05 - Récursivité

Avant de commencer, créer un dossier personnel appelé TD05, où l'on sauvegardera le fichier de travail et le fichier `tous_les_mots.txt`***, utile pour l'exercice 6.***

I D'itératif à récursif

I.1 Calculs de suites et de séries

Exercice 1 On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie de la façons suivante :

- $u_0 = 1$
- $\forall n \in \mathbb{N}, u_{n+1} = 3u_n^2 + 4$

Proposer un codage itératif `u_iter(n:int)->int` et un codage récursif `u_rec(n:int)->int` retournant la valeur de u_n.

Exercice 2 On considère la suite $(s_n)_{n\in\mathbb{N}}$ définie par :

$\forall n \in \mathbb{N}, s_n = \sum_{k=1}^{n} \frac{1}{k^3}$

Proposer un codage itératif `s_iter(n:int)->float` et un codage récursif `s_rec(n:int)->float` retournant la valeur de s_n.

Exercice 3 Méthode de Héron On peut montrer qu'une bonne approximation de $\sqrt{a}$ calculable par la suite $(r_n)_{n\in\mathbb{N}}$,

- $r_0 = \frac{a}{2}$
- $\forall n \in \mathbb{N}, r_{n+1} = \frac{r_n + \frac{a}{r_n}}{2}$

Proposer un codage itératif `r_iter(n:int,a:float)->float` et un codage récursif `r_rec(n:int,a:float)->float` retournant la valeur de r_n, et vérifier l'assertion du début de l'exercice sur quelques exemples bien choisis.

Exercice 4 *(Difficile)***Fibonacci, ou les premiers problèmes**

On définit la (célèbre) suite de Fibonacci par :

- si $n \in \{0, 1\}$, $f_n = n$
- $\forall n \in N^*, f_{n+2} = f_{n+1} + f_n$

4.1 Proposer une version itérative `f_iter(n:int)->int` calculant f_n (on pourra soit traîner deux variables, soit utiliser une liste pour stocker les valeurs des f_n au fur et à mesure. En déduire f_{50}.

4.2 Proposer une version récursive (plus simple à coder) `f_rec(n:int)->int`, et essayer d'obtenir $f_{50}\cdots$

I.2 Algorithmes classiques de liste

Exercice 5 Codage de la recherche du maximum

On pourra générer une liste d'entiers aléatoirement choisis via :

```
from random import randint
L=[randint(1,1000) for i in range(200)]
```

5.1 On cherche à coder une fonction retournant le maximum de L (pas sa position). Donner une version itérative `max_iter(L:list)` et une version récursive `max_rec{L:list}`, en remarquant que le maximum d'une liste est le maximum entre sa première valeur et le maximum de la liste restante.

5.2 Comment faire la même chose avec le minimum ?

5.3 On cherche à coder une fonction retournant la moyenne de L. Donner une version itérative `max_iter(L:list)` et une version récursive `max_rec{L:list}` (sans indication, attention ce n'est pas aussi simple que le maximum).

Exercice 6 Algorithmes de recherche

On considère une liste triée constituée de tous les mots admissibles au scrabble, contenue dans le fichier `tous_les_mots.txt`. On va tout d'abord ouvrir ce fichier afin de le convertir en une liste contenant tous ces mots, via le script suivant, à exécuter au début du programme :

```
with open('datas/touslesmots.txt','r',encoding='utf-8') as f:
    table=f.read().split(' ')
```

`table` contient alors tous les mots.

On va ensuite se choisir un mot au hasard dans ce dictionnaire. Pour ce faire, après avoir compilé le code précédent, taper en console :

- `from random import randint`
- `table[randint(0,len(table)-1)`

Copier/coller le résultat et le stocker dans une variable `mon_mot`. Le but de cet exercice est de trouver la position de ce mot dans la liste, en minimisant le temps de recherche.

6.1 *Algorithme naïf* Pour trouver la position d'un élément `e` dans une liste `L`, on peut parcourir la liste élément par élément et retourner sa position quand on tombe dessus.

a) Coder une fonction `recherche0(e:str,L:list)->int` qui effectue cela. On devra retourner `-1` si jamais `e` n'est pas dans la liste.

b) Placer une variable `compteur`, initialisée à 0 avant la boucle et s'incrémentant de 1 à chaque étape de la boucle, et en afficher la valeur juste avant le retour. Combien d'étapes à pris cette recherche pour `mon_mot` ? Demander également à vos camarades.

6.2 *Algorithme dichotomique itératif* On va profiter du fait que la liste soit triée [1] pour diminuer drastiquement le nombre d'étapes. L'algorithme est le suivant :

1. on initialise une variable `m` à 0 et une variable `M` à `len(L)-1`
2. tant que $M \geq m$:
   (a) on calcule `mil` tel que $mil = (M+m)//2$ (valeur moyenne)
   (b) si `e` est à la position `mil`, on retourne `mil`
   (c) si `e` est inférieur à `L[mil]`, alors il se trouve dans la première moitié de la liste : `M` prend la valeur de `mil-1` (car `e` n'est pas en `mil`)
   (d) sinon `e` se trouve dans la deuxième moitié de la liste : `m` prend la valeur de `mil+1`
3. si on sort du `while`, ceci signifie que l'on n'a pas trouvé `e` : on retourne -1

a) Coder une fonction `recherche_dicho(e:str,L:list)->int` qui reproduit cet algorithme, en plaçant comme à la question précédente une variable compteur qui s'incrémente de 1 à chaque boucle et qui s'affiche avant de retourner la position trouvée (ou -1).

b) Refaire la même recherche que précédemment : en combien d'étapes a-t-on trouvé `mon_mot` ? Demander également à vos camarades.

1. `Python` connaît l'ordre alphabétique, via les symboles usuels ==, $<$ et $>$.

6.3 *Recherche dichotomique avec fonction récursive auxiliaire*

a) Écrire une fonction récursive `aux_rec(e:str,L:list,m:int,M:int)->int`, qui recherche de façon récursive, par dichotomie, `e` entre les indices `m` et `M` de la liste `L`.

b) A quelles valeurs de `m` et `M` correspond une recherche dichotomique normale ? En déduire une fonction `dicho_rec(e:str,L:list)->int`, utilisant `aux_rec`, réalisant une recherche dichotomique de `e` dans `L`.

II Quelques algorithmes supplémentaires

Exercice 7 Algorithme d'Euclide

On appelle $PGCD$ le plus grand commun diviseur de deux nombres, c'est-à-dire le plus grand nombre n tq a et b soient tous deux divisibles par n. Par exemple le plus grand commun diviseur de 48 et 128 est 16. On rappelle la relation suivante : $PGCD(a,b) = PGCD(b,r)$ avec $r = a\%b$ et, $\forall a \in \mathbb{N}$, $PGCD(a,0) = a$

Coder l'algorithme d'Euclide en récursif.

Exercice 8 Conversion en base 2

Rappel : un nombre est écrit en base 2 si il ne comporte que des 0 ou des 1. Par exemple $10110_2 = 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 24$. L'algorithme pour trouver l'écriture d'un nombre n en base deux est le suivant :

1. on calcule le reste $r = n\%2$ et le quotient $q = n//2$
2. r sera alors le chiffre des unités, et les chiffres suivants procèdent du même algorithme avec q.

On a donc `base2(n)=base2(n//2)` concaténé au reste.

8.1 Expliquer pourquoi une représentation d'un nombre en base 2 se fait nécessairement avec une chaîne de caractères (sinon par exemple que vaudrait 101) ?

8.2 Coder l'algorithme récursif associé `base2(n:int)->str`.

Exercice 9 Inversion d'une chaîne

On cherche à coder de façon récursive l'inversion d'une chaine, par exemple `"truc"` donnerait `"curt"`.

9.1 Rappeler la nomenclature qui permet d'extraire la sous-chaîne ne contenant ni le premier, ni le dernier élément.

9.2 Coder en utilisant cette nomenclature une fonction `inversion_rec(s:str)->str` qui retourne la chaîne inverse de `s`. Attention il doit y avoir deux cas d'arrêt.

Exercice 10 Exponentiation rapide

On veut déterminer x^n sans utiliser l'opérateur `**`

10.1 Trouver une relation naïve entre x^n et x^{n-1}. En déduire une fonction récursive `expo_naif_rec(x,n)`. Estimer le nombre d'appels à la fonction que l'on a pour cet algorithme.

10.2 On remarque la propriété de récurrence suivante :

$$x^n = \begin{cases} (x^2)^{\frac{n}{2}} & \text{si } n \text{ pair} \\ x*(x^2)^{\frac{n-1}{2}} & \text{si } n \text{ impair} \end{cases}$$

En déduire une fonction `expo_rapide_rec(x,n)` permettant de calculer plus rapidement x^n, et en évaluer le nombre d'appels.

Exercice 11 Les tours de Hanoï

On cherche à coder le célèbre jeu de patience des tours de Hanoï, décrit figure 1. Le but est de passer de la configuration du haut à la configuration du bas en suivant les règles suivantes :

- on ne peut déplacer qu'un palet à la fois
- on ne peut pas mettre un palet plus grand sur un palet plus petit

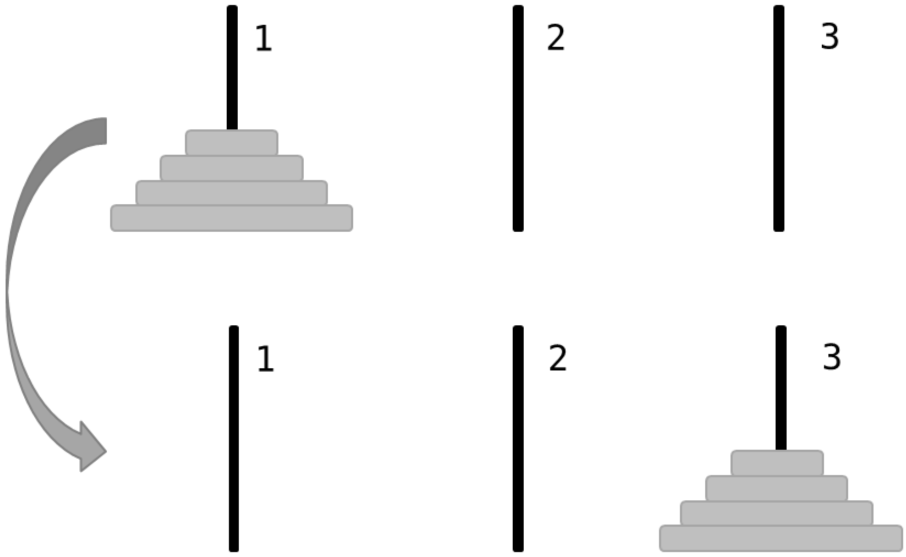

FIGURE 1 – Jeu des tours de Hanoï

On cherche à coder une fonction `hanoi(n,d:int=1,f:int=3,m:int=2)->None` qui indique les déplacements à faire pour résoudre le problème à `n` palets, à déplacer de `d` à `f` en utilisant la position intermédiaire `m`. Les `=` dans la fonction indiquent des valeurs par défaut, ce qui implique qu'appeler `hanoi(5)` revient à appeler `hanoi(5,1,3,2)`

11.1 Expliquer pourquoi `hanoi(1,d,f,m)` représente un déplacement et doit afficher être équivalente à `print("Dplacement de ",d,"vers",f)`.

11.2 "Montrer" par un schéma la relation de récurrence : $hanoi(n,d,f,m) = hanoi(n-1,d,m,f); hanoi(1,d,f,m); hanoi(n-1,m,f,d)$ où le ; signifie :"suivi de".

11.3 En déduire un codage de la fonction `hanoi` de façon récursive

Exercice 12 *MPSI** **Séparation d'une liste par parité**

On considère une liste L de nombres et on veut retourner une liste de listes, la première contenant les éléments pairs de L, la seconde les éléments impairs.

Coder une fonction récursive `parite_recursive(L:list)->list` qui réalise cela. On se demandera ce que doit retourner `parite_recursive([])`.