Algorithmique
Travaux dirigés, 29 octobre 2004
Louis Granboulan

1 Tri

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. 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
3. 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 ?
4. 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 $S(5)$.
5. 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.

2 Manipulation d'ensembles

Nous voulons réaliser efficacement des opérations de manipulation d'ensembles, telles celles énumérées ci-dessous :

1. **ELEMENT** : donne un élément de l'ensemble. Cette opération permet de savoir si l'ensemble est vide.
2. **APPARTENANCE** : savoir si un élément est membre de l'ensemble.
3. **INSERTION** : ajout d'un élément sachant qu'il n'est pas membre.
4. **SUPPRESSION** : enlèvement d'un élément sachant qu'il est membre.
5. **INTERSECTION** : intersection de deux ensembles.
6. **UNION** : réunion de deux ensembles disjoints.
7. **LOCALISATION** : trouver, parmi une collection d'ensembles disjoints, lequel contient un élément donné.
   Les opérations ci-dessous supposent qu'il existe une relation d'ordre total sur les éléments.
8. **MINIMUM** : extrait le plus petit élément de l'ensemble.
9. **PARTAGE** : un pivot sert à séparer l'ensemble en deux parties : les éléments plus petits et les éléments plus grands que le pivot.
10. **CONCAT** : réunion de deux ensembles tels que tous les éléments de l'un soient inférieurs à tous les éléments de l'autre.

Nous appellerons univers l'ensemble de tous les éléments possibles. Nous devons distinguer le cas où l'univers est fini, de taille raisonnable ou s'il est (presque) infini. On s'intéresse à la complexité d'une opération, ou bien d'une séquence d'opérations, dans le cas le pire ou en moyenne, en fonction de la taille de l'ensemble ou bien de la taille de l'univers.

1. Exemples
   (a) On appelle **dictionnaire** une structure de données permettant les opérations APPARTENANCE, INSERTION et SUPPRESSION.
   Existe-t-il une structure de données pour faire les opérations ELEMENT, APPARTENANCE, INSERTION et SUPPRESSION en temps constant ?
   (b) Existe-t-il une structure de données pour faire une ou plusieurs des opérations INTERSECTION, UNION et LOCALISATION en temps constant ?
   (c) Une **file de priorité** doit permettre INSERTION, SUPPRESSION et MINIMUM.
   Peut-on faire plusieurs parmi INSERTION, SUPPRESSION, MINIMUM et PARTAGE en temps constant ?
2. Remarques générales
   (a) Montrer quelques exemples d'opérations définies ci-dessus qu'on peut réaliser en utilisant une ou plusieurs autres. Quelle est la complexité de ceci ?
   (b) Proposer une liste d'opérations qu'il suffit de savoir faire pour faire toute les autres.
3. Structures de données élémentaires
   Les deux représentations les plus simples sont les suivantes :
   – si l'univers est petit, tous les éléments sont numérotés et on représente un ensemble par sa fonction caractéristique, sous la forme d'un vecteur de bits ;
   – dans tous les cas, on peut représenter un ensemble fini par une liste chaînée de ses éléments.
   On va étudier ces deux structures :
   (a) Liste chaînée. Quelles sont les limitations de cette représentation ? Quelles sont les opérations qu'on peut réaliser efficacement avec cette représentation ? Leur complexité ? Qu'y gagne-t-on à utiliser une liste doublement chaînée ?
   (b) Vecteur de bits. Quelles sont les limitations de cette représentation ? Quelles sont les opérations qu'on peut réaliser efficacement avec cette représentation ? Leur complexité ?

3 Hachage

Le hachage permet d'utiliser une représentation de type "vecteur de bits" avec un univers a priori infini.
On se donne fonction h de l'univers dans l'intervalle $\{1, ..., u\}$. On représente un ensemble E par à l'aide de son image par h. Cela permet de représenter dans une table indicée par $\{1, ..., u\}$ tous les ensembles pour lesquels h n'a pas de collision. On peut généraliser ceci aux cas où h a peu de collisions. On note n le cardinal de l'ensemble et on appelle $r = n/u$ le taux de remplissage.

1. S'il n'y a pas de collisions
   (a) Quelles sont les opérations qu'on peut réaliser efficacement ?
   (b) Quelle est leur complexité en fonction de n ou u.
2. Résolution des collisions
   Il s'agit de représenter l'ensemble $h^{-1}(i) \cap E$, lorsque celui-ci n'est pas un singleton.
   Étudiez les cas suivants (complexité en temps, en mémoire) :
   (a) Résolution par chainage : chaque case de la table de hachage contient la liste chaînée des préimages.
   (b) Hachage à adressage ouvert : chaque case de la table de hachage contient un unique élément. On suppose que $n \leqslant u$ et l'insertion d'un élément c se fait comme suit : si la case $h(c)$ est remplie, on stocke c dans la première case libre après $h(c)$ (i.e. on regarde $h(c) + 1$, $h(c) + 2$, ...
3. Améliorations du hachage à adressage ouvert
   Il s'agit d'éviter les phénomènes de regroupement : si $h(c)$ est distribuée uniformément et si plusieurs case consécutives sont remplies, la case libre qui les suit aura une grande probabilité d'être la prochaine case à remplir.
   Les solutions classiques modifient la gestion des collisions :
   (a) Double hachage : on a une seconde fonction h' à image dans $\{1, ..., r\}$ et on regarde les cases $h(c)$, $h(c) + h'(c)$, $h(c) + 2h'(c)$, ...
   (b) Hachage quadratique : on regarde les cases $h(c)$, $h(c) + 1$, $h(c) + 4$, $h(c) + 9$, ...