Informatique

5
Fonctions récursives

TD 5-5
Exercices divers

Exercice 1: Eléments distincts

Soit une liste L non triée.

Dans cet exercice, on veillera à ce que la complexité à chaque étape soit en :

- 0(1) dans le meilleur des cas : Tous les éléments de L sont identiques
- 0(n) dans le pire des cas : Tous les éléments de L sont différents

Ce qui assurera pour la fonction complète une complexité en 0(n) dans le meilleur des cas, et en $O(n^2)$ dans le pire des cas.

Question 1: Ecrire une fonction récursive elements_distincts_1(L) renvoyant les éléments distincts d'une liste L sans modifier L

Question 2: Ecrire une fonction récursive elements_distincts_2(L,M) renvoyant les éléments distincts d'une liste L sans modifier L, où M sera une liste vide lors du premier appel qui servira à stocker les éléments distincts en cours d'exécution

Remarque : on pourra tenter d'écrire deux versions de cette fonction, l'une où M est vide au cas de base, l'autre ou M est pleine au cas de base.

Exercice 2: Autour des chiffres d'un entier

Question 1: Si x est un entier, que renvoient les instructions x%10 et x//10 ?

Question 2: Ecrire une fonction récursive nbre_chiffre(n) qui renvoie le nombre de chiffres de l'entier naturel n

```
>>> nbre_chiffre(4528)
4
```

Question 3: Ecrire une fonction récursive liste_chiffre(n) qui renvoie la liste des chiffres d'un entier naturel n

```
>>> liste_chiffre(4528)
[4, 5, 2, 8]
```

Exercice 3: Somme des chiffres d'un entier

On souhaite réaliser de manière **récursive** la somme des chiffres d'un entier n, par exemple :

- 123 donne 6
- 76 donne 13

On propose différentes méthodes :

- Méthode 1 : Réalisation d'opération sur n. Attention, exécutez 12/10-int(12/10))*10 dans la console pour identifier un risque dans la programmation de cette méthode
- Méthode 2 : Transformation de n en chaine de caractères et utilisation de celle-ci

Question 1: Ecrire une fonction récursive Somme_Ch_1(n) basée sur la méthode 1

Question 2: Ecrire une fonction récursive Somme_Ch_2(n) basée sur la méthode 2

Pour rappel, un nombre est divisible par 9 si la somme de ses chiffres l'est. Si cette somme est supérieure à 9, il est divisible par 9 si la somme des chiffres du résultat est divisible par 9 etc.

Question 3: Ecrire une fonction Div9(n) utilisant une des fonctions Somme_Chiffres et renvoyant le booléen réponse de la question « n est divisible par 9 ? »

Exercice 4: Moyenne arithmético géométrique

Soient les deux suites définies ainsi :

$$\begin{cases} \begin{cases} u_0 = a \\ v_0 = b \end{cases} \\ \forall n \in \mathbb{N}, \begin{cases} u_{n+1} = \frac{1}{2}(u_n + v_n) \\ v_{n+1} = \sqrt{u_n v_n} \end{cases} \end{cases}$$

Question 1: Ecrire deux fonctions suite_u(a,b,n) et suite_v(a,b,n) contenant des appels récursifs réciproques qui, recevant deux réels positifs a et b, retournent respectivement les termes u_n et v_n des suites définies par :

Vérifiez :

```
>>> suite_u(2,3,5)
2.474680436236304

>>> suite_v(2,3,5)
2.4746804362363046
```

Exercice 5: Insertion

On souhaite insérer un élément x à la bonne place dans une liste L (vide ou non) supposée triée.
Exemples :

- L=[] et x=1 → [1]
- L=[1,3,5,7,8] et x=0 → [0,1,3,5,7,8]
- L=[1,3,5,7,8] et x=1 → [1,1,3,5,7,8]
- L=[1,3,5,7,8] et x=4 → [1,3,4,5,7,8]
- L=[1,3,5,7,8] et x=5 → [1,3,5,5,7,8]
- L=[1,3,5,7,8] et x=8 → [1,3,5,7,8,8]
- L=[1,3,5,7,8] et x=9 → [1,3,5,7,8,9]

Question 1: Créer une fonction Insertion_rec(L,x) insérant x au bon endroit dans L

Remarques :

- Bien rappeler L entièrement afin que x soit inséré dans la liste L, et non une sous liste de L du genre L[…] qui serait traitée comme une nouvelle liste
- On pourra utiliser une sous fonction rec(L,x,i) qui sera appelée pour un i bien choisi dans la fonction Insertion_rec
- Votre fonction doit fonctionner pour tous les cas proposés ci-dessus, en particulier si L est vide

Exercice 6: Insertion dichotomique

On souhaite insérer un élément x à la bonne place dans une liste L (vide ou non) supposée triée en procédant par dichotomie.

Exemples :

- L=[] et x=1 → [1]
- L=[1,3,5,7,8] et x=0 → [0,1,3,5,7,8]
- L=[1,3,5,7,8] et x=1 → [1,1,3,5,7,8]
- L=[1,3,5,7,8] et x=4 → [1,3,4,5,7,8]
- L=[1,3,5,7,8] et x=5 → [1,3,5,5,7,8]
- L=[1,3,5,7,8] et x=8 → [1,3,5,7,8,8]
- L=[1,3,5,7,8] et x=9 → [1,3,5,7,8,9]

On propose trois méthodes dichotomiques :

- Méthode non récursive : A partir des indices ig (premier indice) et id(dernier indice) des termes de la liste L, une boucle permet de diviser par 2 (à 1 près) cet intervalle jusqu'à ce qu'un seul indice persiste. La fonction ajoute alors x avec L.insert(i,x) du bon côté du terme à cet indice.
- Méthode récursive 1 : A chaque appel, la liste est découpée en deux parties de tailles identiques (à 1 près). La fonction s'auto appelle alors sur la moitié dans laquelle doit être inséré x afin de l'y placer correctement, puis renvoie la liste recomposée (dans le bon ordre) de la moitié non traitée et de la moitié contenant x. A la fin, L n'est pas modifiée, une nouvelle liste est renvoyée.
- Méthode récursive 2 : A chaque appel, la fonction calcule l'indice milieu im (à 1 près) de la zone d'étude entre i inclus et j exclus, et se rappelle sur l'intervalle dans lequel placer x. Au cas de base, x est inséré dans L au bon endroit avec L.insert(i,x). L est donc modifiée, aucun renvoie n'est attendu.

Remarque pour ces trois fonctions : je vous recommande fortement d'exclure le milieu des sous listes !

Question 1: Créer une fonction Insertion_Dicho_(L,x) basée sur la méthode non récursive

Question 2: Créer une fonction Insertion_Dicho_rec_1(L,x) basée sur la méthode récursive 1

Remarque : notez que cette fonction crée une nouvelle liste de taille n à chaque étape, ce qui consomme de la mémoire avec les différents appels successifs. La méthode 2 va permettre d'éviter cela.

Question 3: Créer une fonction Insertion_Dicho_rec_2(L,x) basée sur la méthode récursive 2

Aide : je vous suggère de définir dans cette fonction une fonction rec(L,x,i,j) et de l'y appeler une fois avec rec(L,x,0,len(L)) :

```
def Insertion_Dicho_rec_2(L,x):
    def rec(L,x,i,j):
        # A compléter
    rec(L,x,0,len(L))
```

Vérifiez bien vos deux fonctions sur tous les exemples proposés en début d'exercice, vous mettrez peut-être en évidence des boucles infinies…

Exercice 7: Listes quelconques

Reprenons un exemple du TP sur les piles avec une liste quelconque contenant des réels du type :

```
L = [1,[[2,3,[2]]],[[1,2],3]]
```

On souhaite dans un premier temps réaliser une fonction récursive qui réalise la somme de tous les termes de L. On propose deux méthodes :

- Si L est une liste non vide, on la découpe en deux et on réalise la somme des sommes (appels récursifs) de ces deux listes
- Si L est une liste, on effectue la somme des sommes (appels récursifs) de chacun de ses éléments

Aide : On pourra utiliser le test `type(t) == list`

Question 1: Créer une fonction Somme_Rec_1 basée sur la méthode 1
Question 2: Créer une fonction Somme_Rec_2 basée sur la méthode 2

On souhaite maintenant créer une fonction récursive qui met dans une seule liste tous les éléments trouvés dans L, par exemple [1,2,3,2,1,2,3] dans le cas présenté ci-dessus.

Vous reprendrez les principes des méthodes 1 et 2 que vous adapterez pour ces nouvelles fonctions.

Question 3: Créer une fonction Regroupe_1 basée sur la méthode 1
Question 4: Créer une fonction Regroupe_2 basée sur la méthode 2

Exercice 8: Distribution d'argent

Ayant beaucoup d'argent, je décide de le distribuer de la manière suivante :

- 1€ à la première personne que je croise
- 2€ à la seconde
- 3€ à la troisième…

Je me demande combien de personnes je vais pouvoir rendre heureuses et combien il me restera à la fin.

Question 1: Proposer un code récursif répondant à cette question

On décide de recommencer le jeu tant qu'il reste de l'argent.

Question 2: Proposer un code récursif permettant de prendre en compte ce choix en utilisant la fonction récursive précédente – Oui, il y a donc 2 récursivités

Remarque : il est possible de créer un code avec while utilisant la fonction de la question 1. Toutefois, je souhaite que la nouvelle fonction entière s'appelle par récursivité ;)

Exercice 9: Algorithme de Horner

On souhaite évaluer un polynôme $P(x)$ en un valeur a avec :

$$P(x) = b_n x^n + b_{n-1} x^{n-1} + \cdots + b_1 x + b_0 = \sum_{k=0}^{n} b_k x^k$$

On décrire le polynôme par la liste L de ses coefficients b_k, par exemple :

$$3x^2 + x - 1 \quad ; \quad L = [3,1,-1]$$

Question 1: Créer la fonction eval(L,x) qui évalue ce polynôme par la somme de tous ses termes $b_k x^k$

L'algorithme de Horner consiste à exploiter l'égalité suivante :

$$P(x) = b_n x^n + b_{n-1} x^{n-1} + \cdots + b_1 x + b_0$$
$$P(x) = \Big(\big((b_n x + b_{n-1})x + b_{n-2}\big)x + \cdots\Big)x + b_1\Big)x + b_0$$

Question 2: Créer la fonction Horner1(L,x) exploitant cette égalité de manière itérative

Question 3: Créer la fonction Horner2(L,x) exploitant cette égalité de manière récursive

Vous vérifierez que vos 3 fonctions donnent les mêmes résultats sur différents exemples.

Exercice 10: Chiffres romains

Un chiffre romain s'écrit sous la forme de lettres majuscules : MMXVII. Il est lu de gauche à droite.

I	V	X	L	C	D	M
1	5	10	50	100	500	1000

Lien pour transcrire des entiers en notation romaine et inversement

Question 1: Ecrire une fonction Correspondance(X) qui retourne l'entier correspondant à l'un des chiffres X écrit en chiffres romains sous forme de chaîne de caractères

Le principe de déchiffrage est le suivant :

- Si la valeur d'une des lettres romaines est inférieur à la suivante, alors on doit la soustraire de ce qui est à sa droite
- Si la valeur d'une des lettres romaines est supérieure à la suivante, alors on doit l'ajouter à tout ce qui est à droite
- Si le nombre romain à une seule lettre, on prend la correspondance du tableau ci-dessus
- Lorsque le même symbole est répété, on le somme autant de fois qu'il est présent, puis on l'ajoute ou on le retranche en fonction du premier symbole qui suit, selon les deux règles énoncées précédemment : XXXC = 70, CXXX = 130
- L'écriture des chiffres romains se fait de manière décroissante. Ainsi, on admettra qu'il n'est pas possible d'avoir à la suite, 3 termes croissants. Par exemple, 89 ne s'écrit pas IXC mais LXXXIX. D'ailleurs, les 2 convertisseurs en lien ici (Lien 1 – Lien 2) ne donnent pas le même résultat pour IXC… Car ce n'est pas prévu !

Bien que l'écriture romaine respecte certaines règles du type :

- On écrit 99 sous la forme XCIX plutôt que IC
- Un même symbole ne peut être écrit plus de 3 fois à la suite sauf le M
- …

Nous allons mettre en place un code qui décode tout nombre romain même s'il ne respecte pas les règles restrictives : qui peut le plus peut le moins !

Supposons dans un premier temps qu'un même symbole ne peut être répété plusieurs fois de suite.

Question 2: Ecrire une fonction récursive Chiffre_Romain(X) qui retourne l'entier correspondant à un nombre X écrit en chiffres romains sous forme de chaîne de caractères

Devant un ordinateur, vérifiez que : MXVI = 1016

Question 3: Proposez une légère modification à votre code pour qu'il permette de prendre en compte la répétition du même terme

Remarque : j'impose que XVVV donne 25, même si certes, 25 s'écrira XXV… Un élève a une fois programmé un code en question 1 qui prenait déjà en compte la répétition, l'idée étant de supposer qu'il suffit de comparer le nombre actuel à la somme de tous les nombres suivants pour savoir si on l'ajoute ou le retranche. Je ne souhaite pas cette méthode, afin de vous faire réfléchir un peu plus !

Devant un ordinateur, vérifiez que : XXXC = 70 & CXXX = 130

Exercice 11: Partitions croissantes d'un entier – V1

Soit un entier $n \geq 1$. Une partition croissante de n est un ensemble d'entiers $k_i \geq 1$ tels que :

$$k_1 \leq k_2 \ldots \leq k_n \quad ; \quad n = \sum_{i=1}^{n} k_i$$

Voici des répartitions croissantes :

1	[1]
2	[1,1] ; [2]
3	[1,1,1] ; [1,2] ; [3]
4	[1,1,1,1] ; [1,1,2] ; [1,3] ; [2,2] ; [4]

On notera que l'on obtient toutes les partitions (croissantes ou non) de n (avec des doublons possibles) en prenant toutes les partitions de $n-1$, puis en y intégrant un nouveau 1 (pas besoin de le mettre à chaque place possible pour éviter trop de doublons) ou en ajoutant 1 soit au premier terme, soit au second etc.

Question 1: Ecrire une fonction récursive Partition_1(n) qui renvoie la liste de toutes les partitions (listes) de l'entier n, croissantes ou non, avec doublons (le moins possible)

Remarque : vous êtes autorisés à utiliser p.copy()

Exemple :

```
>>> Partition_1(4)
[[1, 1, 1, 1], [2, 1, 1], [1, 2, 1], [1, 1, 2], [1, 2, 1], [
3, 1], [2, 2], [1, 1, 2], [2, 2], [1, 3], [1, 1, 2], [2, 2],
[1, 3], [1, 3], [4]]
```

Question 2: Améliorer la fonction 1 en proposant une fonction Partition_2(n) qui ne renverra que les partitions croissantes de n, sans doublons

Remarque : on pourra utiliser p.sort() pour trier une partition.

Exemple :

```
>>> Partition_2(4)
[[1, 1, 1, 1], [1, 1, 2], [2, 2], [1, 3], [4]]
```

Question 3: Compléter votre code afin de tracer sur un graphique le nombre de partitions croissantes d'un entier en fonction de l'entier de 1 à 20

Pour finir, on se pose la question suivante : comment répartir $n = 20$ élèves en $q = 6$ groupes de $m = 3$ élèves min et $M = 5$ élèves max

Question 4: Créer et utiliser la fonction Répartition(n,q,m,M) permettant de déterminer la réponse à la question en utilisant Partition_2 et la fonction max de python

Exercice 12: Partitions croissantes d'un entier – V2

On reprend la définition de la partition croissante d'un entier de l'exercice précédent.
On notera que l'on obtient toutes les partitions croissantes de n en q termes $k_i \leq m$ en prenant toutes les partitions croissantes en $q-1$ termes de

- $n - \mathbf{1}$ ne dépassant pas $\mathbf{1}$ auxquelles on ajoute $\mathbf{1}$ à droite
- $n - \mathbf{2}$ ne dépassant pas $\mathbf{2}$ auxquelles on ajoute $\mathbf{2}$ à droite
- …
- $n - \boldsymbol{A}$ ne dépassant pas $\boldsymbol{A}$ auxquelles on ajoute $\boldsymbol{A}$ à droite
  Sans que l'**ajout** $\boldsymbol{A}$ ne dépasse ni n, ni m.

Remarques :

- L'ajout **à droite** permet de garder une partition croissante
- Si on écrit « $n - \boldsymbol{A}$ ne dépassant pas $\boldsymbol{m}$ auxquelles on ajoute $\boldsymbol{A}$ à droite », on obtient des partitions croissantes ou non, comme dans l'exercice V1

Question 1: Ecrire une fonction récursive Partition_qm(n,q,m) qui renvoie la liste des partitions (listes) croissantes de n composées de q termes $k_i \leq m$

Exemples :

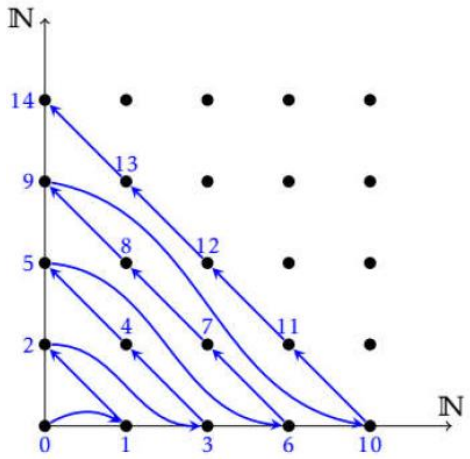
```
>>> Partition_qm(5,3,3)
[[1, 2, 2], [1, 1, 3]]

>>> Partition_qm(5,4,3)
[[1, 1, 1, 2]]

>>> Partition_qm(5,3,2)
[[1, 2, 2]]

>>> Partition_qm(5,3,1)
[]

>>> Partition_qm(5,3,5)
[[1, 2, 2], [1, 1, 3]]

>>> Partition_qm(5,2,5)
[[2, 3], [1, 4]]
```

De l'aide pour les cas de base ? Réfléchissez à ce qu'il est possible de renvoyer en un terme $(q = 1)$, selon n et m. Attention, dans tous les cas c'est une liste, et s'il y a quelque chose dedans, c'est une liste de listes.

Question 2: Ecrire une fonction Partition(n) utilisant la fonction Partition_qm(n,q,m) et renvoyant toutes les partitions croissantes de n

Exercice 13: Diagonale de Cantor

Plaçons les entiers naturels selon le schéma ci-dessous :



À un point de coordonnées (x, y), on associe alors l'entier correspondant.
Par exemple :

$$Cantor(0,0) = 0$$
$$Cantor(2,0) = 3$$
$$Cantor(2,2) = 12$$

On suppose $x \geq 0$ et $y \geq 0$
Le seul point supposé connu est l'origine telle que : Entier 0 et X=Y=0.
Les deux questions sont indépendantes !

Question 1: Proposer un algorithme récursif $Cantor(x, y)$ renvoyant l'entier associé aux coordonnées x et y

Remarque : cela revient à imaginer le jeu dans lequel des personnes portent un numéro qu'elles ne connaissent pas. Seule celle qui a le zéro le sait. Et chaque personne sait identifier où est celle qui a le numéro précédent. Demander le numéro d'une personne, elle vous dit ou est celle d'avant…

On suppose $n \in \mathbb{N}$

Question 2: Proposer un algorithme récursif $Cantor\_Inv(n)$ renvoyant les coordonnées associées à un entier quelconque n

Exercice 14: Complexité de Fibonacci

Reprenons l'exemple du cours. La suite de Fibonacci est définie par récurrence telle que :

$$\forall n \in \mathbb{N}, f_n = \begin{cases} 0 & si\ n = 0 \\ 1 & si\ n = 1 \\ f_{n-1} + f_{n-2} & sinon \end{cases}$$

$$[0,1,1,2,3,5,8,13,21 \dots]$$

On propose le code suivant :

```
def fibonacci(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return fibonacci(n-1)+fibonacci(n-2)
```

On appelle a_n le nombre de sommes réalisées lors de l'appel de la fonction Fibonacci à l'ordre n.

Question 1: Evaluez le nombre de sommes a_n de la fonction de Fibonacci pour n allant de 0 à 4
Question 2: Exprimer a_n en fonction de a_{n-1} et a_{n-2} pour $n \geq 2$
Question 3: Introduire une suite $b_n = a_n + k$ où k sera explicité telle que la suite b_n vérifie une relation de récurrence du type $ab_n + bb_{n-1} + cb_{n-2} = 0$ où a, b, c seront déterminés
Question 4: Déterminer l'expression de b_n en fonction de n
Question 5: Déterminer l'expression de a_n en fonction de n
Question 6: En déduire la complexité de la résolution étudiée

On reconnaîtra le nombre d'or :

$$\varphi = \frac{1 + \sqrt{5}}{2}$$

Question 7: Programmer l'algorithme de Fibonacci, ajouter un compteur qui compte le nombre de sommes réalisées et comparer le résultat à la suite a_n

Exercice 15: Complexités diverses

Soient les fonctions suivantes :

```python
def f1(n):
    if n==1:
        return 1
    else:
        return f1(n-1)
```

```python
def f2(n):
    if n==1:
        return 1
    else:
        for i in range(n):
            a = 1
        return f2(n-1)
```

```python
def f3(n):
    if n==1:
        return 1
    else:
        return f3(n-1)+ f3(n-1)
```

Question 1: Créer ces fonctions dans votre code
Question 2: Estimer leur complexité à l'aide du cours
Question 3: Créer la fonction Temps(f,n) qui renvoie le temps d'éxécution de la fonction f(n)
Question 4: Créer la fonction Etude_Tps(f,nmax) qui renvoie deux listes de tailles identiques Ln des n pour lesquels f(n) est appelée de 1 à nmax et LT des temps d'exécution ti(n) associés – On stoppera l'analyse si le temps d'exécution de f(n) dépasse par exemple 1 seconde
Question 5: Créer une fonction Affiche(fig,Lx,Ly,Leg) qui affiche sur la figure fig la courbe Ly en fonction de Lx avec la légende Leg
Question 6: Utiliser vos fonctions dans un code qui analyse et affiche sur une figure les temps d'exécution (et donc les complexités) des fonctions proposées en fonction de n
Pour rappel, nmax ne peut dépasser 986 (problème de taille de pile d'exécution).

On appelle $t_i(n)$ le temps d'exécution de la fonction $f_i(n)$ et on propose les fonctions suivantes :

$g_1(n) = \frac{t_1(n)}{n}$	$g_2(n) = \frac{t}{n^2}$	$g_3(n) = e^{\frac{\ln t}{n}}$

Question 7: Préciser vers quoi doivent tendre ces fonctions pour n>>1
Question 8: Créer ces fonctions dans votre code
Question 9: Mettre en place une fonction Etude(gi,Lni,Lti) qui renvoie la liste des valeurs de gi(n)
Question 10: Afficher sur 3 graphiques différents les fonctions gi(n) et en déduire si la complexité attendue est obtenue

Remarque : malheureusement, pour g3, il faudra augmenter le temps limite que nous avons fixé précédemment et attendre assez longtemps pour voir se dessiner l'arrivée d'une asymptote horizontale… Voilà ce que j'ai obtenu avec 3600 s :