Sujet n°2

Algorithmique & Programmation
Problèmes algorithmiques récursifs

`Temps de réalisation: 3h`

L'objectif du TP n°2 est de :
- ▶ identifier les problèmes adaptés à une résolution récursive
- ▶ décliner la récurrence mathématique en programme récursif
- ▶ définir et implémenter les éléments d'une résolution récursive

Échelle de progression :

● « débutant·e » ● « confirmé·e » ● « avancé·e » ● « expert·e »

Ex.	01	02	03	04	05	06	07	08	09	10	11	12	13	14
	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Pour acquérir un niveau de compétence donné, il faut achever **tous** les exercices de ce niveau et des niveaux inférieurs. L'objectif de chacun·e est d'atteindre au moins le niveau *confirmé·e* (●) à l'issue du TP.

La programmation est une activité délicate et incertaine : pensez dès le début à tester votre code régulièrement, en procédant par petites touches successives, tel un diamantaire taillant sa gemme...

La documentation officielle de Python est toujours accessible ici :

https://docs.python.org/3/

Le contenu intégral des exercices du TP2 doit figurer dans un paquet `tp2` de votre projet PyCharm *cahier d'exercices*.

Table des matières

1 Premiers pas en récursivité

Tous les problèmes de cet énoncé doivent être résolus par un algorithme récursif, sauf mention contraire.

Exercice 1: Factorielle

Écrire une fonction récursive qui calcule la factorielle d'un entier naturel n. On rappelle que

$$n\,! = n \times (n-1)\,! \quad \text{si } n \geq 1 \qquad \text{et} \quad 0\,! = 1.$$

Exercice 2: PGCD

QUESTION 1. Écrire une fonction récursive qui retourne le plus grand commun diviseur (pgcd) de deux nombres entiers positifs par l'algorithme d'EUCLIDE :

$$\mathsf{pgcd}(a,b) = \begin{cases} a & \text{si } b = 0 \\ \mathsf{pgcd}(b, a \bmod b) & \text{sinon} \end{cases}$$

Remarque: *L'algorithme d'*EUCLIDE *est un cas emblématique de* récursion terminale. *La version itérative se décline donc aisément.*

En marge: *L'algorithme d'*EUCLIDE *et sa version étendue sont utilisés dans le cours de cryptographie (S7), et sont des outils fondamentaux en géométrie algorithmique.*

Exercice 3: L'incontournable Fibonacci

La suite de FIBONACCI notée F_n est définie par $F_0 = 0$, $F_1 = 1$, et $F_n = F_{n-1} + F_{n-2}$ pour $n \geq 2$.

QUESTION 1. Écrire une fonction récursive qui calcule le $n$ième terme de la suite de FIBONACCI.

QUESTION 2. Afficher les 100 premiers termes de la suite. Commenter.

QUESTION 3. Construire l'arbre d'appel de fonctions pour $n = 6$. Combien de fois est calculé F_3 pour obtenir le terme F_6 ?

QUESTION 4. Écrire une version itérative du calcul du $n$ième terme de la suite de FIBONACCI. Recalculer les 100 premiers termes. Commenter.

Exercice 4: Dissection de nombres

QUESTION 1. Construire une fonction

```
def nb_digits(n: int) -> int
```

prenant en paramètre un entier naturel n et retournant le nombre de chiffres de cet entier.

Remarque: *Le quotient de la division euclidienne d'un entier n par 10 donne le nombre de dizaines de cet entier. Par exemple, le quotient de la division euclidienne de* $n = 5478$ *par* 10 *vaut* 547.

QUESTION 2. Modifier cette fonction en ajoutant un deuxième paramètre `base` permettant de donner le nombre de chiffres de l'entier n –toujours exprimé en base 10– en base `base`.

QUESTION 3. Créer une fonction récursive

```
def convert(n: int, base: int) -> str
```

affichant un nombre n exprimé en base 10 en une base `base` fournie en paramètre (pour simplifier l'affichage, on considère `base` < 10).

Ex. : `convert(10,2)` *affiche* `1010`

QUESTION 4. Créer une fonction récursive `convert_mirror` affichant en sens inverse un nombre n exprimé en base 10 en une base `base` fournie en paramètre (pour simplifier l'affichage, on considère `base` < 10).

Ex. : `convert_mirror(10,2)` *affiche* `0101`

Exercice 5: Coefficients binomiaux

On connait la définition d'un coefficient binomial :

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

que l'on note aussi C_n^k comme le nombre de parties ou « combinaisons » de k éléments parmi n.

La formule de PASCAL définit la série des coefficients binomiaux à l'aide de la relation de récurrence :

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{pour } 0 < k < n$$

De plus, on pose :

$$\binom{n}{0} = \binom{n}{n} = 1$$

QUESTION 1. Écrire la fonction de calcul d'un coefficient binomial par la formule de PASCAL :

```
def binom(n: int, k: int) -> int
```

QUESTION 2. Retrouver $\binom{5}{4} = 5$, $\binom{10}{5} = 252$, $\binom{100}{1} = 1$ et calculer $\binom{100}{50}$. Commenter.

QUESTION 3. Proposer une version avec *mémoïsation*[1].

Astuce: *Dans le cas des coefficients binomiaux, la mémoïsation consiste à conserver en mémoire les termes déjà calculés du* triangle de PASCAL, *et les piocher dans la mémoire à chaque réutilisation.*

Remarque: *Par défaut, la mémoïsation est implémentée à l'aide d'un dictionnaire (*`dict` *en Python).*

QUESTION 4. Recalculer le terme $\binom{100}{50}$. Commenter.

1. Usage d'une mémoire tampon dans laquelle figurent les résultats de calculs coûteux qui devraient sinon être réalisés plusieurs fois.

2 Jeux et dessin

Exercice 6: Le robot cupide

Exercice proposé par O. Bodini de l'Institut Galilée, Univ. Paris-Nord.

Well-A le robot se trouve sur une case quelconque (x, y) d'un damier rectangulaire de taille $N \times M$. La case $(0, 0)$ se situe au Nord-Ouest. Il doit sortir par la case Sud-Est de coordonnées $(N-1, M-1)$. Il a le choix à chaque pas (un pas = une case) entre :

— descendre verticalement vers le Sud ;

— ou avancer horizontalement vers l'Est.

Il y a un sac de ₿itcoins sur chaque case, dont la valeur est lisible depuis la position initiale de Well-A. Le but de Well-A est de ramasser le plus de ₿itcoins possible durant son trajet.

QUESTION 1. On veut donc écrire la fonction :

```
def robot_cupide(damier: list[list[int]], x: int = 0, y: int = 0) -> int
```

qui, étant donné le damier $N \times M$ renseigné avec les gains potentiels dans chaque case, et les coordonnées x et y d'une case de départ, rend la quantité maximum de ₿itcoins (gain) que peut ramasser le robot en se déplaçant de (x, y) jusqu'à la sortie Sud-Est.

QUESTION 2. Faire évoluer la fonction `robot_cupide` de telle sorte qu'elle enregistre le trajet emprunté par Well-A, en plus de la somme récoltée.

Exercice 7: Tours de Hanoï

Le jeu des Tours de Hanoï est constitué de trois piquets verticaux, notés 1, 2 et 3 et de n disques superposés de tailles strictement décroissantes avec un trou au centre et enfilés autour du piquet 1 ; ces empilements forment les tours, comme le montre la Figure 1.

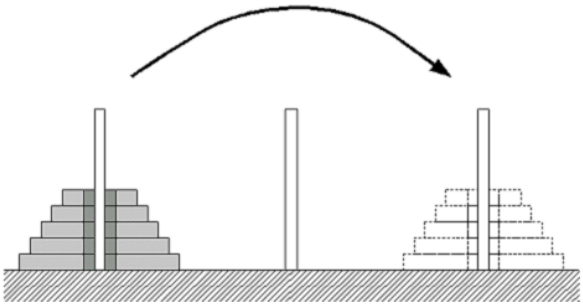

FIGURE 1 – Schéma des Tours de Hanoï.

Le but du jeu consiste à déplacer l'ensemble des disques pour que ceux-ci se retrouvent enfilés autour du piquet 3, en respectant les deux règles suivantes :

1. les disques sont déplacés un par un ;
2. un disque ne doit pas se retrouver au-dessus d'un disque plus petit.

On suppose évidemment que la règle n°2 est également respectée dans la configuration de départ.

Le problème a toujours une solution en $2^n - 1$ coups (admis). Il se résout de manière récursive. En effet, supposons le problème résolu pour $n-1$ disques, c'est-à-dire que l'on sache transférer $n-1$ disques depuis le piquet $i \in \{1, 2, 3\}$ jusqu'au piquet $j \in \{1, 2, 3\} \setminus \{i\}$ en respectant les règles du jeu. Pour transférer n disques du piquet i vers le piquet j, on procède alors comme suit :

1. on amène les $n-1$ disques du haut du piquet i sur le piquet intermédiaire, qui a le numéro $6-i-j$;
2. on prend le dernier disque du piquet i et on le met seul en j ;
3. on ramène les $n-1$ disques de $6-i-j$ en j.

QUESTION 1. Écrire une fonction récursive

```
def hanoi(n: int, i: int, j: int) -> list[tuple[int,int]]
```

qui construit la séquence des mouvements élémentaires à accomplir pour déplacer n disques du piquet i au piquet j. Chaque mouvement est représenté par un couple (s,t) indiquant que l'on déplace un disque du piquet s vers le piquet t. Afficher la séquence une fois construite.

QUESTION 2. Considérant que les disques sont identifiés de 1 à n, à partir du plus grand, modifier la fonction récursive pour qu'elle ajoute les numéros des disques à chaque mouvement. Un mouvement est alors un triplet (d,s,t) avec d le numéro du disque, s et t resp. les piquets source et cible.

2.1 Aperçu de `Tkinter`

Le code ci-dessous est un squelette opérationnel de programme `tkinter` :

```
from tkinter import *

# window creation
window = Tk()

# canvas creation
w = Canvas(window, width=400, height=300)
w.pack()

# drawing functions
# put your own drawing below
w.create_line(0, 0, 400, 300)
w.create_line(0, 300, 400, 0, fill="red")
w.create_rectangle(100, 75, 300, 225, fill="green")

# main event loop to allow interaction
mainloop()
```

La liste complète des figures géométriques disponibles n'est pas donnée ici, la documentation de *Tkinter* étant facilement disponible sur Internet.

Curieusement, il n'existe pas de fonction permettant de dessiner un point à l'écran dans ce module. La fonction peut cependant être émulée par le code suivant :

```
# function definition
def draw_point(canvas: Canvas, x: int, y: int, color: str="black") -> None:
    canvas.create_rectangle(x, y, x, y, fill=color, width=0)

# effective point drawing - to insert after pack() statement
draw_point(w,10,10)
```

Exercice 8: Tracé d'un segment

En informatique graphique, tous les points ont des coordonnées entières, exprimées en fonction d'une grille de résolution, ici la taille du canvas. Soient donc deux points A de coordonnées (x_A, y_A) et B de coordonnées (x_B, y_B) ; proposer un algorithme récursif pour tracer à l'écran le segment $[A, B]$ uniquement à l'aide de la fonction `draw_point` précédente.

En marge: *L'algorithme de* Bresenham *résoud ce problème élémentaire de tracé de segment.*

Exercice 9: Triangle de Sierpinski

Le triangle de Sierpiński est une fractale composée à partir de triangles, et illustrée sur la Figure 2.

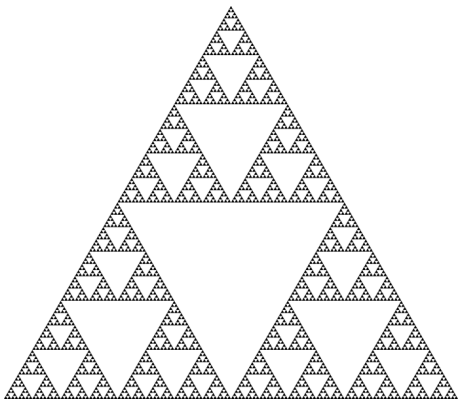

Figure 2 – Illustration d'un triangle de Sierpiński.

L'algorithme permettant d'obtenir cette figure consiste à tracer un triangle puis le triangle qui joint les milieux des côtés et ainsi de suite jusqu'à obtenir des segments plus petits qu'un pixel.

Écrire une fonction récursive permettant de dessiner ce triangle pour une taille de côté donnée.

3 Pour aller plus loin...

Exercice 10: Les permutations

Proposer une fonction récursive qui calcule toutes les permutations des entiers de 1 à n. Par exemple, les permutations de $\mathbb{N}_3$ sont $(1,2,3)$, $(1,3,2)$, $(2,1,3)$, $(2,3,1)$, $(3,1,2)$ et $(3,2,1)$.

Astuce: *La fonction se construit en observant que pour passer de* $\mathbb{N}_2$ *à* $\mathbb{N}_3$*, il « suffit » d'insérer 3 à toutes les positions des permutations de* $\mathbb{N}_2$*. En effet, à partir de* $(1,2)$ *on obtient* $(1,2,\mathbf{3})$*,* $(1,\mathbf{3},2)$ *et* $(\mathbf{3},1,2)$ *tandis qu'avec* $(2,1)$ *on obtient* $(2,1,\mathbf{3})$*,* $(2,\mathbf{3},1)$ *et* $(\mathbf{3},2,1)$*.*

Exercice 11: La diagonale de Cantor

La fonction de couplage de Cantor (cf. Fig.3) réalise une bijection de $\mathbb{N} \times \mathbb{N}$ dans $\mathbb{N}$. Par exemple $\mathsf{cantor}(1,1) = 4$ et $\mathsf{cantor}(1,3) = 13$.

Définir par récurrence et implémenter la fonction de couplage de Cantor.

Astuce: *Élaborer les différents cas de figure pour définir* $\mathsf{cantor}(x,y)$ *à partir du point précédent sur le tracé de couleur bleue sur la Figure 3.*

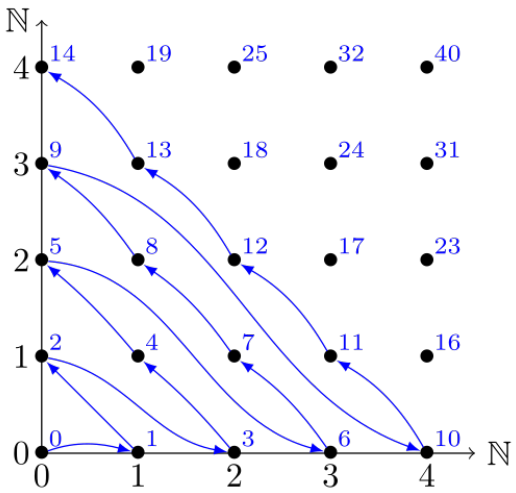

FIGURE 3 – Représentation de la fonction de couplage de CANTOR (src : Florent Demeslay, Wikipédia, 2021 - CC BY-SA 4.0).

Exercice 12: Le monnayeur

Supposons qu'une caisse enregistreuse dispose d'un stock inépuisable de pièces de 2 cts, 5 cts, 10 cts, 20 cts, 50 cts et 100 cts (1€). Le monnayeur automatique doit rendre une somme n, exprimée en centimes, avec *le minimum de pièces*.

Par exemple, si $n = 45$ cts, le monnayeur rend $(20, 20, 5)$ avec 3 pièces. On peut bien sûr envisager $(20, 10, 5, 2, 2, 2, 2, 2)$ ou tout autre arrangement de somme 45, néanmoins ils sont tous plus longs que 3.

QUESTION 1. Écrire la fonction $\mathsf{NP}(n)$ qui fournit le nombre de pièces nécessaires pour rendre n centimes.

Remarque: *Le monnayeur ne peut pas appliquer de méthode gloutonne qui consisterait à épuiser les pièces par ordre décroissant de leur valeur faciale. Si cela semble fonctionner pour $n = 45$, ce n'est pas applicable par exemple pour $n = 11$. En effet, on obtiendrait d'abord 10 puis rien (!?), or il existe une solution avec $(5, 2, 2, 2)$.*

Astuce: *Si le monnayeur sait rendre 72 cts avec un minimum de pièces, alors il sait rendre $72 - i$ cts pour i dans $\{2, 5, 10, 20, 50, 100\} \setminus \{100\}$. Cette observation esquisse la relation de récurrence.*

QUESTION 2. Retrouver par le calcul $\mathsf{NP}(11) = 4$ et $\mathsf{NP}(53) = 7$. Calculer $\mathsf{NP}(173)$.

En marge: *Le problème du monnayeur, plus connu en anglais sous l'appellation* Change-making Problem*, est un cas particulier du célèbre problème du « sac à dos » (*Knapsack Problem *en anglais).*

Exercice 13: Labyrinthe

Dans cet exercice, nous vous proposons de programmer un algorithme qui permet de s'orienter dans un labyrinthe. Le labyrinthe est représenté sous la forme d'un tableau à deux dimensions (des listes de listes en Python). La valeur 0 représente un mur, tandis que 1 désigne un espace que l'on peut occuper. La déclaration initiale du tableau ressemble donc à ceci :

```
maze: list[list[int]] = [
  [1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
  [0, 0, 1, 0, 1, 1, 0, 1, 0, 1],
  [0, 0, 1, 0, 1, 1, 1, 1, 1, 1],
  [1, 1, 1, 1, 1, 0, 0, 1, 0, 1],
  [0, 1, 0, 1, 0, 1, 1, 1, 0, 1],
  [1, 1, 1, 1, 1, 1, 0, 1, 1, 1],
  [0, 1, 0, 0, 1, 0, 1, 1, 0, 1],
  [0, 1, 0, 1, 1, 0, 1, 0, 1, 0],
  [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
  [0, 0, 1, 0, 0, 1, 0, 0, 0, 1]
]
```

Pour compléter le problème, on définit une case départ à l'aide de ses coordonnées dans le tableau, ainsi qu'une case arrivée.

Question 1. Créer un algorithme récursif permettant de trouver un chemin dans le labyrinthe, de la case départ jusqu'à la case arrivée. S'il n'existe aucun chemin, la fonction retourne une liste vide `[]`, sinon elle produit le chemin parcouru.

Question 2. Proposer une version de cet algorithme qui garantisse que le chemin emprunté est le plus court.

Exercice 14: Les nombres dominants

Un « nombre dominant » est un entier positif dont plus de la moitié de ses chiffres sont égaux.

Par exemple, 2022 est un nombre dominant car il contient trois occurrences du chiffre 2. À l'inverse, 2021 n'est pas un nombre dominant.

Soit $D(N)$ la quantité de nombres dominants inférieurs à 10^N.

Question 1. Retrouver par le calcul $D(4) = 603$ et $D(10) = 21\,893\,256$

Question 2. Calculer $D(2022)$. Donner la réponse modulo $1\,000\,000\,007$.

Remarque: *Problème n°788 de projecteuler.net.*