

Informatique 06

Arbres AVL

Dictionnaires arborescents

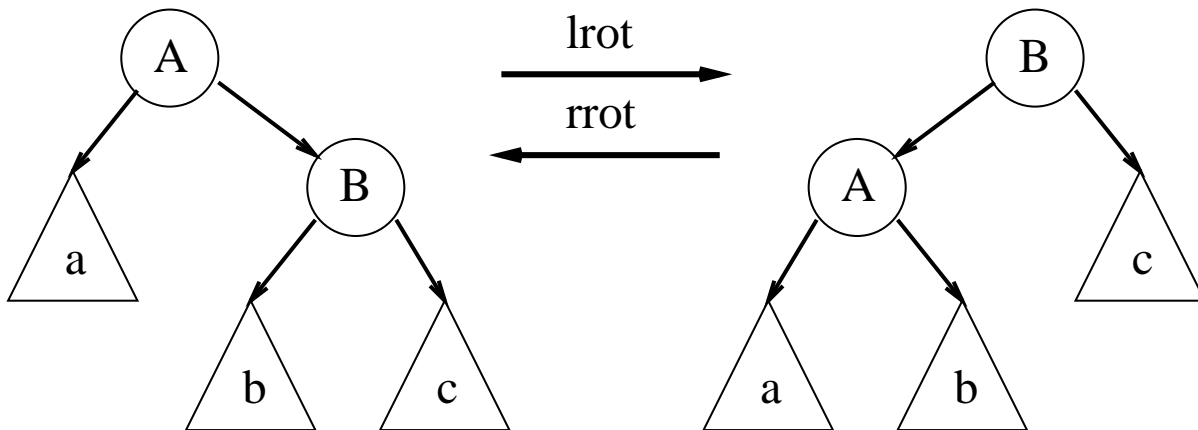
François Bourdoncle

`Francois.Bourdoncle@ensmp.fr`

`http://www.ensmp.fr/~bourdonc/`

Arbres AVL — définition

- Différence de hauteur ≤ 1
- Recherche/insertion/suppression en $O(\log n)$
- Rotations



Arbres AVL — Java

```
class Tree {
    int value;
    int height;
    Tree left;
    Tree right;

    static Tree empty = null;

    Tree(int value) {
        this.value = value;
        this.height = 1;
        this.left = empty;
        this.right = empty;
    }

    static boolean isEmpty(Tree t) {
        return t == empty;
    }

    static int h(Tree t) {
        return isEmpty(t) ? 0 : t.height;
    }

    static int delta(Tree t) {
        return isEmpty(t) ? 0 : h(t.right) - h(t.left);
    }

    static void adjust(Tree t) {
        t.height = 1 + Math.max(h(t.left), h(t.right));
    }
}
```

Rotations dans un arbre AVL

```
static Tree rrot(Tree t) {  
    Tree u = t;  
    t = t.left;  
    u.left = t.right;  
    t.right = u;  
    adjust(u);  
    adjust(t);  
  
    return t;  
}
```

```
static Tree lrot(Tree t) {  
    Tree u = t;  
    t = t.right;  
    u.right = t.left;  
    t.left = u;  
    adjust(u);  
    adjust(t);  
  
    return t;  
}
```

Insertion dans un arbre AVL

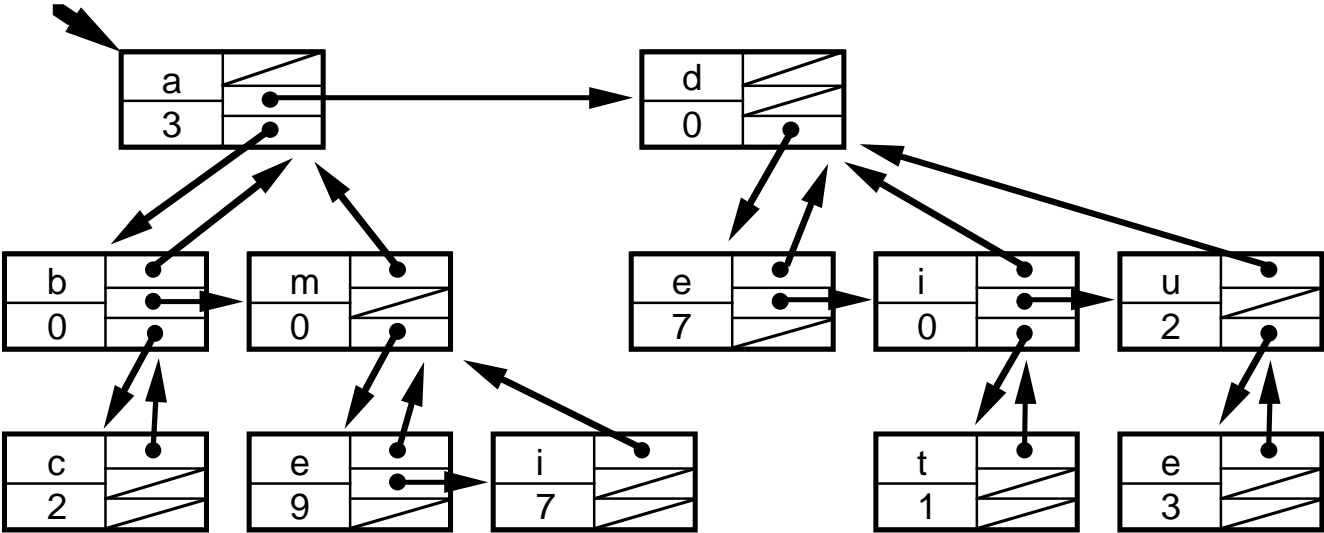
```
static Tree insert(Tree t, int x) {
    if (isEmpty(t)) {
        return new Tree(v);
    } else {
        if (x < t.value) {
            t.left = insert(t.left, x);
        } else {
            t.right = insert(t.right, x);
        }
        adjust(t);
        if (delta(t) < -1) {
            if (delta(t.left) < 0) {
                t = rrot(t);
            } else {
                t.left = lrot(t.left);
                adjust(t);
                t = rrot(t);
            }
        } else if (delta(t) > 1) {
            if (delta(t.right) > 0) {
                t = lrot(t);
            } else {
                t.right = rrot(t.right);
                adjust(t);
                t = lrot(t);
            }
        }
    }
    return t;
}
```

Dictionnaire arborescent

- Nombre d'occurrences d'un mot
- Affichage d'un mot du dictionnaire
- Affichage du dictionnaire entier
- Suppression d'un mot
- Insertion d'un mot

Structure de données — exemple

a: 3
 abc: 2
 ame: 9
 ami: 7
 de: 7
 dit: 1
 du: 2
 due: 3



Structure de données — Java

```
public class Mot {
    char c;
    int nb;
    Mot pere;
    Mot frere;
    Mot fils;

    static Mot empty = null;

    Mot(char c, int nb, Mot pere,
        Mot frere, Mot fils) {
        this.c = c;
        this.nb = nb;
        this.pere = pere;
        this.frere = frere;
        this.fils = fils;
    }

    static boolean isEmpty(Mot m) {
        return (m == empty);
    }

    static int nombre(Mot m, String s) {...}
    static void afficherMot(Mot m) {...}
    static void afficherDico(Mot m) {...}
    static void afficherIteratif(Mot m) {...}
    static Mot enlever(Mot m, String s) {...}
    static Mot ajouter(Mot m, String s) {...}
}
```

Nombre d'occurrences d'un élément

```
static int nombre(Mot m, String s) {
    return nombre(m, 0, s);
}

static int nombre(Mot m, int k, String s) {
    if (isEmpty(m) ||
        k == s.length() ||
        s.charAt(k) < m.c) {
        return 0;
    } else if (s.charAt(k) == m.c) {
        if (k == s.length() - 1) {
            return m.nb;
        } else {
            return nombre(m.fils, k + 1, s);
        }
    } else {
        return nombre(m.frere, k, s);
    }
}
```

Affichage d'un mot

```
static void afficherMot(Mot m) {  
    if (!isEmpty(m)) {  
        if (m.nb > 0) {  
            afficherLettres(m);  
            write(": ");  
            write(m.nb);  
        }  
    }  
}
```

```
static void afficherLettres(Mot m) {  
    if (!isEmpty(m)) {  
        afficherLettres(m.pere);  
        write(m.c);  
    }  
}
```

Affichage d'un dictionnaire par ordre alphabétique

```
static void afficherDico(Mot m) {  
    if (!isEmpty(m)) {  
        afficherMot(m);  
        afficherDico(m.fils);  
        afficherDico(m.frere);  
    }  
}
```

Affichage d'un dictionnaire (version itérative)

```
static void afficherIteratif(Mot m) {  
    boolean bas = true;  
    while (!isEmpty(m)) {  
        if (bas) {  
            afficherMot(m);  
        }  
        if (bas && !isEmpty(m.fils)) {  
            m = m.fils;  
        } else if (!isEmpty(m.frere)) {  
            bas = true;  
            m = m.frere;  
        } else {  
            bas = false;  
            m = m.pere;  
        }  
    }  
}
```

Suppression d'un élément

```
static Mot enlever(Mot m, String s) {
    return enlever(m, 0, s);
}

static Mot enlever(Mot m, int k, String s) {
    if (isEmpty(m) ||
        k == s.length() ||
        s.charAt(k) < m.c) {
        return m;
    } else if (s.charAt(k) == m.c) {
        if (k == s.length() - 1) {
            if (m.nb > 0) {
                --m.nb;
            }
            if (m.nb == 0 && isEmpty(m.fils)) {
                return m.frere;
            } else {
                return m;
            }
        } else {
            m.fils = enlever(m.fils, k + 1, s);
            return m;
        }
    } else {
        m.frere = enlever(m.frere, k, s);
        return m;
    }
}
```

Ajout d'un élément

```
static Mot ajouter(Mot m, String s) {  
    return ajouter(m, empty, 0, s);  
}
```

```
static Mot ajouter(Mot m, Mot p, int k, String s) {  
    if (k == s.length()) {  
        return m;  
    } else if (isEmpty(m) || s.charAt(k) < m.c) {  
        m = new Mot(s.charAt(k), 0, p, m, empty);  
        return ajouter(m, p, k, s);  
    } else if (s.charAt(k) == m.c) {  
        if (k == s.length() - 1) {  
            ++m.nb;  
            return m;  
        } else {  
            m.fils = ajouter(m.fils, m, k+1, s);  
            return m;  
        }  
    } else {  
        m.frere = ajouter(m.frere, p, k, s);  
        return m;  
    }  
}
```