Algorithmique Avancée

Michèle Soria

Michele.Soria@lip6.fr

Master Informatique M1-STL

http://www-master.ufr-info-p6.jussieu.fr/2009/algav

Année 2009-2010

Plan

Problème de Recherche

Bases de données

- Ensemble d'éléments
  - Chaque élément a une clé
  - (Ordre total sur les clés)
- Opérations
  - Rechercher un élément
  - Ajouter un élément
  - Supprimer un élément
  - *Construction*
  - *Rechercher tous les éléments dans un intervalle*

Structures concurrentes

- Arbres de Recherche
- Hachage

Structures-Efficacité

Nombre de comparaisons en moyenne//au pire

	ABR	ABR-Equilibré	Hachage
Recherche(n)	$O(\log n)//O(n)$	$O(\log n)//\ O(\log n)$	$O(1)//\ O(n)$
Ajout(n)	$O(\log n)//O(n)$	$O(\log n)//\ O(\log n)$	$O(1)//\ O(n)$
Suppression(n)	$O(\log n)//O(n)$	$O(\log n)//\ O(\log n)$	————

Arbres de Recherche

1. **Arbres binaires de recherche**
   - en moyenne hauteur en $O(\log n)$, mais au pire dégénère en liste $O(n)$
   - algorithmes recherche, ajout et suppression : parcours d'une branche
   - algorithmes simples : modifications minimes
2. **Arbres de recherche équilibrés**
   - hauteur toujours en $O(\log n)$
   - algorithmes recherche, ajout et suppression : sur une branche
   - algorithmes sophistiqués : modifications locales (branche) rotations, éclatement, ... pour maintenir la hauteur en $O(\log n)$

Primitives sur les arbres binaires

`ArbreVide :` → arbre bin
  renvoie *l'arbre vide*
`ArbreBinaire :` *elt × arbre bin × arbre bin → arbre bin*
  ArbreBinaire(x,G,D) ***renvoie*** *l'arbre binaire dont la racine a pour contenu x et dont les sous-arbres gauche et droit sont G et D*
`EstArbreVide :` *arbre bin → booléen*
  renvoie *vrai ssi l'arbre binaire est vide*
`Racine :` *arbre bin → elt*
  renvoie *le contenu de la racine de l'arbre binaire*
`SousArbreGauche :` *arbre bin → arbre bin*
  renvoie *une copie du sous-arbre gauche de l'arbre*
`SousArbreDroit :` *arbre bin → arbre bin*
  renvoie *renvoie une copie du sous-arbre droit de de l'arbre*

Opérations sur les ABR

recherche, ajout, suppression

*Abr-ajout : elt * arbre bin ⟶ arbre bin*
renvoie *l'arbre binaire de recherche résultant de l'ajout de x*

```
Fonction Abr-ajout (x, ABR)
   Si EstArbreVide (ABR)
     Retourne ArbreBinaire(x, ArbreVide, ArbreVide)
   Si x= Racine(ABR) Retourne ABR
   Si x< Racine(ABR)
     Retourne ArbreBinaire(Racine(ABR),
                           abr-ajout (x,SousArbreGauche(ABR)),
                           SousArbreDroit(ABR))
   Sinon Retourne ArbreBinaire(Racine(ABR),
                               SousArbreGauche(ABR),
                               abr-ajout (x, SousArbreDroit(ABR)))
Fin Fonction Abr-ajout
```


Arbres de recherche équilibrés

Idéal ABR parfait : hauteur toujours $\sim \log n$
Mais il faut pouvoir réorganiser l'arbre en $O(\log n)$ après un ajout ou une suppression (ex : ajouter 1 dans l'ABR parfait contenant 2,3,4,5,6).
Assouplir contraintes sur forme des arbres en autorisant déséquilibre

- soit en hauteur → Arbres AVL
- soit en largeur → Arbres B

Arbres AVL

Définition d'un AVL

Un *AVL* (Adelson–Velskii, Landis) est un ABR t.q. en chaque nœud, la hauteur du sous-arbre gauche et celle du sous-arbre droit diffèrent au plus de 1.

Hauteur d'un AVL

Soit h la hauteur d'un AVL avec n nœuds :
$\log_2(n+1) \leq h+1 < 1,44 \log_2 n$

Au pire les arbres de Fibonacci :
$F_0 =< \bullet, , >, F_1 =< \bullet, F_0, >, F_n =< \bullet, F_{n-1}, F_{n-2} >$

Rotations

Rotations pour rééquilibrer, tout en gardant la propriété d'ABR

1. $A =< q, < p, U, V >, W > \Longrightarrow$
   $RD(A) =< p, U, < q, V, W >>$
2. $A =< p, U, < q, V, W >> \Longrightarrow$
   $RG(A) =< q, < p, U, V >, W >$
3. $A =< r, < p, T, < q, U, V >>, W > \Longrightarrow$
   $RDG(A) =< q, < p, T, U >, < r, V, W >>$
4. $A =< r, T, < p, < q, U, V >, W >> \Longrightarrow$
   $RGD(A) =< q, < r, T, U >, < p, V, W >>$

Opérations sur les AVL

- primitive `Hauteur` : *arbre bin → nat*
- fonctions de rotation : `RG, RD, RGD, RGD`
- fonction de rééquilibrage d'un arbre
  `Equilibrage` : *arbre bin → AVL*
  ***renvoie** l'arbre AVL obtenu en rééquilibrant l'arbre initial*
  ***hypoyhèse** : T est un arbre de recherche, les sous-arbres de T sont des arbres AVL et leurs hauteurs diffèrent d'au plus 2*
- recherche, ajout, suppression

Ajout dans un AVL

*AVL-ajout : elt * AVL ⟶ AVL*
***renvoie** l'AVL résultant de l'ajout de x à A*
Fonction AVL-ajout (x, A)
  Si EstArbreVide (A)
    Retourne ArbreBinaire(x, ArbreVide, ArbreVide)
  Si x= Racine(A) **Retourne** A
  Si x< Racine(A) **Retourne**
    Equilibrage (ArbreBinaire(Racine(A),
      AVL-ajout (x,SousArbreGauche(A)),
      SousArbreDroit(A))
  Sinon Retourne
    Equilibrage (ArbreBinaire(Racine(A),
      SousArbreGauche(A),
      AVL-ajout (x, SousArbreDroit(A)))
Fin Fonction *AVL-ajout*

Arbre de recherche général

Arbre de recherche général

Dans un *arbre de recherche général*

- chaque nœud contient un k-uplet $(e_1 < \ldots < e_k)$ d'éléments distincts et ordonnés,
- et chaque nœud a $k+1$ sous-arbres $A_1, \ldots, A_{k+1}$ tels que
  - tous les éléments de A_1 sont $\leq e_1$,
  - tous les éléments de A_i sont $> e_{i-1}$ et $\leq e_i$, pour $i = 2, \ldots, k$
  - tous les éléments de A_{k+1} sont $> e_k$

Arbres 2-3-4

Définition d'un arbre 2-3-4

Un *arbre 2-3-4* est un arbre de recherche

- dont les nœuds contiennent des k-uplets de soit 1, soit 2, soit 3 éléments,
- et dont toutes les feuilles sont situées au même niveau

Hauteur d'un arbre 2-3-4

Soit h la hauteur d'un arbre 2-3-4 avec n éléments :
$h = \Theta(\log n)$

- arbre qui ne contient que des 2-nœuds : $h + 1 = \log_2(n+1)$,
- vs. arbre qui ne contient que des 4-nœuds : $h + 1 = \log_4(n+1)$

Primitives des Arbres 2-3-4

- Notations
  2-nœud : $< (a), T_1, T_2 >$
  3-nœud : $< (a, b), T_1, T_2, T3 >$
  4-nœud : $< (a, b, c), T_1, T_2, T3, T4 >$
- Primitives
  `EstVide` : A2-3-4 $\rightarrow$ booleen
  `Degre` : A2-3-4 $\rightarrow$ entier
  `Contenu` : A2-3-4 $\rightarrow$ LISTE/de longueur 1 à 3/[entier]
  `EstDans` : entier*LISTE[entier] $\rightarrow$ booleen
  `Elem-i` : A2-3-4 $\rightarrow$ entier
  renvoie *le i-ème élément du nœud (et sinon $+\infty$)*
  `Ssab-i` : A2-3-4 $\rightarrow$ A2-3-4
  renvoie *le i-ème sous-arbre du nœud (et sinon Ø)*

Algorithme de recherche

*234Recherche : entier * A2-3-4 ⟶ booleen*
renvoie *vrai ssi x est dans A*

Fonction 234Recherche(x, A)
  Si EstVide(A) **Retourne** FAUX
  Si EstDans(x,Contenu(A)) **Retourne** VRAI
  Si x<Elem-1(A) **Retourne** 234Recherche(x, Ssab-1(A))
  Si x<Elem-2(A) **Retourne** 234Recherche(x, Ssab-2(A))
  Si x<Elem-3(A) **Retourne** 234Recherche(x, Ssab-3(A))
  Retourne 234Recherche(x, Ssab-4(A))
Fin Fonction *234Recherche*

Complexité en nombre de comparaisons : $O(\log n)$

Ajout d'un élément

- Ajout aux feuilles (guidé par la recherche)
- un i-nœud se transforme en $(i+1)$-nœud, par insertion dans la liste
- sauf lorsque la feuille contient déjà 3 éléments ! ! !

Exemple : Construire par adjonctions successives un arbre 2-3-4 contenant les éléments $(4, 35, 10, 13, 3, 30, 15, 12, 7, 40, 20, 11, 6)$

Deux méthodes de rééquilibrage

- Eclatements en remontée (au pire en cascade sur toute une branche)
- Eclatements en descente (éclatement systématique de tout 4-nœud)

Comparaison des méthodes

Les deux méthodes ne donnent pas forcément le même arbre. Elles opèrent toutes les deux en $O(\log n)$ comparaisons (transformations sur une branche)

Avantages de la méthode d'éclatements en descente

- parcours de branche uniquement de haut en bas
- transformation très locale : accès parallèles possibles

Inconvénients de la méthode d'éclatements en descente

- taux d'occupation des nœuds plus faible
- hauteur de l'arbre plus grande

Eclatements en descente

Transformations de rééquilibrage locales : sur le chemin de recherche, on éclate systématiquement les 4-nœuds

1. Le père du nœud à éclater ne peut pas être un 4-nœud
2. Le père du nœud à éclater est un 2-nœud
   $P2 = <(x), A1, A2>$, avec $A1 = <(a, b, c), U1, U2, U3, U4>$
   $\Longrightarrow \quad P2 = <(b, x), <(a), U1, U2>, <(c), U3, U4>, A2>$
3. Le père du nœud à éclater est un 3-nœud
   $P3 = <(x, y), A1, A2, A3>$ et $A2 = <(a, b, c), U1, U2, U3, U4>$
   $\Longrightarrow \quad P3 = <(x, b, y), A1, <(a), U1, U2>, <(c), U3, U4>$
   $, A3>$
4. autres cas analogues

Ne modifie pas la profondeur des feuilles (sauf lorsque la racine de l'arbre éclate, la profondeur est alors augmentée de 1)

Algorithme d'ajout

Fonction ajout(x, A)
    Si Degré(A)= 4 **Retourne** ajoutECR(x, A)
    Retourne ajoutSimple(x, A)
Fin Fonction *ajout*

*ajoutECR : entier * A2-3-4/racine4noeud / $\longrightarrow$ A2-3-4*
Fonction ajoutECR(x, A)
  Retourne ajoutSimple(x, A') *A' résulte de l'éclatement de la racine de A*
Fin Fonction *ajoutECR*

*ajoutSimple : entier * A2-3-4/racineNon4noeud/ $\longrightarrow$ A2-3-4*
Fonction ajoutSimple(x, A)
    Si Degré(A)< 4 **Retourne** ajoutSimple(x, Ui)
    Si Degré(Pere(A))=2 **Retourne** ajoutSimple(x, P2)
    Retourne ajoutSimple(x, P3)
Fin Fonction *ajoutSimple*

Remarques

- Ui est le sous-arbre dans lequel doit se poursuivre l'ajout (comme pour une recherche)
- P2 est le transformé2 du père de A (cf. éclatements)
- P3 est le transformé3 du père de A (cf. éclatements)

- Complexité en nombre de comparaisons : $O(\log n)$
- Implantation : représentation des arbres 2-3-4 par des arbres binaires bicolores (voir TD).

Arbres-B

- Recherche Externe : éléments stockés sur disque
- Mémoire secondaire paginée (allouer et récupérer les pages)
- Temps d'accès MS 100000 fois supérieur à MC.
- D'où organisation pour avoir peu de transferts de pages.

Un *B-arbre d'ordre m* est un arbre de recherche

- dont les nœuds contiennent des k-uplets d'éléments, avec $m \leq k \leq 2m$,
- sauf la racine, qui peut contenir entre 1 et $2m$ éléments,
- et dont toutes les feuilles sont situées au même niveau

Arbre Bm, n nœuds : $\log_{2m+1}(n+1) \leq h+1 \leq 1+\log_{m+1}[(n+1)/2]$

Algorithmes et complexité

- Algorithmes analogues à ceux des arbres 2-3-4
- Seul le nœud racine est en mémoire centrale $\Longrightarrow$ nombre d'accès à la mémoire secondaire $=$ hauteur de l'arbre
- hauteur inférieure à $\log_{m+1}[(n+1)/2] \Longrightarrow$ prendre m grand. $m = 250$ peut contenir 125.10^6 éléments dans un arbre de hauteur 2
- éclatement $\rightarrow$ écrire 2 pages en MS (# éclatements borné par hauteur)
- Analyse amortie : # éclatements dans construction par adjonctions successives d'un B-arbre d'ordre m compris entre $1/m$ et $1/2m$
- Analyse de frange : 1 éclatement pour $1,38m$ adjonctions