Algorithmes de recherche et structures de données associées

921

Motivations : On veut pouvoir gérer efficacement une grosse base de données.

idée : associer à chaque élément de l'ensemble à gérer une clé, ces clés auront des propriétés telles qu'on puisse les classer les unes par rapports aux autres et ainsi les chercher plus efficacement.

I) Le dictionnaire, sa première implémentation.

La base de donnée peut être vue exactement comme un dictionnaire qui est un type de donné abstrait donné par :

def :

* dico_vide : $\Rightarrow$ Dictionnaire
* Insérer : Dictionnaire × élément $\Rightarrow$ Dictionnaire
* Supprimer : Dictionnaire × élément $\Rightarrow$ Dictionnaire
* Rechercher : Dictionnaire × élément $\Rightarrow$ Booléen (ou élément).

ex : dictionnaire, annuaire, registre, ...

1) Le tableau

sans hypothèse de plus la recherche est alors séquentielle.

complexité :

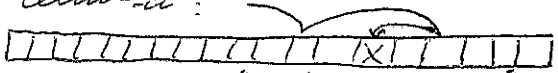
insertion est en $O(1)$ mais la recherche est alors au pire en n et en moyenne en $\frac{n}{2}$ avec n la taille du tableau

2) amélioration

grâce aux clés, on peut trier le tableaux et ainsi faire une recherche dichotomique

on passe ainsi de ce principe :

à celui-ci :



en comparant la clé avec à chaque fois l'élément médiant du tableau.

la complexité au pire et en moyenne passe alors en $O(\log_2 n)$.

th : Un algorithme de recherche qui procède uniquement par comparaison a au mieux une complexité en moyenne et au pire en $\log_2 n$.

Donc la recherche dichotomique est optimale.

II) Arbre binaire de recherche

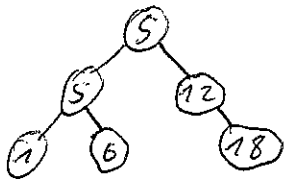
Le tableau n'est pas pratique pour l'insertion ou la suppression on s'intéresse donc aux structures arborescente.

def : un arbre binaire de recherche est un arbre binaire étiquetée tel que pour tout noeud x de l'arbre :

- les noeuds du sous arbre gauche ont des étiquettes inférieures ou égales à l'étiquette de x

- les noeuds du sous arbre droit ont des étiquettes supérieures ou égales à l'étiquette de x

ex :



algorithme de recherche :

```
recherche (ot, k) =
    si est_vide(ot) alors faux sinon
        si k < clé(ot) alors recherche(G(ot), k)
        si k = clé(ot) alors vrai
        sinon recherche(D(ot), k)
;;
```

insertion :

```
insère(ot, k) =
    si est_vide(ot) alors ot ← k comme feuille.
    sinon si k < clé(ot) alors insère(G(ot), k)
        sinon insère(D(ot), k)
;;
```

La suppression est plus délicate mais finalement :

complexité

Toutes les opérations sont en $O(h)$ où h est la hauteur de l'arbre qui est en moyenne un $O(\log n)$.

III) Le hachage

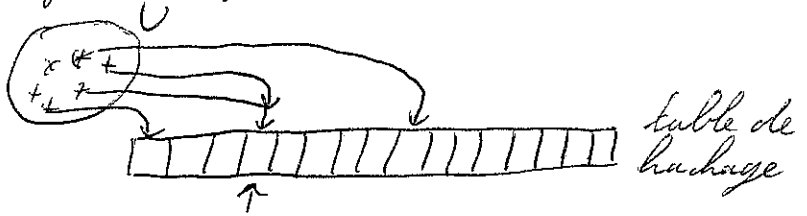
L'idée est de calculer rapidement la place d'un élément grâce à une fonction sur l'ensemble des clés dite fonction de hachage.

$$h : U \longrightarrow \{0, 1, \dots m-1\}$$

avec U l'ensemble des clés.

L'idée est que h est en général une heuristique choisie intelligemment.

Cependant h n'est pas toujours injective, il faut donc gérer des cas où $h(k_1) = h(k_2)$



On dit ici qu'il y a collision

Gérer les collisions revient finalement à faire une recherche parmis les éléments dont les clés ont même image par h. Cependant si h est bien choisie ce nombre d'élément a fortement chuté : en moyenne $\frac{|U|}{m}$.

ex : le vrai dictionnaire avec comme heuristique la première lettre du mot.

en général la gestion des collisions se fait par recherche dans une liste chaîné.

On est quand même motivé par ce qui s'appel le hachage parfait qui garentissait un temps de recherche constant. (DEV 1)

IV) autres types de recherche

Des fois on veut chercher des éléments sans connaître les clés.
On fournit donc des informations partielles et on veut pouvoir retrouver l'élément qui est "le plus proche" de ce que l'on cherche.

ex : recherche d'un fichier dont on a pas la totalité du titre sur le moteur de recherche windows.

Le cas le plus courant de ce type de recherche est lorsque l'on se trouve en possession d'un extrait de texte et que l'on veut trouver de quel texte il est tiré parmis une grande base de donnée.

En général on peut là aussi éliminer une grande quantité de texte par des heuristique comme la fréquence d'apparition des lettres.

ex : la lettre w sépare extrêmement bien les textes allemands des textes français.

Il faut ensuite pouvoir comparer les textes restants à notre extraits : distance d'édition (DEV 2).

l'intérêt de cette remarque est qu'elle souligne un avantage des tables de hachage sur les structures arborescente dans le sens qu'elles permettent de gérer des types de recherche qui ne se font pas forcement par clés.

Mais les arbres ne sont pas forcement exclu car certains, notemment les B-arbres, permettent de fortement diminuer le nombre d'accès à la mémoire lors des recherche ce qui permet aussi une grande efficacité sur de grande bases de données informatique.

ref : Carmen.