

Feuille d'exercices 5

Programmation récursive

Exercice 1. Permutations.

Obtenir les permutations d'une séquence d'objets est une opération qui a de nombreuses applications : mathématiques, énumération de configurations, aide au jeu du scrabble, etc. Elles s'obtiennent naturellement par récursivité.

Permutations d'un mot

On se donne une chaîne de caractères S . On souhaiterait obtenir la liste de toutes les permutations de la chaîne. Par exemple pour la chaîne '123' on souhaiterait obtenir la liste : ['123', '132', '213', '231', '312', '321'].

Pour l'obtenir par récursivité, on remarque que :

- si S est de longueur ≤ 1 , le résultat est $[S]$;
- si S est de longueur > 1 , le résultat est la liste constituée de toutes les chaînes ep pour e un caractère quelconque de S , et p une chaîne permutation quelconque de la chaîne S privée de e .

Ainsi écrit une fonction récursive où :

- le cas terminal est pour une chaîne S de longueur ≤ 1 : on renvoie la liste ne contenant que S ;
- pour une chaîne de longueur > 1 : on parcourt les caractères e de S au sein d'une boucle `for` ; soit i leur indice. La chaîne $Se = S[:i]+S[i+1:]$ est la chaîne obtenue de S après en avoir retiré e . L'appel récursif sur Se renvoie la liste des permutations de Se ; pour chacune, on la concatène à e pour obtenir une nouvelle permutation de s qu'on insère dans la liste à renvoyer.

Écrire le code de la fonction récursive `permutations(S)`.

Exercice 2. Algorithme de Horner.

Soit $P \in \mathbb{K}[X]$ un polynôme de degré n et $a \in \mathbb{K}$, le calcul naïf de $P(a)$:

$$P(a) = \sum_{k=0}^n b_k \times a^k = b_0 + b_1 \times a + b_2 \times a^2 + \dots + b_n \times a^n$$

nécessite n additions et $1 + 2 + \dots + n = \frac{n(n+1)}{2}$ multiplications, soit $\frac{n(n+3)}{2}$ opérations, en quantité quadratique.

L'algorithme de Horner pour le calcul de $P(a)$ repose sur l'égalité :

$$P(a) = b_0 + a \times (b_1 + a \times (b_2 + a \times (\dots \times (b_{n-1} + a \times b_n))))$$

soit en tout n additions et n multiplications, $2n$ opérations, en quantité linéaire.

1. Écrire une fonction récursive qui prend en paramètres la liste des coefficients du polynôme P par degré croissant (par exemple `[-1, 0, 1]` pour $X^2 - 1$) et le nombre a , et qui renvoie la valeur $P(a)$ calculée par l'algorithme de Horner.
2. Écrire une version itérative de l'algorithme.
3. Quelle est leur complexité ?

Exercice 3. Calcul approché de la constante de Prouhet-Thue-Morse.

La suite de Prouhet-Thue-Morse est une suite infinie $(t_n)_{n \in \mathbb{N}}$ de 0 et de 1 construite de la façon suivante : on part du mot : 0 et on applique indéfiniment sur chaque caractère du mot les transformations :

$$0 \rightarrow 01 \quad 1 \rightarrow 10$$

Les premières itérations donnent :

0
01
0110
01101001
0110100110010110

Comme on le voit, un mot obtenu après itération est un préfixe de celui obtenu à l'itération suivante. Cela permet de définir le mot infini obtenu ; c'est la suite $(t_n)_n$ de Prouhet-Thue-Morse. Vu comme l'écriture binaire après la virgule d'un nombre dans $[0, 1[$, c'est-à-dire comme le nombre :

$$\sum_{n=0}^{\infty} \frac{t_n}{2^{n+1}} \approx 0,412454\dots$$

c'est un nombre transcendant (ce n'est la racine d'aucun polynôme à coefficients rationnels, comme e et π). On l'appelle la constante de Prouhet-Thue-Morse.

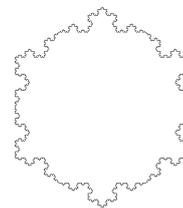
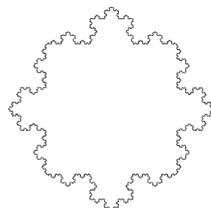
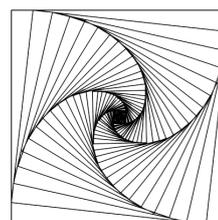
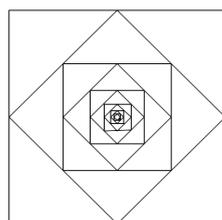
Écrire une fonction prenant en paramètre un entier n et qui renvoie une valeur approchée à 2^{-n} près par défaut de la constante de Prouhet-Thue-Morse.

Exercice 4. Tracés de fractales avec turtle.

Dans cet exercice, les tracés se feront avec le module `turtle` dont le fonctionnement est décrit dans le cours.

1. Courbes de Von Koch

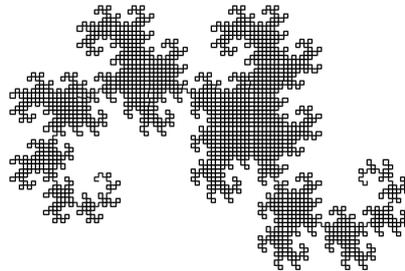
- a) Écrire une fonction qui permet le tracé d'une courbe de Von Koch.
- b) L'utiliser pour obtenir le tracé des 2 courbes fractales :

**2. Spirales carrées**

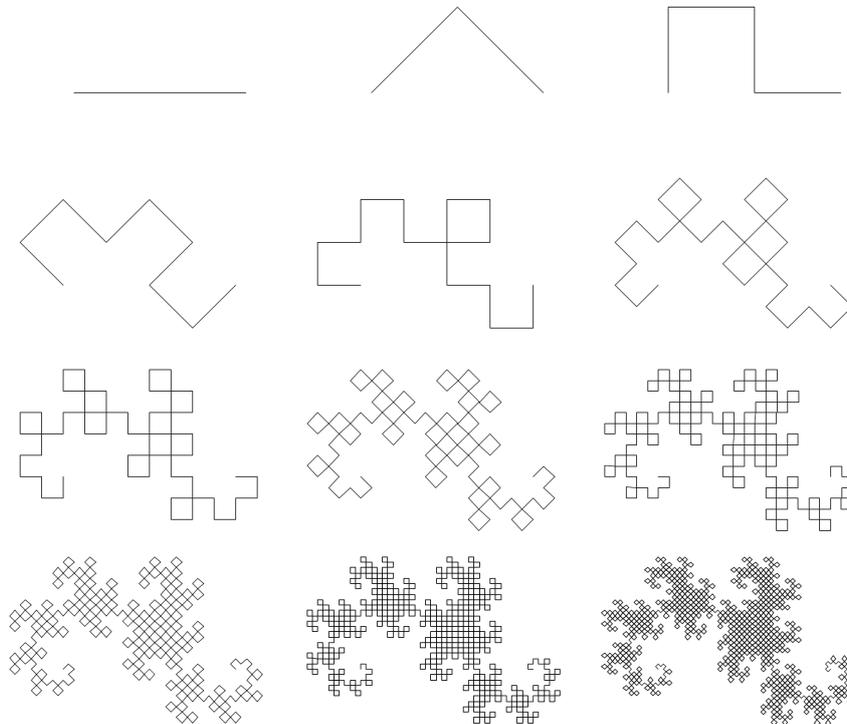
- a) Écrire une fonction récursive qui réalise le graphique ci-dessus, à gauche, de carrés imbriqués. Elle prendra en paramètres un entier n , c'est le nombre de carrés, et la longueur L d'un côté du plus grand carré. (On peut prendre pour le tracé $L = 200$.)

- b) Écrire une fonction récursive qui permet de réaliser le graphique ci-dessus, à droite, de carrés spiralés. Elle prendra en paramètres un entier n , c'est le nombre de carrés, la longueur L d'un côté du plus grand carré, et la distance l séparant un sommet du plus grand carré à un sommet du carré suivant. (Dans ce graphique on a pris $l = L/10$.)
- c) Déterminer leurs complexités en fonction de n .

3. Fractale du dragon



Elle s'obtient en partant d'un segment que l'on remplace par les 2 autres côtés d'un triangle rectangle isocèle. A chaque itération on parcourt les segments de la courbe en appliquant cette transformation en alternant les 2 possibilités dessus/dessous.



- a) Écrire 2 fonctions récursives, `dragon_gauche(L,n)`, `dragon_droite(L,n)`. Le fonctionnement de `dragon_gauche(L,n)` consistera à :
- si $n = 0$: tracer une trait de longueur L ;
 - sinon :
 - tourner à gauche de 45° ,
 - appeler `dragon_gauche(L/√2,n-1)`,
 - tourner à droite de 90° ,
 - appeler `dragon_droite(L/√2,n-1)`,
 - tourner à gauche de 45° .

Le fonctionnement de `dragon_droite(L,n)` est semblable à ceci près que les rotations se font dans le sens contraire.

Pour le tracé, on appellera l'une de ces 2 fonctions avec les paramètres adéquats.

La courbe construite (à l'infini) a des propriétés remarquables : sa dimension fractale (ou dimension de Hausdorff) est 2, ce qui implique que la courbe couvre le plan, elle est d'aire non nulle. De plus on peut paver le plan avec des copies de fractales du dragon, qui plus est de multiples façons.

- b) Tracer quatre courbes du dragon de couleurs différentes, chacune issue de l'origine après rotation de 90° .
- c) Même démarche mais avec deux courbes du dragon, après rotation de 180° et translation de $(L, 0)$.