

Initiation à la programmation Java

IP2 - Séance No 9

Yan Jurski

Dernière partie du cours

Le plan que nous avons suivi :

- Première partie : modélisation objet + style statique/non statique
- Deuxième partie : les listes + la récursion
- Troisième partie : les arbres (3 séances - 2 cette année ?)

Rappel sur les listes doublement chaînées

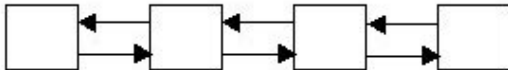
(et introduction indirecte aux arbres)

- Avec l'enregistreur à bande (bidirectionnel) nous avons vu :

Fichier CelluleTape.java

```
public class CelluleTape{  
    private E content;  
    private CelluleTape next;  
    private CelluleTape previous;  
}
```

- La syntaxe est associée à une propriété sous-jacente :
 - si $\text{next} \neq \text{null}$ alors $\text{this.next.previous} = \text{this}$
 - si $\text{previous} \neq \text{null}$ alors $\text{this.previous.next} = \text{this}$
- Les modèles sont linéaires : des listes chaînées



- Avoir d'autres liaisons serait considéré comme une erreur

Rappel sur les listes doublement chaînées

(et introduction indirecte aux arbres)

- Avec l'enregistreur à bande (bidirectionnel) nous avons vu aussi :

Fichier MonMagneto.java

```
public class MonMagneto {  
    private CelluleTape first;  
    private CelluleTape last;  
}
```

- La syntaxe est associée à une propriété différente :
 - $first \neq null$ ssi $last \neq null$
 - et lorsque non null :
 - $first.previous = null$ ainsi que $last.next = null$
 - $first = last.previous.previous \dots$ (sans savoir combien de fois)
 - $last = first.next.next \dots$
- Mais MonMagneto n'est pas récursif, il est simplement "descendant" au sens où MonMagneto "domine" CelluleTape

Rappel sur les listes doublement chaînées

(et introduction indirecte aux arbres)

Fichier MonMagneto.java

```
public class MonMagneto {  
    private CelluleTape first;  
    private CelluleTape last;  
}
```

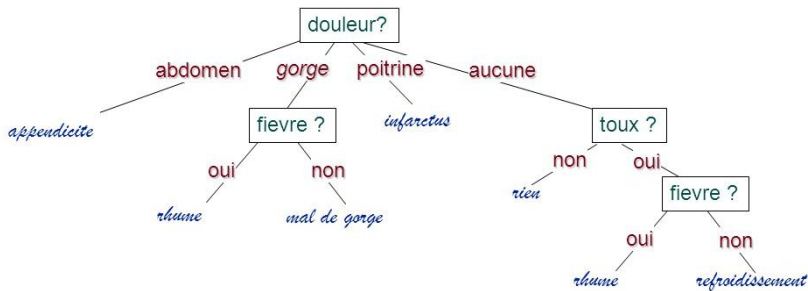
Fichier CelluleTape.java

```
public class CelluleTape{  
    private E content;  
    private CelluleTape next;  
    private CelluleTape previous;  
}
```

- On va étudier cette dimension verticale
- Eventuellement combiner dimension verticale + horizontale

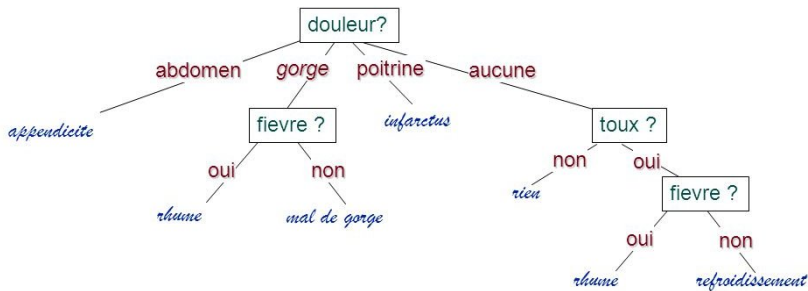
Généralités sur les arbres - Exemples concrets

Arbre de décision



Généralités sur les arbres - Exemples concrets

Arbre de décision



- Rq : pour prendre une décision on aurait pu écrire un programme ...

Sketch de programme

```
poserQuestion("douleur où ?")
selon (réponse) :
  cas "abdomen" : repondre appendicite
  cas "gorge" :
    poser question ("fièvre ?")
    selon (réponse) :
      cas oui : repondre rhume
      cas non : repondre mal de gorge
  fin selon
cas "poitrine" : repondre infractus
cas "aucune" :
  poser question ("toux?")
  selon (reponse) :
    cas non : répondre rien
    cas oui :
      poser question("fievre ?")
      selon (réponse) :
        cas oui : répondre rhume
        cas non : répondre refroidissement
```


Cela aurait été une approche maladroite

- Imaginez la taille de ce programme sur un cas réel ...
- Comment vont travailler le spécialiste et le programmeur ?
- Comment va-t-on prendre en compte les progrès de la médecine ?
- Une meilleure approche : séparer les données et le contrôle
 - Utiliser un arbre pour les données
 - Programmer le diagnostic en partant du haut de l'arbre initial :

Sketch du programme de diagnostic

```
si l'état est bien déterminé donner le nom de la maladie  
sinon
```

```
    poser une question qui ferait progresser davantage
```

```
    refaire diagnostic en partant du branchement correspondant à la réponse
```

```
fin sinon
```

- diagnostic va rappeler diagnostic (solution récursive)

comparez bien avec le programme précédent, et observez le rôle de l'arbre : une structure de données que l'on parcourt

Alphabet morse

Ces séquences de points (courts) et de tirets (longs) étaient ensuite transformées en lettres, et donc en messages.

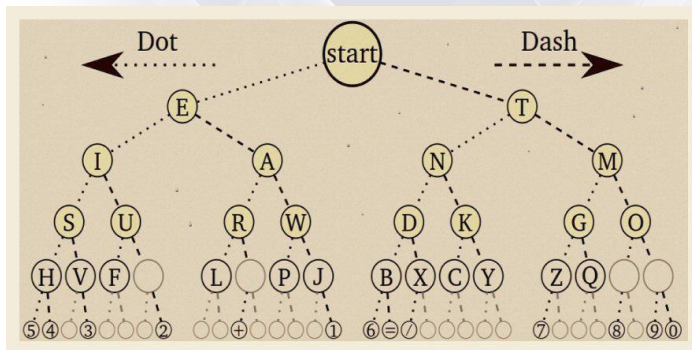


Autres exemples concrets

Alphabet morse

Pour décoder une chaîne de points et de tirets et en donner la lettre :

- avez vous le courage d'écrire un programme de la forme `switch/case` ?
- par contre, si l'arbre est une donnée, il suffit de parcourir la chaîne en même temps que l'arbre

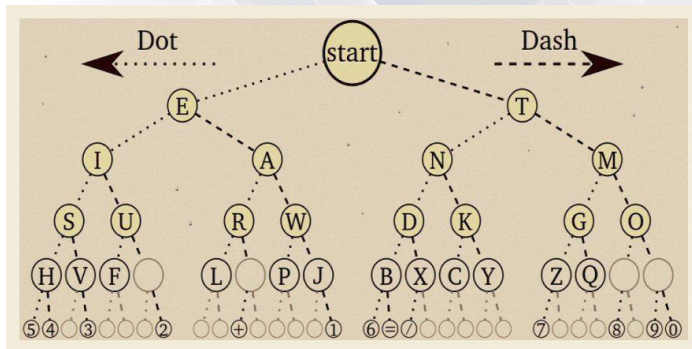


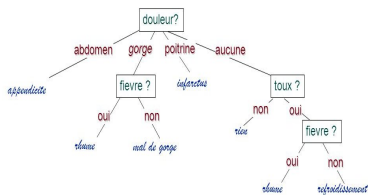
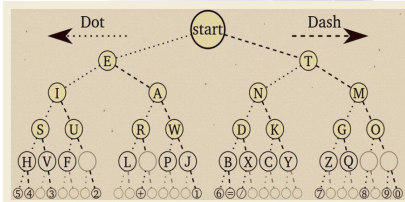
Autres exemples concrets

Alphabet morse etc. . .

L'arbre étant une donnée du programme, on peut lui en passer d'autres.

- Le même programme fonctionnera avec un alphabet différent
- On peut optimiser selon les fréquences de lettres par pays
- Raisonner ainsi conduit à la compression de texte en binaire (c'est comme cela que marche Zip)



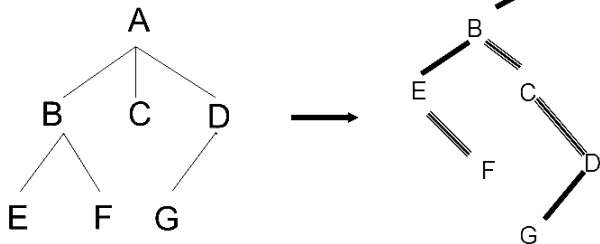


Points communs - Vocabulaire :

- **La racine** : le point d'entrée de l'arbre
- **Les noeuds** : toutes les racines des sous-arbres (arbre initial compris)
- **Les étiquettes** : la direction du branchement, les valeurs sur les noeuds
- **Père et fils** : les noeuds liés par un branchement (du haut vers le bas)
- **Les feuilles** : les noeuds sans fils
- **Un chemin** : une suite de noeuds, liés verticalement de père à fils
- **Le degré d'un noeud** : le nombre de branchement sortants possible

Modélisation - Importance du cas binaire

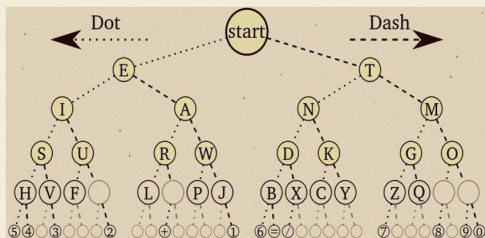
Principe : on peut se ramener à un arbre binaire avec des primitives fils aîné (fa —) et frère cadet (fc ==)



- Comprendre d'abord le cas de l'arbre binaire est donc fondamental
- On s'adaptera ensuite au cas n-aire

Modélisation

Cas binaire



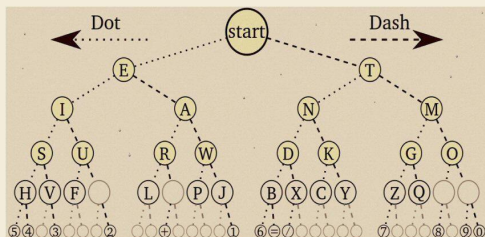
- Si tous les noeuds ont des étiquettes sortantes identiques (point et trait), on peut associer ce lien (entre noeuds pères et fils) à une variable propres aux noeuds. Ici un noeud père nomme ses deux fils :

Fichier NoeudMorse.java

```
public class NoeudMorse{  
    private char content;  
    private NoeudMorse fils_dot;  
    private NoeudMorse fils_dash;  
}
```

Modélisation

Cas binaire



- Si tous les noeuds ont des étiquettes sortantes identiques (point et trait), on peut associer ce lien (entre noeuds pères et fils) à une variable propres aux noeuds. Ici un noeud père nomme ses deux fils :

Fichier NoeudMorse.java

```
public class NoeudMorse{
    private char content;
    private NoeudMorse fils_dot;
    private NoeudMorse fils_dash;
}
```

Fichier CelluleTape.java

```
public class CelluleTape{
    private E content;
    private CelluleTape next;
    private CelluleTape previous;
}
```

et comparez attentivement ces deux syntaxes !!!

Les arbres binaires en général - Modèle à 3 classes

Fichier E.java

(Contenu sans importance)

Fichier Noeud.java

```
public class Noeud{  
    private E content;  
    private Noeud filsG;  
    private Noeud filsD;  
}
```

Fichier Arbre.java

```
public class Arbre{  
    private Noeud racine;  
    public Arbre(){  
        racine=null;  
    }  
    public boolean estVide(){  
        return racine==null;  
    }  
}
```

Modèle à 3 classes (construction)

Fichier Noeud.java

```
public class Noeud{
    private E content;
    private Noeud filsG;
    private Noeud filsD;
    public Noeud(E x, Noeud g, Noeud d){
        content=x;
        filsG=g;
        filsD=d;
    }
}
```

Fichier Arbre.java

```
public class Arbre{
    private Noeud racine;
    public Arbre(){ racine=null; }
    public boolean estVide(){ return racine==null;}
    public Arbre(E x, Arbre a1, Arbre a2){
        racine = new Noeud(x,a1.racine,a2.racine);
    }
}
```

Quelques algorithmes - contient

- Ecrivons une méthode d'arbre `contient` qui retourne si oui ou non un élément de type `E` est situé quelque part dans un noeud d'un arbre.

Fichier Noeud.java

```
public class Noeud{
    private E content;
    private Noeud filsG, filsD;
    ...
}
```

Fichier Arbre.java

```
public class Arbre{
    private Noeud racine;
    public boolean contient(E x){
        if (this.estVide()) return false; // cas facile
        return racine.contient(x); // sinon on délègue (comme on le faisait
            avec les listes)
    }
}
```

Fichier Noeud.java

```
public class Noeud{
    private E content;
    private Noeud filsG, filsD;
    public boolean contient(E x){
        if (content==x) return true;
        boolean testg=false, testd=false;
        if (filsG !=null) testg=filsG.contient(x);
        if (filsD !=null) testd=filsD.contient(x);
        return (testg || testd);
    } // difficile à écrire sans récursion,
} // qu'il faut maîtriser à présent !!
```

Fichier Arbre.java

```
public class Arbre{
    private Noeud racine;
    public boolean contient(E x){
        if (this.estVide()) return false;
        return racine.contient(x);
    }
}
```

Quelques algorithmes - Ordre de parcours

- A la différence des listes pour lesquelles l'ordre de parcours était évident car il était naturellement linéaire, pour les arbres il y a plusieurs façons de les décrire. Nous l'illustrons ici en proposant plusieurs "parcours" qui affichent le contenu d'un arbre.

Fichier Noeud.java

```
public class Noeud{  
    private E content;  
    private Noeud filsG, filsD;  
    ...  
}
```

Fichier Arbre.java

```
public class Arbre{  
    private Noeud racine;  
    public void affiche(){  
        if (this.estVide()) System.out.println("Arbre vide"); // cas simple  
        else racine.affiche(); // on délègue aux noeuds  
    }  
}
```

Quelques algorithmes - Ordre de parcours

Fichier Noeud.java

```
public class Noeud{
    private E content;
    private Noeud filsG, filsD;
    public void affiche(){
        System.out.print(content.toString()+" , "); // supposé exister pour E
        if (filsG!=null) filsG.affiche();
        if (filsD!=null) filsD.affiche();
    }
}
```

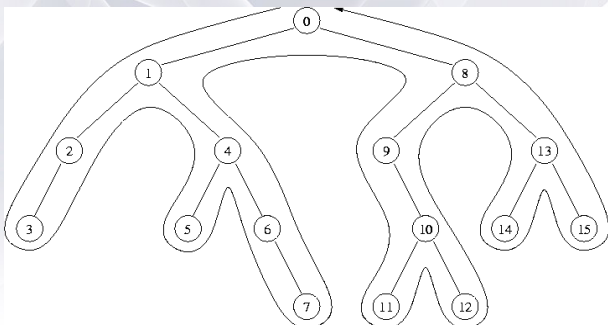
Fichier Arbre.java

```
public class Arbre{
    private Noeud racine;
    public void affiche(){
        if (this.estVide()) System.out.println("Arbre vide");
        else racine.affiche();
    }
}
```

Fichier Noeud.java

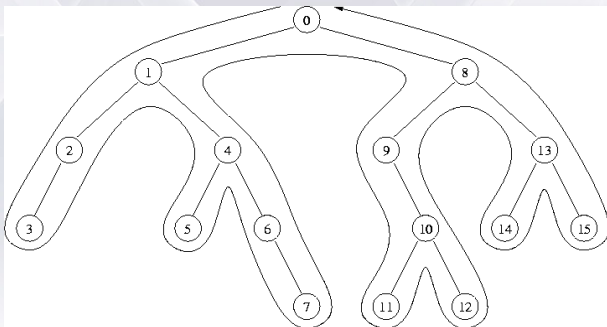
```
public class Noeud{  
    private E content;  
    private Noeud filsG, filsD;  
    public void affiche(){  
        System.out.print(content.toString()+" , "); // supposé exister  
        if (filsG!=null) filsG.affiche();  
        if (filsD!=null) filsD.affiche();  
    }  
}
```

- Le point de contrôle du programme colle à la structure de l'arbre



```
private E content;  
private Noeud filsG, filsD;  
public void affichePrefixe(){ // c'est le nom donné à ce parcours  
    System.out.print(content.toString()+" , " ) ; // supposé exister  
    if (filsG!=null) filsG.affichePrefixe();  
    if (filsD!=null) filsD.affichePrefixe();  
}
```

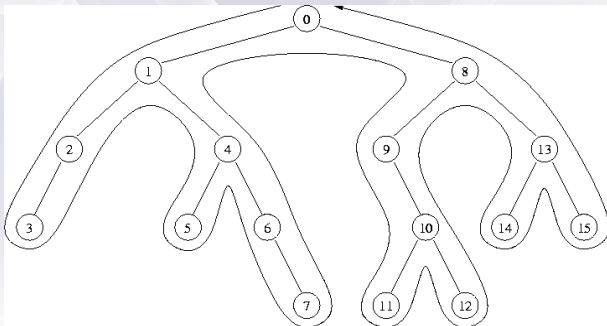
- Le point de contrôle du programme colle à la structure de l'arbre



0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

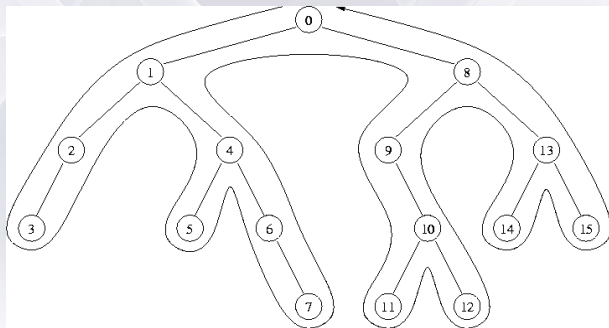

```
private E content;  
private Noeud filsG, filsD;  
public void afficheSuffixe(){ // on déplace le print à la fin  
    if (filsG!=null) filsG.afficheSuffixe();  
    if (filsD!=null) filsD.afficheSuffixe();  
    System.out.print(content.toString()+" , ") ; // supposé exister  
}
```

- Le point de contrôle du programme colle à la structure de l'arbre



```
private E content;  
private Noeud filsG, filsD;  
public void afficheSuffixe(){ // on déplace le print à la fin  
    if (filsG!=null) filsG.afficheSuffixe();  
    if (filsD!=null) filsD.afficheSuffixe();  
    System.out.print(content.toString()+" , " ) ; // supposé exister  
}
```

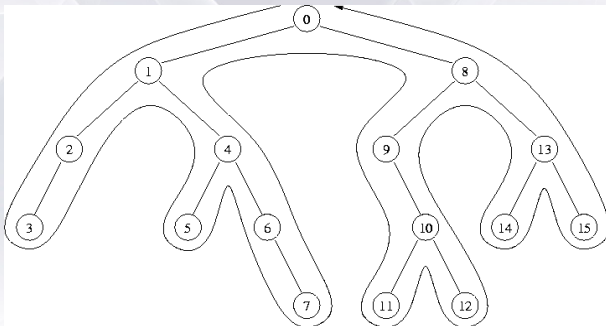
- Le point de contrôle du programme colle à la structure de l'arbre



3, 2, 5, 7, 6, 4, 1, 11, 12, 10, 9, 14, 15, 13, 8, 0

```
private E content;  
private Noeud filsG, filsD;  
public void afficheInfixe(){ // le print s'intercale avec les appels  
    if (filsG!=null) filsG.afficheInfixe();  
    System.out.print(content.toString()+" , ") ; // supposé exister  
    if (filsD!=null) filsD.afficheInfixe();  
}
```

- Le point de contrôle du programme colle à la structure de l'arbre

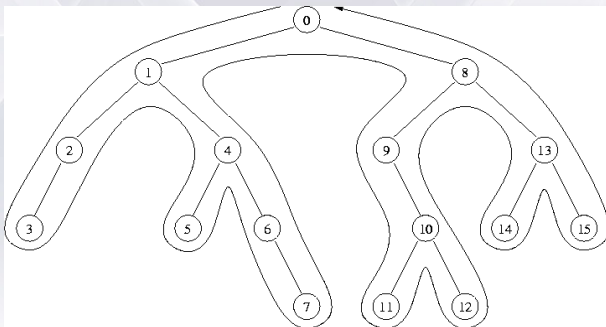


```

private E content;
private Noeud filsG, filsD;
public void afficheInfixe(){ // le print s'intercale avec les appels
    if (filsG!=null) filsG.afficheInfixe();
    System.out.print(content.toString()+" , " ) ; // supposé exister
    if (filsD!=null) filsD.afficheInfixe();
}

```

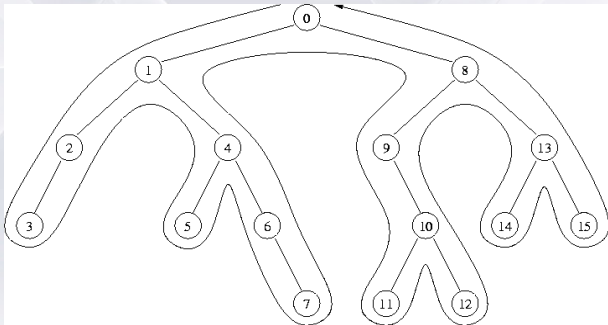
- Le point de contrôle du programme colle à la structure de l'arbre



3, 2, 1, 5, 4, 6, 7, 0, 9, 11, 10, 12, 8, 14, 13, 15

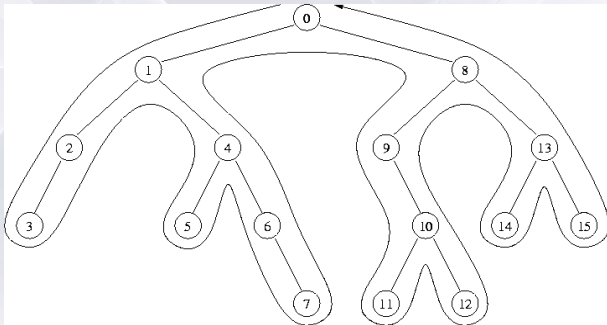
Exercice

```
private E content;  
private Noeud filsG, filsD;  
public void afficheBizarre(){ // appelé uniquement sur notre exemple  
    System.out.print(content.toString()+" ");  
    filsG.afficheInfixe();  
    filsD.afficheSuffixe();  
}
```



Exercice

```
private E content;  
private Noeud filsG, filsD;  
public void afficheBizarre(){ // appelé uniquement sur notre exemple  
    System.out.print(content.toString()+" ");  
    filsG.afficheInfixe();  
    filsD.afficheSuffixe();  
}
```



0, 3, 2, 1, 5, 4, 6, 7, 11, 12, 10, 9, 14, 15, 13, 8

- Il vous faut connaître ces différences
- « *le point de contrôle colle à la structure de l'arbre* » signifie aussi qu'on utilise la pile des appels pour stocker les noeuds dont l'exploration du sous arbre n'est pas terminé

Quelques algorithmes - Echange de noeud

- Ecrire **miroir** qui intervertit les fils gauche et droit de chaque noeud de l'arbre dont **this** est racine

Fichier Noeud.java

```
public class Noeud{
    private E content;
    private Noeud filsG, filsD;
    public miroir(){
        Noeud tmp;
        tmp=filsG;    // -----
        filsG=filsD; // échange local
        filsD=tmp;    // -----
        ...
    }
}
```


Quelques algorithmes - Echange de noeud

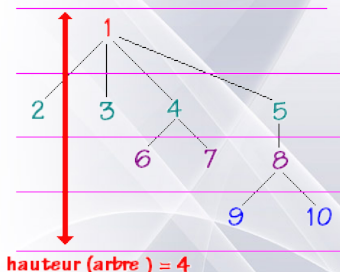
- Ecrire **miroir** qui intervertit les fils gauche et droit de chaque noeuds dont **this** est racine

Fichier Noeud.java

```
public class Noeud {  
    private E content;  
    private Noeud filsG, filsD;  
    public miroir(){  
        Noeud tmp;  
        tmp=filsG;    // -----  
        filsG=filsD; // échange local  
        filsD=tmp;    // -----  
        if (filsG!=null) filsG.miroir(); // continuation réursive  
        if (filsD!=null) filsD.miroir();  
    }  
}
```

Quelques algorithmes - hauteur

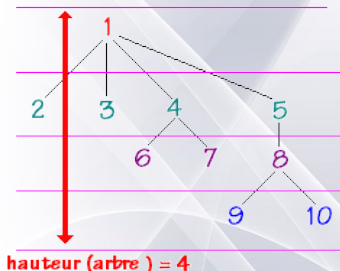
- Hauteur : taille du plus long chemin de la racine à une feuille



(illustré ici sur un arbre n-aire)

Quelques algorithmes - hauteur

- Hauteur : longueur du plus long chemin de la racine à une feuille



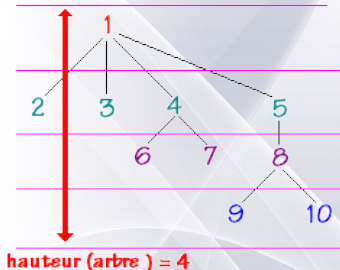
(illustré ici sur un arbre n-aire)

Fichier Arbre.java

```
public class Arbre {  
    public int hauteur(){  
        if (estVide()) return 0;  
        else return racine.hauteur();  
    }  
}
```

Quelques algorithmes - hauteur

- Hauteur : longueur du plus long chemin de la racine à une feuille



(illustré ici sur un arbre n-aire)

Fichier Arbre.java

```
public class Arbre {  
    public int hauteur(){  
        if (estVide()) return 0;  
        else return racine.hauteur;  
    }  
}
```

Fichier Noeud.java

```
public int hauteur(){  
    if (estFeuille()) return 1;  
    int hg=0;hd=0;  
    if (filsG!=null) hg=filsG.hauteur();  
    if (filsD!=null) hd=filsD.hauteur();  
    return 1+Math.max(hg,hd);  
}
```

Fichier Noeud.java

```
public boolean estFeuille(){  
    return ( (filsG==null) &&  
            (filsD==null) );  
}
```

(résolu ici sur un arbre binaire)

Quelques algorithmes - Compter les feuilles

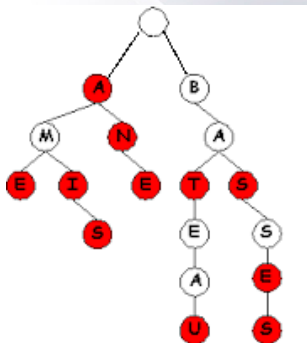
Fichier Arbre.java

```
public class Arbre{
    private Noeud racine;
    public int nbFeuilles(){
        if (this.estVide()) return 0;
        else return racine.nbFeuilles();
    }
}
```

Fichier Noeud.java

```
public class Noeud{
    private E content;
    private Noeud filsG, filsD;
    public int nbFeuilles(){
        if (estFeuille()) return 1;
        int nbG=0, nbD=0;
        if (filsG!=null) nbG=filsG.nbFeuilles();
        if (filsD!=null) nbD=filsD.nbFeuilles();
        return nbG+nbD;
    }
}
```

Pour aller plus loin - Concision des arbres



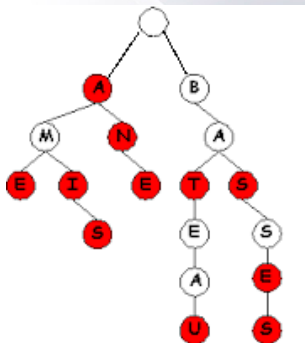
Explication de ce dessin :

- Un noeud contient une lettre et une couleur
- Les chemins représentent des mots
- Ils sont valides s'ils terminent sur rouge

Question :

- Combien de mots sont représentés ?

Pour aller plus loin - Concision des arbres



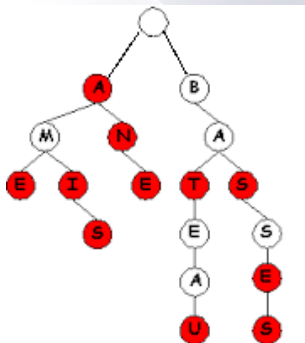
Explication de ce dessin :

- Un noeud contient une lettre et une couleur
- Les chemins représentent des mots
- Ils sont valides s'ils terminent sur rouge

Question :

- Combien de mots sont représentés ?
 - ici : 11 (autant que de noeuds rouges)
- Combien de lettres pour les écrire tous ?

Pour aller plus loin - Concision des arbres



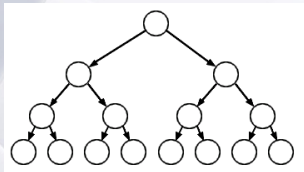
Explication de ce dessin :

- Un noeud contient une lettre et une couleur
- Les chemins représentent des mots
- Ils sont valides s'ils terminent sur rouge

Question :

- Combien de mots sont représentés ?
 - ici : 11 (autant que de noeuds rouges)
- Combien de lettres pour les écrire tous ?
- *a, an, ane, ame, ami, amis, bat, bateau, bas, basse, basses* : soit 39 lettres
- L'arbre a 18 noeuds : économie des préfixes

Pour aller plus loin - Concision des arbres (2)

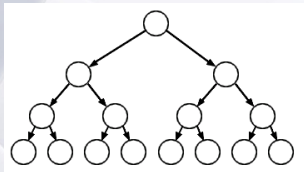


- On a représenté ici un arbre binaire complet de hauteur 4
- Il contient 15 noeuds
- 1 chemins de longueur 1 noeud
- 2 chemins de longueur 2 noeuds
- 4 chemins de longueur 3 noeuds
- 8 chemins de longueur 4 noeuds
- Potentiellement 15 mots d'un total de 49 lettres

Pour aller plus loin - Concision des arbres (3)

- Si l'arbre binaire complet était de hauteur 5
- Il contiendrait $15 + 16 = 31$ noeuds
- 1 chemins de longueur 1 noeud
- 2 chemins de longueur 2 noeuds
- 4 chemins de longueur 3 noeuds
- 8 chemins de longueur 4 noeuds
- 16 chemins de longueur 5 noeuds
- Potentiellement 31 mots d'un total de $49 + 16 * 5 = 129$ lettres

Pour aller plus loin - Concision des arbres (4)



- La comparaison exponentiellement plus grande entre :
 - la hauteur
 - et le nombre de noeud de l'arbre (ou de feuilles)
- Cette différence d'ordre de grandeur sera très utilisée l'an prochain

Ordre de parcours - Version itérative vs Version récursive ...

- Voyons où cela nous mènera si on voulait réfléchir à une version itérative de ce programme de parcours vu précédemment ...

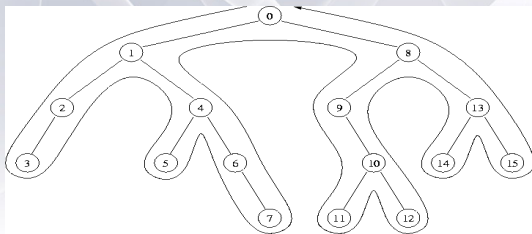
Fichier Noeud.java

```
private E content;  
private Noeud filsG, filsD;  
public void affichePrefixe(){  
    System.out.print(content.toString()+" , ") ; // supposé exister  
    if (filsG!=null) filsG.affichePrefixe();  
    if (filsD!=null) filsD.affichePrefixe();  
}
```

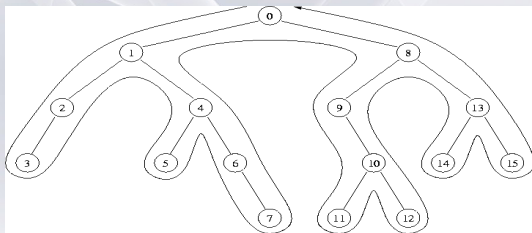
Fichier Noeud.java

```
private E content;  
private Noeud filsG, filsD;  
public void affichePrefixe(){  
    System.out.print(content.toString()+" , ") ; // supposé exister  
    if (filsG!=null) filsG.affichePrefixe();  
    if (filsD!=null) filsD.affichePrefixe();  
}
```

- nous avons remarqué que « *le point de contrôle colle à la structure de l'arbre* » c.à d. qu'on utilise la pile des appels pour stocker les noeuds dont l'exploration du sous arbre n'est pas terminées



```
public void parcoursPrefixeIteratif (){  
    MaPile p = new MaPile(); // construite pour des Noeuds  
    Noeud tmp ;  
    p.push (this);  
    while (! p.isEmpty() ){  
        tmp = p.pop();  
        System.out.print( tmp.content.toString() ); // en premier  
        if ( tmp.filsD != null ) p.push( tmp.filsD ); // en second  
        if ( tmp.filsG != null ) p.push( tmp.filsG ); // en troisième  
    }  
}
```



- Qui donne ?
- Et si on inverse l'ordre des instructions ?

Fichier Noeud.java

```
public void parcoursPrefixeIteratif (){  
    MaPile p = new MaPile(); // construite pour des Noeuds  
    Noeud tmp ;  
    p.push (this);  
    while (! p.isEmpty() ){  
        tmp = p.pop();  
        System.out.print( tmp.content.toString() ); // en premier  
        if ( tmp.filsD != null ) p.push( tmp.filsD ); // en second  
        if ( tmp.filsG != null ) p.push( tmp.filsG ); // en troisième  
    }  
}
```

- **MaPile** est associée à un ordre, liée aux opérations : **push, pop**
 - dernier entré premier sorti

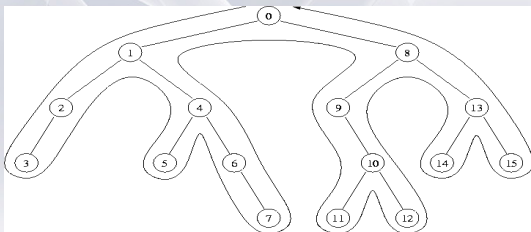
Fichier Noeud.java

```
public void parcoursPrefixeIteratif (){
    MaPile p = new MaPile(); // construite pour des Noeuds
    Noeud tmp ;
    p.push (this);
    while (! p.isEmpty() ){
        tmp = p.pop();
        System.out.print( tmp.content.toString() ); // en premier
        if ( tmp.filsD != null ) p.push( tmp.filsD ); // en second
        if ( tmp.filsG != null ) p.push( tmp.filsG ); // en troisième
    }
}
```

- **MaPile** est associée à un ordre, liée aux opérations : **push, pop**
 - dernier entré premier sorti
- On peut « *bousculer* » un peu notre code, et utiliser : **MaFile**
 - premier entré premier sorti

Fichier Noeud.java

```
public void parcoursIteratifFile1 () {  
    MaFile p = new MaFile(); // construite pour des Noeuds  
    Noeud tmp ;  
    p.add (this);  
    while (! p.isEmpty() ){  
        tmp = p.get();  
        System.out.print( tmp.content.toString() ); // en premier  
        if ( tmp.filsD != null ) p.add( tmp.filsD ); // en second  
        if ( tmp.filsG != null ) p.add( tmp.filsG ); // en troisième  
    }  
}
```

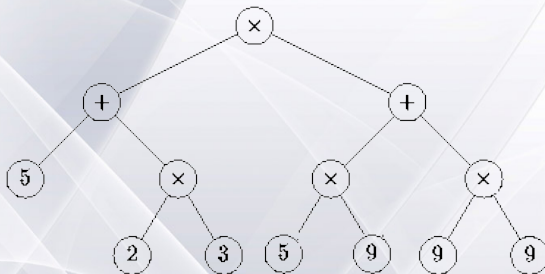


- Qui donne ?

Le parcours que nous venons de voir s'appelle le **parcours en largeur**.

Dans la suite, nous présentons des petits exercices auxquels on va réfléchir encore un peu ensemble.

Exercice - (non corrigé)



- Les feuilles d'un arbre portent un chiffre
- Les autres noeuds portent une opération '+' ou '*'
- Calculez la valeur de l'expression portée par un noeud

Exercice - Ajout orienté

- Ecrivez une méthode **boolean ajoutOrienté(E x,String dir)**
- **dir** est un mot composé des lettres 'g' et 'd'
- pour que l'ajout fonctionne :
 - **dir** privé de sa dernière lettre doit référencer un noeud existant
 - la direction finale, partant de ce noeud doit être inoccupée
- Rappel : la classe `String` possède les méthodes
 - `String substring(int beginIndex)`
 - `char charAt(int index)`

Exercice - Ajout orienté

- Ecrivez une méthode **boolean ajoutOrienté(E x,String dir)**
- **dir** est un mot composé des lettres 'g' et 'd'
- pour que l'ajout fonctionne :
 - **dir** privé de sa dernière lettre doit référencer un noeud existant
 - la direction finale, partant de ce noeud doit être inoccupée
- Rappel : la classe String possède les méthodes
 - String substring(int beginIndex)
 - char charAt(int index)

Fichier Arbre.java

```
public boolean ajoutOrienté (E x, String dir){  
    if (dir==null || dir.length()==0 || racine==null) return false;  
    return racine.ajoutOrienté(x,dir);  
}
```

Exercice - Ajout orienté

Fichier Noeud.java

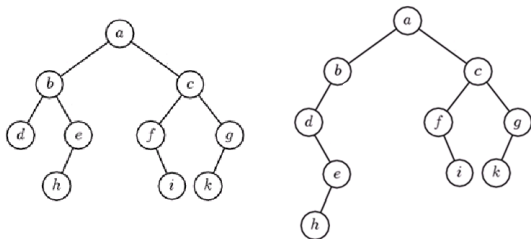
```
public boolean ajoutOriente (E x, String dir){
    char d=dir.charAt(0);
    if ( (dir.length()==1) && (d=='g') ) {
        if (filsG != null) return false;
        else { filsG=new Noeud(x, null, null); return true; }
    }
    if ( (dir.length()==1) && (d=='d') ) {
        if (filsD != null) return false;
        else { filsD=new Noeud(x, null, null); return true; }
    }
    dir=dir.substring(1);
    if ( (d=='g') && (filsG != null) ) return filsG.ajoutOriente(x,dir);
    if ( (d=='d') && (filsD != null) ) return filsD.ajoutOriente(x,dir);
    // cas restant : ni 'g' ni 'd', ou noeud intermediaire inexistant
    return false;
}
```

Exercice

- Dessinez deux arbres dont le parcours préfixe est : **a,b,d,e,h,c,f,i,g,k**

Exercice

- Dessinez deux arbres dont le parcours préfixe est : **a,b,d,e,h,c,f,i,g,k**



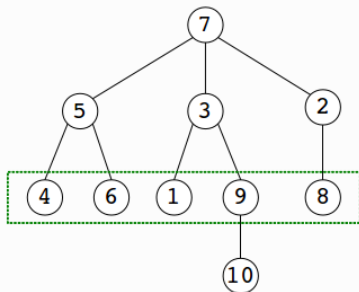
Exercice

- Si on connaît le résultat d'un parcours en ordre préfixe et celui d'un parcours en ordre suffixe pour un même arbre, peut-on le dessiner ?

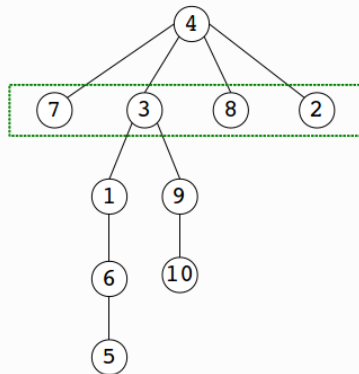
Exercice

- Si on connaît le résultat d'un parcours en ordre préfixe et celui d'un parcours en ordre suffixe pour un même arbre, peut-on le dessiner ?
- ... non ... avec 'abc' en préfixe, et 'cba' en suffixe on peut trouver plusieurs arbres

Exercice - (Difficile ...)



La largeur vaut 5.



La largeur vaut 4.

rq : sur le dessin les arbres représentés sont n-aire, mais la même définition s'applique évidemment aux arbres binaires

- Ecrivez un algorithme (et ses outils intermédiaires) pour calculer la largeur d'un arbre binaire