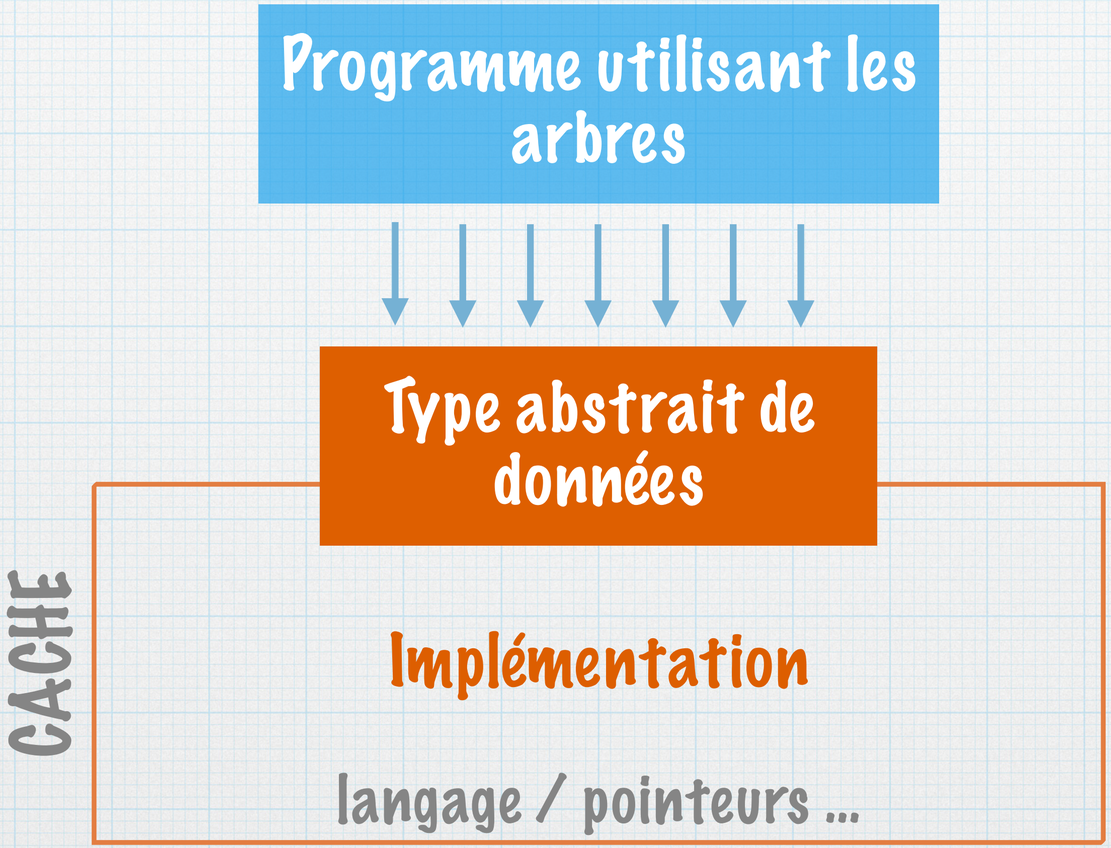
Algorithmique et structures de données

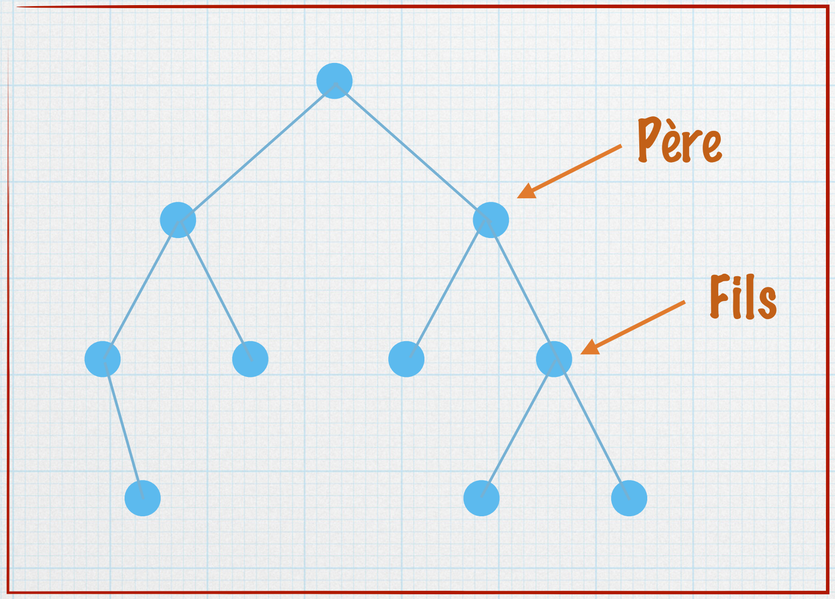
Cours 4 : arbres

Structures de données : spécification et implémentation

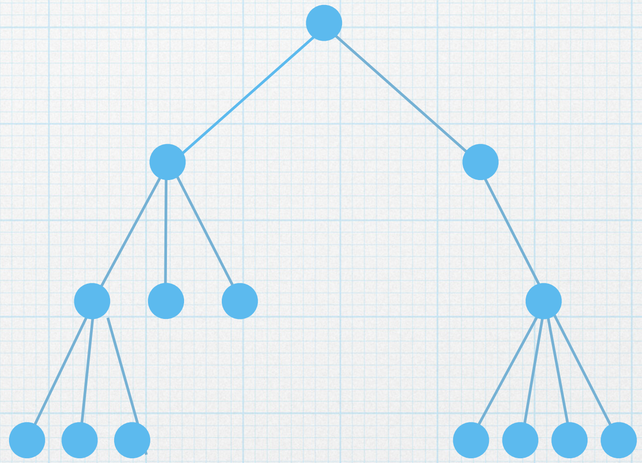


Arbres

Données stockées de manière hiérarchique

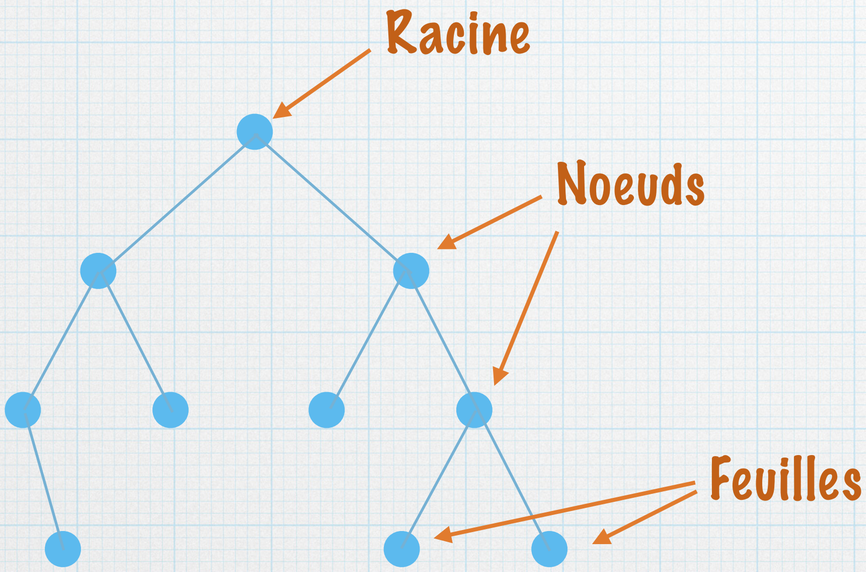


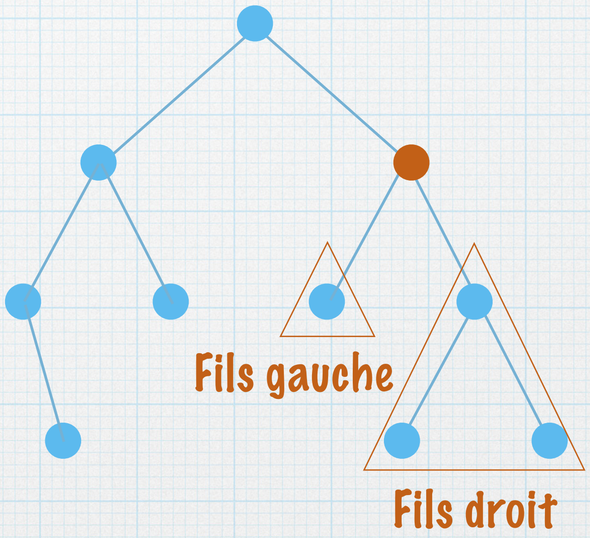
Arbre binaire :
Chaque noeud a au plus 2 fils



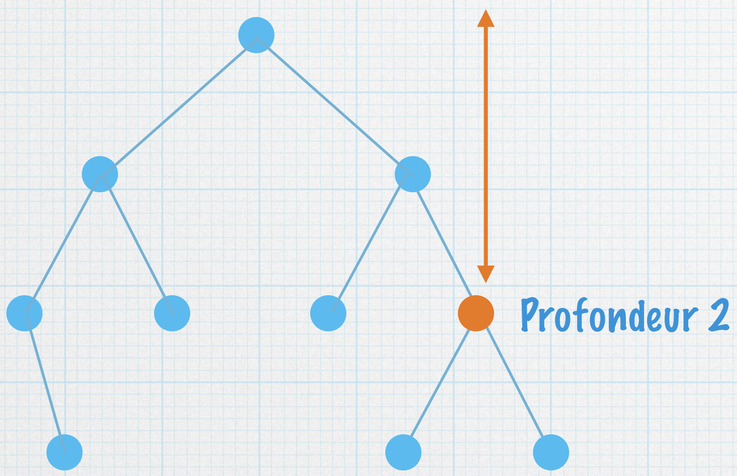
Arbre d'arité variable :
Chaque noeud a un nombre variable de fils

Vocabulaire (1/3)

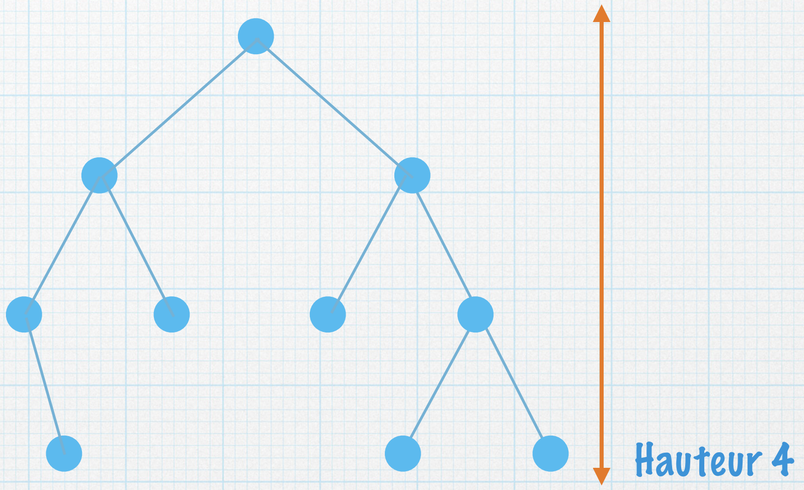




Vocabulaire (2/3)



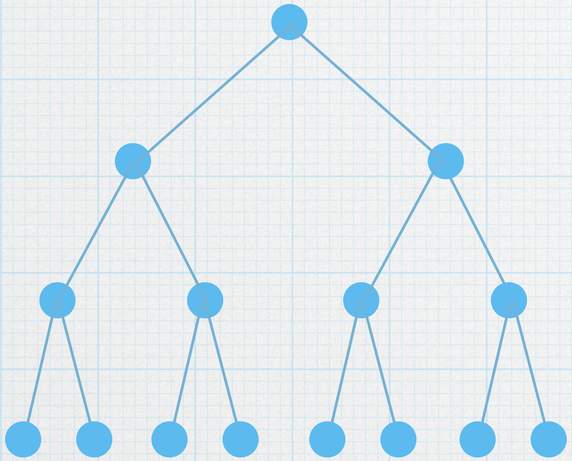
Profondeur d'un noeud
longueur du chemin
depuis la racine



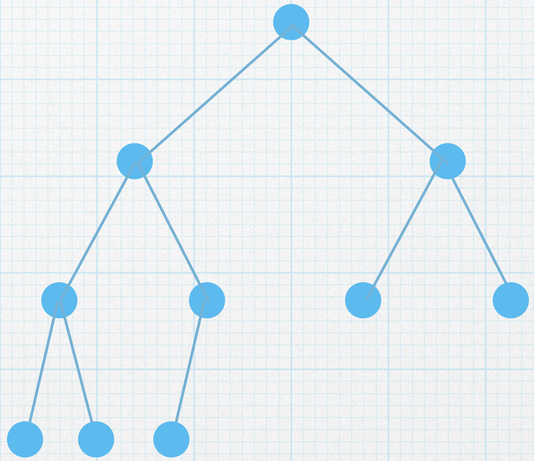
Hauteur d'un arbre
maximum des profondeurs
des noeuds + 1

$$\text{hauteur}(a) = \begin{cases} 1 \text{ si } a \text{ est une feuille} \\ \max(\text{hauteur}(\text{filsg}(a)), \text{hauteur}(\text{filsd}(a))) + 1 \end{cases}$$

Vocabulaire (3/3)



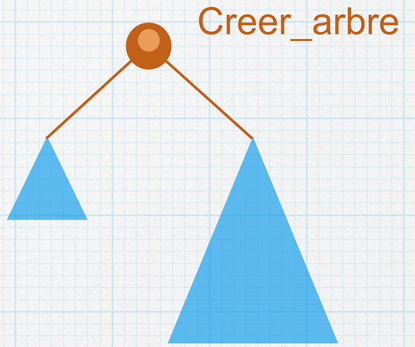
Arbre binaire parfait
Toutes les feuilles sont à la même distance de la racine

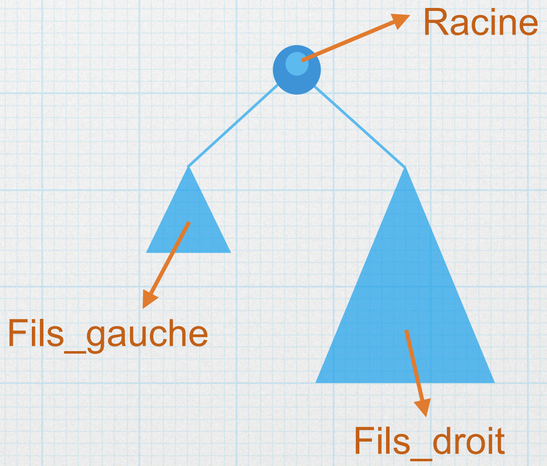


Arbre binaire complet
Tous les niveaux sont complets sauf éventuellement le dernier (où les feuilles sont stockées à gauche)

TAD
Arbres

- Type abstrait Arbres binaires
  - Nom : Arbre
  - Utilise : Element, Bool
  - Opérations
    - Constructueurs
      - Creer_arbre_vide : void -> Arbre
    - Transformateurs
      - Creer_arbre : Element, Arbre, Arbre -> Arbre
      - Racine : Arbre -> Element
      - Fils_gauche, File_droit : Arbre -> Arbre
    - Observateurs
      - Est_arbre_vide : Arbre -> Bool
  - Préconditions
    - Racine(a) : NON(est_arbre_vide(a)
    - Fils_gauche(a), Fils_droit(a) : NON(est_arbre_vide(a)
  - Axiomes
    - Est_arbre_vide(Creer_arbre_vide()) == VRAI
    - Est_arbre_vide(Creer_arbre(e,a1,a2)) == FAUX
    - Racine(Creer_arbre(e,a1,a2)) == e
    - Fils_gauche(Creer_arbre(e,a1,a2)) == a1
    - Fils_droit(Creer_arbre(e,a1,a2)) == a2

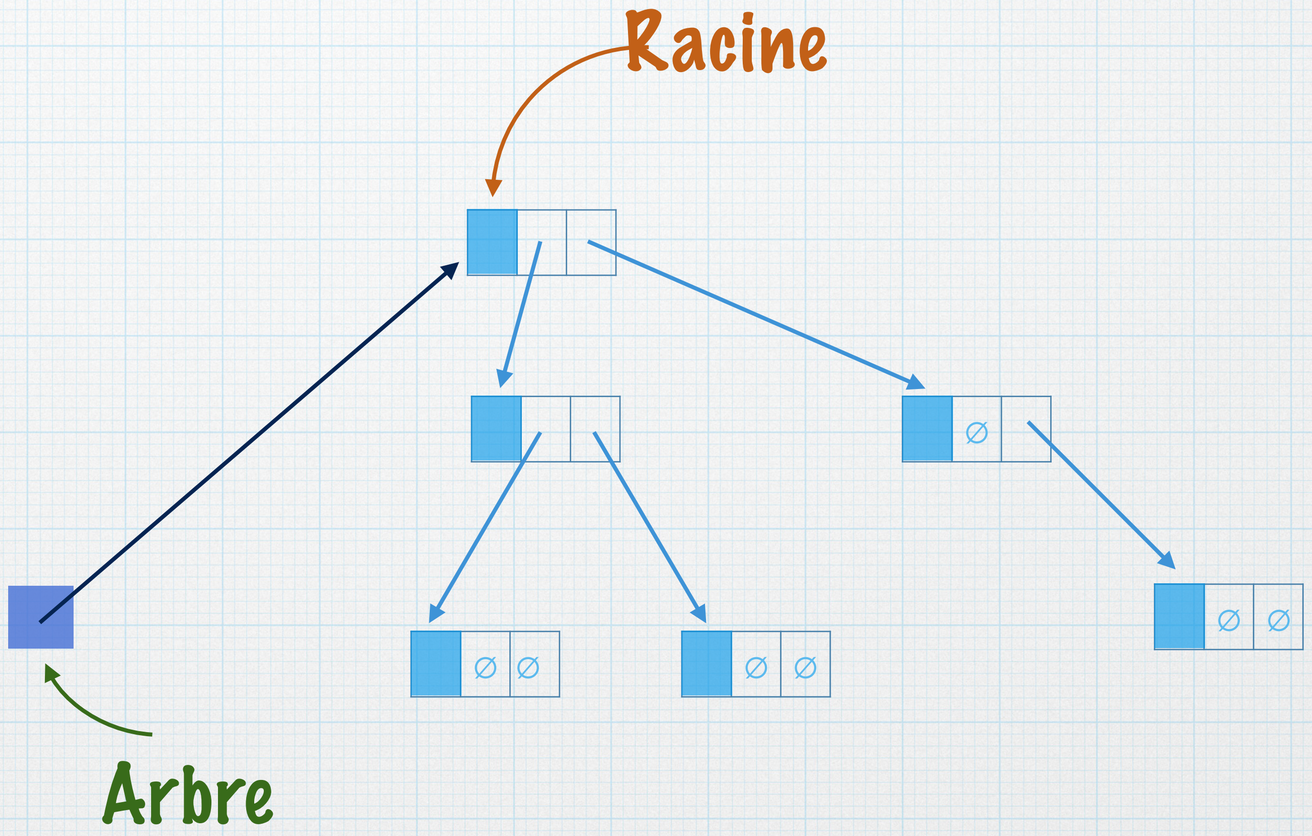




Implémentation

Implémentation du TAD Arbre

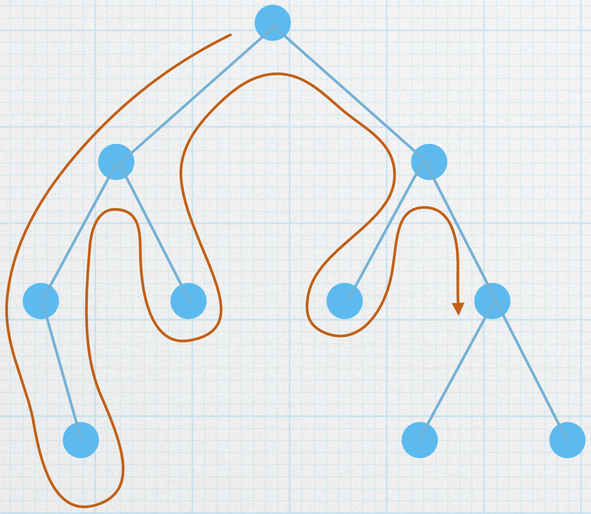
```
typedef int Data ;
struct z_noeud {
    Data elt ;
    struct z_maillon * fils_g, *fils_d ; } ;
typedef struct z_noeud noeud ;
typedef noeud * Arbre ;
```



Algorithmique

Parcours en profondeur

Depth-first search



Récursif

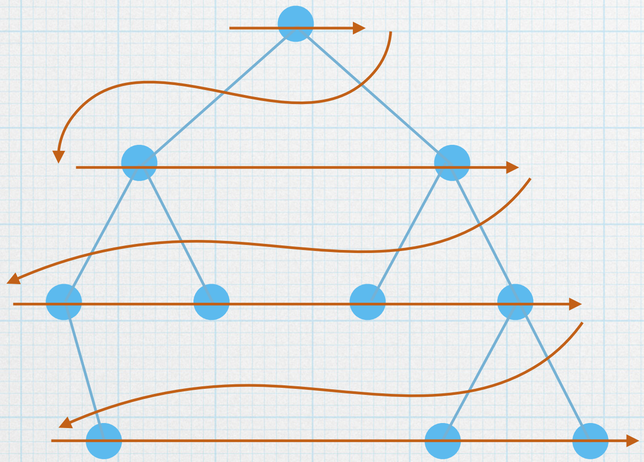
Parcours ordonné par :

- Parcours du fils gauche en premier
- Puis du fils droit

```
Type parcours_profondeur (Arbre a)
{
    if (est_arbre_vide(a))
        // Traiter arbre vide …
    else
    {
        // Traiter racine …
        parcours_profondeur (fils_g(a)) ;
        parcours_profondeur (fils_d(a)) ;
    }
}
```

Parcours en largeur

Breadth-first search



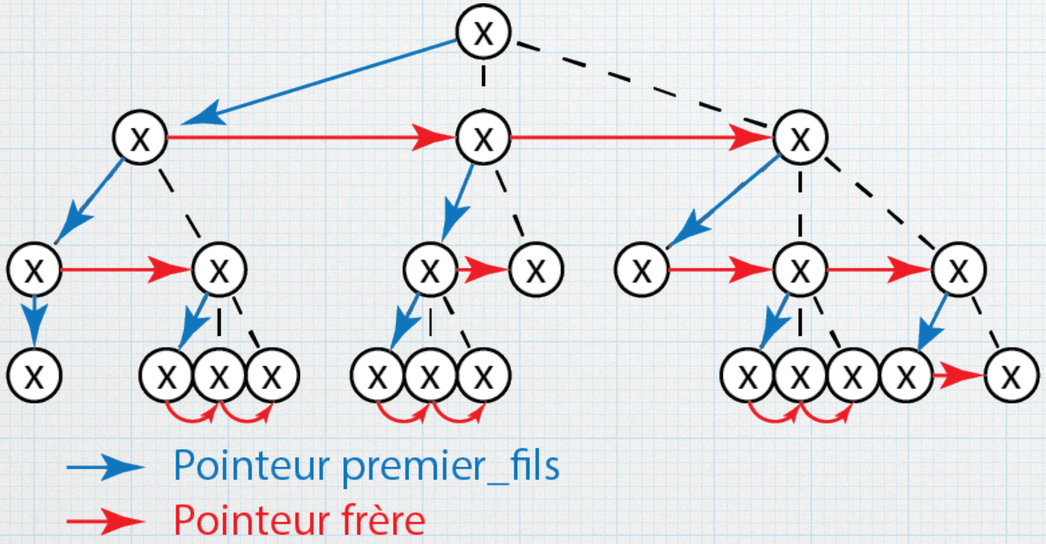
Utilise une file

Parcours ordonné par profondeurs successives

```
Type parcours_largeur (Arbre a)
{
    Queue q = create_empty_queue () ;
    Arbre a_tmp ;
    enqueue(q, a) ;
    while (! is_empty_queue(q))
    {
        a_tmp = dequeue(q) ;
        if (! est_arbre_vide(a_tmp))
        {
            // Traiter racine …
            enqueue (fils_g(a)) ;
            enqueue (fils_d(a)) ;
        }
    }
}
```

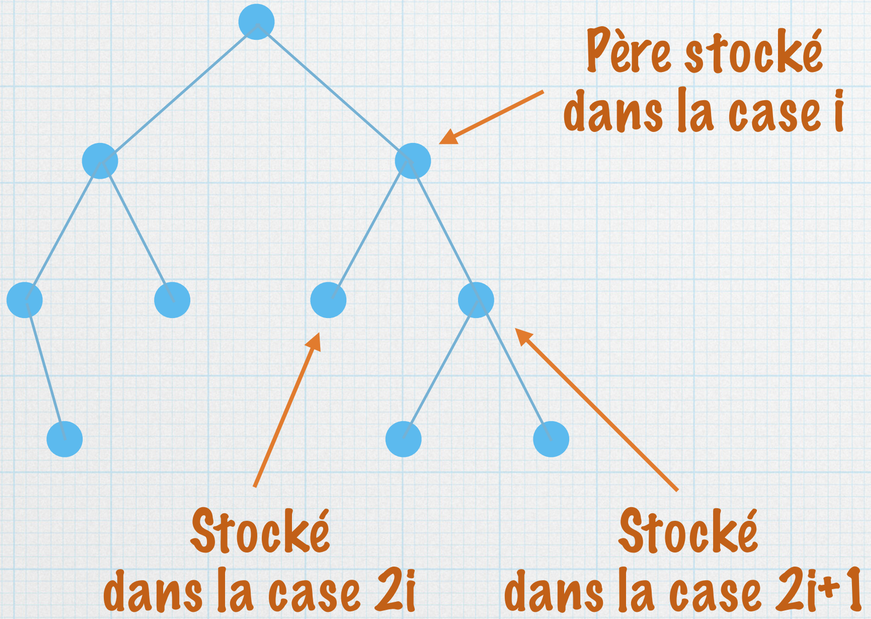
Autres implémentations ...

Implémentation des arbres n-aires

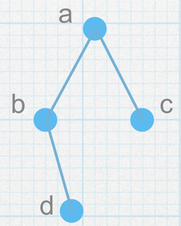


Implémentation par
premier fils / frère
(permettant un nombre de fils
quelconque)

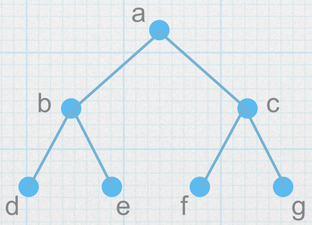
Implémentation par tableaux (arbres complets / parfaits / tas)



Base du « tri par tas »



0	1	2	3	4	5	6	7
	a	b	c		d		



0	1	2	3	4	5	6	7
	a	b	c	d	e	f	g