Algorithmique TD6

Magistère Informatique 1ère Année

21 Octobre 2010

Exercice 1 *Arbres optimaux (Extrait Partiel 2007)*

On considère un ensemble de n clés $c_1 < c_2 < \cdots < c_n$. La clé c_i a la probabilité p_i d'être tirée, avec bien sûr $p_1 + \cdots + p_n = 1$. On veut construire un arbre binaire de recherche (ABR) qui contient ces n clés et qui optimise le temps moyen de recherche. Le temps de recherche d'une clé dans un ABR est le nombre de nœuds visités lors de cette recherche.

Soit t un ABR contenant les n clés. On note $\text{prof}_t(c_i)$ la profondeur du nœud de t qui contient la clé c_i. La profondeur de la racine est 1. Le temps moyen de recherche (coût) pour l'arbre t est donc :

$$\text{coût}(t) = \sum_{i=1}^{n} p_i \cdot \text{prof}_t(c_i).$$

Un arbre optimal pour $[p_1, \ldots, p_n]$ est un ABR de coût minimal.

(a) Pour $n = 3$, donner tous les ABR possibles et pour chacun d'eux le coût en fonction de p_1, p_2 et p_3. Application numérique : Calculer les coûts des ABR pour $[6, 3, 4]$.

(b) Un ABR de hauteur minimale est-il nécessairement optimal ? Prouver que oui ou donner un contre-exemple.

(c) On considère l'algorithme (glouton) suivant : on trie les clés par probabilités décroissantes et on construit un ABR, à partir de l'arbre vide, par insertions successives des clés par probabilités décroissantes (on utilise la procédure d'insertion usuelle, i.e., aux feuilles).
Donner un exemple dans lequel l'arbre obtenu par cet algorithme n'est pas de hauteur minimale mais dont le coût est inférieur aux coûts des arbres de hauteur minimale.
Cet algorithme donne-t-il nécessairement un arbre optimal ? Prouver que oui ou donner un contre-exemple.

Dans ce qui précède, nous avons uniquement considéré les recherches fructueuses. En général, il faut aussi considérer les recherches infructueuses. On note toujours p_i la probabilité de tirer la clé c_i. Pour $0 \leq i \leq n$, notons q_i la probabilité de tirer une clé dans l'intervalle $]c_i, c_{i+1}[$ (avec la convention $c_0 = -\infty$ et $c_{n+1} = +\infty$). Nous avons maintenant

$$p_1 + \cdots + p_n + q_0 + \cdots + q_n = 1.$$

Pour $0 \leq i \leq n$, on note d_i une clé factice prise dans l'intervalle $]c_i, c_{i+1}[$. La clé d_i représente une recherche infructueuse dans l'intervalle associé $]c_i, c_{i+1}[$.
Soit t un ABR dont les nœuds internes contiennent les n clés c_1, ..., c_n et dont les feuilles contiennent les $n+1$ clés factices d_0, ..., d_n. Le coût de la recherche de c_i dans t est toujours $\text{prof}_t(c_i)$. Le coût d'une recherche infructueuse dans l'intervalle $]c_i, c_{i+1}[$ est $\text{prof}_t(d_i) - 1$. Le -1 est dû au fait que l'arbre réel ne contient pas la clé factice d_i et que le nombre de nœuds visités lors de cette recherche infructueuse est donc la profondeur du père de d_i dans t. Finalement, le coût de l'arbre t est maintenant :

$$\text{coût}(t) = \sum_{i=1}^{n} p_i \cdot \text{prof}_t(c_i) + \sum_{i=0}^{n} q_i \cdot (\text{prof}_t(d_i) - 1)$$

et un arbre optimal pour $[p_1, \ldots, p_n \mid q_0, \ldots, q_n]$ est un ABR de coût minimal.

Dans cette définition, il est inutile de supposer que les p_i et q_i soient des probabilités, i.e., que $p_1 + \cdots + p_n + q_0 + \cdots + q_n = 1$. Dans la suite, on abandonne donc cette restriction et on parlera de *poids* au lieu de *probabilités* et on supposera les poids positifs ou nuls.

(d) Est-ce qu'un arbre optimal pour $[p_1, \ldots, p_n \mid q_0, \ldots, q_n]$ est aussi un arbre optimal pour $[q_0, p_1, q_1, \ldots, p_n, q_n]$, i.e., si on considère $d_0, \ldots, d_n$ comme de vraies clés ? Prouver que oui ou donner un contre-exemple.

(e) Montrer qu'il y a un unique arbre optimal pour $[1, \ldots, 1 \mid 0, \ldots, 0]$ lorsque $n = 2^k - 1$ $(k > 0)$ et calculer son coût.

Dans la suite, on fixe les poids $[p_1, \ldots, p_n \mid q_0, \ldots, q_n]$. Pour $0 \leq i \leq j \leq n$, on note $T(i, j)$ l'ensemble des ABR dont les nœuds internes contiennent les clés $c_{i+1}, \ldots, c_j$ et dont les feuilles contiennent les clés factices $d_i, \ldots, d_j$. On remarque que $T(i, i)$ est réduit à l'arbre contenant uniquement la clé factice d_i. On note aussi $\mathrm{Opt}(i, j)$ l'ensemble des arbres optimaux pour $[p_{i+1}, \ldots, p_j \mid q_i, \ldots, q_j]$, i.e., l'ensemble des arbres de coût minimal dans $T(i, j)$.

(f) On suppose $0 \leq i < j \leq n$. Soit $t \in \mathrm{Opt}(i, j)$ un arbre de racine c_k et de sous-arbres gauche et droit t_g et t_d. Montrer que $t_g \in \mathrm{Opt}(i, k-1)$ et $t_d \in \mathrm{Opt}(k, j)$.

Pour $0 \leq i \leq j \leq n$, on note $w(i, j)$ le poids total $p_{i+1} + \cdots + p_j + q_i + \cdots + q_j$, en particulier $w(i, i) = q_i$. Pour $0 \leq i \leq j \leq n$, on note $C(i, j)$ le coût optimal pour les poids $[p_{i+1}, \ldots, p_j \mid q_i, \ldots, q_j]$, i.e., le coût d'un arbre quelconque de $\mathrm{Opt}(i, j)$, en particulier $C(i, i) = 0$.

(g) Prouver que pour $i < j$ on a

$$C(i, j) = w(i, j) + \min_{i < k \leq j} C(i, k-1) + C(k, j).$$

(h) En déduire un algorithme permettant de calculer le tableau C des coûts optimaux en temps $\mathcal{O}(n^3)$. Prouver que votre algorithme réalise bien cette complexité.

(i) Écrire une fonction `arbreOpt` ayant deux paramètres entiers telle que `arbreOpt`(i, j) retourne un arbre de $\mathrm{Opt}(i, j)$.

La suite du problème vise à calculer les coûts optimaux en temps quadratique. Lorsque $0 \leq i < j \leq n$, on note $R(i, j)$ l'ensemble des entiers k tels que c_k est racine d'un arbre de $\mathrm{Opt}(i, j)$. Soient A et B deux ensembles d'entiers. On note $A \leq B$ si $\forall a \in A$, $\forall b \in B$, $b < a$ implique $b \in A$ et $a \in B$. Pour $1 < s \leq n$, on considère la propriété :

$$\forall 1 \leq i + 1 < j \leq n, \quad j - i \leq s \Longrightarrow R(i, j-1) \leq R(i, j) \leq R(i+1, j) \qquad P(s)$$

(j) Montrer $P(2)$.

Dans la suite, on suppose que $P(n)$ est vraie (la preuve de ce résultat n'est pas simple). Pour $0 \leq i < j \leq n$, on note $r(i, j) = \min R(i, j)$, i.e., $r(i, j)$ correspond à la racine minimale d'un arbre optimal pour $[p_{i+1}, \ldots, p_j \mid q_i, \ldots, q_j]$.

(k) Pour $1 \leq i + 1 < j \leq n$, montrer que $r(i, j)$ est l'entier k qui réalise le minimum de $C(i, k-1) + C(k, j)$ avec $r(i, j-1) \leq k \leq r(i+1, j)$.

(l) Soit $1 < s \leq n$. Montrer que

$$\sum_{i=0}^{n-s} 1 + r(i+1, i+s) - r(i, i+s-1) \leq 2n.$$

(m) Donner un algorithme *quadratique* qui calcule le tableau C des coûts optimaux et le tableau r des racines minimales des arbres optimaux. Prouver que l'algorithme est correct et qu'il est bien quadratique.

Quelques questions supplémentaires pour que personne ne s'ennuie.

(n) Caractériser les arbres optimaux pour $[1, \ldots, 1 \mid 0, \ldots, 0]$ lorsque $n > 0$ est arbitraire et calculer le coût optimal.

(o) Donner un arbre optimal pour $[5, 1, 1, \ldots, 1 \mid 0, \ldots, 0]$ lorsque $n = 40$ et calculer le coût optimal.

(p) Montrer $P(3)$.

(q) Montrer $P(n)$.

Exercice 2 *Codage de Huffman, Algorithmes Gloutons*

Soit Σ un alphabet fini et de cardinal supérieur ou égal à deux. On appelle mot de code toute suite finie de 0 et de 1, soit $\mathcal{C}$ l'ensemble des mots de code. On appelle codage binaire une application injective α de l'alphabet Σ dans $\mathcal{C}$. En utilisant l'opération concaténation, α s'étend de manière naturelle en :

$$\alpha : \Sigma^* \to \mathcal{C}$$

On dit qu'un codage est de longueur fixe quand toutes les lettres sont codées par un mot de code de même longueur. Un codage est dit préfixe si aucune lettre n'est codée par un mot de code qui est préfixe du codage d'une autre lettre.

1) Montrer que pour un codage de longueur fixe et un codage préfixe, α est injective sur Σ^*.

2) Montrer qu'on peut représenter un codage préfixe par un arbre binaire dont les feuilles sont les lettres de l'alphabet.

On considère un mot w. A chaque codage préfixe de w, représenté par un arbre T, est associé un coût défini par :

$$C_w(T) = \sum_{a \in \Sigma} f(a) l_T(a)$$

avec $f(a)$ le nombre d'occurences de a dans w et $l_T(a)$ la taille du mot de code associé à a par T. Un codage préfixe est dit optimal si il minimise la fonction C_w

3) Montrer qu'à un codage préfixe optimal correspond un arbre binaire où tout nœud interne a deux fils.

4) Montrer qu'il existe un codage préfixe optimal pour lequel les deux lettres dont le nombre d'occurences est le plus faible sont sœurs dans l'arbre.

Etant donnés x et y les deux lettres dont le nombre d'occurences est le plus faible dans w, on considère l'alphabet $\Sigma' = \Sigma - \{x, y\} + z$ où z est une nouvelle lettre à laquelle on associe $f(z) = f(x) + f(y)$.

5) Soit T' l'arbre d'un codage optimal pour Σ', montrer que l'arbre T obtenu à partir de T' en remplaçant la feuille associée à z par un nœud interne ayant x et y comme feuilles représente un codage optimal pour Σ.

6) En déduire un algorithme recherchant un codage optimal et donner sa complexité.