Algorithmique et programmation
Travaux dirigés, 24 novembre 2000
Louis Granboulan

1 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) <u>Liste chaînée.</u> 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) <u>Vecteur de bits.</u> Quelles sont les limitations de cette représentation ? Quelles sont les opérations qu'on peut réaliser efficacement avec cette représentation ? Leur complexité ?

2 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 $h(c)$, $h(c) + 1$, $h(c) + 4$, $h(c) + 9$, ...

3 Arbres

Les structures chainées les plus efficaces pour manipuler des ensembles ne sont pas des listes mais des arbres. De nombreuses variantes existent.

1. Tas

   Dans un univers ordonné, on représente un ensemble par un arbre binaire dont les nœuds internes et les feuilles contiennent les valeurs des éléments et tel que la valeur des fils soit supérieure à la valeur du père. On veut aussi que l'arbre soit quasi-complet (les feuilles sont de hauteur h ou $h-1$ et celles de hauteur $h-1$ sont à droite).

   (a) Montrer qu'un tas permet de gérer les opérations d'une file de priorité en $\mathcal{O}(\log n)$ instructions.

   (b) Que dire des autres opérations ?

2. Arbre binaire de recherche

   Dans un univers ordonné, on représente un ensemble par un arbre binaire dont les nœuds sont étiquetés de telle sorte qu'un parcours infixe donne une suite croissante. On dit que l'arbre est équilibré (on parle d'arbre AVL, d'après Adel'son, Vel'skii et Landis) si pour tout nœud la différence entre son sous-arbre gauche et son sous-arbre droit n'excède pas 1.

   (a) Quelle est la complexité des opérations APPARTENANCE, INSERTION, SUPPRESSION et MINIMUM avec une représentation des ensembles par des arbres de recherche non nécessairement équilibrés.

   (b) Encadrer le nombre de nœuds d'un arbre AVL de hauteur h.

   (c) Montrer que pour une représentation d'un ensemble de taille n par un arbre AVL, les quatre opérations ci-dessus peuvent être effectuées en $\mathcal{O}(\log n)$ instructions.

   (d) Que dire des autres opérations ?

   (e) Comment construire un arbre binaire de recherche pour un ensemble donné, tel que le coût moyen de l'opération APPARTENANCE soit minimal, connaissant une distribution de probabilité sur l'univers ?

3. Arbre 2-3

   On représente un ensemble par un arbre dont les nœuds internes ont deux ou trois fils et dont toutes les feuilles sont à la même hauteur. Les feuilles contiennent les valeurs des éléments.
   Si l'univers est ordonné, on propose deux façons d'étiqueter les nœuds internes : 1. Les feuilles sont en ordre quelconque et chaque nœud est étiqueté par la valeur minimum de son sous-arbre. 2. Les feuilles sont en ordre croissant et chaque nœud est étiqueté par la valeur minimum de chacun de ses sous-arbres.

   (a) Encadrer le nombre de nœuds d'un arbre 2-3 de hauteur h.

   (b) Quelles sont les opérations qu'on peut effectuer en $\mathcal{O}(\log n)$ instructions ?

4. Forêt

   On s'intéresse à un univers de taille u et aux opérations d'UNION et LOCALISATION. On représente chaque ensemble par un arbre dont les nœuds sont étiquetés par les éléments de l'ensemble. On aimerait réaliser ces opérations en temps presque constant.

   (a) L'opération d'UNION se réalise naturellement en temps constant, en ajoutant l'un des deux arbres comme fils supplémentaire de la racine de l'autre. Montrer qu'en l'absence de rééquilibrage de l'arbre, une opération de LOCALISATION peut prendre un temps $\mathcal{O}(u)$.

   (b) Montrer que si pour une UNION, on choisit de rajouter le plus petit des deux ensembles comme fils supplémentaire de la racine du plus grand (en nombre de nœuds), une opération de LOCALISATION prend un temps $\mathcal{O}(\log u)$.

5. Compression d'une forêt

   On suppose qu'à chaque LOCALISATION, on modifie l'ensemble trouvé de telle sorte que tous les ancêtres de l'élément cherché deviennent des fils directs de la racine. On définit $F(0) = 1$ et $F(i+1) = 2^{F(i)}$, puis $G(u) = \min\{k \text{ tq } F(k) \geqslant u\}$.
   Pour $\alpha \in \mathbb{R}$, on s'intéresse à l'exécution d'une séquence de αu opérations d'UNION et LOCALISATION à partir d'une partition de l'univers en singletons.
   On définit le *rang* d'un élément v comme sa hauteur dans la forêt résultant de la séquence des opérations d'UNION seules. On partitionne les éléments en groupes, selon l'image par G de leur rang.

   (a) Montrer que la fonction G est quasi constante.

   (b) Montrer qu'il y a au plus $u/F(g)$ éléments de groupe g.

   (c) On sait que pour chaque opération de LOCALISATION, le coût est le nombre d'ancêtres de l'élément cherché. On va répartir le coût comme suit : si v est la racine, ou bien si v et son père sont dans des groupes distincts, le coût attribué directement aux opérations de LOCALISATION est incrémenté de 1. Sinon, c'est le coût attribué à l'élément v qui est incrémenté de 1.
   Montrer que le coût attribué directement à chaque opération de LOCALISATION est au maximum $G(u)$ et que le coût attribué à l'ensemble des éléments d'un même groupe est au plus u.

   (d) En déduire que l'ensemble des opérations de LOCALISATION se fait en temps $\mathcal{O}((1+\alpha)uG(u))$.

4 Application

Soit un graphe non orienté connexe (V, E) dont les arêtes ont un poids réel. On cherche une arborescence recouvrante de poids minimal. Voici un algorithme qui va construire l'ensemble T des arêtes de l'arborescence recouvrante. Le principe de l'algorithme est de partir d'une forêt recouvrante de poids minimal et de la rendre connexe.
On manipule l'ensemble VS dont les éléments sont l'ensemble des sommets pour chaque arborescence de la forêt. C'est donc une partition de V. On initialise T à l'ensemble vide et VS à la partition en singletons.
À chaque étape, on choisit l'arête de poids minimal. On la suprime de E. si ses deux extrémités sont dans des éléments de VS distincts, on rajoute cette arête à T et on réunit les deux éléments de VS correspondants. On s'arrête quand VS est un singleton.

1. Montrer que cet algorithme est correct
2. Évaluer sa complexité.

5 Partage d'informations

Si on suppose qu'on sait faire certaines opérations sur les éléments, la représentation d'un ensemble peut être encore plus efficace.

1. Évaluer ainsi la représentation d'un «dictionnaire» comme on l'a vu dans le TP4 : on optimise la représentation d'un ensemble de chaînes de caractères en partageant des sous-chaînes communes.
2. Trouver l'équivalent pour les ensembles d'arbres (cf. thèse de Laurent Mauborgne `http ://www.di.ens.fr/~mauborgn/publi/these.html`).