0	0	0	0	2	0	0	0	0	0
a	0	0	0	0	0	0	0	3	0	0	0	0
p	1	0	0	0	0	0	0	0	4	0	0	0
p	1	0	0	0	0	0	0	0	1	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0	1

Q 2. Compléter l'algorithme précédent pour retourner également un facteur commun de longueur maximale.

Exercice 4 : Partage

Le problème de décision 3−Partition consiste en répartir en trois ensembles des entiers de façon à ce que les sommes des trois ensembles soient identiques. Formellement , $3 - Partition$ est défini par:

Donnée:

n –un entier

x – un tableau de n entiers

Sortie:

Oui Ssi il existe J, K, L trois sous-ensembles de $[1..n]$ tels que

1. $J \cap K = J \cap L = K \cap L = \emptyset$ et $J \cup K \cup L = [1..n]$, i.e. J, K, L partition de $[1..n]$
2. $\sum_{i \in J} x_i = \sum_{i \in K} x_i = \sum_{i \in L} x_i$

Q 1. Combien de partitions de $[1..n]$ en 3 sous-ensembles existe-t-il?

Q 2. Montrer que 3−Partition est NP.

Q 3. On rappelle que 2−Partition est NP-dur, avec 2−Partition défini par:

Donnée:

n –un entier

x – un tableau de n entiers

Sortie:

Oui Ssi il existe J, K deux sous-ensembles de $[1..n]$ tels que

1. $J \cap K = \emptyset$
2. $J \cup K = [1..n]$
3. $\sum_{i \in J} x[i] = \sum_{i \in K} x[i]$

Q 3.1. Montrer que 2−Partition se réduit polynomialement en 3−Partition.

Q 3.2. Qu'en déduire pour 2−Partition?

Q 3.3. Pensez-vous que 3−Partition se réduit polynomialement en 2−Partition? Justifier brièvement.

Q 4. On s'intéresse maintenant au problème d'optimisation. On suppose ici que tous les entiers $x[i]$ sont positifs ou nuls. On cherche donc à partitionner les $x(i]$ en trois, de façon la plus équilibrée. Le problème 3 – *PartitionOpt* est donc défini par:

Donnée:
n –un nb d'entiers
x – un tableau de n entiers
Sortie:
J, K, L trois sous-ensembles de $[1..n]$ tels que

1. $J \cap K = J \cap L = K \cap L = \emptyset$
2. $J \cup K \cup L = [1..n]$
3. $max(\sum_{i \in J} x[i], \sum_{i \in K} x[i], \sum_{i \in L} x[i])$soit minimal.

Q 4.1. D'après ce qui précède, que pensez-vous de la complexité du problème 3 – *PartitionOpt*?
Q 4.2. Justifier qu'il existe un algorithme exponentiel pour 3 – *PartitionOpt*.
Q 4.3. On suppose ici qu'on dispose d'une classe Ensemble pour implémenter les ensembles d'entiers, avec $ajout(x)$ qui permet d'ajouter un entier, et $somm()$ qui calcule la somme des éléments de l'ensemble. Soit l'algorithme:

```
// x tableau de n entiers
boolean Trois_Part(){
Ensemble I,J,K; //I,J,K ensembles initialises au vide
for (i=0;i<x.length;i++){
    if (I.somme() <= J.somme())
          if  (I.somme() <=K.somme()) I.ajout(x[i]);
          else K.ajout(x[i]);
    else
          if  (J.somme() <=K.somme()) J.ajout(x[i]);
          else K.ajout(x[i];}
```

Montrer par un exemple que l'algorithme ne donne pas toujours la solution optimale.
Q 4.4.*Bonus* Montrer que l'algorithme offre une garantie de 5/3.

Exercice 5 : Réservations et satisfaction des clients

Soient p trains numérotés dans l'ordre de leur heure de départ et n clients. Pour chaque train, on connaît le nombre de places libres, $plac_1, ..., plac_p$. et pour chaque client le numéro du train qu'il souhaite prendre, $dem_1, ..., dem_n$ (on suppose que chaque client veut une et une seule place). On suppose que si le train demandé est complet, le client accepte de prendre un train après celui qu'il avait choisi. Le problème est d'affecter une place à chaque client dans un train, soit formellement:

Entrée:
n, p – le nombre de clients, le nombre de trains
$plac_1, ..., plac_p$ – le nombre de places libres par train
$dem_1, ..., dem_n$ – les demandes des clients (une demande: un numéro de train)
Sortie: Une affectation i.e. $aff : [1..n] \rightarrow [1..p]$ telle que
$aff(i) >= dem_i$ $1 \leq i \leq n$ –le client i ne prend pas un train avant celui qu'il avait choisi
Et
$Card\{i/aff(i) = j\} <= plac_j$ $1 \leq j \leq p$ – il n'y a pas de surbooking!

Q 1. Soient $p = 3$, $n = 6$, les demandes $2, 1, 3, 2, 2, 3$ avec deux places disponibles dans chacun des trois trains. Y-a-t-il une affectation possible?
Q 2. Proposer un algorithme qui vérifie si il existe une affectation possible.
Q 3. Bien sûr toutes les affectations ne satisfont pas les clients de la même façon! On impose donc d'abord la condition suivante:

(Cond) si $dem_i < dem_j$, alors $aff_i <= aff_j$: si un client a demandé un train qui part avant celui demandé par un autre, il ne peut partir après!

Par exemple, pour $p = 3$, $n = 6$ et les demandes $2, 1, 1, 2, 1, 1$ avec deux places libres dans chaque train, si on a $aff(1) = 2, aff(2) = 1, aff(3) = 1, aff(4) = 2, aff(5) = 3, aff(6) = 3$, la condition n'est pas respectée: le client 4 part avant le client 5 alors qu'il avait demandé à partir après!
Q 3.1. Soient $p = 3$, $n = 6$ et les demandes $2, 2, 1, 3, 1, 1$ avec deux places de disponibles dans chacun des trois trains. Proposer une affectation qui respecte la condition (Cond).
Q 3.2. Montrer que si il existe une affectation possible, il existe une affectation qui vérifie la condition (Cond).
Q 4. La condition (Cond) n'est pas suffisante pour exprimer le contentement des passagers: par exemple, si le dernier train contient suffisamment de places, on pourrait mettre tous les clients dans ce dernier train!

On ajoute donc une mesure du mécontentement: on mesure le degré de mécontentement d'un client par $aff(i) - dem(i)$, qui est donc nul si il a eu le train désiré, >0 sinon.

Le degré total de mécontentement est simplement la somme des degrés de mécontentement des clients, i.e. $\sum_{i=1}^{i=n}(aff(i) - dem_i)$.

Par exemple, pour $p = 3$, $n = 7$ et les demandes $2, 1, 1, 2, 1, 1, 3$ avec trois places libres dans chaque train, si on a $aff(1) = 3, aff(2) = 1, aff(3) = 1, aff(4) = 3, aff(5) = 2, aff(6) = 1, aff(7) = 3$, le degré total de mécontentement est $(3 - 2) + (1 - 1) + (1 - 1) + (2 - 1) + (2 - 1) + (1 - 1) + (3 - 3)$, soit 3.
Q 4.1. Pour l'exemple précédent, donner une affectation qui vérifie la condition (Cond) et qui minimise le mécontentement.
Q 4.2. Proposer un algorithme polynomial (de type glouton) qui permet de réaliser une affectation qui respecte la condition (Cond) et minimise le degré de mécontentement.

Justifier que votre algorithme produit bien une solution optimale et donner l'ordre de grandeur de sa complexité.