Leçon 6 : Implémentations et Applications des ensembles et des dictionnaires

Prérequis : type abstrait, arbre

Niveau : MPII

I - Type abstrait et motivation

Definition 1 : (Dictionnaire). Soit X un ensemble d'éléments appelés valeurs, et K un ensemble d'éléments appelés clefs. Un dictionnaire D est une structure de données abstraite ayant les opérations :

- insertion (D, k, x) : insère le couple (k, x) dans D, en écrasant un éventuel couple (k, y) préexistant.
- recherche (D, k) : renvoie la valeur x tq $(k, x) \in D$ si elle existe.
- suppression (D, k) : supprime un éventuel couple $(k, x) \in D$

Exemple 2 : Un annuaire téléphonique : les clefs sont les noms des personnes, les valeurs leur numéro de téléphone.

Remarque 3 : Les dictionnaires sont aussi appelés tableaux associatifs.

Definition 4 (Ensemble) : Un ensemble est un dictionnaire dans lequel il n'y a pas de clefs. La fonction recherche renvoie un booléen qui indique la présence ou non de l'élément.

Remarque 5 : On déduira donc l'implémentation des ensembles de celle des dictionnaires.

Implémentation 6 : On pourrait stocker tous les éléments dans une liste, sans ordre particulier.

Exercice 7 : Implémenter les différentes fonctions. Quelles sont leurs complexités ?

II - Implémentations

II.1 - Arbres binaires de recherche (ABR)

Définition 8 : Un arbre binaire de recherche (ABR) est un arbre binaire dont les éléments sont munis d'un ordre total et où, pour chaque sous-arbre $N(g, x, d)$, x est supérieur à tous les elts de g et inférieur à tous les elts de d.

Implémentation 9 : On peut implémenter les dictionnaires à l'aide d'un ABR à condition que l'ensemble des clefs soit muni d'un ordre total.

Exemple 10 : deux ABR implémentant l'ensemble $\{10, 3, 8, 5\}$

Insertion dans un ABR :
On insère un élément x en descendant depuis la racine, en prenant le fils gauche ou le fils droit selon si x est plus grand ou plus petit que la racine. On crée un nouveau nœud étiqueté par x lorsqu'on ne peut plus avancer.

Recherche dans un ABR : analogue à l'insertion

Suppression d'un elt : Si le nœud est une feuille ou n'a qu'un enfant, l'opération est simple.
Sinon, il faut retirer le maximum du sous-arbre gauche pour le remplacer.

TP11 : Implémentation du type ABR et des fonctions recherche, insertion et suppression en OCaml.

Complexité : Soit un ABR à n nœuds de hauteur h. Les opérations sont en $O(h)$, donc en $O(n)$ dans le pire des cas.

II.2 - Arbres Rouge Noir (ARN)

Def 12 : Un ARN est un ABR où chaque nœud porte une couleur rouge ou noir, et qui vérifie :
1) la racine est noire
2) les potentiels fils d'un nœud rouge sont noirs
3) le nombre de nœuds noirs le long de n'importe quel chemin de la racine à une feuille est le même, que l'on appelle hauteur noire.

Propriété 13 : Les ARN sont des arbres équilibrés

Propriété 14 : Dans un ARN, on peut effectuer les opérations d'insertion, de recherche et de suppression en $O(h)$, ie $O(\log n)$

Développement 1 : Preuve de la propriété 13 et présentation de l'insertion dans un ARN.

II.3 - Table de hachage

Idée 14 : Le but est de stocker nos données dans un tableau. Cela se fait en deux étapes :
- Associer à notre donnée un nombre appelé hachage par une fonction appelée fonction de hachage.
- Avoir un tableau avec une case par hachage possible.

II.3.1 - Fonctions de hachage

Def 15 : Une fonction de hachage est une fonction $h : K \to [\![0, m-1]\!]$ (avec $m << |K|$)

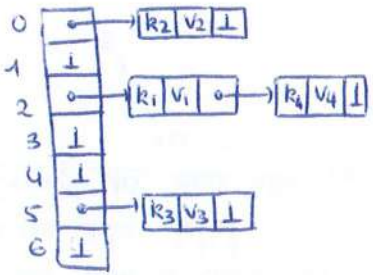
Propriété 16 : Une fonction de hachage a la propriété du hachage parfait si $\forall x, y \in K,\ x \neq y$,
$P_r(h(x) = h(y)) = 1/m$

Exemple 17 : Construction d'une fonction de hachage. On choisit un flottant A. On interprète les bits de données de la structure comme un entier x. On prend alors
$h(d) = [A * x] \bmod m$

II.3.2 - Table de hachage

Il y a trois étapes dans le stockage dans un tableau :
1 - Hacher la valeur et la mettre dans la case
2 - Si la case était déjà occupée (collision), stocker la donnée dans une structure annexe (exemple : chaque case du tableau contient une liste chaînée)
3 - Si le tableau est trop plein, augmenter sa taille m (donc recopier les données avec le nouveau hachage)

Schéma 18 : Résolution des collisions par chaînage



Propriété 19 : Dans une table de hachage où les collisions sont résolues par chaînage, les opérations d'insertions sont en moyenne en $O(1+\alpha)$, où $\alpha = \frac{n}{m}$, sous l'hypothèse d'un hachage uniforme simple.
Si le nombre d'alvéole de la table est supérieur à n, l'insertion est en moyenne en $O(1)$.

Récapitulatif 20 :

Structure	Insertion	Suppression	Recherche
liste	$O(n)$	$O(n)$	$O(n)$
liste triée	$O(n)$	$O(n)$	$O(\log n)$
ABR	$O(n)$	$O(n)$	$O(n)$
ARN	$O(\log n)$	$O(\log n)$	$O(\log n)$
table de hachage	$O(n)$ $O(1)$ moy.	$O(n)$ $O(1)$ moy	$O(n)$ $O(1)$ moy

TP21 : Choisir deux implémentations des dictionnaires en C et comparer leurs efficacités.

III - Applications

III.1 - Dictionnaire d'adjacence

Soit $G = (S, A)$ un graphe, $S = [\![1, n]\!]$.
On peut représenter G par un tableau de dictionnaire D où $D[u]$ est un dictionnaire stockant les voisins de u.

Intérêt : On a les avantages de la matrice d'adjacence et des listes d'adjacence (stockage en $|A|$ et obtention de tous les voisins linéairement).

Exemple 22 : Dans un graphe des personnes se connaissant, on sait si A connaît B rapidement (recherche(D[A], B)) sans stocker la matrice d'adjacence du graphe.

III.2 - Parcours de graphe

La forme générale d'un parcours est la suivante :

Algo 23 : Parcours (G, s) :
```
visités = {} // ensemble vide
a_visiter = {s}
Tant que (a_visiter non vide):
    extraire u de a_visiter.
    insertion (visités, u)
    pour v ∈ G[u]:
        Si v ∉ visités :
            ajouter v à a_visiter
```
où à_visiter peut être une liste, une file...
Quand les nœuds de G sont $\{1, .., n\}$, visités peut être un tableau de booléens. Sinon, visités peut être un ensemble.

III.3 - Autres applications

Les dictionnaires servent dès qu'on veut associer des informations à la valeur d'une structure.

Exemple 24 : En algorithmique du texte, pour associer des valeurs à des chaînes de caractères (Huffmann, Boyer-Moore)

Mémoïsation : On associe la valeur d'une fonction à son argument. On stocke ces associations dans un dictionnaire pour ne pas rappeler récursivement le même calcul.

Développement 2 : Amélioration du tri par comptage par l'usage de dictionnaire.