# Algo à gogo

McRoss "Beat" DeKoualitat *feat.* FireGrain

4 octobre 2014

# Chapitre 1

# Récursivité et thé citronné

"LOURD MADDIE"

je sais pas qui

## 1.1 Leçon

### 1.1.1 La notion de récursivité

Que signifie *récursivité* ?. Cela signifie que la fonction va *s'appeler elle-même*, mais avec *différents paramètres*.
Prenons par exemple la fonction factorielle. Imaginons que l'on cherche à calculer factorielle(5), qui correspond à 5*4*3*2*1. On a deux méthodes différentes :

```
/* on fait ici de manière itérative */
int factorielle(n)
{
        int i,j;
        j=1;
        for(i=1;i<=n;i++)
                j=j*i;
        return j;
}
```

On opère maintenant de manière récursive :

```
int factorielleRec(n)
{
        if (n==1)
                return(1);
        else
                return(n * (factorielleRec(n-1)) );
}
```

Comme vous le voyez, factorielleRec( ) s'appelle elle-même. Il faut noter le cas d'arrêt (quand on arrive à n=1), car on ne veut pas non plus que la fonction s'appelle indéfiniment.

### 1.1.2 Les listes Récursive

Ici, une liste récursive est composé de deux choses :

— Une valeur (un entier)
— D'une autre liste récursive

En java, cela donnerait ça :

```
public ListeRecursive{
        int tete;
        ListeRecursive liste;
}
```

Une liste est donc récursive puisque *elle contient une autre liste*. On peut comparer à des dossiers et des fichiers, un dossier possède des fichiers et des dossiers. Ici on a qu'un seul fichier et qu'un seul dossier.

M.TOUTLEMONDE: Oulala, mais pourquoi se compliquer la vie avec ça ?

FIREGRAIN: eh bien, l'interêt est l'allocation de la mémoire, avec cette liste on peut mettre autant de valeur que l'on veut car les listes sont répartis sur plusieur cases mémoire pointé en somme par la liste précédente.

### 1.1.3 Comment fait-on de la récursivité ? ? ? ? ? ? ?

Il faut appliquer 3 étapes :

— Trouver le traitement à faire sur l'élément actuel (trouver les cas particuliers, en général, ce sont les objectif de la fonction)
— L'appliquer sur tous les autres éléments (appel récursif)
— Arrêter le traitement (trouver le cas d'arrêt)

En reprenant l'exemple de la factorielle, on a bien trouvé le traitement (élément actuel * le reste), fait l'appel récursif (on fait élément actuel * factorielle(reste)), et arrêté le traitement (on vérifie si l'élément actuel est égal à 1).

## 1.2 Exercices

je veux trouver le nombre de multiple de 5 dans une liste récursif.

— Cas particulier / Objectif : l'élément actuel est un multiple de 5. Donc je comptabilise 1 et je rajoute ce que la fonction va comptabiliser sur le reste. Je délégue le travail au reste (aux putain de random que je ne connais pas encore) donc j'utilise la fonction sur la liste suivante qui elle même se chargera de délégué le travail aux autres et ainsi de suite jusqu'aux cas d'arrêt.
— Cas d'arrêt : tout simplement, si tombe sur une liste vide, bah j'arrête et je renvoie 0. (il s'agit du cas le plus courant)
— Cas général : Dans ce cas, le cas général est similaire au cas particulier, si l'élément actuel n'est pas un multiple de 5 et bien je regarde ce que ça donne sur le reste. Donc j'applique la fonction à la liste suivante.

```
public int nbMulitple5(LR liste)
{
    // CAS D'ARRET
    if(liste.vide()) return 0;

    // OBJECTIF
    if(liste.tete() \% 5 ==0 ) return 1 + nbMultiple(liste.reste());

    // CAS GENERAL
    else return nbMultiple(liste.reste());

}
```

# Chapitre 2

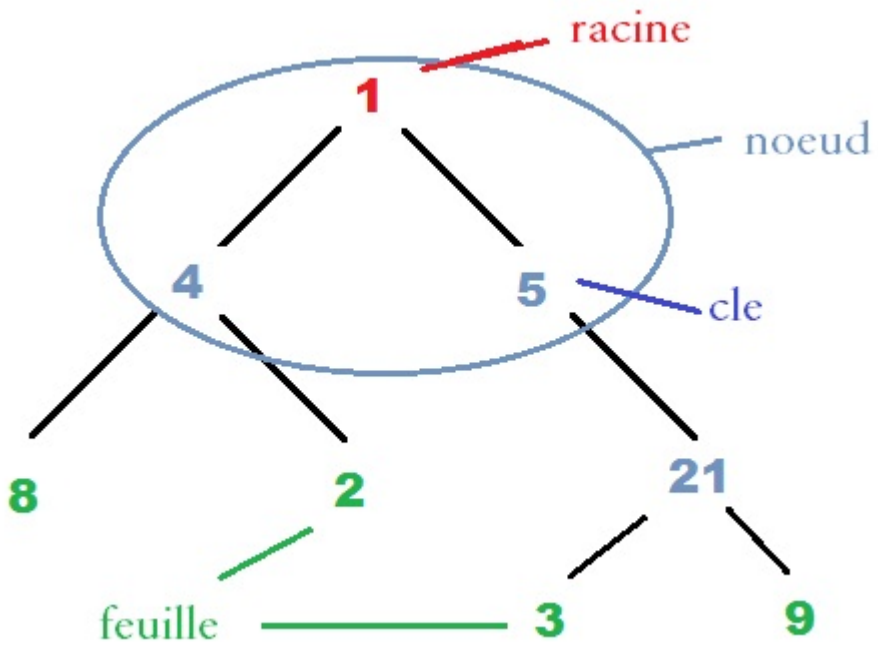
# Arbres

"moi j t'envoie yichen"

Mr.Maths menace #boloss aka l'évryien vol.2

## 2.1 Les arbres binaires

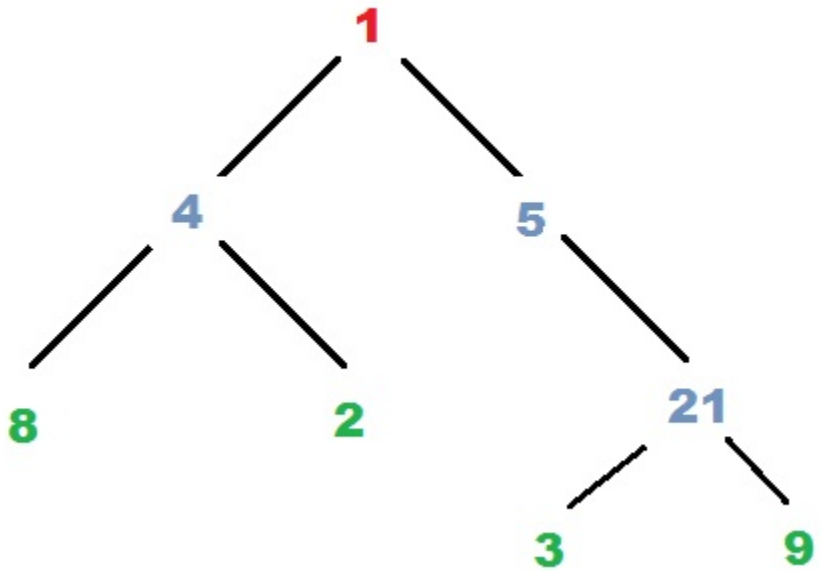
Tout d'abord, il faut savoir ce qu'est un arbre binaire. Il s'agit d'un arbre enraciné ne possédant que deux fils. Il existe tout un lexique pour définir un arbre, mais vu qu'une image vaut mieux que 1000 mots, voici un schéma.



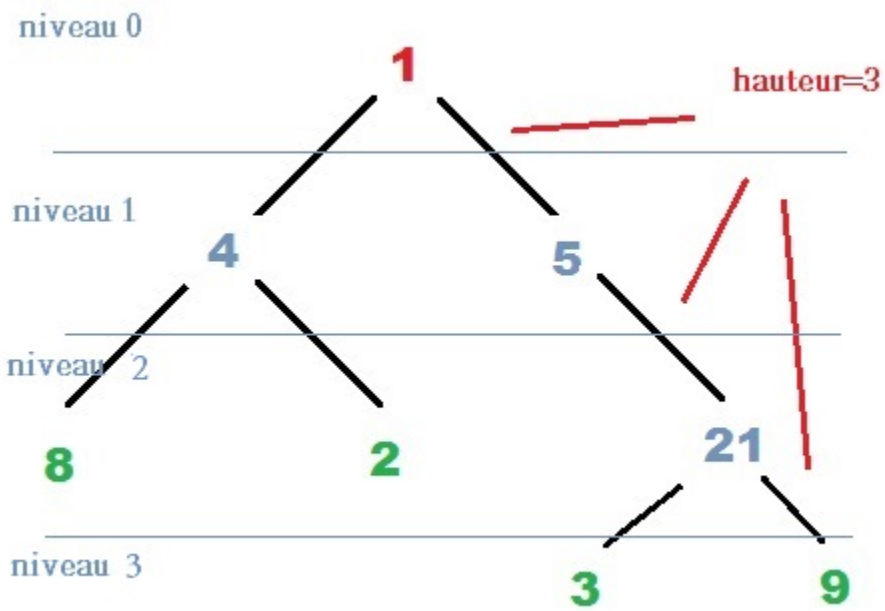

Un **noeud**, à ne pas confondre avec une clé (sans vouloir désigner personne), est un sous arbre binaire. l'arbre que vous voyez ici possède plusieur arbre binaire qu'on appelle noeud. Alors qu'une **clé** est la valeur que possède un arbre binaire, il s'obtient grâce à la fonction info() de la classe ArbreBinaire. La **taille** d'un

arbre correspond au nombre de noeuds moins le nombre de feuilles.
La **hauteur** d'un arbre correspond à la *profondeur maximale*. La hauteur s'applique à un arbre, la profondeur à un noeud. On peut le déduire en comptant le nombre de branche maximal.

Par exemple, l'arbre exemple possède une taille de **4** (8-4 = 4 feuilles) :



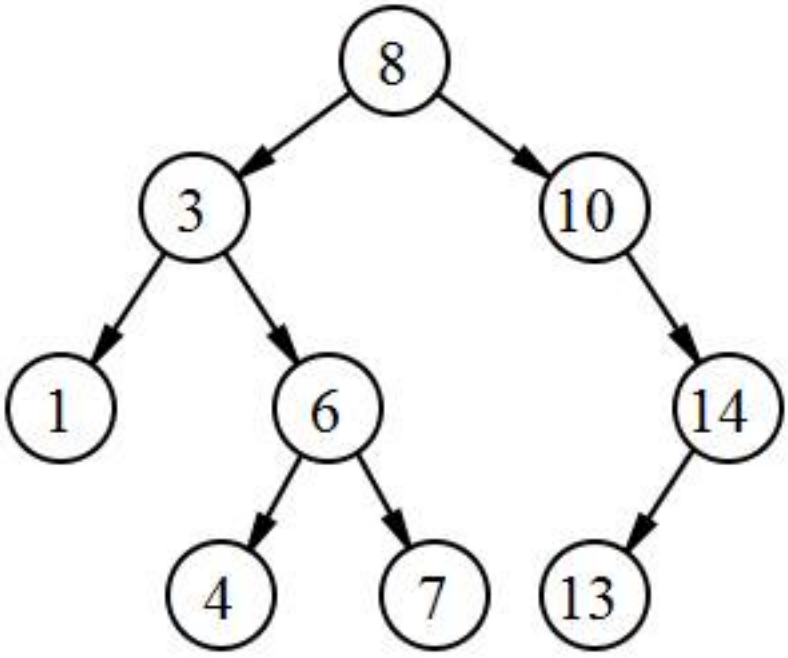

Et une hauteur de **3** :




ET C TOU.

## 2.2 Les arbres binaires de recherche

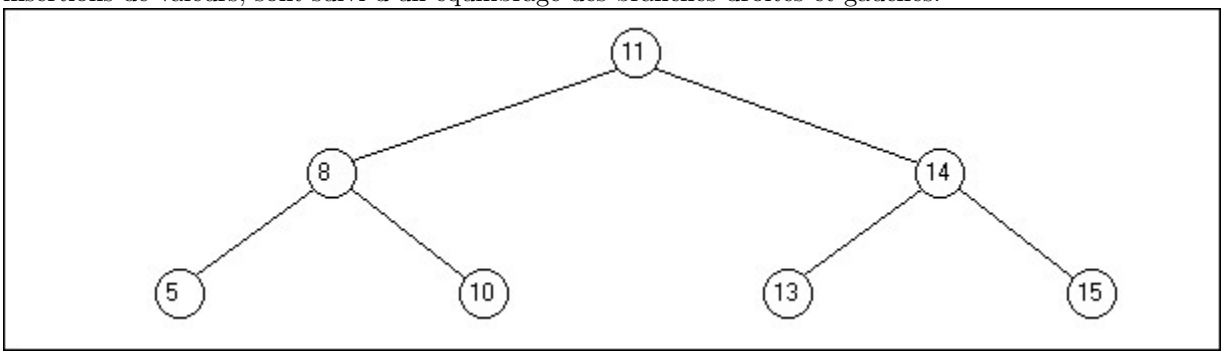
Un arbre binaire de recherche, c'est tout con : c'est un arbre dans lequel le fils gauche est *inférieur* à la valeur du noeud alors que le fils droit est *supérieur*.
Voici un exemple :




On voit ici que tous les membres à gauche sont inférieur à 8, alors que tous les membres de droite sont supérieur à 8.

## 2.3 Les arbres AVL

les arbres AVL, sont des arbres binaires de recherches qui sont automatique équilibrés. Cela implique les insertions de valeurs, sont suivi d'un équilibrage des branches droites et gauches.




ceci est un arbre parfaitement équilibré car il possède autant de branches droites que de branches gauches
un Arbre binaire de recherche est un AVL, si et seulement si en chaque somment, l'équilibre (hauteur(filsGauche)-hauteur(filsDroit)) vaut 1,0 ou -1. NI PLUS NI MOINS BANDE DE...