Slide 1

Les arbres de recherche

Slide 2

0. Résumé des épisodes précédents

Slide 3

Types de données dynamiques = enregistements + récursivité

Un champ récursif : type de listes

Plusieurs champ récursif : type d'arbres

Algorithmes sur les listes

Algorithmes sur les arbres :

arbres de recherche, parcours et files de priorité

Slide 4

I. Les arbres

Types d'arbres

Slide 5

Type enreg. avec au moins deux champs récursifs (*v.s.* listes)

Type d'arbres binaires : exactement deux champs récursifs

```
class Arbre {
 Arbre gauche;
 Arbre droit;}
```

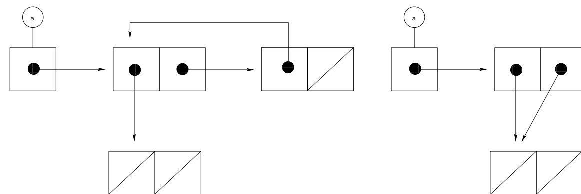
```
class Arbre {
 int val;
 Arbre gauche;
 Arbre droit;}
```

Les arbres

Slide 6

Un arbre : une valeur d'un type d'arbres

Chaque référence utilisée une fois (ni val. rationnelles ni partage)



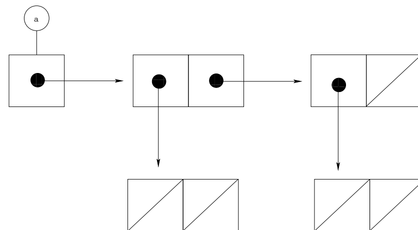
Les arbres

Slide 7

Un arbre : `null` (arbre vide) ou une référence associée dans la mémoire à deux arbres

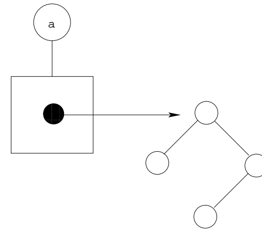
La représentation graphique des arbres

Slide 8



Slide 9

La représentation graphique des arbres



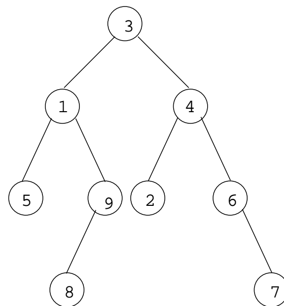
Un rond : une cellule du tas

Qu'est-ce que la valeur de `a` ?

Slide 10

Avec un champ pour le contenu

```
class Arbre {int val; Arbre gauche; Arbre droit;}
```



Un peu de vocabulaire

Slide 11

Un arbre (non vide) = une référence = une cellule

Ensemble des nœuds de l'arbre `a` : ensemble des références accessibles depuis `a`

i.e. contient `a` et clos par `.gauche` et `.droit` (si références)

Un nœud = une référence = une cellule

Un nœud est de la forme
`a.gauche.gauche.droit.gauche...`

deux suites binaires distinctes : deux références distinctes

Un peu de vocabulaire

Slide 12

Nœud `c` : enfant gauche de `b` si `c = b.gauche`

enfant droit, parent

feuille : pas d'enfants

Racine de l'arbre `a` : `a` (pas de parent)

Sous-arbre gauche d'un nœud : enfant gauche (vu comme un arbre)

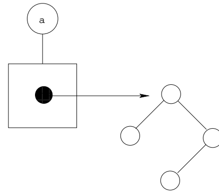
sous-arbre droit

Un peu de vocabulaire

Branche : suite de nœuds où chaque nœud enfant du précèdent

Hauteur : longueur de la plus longue branches - 1

Slide 13




```
h(vide) = - 1
```

Slide 14

II. Les arbres de recherche

Slide 15

Les ensembles comme des listes

Rechercher un élément : linéaire en le nombre d'éléments de l'ensemble (dictionnaire, annuaire, ...)

Recherche dichotomique : complexité logarithmique

Liste ordonnée

Mais pas d'accès en temps constant à l'élément médian (volumen)

Tableau : mais taille choisie au moment de l'allocation, pas d'évolution possible (insertion et suppression linéaires)

Slide 16

Arbre de recherche

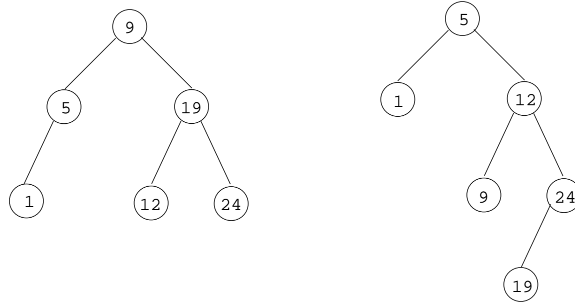
Ordre sur les données

Arbre binaire (avec un champ `val`)

`x.val` > contenu des noeuds du sous-arbre gauche de `x`

`x.val` < contenu des noeuds du sous-arbre droit de `x`

Slide 17



La complexité

Slide 18

Recherche, insertion, suppression :

temps proportionnel à la hauteur de l'arbre (grosso modo logarithmique en le nombre d'éléments de l'ensemble)

La recherche

Slide 19

```
static boolean recherche (int x, Arbre a) {
 if (a == null) return false;
 if (x == a.val) return true;
 if (x < a.val) return recherche(x,a.gauche);
 return recherche(x,a.droit);}
```

L'insertion

Slide 20

L'élément inséré devient une feuille

Il suffit de trouver le nœud auquel l'accrocher

Modifier ou recopier

`insert(n,a);` *v.s.* `a = insert(n,a);`

L'insertion

L'élément inséré devient une feuille

Il suffit de trouver le nœud auquel l'accrocher (sauf dans le cas ou l'arbre est vide)

Slide 21

Modifier ou recopier

`insert(n,a);` *v.s.* `a = insert(n,a);`

Même si on modifie `insert(n,a);` impossible si l'arbre n'est pas enveloppé

La suppression

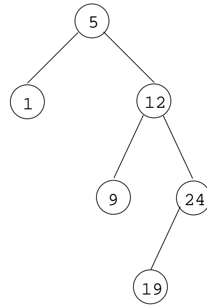
On recherche le nœud à supprimer

Slide 22

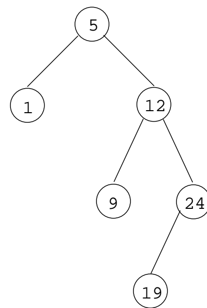
Puis analyse de cas :

- 0 enfant (feuille)
- 1 enfant
- 2 enfants

Slide 23



Slide 24



On supprime le `12` : le `9` doit prendre sa place

Supprimer un nœud qui a deux enfants :

On cherche le max du sous-arbre gauche

Slide 25

Nouvelle valeur du nœud qui contient la valeur à supprimer

On le supprime récursivement dans le sous-arbre gauche

Pas d'enfant droit

Une application : les dictionnaires

Liste d'associations : liste de couples

Idem : arbre de couples

Slide 26

```
class Arbre {
 int cle; int val; Arbre gauche; Arbre droit;}

static int assoc (int x, Arbre a) {
 if (a == null) return -1;
 if (x == a.cle) return a.val;
 if (x < a.cle) return assoc (x,a.gauche);
 return assoc (x,a.droit);}
```

Slide 27

III. Les arbres équilibrés

Grosso modo

Slide 28

Recherche, insertion, suppression : temps proportionnel à la hauteur de l'arbre

Logarithmique en le nombre d'éléments de l'ensemble

La relation entre la taille et la hauteur

Slide 29

Hauteur h, taille maximale : $2^{h+1} - 1$

$max(h+1) = 1 + 2\ max(h)$

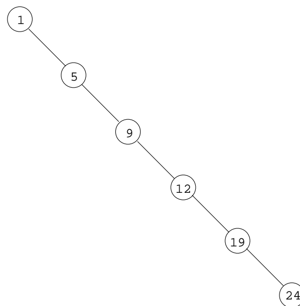
(ou suites binaires longueur $\leq h$: $1 + 2 + 2^2 + ... + 2^h$)

Hauteur minimale d'un arbre de taille n : $\lfloor \log_2(n) \rfloor$

La relation entre la taille et la hauteur

Slide 30

Mais hauteur maximale : $n - 1$



Arbres de recherche

Slide 31

Insertion aléatoire : espérance de la hauteur proportionnelle à `ln(n)`

Complexité en moyenne de la recherche, l'insertion et la suppression : `ln(n)`

Mais complexité dans le pire des cas : `n`

Les cas dégénérés se produisent en pratique : insertion de données ordonnées

Équilibrer

Slide 32

Une contrainte supplémentaire : garder toujours des arbres de hauteur minimale

Mais difficile (impossible ?) de programmer l'insertion et la suppression en temps logarithmique : rééquilibrer trop cher

Ensembles d'arbres équilibrés

Hauteur minimale : contrainte trop forte (rééquilibrage trop cher)

Un ensemble d'arbres équilibrés : un ensemble d'arbre tel qu'il existe une constante k telle que $h \leq k \ln(n)$

Slide 33

Ensemble des arbres de taille minimale : oui

Ensemble de tous les arbres : non

Attention : un singleton est toujours un ensemble équilibré

L'équilibre est une propriété collective

Recherche, insertion et suppression logarithmiques

Choisir un ensemble d'arbres équilibrés

Slide 34

Pour lequel le rééquilibrage soit logarithmique

De nombreuses possibilités : AVL, arbres 2-3, ...

Les arbres d'Adel'son-Velskii et Landis

À chaque nœud la différence de hauteur du sous-arbre gauche et droit est `-1`, `0` ou `1`

Slide 35

$h \leq \ln(n+2) / \ln(\tau) - 1$

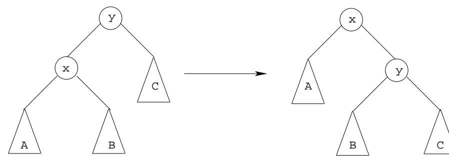
Rotations

Après chaque insertion ou suppression

Tous les nœuds d'une branche : déséquilibre `-2` ou `2` possible

Rotations (conserve la propriété arbre de recherche)

Slide 36



Slide 37

IV. D'autres méthodes pour représenter les ensembles et les fonctions finies

Slide 38

Un cas rare

Une fonction dont le domaine est {`0, ..., n-1`}

On utilise un tableau de taille `n`

Rare, mais pas inexistant : Top 50, ATP, X, ...

Adressage direct

Un autre cas d'adressage direct

Slide 39

On peut se ramener à {`0, ..., n-1`} par une fonction injective sur les clés

Exemple : le domaine est {`a, ..., z`} : méthode de César, Morse, Braille, ...

Les tables de hachage

Slide 40

Se ramener à {`0, ..., n-1`} par une fonction non nécessairement injective : une fonction de hachage

Exemples :

Dans un carnet d'adresse : la première lettre du nom

La somme des codes ASCII des lettres du nom modulo 100

Un exemple

Slide 41

On entre dans son carnet d'adresse : Basile (case 1), Odilon (case 14), Édouard (case 4), Mélanie (case 12), Raymond (case 17), Lucien (case 11), Alix (case 0), Guillaume (case 6), Paulin (case 16), Tatiana (case 19), Yvette (case 24), Nina (case 13), Rémi (case 17)

Une collision

Comment résoudre les collisions ?

Slide 42

Un tableau de listes d'association : chaînage

Si la case 17 est occupée, on met Rémi dans la case 18

(première case libre du tableau après 17) : adressage ouvert

Assymptotiquement

Slide 43

Résolution des collisions par chaînage : linéaire

Résolution des collisions par adressage ouvert : ne marche pas

Mais si on connaît la taille n de la table

Slide 44

Résolution des collisions par chaînage : tableau de taille $\sqrt{n}$, nombre de comparaisons $\sqrt{n}$

Résolution des collisions par adressage ouvert : si taille $2n$, 3 comparaisons en moyenne

Mieux : double hachage

Slide 45

V. Les exceptions

Slide 46

Les dictionnaires

```
static int assoc (int x, Arbre a) {
 if (a == null) return -1;
 if (x == a.cle) return a.val;
 if (x < a.cle) return assoc (x,a.gauche);
 return assoc (x,a.droit);}
```

Fonction partielle

Que retourner si la clé n’est pas dans l’arbre ?

Que retourner ?

Slide 47

- `-1` : une valeur conventionnelle (si possible disjointe de l'ensemble des valeurs)
- un couple booléen , valeur
- un élément d'un type disjonctif singleton $\uplus$ valeurs
- on interdit d'utiliser `assoc` si la clé n'est pas dans l'arbre (fonction de test)

Obscur, lourd, inefficace

Les exceptions

Slide 48

```
static int assoc(int x, Arbre a)
                                throws Exception {
 if (a == null) throw new Exception ();
 if (x == a.cle) return a.val;
 if (x < a.cle) return assoc(x,a.gauche);
 return assoc(x,a.droit);}}
```

Si la clé n'est pas dans la liste :

- la fonction ne retourne pas de valeur et
- elle lève une exception

Récupérer les exceptions

Slide 49

```
try {System.out.println(assoc(6,a));}
catch (Exception e) {
        System.out.println("Pas dans l’arbre");}
```

La propagation des exceptions

Slide 50

```
static int assocplus (int x, Arbre a)
                              throws Exception {
 return assoc(x,a) + 1;}
```

Appel `assoc` non protégé par un `try`

Si `assoc` lève une exception alors `assocplus` aussi

La propagation des exceptions

```
static int assocplus (int x, Arbre a)
                              throws Exception {
 return assoc(x,a) + 1;}
```

Slide 51

Appel `assoc` non protégé par un `try`

Si `assoc` lève une exception alors `assocplus` aussi, idem :

```
static int assocplus (int x, Arbre a)
                              throws Exception {
 try {return assoc(x,a) + 1;}
 catch (Exception e) {throw new Exception ();}}
```

Exceptions et messages d’erreur

```
System.out.println(1/0);
```

Slide 52

```
java.lang.ArithmeticException: / by zero
```

La fonction Σ

Slide 53

Si `p1` lève une exception

alors dans `{p1 p2}` on n'exécute pas `p2`

et on lève cette même exception

Similaire à `return`

Slide 54

Exercice 9.5 et 9.6

Slide 55 La prochaine fois : le parcours d’un arbre et les files de priorité