

TP8 - Récursivité

→ Q.1) Créez un fichier TP8.py dans lequel vous pourrez sauvegarder votre travail.

A) Quelques exemples mathématiques

- Q.2) Écrire une fonction récursive `suite(f, u0, n)` qui prend en paramètre une fonction f , un réel u_0 et un entier naturel n et qui renvoie la valeur du n -ième terme de la suite définie par u_0 et pour $k \geq 0$, $u_{k+1} = f(u_k)$.
Tester cette fonction pour calculer u_{10} lorsque $f : x \mapsto \sin(x)$ et $u_0 = 1$.
- Q.3) Écrire une fonction récursive `PGCD(a, b)` qui renvoie le PGCD des entiers naturels a et b (avec $a \geq b \geq 0$) en utilisant l'algorithme d'Euclide.
- Q.4) Écrire une fonction récursive `binom(n, p)` qui renvoie la valeur de $\binom{n}{p}$. On utilisera la formule de Pascal!

On considère la suite de Fibonacci :

$$\begin{cases} F_0 = 0, F_1 = 1 \\ \forall n \in \mathbb{N}, F_{n+2} = F_{n+1} + F_n \end{cases}$$

- Q.5) Écrire une fonction récursive `fibo_rec1(n)` qui prend en argument un entier naturel n et renvoie la valeur de F_n .
- Q.6) On note C_n le coût de l'exécution de l'instruction `fibo_rec1(n)`. Déterminer une relation de récurrence entre C_n , C_{n-1} et C_{n-2} .
- Q.7) Montrer que pour tout $n \geq 0$, $C_n \geq \left(\frac{1+\sqrt{5}}{2}\right)^n$, puis déterminer la complexité de la fonction `fibo_rec1`.
- Q.8) Écrire une fonction itérative `fibo_iter(n)` qui prend en argument un entier naturel n et renvoie la valeur de F_n puis évaluer sa complexité.
- Q.9) Écrire une fonction récursive `fibo_rec2(n)` qui prend en argument un entier naturel n et renvoie la valeur de F_n en sauvegardant les termes de la suite déjà calculés dans une liste.

B) Versions récursives d'algorithmes classiques

```
def test(L):
    if len(L) <= 1:
        return True
    else:
        if L[0] > L[1]:
            return False
        else:
            return test(L[1:])
```

→ Q.10) Que fait la fonction `test` ci-dessus?

→ Q.11) En s'inspirant de la fonction `test`, écrire une fonction récursive `max_liste(L)` qui renvoie la valeur maximale des termes d'une liste fournie en paramètre.

Pour éviter toutes les copies de listes, on propose l'alternative suivante à la fonction `test` :

```
def test(L):
    def test_aux(L, i):
        """Renvoie True si L[i:] est triée et False sinon"""
        if i >= len(L) - 1:
            return True
        else:
            if L[i] > L[i + 1]:
                return False
            else:
                return test_aux(L, i + 1)
    return test_aux(L, 0)
```

- Q.12) Réécrire la fonction `max_liste(L)` en utilisant une fonction auxiliaire.
- Q.13) Écrire une fonction récursive `recherche(L, e)` qui renvoie `True` si `e` est dans `L` et `False` sinon.
- Q.14) Écrire une fonction récursive `recherche_dicho_aux(L, g, d, e)` qui prend une liste triée `L` et deux indices $g \leq d \leq \text{len}(L)$ et un objet `e` et renvoie `True` si `e` est dans `L[g:d]` et `False` sinon.
- Q.15) En déduire une fonction `recherche_dicho(L, e)` qui implémente la recherche dichotomique de `e` dans `L` de façon récursive.
- Q.16) Écrire une fonction récursive qui permet de trouver l'indice de la première occurrence d'un élément `e` dans `L` en utilisant la dichotomie.

C) Exponentiation rapide

On considère la suite (u_n) définie par :

$$\begin{cases} u_0 = a \\ \forall n \in \mathbb{N}, u_{n+1} = au_n \end{cases}$$

- Q.17) Que vaut u_n en fonction de a et de n ?
- Q.18) Programmer une fonction récursive `u(a, n)` qui renvoie la valeur de u_n .
- Q.19) Évaluer le nombre de multiplications effectuées pour calculer u_{128} .

On note C_n le coût de l'appel de la fonction `u(a, n)`.

- Q.20) Déterminer une relation de récurrence entre C_n et C_{n-1} pour $n \geq 1$.
- Q.21) En déduire la complexité de la fonction `u`.

On considère l'algorithme récursif suivant qui permet de calculer a^n :

- si n est pair, on calcule $a^{n/2} * a^{n/2}$;
- si n est impair, on calcule $a^{(n-1)/2} * a^{(n-1)/2} * a$.

Par exemple, pour $n = 22$:

- pour calculer a^{22} , on calcule $a^{11} * a^{11}$;
- pour calculer a^{11} , on calcule $a^5 * a^5 * a$;
- pour calculer a^5 , on calcule $a^2 * a^2 * a$;
- pour calculer a^2 , on calcule $a * a$.

Cet algorithme s'appelle **exponentiation rapide** : il permet de calculer plus rapidement une puissance en utilisant le principe de dichotomie.

- Q.22) Programmer une fonction récursive `puissanceR(a, n)` implémentant cet algorithme.
- Q.23) Évaluer le nombre de multiplications nécessaires pour calculer a^{128} .

On note D_n le coût de l'appel de `puissanceR(a, n)`.

- Q.24) Justifier qu'il existe un entier naturel M tel que $D_n \leq D_{\lfloor n/2 \rfloor} + M$ pour tout $n \geq 1$.
- Q.25) En déduire que $D_n = O(\log_2(n))$.

La terminaison et la correction d'un algorithme récursif se démontre généralement par récurrence ! Considérons la propriété $\mathcal{P}(n)$: l'instruction `puissanceR(a, n)` termine et renvoie la valeur de a^n quel que soit a .

- Q.26) Montrer par récurrence que $\mathcal{P}(n)$ est vraie pour tout $n \geq 0$.

D) Sous-listes d'une liste

On souhaite écrire une fonction qui étant donnée une liste renvoie une liste composée de toutes les sous-listes de celle-ci. Par exemple, pour `L = [1, 2]`, la fonction devrait renvoyer `[[], [1], [2], [1, 2]]`, à l'ordre près. Nous allons procéder récursivement.

- Q.27) Quelles sont les sous-listes de la liste vide ?
- Q.28) Quelle relation peut-on trouver entre les sous-listes de `L` et les sous-listes de `L[:len(L)-1]` ?

Rappelons la différence entre deux instructions : si `L` est une liste et `x` un élément :

- `M = L.append(x)` ajoute `x` à la fin de `L`, `M` est de type `None` ;
- `M = L + [x]` crée une nouvelle liste constituée des éléments de `L` et de `x` à la fin, `M` renvoie vers cette liste.

- Q.29) Écrire une fonction récursive `souslistes(L)` qui renvoie la liste des sous-listes de `L`.
- Q.30) Démontrer par récurrence sur la taille de la liste `L` que l'algorithme précédent termine et est correct.

E) Flocon de Koch

On souhaite tracer le flocon de Koch de profondeur n entre les points A et B qui est défini de la façon suivante :

- le flocon de profondeur 1 est le segment $[AB]$;
- pour obtenir le flocon de profondeur n on découpe $[AB]$ en trois segments de mêmes longueurs $[AC]$, $[CD]$ et $[DB]$ et on prend E tel que CDE soit équilatéral direct. On trace alors le flocon de profondeur $n-1$ entre A et C , entre C et E , entre E et D et entre D et B .

Voici les flocons pour $n = 1$, $n = 2$ et $n = 3$ entre les points $(0, 0)$ et $(4, 0)$.

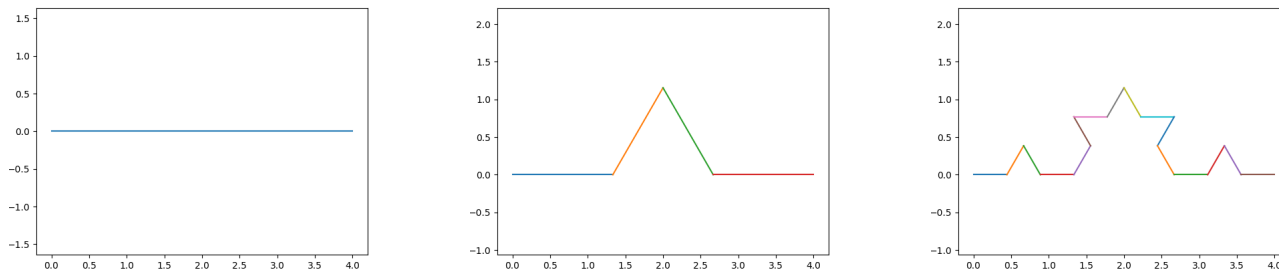


FIGURE 1 – Flocon de Koch pour $n = 1$, $n = 2$ et $n = 3$

Pour réaliser ce tracé, on va utiliser les nombres complexes. Pour utiliser le nombre $a + ib$ sous python, on utilise la syntaxe : `a + 1j*b`. Ainsi, `1j` joue le rôle de i .

Pour récupérer la partie réelle/imaginaire d'un complexe z , on utilise les fonctions `real` et `imag` du module `numpy` : par exemple `np.real(1+1j)` renvoie 1 et `np.imag(1+1j*3)` renvoie 3.

→ Q.31) Importer le module `numpy` sous l'alias `np` et le module `matplotlib.pyplot` sous l'alias `plt`.

Pour tracer un segment entre deux complexes z_1 et z_2 , on récupère leurs parties réelles et leurs parties imaginaires, pour les passer à la fonction `plot` du module `matplotlib.pyplot` : `plt.plot([np.real(z1), np.real(z2)], [np.imag(z1), np.imag(z2)])`.

On rappelle enfin que l'affixe du vecteur joignant les points $A(z_A)$ et $B(z_B)$ est $z_B - z_A$ et l'affixe du vecteur obtenu à partir de la rotation de $u(z_u)$ d'angle $\frac{\pi}{3}$ est $z_u \times e^{i\frac{\pi}{3}}$.

→ Q.32) Écrire une fonction récursive `Koch(n, A, B)` qui trace le flocon de profondeur n entre les points A et B (qui sont des complexes).

→ Q.33) Tracer le flocon (on prendra des petites valeurs de n d'abord).

On peut ajouter l'instruction `plt.axis('equal')` qui met la même échelle sur les deux axes de coordonnées, et on n'oubliera pas l'instruction `plt.show()` pour afficher le tracé.

→ Q.34) On note C_n le coût d'exécution de l'instruction `Koch(n, A, B)`. Déterminer une relation de récurrence entre C_n et C_{n-1} .

→ Q.35) Évaluer la complexité de la fonction.

F) Bilan

- ▷ Une fonction récursive est une fonction qui s'appelle elle-même :
 - elle a une (ou des) condition d'arrêt;
 - le (ou les) paramètres changent à chaque appel récursif pour se rapprocher de la condition d'arrêt.
- ▷ Attention aux dépassement de pile.
- ▷ Pour justifier la terminaison et la correction d'une fonction récursive, on procède en général par récurrence.
- ▷ Pour évaluer la complexité d'une fonction récursive, on peut introduire une suite récurrente.