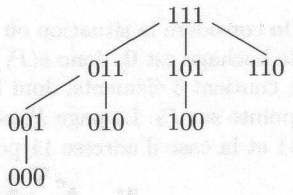
Algorithmique avancée – Examen Réparti 1
UPMC — Master d'Informatique —
Novembre 2010 – durée 2h

Les seuls documents autorisés sont les polys de cours, ainsi que la copie double personnelle.

1 Arbres et tournois binômiaux [7 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. Par exemple, pour B_3, on obtient :



Question 1. Dire comment on étiquette un arbre binômial B_{k+1} à partir des étiquetages des deux arbre binômiaux B_k qui le forment.

Question 2. Dans B_k, soit x un nœud à profondeur i, dont l'étiquette est e. Montrer que le nombre de 1 dans e est égal à $k - i$. Combien d'étiquettes de B_k contiennent exactement $k - i$ bits égaux à 1 ?
Montrer que le degré 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).

Tournois binômiaux et files binômiales On considère maintenant des *tournois binômiaux* (qui sont des arbres binômiaux dont les clés sont étiquetées de manière croissante sur chaque branche) et des files binômiales) (qui sont des ensembles de tournois binômiaux de tailles toutes différentes).

Question 3.
Montrer que l'on peut effectuer les opérations suivantes en temps amorti $O(1)$ (vous donnerez aussi les algorithmes)
- Créer un tournoi ayant un unique sommet étiqueté i.
- Retourner l'élément de clé minimale d'un tournoi binômial de n éléments.
- Insérer un sommet de clé i dans une file binômiale de n éléments.

Montrer que l'on peut effectuer les opérations suivantes en temps amorti $O(\log n)$ (vous donnerez aussi les algorithmes)
- Faire l'union de deux files binômiales ayant au total n éléments.
- Supprimer l'élément de clé minimale d'une file binômiale de n éléments.

Combien de comparaisons doit-on faire, au pire des cas, pour une union ou une suppression ?

2 Algorithme de Huffmann statique [6 points]

Les 4 questions de cet exercice sont indépendantes.

Question 1. On utilise l'algorithme de Huffmann statique pour coder un texte sur trois lettres $\{a, b, c\}$ de fréquences f_a, f_b, f_c. Dans chacun des 3 cas suivants, vous devez : ou bien donner un exemple de fréquences f_a, f_b, f_c à partir desquelles on obtient le code proposé, ou bien expliquer pourquoi il est impossible d'obtenir ce code : Code1 : $\{0, 10, 11\}$, Code2 : $\{0, 1, 00\}$, Code3 : $\{10, 01, 00\}$.

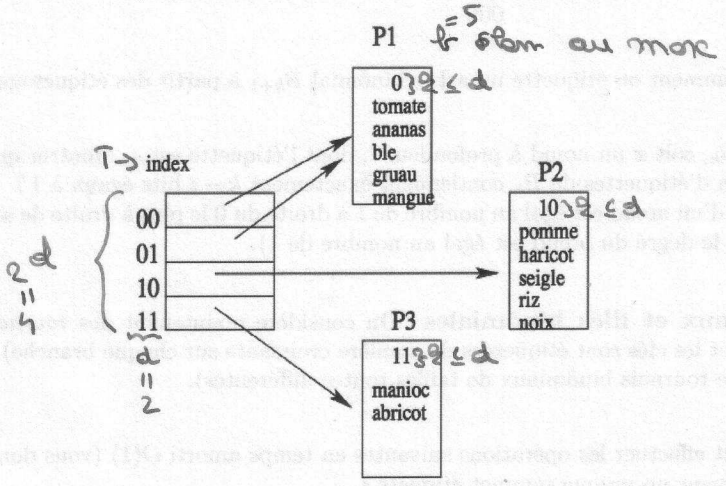
Question 2. Montrer que la propriété suivante est vraie pour tout code de Huffmann statique :
(P) Si toutes les lettres du texte à coder ont une fréquence inférieure à 1/3, alors aucune lettre n'a un codage de longueur 1.

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 $f_1, f_2, ..., f_n$? Donner un exemple.

Question 4. Pour le texte $T = abracadabra$ de 5 lettres, quel est le nombre moyen de bits par lettre dans un codage de Huffmann statique ? Quelle est l'entropie de T ? Comparer les 2 valeurs précédentes ; y a-t-il contradiction avec l'optimalité du codage de Huffmann statique ?

3 Hachage extensible [7 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.