Les Fichiers structurés

II Les méthodes d'organisation

Introduction

- Pour un fichier d'enregistrements, les opérations de manipulation du fichier se font au niveau de l'enregistrement : on recherche, insère, supprime ou modifie un enregistrement.

- La nature exacte des opérations possibles va dépendre de la façon dont les enregistrements sont organisés dans le fichier.

- L'organisation du fichier aura aussi un impact déterminant sur les performances des opérations.

L'organisation séquentielle

- Dans l'organisation séquentielle, les enregistrements sont simplement placés les uns après les autres dans le fichier sans structure particulière.

- La *lecture* du fichier se fait donc *séquentiellement*, en commençant au début du fichier. Une opération de lecture accède donc à l'enregistrement suivant par rapport à la position courante dans le fichier. La recherche d'un enregistrement donné (par exemple par sa clé) nécessite un parcours séquentiel du fichier.

- L'*écriture* se fait en ajoutant des enregistrements en fin de fichier.

- Il est possible d'envisager des opérations de *suppression* ou d'*insertion* à la position courante dans le fichier, mais ceci n'est pas compatible avec une implémentation simple des fichiers séquentiels.

- Les enregistrements d'un fichier séquentiel peuvent apparaître triés par ordre de clé, mais ce n'est pas une obligation.

Fichier séquentiel : exemple

```
type student_record =
        record
        cle:  integer;
        nom:  packed array[1..20] of char;
        prenom:  packed array[1..20] of char
        end
```

841939	Leonard	Anne
841518	Mennicken	Marc
851286	Irving	Carl
842516	Bizimana	Javan
840558	Suarez Perez	Yolanda
850056	Petitjean	Pierre

L'implémentation des fichiers séquentiels

- Les fichiers séquentiels s'implémentent comme les fichiers génériques étudiés précédemment.

- La seule adaptation nécessaire est liée au fait que l'entité de base d'accès au fichier est maintenant l'enregistrement.

- Si l'on veut permettre des insertions ou suppressions à une position quelconque dans le fichier, il faut pouvoir ajouter ou supprimer de l'espace (des blocs) à une position quelconque dans le fichier. Cela est par exemple possible avec une implémentation utilisant une liste liée.

L'accès direct

- Dans de nombreuses applications des fichiers d'enregistrements, il est nécessaire de pouvoir accéder rapidement à un enregistrement à partir de sa clé.

- On parle dans ce cas *d'accès direct*, par opposition à l'*accès séquentiel*.

- L'accès direct nécessite une organisation particulière des fichiers.

- Il y a deux organisation principales permettant l'accès direct :

  – l'organisation aléatoire et

  – l'organisation indexée.

Fichiers en accès direct: structure de base

- Pour permettre l'accès direct, il faut que l'espace utilisé pour le fichier soit divisé en unités numérotées et accessibles directement.

- Le choix naturel est de prendre les blocs comme unités accessibles directement, mais il est parfois utile de prévoir des groupes de blocs.

- On appelle les unités dont l'accès direct est prévu des *bacs* (*buckets*).

- Supposons qu'un bac soit un bloc où une liste liée de blocs. L'accès direct aux bacs nécessite que l'on puisse trouver rapidement l'adresse sur le disque du premier bloc de chaque bac à partir de son numéro. Cela peut se faire si soit
  - les premiers blocs des bacs sont dans un espace contigu, ou
  - un index des bacs (éventuellement à plusieurs niveaux) est utilisé.

L'organisation aléatoire

- Considérons un fichier implémenté à l'aide de bacs numérotés accessibles directement.

- Pour permettre l'accès direct aux *enregistrements* du fichier, il faut encore trouver une technique qui permette de choisir (et de retrouver par la suite) le bac dans lequel on place un enregistrement, sous l'hypothèse que l'information identifiant un enregistrement est sa clé.

- L'organisation aléatoire utilise pour ce faire une technique basée sur le calcul du numéro de bac à partir de la clé par une fonction appelée *fonction hash*.

Organisation aléatoire : Exemple introductif

Fonction hash : clé *modulo* 7

enregistrement			bac
841939	Leonard	Anne	0
841518	Mennicken	Marc	6
851286	Irving	Carl	2
850640	Demoulin	Pierre	0
842516	Bizimana	Javan	3
840558	Suarez Perez	Yolanda	5
850056	Petitjean	Pierre	4

On constate que la fonction hash calcule le même numéro de bac pour deux enregistrements. Si les bacs ne peuvent contenir qu'un enregistrement, c'est ce qu'on appelle une *collision*. Il faut donc prévoir un mécanisme de traitement des collisions.

bac	enregistrement		
0	841939	Leonard	Anne
1	850640	Demoulin	Pierre
2	851286	Irving	Carl
3	842516	Bizimana	Javan
4	850056	Petitjean	Pierre
5	840558	Suarez Perez	Yolanda
6	841518	Mennicken	Marc

Ici, la collision a été traitée en déplaçant l'enregistrement en surnombre vers le bac suivant qui était libre.

Les fonctions hash

- La version la plus simple de l'organisation aléatoire suppose que l'on a une idée approximative du nombre d'enregistrements et donc que l'on peut fixer la taille du fichier et donc son nombre de bacs N.

- Une fonction "hash" est nécessaire vu que la clé ne peut pas être directement utilisée comme numéro de bac.

- En effet, la clé n'est pas forcément numérique et le nombre de ses valeurs possibles est, le plus souvent de très loin supérieur au nombre d'enregistrements que peut contenir le fichier.

- Le but d'une fonction hash est donc de calculer un numéro de bac à partir de la clé de façon à bien répartir les enregistrements dans les différents bacs.

Les fonctions hash : une tâche impossible

- Clés : entiers de 32 bits.
- Le nombre de clés possibles est donc $2^{32} \approx 4 \times 10^9$.
- Capacité d'un bac : 5 enregistrements.
- Nombre d'enregistrements : 10^4.
- Nombre de valeurs possibles de la fonction hash : $10^4/5 = 2 \times 10^3$

- Nombre moyen de clés auxquelles la fonction hash attribue la même valeur :

$$\frac{4 \times 10^9}{2 \times 10^3} = 2 \times 10^6$$

Quelle que soit la fonction hash choisie, il est donc toujours possible de trouver des ensembles de clés pour lesquelles le résultat sera désastreux.

Les fonction hash : Une justification probabiliste

- Supposons que l'on a une fonction hash qui affecte le même nombre de clés à chaque bac et que les clés soient choisies *au hasard* suivant une distribution uniforme.

- Dans ce cas, la probabilité d'avoir un nombre excessif de collisions devient très faible.

- Le problème est que, en général, les clés ne sont pas choisies au hasard.

- Heureusement, pour avoir une situation proche de celle des clés aléatoires, il suffit qu'il y ait indépendance entre le choix des clés et le choix des bacs effectué par la fonction hash.

- Dans la pratique, il existe des types simples de fonction hash qui satisfont cette contrainte de façon assez générale.

Les fonctions hash : la méthode par division

Ce type de fonction hash se calcule par

$$h(K) = K \bmod M$$

où K est vu comme un nombre binaire. Le problème est le choix de M

Exemple de mauvais choix :

- un multiple de 2 (donne une importance considérable au bit le moins significatif),
- une puissance de 2 (ignore les bits de poids fort),
- des nombres proches d'une puissance de 2.

Pour comprendre ce dernier point, travaillons un moment en base 10.

Les modulos 3,9,11,99,. . . sont de mauvais choix s'il y a couramment des clés qui sont des permutations différentes du même ensemble de chiffres décimaux.

En effet, vu que

$$10 = 1 \pmod 9$$

$$10 = -1 \pmod{11}$$

on a

$$x \bmod 9 = (\sum_{i=0}^{i=k} c_i) \bmod 9$$

et

$$x \bmod 11 = (\sum_{i=0}^{i=k} (-1)^i c_i) \bmod 11$$

Exemple :

$$34567 \bmod 18 = 7$$
$$76534 \bmod 18 = 16$$
$$65347 \bmod 18 = 7$$

le résidu modulo 18 de toute permutation de 34567 sera toujours 7 ou 16

La situation est similaire dans le cas d'informations codées sur k bits, si M a un facteur qui divise $2^k \pm 1$

Exemple :

- Clés : chaînes de caractères (8 bits) ;
- $M = 255$;
- La valeur de la fonction hash sera la même pour toutes les chaînes de caractères qui sont des permutations l'une de l'autre.

Règle de bonne pratique : éviter tout nombre ayant des facteurs

$$2^{kn} \pm a$$

où a et n sont de petits entiers.

Méthode par division : règles pour le choix de M

- Choisir M premier ;
- vérifier que M ne divise pas un nombre de la forme $2^{kn} \pm a$.

La méthode par multiplication

- Clés : entiers de b bits
- Nombre de bacs : $N = 2^m$, $m \leq b$
- $AK \leftarrow K \times A$, K est la clé et A une constante d'au plus b bits.
- $\Rightarrow AK$ est un entier au plus de $2b$ bits
- On en conserve les b bits inférieurs
- Ensuite on en conserve seulement les m bits supérieurs

Exemple

- $b = 4$, $m = 3$ et $A = 1101$
- $K = 111$
- $AK = 1011011$
- partie inférieure de AK : 1011
- $h(K) = 101$

- Le choix de A est bon si
  - A n'est pas un multiple d'une puissance de 2 et
  - la valeur de A est proche de 2^b.

- Méthode apparentée : milieu du carré (*mid-square method*)