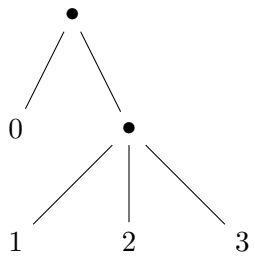
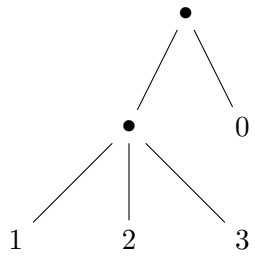
Récursivité

Opérations sur les arbres

On considère des arbres finis avec une racine, où chaque noeud interne a au moins un fils, et où chaque feuille est étiquetée par un entier naturel. Par exemple :



Dans cet exemple, les noeuds internes sont les sommets étiquetés par • et les feuilles sont les noeuds étiquetés par les entiers. La racine est le noeud interne se trouvant tout en haut. Notons aussi que les fils d'un noeud interne sont ordonnés. Ainsi, cet arbre n'est pas le même que l'arbre suivant :



Formellement on peut définir l'ensemble des arbres $\mathcal{A}$ comme suit. L'ensemble des arbres de hauteur au plus 0 est $\mathcal{A}_0 = \mathbb{N}$. Pour $h > 0$, l'ensemble des arbres de hauteur au plus h est

$$\mathcal{A}_h = \mathcal{A}_{h-1} \cup \{(A_1, \ldots, A_p) \mid p \geqslant 1, A_i \in \mathcal{A}_{h-1}\}.$$

Enfin $\mathcal{A} = \bigcup_{h \in \mathbb{N}} \mathcal{A}_h$.

Une représentation naturelle de ces arbres en Python est obtenue en utilisant des listes : la représentation de l'arbre n (réduit à une feuille étiquetée n) est l'entier `n` (de type `int`), la représentation de $(A_1, \ldots, A_p)$ est la liste $[R_1, \ldots, R_p]$ où R_i est la représentation de A_i. Par exemple, les deux arbres ci-dessus ont pour représentation `[0,[1,2,3]]` et `[[1,2,3],0]`.

(a) Écrire une fonction qui calcule le nombre de feuilles d'un arbre.

(b) Écrire une fonction qui calcule la somme des étiquettes des feuilles d'un arbre.

(c) La hauteur d'un arbre est la longueur maximale d'un chemin (descendant) de la racine à une feuille. Donner une définition récursive de la hauteur d'un arbre puis la traduire en une fonction qui calcule la hauteur d'un arbre donné en argument.

(d) Le symétrique d'un arbre A est obtenu en inversant l'ordre des fils en tout noeud interne. Écrire une fonction qui retourne l'arbre symétrique d'un arbre donné en entrée.

(e) Deux arbres sont égaux à rotation près si l'un est obtenu à partir de l'autre par permutation de l'ordre des fils des noeuds internes. Écrire une fonction qui teste si deux arbres sont égaux à rotation près (on pourra dans un premier temps le faire dans le cas où tout noeud interne a exactement 2 fils).

Courbes définies récursivement

On va définir une ligne polygonale récursivement. Le but est d'écrire un algorithme récursif pour la tracer. Pour réaliser les tracés, on va utiliser la librairie `matplotlib`. Voici un exemple qui trace un quadrillage.

```
import matplotlib.pyplot as plt

plt.axis([-0.5, 1.5, -0.5, 1.5])
for i in range(11):
    plt.plot([0,1],[i*0.1,i*0.1],color='b') # [x1,x2],[y1,y2]
    plt.plot([i*0.1,i*0.1],[0,1],color='b') # [x1,x2],[y1,y2]
plt.show()
```

Fixons deux points U et V dans le plan (par exemple les points de coordonnées $(0,0)$ et $(1,0)$). La courbe C_k de niveau k ($k \in \mathbb{N}$) est définie comme ceci :

— Si $k = 0$, c'est le segment $[U, V]$;

— Si $k > 0$, la courbe de niveau k est définie à partir de la courbe de niveau $k-1$ de la façon suivante. Supposons que C_{k-1} est la courbe polygonale de sommets successifs $P_0, P_1, \ldots, P_n$. La courbe C_k est obtenue en remplaçant chaque segment $[A, B]$ de C_{k-1} par la ligne polygonale de sommets successifs A, A_1, A_2, A_3, B où les points A_1, A_2, A_3 sont définis par ce qui suit. Si le vecteur $\overrightarrow{AB}$ a pour composantes (x, y), notons $\vec{N}$ le vecteur de composantes $(-y, x)$: c'est le vecteur obtenu en applicant une rotation d'angle $\pi/2$ à $\overrightarrow{AB}$. Les points A_1, A_2, A_3 sont alors définis par :

$$\begin{cases} \overrightarrow{AA_1} = \frac{1}{3}\overrightarrow{AB} \\ \overrightarrow{AA_2} = \frac{1}{2}\overrightarrow{AB} + \frac{1}{3}\cos\left(\frac{\pi}{6}\right)\vec{N} \\ \overrightarrow{AA_3} = \frac{2}{3}\overrightarrow{AB} \end{cases}$$

Le point A_2 est calculé pour que les vecteurs $\overrightarrow{A_1A_2}$ et $\overrightarrow{A_2A_3}$ soient de norme $\frac{1}{3}\|\overrightarrow{AB}\|$, tout comme $\overrightarrow{AA_1}$ et $\overrightarrow{A_3B}$, et que A_2 soit "du bon côté" de la droite (AB).

(a) Écrire une fonction `courbe(A,B,n)` dessinant la courbe de niveau n obtenue à partir des points de départ $U = A$ et $V = B$.

(b) Écrire un algorithme dessinant la courbe de niveau n obtenue en partant d'un triangle équilatéral comme courbe de niveau 0.

(c) Écrire une version itérative `courbeIter(A,B,n)` de la procédure demandée à la question (a) qui fonctionne comme ceci : elle calcule tout d'abord la suite des sommets de la courbe polygonale de niveau n de A à B, puis affiche tous les segments.

(d) Comparer les deux procédures `courbe` et `courbeIter` en terme de temps et d'espace mémoire.