Algorithmique et programmation
Travaux dirigés, 31 octobre 2002
Louis Granboulan

1 Structures de données

1. Files et piles

   Rappels : une file est munie de deux opérations, `enfiler` et `défiler` et d'un test `vide`. Fonctionnement FIFO (first in, first out). Une pile munie de deux opérations, `push` et `pop` et d'un test `vide`. Fonctionnement LIFO (last in, first out).

   (a) Trouvez une méthode pour simuler une file avec des piles, telle que la complexité amortie d'une opération sur la file soit une constante.

2. Arbres binaires

   Rappels : tout nœud d'un arbre binaire a éventuellement un fils gauche et un fils droit. Si on complète cet arbre en rajoutant des fils à tous les nœuds qui en ont 0 ou 1, on obtient un arbre binaire complet et les nœuds rajoutés sont appelés feuilles de l'arbre. On note n la taille d'un arbre binaire, c'est-à-dire le nombre de ses nœuds. L'arbre binaire complet correspondant a donc $n+1$ feuilles et n nœuds (internes).
   On se propose de calculer le nombre C_n d'arbres binaires de taille n, appelé *nombre de Catalan*.

   (a) Commencez par établir une correspondance entre les arbres binaires complets de taille n dont un nœud ou une feuille est distingué et les arbres binaires complets de taille $n+1$ dont une feuille est distinguée. En déduire que $(n+2)C_{n+1} = 2(2n+1)C_n$.

   (b) En déduire que $C_n = \frac{1}{n+1}\binom{2n}{n}$.

   (c) Donner un algorithme de génération aléatoire et équiprobable d'un arbre binaire de taille n.

2 Comparaisons et tri

1. Minimum

   (a) Montrer, à l'aide d'un algorithme, que le minimum d'un ensemble totalement ordonné de n éléments peut être déterminé en $n-1$ comparaisons.

   (b) Est-ce la meilleure borne possible ?

   (c) Évaluer le nombre moyen d'opérations effectuées par votre algorithme

2. Minimum et maximum

   (a) Montrer, à l'aide d'un algorithme, que le minimum et le maximum d'un ensemble totalement ordonné de n éléments peuvent être déterminés en $\lceil 3n/2 \rceil - 2$ comparaisons.

   (b) Est-ce la meilleure borne possible ?

3. Tri d'un petit nombre d'éléments

   Soit $S(n)$ le nombre minimal de comparaisons pour trier n éléments.

   (a) Montrer que $S(n) \geqslant \lceil \log_2 n! \rceil$.

   (b) Déterminer les valeurs exactes de $S(1)$, $S(2)$, $S(3)$, $S(4)$ et $S(5)$.

4. Sélection de la médiane ou de l'élément d'un rang donné

   (a) Trouver un algorithme qui donne la médiane de n éléments en temps linéaire. On partitionnera par exemple en groupes de m éléments dont on déterninera la médiane.
   Montrer que cet algorithme permet aussi de donner l'élément de rang i en temps linéaire.

   (b) Si on appelle $M(n)$ le nombre minimal de comparaisons pour obtenir la médiane de n éléments, encadrer et essayer de déterminer sa valeur pour les petites valeurs de n.

3 Tri Shell

1. Rappels sur le tri par insertion directe

   Soit un tableau contenant n valeurs distinctes. Le tri par insertion directe consiste à insérer un par un les éléments de la liste dans la fin du tableau, contenant les éléments déjà triés.

   (a) Pour une permutation σ, on appelle inversion un couple $i < j$ tel que $\sigma(i) > \sigma(j)$. Évaluer le nombre moyen d'inversions d'une permutation de n éléments.

   (b) Étudier la complexité de l'algorithme en nombre de comparaisons et d'affectations.

   (c) Réfléchir aux différentes façons d'améliorer cet algorithme

2. Tri Shell

   L'idée est d'essayer de gagner sur le nombre d'affectations en insérant dans des sous-listes (et donc en faisant faire de plus grands déplacements). L'algorithme consiste en un tri par insertion dans des sous-listes contenant de plus en plus d'éléments, de telle sorte que les éléments soient de plus en plus proches de leur position finale (D. Shell, 1959).
   On dit qu'un tableau est h-trié si les h sous-tableaux regroupant les éléments d'indice différant d'un multiple de h sont triés. C'est-à-dire que les suites $(x_i, x_{i+h}, x_{i+2h}, ...)$ pour $i = 1, ..., h$ sont triées.
   Pour le tri Shell, on choisit une suite entière décroissante $h_t, h_{t-1}, ..., h_1$ avec $h_1 = 1$. On commence par h_t-trier le tableau, puis on continuer avec h_{t-1}, etc. Le h_s-tri se fait par exemple par insertion directe, pour chaque sous-tableau.

   (a) Prouver la correction de cet algorithme de tri.

   (b) Prouver que le h-tri d'un ensemble k-trié donne un ensemble k-trié. En déduire que le cas $h_s = 2^{s-1}$ n'est pas optimal.

   (c) Évaluer le nombre moyen d'inversions d'une permutation 2-triée.

      i. Calculer le nombre de permutations 2-triées.

      ii. Montrer que la somme des nombres d'inversions des permutations 2-triées de n éléments (avec $k = \lfloor n/2 \rfloor$) est $A_n = \sum_{i=0..k,j=0..k} |i-j| \binom{i+j}{j} \binom{n-i-j-1}{k-j}$.

      iii. Prouver $A_{2k+1} = 2A_{2k}$. On peut montrer que $A_{2k} = k4^{k-1}$.

      En déduire la complexité de l'algorithme dans le cas où on fait un 2-tri suivi d'un 1-tri.

   (d) Regarder le cas où on fait un h-tri suivi d'un 1-tri.

   (e) Regarder le cas où $h_s = 2^s - 1$.

Knuth, *The Art of Computer Programming, vol. 3 :* ***Sorting and Searching***, pp. 87–92.