

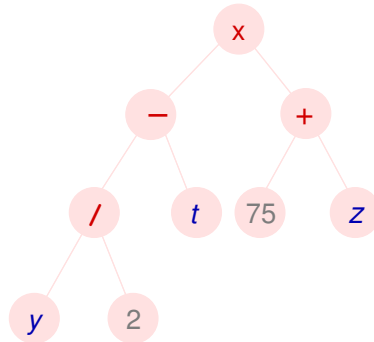
Chapitre 5

Arbres binaires de recherche, arbres lexicographiques

Les arbres et les structures arborescentes sont très utilisés en informatique. D'une part les informations sont souvent hiérarchisées, et se présentent donc naturellement sous une forme arborescente, et d'autre part de nombreuses structures de données parmi les plus efficaces sont représentées par des arbres (les tas, les arbres binaires de recherches, les B-arbres, les forêts, ...).

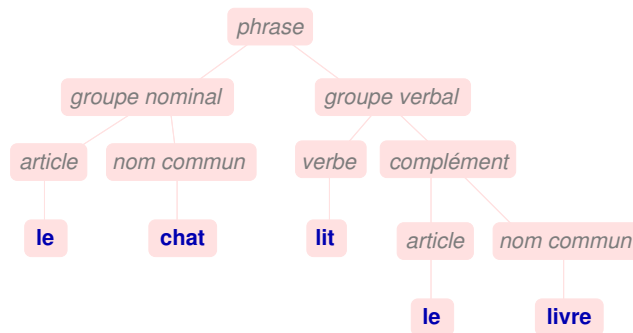
5.1 Quelques exemples

Expressions arithmétiques. On représente souvent des expressions arithmétiques avec des arbres étiquetés par des opérateurs, des constantes et des variables. La structure de l'arbre élimine les ambiguïtés qui peuvent apparaître en fonction des priorités des opérateurs et qui sont supprimées en utilisant des parenthèses.

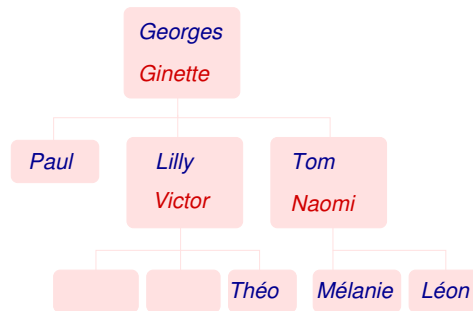


$$(y/2 - t) \times (75 + z)$$

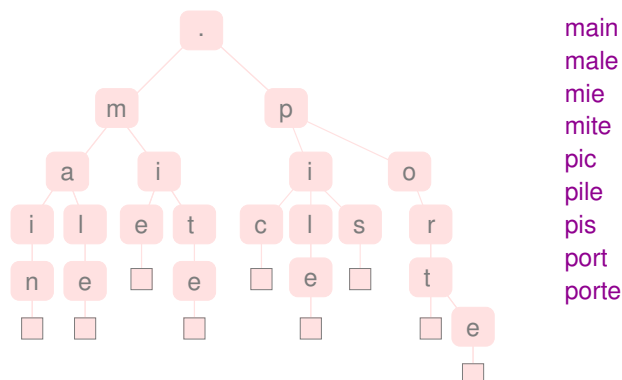
Arbres syntaxiques. Un arbre syntaxique représente l'analyse d'une phrase à partir d'un ensemble de règles qui constitue la grammaire : une phrase est composée d'un groupe nominal suivi d'un groupe verbal, un groupe nominal peut-être constitué d'un article et d'un nom commun,...



Arbres généalogiques. Un arbre généalogique (descendant dans le cas présent) représente la descendance d'une personne ou d'un couple. Les noeuds de l'arbre sont étiquetés par les membres de la famille et leurs conjoints. L'arborescence est construite à partir des liens de parenté (les enfants du couple).



Arbre lexicographique. Un arbre lexicographique, ou arbre en parties communes, ou dictionnaire, représente un ensemble de mots. Les préfixes communs à plusieurs mots apparaissent une seule fois dans l'arbre, ce qui se traduit par un gain d'espace mémoire. De plus la recherche d'un mot est assez efficace, puisqu'il suffit de parcourir une branche de l'arbre en partant de la racine, en cherchant à chaque niveau parmi les fils du noeud courant la lettre du mot de rang correspondant.

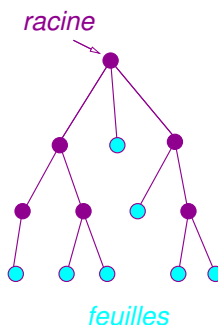


5.2 Définitions et terminologie

Définition. Un arbre est un ensemble organisé de noeuds dans lequel chaque noeud a un *père* et un seul, sauf un noeud que l'on appelle la *racine*.

Si le noeud p est le père du noeud f , nous dirons que f est un *fils* de p , et si le noeud p n'a pas de fils nous dirons que c'est une *feuille*. Chaque noeud porte une *étiquette* ou *valeur* ou *clé*. On a l'habitude, lorsqu'on dessine un

arbre, de le représenter avec la tête en bas, c'est-à-dire que la racine est tout en haut, et les noeuds fils sont représentés en-dessous du noeud père.



Un noeud est défini par son *étiquette* et ses *sous-arbres*. On peut donc représenter un arbre par un n -uplet $\langle e, a_1, \dots, a_k \rangle$ dans lequel e est l'étiquette portée par le noeud, et a_1, \dots, a_k sont ses sous-arbres. Par exemple l'arbre correspondant à l'expression arithmétique $(y/2 - t) \times (75 + z)$ sera représenté par

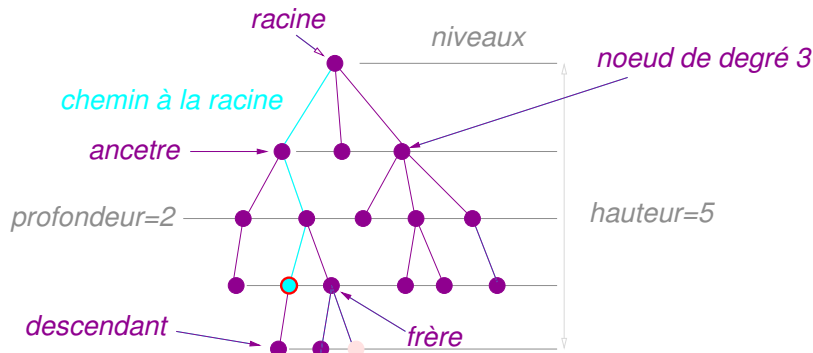
$$\langle \times, \langle -, \langle /, \langle y, \langle 2 \rangle \rangle, \langle t \rangle \rangle, \langle +, \langle 75 \rangle, \langle z \rangle \rangle \rangle$$

On distingue les *arbres binaires* des arbres généraux. Leur particularité est que les fils sont singularisés : chaque noeud a un fils gauche et un fils droit. L'un comme l'autre peut être un arbre vide, que l'on notera $\langle \rangle$. L'écriture d'un arbre s'en trouve modifiée, puisqu'un noeud a toujours deux fils. En reprenant l'expression arithmétique précédente, l'arbre binaire qui la représente s'écrit

$$\langle \times, \langle -, \langle /, \langle y, \langle \rangle, \langle \rangle \rangle, \langle 2, \langle \rangle, \langle \rangle \rangle \rangle, \langle t, \langle \rangle, \langle \rangle \rangle \rangle, \langle +, \langle 75, \langle \rangle, \langle \rangle \rangle, \langle z, \langle \rangle, \langle \rangle \rangle \rangle \rangle$$

On utilise pour les arbres une terminologie inspirée des liens de parenté :

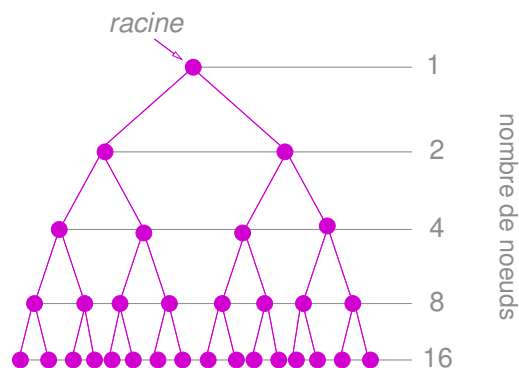
- les *descendants* d'un noeud p sont les noeuds qui apparaissent dans ses sous-arbres,
- un *ancêtre* d'un noeud p est soit son père, soit un ancêtre de son père,
- le chemin qui relie un noeud à la racine est constitué de tous ses ancêtres (c'est-à-dire de son père et des noeuds du chemin qui relie son père à la racine),
- un *frère* d'un noeud p est un fils du père de p , et qui n'est pas p .



Les noeuds d'un arbre se répartissent par *niveaux* : le premier niveau (par convention ce sera le niveau 0) contient la racine seulement, le deuxième niveau contient les deux fils de la racine, ..., les noeuds du niveau k sont les fils des noeuds du niveau $k - 1$, ... La *hauteur* d'un arbre est le nombre de niveaux de ses noeuds. C'est donc aussi le nombre de noeuds qui jalonnent la branche la plus longue. Attention, la définition de la hauteur varie en fonction des auteurs. Pour certains la hauteur d'un arbre contenant un seul noeud est 0.

Arbres binaires : hauteur, nombre de noeuds et nombre de feuilles.

Un arbre binaire est *complet* si toutes ses branches ont la même longueur et tous ses noeuds qui ne sont pas des feuilles ont deux fils. Soit A un arbre binaire complet. Le nombre de noeuds de A au niveau 0 est 1, le nombre de noeuds au niveau 1 est 2, ..., et le nombre de noeuds au niveau p est donc 2^p . En particulier le nombre de feuilles est donc 2^{h-1} si h est le nombre de niveaux.



Le nombre total de noeuds d'un arbre binaire complet de h niveaux est

$$\sum_{i=0}^{h-1} 2^i = 2^h - 1$$

On en déduit que la hauteur $ht(A)$ (le nombre de niveaux) d'un arbre binaire A contenant n noeuds est au moins égale à

$$\lfloor \log_2 n \rfloor + 1$$

Preuve. Si A est arbre de hauteur h et comportant n noeuds, pour tout entier m , on a : $h \leq m-1 \Rightarrow n < 2^{m-1}$. Par contraposée, on a : $n \geq 2^{m-1} \Rightarrow h \geq m$. Le plus grand entier m vérifiant $n \geq 2^{m-1}$ est $\lfloor \log_2 n \rfloor + 1$. On a donc $h \geq \lfloor \log_2 n \rfloor + 1$.

La hauteur minimale $\lfloor \log_2 n \rfloor + 1$ est atteinte lorsque l'arbre est complet. Pour être efficaces, les algorithmes qui utilisent des arbres binaires font en sorte que ceux ci soient équilibrés (voir les tas ou les AVL-arbres par exemple).

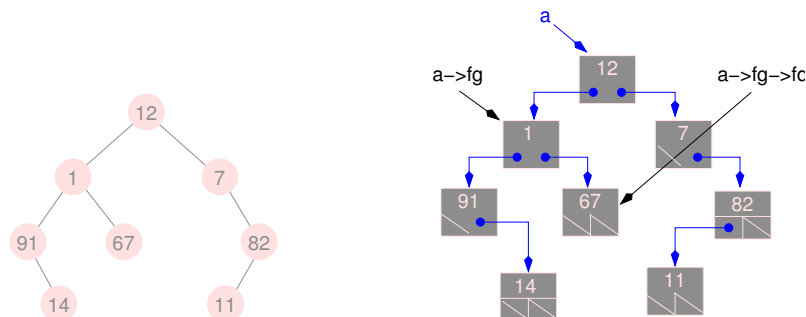
Les arbres en théorie des graphes. En théorie des graphes un arbre est un graphe *connexe* et *sans cycles*, c'est-à-dire qu'entre deux sommets quelconques du graphe il existe un chemin et un seul. On parle dans ce cas d'arbre non planté, puisqu'il n'y a pas de sommet particulier qui joue le rôle de racine. Les sommets sont voisins les uns des autres, il n'y a pas de hiérarchie qui permet de dire qu'un sommet est le père (ou le fils) d'un autre sommet. On démontre qu'un graphe de n sommets qui a cette propriété contient exactement $n - 1$ arêtes, et que l'ajout d'une nouvelle arête crée un cycle, tandis que le retrait d'une arête le rend non connexe.

5.3 Arbres binaires

En C on utilise en général des pointeurs pour représenter les liens entre un noeud et ses fils dans un arbre binaire. Il existe plusieurs façons de définir le type `ARBRE`. Nous proposons la définition suivante, qui permet d'imbriquer différents types les uns dans les autres (par exemple si on veut des arbres dont les valeurs sont des listes chaînées dont les valeurs sont elles-mêmes des arbres, ...).

```
typedef struct noeud * ARBRE;
struct noeud
{
    TYPE_VALEUR val;
    ARBRE fg, fd;
};
```

Un arbre en C est donc désigné par l'adresse de sa racine. Il est facile d'accéder à un noeud quelconque de l'arbre à partir de la racine, en suivant les liens explicites vers le fils droit ou le fils gauche.



La construction d'un arbre consiste, à partir d'un arbre vide, à insérer des nouveaux noeuds. La fonction suivante fait l'allocation d'un noeud et l'initialisation des champs de la structure. Il est recommandé d'utiliser cette fonction chaque fois qu'un nouveau noeud doit être créé.

```
ARBRE CreerNoeud(TYPE_VALEUR v, ARBRE fg, ARBRE fd)
{
    ARBRE p;
    p = malloc(sizeof(struct noeud));
    assert(p != NULL);
    p->val = v;
    p->fg = fg;
    p->fd = fd;
    return p;
}
```

Arbres binaires : parcours. Le principe est simple : pour parcourir l'arbre *a*, on parcourt récursivement son sous-arbre gauche, puis son sous-arbre droit. Ainsi le sous-arbre gauche sera exploré dans sa totalité avant de commencer l'exploration du sous-arbre droit.

```

void Parcours(ARBRE a)
{
    if (a != NULL)
    {
        /* traitement avant */
        Parcours (a->fg);
        /* traitement entre */
        Parcours (a->fd);
        /* traitement apres */
    }
}

```

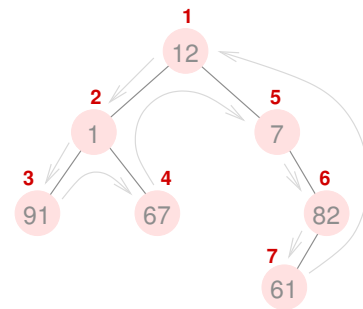
Dans ce schéma l'exploration descend d'abord visiter les noeuds les plus à gauche : on parle de *parcours en profondeur d'abord*. On distingue le parcours préfixe, le parcours infixe et le parcours postfixe. La différence entre ces parcours tient uniquement à l'ordre dans lequel sont traités les noeuds du sous-arbre gauche, le noeud courant, et les noeuds du sous-arbre droit.

Parcours préfixe : la valeur du noeud courant est traitée *avant* les valeurs figurant dans ses sous-arbres. Si le traitement consiste simplement à afficher la valeur du noeud, le parcours préfixe de l'arbre ci-dessous produirait l'affichage 12 1 91 67 7 82 61.

```

void ParcoursPrefixe(ARBRE a)
{
    if (a != NULL)
    {
        TraiterLaValeur (a->val);
        ParcoursPrefixe (a->fg);
        ParcoursPrefixe (a->fd);
    }
}

```

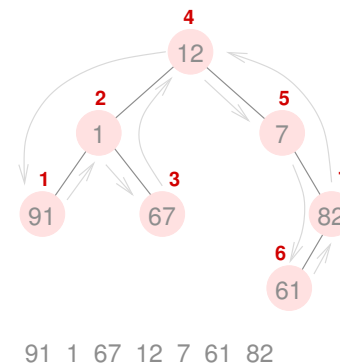


Parcours infixe : la valeur du noeud courant est traitée *après* les valeurs figurant dans son sous-arbre gauche et *avant* les valeurs figurant dans son sous-arbre droit.

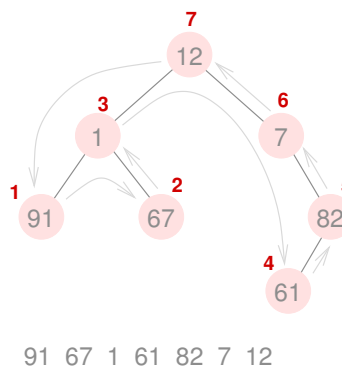

```

void ParcoursInfixe(ARBRE a)
{
    if (a != NULL)
    {
        ParcoursInfixe (a->fg);
        TraiterLaValeur (a->val);
        ParcoursInfixe (a->fd);
    }
}

```



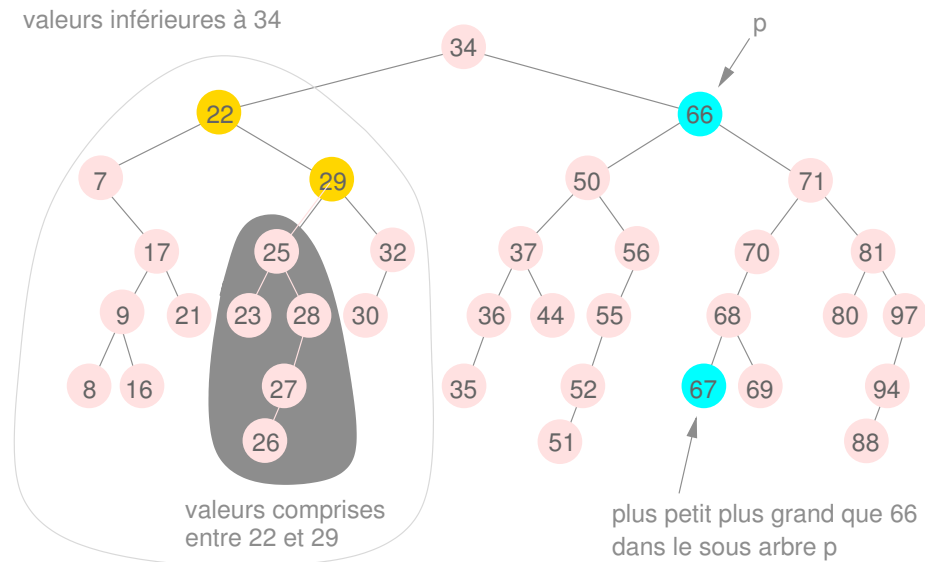
Parcours postfixé : la valeur du noeud courant est traitée *après* les valeurs de ses sous-arbres.



5.4 Arbres binaires de recherche.

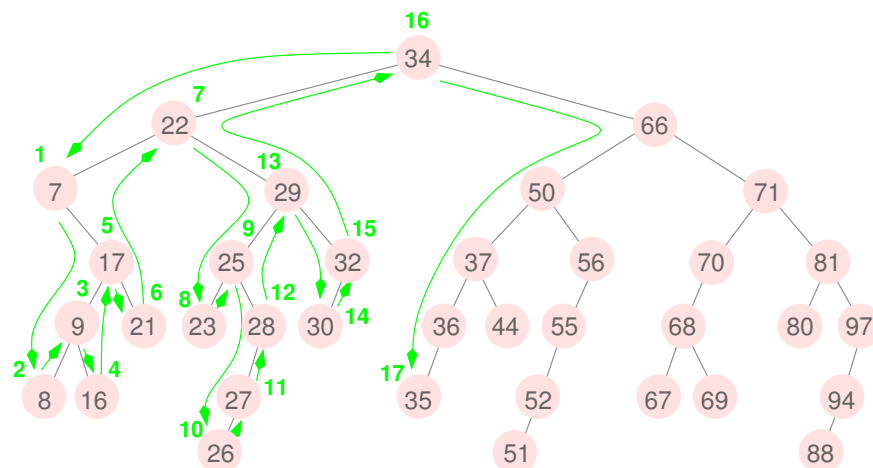
Un arbre binaire de recherche (ou ABR) est une structure de donnée qui permet de représenter un ensemble de valeurs si l'on dispose d'une relation d'ordre sur ces valeurs. Les opérations caractéristiques sur les arbres binaires de recherche sont l'**insertion**, la **suppression**, et la **recherche** d'une valeur. Ces opérations sont peu coûteuses si l'arbre n'est pas trop déséquilibré.

Soit E un ensemble muni d'une relation d'ordre, et soit A un arbre binaire portant des valeurs de E . L'arbre A est un *arbre binaire de recherche* si pour tout noeud p de A , la valeur de p est plus grande que les valeurs figurant dans son sous-arbre gauche et plus petite que les valeurs figurant dans son sous-arbre droit.



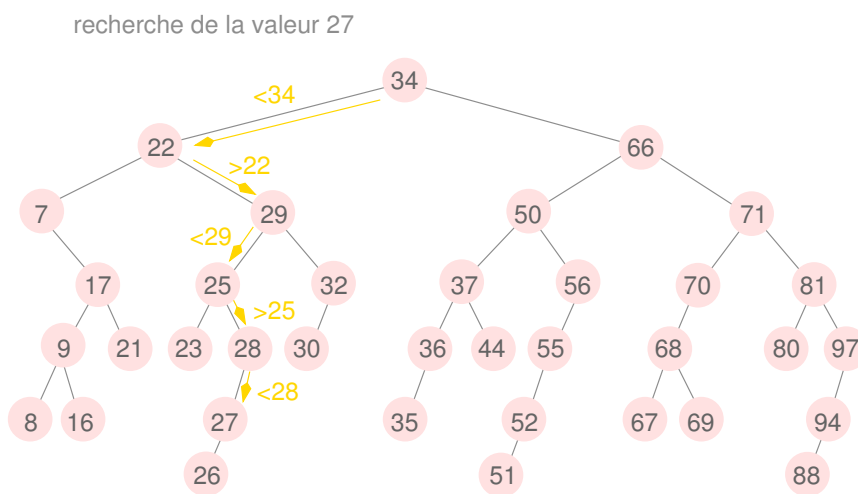
Pour accéder à la clé la plus petite (resp. la plus grande) dans un ABR il suffit de descendre sur le fils gauche (resp. sur le fils droit) autant que possible. Le dernier noeud visité, qui n'a pas de fils gauche (resp. droit), porte la valeur la plus petite (resp. la plus grande) de l'arbre.

Le parcours infixe de l'arbre produit la suite ordonnée des clés



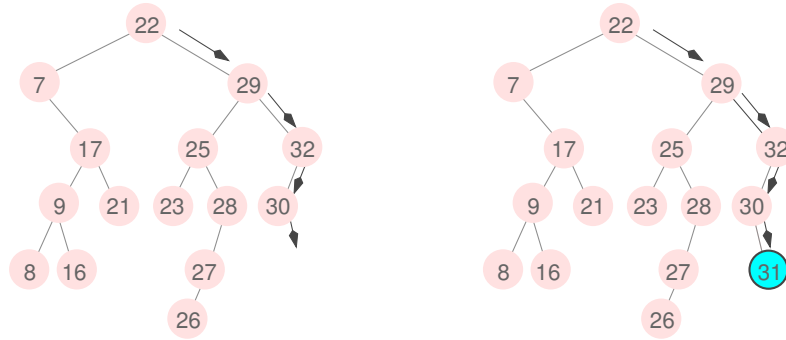
7 8 9 16 17 21 22 23 25 26 27 28 29 30 32 34 35 36 37 ...

Recherche d'une valeur. La recherche d'une valeur dans un ABR consiste à parcourir une branche en partant de la racine, en descendant chaque fois sur le fils gauche ou sur le fils droit suivant que la clé portée par le noeud est plus grande ou plus petite que la valeur cherchée. La recherche s'arrête dès que la valeur est rencontrée ou que l'on a atteint l'extrémité d'une branche (le fils sur lequel il aurait fallu descendre n'existe pas).



```
ARBRE recherche (TYPE_VALEUR x, ARBRE a)
{
    while ((a != NULL) && (x != a->val))
        if (x < a->val)
            a = a->fg;
        else
            a = a->fd;
    return a;
}
```

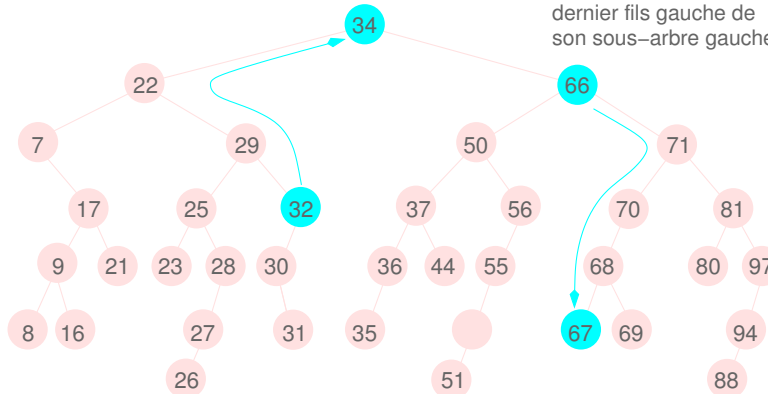
Insertion d'une nouvelle valeur. Le principe est le même que pour la recherche. Un nouveau noeud est créé avec la nouvelle valeur et inséré à l'endroit où la recherche s'est arrêtée.



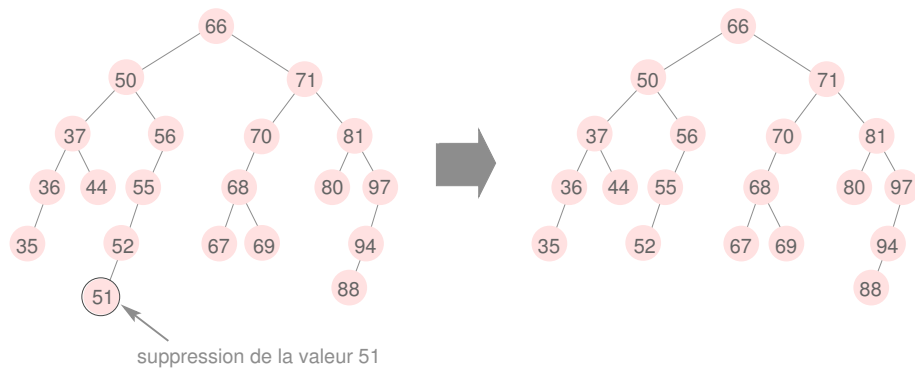
Recherche du successeur d'un noeud. Étant donné un noeud p d'un arbre A , le successeur de p si il existe, est le noeud de A qui porte comme valeur la plus petite des valeurs qui figurent dans A et qui sont plus grandes que la valeur de p . Si p possède un fils droit, son successeur est le noeud le plus à gauche dans son sous-arbre droit (on y accède en descendant sur le fils gauche autant que possible). Si p n'a pas de fils droit alors son successeur est le premier de ses ascendants tel que p apparaît dans son sous-arbre gauche. Si cet ascendant n'existe pas c'est que p portait la valeur la plus grande dans l'arbre. Remarquez qu'il est nécessaire d'avoir pour chaque noeud un lien vers son père pour mener à bien cette opération.

32 n'a pas de fils droit :
son successeur est 34, premier ascendant de 32
tel que 32 figure dans son sous-arbre gauche

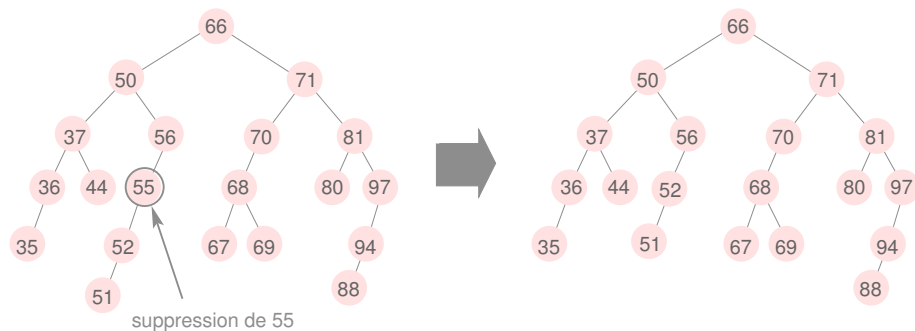
66 a un fils droit :
son successeur est 67
dernier fils gauche de
son sous-arbre gauche



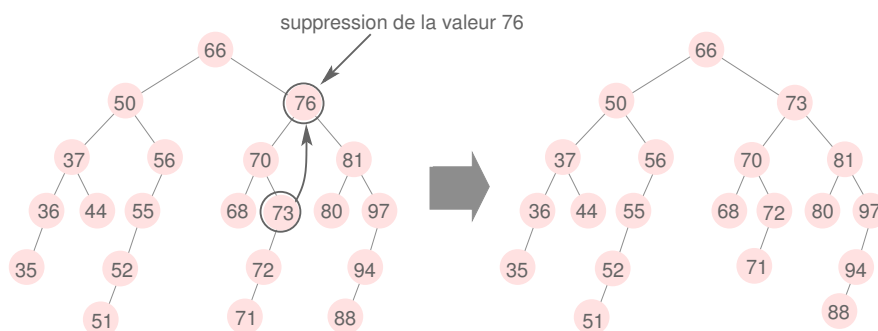
Suppression d'un noeud. L'opération dépend du nombre de fils du noeud à supprimer.



Cas 1 : le noeud à supprimer n'a pas de fils, c'est une feuille. Il suffit de *décrocher* le noeud de l'arbre, c'est-à-dire de l'enlever en modifiant le lien du père, si il existe, vers ce fils. Si le père n'existe pas l'arbre devient l'arbre vide.



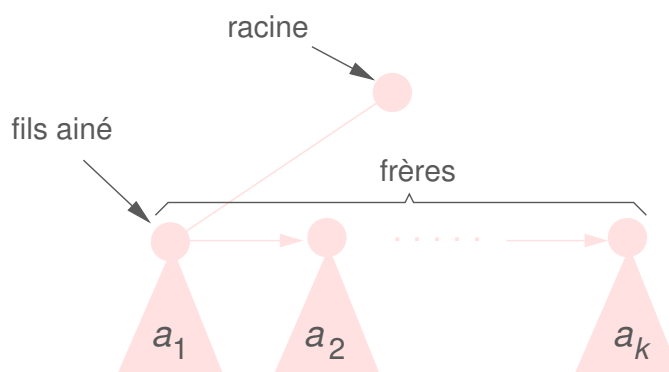
Cas 2 : le noeud à supprimer a un fils et un seul. Le noeud est *décroché* de l'arbre comme dans le cas 1. Il est remplacé par son fils unique dans le noeud père, si ce père existe. Sinon l'arbre est réduit au fils unique du noeud supprimé.



Cas 3 : le noeud à supprimer p a deux fils. Soit q le noeud de son sous-arbre gauche qui a la valeur la plus grande (on peut prendre indifféremment le noeud de son sous-arbre droit de valeur la plus petite). Il suffit de recopier la valeur de q dans le noeud p et de décrocher le noeud q . Puisque le noeud q a la valeur la plus grande dans le fils gauche, il n'a donc pas de fils droit, et peut être décroché comme on l'a fait dans les cas 1 et 2.

5.5 Arbres généraux ou n -aires

Un arbre n -aire est un arbre dans lequel le nombre de fils d'un noeud n'est pas limité. On les représente avec des pointeurs, en liant les fils d'un même noeud entre eux, comme dans une liste chaînée. Ainsi, à partir d'un noeud quelconque, on accède à son fils aîné (la liste de ses fils) et à son frère.



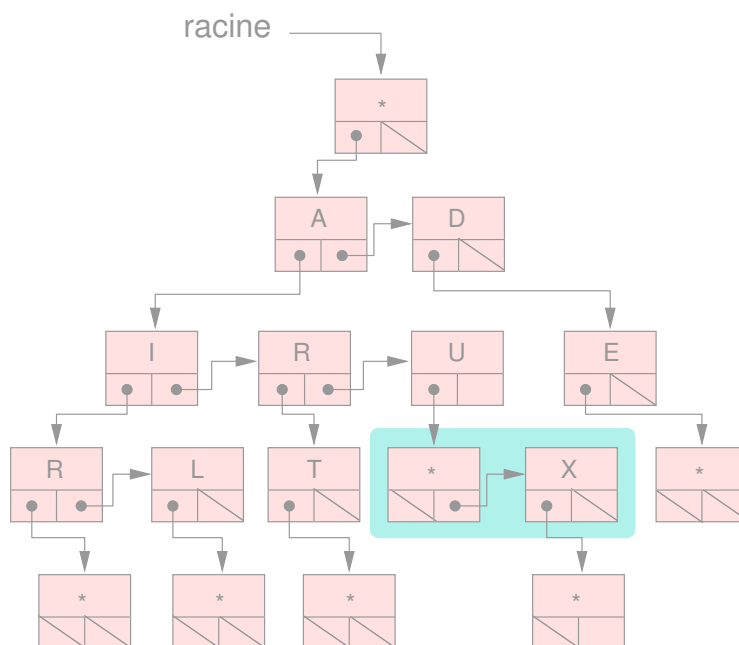
On définit en C les arbres n -aires de la façon suivante :

```
typedef struct noeud * ARBRE;

struct noeud
{
    TYPE_VALEUR val;
    ARBRE fa, fr;
};
```

5.6 Exercices

Les arbres lexicographiques, ou arbres en parties communes, ou dictionnaires, sont un exemple d'arbres n -aires. Ils sont utilisés pour représenter un ensemble de mots, en optimisant l'espace mémoire occupé. Le principe consiste à représenter une seule fois les préfixes communs à plusieurs mots. Pour éliminer toute ambiguïté lorsqu'un mot est le préfixe d'un autre, on signale les fins des mots à l'aide d'un caractère particulier (*) dans la figure ci-dessous, mais en C le caractère de fin de chaîne '\0' est tout à fait adapté).



Question 1. Ecrivez les fonctions d'insertion et de recherche d'un mot dans un arbre lexicographique.

Question 2. Ecrivez une fonction qui affiche tous les mots représentés dans l'arbre.

Question 3. Ecrivez une fonction qui permet de supprimer un mot de l'arbre.