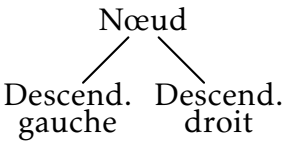
Arbres binaires

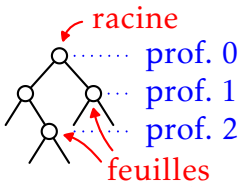
Définition

Un **arbre binaire** peut être défini de façon récursive :

- Un arbre binaire peut être vide.
- S'il n'est pas vide, il est composé d'un **nœud** relié à deux autres arbres binaires, appelés **descendants gauche** et **droite**. Ils sont reliés à l'aide d'**arêtes**.

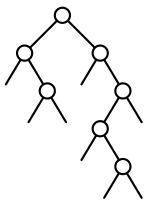


On appelle **racine** le nœud qui n'est pas dans un descendant d'un autre nœud. En général, on le représente en haut. La **taille** d'un arbre est le nombre de nœuds qu'il contient. On dit que les nœuds qui ont deux descendants vides sont des **feuilles**.



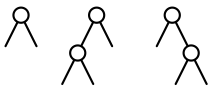
La **profondeur** d'un nœud est le nombre d'arêtes nécessaires pour aller de la racine à ce nœud. On peut définir la **hauteur** d'un arbre comme étant le nombre de nœuds entre la racine et la feuille de profondeur maximale. C'est donc la profondeur maximale plus 1. L'arbre vide a une hauteur de 0 et un arbre avec un unique nœud a une hauteur de 1. L'arbre ci-dessus a donc une taille de 4 et une hauteur de 3.

Exercice 1 : Déterminer la taille et la hauteur de l'arbre ci-contre.



On peut parfois définir la hauteur comme étant la profondeur maximale. Dans ce cas, l'arbre vide n'a pas de hauteur et un arbre avec un seul nœud a une hauteur de 0.

Il y a un seul arbre binaire vide, une seule forme pour les arbres binaires à un seul nœud et 2 pour ceux à 2 nœuds. Même s'ils sont symétriques, on considère que les deux formes sont différentes.



Exercice 2 :

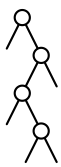
1) Dessiner tous les types d'arbres avec 3 nœuds.
2) Déterminer, en les dessinant ou pas, le nombre d'arbres binaires à 4 nœuds.
3) En utilisant les réponses aux questions précédentes, déterminer le nombre d'arbres à 5 nœuds.

Dans la suite, on pourra omettre les branches amenant à des arbres vides.

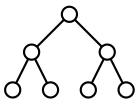
Propriétés

Pour cette partie, on notera N la taille de l'arbre considéré et h sa hauteur. Pour tout arbre binaire, on a :

$$h \leq \mathrm{N} \leq 2^h - 1$$



Pour avoir un arbre de taille minimale et de hauteur maximale, il ne faut qu'un seul nœud par niveau. Par définition, il y a une arête de moins que le nombre de nœuds. On a donc $h = \mathrm{N}$, d'où la minoration.

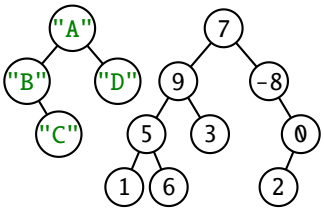


Au contraire, pour avoir la taille maximale et la hauteur minimale, il faut remplir au maximum chaque niveau. On dit qu'un tel arbre est **parfait**. On a alors :

$$\mathrm{N} = 1 + 2 + 4 + \ldots + 2^{h-1} = 2^0 + 2^1 + 2^2 + \ldots + 2^{h-1} = 2^h - 1$$

Parcours d'un arbre binaire

Pour l'instant nous n'avons pas eu besoin de regarder le contenu des nœuds. En pratique, les arbres servent à stocker des informations. On peut ne mettre ces informations que dans les feuilles, mais en général, on le fait dans chaque nœud. Les données peuvent être de n'importe quel type, même si on manipulera principalement des nombres.



La défintion récursive des arbres binaires permet de facilement faire des fonctions récursives. Pour la suite, on utilisera les fonctions suivantes :

Fonction	Description
`racine(arbre)`	Renvoie la valeur à la racine de `arbre`.
`gauche(arbre)`	Renvoie le descendant gauche de `arbre`.
`droite(arbre)`	Renvoie le descendant droit de `arbre`.
`est_vide(arbre)`	Renvoie un booléen indiquant si `arbre` est vide ou non.

Cette définition induit 3 types de parcours : **préfixe**, **infixe** et **suffixe** (ou **postfixe**). À chaque fois, on traite la racine et on fait des appels récursifs sur descendants gauche et droite. La différence vient du moment où est traité la racine.
Voici les structures des parcours préfixe et infixe :

```
def parc_prefixe(arbre):
    if est_vide(arbre):
        faire quelque chose (ou pas)
    else:
        qlq chose avec racine(arbre)
        parc_prefixe(gauche(arbre))
        parc_prefixe(droite(arbre))
```

```
def parc_infixe(arbre):
    if est_vide(arbre):
        faire quelque chose (ou pas)
    else:
        parc_infixe(gauche(arbre))
        qlq chose avec racine(arbre)
        parc_infixe(droite(arbre))
```

Pour un parcours suffixe, la racine est traitée à la fin. À chaque fois, nous avons fait l'appel sur la partie gauche avant celui sur la partie droite. Il est possible de faire le contraire, mais cela ne change pas grand chose dans le principe.

Exercice 3 : On considère les deux fonctions suivantes :

```
def aff_prefixe(arbre):
    if not est_vide(arbre):
        print(racine(arbre))
        aff_prefixe(gauche(arbre))
        aff_prefixe(droite(arbre))
```

```
def aff_infixe(arbre):
    if not est_vide(arbre):
        aff_infixe(gauche(arbre))
        print(racine(arbre))
        aff_infixe(droite(arbre))
```

1) Pour l'arbre avec les lettres ci-dessus, indiquer dans quel ordre sont affichées les valeurs pour chacune des fonctions.
2) Même question avec le second arbre.

On peut remarquer que les nœuds sont toujours visités dans le même ordre, quelque soit le type de parcours.

Exercice 4 : On considère l'arbre ci-contre.

1) Numéroter les flèches pour indiquer dans quel ordre les nœuds sont visités.
2) Indiquer l'ordre dans lequel sont traités les nœuds lors des parcours préfixe, infixe et suffixe.