Algorithmique avancée – Master Informatique – UPMC 2010-11. Examen Réparti-1 – Eléments de Corrigé.

1 Arbres et tournois binômiaux [8 points]

Arbres binômiaux On étiquette l'arbre binômial B_k (formé de 2^k nœuds) en ordre postfixé, chaque étiquette (de 0 à $2^k - 1$) étant un mot binaire sur k bits.

Question 1. Arbre binômial B_k, étiquettes de 0 à $2^k - 1$ (sur k bits) en numérotation suffixe. Pour B_{k+1} on rassemble 2 arbres B_k : rajouter premier bit à 0 dans B_k de gauche et premier bit à 1 dans B_k de droite.

Question 2. Récurrence sur k : c'est vrai pour $k = 0$. Supposons vrai pour B_k, et soit x profondeur i dans B_{k+1}. Si $i = 0$ alors l'étiquette contient k fois 1. Sinon, si x appartient au B_k de gauche alors il a profondeur $i - 1$ dans ce B_k, et donc par hyp de rec, son nombre de 1 dans B_k est égal à $k - (i - 1)$, et comme dans la numérotation de B_{k+1}, x a un 0 supplémentaire, son nombre de 1 dans B_{k+1} est $k+1-i$. Et si x appartient au B_k de droite, il a aussi profondeur i dans ce B_k mais comme on rajoute 1 dans la numérotation, son nombre de 1 dans B_{k+1} est finalement $k + 1 - i$.
Le nombre de nœud s à profondeur i dans B_{k+1} : $\nu_{i,k} = \nu_{i,k-1} + \nu_{i-1,k-1}$, et $\nu_{0,k} = 1$, pour tout k. Donc $\nu_{i,k} = \binom{i}{k}$.
Le degré de la racine d'un nœud est égal au nombre de 1 à droite du 0 le plus à droite de son étiquette (et si l'étiquette ne contient pas de 0, le degré du nœud est égal au nombre de 1). Toujours vrai pour la racine (degre k et k bits à 1). Pour un nœud quelconque, preuve par récurrence sur k. Vrai pour $k = 0$. Supposons vrai pour B_k, et soit x un nœud , non racine, de B_{k+1} ; x a le mÃªme degrÃ© que dans B_k et la numérotation de x dans B_{k+1} est soit $0w$, soit $1w$, ou w est la numérotation de x dans B_k, donc OK.

Tournois binômiaux et files binômiales On considère maintenant des *tournois binômiaux* (arbres binômiaux dont les clés sont étiquetées de manière croissante sur chaque branche) et des files binômiales *élargies*, qui sont des files binômiales pouvant contenir un *nombre arbitraire de tournois de taille 1*. Dans une telle file binômiale élargie, l'ajout d'un élément se fait simplement en ajoutant un tournoi de taille 1 à la file, et ce n'est que lors d'une opération d'union ou de suppression du minimum que l'on consolide la structure, pour obtenir une file binômiale classique (au plus un tournoi de chaque taille).

Question 3. On represente une file binômiale élargie par une liste (circulaire) de tournois binômiaux, dont l'entrée est sur le tournois dont la racine contient le minimum de la file.
Pour le coût amorti, on prend comme fonction de potentiel le nombre d'arbres de la file ($\Phi(D_0) = 0$ et $\Phi(D_i) \geq 0, \forall i$). Et rappel de la formule du coût amorti : $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$. On compte le coût en nombre de comparaisons.
- Créer un tournoi ayant un unique sommet étiqueté i : évident et $\hat{c}_i = 1$
- Retourner l'élément de clé minimale d'une file binômiale élargie de n éléments : évident et $\hat{c}_i = c_i = 0$
- Insérer un sommet de clé i dans une file binômiale élargie de n éléments : on insère dans la file le tournoi composé de cet unique sommet, et on compare avec le min de la file, pour changer éventuellement le pointeur d'entrée dans la file. Coût amorti : $\hat{c}_i = 1$ (comp) +1(arbre supplementaire) = 2.
- Faire l'union d'une file binômiale élargie de m éléments avec une file de n éléments : fusion des 2 listes circulaires + consolidation pour obtenir une file binômiale. On a N et M arbres dans les files de départ et a l'arrivee on a P arbres pour la file binômiale, avec $P < \log_2(n + m)$. Pour consolider les $N + M$ arbres il faut au plus $N + M$ comparaisons (ce qui permet aussi de déterminer le min de la file). Coût amorti : $\hat{c}_i \leq N + M + P - (N + M) = P = 0(\log(n + m))$
- Supprimer l'élément de clé minimale d'une file binômiale élargie de n éléments (et N arbres) : on supprime le min, pour obtenir un ensemble de P arbres, avec $P \leq \log_2 n$, et ensuite on fait l'union avec la liste restante. Coût amorti : $\hat{c}_i = 0(\log(n))$.

2 Algorithme de Huffmann statique [4 points]

Question 1. Code1 : $\{0, 10, 11\}$: $f_a = 1/2, f_b = 1/4, f_c = 1/4$. Code2 : $\{0, 1, 00\}$ impossible pas prefixe. Code3 : $\{10, 01, 00\}$ pas complet donc pas minimal.

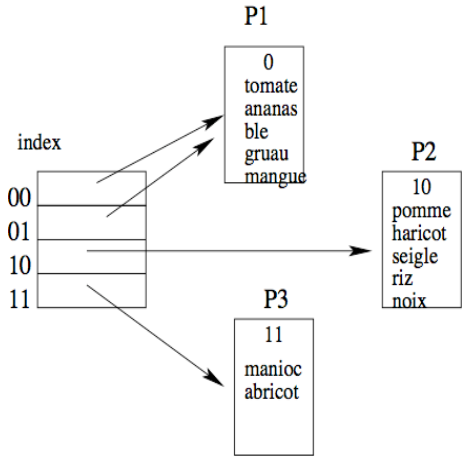
Question 2. Pour tout code de Huffmann statique : Si toutes les lettres du texte à coder ont une fréquence inférieure à 1/3, alors aucune lettre n'a un codage de longueur 1. Comme chaque lettre a une fréquence $< 1/3$, il y a au moins 4 lettres. Si "à la fin", il reste une lettre a de poids p et deux arbres T_1 et T_2 de poids $p_1 \geq p_2$ (T_1 est de hauteur au moins 1), on a $p < 1/3$ donc $p_1 + p_2 > 2/3$ et $p_1 > 1/3$. On rassemble donc a et T_2 en un arbre T puis T et T_1 (et c'est fini).

Question 3. Quelle est la plus grande longueur possible pour le codage d'une lettre, si l'on utilise un codage de Huffmann statique pour un texte de n lettres de fréquences $f_1, f_2, ..., f_n$? C'est $n-1$. Exemple : prendre $f_i = 2^{i-n}$, pour $i = 0..n-2$, et $f_{n-1} = 1/2 - 1/2^n$; on a $f_0 + ... + f_i = (2^{i+1} - 1)/2^n < f_{i+1}$, pour $i = 0..n-1$.

Question 4. Pour le texte $T = abracadabra$ sur 5 lettres, un codage de Huffmann statique donne $|TC| = 23$ (vu en cours), donc le nombre moyen de bits par lettre du texte est $23/11 = 2.09$ L'entropie de T est $\sum p_i \log_2(1/p_i) = 5/11 \log_2(11/5) + 2*2/11 \log_2(11/2) + +2*1/11 \log_2(11) = 2.04...$ Cette valeur est inférieure à la precedente, mais pas contradiction avec l'optimalité du codage de Huffmann statique, qui est optimal pour codage par "lettre".

3 Hachage extensible [8 points]

Dans la méthode de hachage extensible, on dispose d'une fonction de hachage qui associe à chaque élément une suite (non bornée) de bits. L'index T est une table de 2^d cases et l'adresse d'une case est une suite de d bits. Chaque case de T pointe sur une page pouvant contenir au plus b éléments. Chaque page P est caractérisée par une suite $s(P)$ de k bits, avec $k \leq d$: la page P contient tous les éléments dont la valeur de hachage commence par la suite $s(P)$. Et il y a 2^{d-k} cases de T qui pointent sur une même page P : toutes les cases dont l'adresse commence par $s(P)$ (si $k = d$, il y a une seule case de T qui pointe vers P : la case d'adresse $s(P)$).
Sur la figure ci-dessous, on a considéré les éléments suivants, avec leurs valeurs de hachage (sur 6 bits) : h(tomate) = 011101, h(ananas) = 001000, h(blé) = 001110, h(gruau) = 011001, h(mangue) = 000100, h(pomme) = 101110, h(haricot) = 101011, h(seigle) = 100010, h(riz) = 100110, h(noix) = 100101, h(manioc) = 111010, h(abricot) = 110111.
On suppose que la capacité des pages est 5. On considère la situation où la taille de l'index est 4. La page P_1 contient 5 éléments dont le premier bit de la valeur de hachage est 0, donc $s(P_1) = 0$ et les 2 cases dont l'adresse commence par 0 pointent sur P_1 : il y en a 2. La page P_2 contient 5 éléments, dont la valeur de hachage commence par les bits 1 et 0, donc $s(P_2) = 10$ et la case d'adresse 10 pointe sur P_2. La page P_3 contient 2 éléments, dont la valeur de hachage commence par les bits 1 et 1, donc $s(P_3) = 11$ et la case d'adresse 11 pointe sur P_3.



Pour insérer un élément x, on calcule l'adresse v obtenue en prenant les d premiers bits de la valeur de hachage de x. Soit P la page pointée à cette adresse.
- Si x est dans P il n'y a rien à faire.
- Si P n'est pas pleine on y insère x.
- Si P est pleine

– dans le cas où $v = s(P)$ il faut commencer par doubler la taille de l'index et mettre à jour les pointeurs vers les pages.
– et dans tous les cas, on éclate P en 2 pages P_1 et P_2 telles que $s(P_1) = s(P).0$ et $s(P_2) = s(P).1$, on répartit les éléments de P dans ces 2 pages et l'on met à jour les cases de T qui pointaient sur P.
– on insère x dans ce nouvel environnement

Question 1. Donner le résultat de l'insertion, dans la table de la figure précédente, d'un élément x_1 dont la valeur de hachage est h(x_1)=110011, puis d'un élément x_2 dont la valeur de hachage est h(x_2)=011110, et enfin d'un élément x_3 dont la valeur de hachage est h(x_3)=101001.

Question 2. Donner un ensemble de primitives, avec leurs spécifications, permettant de décrire les traitements de hachage extensible. (Il est conseillé de répondre à cette question en traitant la question suivante.)

Question 3. Écrire l'algorithme d'insertion d'un élément en utilisant cet ensemble de primitives.

Question 4. Expliquer les différents cas rencontrés pour la suppression d'un élément, et écrire l'algorithme de suppression en utilisant les primitives.