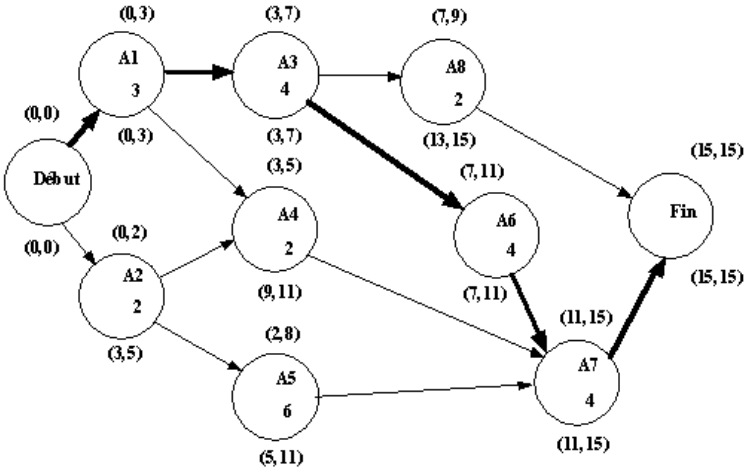
Chapitre 1
Complément Index, arbres B+

I. Notions d'arbres - généralités

A. Graphes

Un **graphe** est un ensemble de points dont certains sont reliés 2 à 2. Les liaisons entre ces points peuvent être orientées ou non. Les points sont appelés **sommets** (*en anglais : vertice*) ou **nœuds**, les liens sont appelés arêtes (*en anglais : edge*) ou arcs (*orientés*). Les graphes peuvent être étiquetés : aux sommets ou arêtes sont associés des valeurs d'un ensemble (*nombres, couleurs, etc.*)

Les graphes permettent la représentation de données complexes comme : les hyperliens du Web, le réseau Internet, les réseaux sociaux, la succession des états d'un système, la planification (*exemple ici d'un graphe PERT*), etc.



B. Arbres binaires

Un **Arbre Binaire** (*anglais : BT, Binary Tree*) est une **structure de données** de type graphe représentée sous forme hiérarchique, arborescente, dont chaque élément est appelé **nœud**. Un nœud permet le stockage de données de manière efficace (*pour la recherche*), extensible (*la limite est la mémoire*) et cohérente (*représentation de hiérarchies diverses, nomenclatures, familles, etc.*).

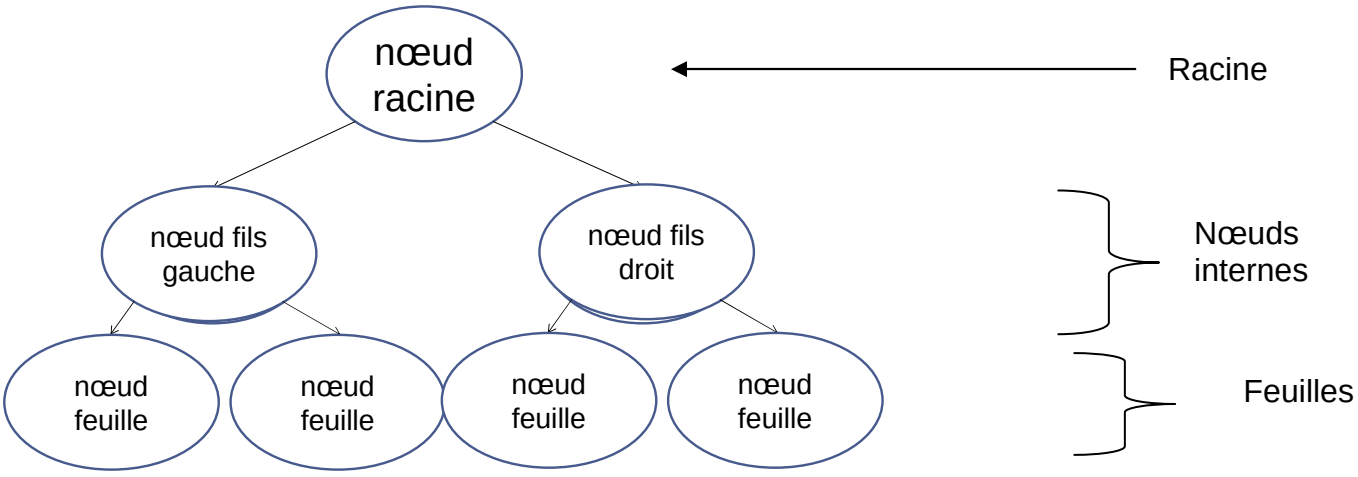
Un nœud peut posséder **au plus 2 nœuds fils** : un nœud fils gauche et un nœud fils droit.

Le nœud initial d'un arbre binaire est appelé **racine**.

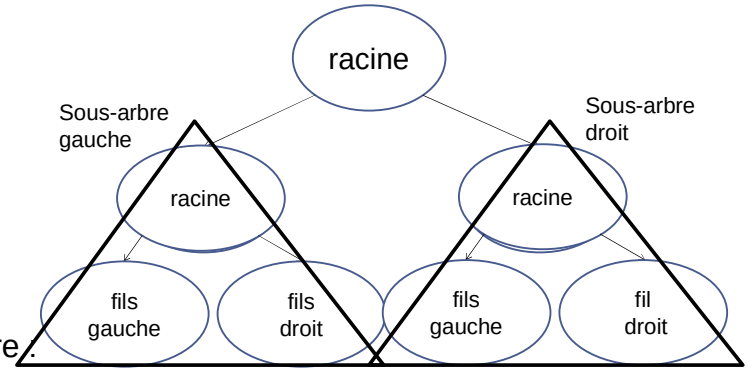
Les nœuds terminaux (*ceux qui ne possèdent aucun noeud*) sont appelés **feuilles.**
Les autres nœuds sont appelés **nœuds internes.**

Les nœuds internes peuvent être considérés comme racines de **sous-arbres.**
Le niveau d'un nœud dans un arbre est appelé **profondeur.**

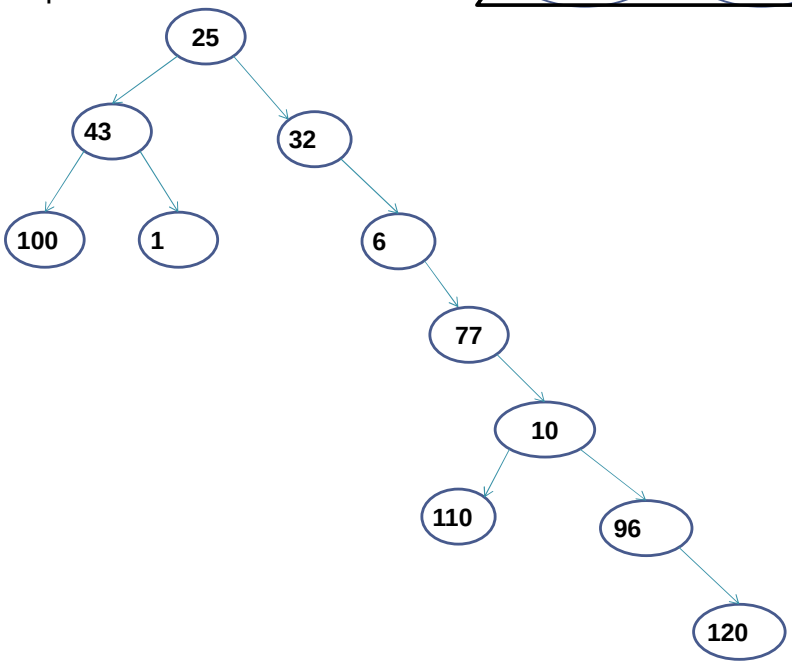
La **hauteur d'un arbre binaire** correspond à la distance entre la racine et la feuille la plus éloignée.



Chaque nœud peut être à son tour racine d'un sous-arbre :



Exemple de représentation d'un arbre binaire :



1. Parcours en profondeur des arbres binaires

Trois formes de parcours dits « en profondeur » sont proposées pour le parcours des nœuds d'un arbre binaires : (*algorithmes récursifs*)

- le parcours préfixe

```
parcourirPrefixe(Noeud n)
    afficher(n)
    Si nonVide(gauche(n))
       parcourirPrefixe (gauche(n))
    Si nonVide(droite(n))
       parcourirPrefixe (droite(n))
```

➔ 25, 43, 100, 1, 32, 6, 77, 10, 110, 96, 120 ←

- le parcours suffixe ou postfixe

```
parcourirSuffixe(Noeud n)
    Si nonVide(gauche(n))
       parcourirSuffixe (gauche(n))
    Si nonVide(droite(n))
       parcourirSuffixe (droite(n))
    afficher(n)
```

➔100, 1, 43, 110, 120, 96, 10, 77, 6, 32, 25 ←

- le parcours infixe

```
parcourirInfixe(Noeud n)
    Si nonVide(gauche(n))
       parcourirInfixe (gauche(n))
    afficher(n)
    Si nonVide(droite(n))
       parcourirInfixe (droite(n))
```

➔ 100, 43, 1, 25, 32, 6, 77, 110, 10, 96, 120 ←

2. Parcours en largeur des arbres binaires

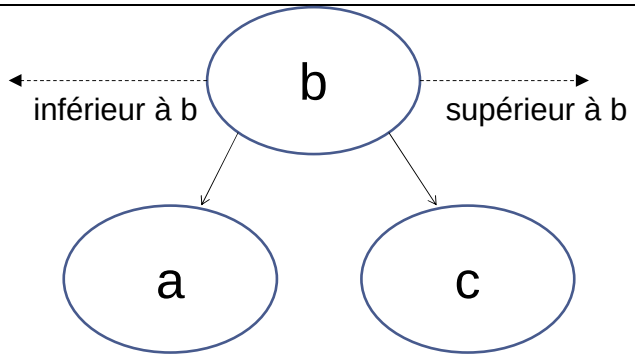
Le parcours en largeur consiste à parcourir les nœuds par niveau

C. Arbres binaires de recherche

Un **ABR**, **Arbre Binaire de Recherche** (*anglais : BST, Binary Search Tree*) est arbre binaire dans lequel **chaque nœud est porteur d'une valeur de clef unique**, et donc les clefs des nœuds fils doivent respecter une relation d'ordre telle que

- toutes les clefs des nœuds du sous-arbre gauche soient inférieures
- toutes les clefs des nœuds du sous-arbre droit soient supérieures.

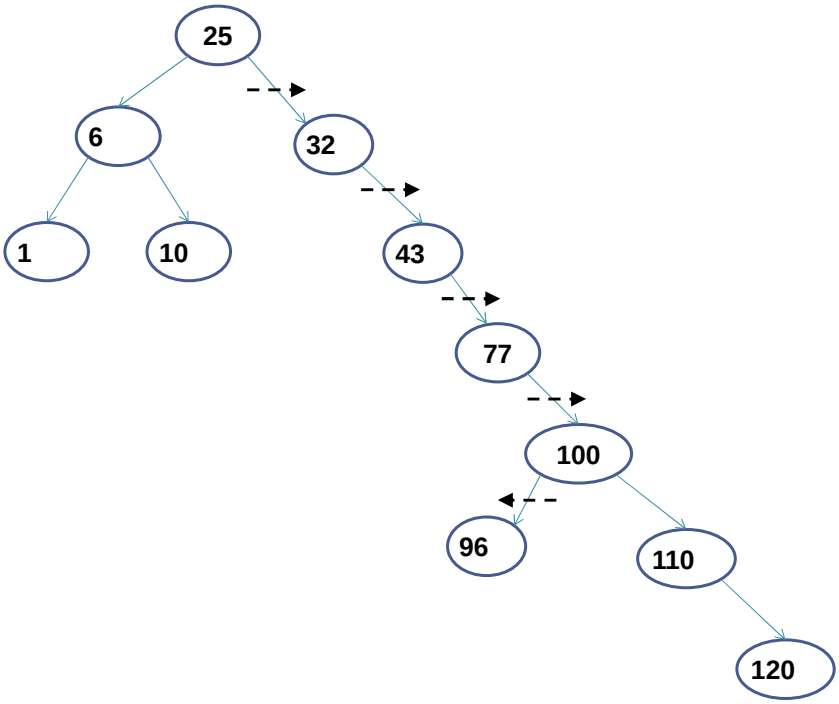
La localisation d'une valeur de clef dans un arbre binaire de recherche procède d'une manière similaire à la recherche dichotomique.



Soient les valeurs suivantes à stocker dans un arbre binaire de recherche :
1,6,10,25,32,43,77,96,100,110,120.

La recherche de la valeur de clef 96 va emprunter les différentes branches indiquées, par comparaison de la clef avec la valeur de chaque nœud :

- vers le fils gauche si la clef est inférieure
- ou vers le fils droit si elle est supérieure.



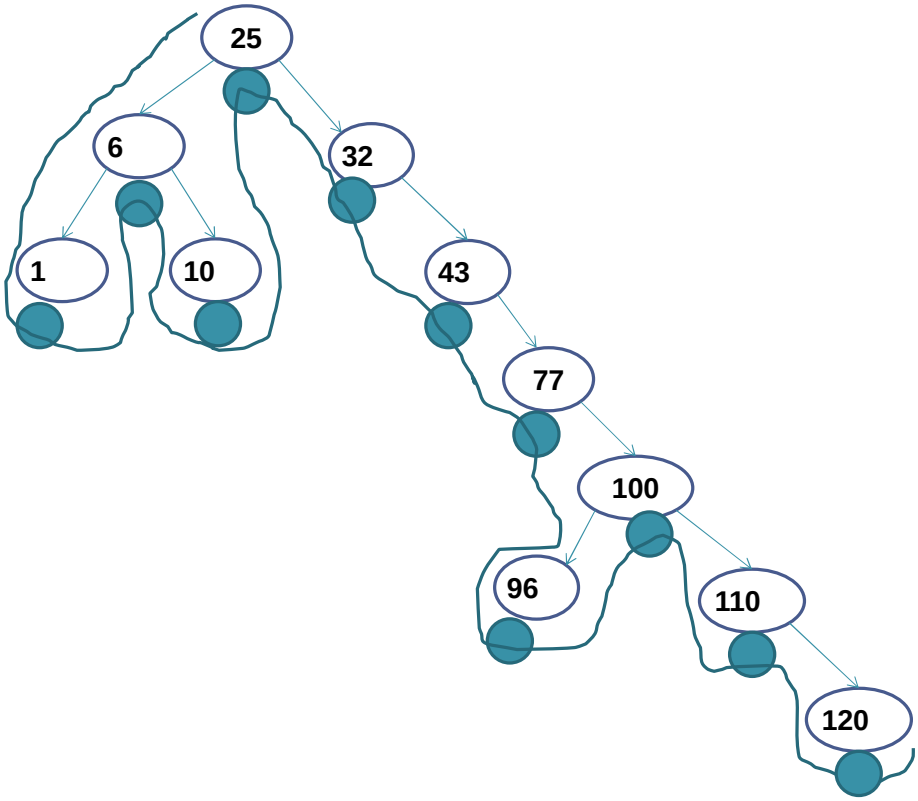
Dans le cas de la recherche d'une clef de valeur 98, le parcours de l'arbre serait identique mais avec un échec.

Dans le cas d'une recherche d'une clef de valeur 6, par exemple, la recherche nécessiterait le parcours de 2 nœuds seulement.

On voit donc que la performance de recherche dans cette forme d'arbre n'est pas garantie et dépend essentiellement de sa hauteur.

1. Parcours d'un arbre binaire de recherche

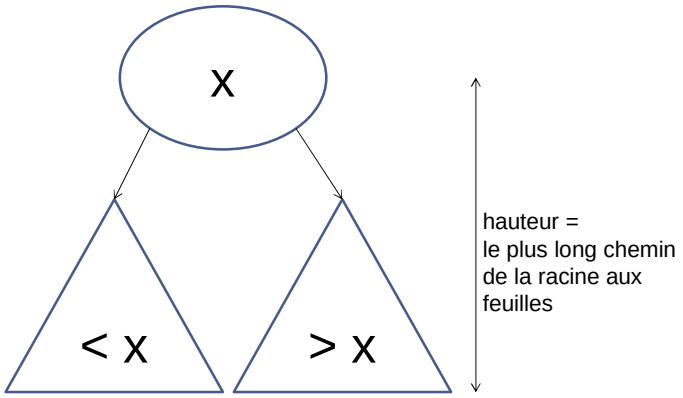
Le parcours infixe d'un arbre binaire de recherche fournit la séquence des clefs classées dans l'ordre croissant :' 1 – 6 – 10 – 25 – 32 – 43 – 77 – 96 – 100 – 110 -- 120

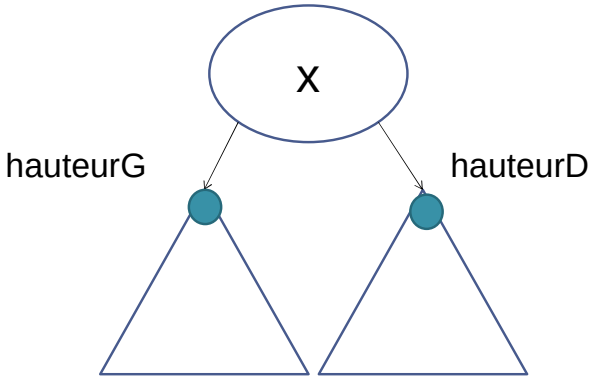


D. Arbres binaires de recherche équilibrés, ou arbres AVL[1]

Un **arbre AVL** (*anglais : AVL tree*) est arbre binaire de recherche **équilibré** dans lequel la différence de hauteur des 2 sous-arbres d'un nœud, appelé **facteur d'équilibrage** *(facteur d'équilibrage = hauteur sous-arbre gauche – hauteur sous-arbre droit*), ne soit jamais supérieure à 1, c'est-à-dire qu'elle ne peut être que l'une des 3 valeurs suivantes:

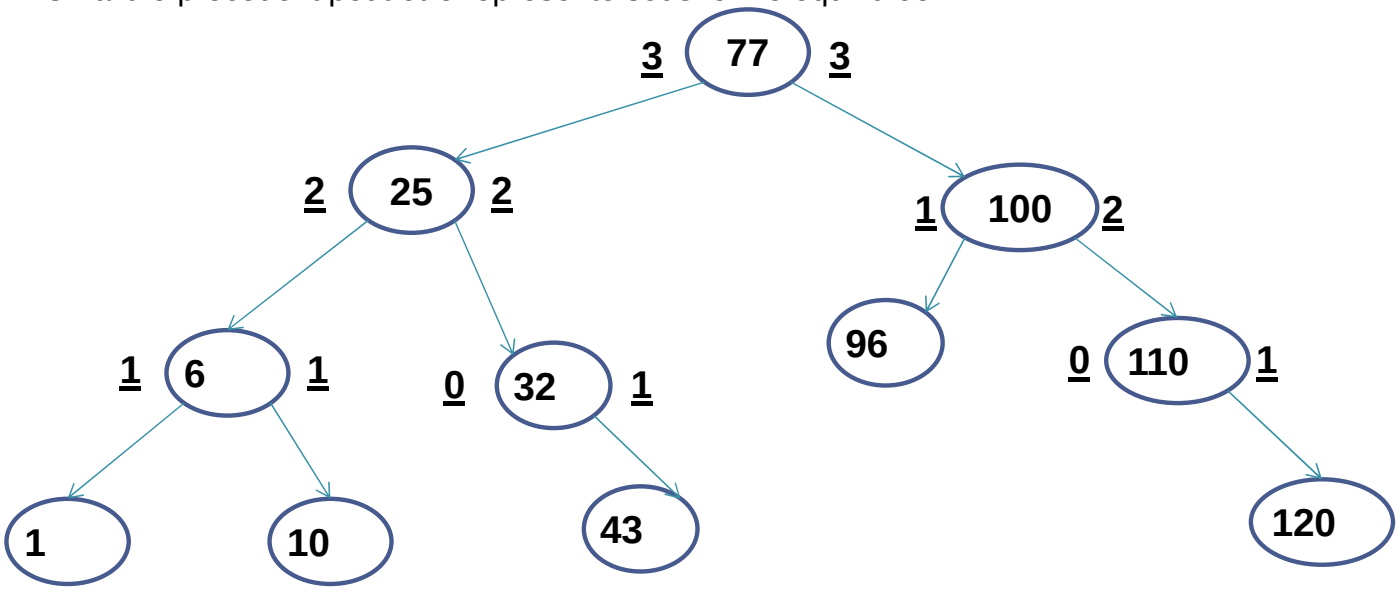
- -1 : sous-arbre droit plus haut de 1 niveau
- 0 : arbre parfaitement équilibré
- +1 : sous-arbre gauche plus haut de 1 niveau





abs(hauteurD – hauteurD) <= 1

Ainsi l'arbre précédent peut être représenté sous forme équilibrée :



1. Equilibrage des nœuds

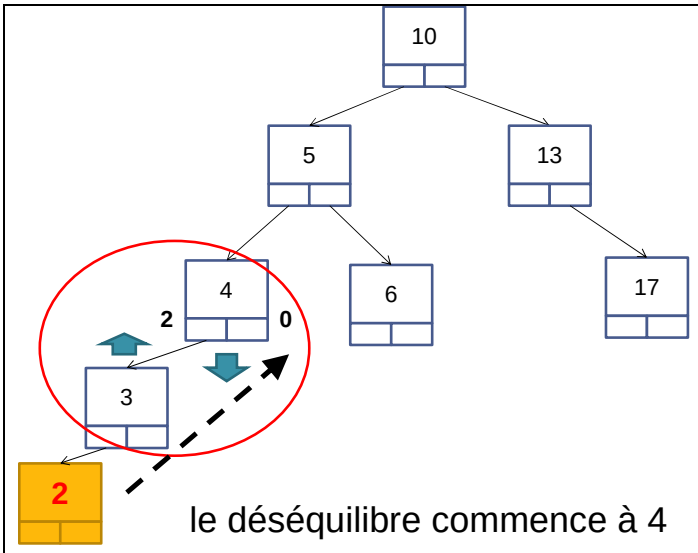
L'équilibrage des nœuds d'un arbre est réalisé à chaque opération d'insertion ou de suppression par un mécanisme de rotation (permutation) de 2 nœuds afin de maintenir la relation d'ordre des sous-arbres et le facteur d'équilibrage.
L'insertion d'un nœud doit s'opérer en 2 ou 3 étapes :

- placer le nouveau nœud au bon endroit, pour maintenir la relation d'ordre entre les clefs

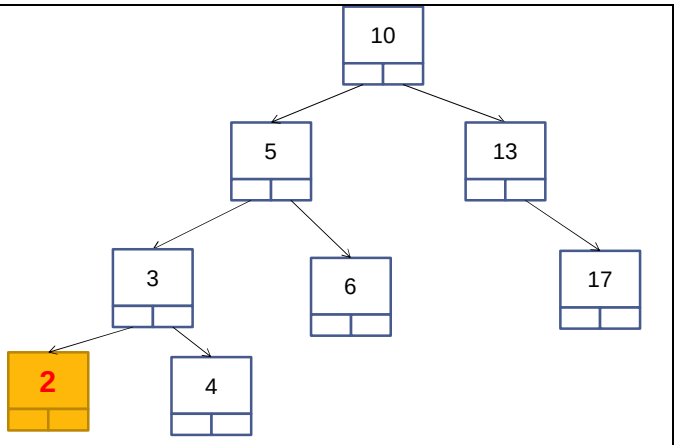
[1] Du nom de leurs inventeurs soviétiques : Adelson-Velsky and Landis

- remonter vers les arbres supérieurs afin de contrôler si l'insertion a été la cause d'un déséquilibre
- si c'est le cas, effectuer un mouvement de rotation afin de restaurer l'équilibre de l'arbre

a) Exemple 1 – insertion de la clef 2

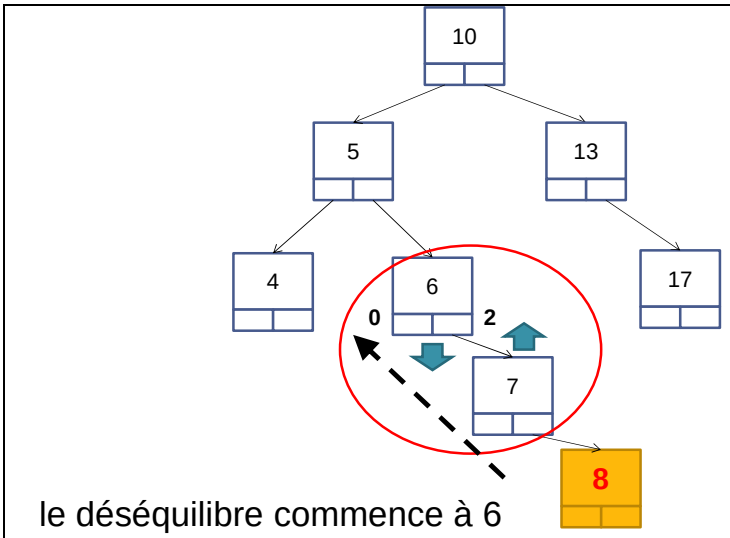


le déséquilibre commence à 4

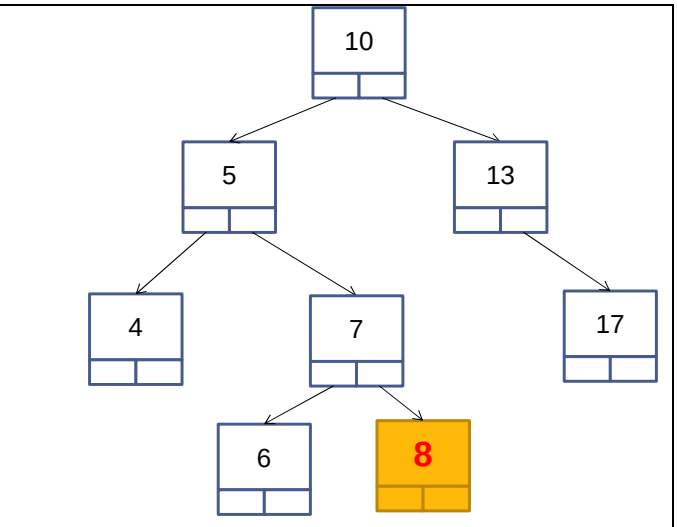


équilibré !

b) Exemple 2 – insertion de la clef 8

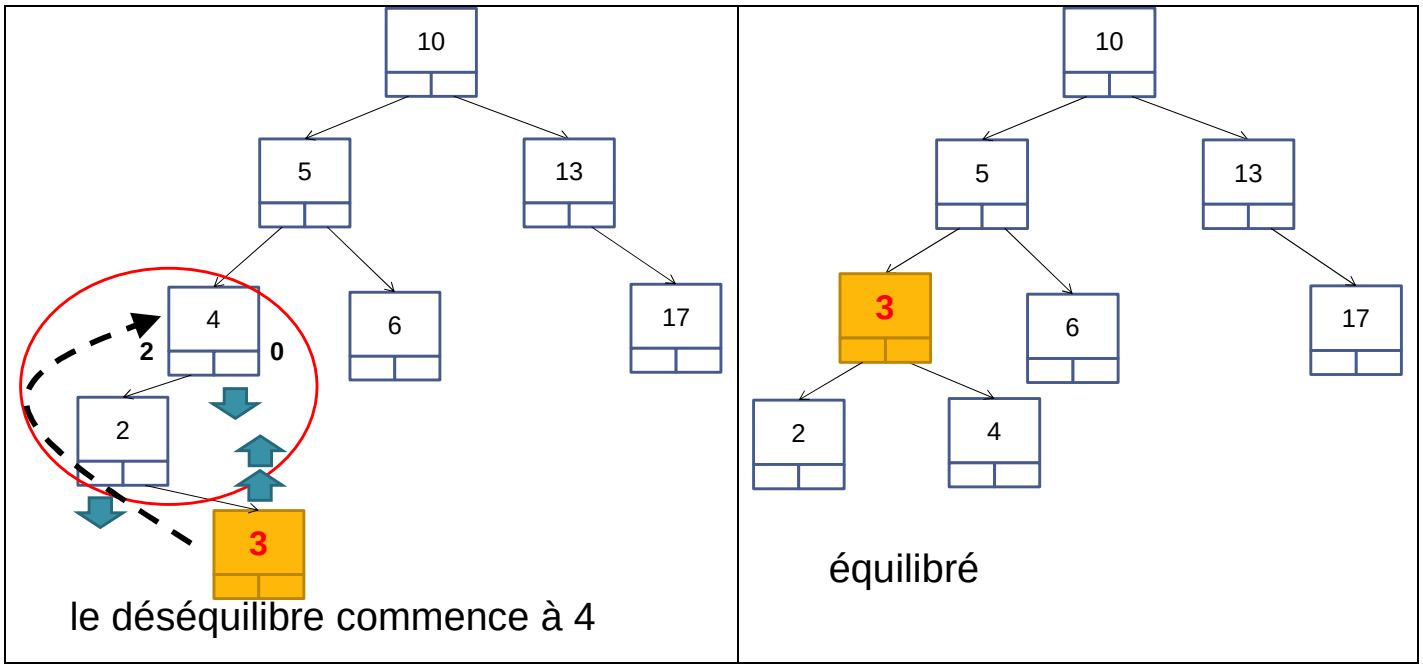


le déséquilibre commence à 6



équilibré

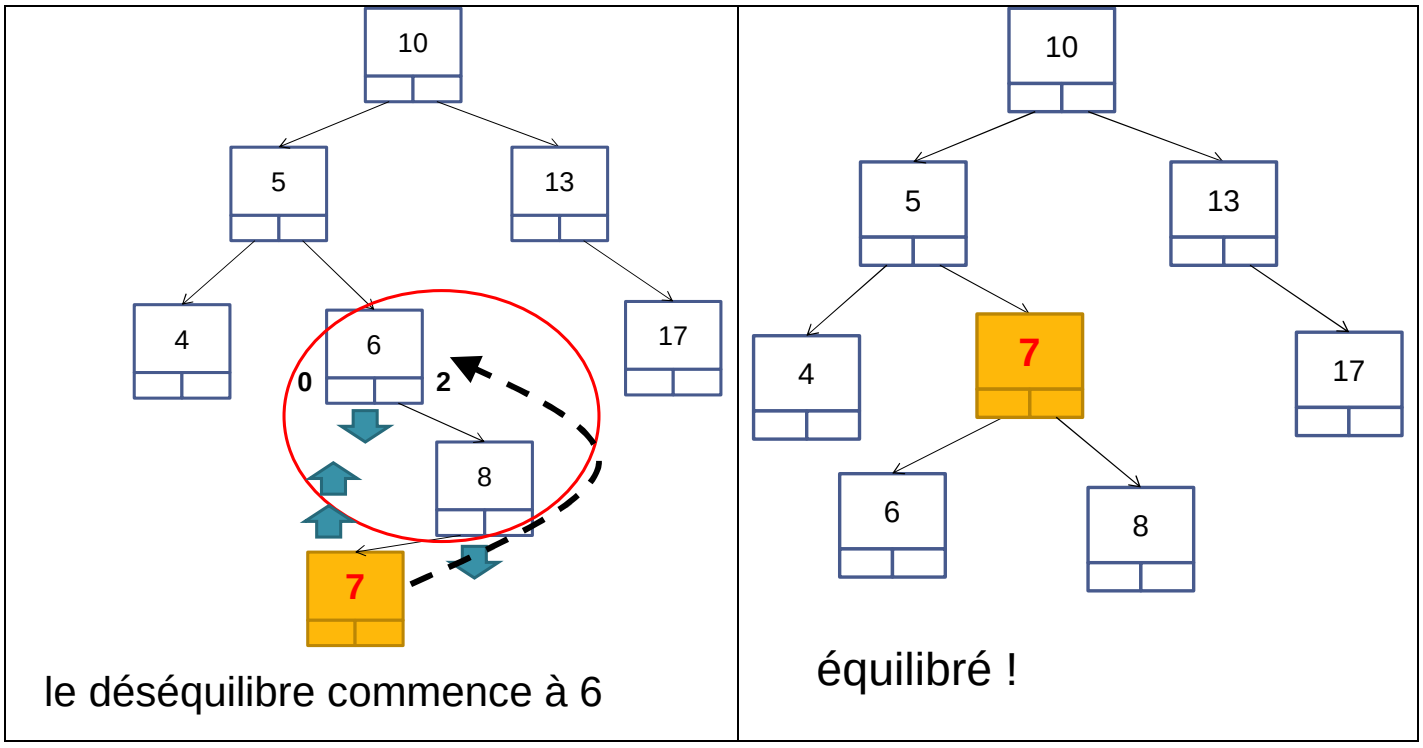
c) Exemple 3 – insertion de la clef 3



On a dans ce cas une double rotation :

- de 3 avec 2 : on revient alors dans la situation de l'exemple 1
- de 3 avec 4

d) Exemple 4 – insertion de la clef 7



On a dans ce cas une double rotation :

- de 7 avec 8 : on revient alors dans la situation de l'exemple 2
- de 7 avec 6

II. Les arbres appliquées aux bases de données

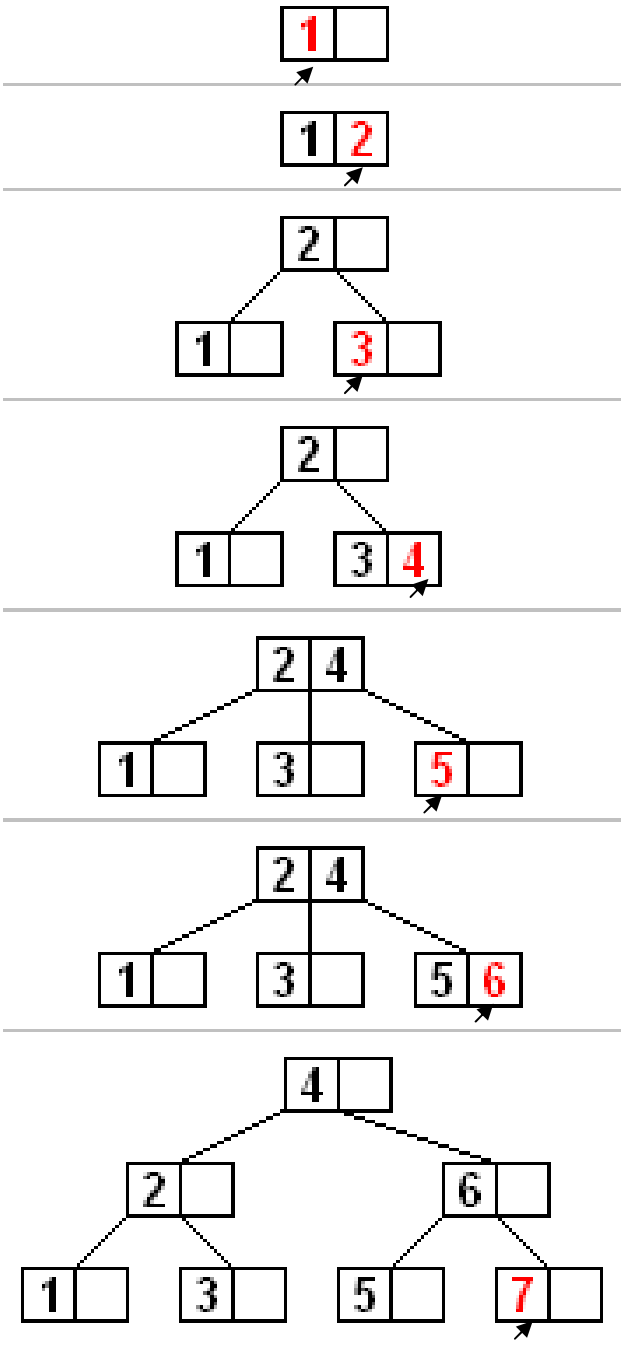
A. B-arbre

Un **B-arbre** (*anglais : B tree, pour Balanced Tree*) est une structure de données mise en œuvre dans les **bases de données** pour la construction des index.

Le B-arbre est un **arbre de recherche toujours parfaitement équilibré**.

Un B-arbre peut comporter **plusieurs clefs par nœuds** (*afin d'optimiser les accès aux supports de persistance –disque dur, par exemple--).*
Dans un B-arbre d'ordre m, chaque nœud a entre m et 2m entrées, sauf la racine qui doit avoir au moins 1 entrée

Chaque nœud occupe une page de base de données.



L'évolution de la construction de l'arbre B à droite (*arbre dont chaque nœud possède 2 clefs et qui définit 3 intervalles de valeurs*) montre les différentes opérations de rotation réalisées lors de l'insertion d'une nouvelle clef afin de maintenir l'arbre toujours équilibré.

Le principe d'insertion est le suivant :

- rechercher la feuille de placement

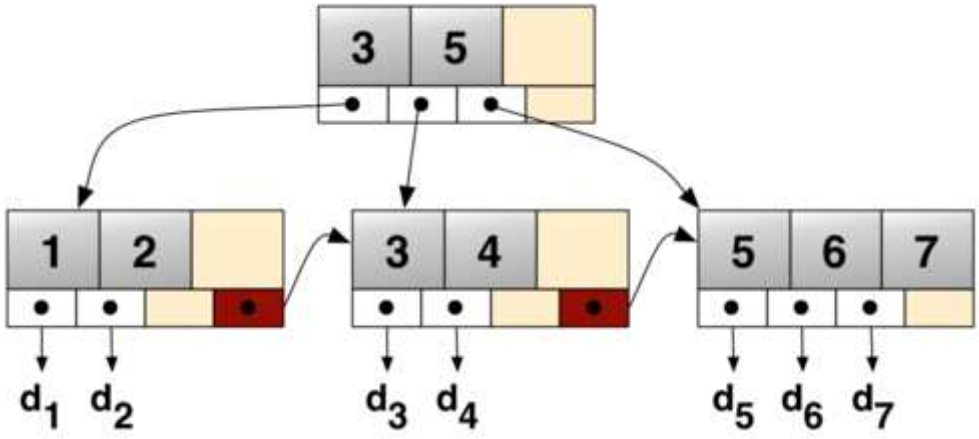
puis, récursivement :

- si la feuille n'est pas pleine, insérer la clef à sa position normale
- sinon (feuille pleine avec 2m clé) :
  - laisser les m clefs les plus petites dans le nœud
  - créer un nouveau nœud et y placer les m clefs les plus grandes
  - remonter la clef médiane dans le nœud père

B. Arbres B+

Un **arbre B+** (*anglais : B+ tree*) est un B-arbre dans lequel

- les nœuds internes contiennent les clefs permettant la recherche
- et les feuilles stockent toutes les clefs et référencent les données dans les pages de base de données.



Exemple simpliste[2] d'un index primaire[3] d'une base de données MySQL/InnoDB :

- Taille des pages : 16ko, soit : 16*1024 octets
- Index sur une valeur de clef primaire de type INT : 4 octets
- *(Pointeur vers un nœud fils : non considéré ici par simplification)*
- Nombre de clefs d'un nœud : 16*1024 / 4 octets par clef = 4096 clefs
- en considérant un index à 2 niveaux :
  - chaque nœud peut avoir 4097 fils, chacun ayant 4096 positions de clef, soit 4097 * 4096 = 16.781.312 valeurs de clefs.
- On voit donc, qu'en 2 accès disques, on peut accéder à plus de 16 millions de valeurs de clefs.

Ainsi pour localiser une ligne de données ayant une clef K, le serveur interroge le nœud racine jusqu'à ce qu'il trouve une clef plus grande ou égale à K, puis il suit le pointeur vers le nœud fils et ainsi de suite jusqu'à arriver à un nœud feuille. Ce nœud feuille contient les pointeurs vers les pages de données où la ligne avec la clef K peut être trouvée.

Plus d'infos sur la structure des pages : http://dev.mysql.com/doc/refman/5.1/en/innodb-table-and-index.html
Plus d'infos sur MySQL : Pro MySQL, Jay Pipes, Mickael Kruckenberg, (téléchargeable sur http://www.it-ebooks.info/book/1709/) page 91

[2] La structure d'une page de base de données comporte de nombreuses informations en plus des informations à mémoriser
[3] Un index secondaire est relatif à une colonne non clef primaire.