Révisions du premier semestre

Buts du TP

- Faire quelques révisions sur le programme du premier semestre ;
- Revenir sur la réalisation de graphiques avec la bibliothèque `matplotlib`.

1 Révisions sur les boucles, les tests, les fonctions, les listes

EXERCICE 1 Persistance d'un entier [1]

Si n désigne un entier naturel, on note prod(n) le produit des chiffres composant n (écrit en base 10). Par exemple, si $n = 128$, alors prod(n) vaut 16.

On appelle persistance de n l'entier égal au nombre de fois qu'il faut itérer la fonction prod pour obtenir un nombre inférieur ou égal à 9 (après, la suite est stationnaire). Par exemple, la persistance de 128 est 2.

1. *Écrire une fonction `decoupe` qui, étant donné un entier naturel `n`, renvoie le couple `(N,u)` où `u` est le chiffre des unités de `n` et `N` est le nombre obtenu en retirant à `n` son chiffre des unités.*

   exemple : `decoupe(145)` renverra `(14,5)` et `decoupe(6)` renverra `(0,6)`

2. *Écrire une fonction* `chiffres` *qui renvoie la liste des chiffres composant l'écriture décimale d'un entier naturel donné `n`.*

   exemple : `chiffres(145)` *renverra `[1,4,5]`.*

3. *Écrire la fonction `prod`.*

4. *Écrire une fonction `persistance` permettant d'obtenir la persistance d'un entier naturel `n` passé en argument d'entrée.*

5. *Écrire une fonction* `max_persistance(n)` *calculant la persistance de tous les entiers compris entre* 10 *et n (au sens large) et permettant de trouver la persistance maximale.*

EXERCICE 2

1. *Écrire une fonction* `maxi(t)` *qui retourne le maximum de son paramètre* `t` *qui est une liste d'entiers supposée non vide.*

2. *Écrire une fonction* `mini(t)` *qui retourne le minimum de son paramètre* `t` *qui est une liste d'entiers.*

3. *Quelle est la complexité des fonctions* `maxi(t)` *et* `mini(t)` *en fonction du nombre d'éléments dans le paramètre* `t` *? On comptera uniquement les comparaisons.*

4. *Écrire une fonction* `min_maxi(t)` *qui retourne un couple constitué du minimum et du maximum de son paramètre* `t` *qui est un tableau d'entiers.*

5. *Si ce n'est déjà le cas, modifier la fonction* `min_maxi(t)` *pour qu'elle implémente (si la taille de* `t` *est impaire, on dupliquera le dernier élément pour avoir une liste paire) l'algorithme suivant :*

```
Initialiser les variables
Parcourir le tableau par paire de deux éléments successifs
   Pour chaque paire, déterminer son minimum et son maximum
   Mettre à jour le maximum et le minimum global
Retourner le minimum et le maximum global
```

1. Exercice de M.Lalauze

```
>>> min_maxi([10,5,18,6])
(5, 18)
>>> min_maxi([10,5,18,6,843,-15,0])
(-15, 843)
```

Comparer les complexités des deux versions de la fonction `min_maxi(t)`. *On comptera uniquement les comparaisons.*

EXERCICE 3 Run-length encoding [2]

On considère des listes qui ne peuvent contenir que des 0 *et des* 1, *que l'on souhaite compresser. À partir d'une liste u (par exemple* `u = [1, 1, 1, 0, 1, 0, 0, 1]`*), on construit ainsi une liste v :*

- *le premier élément est égal au premier élément de u (ici* 1*);*
- *ensuite, on indique le nombre de fois que cet élément est répété (ici* 3*);*
- *à chaque fois qu'une série de* 0 *ou de* 1 *se termine, on indique la longueur de cette série.*

À partir du u donné en exemple, on doit donc obtenir `v = [1, 3, 1, 1, 2, 1]`, *et à partir de* `u = [0, 0, 0, 1, 1]` *on obtiendrait* `v = [0, 3, 2]`.

1. *Écrire une fonction* `compresse(u)` *qui prend en entrée une liste u (on supposera qu'elle ne contient que des* 0 *et des* 1*) et qui retourne la liste compressée correspondante.*
2. *Écrire une fonction* `decompresse(v)` *qui prend en entrée une liste compressée suivant la méthode décrite et retourne la liste initiale.*

EXERCICE 4 Défi Turing Problème 25

La suite de Fibonacci est définie par $\begin{cases} f_0 = 1,\ f_1 = 1 \\ \forall n \in \mathbb{N},\ f_{n+2} = f_{n+1} + f_n \end{cases}$.
Ainsi, les douze premiers termes sont les suivants : 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ...
$f_{11} = 144$ *est le premier terme avec trois chiffres.*
Déterminer le premier terme de la suite avec 2020 *chiffres.*

EXERCICE 5 Défi Turing Problème 67

La suite diatomique de Stern est définie par $\begin{cases} s_0 = 0 \\ s_1 = 1 \\ \forall n \in \mathbb{N},\ s_{2n} = s_n \\ \forall n \in \mathbb{N},\ s_{2n+1} = s_n + s_{n+1} \end{cases}$ *. Calculer la valeur de* $s_{10000001}$.

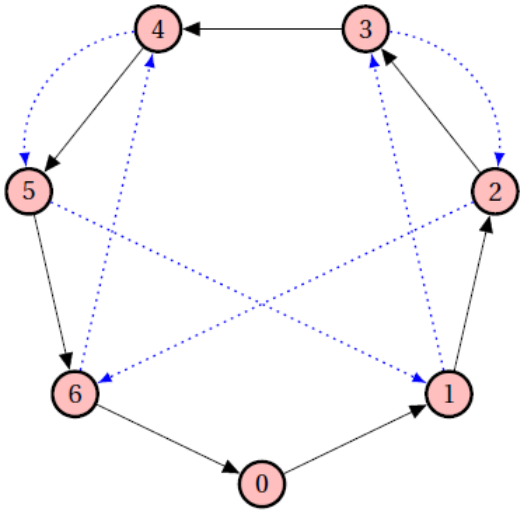
EXERCICE 6

1. *Écrire une fonction* `nombre2chiffres(n, b)` *qui prend en argument un entier naturel* `n` *écrit en base* 10 *et une base* `b` *et qui retourne la liste des chiffres de ce nombre dans la base choisie.*
2. *Écrire une fonction* `chiffres2nombre(n, b)` *réciproque de la précédente qui retourne l'écriture en base* 10 *d'un nombre à partir de la liste de ses chiffres dans une base fixée. La fonction implémentera l'algorithme d'Horner, par exemple à partir de la liste de chiffres* `[8, 4, 2]` *en base dix, on écrira* $((((0) \times 10 + 8) \times 10 + 4) \times 10 + 2) = 842$.
3. *Écrire une fonction* `palindrome(n, b)` *qui retourne un booléen indiquant si l'entier naturel* `n` *est un palindrome dans la* `base` *(*$\leqslant 10$*) sélectionnée. Par exemple* 404 *est un palindrome en base dix mais pas en base deux (*$\overline{110010100}^2$*).*
4. *Le nombre décimal 585 (*$\overline{1001001001}^2$ *en base deux) est un palindrome dans les bases dix et deux.*

   Déterminer la somme de tous les nombres inférieurs à 10 millions, qui sont des palindromes en base dix et en base deux.

2. Exercice de M.Bianquis

2 Critère de divisibilité par 7



EXERCICE 7 *On connaît bien des critères de divisibilité par* 2, 3, 4, 5 *et* 9 *mais pour la divisibilité par* 7 *?*

Pour déterminer si un entier naturel en écriture décimale est divisible par 7 *on va utiliser le graphe ci-dessus. Par exemple, pour* 308 *on procède ainsi :*

- ☞ *On part du sommet avec l'étiquette* 0.
- ☞ *Aucune flèche en pointillé ne part de* 0 *donc on ne parcourt pas de flèche en pointillé. Le premier chiffre par poids décroissant de* 308 *est* 3 *donc on parcourt trois flèches en trait plein et on arrive au sommet* 3.
- ☞ *Depuis le sommet* 3 *on parcourt une flèche en pointillé pour arriver au sommet* 2. *Le second chiffre par poids décroissant est* 0 *donc on parcourt zéro flèche en trait plein et on reste en* 2.
- ☞ *Depuis le sommet* 2, *on suit une flèche en pointillés pour arriver au sommet* 6. *Le troisième chiffre par poids décroissant est* 8 *donc on parcourt huit flèches en trait plein et on arrive au sommet* 0.
- ☞ *On a balayé tous les chiffres de* 308 *donc on s'arrête.*
- ☞ *Le dernier sommet atteint est* 0, *il représente le reste de la division euclidienne de* 308 *par* 7 *qui est donc divisible par* 7.

1. *Vérifier que l'algorithme décrit ci-dessus permet bien de déterminer le reste de la division euclidienne de* 349 *par* 7.
2. *Écrire une fonction* `divisible7(n)` *qui retourne un booléen indiquant si* `n` *est divisible par* 7.
3. *Avec une liste en compréhension déterminer la liste des restes des* 10^k *pour* k *variant dans l'intervalle d'entiers* $[\![0\,;7]\!]$. *Quelle remarque peut-on faire par rapport au graphe ?*
4. *On rappelle sur un exemple l'algorithme d'Horner pour déterminer l'écriture décimale d'un entier à partir de la liste de ses chiffres :* $((((0)\times 10+3)\times 10+0)\times 10+8)=308$.

   Justifier que la fonction `divisible7` *écrite précédemment est correcte.*

3 Algorithme glouton

Dans un système monétaire fixé, le **problème du rendu de monnaie** consiste à déterminer le nombre minimal de pièces pour rendre la monnaie quelle que soit la somme. Pour simplifier la terminologie nous considérerons uniquement des systèmes de pièces, ce qui correspond au problème de rendu de monnaie pour un distributeur de café et nous utiliserons le terme somme pour désigner une somme d'argent.

EXERCICE 8 *Jean veut boire un café, le distributeur propose un café à* 40 *centimes et il ne dispose que d'une pièce de* 2 *euros. La machine est programmée pour rendre la monnaie de Jean en un* ***nombre minimal de pièces****.*

Pour simplifier, on considère que les pièces disponibles dans la machine sont toutes les pièces du système monétaire de l'euro et on rappelle que leurs valeurs en ***centimes*** *sont* $[1,2,5,10,20,50,100,200]$.

1. *Déterminer le nombre de pièces rendues par la machine à Jean.*
2. *Déterminer le nombre de pièces rendues par la machine pour une somme s'élevant à* 598 *centimes.*

Un algorithme simple pour le rendu de monnaie est **l'algorithme glouton** qui consiste à choisir en priorité la plus grande pièce du système monétaire inférieure ou égale à la somme restante puis à recommencer tant que la somme restante n'est pas nulle.

EXERCICE 9 Algorithme glouton pour le rendu de monnaie

1. *La fonction Python ci-dessous prend en argument un entier* `somme` *et une liste* `systeme` *et retourne la décomposition de* `somme` *sous forme de liste de pièces choisies dans* `systeme`*.*

```
1  def rendu_monnaie_glouton(somme, systeme):
2      systeme_copie = sorted(systeme, reverse = True)
3      rendu = []
4      for idpiece in range(len(systeme_copie)):
5          piece = systeme_copie[idpiece]
6          ..............................
7          ..............................
8          ..............................
9      assert somme == 0
10     return rendu
```

   (a) *Quel rôle joue l'instruction en ligne 3 ? et celle en ligne 10 ?*

   (b) *Récupérer ce code dans* `cadeau.py`*. et le coller dans* `TD-Glouton2.py`*, puis compléter le code de la fonction* `rendu_monnaie_glouton` *qui se trouve dans* `cadeau.py`*.*

   (c) *Tester cette fonction pour déterminer le rendu de monnaie sur une somme de* 520 *centimes d'euros avec pour système monétaire la liste* $[1,2,5,10,20,50,100,200]$ *des valeurs des pièces de la zone euro en centimes.*

   (d) *Que va-t-il se passer si on exécute l'appel* `rendu_monnaie_glouton(53,[2, 5, 10, 20])` *?*

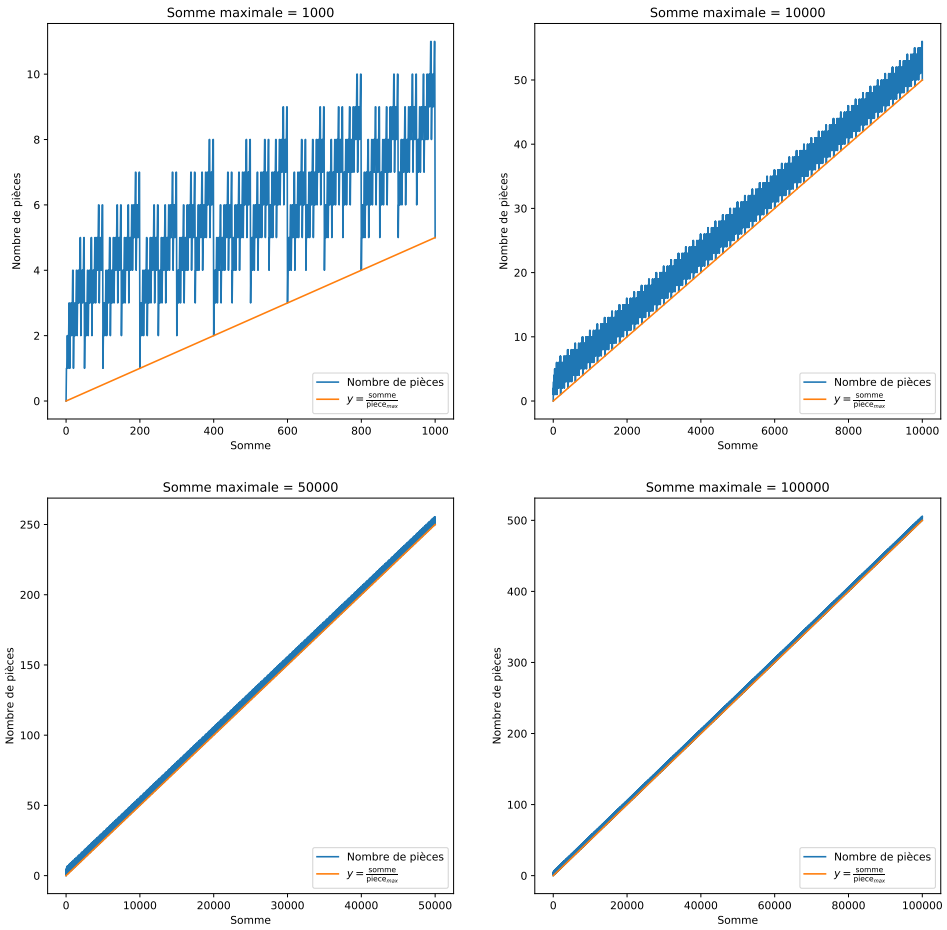
EXERCICE 10 Complexité de l'algorithme glouton

1. *Récupérer le code ci-dessous dans* `cadeau.py`*. Exécuter.*

```
1  import matplotlib.pyplot as plt
2  systeme = [1, 2, 5, 10, 20, 50, 100, 200]
3  bsup = 1000
4  les_sommes = list(range(bsup + 1))
5  les_rendus = [len(rendu_monnaie_glouton(somme, systeme)) for somme in
       les_sommes]
6  plt.plot(les_sommes, les_rendus, label = 'Nombre de pièces')
7  plt.title('Somme maximale = {}'.format(bsup))
8  plt.plot([0,bsup],[0, bsup // systeme[-1]])
9  plt.savefig('complexite-glouton-bsup{}.pdf'.format(bsup))
10 plt.show()
```

2. *Que représentent les graphiques tracés par les instructions en lignes* 6 *et* 8 *? La documentation de la bibliothèque* `matplotlib` *se trouve sur* `https://matplotlib.org/users/pyplot_tutorial.html`*.*

3. *Modifier le code pour enregistrer dans le répertoire de* `TD-Glouton2.py` *les graphiques ci-dessous :*



4. *Quelle conjecture peut-on faire sur l'ordre de grandeur de la* ***complexité temporelle*** *de l'algorithme glouton pour un rendu de monnaie sur une somme n assez grande ? Donner une idée de la preuve de cette conjecture.*

4 Marche aléatoire verte

L'exercice suivant est tirés de l'ouvrage *Python : les bases de l'algorithmique et de la programmation* écrit par Vincent Maille et disponible au CDI.

EXERCICE 11 Marche aléatoire et fougère de Barnsley.

On considère une famille de quatre fonctions affines du plan $\mathscr{P}$ dans lui-même. Ce sont des fonctions qui à tout point $M(x;y)$ du plan associent un point $M'(ax+by+c;\alpha x+\beta y+\gamma)$.

De plus, on définit une variable aléatoire Z sur l'univers constitué de ces quatre fonctions et pour tout entier $k \in [\![1;4]\!]$, on note $\mathbb{P}(Z=k)=p_k$.

k	p_k	a_k	b_k	c_k	α_k	β_k	γ_k
1	0,01	0	0	0	0	0,16	0
2	0,85	0,85	0,04	0	−0,04	0,85	1,6
3	0,07	0,2	−0,26	0	0,23	0,22	1,6
4	0,07	−0,15	0,28	0	0,26	0,24	0,44

On définit une marche aléatoire dans le plan $\mathscr{P}$ en partant d'un point de coordonnées (x_0,y_0) choisies aléatoirement dans $[-1;1]\times[-1;1]$. On note (x_n,y_n) les coordonnées du point lors du pas n et on a :

$$\begin{cases} x_{n+1} = a_k x_n + b_k y_n + c_k \\ y_{n+1} = \alpha_k x_n + \beta_k y_n + \gamma_k \end{cases} \qquad \text{avec} \qquad Z(\omega) = k$$

Autrement dit, lors de chaque étape, les nouvelles coordonnées sont les images des précédentes par la fonction f_k choisie par la variable aléatoire Z.

1. *Récupérer les fonctions* `f1`, `f2`, `f3` *et* `f4` *dans le module* `cadeau.py`.

2. *Écrire une fonction* `choixfonction()` *qui retourne la fonction f_k choisie par la variable aléatoire Z.*

```
In [16]: sum([1 for k in range(1000) if choixfonction().__name__ == 'f2' ])
     /1000
Out[16]: 0.851
```

3. *On rappelle que la fonction* `random` *du module* `random` *simule une loi uniforme sur l'intervalle* $[0;1[$. *En déduire une expression* `Python` *qui simule une loi uniforme sur l'intervalle* $[-1;1[$.

4. *Compléter le code de la fonction* `marche_aléatoire(n)` *qui réalise une marche aléatoire de n pas et qui retourne un tableau avec les coordonnées des différents points du parcours.*

```
from random import random

def marche_aleatoire(n):
    x = ....
    y = ....
    lesx = [x]
    lesy = [y]
    .....
```

5. *Représenter graphiquement une marche aléatoire de* 1 000 *pas avec le code suivant :*

```
(lesx, lesy) = marche_aleatoire(10000)
plt.axis('equal') #repere orthonormal
plt.plot(lesx, lesy, 'k,') #affichage sous forme de pixels noirs
plt.show()
plt.savefig('marche1000.png')
```