

Cours 6

Arbres équilibrés Recherche sur disque

Jean-Jacques.Levy@inria.fr

<http://jeanjacqueslevy.net>

tel: 01 39 63 56 89

secrétariat de l'enseignement:

Catherine Bensoussan

cb@lix.polytechnique.fr

Aile 00, LIX

tel: 01 69 33 34 67

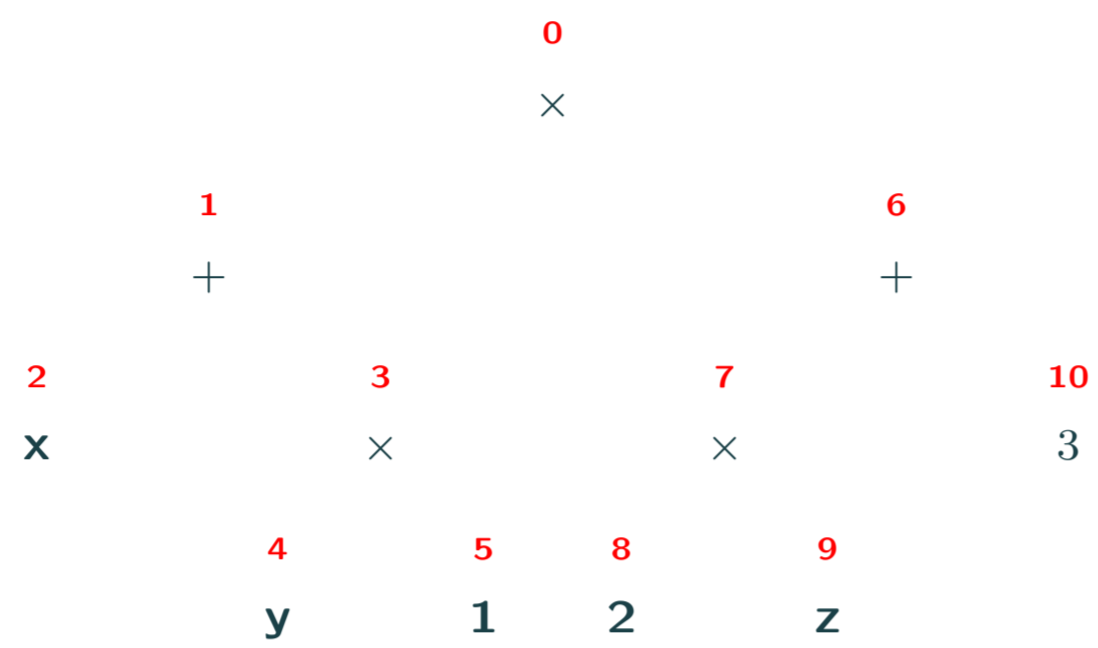
<http://www.enseignement.polytechnique.fr/informatique/>

Plan

1. Parcours d'arbres
2. Tables dynamiques
3. Arbres de recherche
4. Arbres de recherche équilibrés
5. Arbres B
6. Systèmes de fichiers

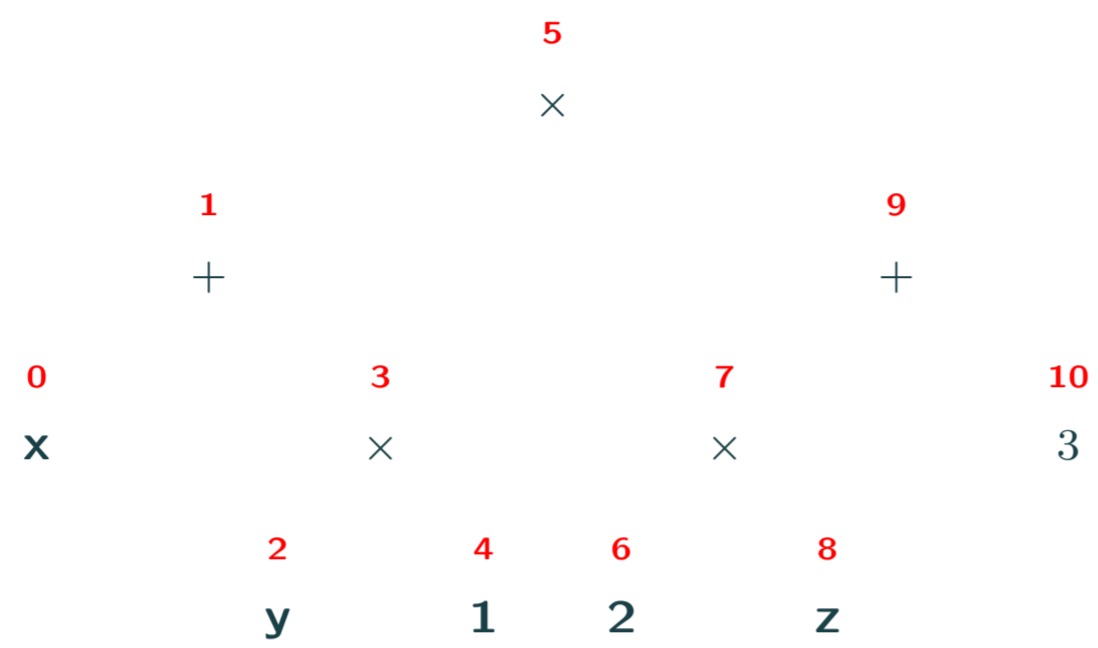
Parcours d'arbre en ordre préfixe – Preorder

Arbre de syntaxe abstraite de $(x + y \times 1) \times (2 * z + 3)$



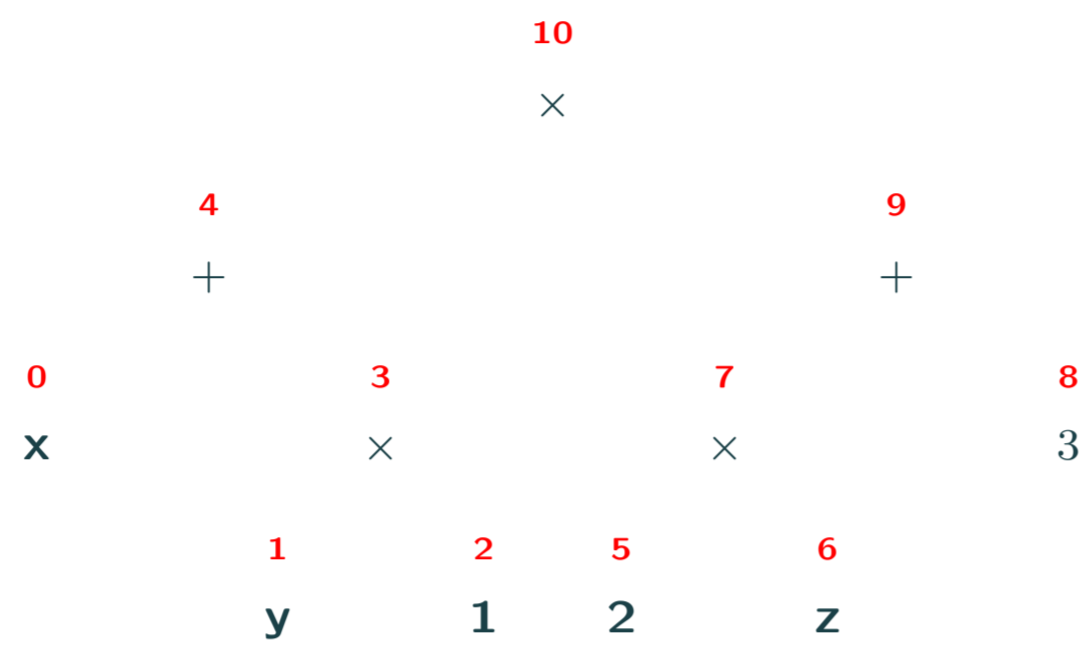
Parcours d'arbre en ordre infix – *Infix*

Arbre de syntaxe abstraite de $(x + y \times 1) \times (2 * z + 3)$



Parcours d'arbre en ordre suffixe – Postorder

Arbre de syntaxe abstraite de $(x + y \times 1) \times (2 * z + 3)$



Arbres dynamiques

```
class Terme {
    final static int ADD = 0, SUB = 1, MUL = 2, DIV = 3, MINUS = 4,
                  VAR = 5, CONST = 6;
    int nature; Terme a1, a2; int valeur; String nom;

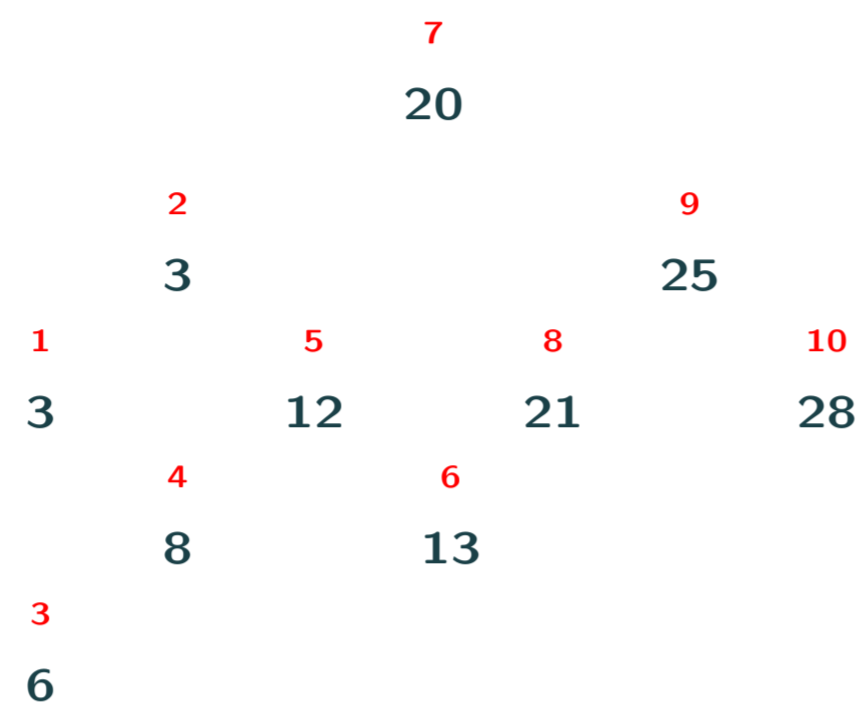
    static String suffixe (Terme t) {
        switch (t.nature) {
            case ADD: return suffixe(t.a1) + suffixe(t.a2) + "+ ";
            case SUB: return suffixe(t.a1) + suffixe(t.a2) + "- ";
            case MUL: return suffixe(t.a1) + suffixe(t.a2) + "* ";
            case DIV: return suffixe(t.a1) + suffixe(t.a2) + "/ ";
            case VAR: return t.nom + " ";
            case CONST: return t.valeur + " ";
            default: throw new Error("Terme illégal");
        }
    }
}
```

Exercice 1 Ecrire une fonction qui calcule la forme polonaise préfixe.

Exercice 2 Ecrire une fonction qui calcule l'ASA à partir de la forme polonaise.

Arbres de recherche

Arbre dont les valeurs sont en ordre croissant dans un parcours préfixe.



Insertion dans un arbre de recherche



```
static Arbre ajouter (int v, Arbre a) {
    if (a == null) return new Arbre (v);
    else if (v <= a.val)
        return new Arbre (a.val, ajouter (v, a.fG), a.fD);
    else
        return new Arbre (a.val, a.fG, ajouter (v, a.fD));
}
```

Insertion/Destruction/Recherche dans un arbre binaire de recherche

L'ajout peut aussi être destructif.

```
static Arbre ajouterD (int v, Arbre a) {
    if (a == null) return new Arbre (v);
    else if (v <= a.val) a.fG = ajouterD (v, a.fG);
    else a.fD = ajouterD (v, a.fD);
    return a;
}

static boolean membre (int v, Arbre a) {
    if (a == null) return false;
    else return v == a.val || membre (v, a.fG) || membre (v, a.fD);
}
```

Exercice 3 Complexité de la recherche?

Exercice 4 Ecrire la suppression d'un élément.

Exercice 5 Que faire avec les éléments dupliqués?

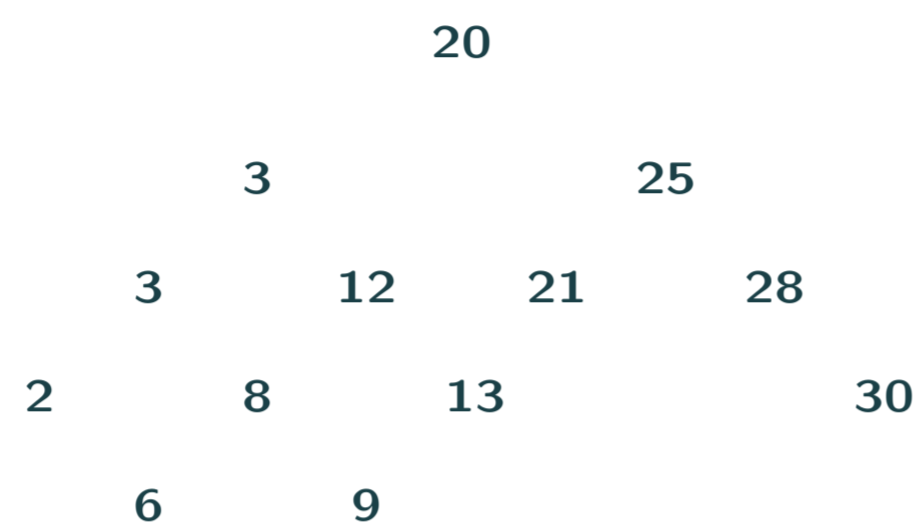
Arbres équilibrés

Un arbre est équilibré si la recherche est toujours garantie en temps $O(\log n)$.

Par exemple, dans les arbres AVL, tout sous-arbre a vérifie

$$|h(a.fG) - h(a.fD)| \leq 1$$

où $h(a)$ est la **hauteur** de a . Quand un élément est inséré ou enlevé, il faut s'assurer de préserver cette relation.

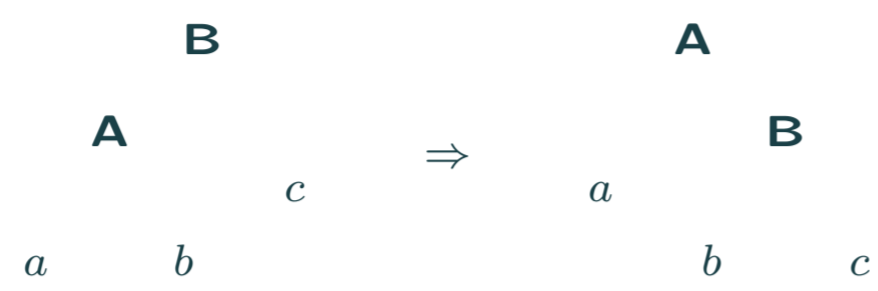


[Adelson-Velskii et Landis, 1962]

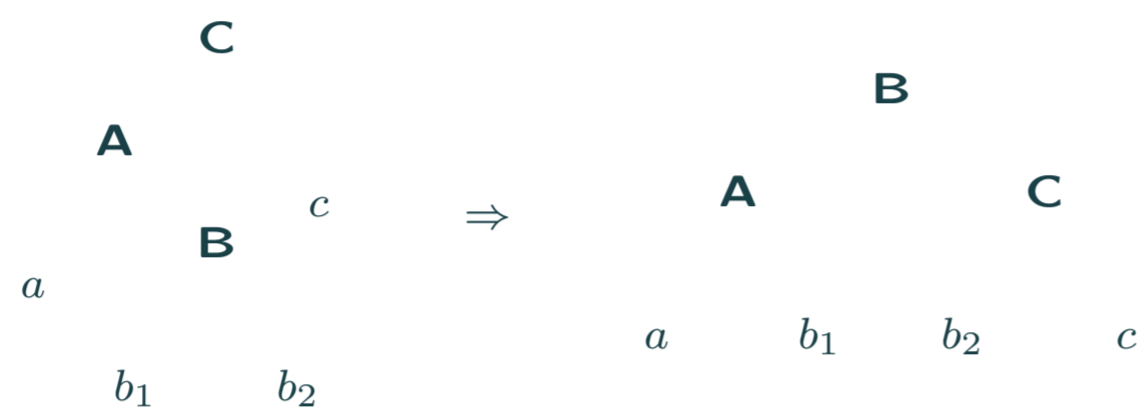
Rotations d'arbres binaires de recherche

Pour rééquilibrer, on peut faire des opérations locales:

(Cf. l'[animation](#) de arsen@geocities.com) avec des **rotations simples**



ou des **rotations doubles**



Programmation des arbres AVL

On a un champ supplémentaire bal donnant l'équilibre de chaque noeud.

```
static Arbre rotD (Arbre a) { // Rotation droite

    Arbre b = a;
    a = a.fG;
    int bA = a.bal; int bB = b.bal;
    b.fG = a.fD; a.fD = b;
    // Recalculer le champ a.bal
    int bBnew = 1 + bB - Math.min(0, bA);
    int bAnew = 1 + bA + Math.max(0, bBnew);
    a.bal = bAnew; b.bal = bBnew;
    return a;
}
```

Exercice 6 Combien de rotations sont nécessaires par insertion?

Exercice 7 Programmer la suppression.

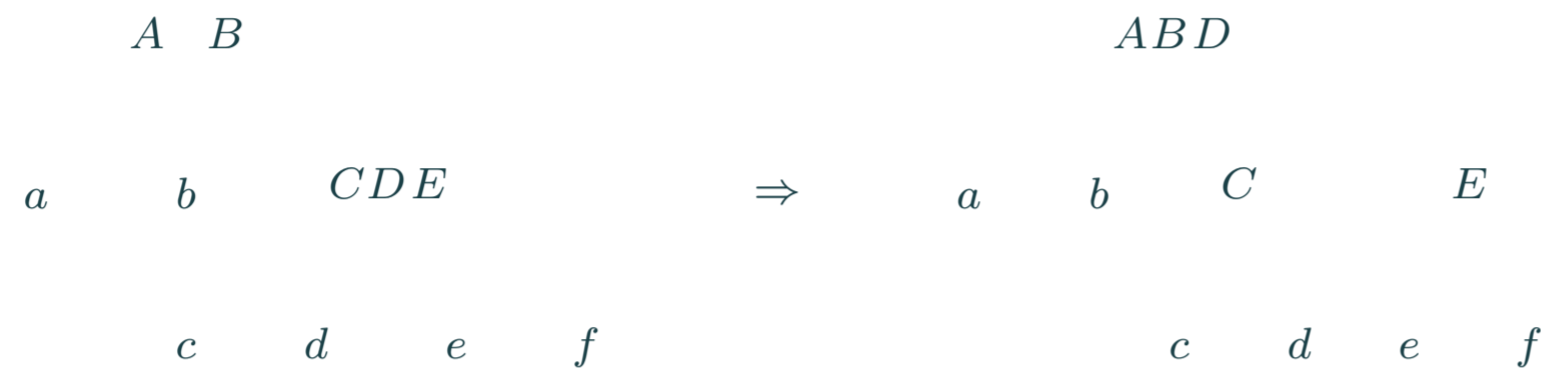
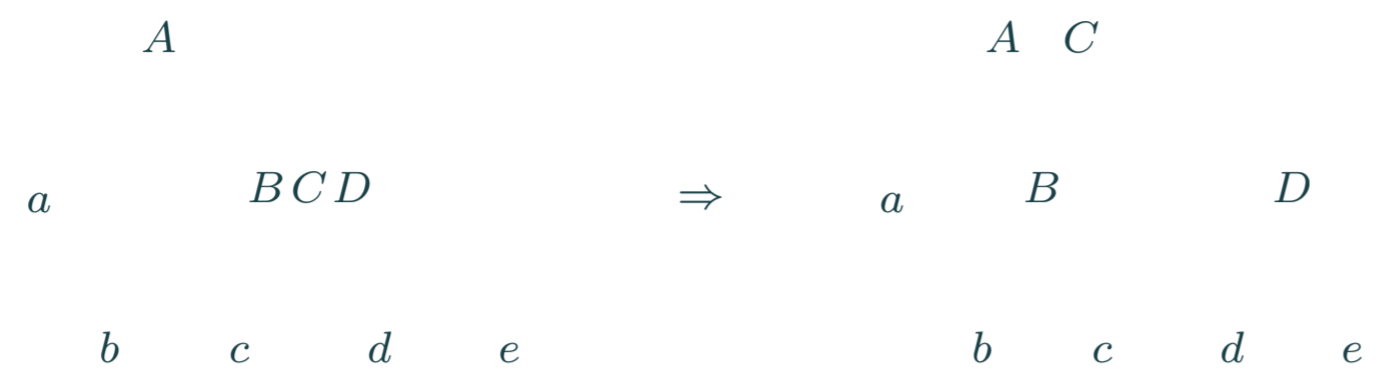
Exercice 8 Combien de rotations sont nécessaires dans la suppression?

Arbres 2-3-4

Les arbres sont un peu plus **flexibles**. Chaque noeud peut avoir 2, 3 ou 4 fils a_1, a_2, a_3, a_4 .



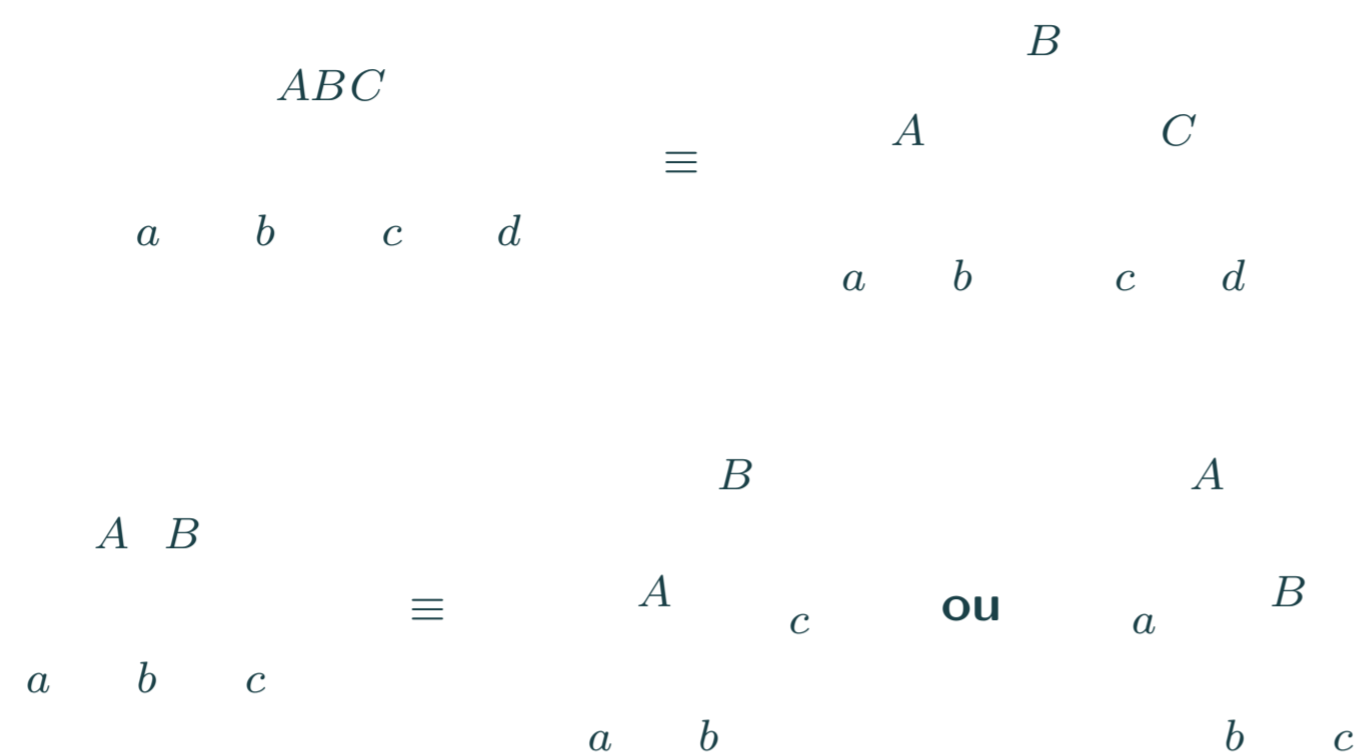
Réorganisation des arbres 2-3-4



Seul l'**éclatement** d'un noeud à la racine augmente la hauteur.
(Les arbres sont toujours équilibrés).

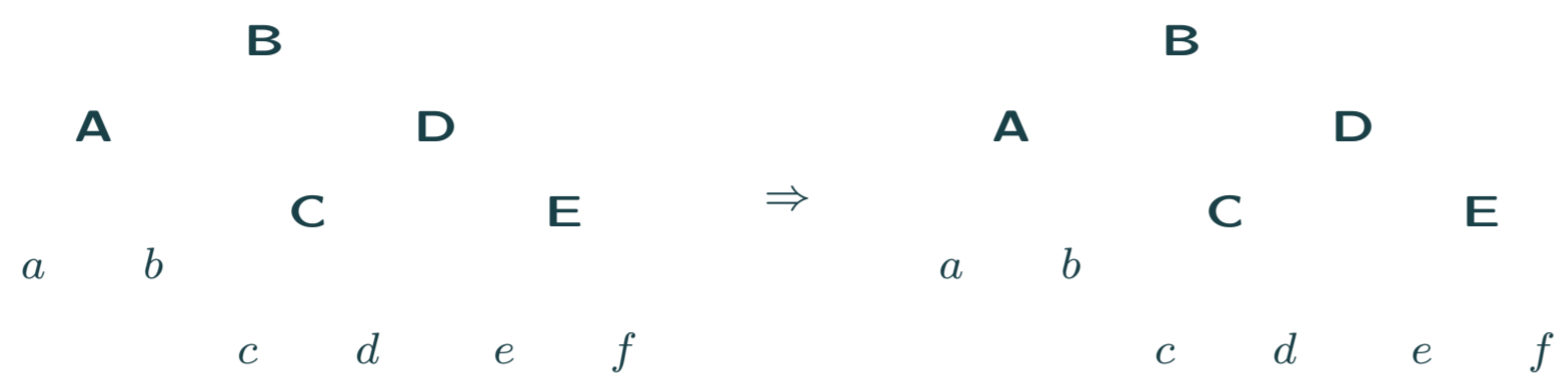
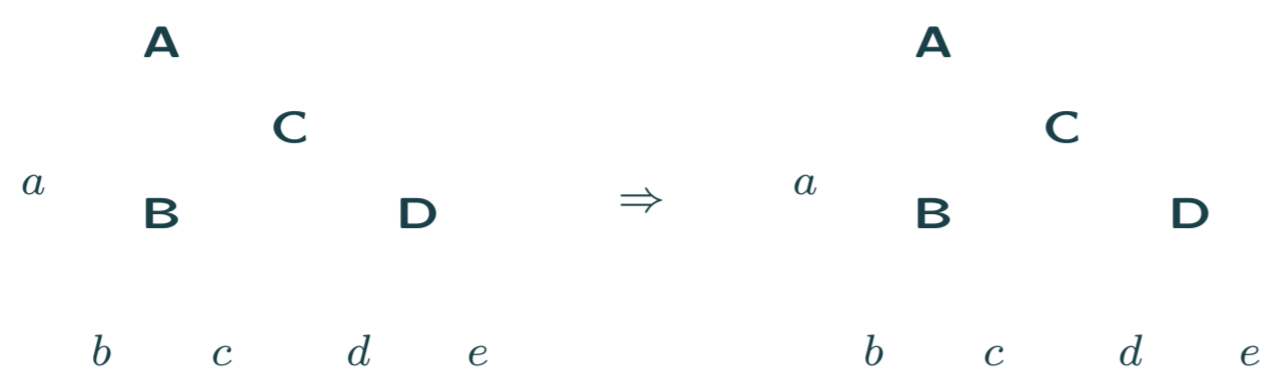
Arbres de recherche binaires bicolores

Pour réduire l'encombrement mémoire, on peut coder les arbres 2-3-4 par des arbres binaires (on se ramène à un problème proche des arbres AVL avec un seul bit pour signaler un déséquilibre.)

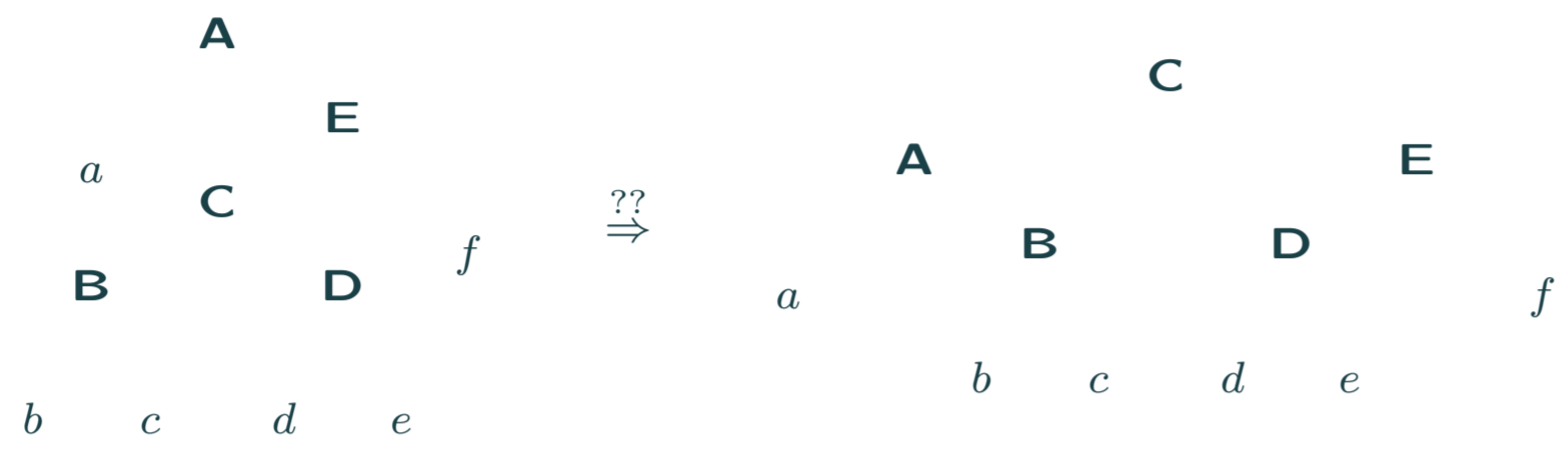
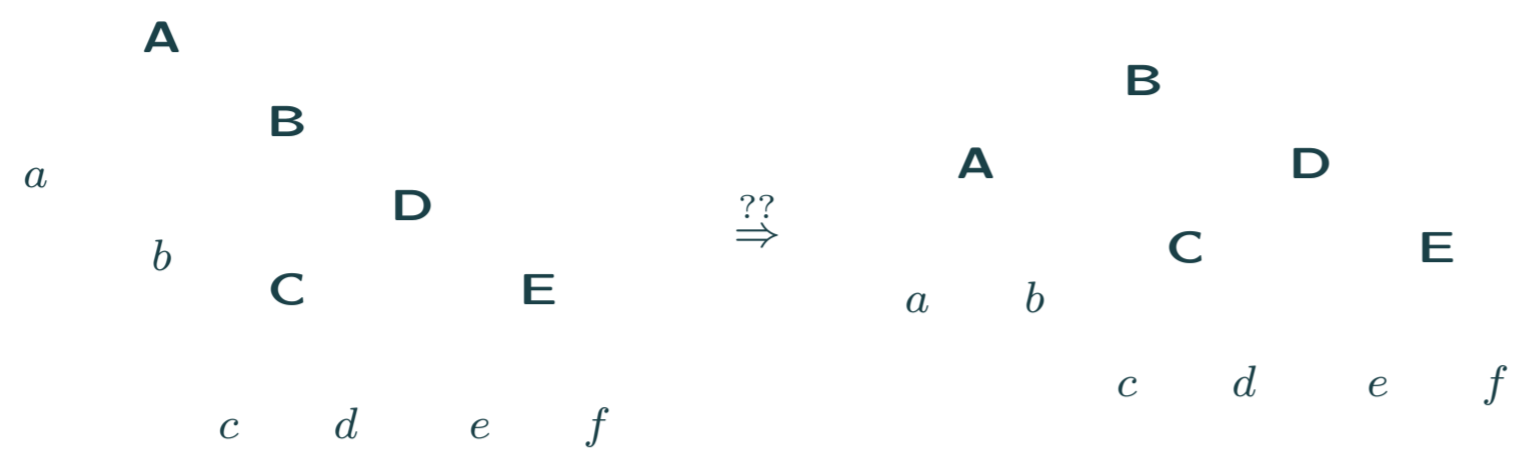


Chaque noeud a un champ supplémentaire (un booléen rouge) donnant la couleur (**rouge** ou noir) du lien avec le père.

Eclatements d'arbres bicolores



Rotations d'arbres bicolores



Recherche sur disque

Une donnée sur disque est plus **longue** à retrouver. Typiquement 10-20 ms pour lire un bloc de 1k octets, alors que $.5\mu s$ pour retrouver un entier en mémoire vive.

Parfois on ne peut pas tout mettre en mémoire. Grosses bases de données. Il faut minimiser le nombre de lectures/écritures disque.

Fichiers en accès séquentiel

On range les informations en ordre croissant \Rightarrow Beaucoup d'accès disque. Autrefois les fichiers sur bandes.

Fichiers en accès séquentiel indexé

En tête de disque un **index** donne la localisation des valeurs stockées sur ce disque. Un index d'index permet de choisir le bon disque. Dans une telle méthode on ne fait qu'un nombre $O(1)$ lectures disque.

Inconvénient: pas très flexible.

C'est +/- la FAT de MS-DOS

Arbres B – B-trees

Arbres équilibrés 2- M . Le cas $M = 4$ correspond aux 2-3-4.

On éclate les noeuds d'arité M comme pour l'insertion dans les 2-3-4, et on le remplace par deux noeuds d'arité $M/2$.

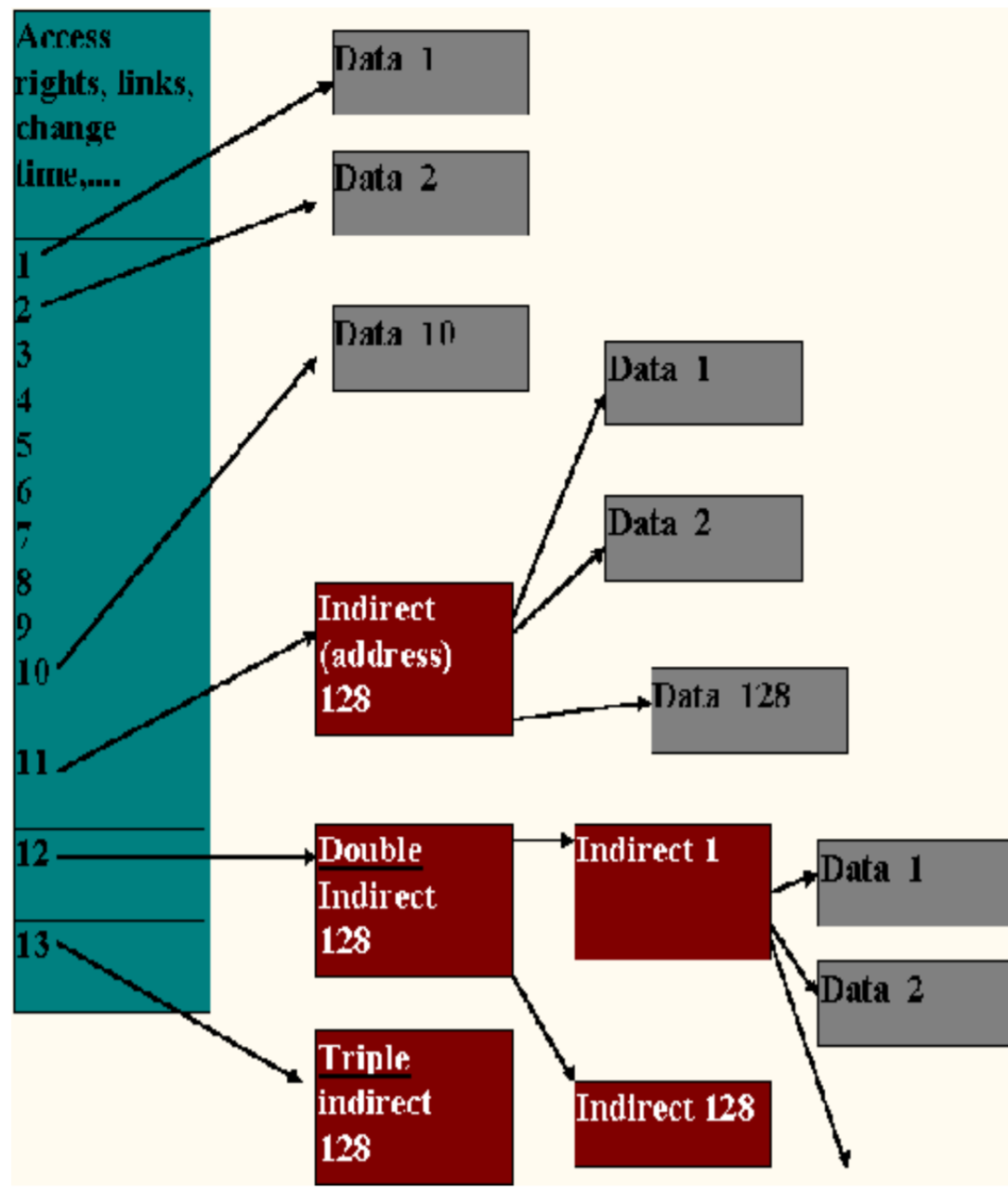
Il y a alors de la place en bas de l'arbre pour ranger l'élément.

On fait donc au plus $\log_{M/2}$ accès disque lors de l'insertion.

Les arbres B sont utilisés dans le système de fichiers de MacOS pour retrouver les blocs d'information correspondant aux fichiers. La clé recherchée est alors la paire (f, b) où f est le numéro de fichier, et b le numéro de bloc à l'intérieur du fichier.

Accès direct

Dans Unix (Linux), un fichier est représenté par un numéro interne i (*inode*) et on cherche des numéros de blocs b (un bloc = 512



octets ou plus).

En TD

- finir le TD sur analyse syntaxique
- construction d'arbres équilibrés avec analyse syntaxique
- gestion d'ensembles dynamiques

- les systèmes de fichiers sont vus en majeure M2, Cours systèmes/réseaux
- les bases de données sont vus en majeure M1, Cours BD
- les analyses d'algorithmes sont vus en majeure M2, Cours Algorithmique et analyse.