Lorsque l'on cherche une information dans une grande base de données, on recourt à un index. Comment concevoir celui-ci pour trouver le plus rapidement possible ce que l'on cherche ?

EVGENII LANDIS co-inventeur en 1962 de l'arbre de recherche équilibré. © COURTESY OF LENA LANDIS

Le classement : un arbre à la hauteur

Hervé Lehning,
professeur de mathématiques spéciales au lycée Janson-de-Sailly.
herve.lehning@prepas.org

Vous entrez dans une bibliothèque pour chercher un livre conseillé par un ami. Il vous a donné le titre et le nom de l'auteur. Vous vous renseignez. On vous envoie à deux fichiers : l'un est classé par thème, l'autre par nom d'auteur. Vous choisissez le second et, très vite, vous trouvez votre auteur. À côté, vous lisez la référence du livre que vous cherchez, un numéro de classement dans les rayonnages. Sans le savoir peut-être, vous avez consulté une base de données. Au numéro dit dans les rayonnages, vous trouvez votre livre.

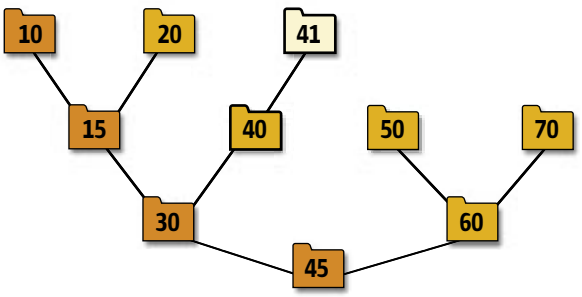
Recherche séquentielle

Les bases de données sont nos meilleurs outils pour trouver rapidement une information ou un objet parmi un grand nombre d'éléments. De façon générale, on peut voir une base de données comme un tableau dont les lignes représentent les données (les livres de notre bibliothèque) et les colonnes les renseignements les concernant (numéro de classement dans la base, nom de l'auteur, thème, etc.). Ce type de formalisme fonctionne aussi bien pour un dictionnaire que pour un fichier de clients d'une entreprise.

Lorsque la base est grande – disons qu'elle comporte un million de fiches, il est difficile d'y trouver ce que l'on cherche si les fiches ne sont pas classées. La seule manière de conduire la recherche est alors de parcourir les fiches une par une, de la première à la dernière. Avec de la chance, la fiche que l'on cherche est la première. Mais elle peut tout aussi bien être la dernière. En moyenne, il faut consulter la moitié des fiches pour trouver la bonne. Une telle recherche est dite séquentielle. Le temps nécessaire pour la mener à bien est proportionnel au nombre d'éléments dans la base. Aussi, même avec l'aide d'un ordinateur, n'est-elle praticable que pour de petites bases de données.

Pour simplifier la consultation, on peut utiliser des index avec un classement suivant un critère tel que le nom de l'auteur. Imaginons que l'on cherche le titre d'un livre dans un million de fiches classées. On prend d'abord la fiche du milieu. La recherche s'arrête si c'est la bonne. Sinon, on poursuit. Si la fiche choisie comporte un nom « antérieur » à celui du titre que nous cherchons, alors notre fiche se trouve entre les fiches 500 001 et 1 000 000 ; sinon elle se situe entre les numéros 1 et 499 999. On poursuit ensuite de la même manière : à chaque étape, la longueur de l'intervalle de recherche est divisée par deux. Dans le pire des cas, la consultation est terminée après 20 tentatives, puisque $2^{20} = 1\,048\,576$.

Fig.1 L'AVL, arbre de recherche équilibré



L'ARBRE est composé de branches et de nœuds portant une étiquette. Les éléments classés sont ici des nombres. Si l'on cherche l'élément 40, on le compare à la racine (45) : il est inférieur. Si cet élément existe, il appartient par convention à la branche (le « fils ») gauche. On le compare ensuite à la racine du fils gauche (30). Il est supérieur donc appartient cette fois à son fils droit. On constate qu'il s'agit de la racine du fils droit. Le processus s'arrête. Si on veut maintenant ajouter l'élément 41, on opère de même.

De façon générale, dans un index ordonné de *N* fiches, la recherche se fait dans un temps proportionnel au logarithme* de base 2 de *N*.

Algorithme récursif

Ce type de classement est bien adapté aux bases de données qui n'évoluent pas ou peu. Mais que faire quand la composition de la bibliothèque évolue et que l'on veut ajouter, modifier ou supprimer une fiche ? Prenons le cas d'un ajout (les autres cas sont semblables). Aucune difficulté pour la base elle-même : on place la nouvelle fiche à la suite des autres. Pour les index, c'est plus difficile. Si vous voulez insérer une nouvelle fiche, il faut la placer au bon endroit. Il est facile de le trouver, il suffit de le chercher comme précédemment. Mais, pour insérer la fiche, il faut décaler une par une celles qui la suivent et corriger leur référence (on change le numéro de ligne dans notre tableau). En moyenne, le temps pour faire une insertion (ou une suppression) est donc proportionnel à la taille de la base. Trop long dès que l'on a beaucoup de données.

Pour simplifier la gestion d'une base de données, on fait appel, pour l'index, à une structure de la forme d'un arbre. Celle-ci est composée de nœuds, chacun portant une étiquette sur laquelle figure le critère de recherche. À chaque nœud, l'arbre se divise en deux. L'étiquette du « fils » gauche est antérieure à celle du nœud et celle du « fils » droit est postérieure. On appelle ce type d'arbre un arbre de recherche « équilibré » ou « arbre AVL » en hommage aux deux mathématiciens russes qui l'ont inventé en 1962, Georgii Adelson-Velskii et Evgenii Landis .

La « hauteur » d'un tel arbre, c'est-à-dire le nombre de branches de bas en haut, est de l'ordre de $\log_2 N$ s'il comporte *N* éléments. En effet, s'il y a 1 élément à la base, 2 au premier niveau, puis 4, 8, etc., alors, pour une hauteur *X*, le nombre total d'éléments est égal à $2^{X+1} - 1$, donc de l'ordre de 2^X.

Comment procède-t-on pour trouver un nom ? On part de la base de l'arbre : si le nom que l'on cherche est « antérieur » au nom étiqueté sur le nœud, alors il se trouve sur la branche de gauche ; sinon il se trouve sur la branche de droite. La recherche se poursuit ainsi en parcourant les arêtes de l'arbre [fig. 1]. Le temps nécessaire en moyenne est proportionnel à la hauteur de l'arbre, donc de l'ordre de $\log_2 N$. Pas mieux qu'avec une liste triée ; en revanche, il est beaucoup plus facile d'ajouter un élément.

Lorsque l'arbre possède déjà un premier nœud (une « racine »), c'est-à-dire s'il n'est pas vide, on ajoute l'élément à la branche de gauche s'il est « antérieur » à ce nœud, à la branche de droite sinon. Un tel algorithme est dit récursif : on peut le réappliquer autant de fois que nécessaire.

Principe de récurrence

Comment montrer que cet algorithme fonctionne toujours ? Grâce au principe de récurrence. L'algorithme s'applique sans problème pour les arbres de hauteur nulle, puisqu'ils sont vides. Imaginons maintenant que l'on a réussi à montrer qu'il fonctionne pour tous les arbres jusqu'à ceux d'une hauteur égale à *H*. On considère un arbre de hauteur $H + 1$. De sa racine partent deux arbres. L'algorithme fonctionne sur ces deux arbres car leur hauteur est inférieure ou égale à *H*. On reprend alors la recette énoncée auparavant : si l'élément à ajouter est antérieur à la racine, il convient de l'ajouter à son fils gauche, sinon à son fils droit. Nous savons le faire dans les deux cas, donc l'algorithme fonctionne pour les arbres de hauteur $H + 1$. Le principe de récurrence permet ainsi d'affirmer que l'algorithme remplit bien son office.

Le temps d'exécution est identique à celui d'une recherche : il est proportionnel à la hauteur de l'arbre, donc de l'ordre de $\log_2 N$. Reste un problème : le nouvel arbre n'est plus forcément équilibré – il peut avoir plus de nœuds d'un côté que de l'autre. Il faut donc veiller à le rééquilibrer. On le fait par « rotations » des parties déséquilibrées, c'est-à-dire en intervertissant les éléments de sorte à avoir un fils de part et d'autre d'un nœud [fig. 2].

Du point de vue théorique, les arbres AVL sont la solution idéale pour structurer les index des bases de données. Mais, dans la pratique, les bases sont stockées sur des disques magnétiques, et donc le temps d'exécution est fortement influencé par l'accès à la mémoire. C'est pourquoi les langages de gestion de bases de données comme SQL *(Structured Query Language)* ou Oracle utilisent plutôt des arbres légèrement différents, appelés arbres B, du nom de leur inventeur, Rudolf Bayer. Ceux-ci, fondés sur les mêmes principes, contiennent plusieurs étiquettes sur chaque nœud, ce qui permet de réduire leur hauteur. La recherche reste active sur ce point car les arbres utilisés doivent toujours être adaptés aux spécificités des nouveaux supports de stockage.

POUR EN SAVOIR PLUS

G. M. Adelson-Velskii et E. M. Landis, *Doklady Akademii Nauk SSSR, 146,* 263, 1962.

R. Bayer et E. McCreight, *Acta informatica, 1,* 173, 1972.

G. Gardarin, *Bases de données,* Eyrolles, 2003.

* Le **logarithme** de base 2 de *N* est le nombre *X* tel que $2^X = N$. On le note $\log_2 N$.

Fig.2 Rééquilibrage par rotation

L'ARBRE se déséquilibre lorsque l'on ajoute les deux éléments 41 et 42. Pour le rééquilibrer, il suffit d'opérer une rotation sur son élément médian. On obtient ainsi un élément de chaque côté. © INFOGRAPHIES BRUNO BOURGEOIS