TD 16
Table indexée

Vincent Thomas(vthomas@loria.fr)

10 juin 2001

1 Description

Une table est une structure permettant de stocker un nombre quelconque d'éléments d'un certain type et les données qui y sont attachées. Ces éléments sont accessibles à partir d'un indice (la **clé d'un élément** ou un index). Par exemple, il peut être intéressant de créer une table d'élèves pour stocker les notes qu'ils auraient obtenues au cours de l'année.

Le problème de base dans une table est d'assurer un accès rapide à une information à partir de l'index fourni. Il existe ainsi plusieurs techniques de lien entre index et informations correspondantes.

2 Énonce

L'objectif de ce TD est d'implémenter une classe Table avec les méthodes suivantes :
• Existence d'un élément à partir d'un indice
• Insertion d'un élément à partir d'un indice et des informations
• Retrait d'un élément à partir de son indice

La classe sera définie par cette interface. Un indice correspondra à un tableau de 10 caractères (nom de login de l'élève). Les informations associées à un indice seront un tableau de 6 entiers (les notes de l'élève dans les différents modules).

3 Organisation de la table

Dans cette partie, plusieurs techniques d'organisation vont être présentées. Essayer d'implémenter ces organisations (qui bien entendues sont transparentes pour l'utilisateur de votre classe).

3.1 Adressage Associatif simple

L'idée de base , **adressage fonctionnel simple** consisterait à créer un tableau de 26^{10} cases. chaque case correspondrait à un nom possible (soit à un mot de 10 lettres) et il serait alors possible d'accéder directement aux informations à partir d'un indice.

Cependant, il est évident que beaucoup de place mémoire serait gâchée (surtout que 26^{10} c'est presque l'infini ! ! !). Pour contourner le problème, on utilise un tableau (mettons de 50 éléments ... on ne s'intéresse pas à pouvoir stocker un nombre quelconque d'élèves) de produits cartésiens cellule.

Une cellule contiendra l'indice et les informations associées à un élément (à savoir un nom et le tableau de 6 notes). Pour obtenir les informations désirées, il suffit d'effectuer une recherche dans ce tableau trié (pour aller plus vite (le tri ralentit l'ajout mais accélère la recherche, c'est un choix à faire !)).

3.2 Adressage dispersé

Afin d'avoir un accès plus rapide, cette organisation consiste à guider la recherche en fonction de la première lettre du nom de l'élève. L'index de la case du tableau est fourni par cette première lettre. Les 26 premières cases du tableau sont donc réservées.

Lorsque 2 noms ont la même première lettre (on parle de **collision**) le premier entré sera à la bonne place ; le second est placé dans la zone de débordement (cases de 27 à 50), il est chaîne au premier.

Afin de gérer ces chaînages, la table sera représentée par un tableau de produits cartésiens cellule2. Celui ci sera défini par 3 champs :
• l'indice (le nom de l'élève)
• les informations (ses notes)
• un entier suivant (pour assurer le chaînage vers un autre élève).

3.3 Fonction de hachage

La fonction que l'on a utilisé au chapitre précédent et qui à un mot donné renvoie la valeur de la première lettre de ce mot est appelé fonction de hachage de la table. L'objectif de cette fonction est de disperser les éléments sur les cases du tableau (d'ou le terme hashing) : il s'agit de bien repartir les indices sur l'intervalle [0..50] domaine de définition du tableau que l'on utilise afin d'éviter le plus possible que deux indices soient envoyés vers le même nombre. En effet dans ce dernier cas, on doit créer un chaînage qui ralentit l'accès à l'élément considéré.

La fonction de hachage utilisé précédemment n'est pas particulièrement performante :

• tout d'abord, elle ne renvoie pas sur l'ensemble du domaine de définition 0..50.
• Ensuite, elle a tendance à renvoyer certains indices plutôt que d'autres. La probabilité pour que le nom d'un élève commence par Z est moins importante que celle pour le nom d'un élève débutant par A. Le risque de collision est donc assez important en 1 alors que celui ci l'est nettement moins en 26. Il est plus adapté d'avoir les mêmes probabilités de collision en 1 et en 26 pour réduire la probabilité globale d'avoir une collision.

A partir de ces considérations proposez une fonction de hachage mieux adaptée.

Implémentez la nouvelle table (Attention, désormais, les cases 27 à 50 ne sont plus réservées pour les collisions... Trouvez une autre solution.)