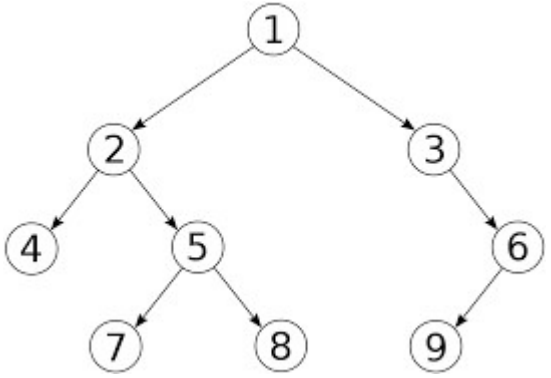
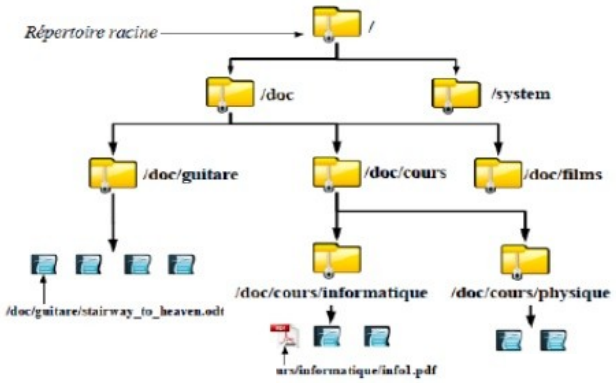
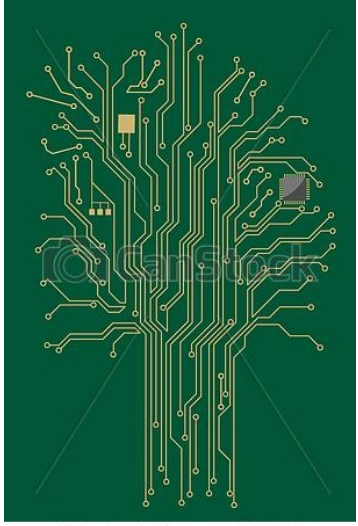
Les structures de données
Les arbres (binaires)

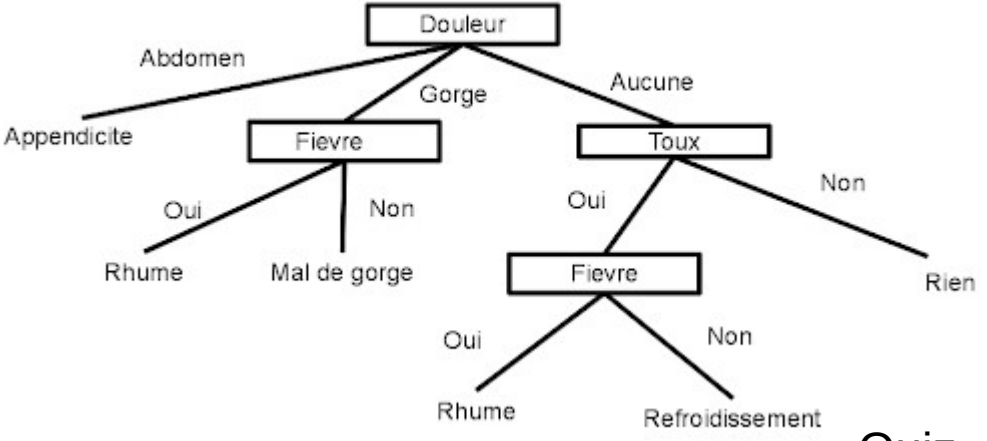
Arbres

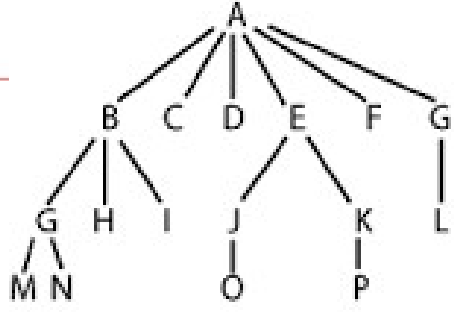


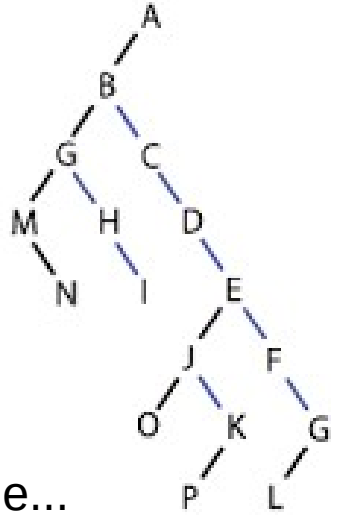
Exemples d'arbres





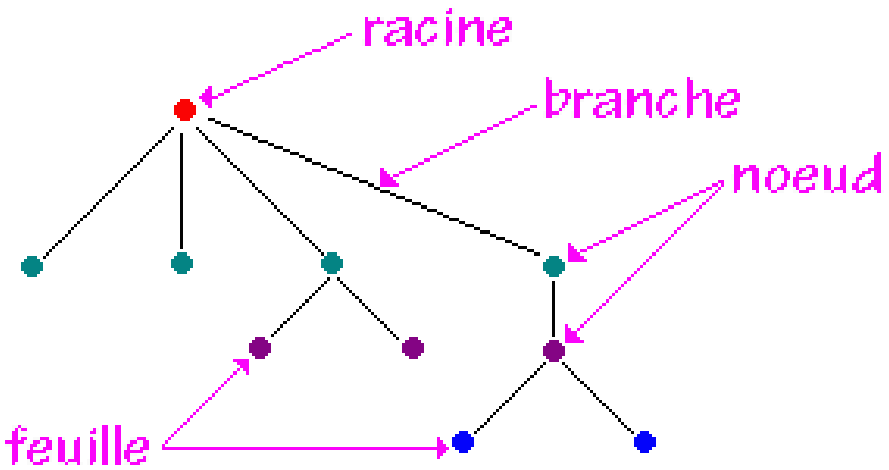


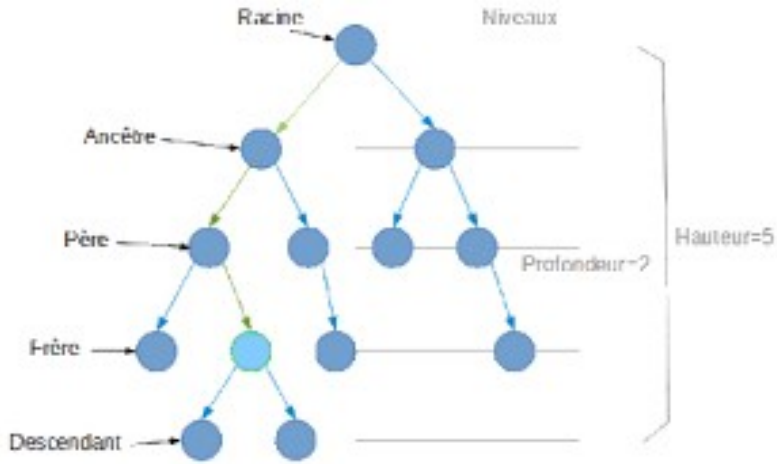


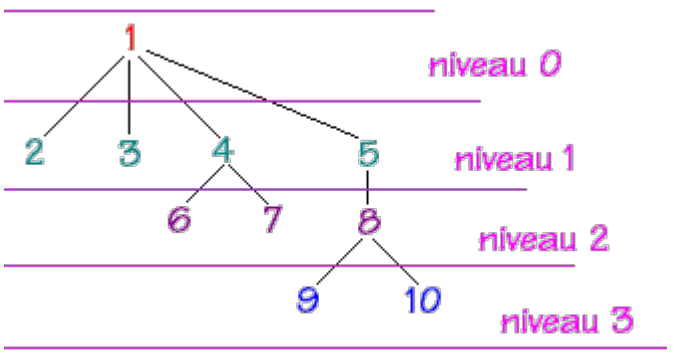


Quiz=(1+2)*(4+5) est un arbre. Dessinez le...

Vocabulaire





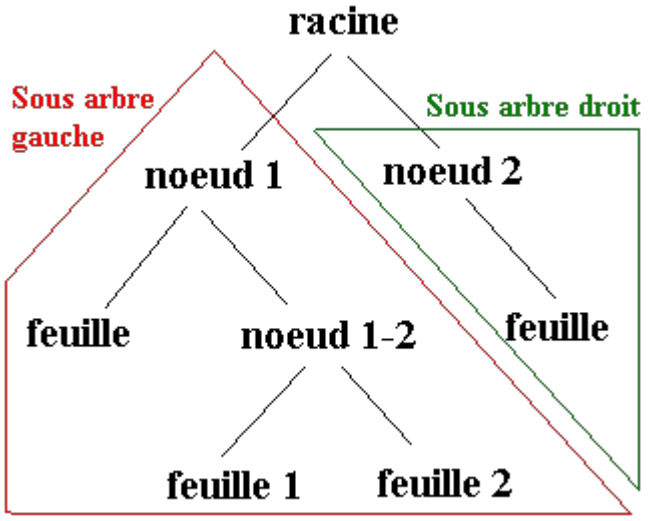


Quiz=Si a chaque nœud, j'ai 2 descendants (on parle des fois de fils/enfants) alors combien y a t'il de nœuds à un niveau n ? Et en tout ? Et le nombre de branches ?

Arbres binaires

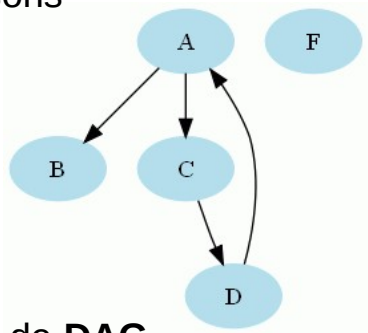
Des arbres, où chaque nœud a au plus 2 descendants.

Notez qu'un arbre vide est un arbre et donc si un descendant est un arbre vide (branche) alors le sous arbre est vide...

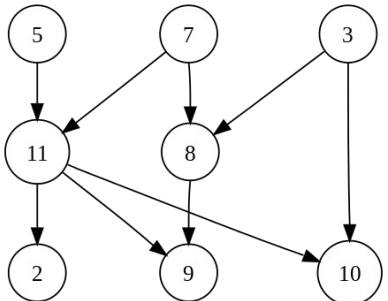


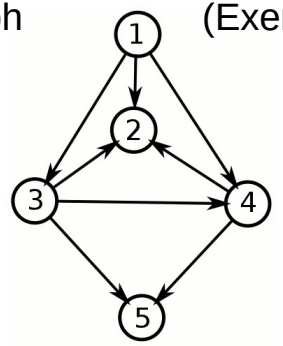
Et en Java ?

- Comme pour les listes, sauf qu'il y aura 2 « pointeurs », filsGauche et filsDroit (pour faire plus simple que DescendantG et D) ; 2 choix
  - Utiliser une classe intermédiaire « cellule » où l'arbre vide sera donc un arbre avec une cellule vide, c'est-à-dire, **null**
  - Écriture directe (une seule et même classe) où un arbre vide sera représenté directement la valeur **null ;** Un arbre réduit à une feuille, de contenu x, sera créé par « new Arbre(null, x, null) ». Nous choisissons cette solution pour changer de manière de faire
- Attention :
  - **Les descendants ne sont pas eux même des ascendants, on parlerais de cycles** ====>

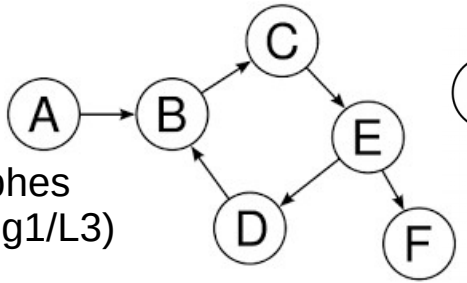


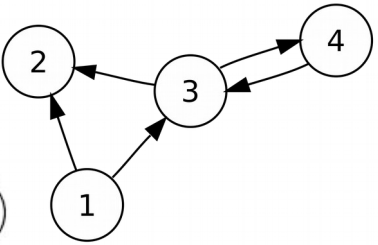
  - **Un nœud n'a qu'un unique père/ascendant** (sauf la racine qui n'a pas de père) ; Sinon, on parle de **DAG** pour Directed Acyclic Graph (Exemple « livre dont vous êtes le héros »)





A droite, des graphes orientés (cours Ing1/L3)



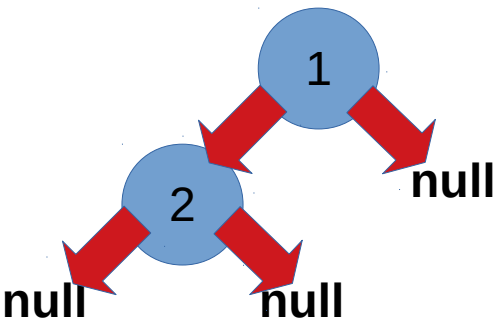


Implantation

```
class Arbre  {
  int contenu;
  Arbre filsG, filsD;
  Arbre(Arbre g, int c, Arbre d)  {
    filsG = g;
    contenu = c;
    filsD = d;
  }
}
```

Utilisation

Arbre t = new Arbre(new Arbre(null,2,null),1,null)



Ok et quelles méthodes ?

- La liste peut être très longue… Quelques unes :
  - Nombre de nœuds
  - Nombre de feuilles
  - Sélection d’un sous-arbre pour un nœud donné
  - Parcours d’arbres
    - En profondeur Suffixe(filsG, filsD, noeud) ou Prefixe(nœud, filsG, filsD) ou Infixe(filsG,noeud,filsD)
    - En largeur
  - Arbres binaires de recherche

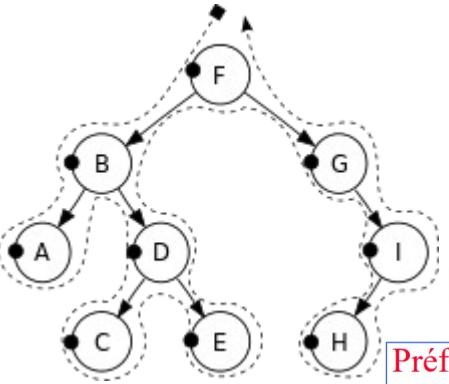
Méthodes statiques

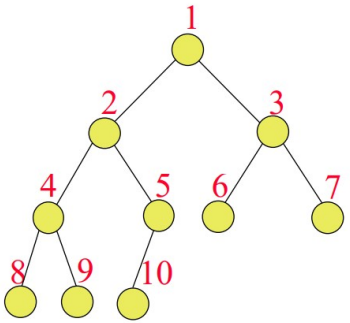
On peut avoir des primitives aux arbres (comme pour les piles/files) mais pour varier, définissons des méthodes statiques (i.e. en dehors de la classe ou indépendantes). Exemples :

```
static int taille(Arbre a) {
  if (a == null) return 0;
  return 1+taille(a.filsG)+taille(a.filsD);
}
static boolean trouver (int x, Arbre a) {
  if (a == null) return false ;
  if (x == a.contenu) return true;
  return (trouver(x, a.filsG) || trouver(x, a.filsD)) ;
}
```

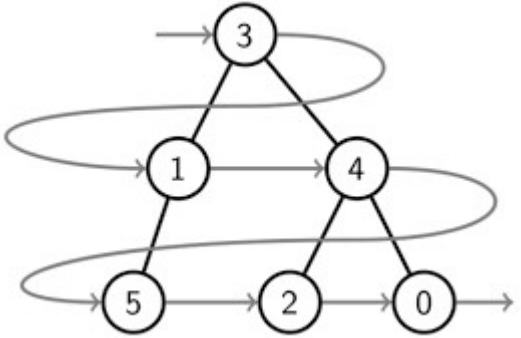
Des parcours : profondeur/largeur

- Parcourir un arbre permet de l'afficher mais surtout d'utiliser les valeurs qui ont pu être intégrées dans un ordre particulier (voir ABR) ; Pleins d'applications (IA)
- **profondeur**(deep-first=DF Search, souvent récursif) **Largeur**(Breadth First, BFS, plutôt itératif)





Préfixe : 1, 2, 4, 8, 9, 5, 10, 3, 6, 7
(VGD)
Infixe : 8, 4, 9, 2, 10, 5, 1, 6, 3, 7
(GVD)
Suffixe : 8, 9, 4, 10, 5, 2, 6, 7, 3, 1
(GDV)



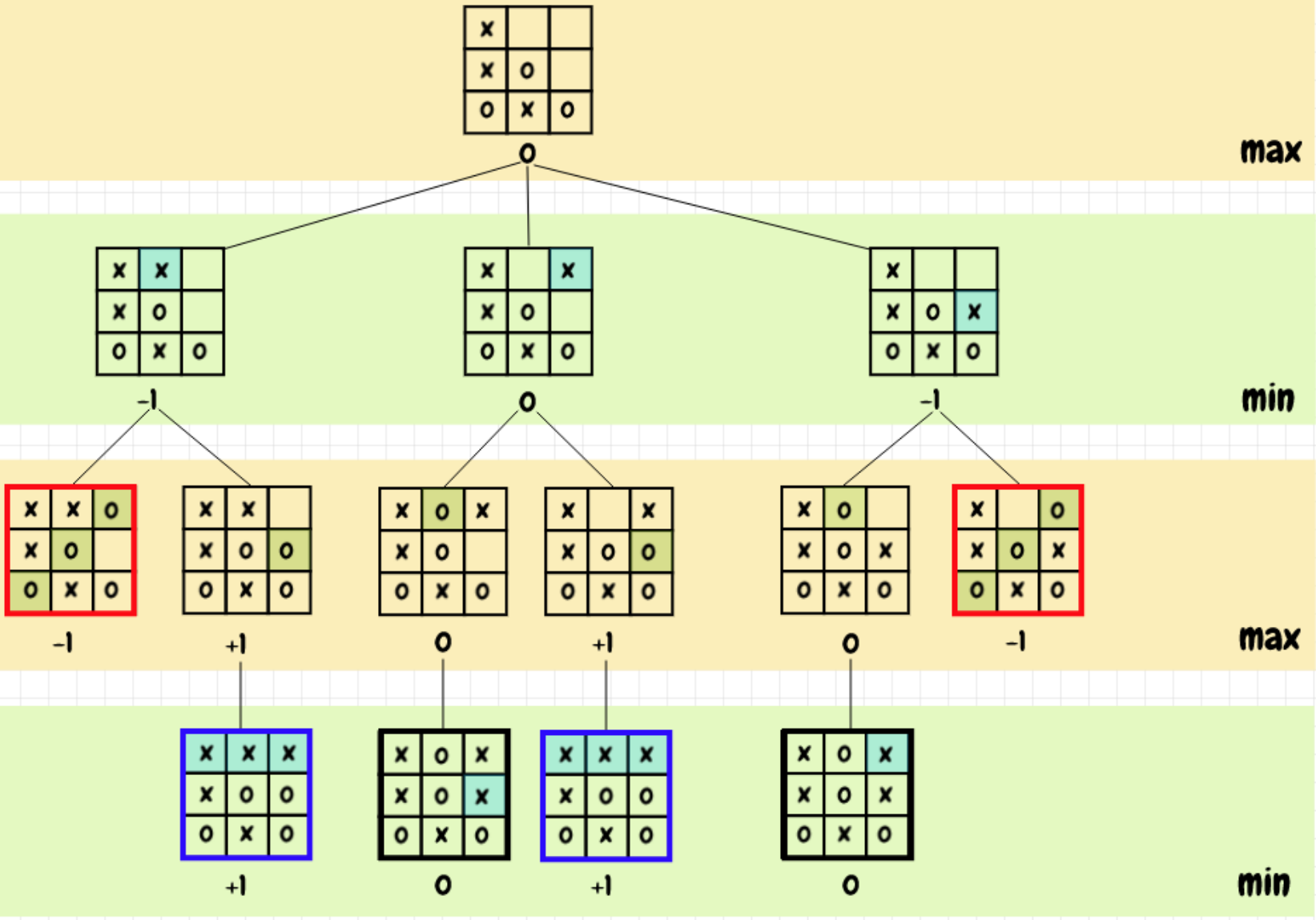
BF = 3 1 4 5 2 0

X wins: +1

O wins: -1

Draw: 0

https://nestedsoftware.com/2019/06/15/tic-tac-toe-with-the-minimax-algorithm-5988.123625.html



Parcours Prefixe

- Récursif

```
static void parcoursPrefixe(Arbre a) {
    if (a == null) return;
    System.out.print(a.contenu + " ");
    parcoursPréfixe(a.filsG);
    parcoursPréfixe(a.filsD);
}
```

- Itératif (impératif)

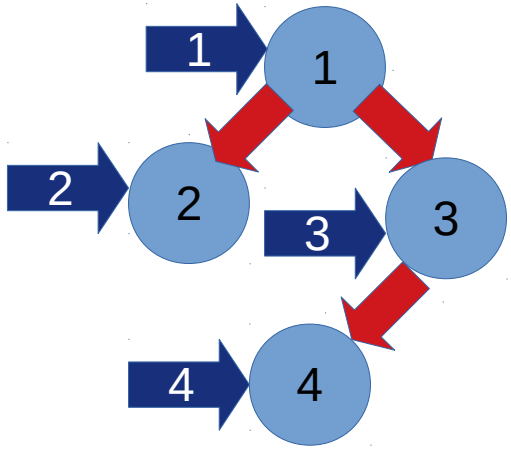
  EXERCICE !!!!!!!!! (après le parcours largeur ; astuce remplacer file par pile)

Remarque : Ici, on affiche les éléments de l'arbre, mais on pourrait vouloir effectuer n' « importe quelle » action ; Il faut passer en argument une méthode à la méthode de parcours (on parle d'ordre supérieur, comme nous l'avons fais avec les flots au premier semestre) ; Voir TP !

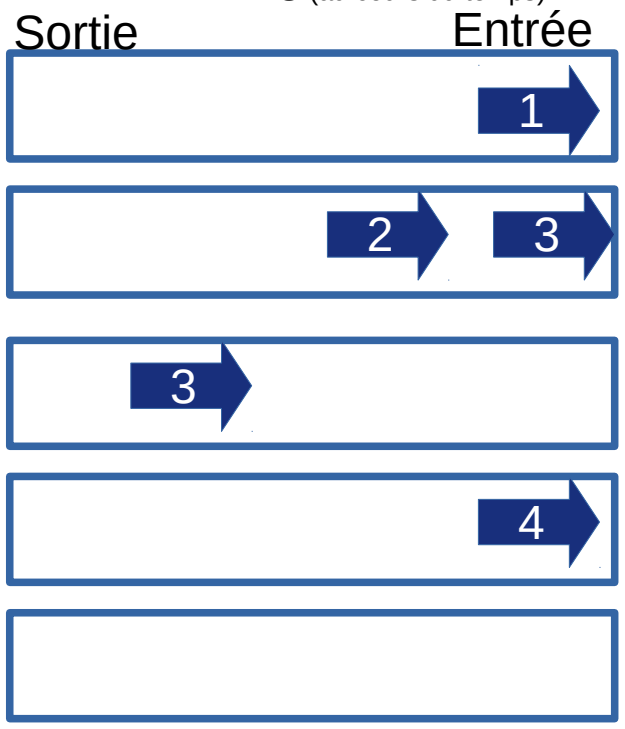
Parcours en largeur

```
static void parcoursLargeurI(Arbre a) {
  if (a == null) return;
  File f = new File();
  f.put(a);
  while (!f.isEmpty()) {
    a = f.first();
    f.get();
    System.out.print(a.contenu + " ");
    if (a.filsG != null) f.put(a.filsG);
    if (a.filsD != null) f.aput(a.filsD);
  }
}
```

Exemple



File (au cours du temps)



- 3 remarques :
  - Le parcours en largeur en récursif n'est pas élégant, on doit avoir avoir une file et donc ce n'est plus très déclaratif (code==equation)
  - Ici, il faut une file d'arbre… mais nous avons une file de « int ». Il faut reprendre le code pour avoir des arbres à la place des **int**… Le code restera similaire… Donc comment faire pour avoir qu'un unique code pour des files d'**int** et d'arbre ⇒ polymorphisme objet (L2) ou généricité (à la Ocaml/Haskell=polymorphisme paramétrique, fin L2, Ing1/L3). Ce n'est pas compliqué mais il y a des subtilités…
  - Quand on met l'arbre « a » (ou a.filsG/D) dans la file, on ne met pas directement TOUT le sous-arbre mais l'objet, donc un pointeur (qui point vers la cellule, pointeurs représentés ici par les flèches bleues numérotées)

Solution parcours prefixe

```
static void parcoursPrefixeIteratif(Arbre a) {
  if (a == null)  return;
  Stack p = new Stack();
  p.push(a);
  while (!p.isEmpty())  {
    a = p.top();
    p.pop();
    System.out.print(a.contenu + " ");
    if (a.filsD != null) p.push(a.filsD);
    if (a.filsG != null) p.push(a.filsG);
  }
}
```

Arbre binaire de recherche

- Très bien expliquer dans
  - http://gallium.inria.fr/~maranget/X/421/poly/arbre-bin.html
  - Un arbre binaire **A** est un arbre binaire de recherche si, pour tout nœud **N** de **A**, les contenus des nœuds du sous-arbre gauche de **N** sont strictement inférieurs au contenu de **N**, et que les contenus des nœuds du sous-arbre droit de **N** sont strictement supérieurs au contenu de **N**
- Manipulation
  - Rechercher si présent (idée==comme un dictionnaire, si on recherche « x », on compare au contenu du nœud et on peut alors savoir s'il faut continuer dans filsG ou filsD)
  - Insérer x == comme pour rechercher, on descend dans l'arbre en respectant l'ordre puis on insère x en tant que nouvelle feuille
  - Supprimer un élément
    - Facile s'il faut supprimer 13...
    - Plus difficile en général car si dans cet exemple, il faille supprimer « 3 » on perd un nœud alors qu'il y a 2 fils… un nœud doit remonter pour remplacer « 3 » (le plus petit des plus grand typiquement)
    - Il faut donc beaucoup jouer avec les pointeurs, plutôt niveau L2…
- Il existe aussi les arbres équilibrés, AVL, rouge-noir, Fibonacci, B-tree, *etc.*