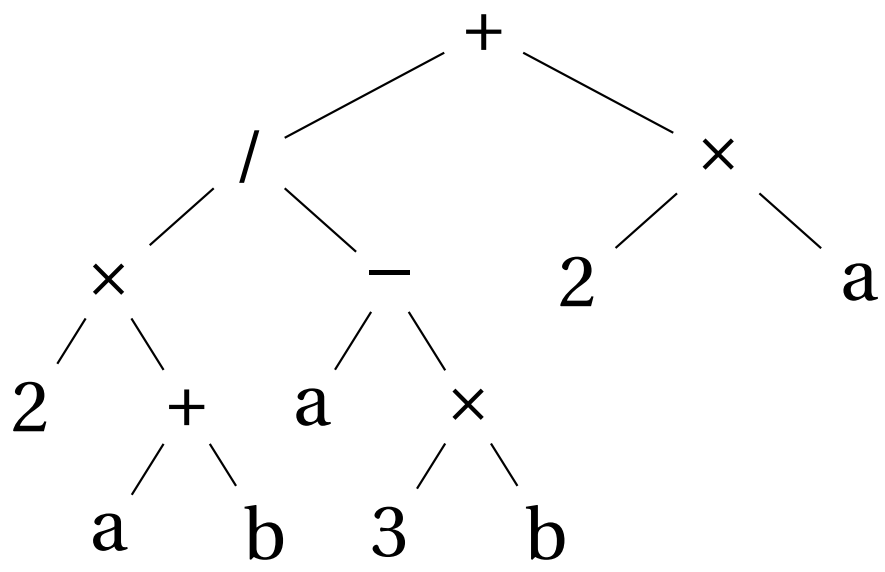
Arbres : propriétés fondamentales

Généralités

Une des structures les plus importantes car une des plus utilisées.

- Modélisation
- Organisation des données : fichiers, processus
- Structures de données : arbres binaire de recherche, tas, arbres AVL
- utilisé dans beaucoup d'algorithmes sur les graphes notamment
- Arbre d'expression

$$(2 \times (a+b)) / (a - 3 \times b) + 2 \times a$$



Terminologie

- **graphe :** ensemble de nœuds reliés entre eux par des arcs (orientés) ou des arêtes (non orienté).
- **arbre :** graphe connexe sans cycles
  - propriété : chemin unique entre deux nœuds.
- **arbre enraciné :** arbre avec un nœud particulier appelé **racine**.
  - propriété : chemin unique de tout nœud vers la racine.
- **Parent** d'un nœud : nœud qui le suit dans le chemin vers la racine.
  - racine : pas de nœud parent
  - autres nœuds : un unique parent
- **Enfants** d'un nœud : ensemble des nœuds qui l'admettent pour parent.
  - un nœud sans enfant est appelé **feuille**
  - sinon nœud **intérieur**
  - tout nœud est soit feuille soit nœud intérieur
- **Arbre binaire :** arbre enraciné où chaque nœud a au plus 2 enfants.

Sous-arbre

- **ancêtres** d'un nœud : ensemble des nœuds sur le chemin entre ce nœud et la racine
  - la racine est un ancêtre de tous les nœuds, racine comprise
- un nœud x est un **descendant** d'un nœud a si et seulement si a est un ancêtre de x
  - tout nœud est descendant de la racine
- un nœud x avec tous ses descendants (et les arêtes induites) est appelé **sous-arbre** (issu de x)
  - le sous-arbre issu de x est un arbre dont la racine est x

Mesures sur les arbres

- **taille** d'un arbre : nombre de nœuds
- **longueur** d'un chemin : nombre de nœuds qui le constituent
- **profondeur d'un nœud** : longueur du chemin qui le relie à la racine
- **hauteur** d'un arbre : profondeur maximale d'un nœud
  - la hauteur d'un arbre à un seul nœud est 1
  - la hauteur d'un arbre vide est 0
- **nombre de feuilles**, etc...

Arbres binaires

Arbre enraciné dont tous les nœuds ont au plus 2 enfants qui sont orientés : on distingue un enfant gauche et un enfant droit.

sous-arbre gauche (resp. droit) d'un nœud x : sous-arbre issu de l'enfant gauche (resp. droit) de x

Arbre binaires particuliers

- **arbre filiforme** : chaque nœud interne a exactement un fils (cf liste). Supposons qu'il soit de taille n :
  - nombre de feuilles : 1
  - hauteur : n
- **arbre binaire complet** de hauteur h : tous les nœuds de profondeur strictement inférieure à h ont exactement deux fils.
  - nombre de nœuds de profondeur d : 2^{d-1}
  - taille de l'arbre : $2^0 + 2^1 + \cdots + 2^{h-1} = 2^h - 1$

Propriétés des arbres binaires

théorème

Tout arbre binaire de taille n et de hauteur h vérifie :

$$\lceil \log(n+1) \rceil \leq h \leq n$$

Vient de l'inégalité suivante :

$$h \leq n \leq 2^h - 1$$

- borne inférieure : arbre filiforme.
- borne supérieure : arbre complet.

Corollaire : borne inférieure pour les tris par comparaisons.

Arbre binaire en tant que type abstrait

- `arbre_vide(a: Arbre): booleen` : teste si l'arbre est vide
- `racine(a: Arbre): entier` : retourne la valeur à la racine de l'arbre
- `gauche(a: Arbre): Arbre` : retourne le sous-arbre gauche
- `droite(a: Arbre): Arbre` : retourne le sous-arbre droit

Précondition pour les trois dernières primitives : l'arbre ne doit pas être vide

Implémentations possibles des arbres binaires

- par l'utilisation d'un tableau (cf tas)
- par chaînage

Par chaînage

```
struct Noeud{
   val: entier
   g: référence sur Noeud
   d: référence sur Noeud
}
```

Le type Arbre est une référence sur Noeud (comme listes chaînées).
Arbre vide vaut `Nil`.

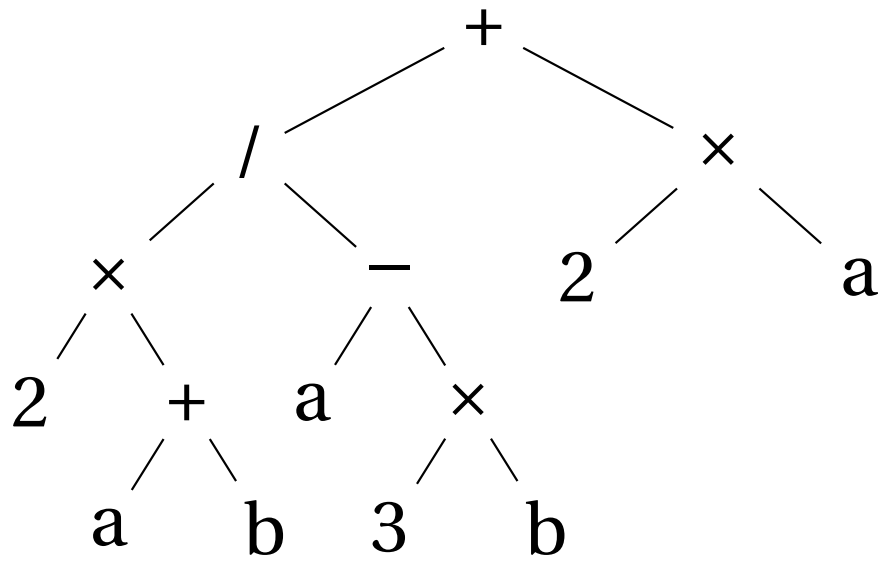
Parcours d'arbre

On veut parcourir l'ensemble des nœuds d'un arbre pour les traiter (par exemple les afficher).

- pas d'ordre naturel comme pour les listes
- par convention, on choisit toujours le côté gauche avant le droit
- plusieurs types de parcours (vrai aussi pour les graphes)
  - en profondeur
  - en largeur

Parcours en profondeur

- on descend à gauche tant que c'est possible et sinon à droite
- on voit 3 fois chaque nœud interne, donc on peut choisir à quel moment on le traite
  - première fois : parcours **préfixe**
  - deuxième fois : parcours **infixe**
  - troisième fois : parcours **postfixe**



Parcours en profondeur

Algorithme : parcours_profondeur(a : Arbre)

Entrées : a : Arbre

Résultat : Traitement de tous les noeuds de l'arbre

```
1 si arbre_vide(a) = Faux
2 |   traiter en préfixe la racine de a;
3 |   parcoursProfondeur(gauche(a));
4 |   traiter en infixe la racine de a;
5 |   parcoursProfondeur(droite(a));
6 |   traiter en postfixe la racine de a;
```

Complexité : n traitements où n est la taille de l'arbre.

Algorithme itératif?

Parcours en largeur

On traite les nœuds par ordre de profondeur :

- racine
- ensemble des enfant de la racine
- ensemble des enfants des enfants de la racine
- etc... on parcourt par profondeur croissante et de gauche à droite

Parcours en largeur

Algorithme : parcours_largeur(a : Arbre)

Entrées : a : Arbre

Sorties : Traitement de tous les noeuds de l'arbre

Variables locales : $a\_traiter$: File d'arbres, a_tmp : Arbre

```
1 a_traiter ← creer_file();
2 insere_file(a_traiter, a);
3 tant que file_vide(a_traiter) = Faux
4 |   a_tmp ← extrait_file(a_traiter);
5 |   si arbre_vide(a_tmp) = Faux
6 |   |   traiter racine(a_tmp);
7 |   |   insere_file(gauche(a_tmp), a_traiter);
8 |   |   insere_file(droite(a_tmp), a_traiter);
```

Complexité : $O(n)$ traitements où n est la taille de l'arbre.

Quel parcours obtient-on si on remplace la file par une pile ?