Université Lille1 Sciences et Technologies
UFR IEEA

Examen AAC 1ère session
Documents autorisés

3 janvier
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 $n^4+\sqrt{n}, n+\log n, n^3, n^2+n, \log(n^2), n\log n, 2^n, \log n+\sqrt{n}$ 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 $\Theta(\log n)$, $\Theta(n)$, $\Theta(\sqrt{n})$, $\Theta(2^n)$ ou $\Theta(n^2)$, en justifiant très brièvement:

```
int T1(int n){
  if (n <= 0) return 1;
  else return 2*T1(n-2);}

int T2(int n){
  int c=0;
  for (int i=1;4*i<n;i++) c++;
  return c;}

int T3(int n){
  int c=0;
  for (int i=n;i>0;i--)
       for (int j=1;j+i<n;j++) c++;
  return c;}
```

```
int T4(int n){
  if (n <= 0) return 1;
  else return 1+ 3*T4 (n/4);}

int T5(int n){
  if (n <= 0) return 1;
  else return T5(n/4)+T5(n/4);}

int T6(int n){
  if (n <= 0) return 1;
  else return T6(n-1)+T6(n-1);}
```

Q 3. *La classe NP*

Soient P_1, P_2, Q des problèmes de décision. Supposons qu'on sache que P_1 est NP-complet et P_2 NP. Pour chacune des affirmations suivantes, dire si elle est "Toujours Vraie" ou non (pas de justification demandée):

(Af1) P_1 se réduit en P_2.

(Af2) P_2 se réduit en P_1.

(Af3) Si Q se réduit polynômialement en P_1, Q est $NP-$dur.

(Af4) Si P_1 se réduit polynômialement en Q, Q est $NP-$dur.

(Af5) Si P_1 se réduit polynômialement en Q, Q est $NP-$complet.

(Af6) Si Q se réduit polynômialement en P_1, Q est NP.

(Af7) Si P_1 se réduit polynômialement en Q, Q est $NP-$complet.

(Af8) Si Q se réduit polynômialement en P_2, Q se réduit polynômialement en P_1.

Q 4. Pour chaque affirmation suivante, dire si elle est vraie ou fausse. Justifier brièvement.

1. Tout langage algébrique peut être décidé par une Machine de Turing.
2. L'ensemble des programmes Java syntaxiquement corrects est récursif.
3. L'ensemble des programmes Java syntaxiquement corrects s'exécutant en temps fini est récursif.
4. Une propriété NP est décidable.

Exercice 2 : Se rapprocher de la cible

Soit un tableau T de n entiers triés dans l'ordre croissant et *cible* un entier: on recherche dans le tableau T la valeur la plus proche de *cible* (si il en existe deux, on pourra choisir l'une ou l'autre).

Par exemple, si $n=6$ et $T[0]=1, T[1]=6, T[2]=12, T[3]=16, T[4]=19, T[5]=24$, alors pour $cible=7$, on retournera 6, pour $cible=9$, on pourra retourner 6 ou 12, pour $cible=25$, on retournera 24.

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 : Motifs

Le problème: On cherche à vérifier qu'un mot est bien filtré par un motif très simple. Plus précisément, un mot sera une suite finie de lettres d'un alphabet donné. Un motif sera un mot sur le même alphabet enrichi de trois symboles ("wildcards"), ?, + et $*$.

Le mot u filtre le motif p si il peut être obtenu à partir de p en remplaçant chaque ? par une lettre, chaque $*$ par un mot quelconque (éventuellement vide), chaque + par un mot quelconque non vide.

Par exemple, *abba* est filtré par les motifs $*$, $a+$, $*a$, $*b$,$abb*a$, $a*b*a$, $a*b*b*a*$, mais n'est pas filtré par les motifs $+a+$, $*b+b*\ a*b$, $b*$, $*bbb*a$.

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

Q 1. *abba* est-il filtré par $*a*$? par $+a+$? par $+b+$? par $a+a$?

Q 2. Proposer un algorithme linéaire, i.e. en $O(|u|+|p|)$ [1], pour le cas où le motif ne contient ni $*$, ni $+$:

Entrée: un mot u sur Σ et un motif p sur $\Sigma \cup \{?\}$

Sortie: Oui si u est filtré par p , non sinon.

Q 3.

Q 3.1. Proposer un algorithme en $O(|u|*|p|)$ dans le cas général:

Entrée: un mot u sur Σ et un motif p sur $\Sigma \cup \{?,*,+\}$

Sortie: Oui si u est filtré par p, non sinon.

Aide: Il existe plusieurs approches efficaces et "classiques" du problème et vous pouvez choisir celle qui vous convient. Une méthode possible et simple utilise la programmation dynamique: dans ce cas, exprimez, par exemple, $filtre(xu, yp)$ en fonction de $filtre(u,p)$, $filtre(xu,p)$, $filtre(u,yp)$, x étant une lettre de Σ, y étant une lettre de $\Sigma \cup \{?,*,+\}$.

Q 3.2. L'algorithme que vous avez conçu se comporte-t-il en $O(n)$ si le motif ne comporte ni *, ni +? Si non, comment le modifieriez-vous pour que ce soit le cas? Dans ce cas , à quoi correspond le pire des cas pour votre algorithme modifié?

Q 4. On cherche maintenant à faire du filtrage de motifs (pattern-matching) "approximatif", c'est à dire qu'on peut admettre des erreurs. Par exemple, `examen` est filtré par `e?men`, `x*n` ou encore `*ae*`, si on admet une erreur de 1. Si on admet une erreur de 2, `examen` est filtré par `x*e`, `amen` ou `*an`.

[1] $|u|$ (resp. $|p|$) désigne la longueur de u (resp. p).

Formellement, soit u un mot sur Σ, p un motif sur $\Sigma \cup \{?, +, *\}$.

$filtrapp(u, p)$ est le plus petit entier k tel qu'il existe un mot v à distance k de u tel que $filtre(v, p)$.

La distance de deux mots u et v est la distance de Levensthein, i.e. le nombre minimal de suppressions, insertions, modifications de caractères que l'on doit faire pour passer de u à v.

Par exemple, $filtrapp(examen, *) = 0$, $filtrapp(examen, e?men) = 1$, $filtrapp(examen, x * e) = 2$, $filtrapp(examen, *abc*) = 2$.

Q 4.1. Que vaut $filtrapp(examen, ???)$? $filtrapp(examen, +e)$? $filtrapp(examen, *z*)$? $filtrapp(examen, *z)$? $filtrapp(examen, z)$?

Q 4.2. Proposer un algorithme efficace pour calculer $filtrapp(u, p)$.

Exercice 4 : Terminer le plus tôt possible

On doit exécuter n tâches. Chacune de ces tâches se décompose en deux parties, la première devant être exécutée sur une machine A, la deuxième sur une machine de type B. La deuxième ne peut être exécutée que si la première est terminée. On dispose d'une seule machine de type A -par exemple, un supercalculateur- mais de n machines de type B.

Pour chaque tâche i, on connaît la durée des deux parties p_i, d_i; on cherche à trouver l'ordre d'exécution des tâches pour finir les tâches au plus tôt. On suppose qu'on peut lancer la première exécution au temps 0.

Par exemple, soient la tâche 1 avec $p_1 = 5, d_1 = 7$ et la tâche 2 avec $p_2 = 6, d_2 = 8$. Si on exécute d'abord la 1 puis la 2, il faudra une durée totale de 19:

- Machine A: tâche 1 lancée au temps 0, terminée au temps 5 (i.e. à la fin du temps 4), tâche 2 lancée au temps 5, terminée au temps 11
- Machine B_1: tâche 1 lancée au temps 5, terminée au temps 12.
- Machine B_2: tâche 2 lancée au temps 11, terminée à 19.

Si on exécute d'abord la 1 puis la 2, il faudra une durée totale de 18:

- Machine A: tâche 2 lancée au temps 0, terminée au temps 6, tâche 1 lancée au temps 6, terminée au temps 11.
- Machine B_1: tâche 2 lancée au temps 6, terminée au temps 14.
- Machine B_2: tâche 1 lancée au temps 11, terminée à 18.

Donc, dans ce cas il, est préférable de lancer la tâche 2 avant la tâche 1.

Q 1. Soient la tâche 1 avec $p_1 = 5, d_1 = 8$ et la tâche 2 avec $p_2 = 6, d_2 = 4$. Quel est le meilleur ordre d'exécution?

Q 2. Proposer un algorithme en $O(n \log n)$ pour résoudre le problème. Justifier qu'il est correct, i.e. qu'il produit une solution optimale.

On supposera qu'on dispose par exemple d'un tableau des tâches, et pour chaque tâche i d'un accesseur à p_i, d_i.

Exercice 5 : Répartition

Le problème consiste à répartir n étudiants en au plus k groupes, chaque groupe ne comportant que des amis. Chaque étudiant a donné a priori la liste de ses amis (remarque: la relation "être ami" n'est pas nécessairement transitive -un ami d'un ami n'est pas nécessairement un ami- ni même symétrique -ce qui est plus discutable-).

Formellement, le problème de décision *Repart* est défini par:

Entrée :

n, un nombre d'étudiants

k, un nombre de groupes

n listes $A_1, ...A_n$, la liste A_i donnant les amis de l'étudiant i.

Sortie :

Oui si il existe une répartition en k groupes, rep qui associe à chaque étudiant i un numéro de groupe $rep(i)$ de 1 à k, tel que si $aff(i) = aff(j)$ pour deux étudiants distincts i et j, i est dans A_j et j est dans A_i.

Non, sinon.

Q 1. Montrer que *Repart* est NP.

Q 2. *Rappel: Col*, le problème de coloriage d'un graphe est $NP-$complet. Il est défini par:

Entrée : $G = (S, A)$ un graphe non-orienté, k un entier

Sortie : Oui, si il existe un coloriage correct de G en k couleurs, i.e. une application $coul : S \to [1..k]$ telle que $\forall x, y \in S,\ (x, y) \in A \Longrightarrow coul(x) \neq coul(y)$.

Non, sinon.

Montrer que *Col* se réduit en *Repart*. Qu'en déduire pour *Repart*?

Q 3. Que pensez-vous du problème si on veut répartir en 2 groupes? en 3 groupes?

Q 4. On suppose maintenant que la relation "être ami" est symétrique -si X est ami de Y, Y est ami de X- et transitive: si X est ami de Y et Y ami de Z, X est ami de Z. Que devient la complexité du problème?

Bonne Année 2011!