Université Lille1
UFR IEEA

Algorithmique Avancée et Complexité

2012 2013
Master d'Informatique-S1
Documents autorisés

12 Janvier 2013

Bonne Année à tous!

La clarté et la rigueur seront prises en compte dans l'évaluation. Vous pouvez écrire vos algorithmes en pseudo-langage ou dans un langage de programmation de votre choix.

Exercice 1 : Connaissances de base

Q 1. *Complexité d'algorithmes*

Pour chacun des algorithmes suivants, dire si sa complexité (temporelle) dans le pire des cas est en $\Theta(log\ n)$, $\Theta(n)$, $\Theta(n\ log\ n)$, $\Theta(n^2)$, $\Theta(n^3)$, $\Theta(3^n)$ ou $\Theta(4^n)$ en justifiant brièvement:

```
int T1(int n){
  int c=0;
  for (int j=n;j>0;j--)
     for (int k=1;k<n; k=k*3)
        for (int i=k;i<k+10;i=i+1) c++;
  return c;}
```

```
int T2(int n){
  if (n>1)
    return T2(n-1)+T2(n-1)+T2(n-1)+T2(n-1);
  else return 1;}
```

```
int T3(int n){
  if (n>3)
    return T3(n/4)+T3(n/4)+T3(n/4)+T3(n/4);
  else return 1;}
```

```
int T6(int n){
  int c=0;
  int sq=n*n;
  for (int i=1;i<sq;i=4*i) c++;
  return c;}
```

Q 2. *Polynomial?*

On cherche un algorithme pour décider si un entier naturel est un carré parfait, i.e. si un entier x est de la forme i^2 avec i entier. Soit l'algorithme suivant:

```
boolean EstCarre(int x){
  for (int i=0; i<=x; i++)
     if (i*i==x) return true;
  return false;}
```

L'algorithme est-il correct? Est-il polynomial? Pourquoi? La propriété "être un carré parfait" est-elle polynomiale"?

Q 3. *La classe NP*

Q 3.1. La propriété de décision *Dec* a été prouvée NP-complète. Que pensez-vous de chacune des trois affirmations suivantes? Justifier brièvement.

- La propriété est décidable.
- Il existe un algorithme non déterministe polynomial pour décider *Dec*.
- Il n'existe pas d'algorithme déterministe polynomial pour décider *Dec*.
- Il existe un algorithme déterministe non polynomial pour décider *Dec*.
- Il existe un algorithme déterministe exponentiel pour décider *Dec*.

Q 4. *Récursif?*

Pour chaque affirmation suivante, dire si elle est vraie ou fausse. Attention, sera pris en compte le nombre de réponses justes moins le nombre de réponses fausses.

1. L'union de deux langages algébriques est un langage récursif.
2. Le complémentaire d'un langage récursif est récursivement énumérable.
3. Le complémentaire d'un langage récursif est récursif.
4. Le complémentaire d'un langage récursivement énumérable est récursivement énumérable.
5. Le complémentaire d'un langage récursivement énumérable est récursif.

Exercice 2 : Stratégie

Soit le jeu suivant: une séquence de n cartes $c_0, c_1, \ldots, c_{n-1}$ est tirée au hasard et placée en une rangée, face visible sur la table. Chaque carte c_i a une valeur v_i. Il y a deux joueurs A et B qui jouent tour à tour: à chaque tour, le joueur qui a la main peut prendre une carte, soit celle la plus à gauche de la rangée, soit celle la plus à droite. L'objectif est de ramasser la valeur totale maximale. L'objectif est de déterminer si il y a une stratégie qui permet d'optimiser ce gain.

On peut supposer qu'au départ le nombre de cartes est pair pour que chaque joueur ait le même nombre de cartes à prendre mais cela ne change pas le problème.

Q 1. Une stratégie gloutonne pourrait être de prendre à chaque tour la carte de plus grande valeur parmi les deux cartes possibles. Que pensez-vous de cette stratégie? Vous pouvez illustrer votre réponse d'un exemple.

On voudrait donc un algorithme qui étant donné un tirage de cartes produise la stratégie optimale pour le premier joueur.

Dans un premier temps, on se contentera de calculer le gain maximal possible pour un joueur, en **supposant que son adversaire joue selon la stratégie optimale**.

Une configuration du jeu, pour un tirage donné, sera représentée par un couple (i, j) qui indique que les cartes restantes sont les cartes $c_i, \ldots, c_j$, avec $0 \leq i \leq j$.

Le problème, appelé *GainMaximal* est donc:

Entrée:

n – entier >0 le nombre de cartes

$v_0, \ldots, v_{n-1}, v_n$ – les valeurs des cartes dans l'ordre de la rangée..

Sortie: le gain maximal du joueur qui a la main en premier (toujours en supposant que son adversaire adopte aussi la stratégie optimale).

On notera $G(i, j)$ le gain maximal pour le joueur qui a la main, à partir de la configuration (i, j).

Par exemple, supposons que $n = 5$ et que les valeurs des cartes dans l'ordre de la rangée soient $v_0 = 7, v_1 = 2, v_2 = 8, v_3 = 1, v_4 = 2$. La table des $G(i, j)$ est donnée par:

$i \backslash j$	0	1	2	3	4	5
0	7	7	10	15	10	17
1		2	8	3	10	6
2			8	8	9	10
3				1	2	4
4					2	3
5						3

Q 2. Que vaut $G(i, i)$?

Q 3. Soit $Total(i, j)$ la somme des valeurs des cartes de i à j, c.à.d. $Tot(i, j) = \sum_{k=i}^{k=j} v_k$;
Exprimer $G(i, j)$ en fonction de $G(i + 1, j), G(i, j - 1)$ et de $Total(i, j)$.

Q 4. Proposer un algorithme naïf pour calculer $G(i, j)$. Quelle est sa complexité?

Q 5. Proposer un algorithme en $O(n^2)$ pour le problème *GainMaximal*. Un algorithme en $O(n^3)$ donnera une partie des points.

Q 6. Proposer un algorithme polynomial pour calculer la stratégie optimale. Par exemple, l'algorithme pourrait sortir sur l'exemple ci-dessus cette table, qui donne une configuration (i, j), l'action à faire: G pour prendre la carte à gauche, D pour prendre celle à droite.

$i\backslash j$	0	1	2	3	4	5
0	G	G	D	G	G	G
1		G	D	G	D	G
2			G	G	G	G
3				G	D	D
4					G	D
5						G

Remarque: dans la table, on a choisi arbitrairement de privilégier G quand G et D étaient toutes les deux optimales. Un autre choix peut être fait!

Exercice 3 : Paquets logiciels

Préliminaire: Cet exercice est inspiré de la problématique d'installation des paquets logiciels et en particulier d'une infime partie des travaux de L'Initiative pour la Recherche et l'Innovation sur le Logiciel Libre, l'IRILL: c'est une version très simplifiée et schématique de la problématique. Si la thématique vous intéresse -ce que j'espère!-, je vous invite à aller voir le site de l'IRILL, http://www.irill.org/.

On supposera dans l'exercice qu'un paquet est identifié par un numéro, un entier positif.

Le cadre général: Pour un paquet logiciel P on connait:

- l'ensemble des paquets avec qui il est en conflit, c.à.d. qui ne peuvent être installés en même temps que lui.
- un ensemble de dépendances qui est un **ensemble d'ensembles** de paquets: si les dépendances de P sont $\{E_1, \ldots, E_k\}$, cela signifie que pour une installation réussie de P, il faut pour chaque i, qu'au moins un paquet de E_i soit installé. Par exemple, si les dépendances sont $\{\{P_1, P_4\}, \{P_2\}, \{P_1, P_3\}$, avec P_1, P_2 ou P_4, P_2, P_3 suffisent pour installer le paquet, P_4, P_2 ne suffisent pas.

Ces informations sur le paquet P sont incluses dans le paquet P. On supposera dans l'exercice qu'un paquet est limité à ces informations: un paquet sera donc la donnée d'un triplet $(Id, Conf, Dep)$ avec Id son identifiant, $Conf$ la donnée de l'ensemble des identifiants des paquets en conflit avec P et l'ensemble des dépendances de P. La taille d'un paquet est donc la taille de sa représentation.

Par exemple $(2, \{1, 7\}, (\{3, 4\}, \{5\})$ correspond au paquet de numéro 2, qui est en conflit avec les paquets 1 et 7 et qui nécessite d'une part le paquet 3 ou le paquet 4, d'autre part le paquet 5.

On supposera disposer des primitives sur les ensembles (appartenance d'un élément à un ensemble, intersection, union, complémentaire, ...).

Une **installation** I est juste un ensemble de paquets. On dit qu'elle est **correcte** si elle respecte les contraintes de conflit - elle ne contient pas deux paquets en conflit-, et de dépendance - pour tout paquet de I, les contraintes de dépendances sont vérifiées: si les dépendances du paquet sont $\{E_1, \ldots, E_k\}$, $E_i \cap I$ est non vide pour tout i, $1 \leq i \leq k$. On va étudier la complexité de différents problèmes liés à l'installation de paquets.

Q 1. *Installation immédiate*

Soit le problème suivant:

Entrée: un paquet $P = (id, Conf, Dep)$, une installation correcte I.

Sortie: Oui, Ssi P peut être installé directement, i.e. $I \cup P$ est-elle correcte. Non , sinon.

On supposera pour simplifier que les conflits sont donnés explicitement de façon symétrique: si le paquet Q se dit en conflit avec le paquet R, le paquet R se dit en conflit avec le paquet Q.

Montrer que le problème est P.

Q 2. *Installation de k paquets (problème pas très réaliste)*

Soit maintenant le problème de décision suivant $k - Inst$:

Entrée:

n, entier positif

$P_1, \ldots, P_n$, n paquets.

k, entier , $0 \leq k \leq n$

Sortie: Oui, Ssi on peut choisir k paquets distincts parmi $P_1, ..., P_n$ qui forment une installation correcte.

Q 2.1. Montrer que le problème $k - Inst$ est NP.

Q 2.2. Rappel: *Independent* est le problème donné par:

Entrée:

$G = (S, A)$ - -un graphe

k - -un entier

Sortie:

Oui, si il existe un sous-ensemble indépendant de cardinal k, i.e. une partie I de S de cardinal k telle qu'aucun arc ne relie deux sommets de I ($(I \times I) \cap A = \emptyset$).

Montrer qu'*Independent* se réduit polynomialement dans le problème $k - Inst$.

Qu'en déduire?

Q 3. *Installation possible?*

Soit maintenant le problème de décision *Poss* suivant:

Entrée: un ensemble de paquets disponibles *Dispo*, un paquet P de *Dispo*

Sortie: Oui, Ssi il existe une installation correcte de P, i.e. $I \subset Dispo$ qui soit correcte et qui contienne P.

Q 3.1. Montrer que le problème *Poss* est NP.

Q 3.2. Supposons qu'il n'y ait aucun conflit, juste des dépendances. Quelle est la complexité de la restriction du problème *Poss* à ce cas?

Q 3.3. Montrer qu'*Independent* se réduit polynomialement dans le problème *Poss*.

Qu'en déduire pour la complexité du problème *Poss*?

Exercice 4 :

Le problème est le suivant: on a n objets de valeurs respectives $v_0, v_1, \ldots, v_{n-1}$. On veut constituer avec ces objets le maximum de lots de valeur au moins V. Formellement, une solution est donc: $aff : [0..n-1] \to [0..k-1]$ tel que $\sum_{aff(i)=j} v_i \geq V$, pour tout j, $0 \leq j < k$. Le nombre de lots, k, est donc la fonction objectif à maximiser.

Soit donc le problème d'optimisation *Opt*:

Entrée:

n, entier - le nombre d'objets

$v_0, v_1, \ldots, v_{n-1}$, n entiers - les valeurs des objets

V, entier - la valeur minimale d'un lot

Sortie:

k et une affectation $aff : [0..n-1] \to [0..k-1]$ telle que $\sum_{aff(i)=j} v_i \geq V$, pour tout j, $0 \leq j < k$ et k soit maximal.

Q 1. Proposer un algorithme d'approximation pour *Opt* qui offre une garantie 2 et prouver la garantie.

Les solutions avec garantie strictement supérieure à 2 ou/et sans preuve de la garantie donneront lieu à une partie des points.

Q 2. Que pensez-vous de la complexité d'*Opt*? Justifiez brièvement.