Algorithmique avancée – Examen Réparti 1
UPMC — Master d'Informatique —
Novembre 2011 – durée 2h

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

1 Arbre et File binomiale [2 points]

Question 1. On considère un arbre binomial B tel que la racine possède 6 fils (c'est-à-dire qu'il y a 6 sommets à distance 1 de la racine dans B). Combien y a-t-il de sommets à distance 2 de la racine dans B ? Justifier votre réponse.

Question 2. Si une file binomiale classique (non relachée) F possède 107 éléments, de combien d'arbres binomiaux est-elle composée ? Quelles sont les tailles des différents arbres binomiaux qui la composent.
Même question (nombre d'arbres et tailles) lorsque l'on ajoute un élément à F.

2 Tris [6 points]

Le but de cet exercice est de proposer et comparer deux algorithmes de tri fondés l'un sur l'utilisation de files binomiales, et l'autre sur l'utilisation d'arbres 2-3-4. On suppose que l'on a n éléments, 2 à 2 comparables.

Question 1. Ecrire un algorithme de tri de n éléments, en utilisant la structure de données "file binomiale", et les primitives associées.
Quelle est la complexité de cet algorithme, en nombre de comparaisons entre éléments, dans le pire des cas.

Question 2. Ecrire un algorithme de tri de n éléments, en utilisant la structure de données "arbre 2-3-4", et les primitives associées.
Quelle est la complexité de cet algorithme, en nombre de comparaisons entre éléments, dans le pire des cas.

Question 3. Comparer les 2 algorithmes précédents : complexité en temps et en mémoire, ainsi que contexte d'utilisation.

3 Coût amorti [5 points]

Dans cet exercice, on considère l'insertion aux feuilles dans un arbre 2-3-4, avec éclatement des 4-nœuds à la descente et on s'intéresse à sa complexité, comptée en nombre d'éclatements. L'objectif est de calculer le coût amorti d'une insertion dans un arbre 2-3-4, en utilisant la méthode du potentiel.

Question 1. Pour un arbre 2-3-4 A ayant n_4 4-nœuds et n_3 3-nœuds, on prendra comme potentiel

$$\Phi(A) = 2n_4 + n_3.$$

Montrer que Φ est une fonction de potentiel valide relativement à l'état initial "arbre vide".

Question 2. On considère la construction d'un arbre par adjonctions successives. Calculer le coût amorti d'une insertion.

4 Hachage [7 points]

Cet exercice étudie une stratégie de hachage qui, étant donné un ensemble statique de n clés, effectue une recherche avec une complexité au pire en $O(1)$ comparaisons entre clés, en utilisant une mémoire totale en $O(n)$ (la mémoire est comptée en nombre de cases pouvant contenir une clé).

On suppose que clés sont des entiers, et soit p un nombre premier assez grand pour que toute clé possible soit dans l'intervalle $[0..p-1]$. On considère des tables de hachage de taille $m < p$, et des fonctions de hachage $h_{a,b}(x) = ((ax+b) \mod p) \mod m$. Soit $\mathcal{H}$ l'ensemble des fonctions de hachage définies par

$$\mathcal{H} = \{h_{a,b}, \quad \text{avec} \quad a \in \{1,..,p-1\} \quad \text{et} \quad b \in \{0,..,p-1\}\}$$

Question 1. Quel est le cardinal de l'ensemble $\mathcal{H}$?
Montrer que $\mathcal{H}$ est un ensemble universel de fonctions de hachage.

Question 2.
Montrer que si l'on hache un ensemble statique de n clés dans une table de hachage de taille $m = n^2$, à l'aide d'une fonction de hachage h choisie aléatoirement dans un ensemble universel de fonctions de hachage, alors la probabilité qu'il n'y ait aucune collision est supérieure à $1/2$.

Question 3. On suppose que l'on peut réserver une table de hachage dont la taille est le carré du nombre d'éléments à stocker. Écrire un algorithme permettant de hacher les éléments sans aucune collision, qui a une complexité en $O(1)$ dans le pire des cas.

Question 4. Lorsque l'on ne peut pas réserver une table de hachage dont la taille est le carré du nombre d'éléments à stocker, on procède en 2 niveaux.

- Au premier niveau, en utilisant une fonction aléatoire h de $\mathcal{H}$, on répartit les clés en m sous-ensembles S_j, formés de n_j de clés ayant même valeur de hachage j, pour $j = 0..m-1$.
  (Le nombre total de clés est $n = \sum_{j=0}^{m-1} n_j$).
- Au second niveau, on crée pour chaque S_j, une taille de hachage de taille n_j^2, via une fonction de hachage $h^{(j)}$, choisie aléatoirement dans $\mathcal{H}$.

  1. Décrire l'algorithme précédent en pseudo-code.
  2. Montrer que la taille de la mémoire requise par cet algorithme est en $O(n)$.
     (On pourra montrer – ou admettre – que la valeur moyenne de $\sum_{j=0}^{m-1} n_j^2$ est inférieure à $2n$).