T.P. d'informatique n° 4

Récursivité

Solution

CORRIGE

Exercice 1. Coefficients binomiaux

On rappelle que les coefficients binomiaux vérifient : $\binom{n}{p} = 0$ si $n < p$ ou $p < 0$, $\binom{n}{0} = \binom{n}{n} = 1$ si $n \in \mathbb{N}$ et, pour tout couple d'entiers (n, p) avec $n > 0$, on a $\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}$

1. Ecrire une fonction récursive **binom** d'arguments deux entiers n et p et retournant le coefficient binomial $\binom{n}{p} = 0$
2. Estimer la complexité de cet algorithme.

Solution

```
def binom(n,p):
  if n<p or p<0:
    return(0)
  elif n == p or p == 0:
    return(1)
  else:
    return(binom(n-1,p) + binom(n-1,p-1))
```

Exercice 2. Tours de Hanoï

Le jeu **des tours de Hanoï** est constitué de N pièces (des disques percés) que l'on peut déplacer sur trois tiges numérotées 0, 1 et 2 ;
les N pièces sont toutes de tailles distinctes numérotées de 1 à N (dans l'ordre de taille croissante, i.e. la pièce n° 1 est la plus petite, la n° N la plus grande).

• Dans l'état initial, les N pièces sont sur la tige n° 0 (dans l'ordre de taille décroissant de bas en haut).

• Le jeu consiste à déplacer les pièces de la tige n° 0 à la tige n° 2.
• Le joueur doit obéir aux deux règles suivantes :
i. les pièces ne peuvent être déplacées qu'une par une d'une tige à une autre ;
ii. une pièce ne peut être posée que sur une pièce de plus grande taille (ou éventuellement sur le socle !)

Quest. 1. Notons $H(N)$ le nombre de coups minimal pour réussir le jeu de Hanoi à N pièces :
a. déterminer $H(1)$;
b. trouver une relation de récurrence liant $H(N)$ et $H(N+1)$;
c. en déduire la valeur de $H(N)$, pour tout N entier.

Quest. 2. On décide de représenter une position du jeu par une liste nommée **tours** elle-même composée de 3 piles.

Ainsi en position initiale, par exemple pour $N = 5$, **tours=[[5,4,3,2,1],[],[]]**.
a. Résoudre le problème à la main pour $N = 2$, $N = 3$. Est-ce en adéquation avec le $H(N)$ correspondant ?
b. Ecrire une fonction **initialise** qui prend en argument l'entier N et qui crée la variable globale **tours** dans la position initiale.
c. Ecrire une fonction **verifie** qui prend en argument une pile et qui rend **True** si les éléments de la pile sont dans l'ordre décroissant et **False** sinon.
d. Ecrire une fonction **deplace** qui prend en argument deux piles dites **depart** et **arrivee** et qui déplace la pièce du haut de la pile **depart** vers la pile **arrivee**. Elle enverra un message d'erreur à l'aide de la fonction **print** si la pile d'arrivée est désordonnée (à l'aide de la fonction **verifie**).
e. Ecrire une fonction récursive nommée **Hanoi**, recevant trois arguments : n, i et j, permettant de déplacer n pièces de la tige n^o i à la tige n^o j.

Solution

```
def initialise(N):
    global tours
    tours=[[n for n in range(N,0,-1)
        ],[],[]]
```

```
def verifie(pile):
    for i in range(len(pile)-1):
        if pile[i]<pile[i+1]:
            return False
    return True
```

```
def deplace(depart,arrivee):
    if len(depart)>0:
        arrivee.append(depart.pop())
        print(tours)
        if not(verifie(arrivee)):
            print('erreur')
```

```
def Hanoi(n,i,j):
    if n>=1:
        k=3-i-j #num tige intermediaire
        Hanoi(n-1,i,k)
        deplace(tours[i],tours[j])
        Hanoi(n-1,k,j)
```

```
initialise(5)
Hanoi(5,0,2)
print(tours)
```

Exercice 3. Algorithme de Horner

On considère un polynôme P dont les coefficients sont stockés dans un tableau a. $P = \sum_{k=0}^{n} a_k X^k$. On veut écrire une fonction recevant comme paramètres le tableau a et le flottant (ou entier) x et renvoyant $\sum_{k=0}^{n} a_k x^k$

1. On se donne une valeur réelle x et un tableau a contenant les coefficients du polynôme P. Evaluer le nombre d'additions et de multiplications de réels effectués en cas de programmation "naïve"
2. Version itérative Justifier que l'algorithme de Horner est correct et l'implémenter.
   On pose $y_0 = a_n$ et, pour i allant de 1 à n, $y_i = x\ y_{i-1} + a_{n-i}$ alors y_n vaut $P(x)$.
   Déterminer le nombre d'additions et de multiplications lors de l'appel de cette fonction
3. Version récursive. On transforme un peu l'écriture. En notant $P_j(x) = \sum_{k=j}^{n} a_k x^{k-j}$, on veut calculer $P_0(x)$.
   On a : $P_n(x) = a_n$ et, si $j < n$, $P_j(x) = a_j + x.P_{j+1}(x)$.
   Traduire cet algorithme en un programme récursif.

Solution

```
def HornerI(a,x):
    n = len(a) - 1
    y = a[n]
    for i in range(1,n+1):
        y = x*y+a[n-i]
    return(y)
```

```
def HornerR(a,x):
    if len(a) == 1:
        return(a[0])
    else :
        return(a[0] + x *
            HornerR(a[1:],x))
```

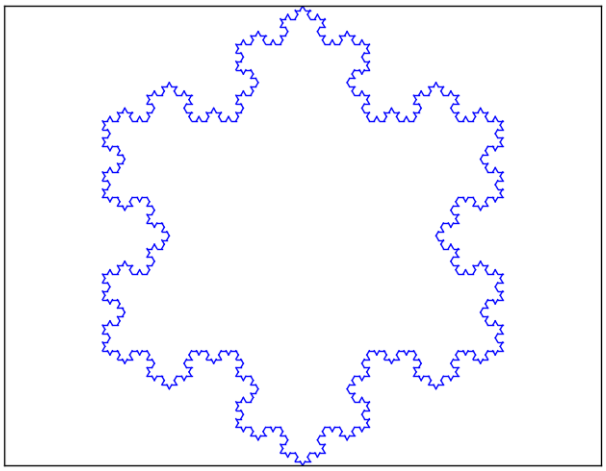
Exercice 4. Le flocon de Von Koch

Le flocon de Von Koch est une célèbre figure fractale obtenue de la façon suivante : à tout segment $[A, B]$ orienté de A vers B, on associe la ligne brisée $AA_1A_2A_3B$ où A_1 et A_3 partagent le segment initial en trois parts égales et où le triangle $A_1A_2A_3$ est équilatéral.

On applique alors cette transformation à un segment initial, puis on l'applique à chacun des 4 segments obtenus, puis à chacun des 16 segments obtenus et ainsi de suite... Ce qui donne la figure ci-dessous.

Ecrire une fonction récursive recevant pour paramètres deux points a et b et traçant la ligne de Von Koch associée. Pour faire des calculs sur les coordonnées des points, il est conseillé de stocker les coordonn{ees $[x, y]$ dans un tableau `numpy.array`

En rassemblant les trois lignes de Von Koch à partir des trois segments d'un triangle équilatéral, on obtient ce type de figure :



Solution

```
import matplotlib.pyplot as plt
import numpy as np

k = 0.5/np.sqrt(3)
xm , seuil = 100 , 3

def flocon(a,b):
    d = np.sqrt((b[0]-a[0])**2 + (b
        [1]-a[1])**2)
    if d < seuil:
        plt.plot([a[0],b[0]],[a[1],b
            [1]], 'b-')
    else :
        a1 = (2*a+b)/3
        a3= (a+2*b)/3
        a2= (a+b)/2 + k*np.array([a
            [1]-b[1],b[0]-a[0]])
        flocon(a,a1)
        flocon(a1,a2)
        flocon(a2,a3)
        flocon(a3,b)
```

```
a =np.array([0,0])
b = np.array([xm,0])
c = np.array([xm/2,-xm*np.sqrt(3)
    /2])
ax=plt.gca()
ax.set_xticks([]); ax.set_yticks([])
plt.axis('equal')
flocon(a,b)
flocon(b,c)
flocon(c,a)
plt.show()
```