Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

# Science informatique : TD 5

Année 2023–24

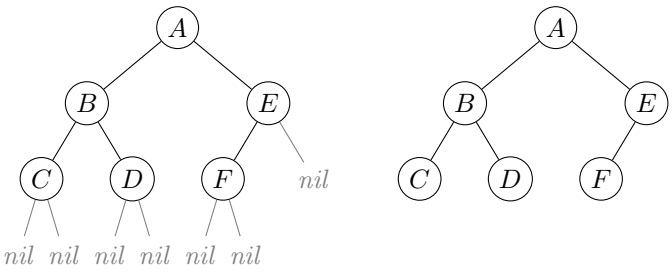
**Arbres**

Un arbre est un graphe non orienté connexe (c'est-à-dire *d'un seul morceau*), sans cycle. Dans la plupart des applications, on distingue un nœud [1] particulier d'un arbre, qu'on appelle la racine. Une restriction consiste alors à considérer des arbres *binaires*, dont on a vu une définition récursive. Un arbre binaire est :

— soit l'arbre vide, qu'on note souvent *nil* et qu'on ne représente généralement pas dans les dessins (on le représentera par la valeur `None` en python) ;

— soit une racine ayant un enfant gauche et un enfant droit, qui sont tous deux des arbres binaires.

Ainsi, à gauche ci-dessous le graphe est un arbre binaire, qu'on représente dans la suite comme dessiné à droite, sans les arbres *nil* :




La racine de cet arbre est le nœud $A$ qui a deux enfants : l'enfant gauche a $B$ pour racine, et l'enfant droit a $E$ pour racine. Les enfants du nœud $C$ sont deux arbres vides *nil*. Une feuille est un nœud qui possède l'arbre vide comme enfants gauche et droit : les feuilles de l'arbre du dessus sont $C$, $D$ et $F$.

**Exercice 1** On appelle *taille* d'un arbre son nombre de nœuds : la taille de l'arbre ci-dessus est 6. La *profondeur* d'un nœud est la longueur de la branche qui mène de la racine à ce nœud : dans l'arbre ci-dessus, le nœud $A$ est à profondeur 0, le nœud $B$ à profondeur 1 et le nœud $F$ à profondeur 2. On appelle *hauteur* d'un arbre la profondeur maximale de l'un de ses nœuds : la hauteur de l'arbre ci-dessus est donc 2. La hauteur d'un arbre vide est arbitrairement fixée à $-1$.

1. Trouver un arbre de taille 8 et de hauteur 5.
2. Quelle est la taille minimale d'un arbre de hauteur 5 ? Et plus généralement pour un arbre de hauteur $h \in \mathbf{N}$ ?
3. Combien existe-t-il d'arbres différents de hauteur $h$ de taille minimale ?
4. Inversement, quelle est la taille maximale d'un arbre de hauteur $h \in \mathbf{N}$ ? Combien existe-t-il de tels arbres de hauteur $h$ et de taille maximale ?
5. En déduire la hauteur minimale d'un arbre de taille $n \in \mathbf{N}$.

**Exercice 2** On connaît trois algorithmes de parcours d'arbre : le parcours préfixe, postfixe (ou suffixe) et infixe. Ces trois parcours ne diffèrent que par l'ordre dans lequel on visite la racine, l'enfant gauche et l'enfant droit :

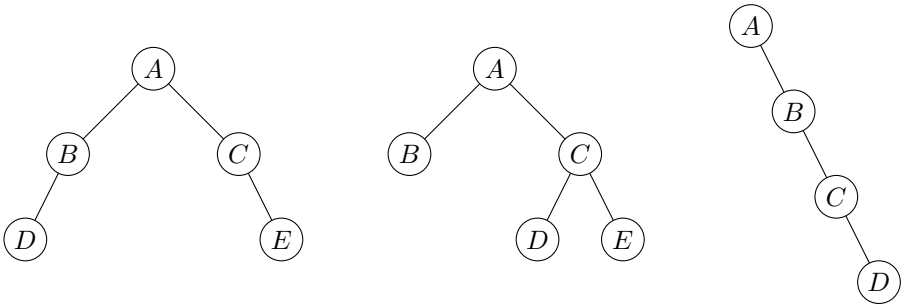
```
1 def parcours_prefixe(noeud):
2     if not(est_vide(noeud)):
3         print(valeur(noeud))
4         parcours_prefixe(enfant_gauche(noeud))
5         parcours_prefixe(enfant_droit(noeud))
```

1. Nœud et sommet sont des synonymes dans ce cours.

```python
def parcours_postfixe(noeud):
    if not(est_vide(noeud)):
        parcours_postfixe(enfant_gauche(noeud))
        parcours_postfixe(enfant_droit(noeud))
        print(valeur(noeud))
```

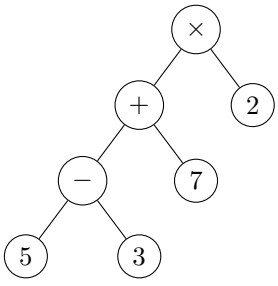
```python
def parcours_infixe(noeud):
    if not(est_vide(noeud)):
        parcours_infixe(enfant_gauche(noeud))
        print(valeur(noeud))
        parcours_infixe(enfant_droit(noeud))
```

1. Appliquer les trois algorithmes à la racine des arbres binaires suivants, en montrant les valeurs affichées.

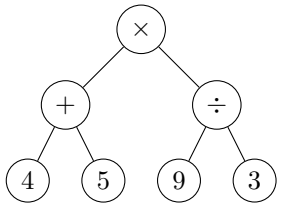



2. Trouver un arbre binaire dont le parcours préfixe est $A, B, C, D, E, F$ et le parcours infixe est $C, B, E, D, F, A$. (*Attention, on cherche bien un seul arbre qui admet ces deux parcours à la fois !*)
3. Trouver un arbre binaire dont le parcours préfixe est $A, B, D, C, E, G, F$ et le parcours postfixe est $D, B, G, E, F, C, A$. En trouver ensuite un deuxième vérifiant ces mêmes conditions.

**Exercice 3** Les arbres sont partout... même dans vos calculatrices. Lorsque vous tapez un calcul, par exemple $\big((5-3)+7\big) \times 2$, la calculatrice, elle, stocke un arbre binaire en mémoire. Pour cela, elle essaie de décomposer le calcul en deux tant que c'est possible. Au début, elle trouve donc l'opération la plus prioritaire, le produit, pour décomposer le calcul en le produit de $(5-3)+7$ et de 2. À nouveau, elle décompose le sous-calcul $(5-3)+7$ comme la somme de $5-3$ et de 7, et ainsi de suite jusqu'à ce que les sous-calculs soient tous des constantes. On obtient donc la représentation par l'arbre ci-dessous, qu'on résume à droite en conservant dans chaque nœud uniquement l'opération qui permet la décomposition :



1. Quelle est l'expression arithmétique dont l'arbre est le suivant ?

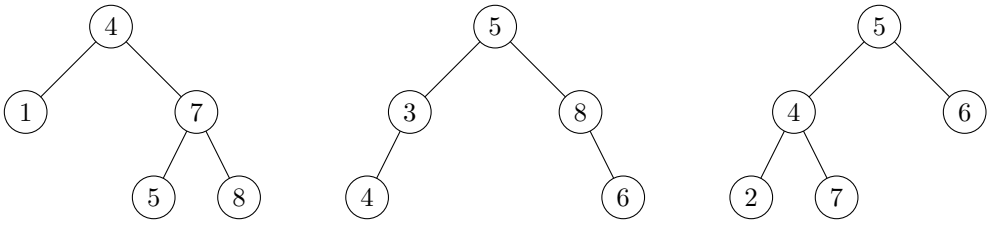



2. L'évaluation d'une expression arithmétique peut se faire par le parcours postfixe de l'arbre binaire le représentant. Modifier l'algorithme de parcours postfixe pour composer une fonction récursive `evaluer_expression` renvoyant la valeur associée au nœud en argument.
3. Que modifier si l'on veut permettre l'utilisation d'un opérateur *unaire* de signe ? Par exemple, on veut pouvoir évaluer l'expression $-4+5$.
4. L'affichage d'une expression sur l'écran de la calculatrice requiert, lui, un parcours infixe de l'expression. Modifier l'algorithme de parcours infixe pour composer une fonction récursive `afficher_expression` qui affiche l'écriture bien parenthésée de l'expression associée au nœud en argument.
5. Une autre notation, utilisée dans certaines calculatrices Hewlett-Packard à partir de la fin des années 1960, est la *notation polonaise inversée*. Elle consiste à écrire dans la calculatrice l'expression produite par le parcours postfixe (non parenthésé) de l'arbre. Donner la notation polonaise inversée des expressions du début de l'exercice.
6. L'intérêt de la notation polonaise inversée est justement qu'il n'est pas utile de construire l'arbre pour l'évaluer, puisqu'on peut le faire *à la volée*. Pour cela, on utilise une structure de pile. En partant d'une pile vide :
   — à chaque fois qu'on lit un nombre, on l'empile ;
   — à chaque fois qu'on lit un opérateur binaire, on dépile deux nombres, on réalise l'opération demandée (attention à l'ordre pour la division et la soustraction !) puis on empile le résultat.

   Une fois arrivé à la fin de l'expression, on dépile le nombre qui reste dans la pile : il s'agit du résultat de l'opération. Illustrer ce procédé sur les expressions obtenues à la question précédente.

**Exercice 4** Un *arbre binaire de recherche* (ABR) est un arbre tel que chaque noeud $x$ de l'arbre vérifie la propriété suivante : toutes les valeurs des noeuds dans l'enfant gauche de $x$ sont inférieures à la valeur de $x$, et toutes les valeurs des noeuds dans l'enfant droit de $x$ sont supérieures à la valeur de $x$.

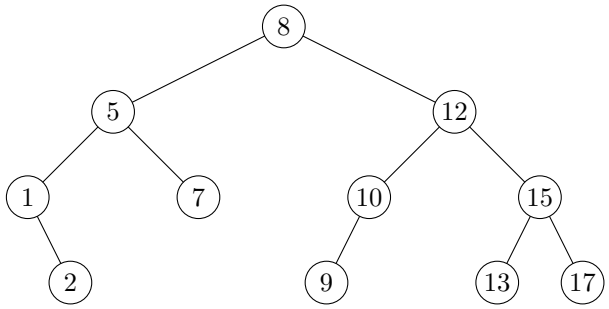
1. Lesquels des arbres suivants sont des ABR et pourquoi ?




2. Un arbre binaire de recherche sert à stocker un ensemble de valeurs (un dictionnaire ou un annuaire par exemple). Une opération importante est donc la recherche d'une valeur particulière dans cet ensemble. On a vu en cours un algorithme récursif permettant d'effectuer cette recherche :

```python
def rechercher_abr(noeud, x):
    if est_vide(noeud):
        return False
    elif x == valeur(noeud):
        return True
    elif x < valeur(noeud):
        return rechercher_abr(enfant_gauche(noeud), x)
    else:
        return rechercher_abr(enfant_droit(noeud), x)
```

Appliquer cet algorithme pour rechercher les valeurs 15, 7, 2, 11, 3 dans l'ABR suivant, en donnant la liste des nœuds où l'algorithme s'appelle récursivement.




3. Une autre opération intéressante pour un ABR est la possibilité d'insérer un élément nouveau à l'intérieur : c'est crucial si on veut représenter un annuaire qu'on peut mettre à jour facilement. Construire un ABR en insérant une par une (à partir de l'arbre vide) les valeurs $9, 5, 12, 2, 15, 7, 11$ dans l'ordre. Ensuite, construire un autre ABR en insérant les mêmes valeurs, mais dans l'ordre $2, 5, 7, 9, 11, 12, 15$. Quelle est la hauteur des deux arbres et quelle différence y aura-t-il en termes de complexité lors d'une recherche dans le pire des cas ?
4. Appliquer l'algorithme de parcours infixe aux arbres binaires construits dans la question précédente. Dans quel ordre l'algorithme affiche-t-il les valeurs ?
5. En déduire un algorithme de tri de tableau (qu'on pourra décrire avec des phrases) qui utilise un ABR comme structure de données auxiliaire.