Algorithmique

Examen du 10 novembre 2011

durée 3 heures

Le polycopié du cours est le seul document autorisé.

Les exercices sont indépendants.

Les procédures et fonctions devront être écrites en pseudo-code (pas en Java, Caml, ...)

Toutes les réponses devront être correctement justifiées. La rigueur des raisonnements, la clarté des explications, et la qualité de la présentation influeront sensiblement sur la note.

Le chiffre en regard d'une question est une indication sur sa difficulté ou sa longueur.

1 Belle impression

Le problème consiste à effectuer une belle impression d'un paragraphe par une imprimante. L'entrée est un texte de n mots de longueur $\ell_1, \ell_2, ..., \ell_n$. Le but est d'imprimer ce paragraphe de manière équilibrée sur un nombre de lignes à calculer qui contiendront au plus m caractères chacune. On suppose $0 < \ell_i \leq m$ pour tout $1 \leq i \leq n$.

Si une ligne contient les mots i à j et qu'on laisse exactement un espace entre deux mots, le nombre d'espaces nécessaires à la fin de la ligne est

$$m - \ell_i - \sum_{k=i+1}^{j} 1 + \ell_k$$

qui doit être positif ou nul pour que les mots tiennent sur la ligne. Une impression est valide si chaque ligne vérifie cette condition.

Le coût (ou équilibre) d'une impression valide est la somme des cubes des nombres d'espaces nécessaires à la fin des lignes, hormis la dernière.

L'objectif est de fournir une impression valide de coût minimal.

[1] **a)** Est-ce que l'algorithme glouton consistant à remplir les lignes une à une en mettant à chaque fois le maximum de mots possible sur la ligne en cours, fournit l'optimum ?

[4] **b)** L'objectif est de donner un algorithme de programmation dynamique pour résoudre ce problème en temps $\mathcal{O}(mn)$.

1. Définir les sous-problèmes et montrer la propriété de sous-structure optimale.
2. Écrire une fonction qui calcule le coût minimal du problème en temps $\mathcal{O}(mn)$.
   On mémorisera dans un tableau `C` les coûts optimaux des sous-problèmes.
3. En utilisant le tableau `C` ci-dessus, écrire une procédure qui imprime le texte de façon optimale en temps $\mathcal{O}(n)$.
   On considère que l'impression d'un mot se fait en temps $\mathcal{O}(1)$.

On suppose maintenant que la fonction de coût à minimiser est la somme des nombres d'espaces nécessaires à la fin des lignes, hormis la dernière.

[3] **c)** Écrire une procédure qui imprime le texte de façon optimale en temps $\mathcal{O}(n)$.

2 Arbres d'intervalles

Un *arbre d'intervalles* est un arbre binaire de recherche (ABR) tel que chaque nœud a contient 3 réels : $a.x \leq a.y \leq a.z$. On note $a.g$ et $a.d$ les sous-arbres gauche et droit du nœud a. L'arbre vide est noté `Null`.

L'intervalle (fermé) représenté par le nœud a est $[a.x, a.y]$ et la clé servant à ordonner l'ABR est $a.x$. Donc $a.g \neq$ `Null` implique $a.g.x \leq a.x$ et $a.d \neq$ `Null` implique $a.x \leq a.d.x$ (noter qu'il peut y avoir égalité et qu'il est possible qu'un même intervalle $[x', y']$ soit représenté par plusieurs nœuds d'un arbre d'intervalles).

Le troisième réel $a.z$ est le maximum des bornes supérieures des intervalles contenus dans le sous-arbre de racine a. En particulier, si a est une feuille alors $a.z = a.y$ et si les sous-arbres $a.g$ et $a.d$ sont non vides alors $a.z = \max(a.y, a.g.z, a.d.z)$.

On s'intéresse à la fonction de recherche suivante :

```
Procédure recherche (a:arbre; x', y':réels; b:arbre)
PRE : a est un arbre d'intervalles et x' ≤ y'
POST : (b ≠ Null et [x', y'] ∩ [b.x, b.y] ≠ ∅) ou
       (b = Null et [x', y'] ∩ [c.x, c.y] = ∅ pour tout nœud c de a)
Début
   b := a
   tant que b ≠ Null et [x', y'] ∩ [b.x, b.y] = ∅ faire
      si b.g ≠ Null et x' ≤ b.g.z alors b := b.g sinon b := b.d fsi
   ftq
Fin
```

Soit b un nœud d'un arbre d'intervalles et soient $x' \leq y'$ deux réels. On définit la formule

$$\varphi(b, x', y') ::= \exists c \text{ nœud du sous-arbre } b \text{ tel que } [x', y'] \cap [c.x, c.y] \neq \emptyset$$

[3] **a)** Montrer que le triplet de Hoare suivant est valide :

$$\{\varphi(b, x', y') \wedge [x', y'] \cap [b.x, b.y] = \emptyset\}$$
$$\texttt{si } b.g \neq \texttt{Null et } x' \leq b.g.z \texttt{ alors } b := b.g \texttt{ sinon } b := b.d \texttt{ fsi}$$
$$\{\varphi(b, x', y'\}$$

[2] **b)** Prouver à l'aide d'un invariant que la procédure `recherche` ci-dessus est correcte.

[1] **c)** Écrire une procédure pour insérer un intervalle $[x', y']$ dans un arbre d'intervalles a avec une complexité au pire $\mathcal{O}(h(a))$ où $h(a)$ est la hauteur de l'arbre a. Il faudra bien sûr maintenir les propriétés d'un arbre d'intervalles, mais il n'est pas nécessaire d'éviter que l'intervalle $[x', y']$ ne figure plusieurs fois dans l'arbre.

3 Ordonnancement de tâches

On considère un ensemble $S = \{1, \ldots, n\}$ de tâches, chacune nécessitant une unité de temps pour son exécution. Pour chaque tâche $i \in S$, on donne une date limite $d[i] \in \mathbb{N}\setminus\{0\}$ de fin d'exécution et une pénalité $p[i] > 0$ en cas de non respect de cette date limite. On suppose que les tâches sont exécutées à partir de l'instant 0.

Un ordonnancement est une permutation $\sigma \in \mathfrak{S}_n$ qui définit l'ordre $\sigma(1), \sigma(2), \ldots, \sigma(n)$ d'exécution des tâches. La date de fin d'exécution de la tâche i est donc $\sigma(i)$ et sa date limite est respectée si $\sigma(i) \leq d[i]$.

Le coût de l'ordonnancement $\sigma \in \mathfrak{S}_n$ est la somme des pénalités associées aux tâches dont la date limite n'est pas respectée. On cherche bien sûr à trouver un ordonnancement de coût minimal.

Un sous-ensemble $A \subseteq S$ de tâches est *réalisable* s'il existe un ordonnancement de A sans pénalité, i.e., si on peut ordonner les éléments de A en une suite $a_1, \ldots, a_k$ telle que $i \leq d[a_i]$ pour tout $1 \leq i \leq k$. On note I l'ensemble des parties réalisables de S.

[2] **a)** Soit $A \subseteq S$ et $a_1, \ldots, a_k$ un ordonnancement de A tel que $d[a_1] \leq d[a_2] \leq \cdots \leq d[a_k]$. Montrer que $A \in I$ si et seulement si $i \leq d[a_i]$ pour tout $1 \leq i \leq k$.

On décide de représenter une partie $A \in I$ par sa taille $k = |A|$ et un tableau a (de dimension n) qui contient les éléments de A rangés par dates limites croissantes : $A = \{a[1], \ldots, a[k]\}$ et $d[a[1]] \leq \cdots \leq d[a[k]]$.

[3] **b)** Écrire une fonction qui réalise la spécification ci-dessous en temps $\mathcal{O}(k)$.
Justifier la correction et la complexité de votre fonction.

```
Fonction testeAjoute (a:tableau; k,i:entiers) : booléen
PRE : k ≥ 0, A = {a[1],...,a[k]} ∈ I, d[a[1]] ≤ ··· ≤ d[a[k]], i ∈ S \ A
POST : retourne FAUX si A ∪ {i} ∉ I et dans ce cas a et k sont inchangés
       retourne VRAI si A ∪ {i} ∈ I et dans ce cas a et k sont modifiés
       pour qu'ils codent A ∪ {i}
```

[5] **c)** Montrer que (S, I) est un matroïde.

[3] **d)** En déduire un algorithme efficace pour trouver un ordonnancement de coût minimal. Donner la complexité (au pire) de votre algorithme.