Université Lille1 Sciences et Technologies
UFR IEEA

Examen AAC 2ème session
Documents autorisés

2009–2010
M1 Informatique
23 février 2010

Exercice 1 : Compétences de base

Q 1. Trier les fonctions $6^n + n^2, n^3, n^2 + n, n + 5\log n, n^2 \log n, 2^n, \sqrt{n}, \log n$ 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 $\Theta(\log n)$, $\Theta(\sqrt(n))$, $\Theta(n)$ ou $\Theta(n^2)$, en justifiant très brièvement (en supposant qu'on est dans le cadre du coût uniforme, i.e. en ne prenant pas en compte la taille des opérandes):

```
int T1(int n){
  if (n <= 0) return 10;
  else return (10 + T1 (n-7));}
```

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

```
int T3(int n){
  if (n <= 0) return 1;
  else return (1+ T3(n/5) +T3(5));}
```

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

Q 3.*Le BABA de la programmation dynamique*
Soit la suite $T(n)$ définie par $T(1) = T(0) = 1$ et pour $n > 1$, $T(n) = T(n-1) - T(n-2)$ si $T(n-1) > T(n-2)$, $T(n) = T(n-1) + T(n-2)$ sinon.
Pour calculer $T(n)$ pour l'entrée n, on propose l'algorithme naïf récursif:

```
function T(n:in NATURAL) return NATURAL is
begin
  if n<2 then return 1;
  else if T(n-1)>T(n-2) return T(n-1)-T(n-2);
  else return T(n-1)+T(n-2);
  end if;
end T;
```

Q 3.1. Soit $A(n)$ le nombre d'appels récursifs à T effectués lors de l'exécution de la function $T(n)$. Exprimez $A(n)$ en fonction de $A(n-1)$ et $A(n-2)$. Qu'en déduire sur la complexité de l'algorithme naïf ?
Q 3.2. Proposez un algorithme en $O(n)$ (en supposant qu'on est dans le cadre du coût uniforme, i.e. en ne prenant pas en compte la taille des opérandes).
Q 4. A quoi sert de montrer qu'un problème est NP-dur?
Q 5. 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'union d'un langage récursif et d'un langage algébrique est un langage récursif.
3. Le complémentaire d'un langage algébrique est récursif.
4. Le complémentaire d'un langage algébrique est récursivement énumérable.

Exercice 2 : Ordonnancement

Les trois questions de cet exercice sont indépendantes.

On a un ensemble de n tâches. Pour chaque tâche i, on connaît la date à laquelle elle est disponible, $disp_i$, celle à laquelle elle doit être finie $dlim_i$ et sa durée dur_i. L'objectif est de savoir si on peut réaliser toutes les tâches: une seule tâche peut être exécutée à la fois, il n'y a pas de préemption: une fois une tâche commencée, elle ne peut être interrompue;

Un ordonnancement consiste donc à attribuer à chaque tâche une date de début deb_i; il est correct si:

1. $disp_i \leq deb_i$: une tâche n'est programmée que quand elle est disponible.
2. $deb_i + dur_i \leq dlim_i$ pour tout i: chaque tâche est terminée à temps.
3. $deb_i + dur_i \leq deb_j$ ou $deb_j + dur_j \leq deb_i$ pour tout i, j tels que $i \neq j$: deux tâches ne sont jamais effectuées simultanément;

Exemple: Pour $n = 4$, et:
$disp_1 = 1, dlim_1 = 6, dur_1 = 3, \quad disp_2 = 2, dlim_2 = 4, dur_2 = 1,$
$disp_3 = 1, dlim_3 = 8, dur_3 = 1, \quad disp_4 = 1, dlim_4 = 8, dur_4 = 2,$
un ordonnancement correct est $deb_1 = 3, deb_2 = 2, deb_3 = 1, deb_4 = 6$
Formellement, le problème de décision associé, noté Ord, est donc:
Entrée:
n un entier, le nombre de tâches
$disp_1, ..., disp_n$, n entiers positifs ou nuls: les dates de disponibilité des tâches
$dlim_1, ..., dlim_n$, n entiers positifs: les dates limites pour chaque tâche
$dur_1, ..., dur_n$, n entiers positifs: les durées de chaque tâche
Sortie: Oui Ssi il existe un ordonnancement correct, non sinon.

Dans tout l'exercice, on pourra supposer que les $disp_i$, $dlim_i$ et dur_i sont données dans une table ou un tableau.

Q 1. *Le cas général*

Q 1.1. Existe-t-il un ordonnancement correct pour $n = 4$ et:
$disp_1 = 1, dlim_1 = 8, dur_1 = 2, \quad disp_2 = 2, dlim_2 = 3, dur_2 = 1$
$disp_3 = 4, dlim_3 = 6, dur_3 = 1, \quad disp_4 = 3, dlim_4 = 6, dur_4 = 2$
Si oui, lequel? sinon, pourquoi?

Q 1.2. Existe-t-il un ordonnancement correct pour $n = 4$ et:
$disp_1 = 1, dlim_1 = 6, dur_1 = 2, \quad disp_2 = 1, dlim_2 = 3, dur_2 = 2$
$disp_3 = 4, dlim_3 = 6, dur_3 = 1, \quad disp_4 = 3, dlim_4 = 6, dur_4 = 2$
Si oui, lequel? sinon, pourquoi?

Q 1.3. Montrer que Ord est NP.

Q 1.4. *Rappel:* le problème de décision $NP-$complet Sum est défini par:
Donnée:
n –un nombre d'entiers
$x_1, \cdots, x_n$ –les entiers
c –un entier cible
Sortie: Oui, si il existe un sous-ensemble de $[1..n]$ tel que la somme des x_i correspondants soit exactement c, i.e. $J \subset [1..n]$, tel que $\sum_{i \in J} x_i = c$

Montrer que Sum se réduit polynômialement en Ord. Qu'en déduire pour Ord?
Aide: on pourra associer à chaque entier une tâche, et définir une tâche supplémentaire de durée 1.

Q 2. *Deux cas particuliers*

Q 2.1. Supposons maintenant que toutes les tâches ont la même date limite: proposer un algorithme glouton exact en $O(n \log n)$ pour le problème. Justifier que votre algorithme est correct.

Q 2.2. Supposons que les tâches peuvent avoir des dates limites différentes mais ont toutes la même date de disponibilité. Que pensez-vous du problème?

Exercice 3 : Partage

Soit un ensemble de n objets et leurs poids $c_1, \ldots, c_n$ (ce sont des entiers). Le problème est de partager en deux cet ensemble de façon la plus équitable possible. Un partage est défini par une partition de $[1..n]$ en deux ensembles A_1, A_2. A un partage, on associe un coût: $max(\sum_{i\in A_1} c_i, \sum_{i\in A_2} c_i)$. Plus cette valeur est petite, plus le partage est équilibré. On peut remarquer que le coût est toujours supérieur ou égal à $1/2 * \sum_{1\leq i\leq n} c_i$ et inférieur ou égal à $\sum_{1\leq i\leq n} c_i$.

Par exemple si $n = 4, c_1 = 5, c_2 = 8, c_3 = 7, c_4 = 3$, un partage optimal sera donné par $A_1 = \{1, 3\}, A_2 = \{2, 4\}$ et son coût est 12. Si $n = 3, c_1 = 3, c_2 = 15, c_3 = 7$, un partage optimal sera donné par $A_1 = \{1, 3\}, A_2 = \{2\}$ et son coût est 15.

On a donc le problème d'optimisation suivant:

Entrée n –un entier $c_1, \ldots, c_n$ – n entiers

Sortie Un partage optimum, i.e. A_1, A_2 deux parties de $[1..n]$ telles que $A_1 \cup A_2 = [1..n], A_1 \cap A_2 = \emptyset$, qui minimise

$$max(\sum_{i\in A_1} c_i, \sum_{i\in A_2} c_i)$$

Q 1. Soit $n = 5, c_1 = 12, c_2 = 5, c_3 = 6, c_4 = 8, c_5 = 11$. Donner un partage optimal.

Q 2. Pour n objets, combien y a-t-il de partages possibles?

Pour équilibrer, l'heuristique suivante est proposée:

```
debut
  trier les c_i par valeurs décroissantes
  //on a maintenant c_1 ≥ c_2 ≥ ... c_n ...
  A_1 := ∅; v_1 := 0;
  A_2 := ∅; v_2 := 0;
  pour i de 2 à n faire
    si v_1 > v_2
    alors A_2 := A_2 ∪ {i}; v_2 := v_2 + c_i;
    // on ajoute l'objet i à A_2.
    sinon A_1 := A_1 ∪ {i}; v_1 := v_1 + c_i;
    // on ajoute l'objet i à A_1.
    fin si;
  fin pour;
 fin
```

Soit (A_1, A_2) le partage produit par l'heuristique ci-dessus sur une donnée $c_1, c_2, \ldots, c_n$. On va étudier la qualité de cette solution. On notera A_l le plus lourd de A_1, A_2 i.e. $A_l = A_1$ si $\sum_{i\in A_1} c_i \geq \sum_{i\in A_2} c_i$, $A_l = A_2$ sinon.

Q 3. Quelle est la complexité de l'heuristique?

Q 4. Supposons que l'heuristique n'a placé qu'un seul élément dans A_l. Montrer qu'alors la solution obtenue par l'heuristique est optimale.

Q 5. Supposons maintenant que A_l contienne au moins deux éléments. Soit le dernier élément placé dans A_l par l'heuristique. On notera p son poids.

Q 5.1. Montrer que le dernier élément placé dans A_l par l'heuristique est de poids au plus un tiers du poids total des objets, i.e. $p \leq 1/3 * \sum_{1\leq i\leq n} c_i$.

Q 5.2. On notera *tot* le poids total, i.e. $tot = \sum_{1\leq i\leq n} c_i$.

Montrer que $tot \leq 2 * \sum_{i\in A_l} c_i \leq tot + p$.

Q 6. Déduire de ce qui précède que, si *opt* est le coût optimal d'un partage et *heur* celui du partage produit par l'heuristique, on a:

$$opt \leq heur \leq 4/3 * opt$$

Q 7. Pour améliorer cette heuristique, on propose la méthode suivante:

- On trie toujours par c_i décroissants.
- On partage de façon équitable les 4 premiers objets dans A_1, A_2, par exemple, en essayant tous les partages.
- Enfin, on finit le partage comme dans l'heuristique précédente.

Montrer qu'alors on aura:

$$opt \leq heur \leq 6/5 * opt$$

Q 8. Soit k un entier quelconque fixé. Comment modifier l'heuristique pour avoir:

$$opt \leq heur \leq (1 + 1/k) * opt$$

L'heuristique reste-t-elle polynômiale? Peut-on en déduire qu'il existe un algorithme polynômial qui donne toujours un partage optimal?