Quelques rappels et compléments

I Rappels d'algorithmique

1 Preuve d'algorithme

On rappelle que la preuve d'un algorithme passe par :

- **Une preuve de terminaison** pour les boucles conditionnelles (`while`). Cela se fait via un *variant de boucle* : en général une suite entière minorée strictement décroissante.
- **La détermination d'*invariants de boucle*** : propriétés décrivant l'état du système (ou une partie de celui-ci) à un instant donné d'une boucle, qui se prouve par récurrence.
- **La correction de l'algorithme** : on peut alors connaître l'état du système en fin d'exécution et donc vérifier que la quantité renvoyée est la bonne, ou que le traitement effectué est le bon.

Exemple : Exponentiation naïve et exponentiation rapide itérative

- Écrire une fonction `puiss(x, n)` renvoyant x^n et prouver sa correction.
- Pour calculer x^n, on remarque la chose suivante $x^{2^{k+1}} = \left(x^{2^k}\right)^2$: il suffit donc d'une multiplication pour passer de x^{2^k} à $x^{2^{k+1}}$.
  L'idée est alors de décomposer n en base 2 : si n s'écrit
  $$n = b_0 + b_1 \cdot 2 + b_2 2^2 + \cdots + b_k 2^k$$
  avec $b_0, \cdots, b_k \in \{0,1\}$ (écriture en base 2) alors
  $$x^n = x^{b_0} \left(x^2\right)^{b_1} \cdots \left(x^{2^k}\right)^{b_k},$$
  le calcul de x^{2^p} s'effectuant à partir de celui de $x^{2^{p-1}}$.
  Par exemple, $19 = 1 + 2 + 2^4$ et $x^{19} = x \times x^2 \times \left(\left(\left(x^2\right)^2\right)^2\right)^2$ (6 multiplications nécessaires), $39 = 1 + 2 + 2^2 + 2^5$ et
  $$x^{39} = x \times x^2 \times \left(x^2\right)^2 \times \left(\left(\left(\left(x^2\right)^2\right)^2\right)^2\right)^2$$
  (8 multiplications nécessaires).
  Écrire une fonction `exprap(x, n)` mettant on œuvre cet algorithme et prouver sa correction.

2 Calculs de complexité

a Principe

Calculer la complexité d'un algorithme revient à évaluer le temps d'exécution et/ou la place occupée dans la mémoire pendant l'exécution.

La mesure peut se faire de manière empirique (`time()`, `timeit()`...) mais dépendra de la machine, d'autres tâches en cours, de l'échantillon de test, etc. ou de manière théorique, mais cela demande de modéliser la machine et de choisir ce qui va être compter, certaines opérations pouvant être plus pertinentes que d'autres (plus lentes ou pour comparer plusieurs algorithmes. Pour calculer la complexité d'un tri d'un tableau, par exemple, on peut choisir de compter plutôt les comparaison entre éléments du tableau ou bien les échanges d'éléments dans le tableau (ou les deux).

Les opérations élémentaires prennent, suivant les machines et les langages, entre 1 ns (10^{-9} s) et 1 μs (10^{-6} s) à être exécutées :

- addition, soustraction, multiplication, division, modulo sur des entiers ou des flottants, comparaison de nombres,
- affectation simple,
- accès aux éléments d'un tableau,
- modification des éléments d'un tableau,
- taille d'un tableau,
- La méthode `append` sur une liste Python peut être considérée comme une opération élémentaire.

b Notations

Notation : O, Ω et Θ

On suppose que $f(n) \geqslant 0$ et $g(n) \geqslant 0$ pour tout entier n.

- On note $f = O(g)$ lorsqu'il existe un rang n_0 et une constante $c > 0$ tels que

$$\forall\, n \geqslant n_0,\quad f(n) \leqslant c \times g(n)$$

  Cela signifie que f croît au plus aussi vite que g.

- On note $f = \Omega(g)$ lorsqu'il existe un rang n_0 et une constante $d > 0$ tels que

$$\forall\, n \geqslant n_0,\quad f(n) \geqslant d \times g(n)$$

  Cela signifie que f croît au moins aussi vite que g.

- On note $f = \Theta(g)$ lorsque $f = O(g)$ et $f = \Omega(g)$ c'est-à-dire lorsqu'il existe un rang n_0 et des constantes $c, d > 0$ tels que

$$\forall\, n \geqslant n_0,\quad c \times g(n) \leqslant f(n) \leqslant c \times g(n)$$

  Cela signifie que f et g sont du même ordre.

Remarque

Les calculs de complexité en moyenne étant souvent compliqués à mener, faisant intervenir des probabilités, le programme se limite aux complexités au mieux ou au pire. En général, une complexité s'exprimera $T(n) = O(f(n))$ avec $f(n)$ optimisé.

C Tableau comparatif

Chaque valeur est multipliée par 10^{-6} s comme majorant du temps d'exécution d'une opération élémentaire.

Croissance	n	10	50	100	500	1000
logarithmique	$\ln n$	2 μs	4 μs	4,6 μs	6 μs	7 μs
	$\sqrt{n}$	3 μs	7 μs	10 μs	20 μs	30 μs
linéaire	n	10 μs	50 μs	100 μs	0,5 ms	1 ms
semi-linéaire	$n\ln n$	20 μs	200 μs	500 μs	3 ms	7 ms
quadratique	n^2	100 μs	2,5 ms	10 ms	0,25 s	1s
polynomiale	n^3	1 ms	0,1 s	1 s	2 min	16 min
exponentielle	2^n	1 ms	36 a	$4\cdot10^6$ a	10^{137} a	$3\cdot10^{287}$ a
	$n!$	4 s	10^{51} a	$3\cdot10^{144}$ a	$3\cdot10^{1121}$ a	$3\cdot10^{2554}$ a

Ordres de grandeurs :

- **Âge de l'univers** : $13,8\times10^9$ années
- **Nombre d'atomes dans l'univers** : 10^{80}.

Échelle de croissances comparées :

$$\ln n \ll \sqrt{n} \ll n \ll n\ln n \ll n^2 \ll n^3 \ll 2^n \ll n! \ll n^n$$

n maximum pour un temps d'exécution d'une seconde :

$\ln n$	$\sqrt{n}$	n	$n\ln n$	n^2	n^3	2^n	$n!$
$\simeq 10^{400\ 000}$	10^{12}	10^6	87848	10^3	100	20	10

On retiendra qu'à partir de n^3, le temps d'exécution n'est pas raisonnable.

Exercice

Calculer les complexités des fonctions d'exponentiation précédentes.

II TABLEAUX ET LISTES EN INFORMATIQUE

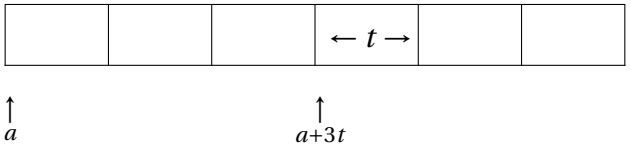
Il convient dans la pratique de l'algorithmique de choisir de représenter ses données en utilisant une structure appropriée, dépendant de ce qu l'on a besoin d'en faire et de ce que cela coûte.

On rencontre deux types de structures : celles pour lesquelles un emplacement dans la mémoire (RAM) aura été alloué dès la création et ne changera pas, dites **statiques** et celles dont l'allocation mémoire et la taille peuvent varier au cours de l'exécution d'un algorithme, dites **dynamiques**. Lorsque l'on peut modifier les éléments de cette structure, elle dite **mutable**.

Par exemple, en python, les tuples sont statiques, non mutables, et les listes sont dynamiques et mutables.

1 Tableau

Un tableau informatique est une structure statique pour laquelle on fixe à la création un nombre d'éléments et des espaces mémoires contigu pour chaque élément (de type fixé). Connaissant l'adresse a de premier élément et la taille t correspondant au type d'élément, on accède à l'élément numéro k à l'adresse $a+kt$.

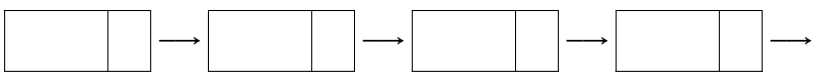


- **Avantages :** chaque élément est accessible en lecture et en écriture à temps constant, ainsi que la taille du tableau.
- **Inconvénients :** structure statique : taille non modifiable. Insertion, suppression lentes.

Le module `numpy` fournit un type `ndarray` correspondant à un tableau informatique.

2 Liste chaînée

Une liste est une structure dynamique qui est telle que chaque élément à stocker est encapsulé dans un objet contenant aussi un pointeur vers l'élément suivant. Pour atteindre l'élément numéro k, on est obligé de parcourir tous les éléments précédents dans la liste.



- **Avantages :** Structure dynamique, insertion, suppression rapides.
- **Inconvénients :** k^{e} élément accessible en $O(k)$.

Il existe d'autres types de listes (doublement-chaînée : chaque élément pointe vers son successeur et son prédécesseur, circulaire : le dernier élément pointe sur le premier, etc.)

⚠ Comme son nom ne l'indique pas, le type `list` de python n'est pas une liste chaînée. C'est un ingénieux mélange de tableau et de liste.

III Python

Après un an d'utilisation régulière de Python, vous devez être à l'aise avec ce langage. Attention à ne pas négliger la présentation du code pour optimiser la lisibilité : docstring et PEP 8 sont de rigueur.

1 Les listes Python

Python permet de travailler avec des objets de type `list` qui est une structure de données dynamique mutable : ce n'est cependant ni un tableau ni une liste au sens informatique du terme.

Le principe est le suivant : lorsque l'on crée une liste ou lorsque l'on ajoute des éléments à une liste, on réserve une place plus grande en mémoire. Plus exactement, la liste est « allongée » lorsque l'on passe certains paliers qui sont :

$$0, 4, 8, 16, 25, 35, 46, 58, 72, 88, 106, 126, 148, 173, 201, 233, \ldots$$

D'où viennent ces valeurs ? Pour le savoir, il faut fouiller de le code source de CPython, consultable librement :

```
/* This over-allocates proportional to the list size, making room
* for additional growth.  The over-allocation is mild, but is
* enough to give linear-time amortized behavior over a long
* sequence of appends() in the presence of a poorly-performing
* system realloc().
* The growth pattern is:  0, 4, 8, 16, 25, 35, 46, 58, 72, 88, ...
*/
new_allocated = (newsize >> 3) + (newsize < 9 ? 3 : 6);

/* check for integer overflow */
if (new_allocated > PY_SIZE_MAX - newsize) {
    PyErr_NoMemory();
    return -1;
} else {
    new_allocated += newsize;
}
```

Le `>> 3` signifie un décalage de 3 bits vers la droite, donc une division entière par 2^3 et `newsize < 9 ? 3 : 6` vaut 3 si `newsize < 9` et 6 sinon.

Autrement dit, si n est la taille de la liste, la place allouée dans la mémoire sera

$$n + \left\lfloor \frac{n}{8} \right\rfloor + \begin{cases} 3 & \text{si } n < 9 \\ 6 & \text{sinon} \end{cases}$$

La gestion détaillée de l'allocation de mémoire lors de la manipulation de pile est très bien décrite ici : http://www.laurentluce.com/posts/python-list-implementation/.

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

>>> L[1] = 0 # Mutable
>>> L
[1, 0, 3]

>>> L[1:3] # Slicing
[0, 3]

>>> len(L)
3

>>> L + [1,2,3] # Concaténation
[1, 0, 3, 1, 2, 3]

>>> 3 in L # Appartenance
True

>>> tuple(L) # Casting
(1, 0, 3)
>>> list(t)
[1, 2.65, True, 'azerty']
>>> str(L)
'[1, 0, 3]'
>>> list("Bonjour")
['B', 'o', 'n', 'j', 'o', 'u', 'r']

# Ajouter un élément à la fin
>>> L.append(5)
>>> L
[1, 0, 3, 5]

# Récupérer le dernier élément
>>> a = L.pop()
>>> a
5
>>> L
[1, 0, 3]
```

```
>>> L2 = L
>>> id(L), id(L2)
(140352491149648, 140352491149648)
>>> id(L) == id(L2)
True

>>> L[1] = 12
>>> L
[1, 12, 0]
>>> L2
[1, 12, 0]


>>> L3 = L.copy()
# ou bien L[:] ou encore list(L)
>>> id(L) == id(L3)
False
>>> L[1] = 36
>>> L
[1, 36, 0]
>>> L3
[1, 12, 0]


>>> LL = [[1],[2],[3]]
>>> LL2 = LL.copy()
# copie superficielle
>>> LL2
[[1], [2], [3]]

>>> LL2[0][0] = 0
>>> LL2
[[0], [2], [3]]
>>> LL
[[0], [2], [3]]

>>> LL2[0] = [1]
>>> LL
[[0], [2], [3]]
>>> LL2
[[1], [2], [3]]
```

- ⚠ Certaines opérations peuvent se révéler dangereuses : on évitera a tout pris d'écrire, par exemple

  `L = [a] * n`

  mais on préférera

  `L = [a for _ in range(n)].`

  Si `a` n'est pas mutable, cela ne pose pas de problème, mais s'il l'est, c'est très embêtant ! Le même genre de problème peut se présenter lorsque l'on utilise `+=` avec des listes de listes.

- Notons que le module `copy` fournit une fonction `deepcopy` permettant de faire des copies profondes de listes.
- On rappelle également que les listes sont passées **par référence** lorsqu'elles sont arguments de fonctions : cela signifie que la liste peut être modifiée à l'intérieur d'une fonction dont elle est l'un des arguments.
- Lu sur https://wiki.python.org/moin/TimeComplexity :

  Generally, 'n' is the number of elements currently in the container. 'k' is either the value of a parameter or the number of elements in the parameter.

Operation	Average Case	Amortized Worst Case
Copy	$O(n)$	$O(n)$
Append(1)	$O(1)$	$O(1)$
Insert	$O(n)$	$O(n)$
Get Item	$O(1)$	$O(1)$
Set Item	$O(1)$	$O(1)$
Delete Item	$O(n)$	$O(n)$
Iteration	$O(n)$	$O(n)$
Get Slice	$O(k)$	$O(k)$
Del Slice	$O(n)$	$O(n)$
Set Slice	$O(k+n)$	$O(k+n)$
Extend(1)	$O(k)$	$O(k)$
Sort	$O(n \log n)$	$O(n \log n)$
Multiply	$O(nk)$	$O(nk)$
x in s	$O(n)$	
min(s), max(s)	$O(n)$	
Get Length	$O(1)$	$O(1)$

(1) = These operations rely on the "Amortized" part of "Amortized Worst Case". Individual actions may take surprisingly long, depending on the history of the container.

2 Les fichiers

Ce paragraphe a pour seul but d'introduire le mot-clé `with` que l'on s'efforcera d'utiliser pour la gestion de flux : ouverture de fichier, de base de donnée, etc.

Par exemple :

```
with open(nomFichier,'r') as file:
    <...>
    <instructions utilisant file>
    <...>
```

Cela permet d'optimiser la gestion de la mémoire en fermant le fichier automatiquement dès que le traitement est terminé.

Exercices

1 *Extrait de Centrale 2015*

1. Donner la valeur des expressions python suivantes :

   (a) `[1, 2, 3] + [4, 5, 6]`

   (b) `2 * [1, 2, 3]`

2. Écrire une fonction python `smul` à deux paramètres, un nombre et une liste de nombres, qui multiplie chaque élément de la liste et renvoie une nouvelle liste : `smul(2, [1, 2, 3])` → `[2, 4, 6]`.

3. ***Arithmétique des listes***

   (a) Écrire une fonction python `vsom` qui prend en paramètre deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la somme terme à terme de ces deux listes :

   `vsom([1, 2, 3], [4, 5, 6])` → `[5, 7, 9]`.

   (b) Écrire une fonction python `vdif` qui prend en paramètre deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la différence terme à terme de ces deux listes (la première moins la deuxième) :

   `vdif([1, 2, 3], [4, 5, 6])` → `[-3, -3, -3]`.

2 *Extrait de CCP 2015 Maths 2*

Les algorithmes demandés doivent être écrits en Python. On sera très attentif à la rédaction et notamment à l'indentation du code.

Voici, par exemple, un code Python attendu si l'on demande d'écrire une fonction nommée `maxi` qui calcule le plus grand élément d'un tableau d'entiers :

```
def maxi(t):
    """Données: t un tableau d'entiers non vide
    Résultat: le maximum des éléments de t"""
    n = len(t) # la longueur du tableau t
    maximum = t[0]
    for k in range(1, n):
        if t[k] > maximum:
            maximum = t[k]
    return maximum
```

L'instruction `maxi([4,5,6,2])` renverra alors `6`.

1. Donner la décomposition binaire (en base 2) de l'entier 21.

On considère la fonction `mystere` suivante :

```
def mystere(n, b):
    """Données: n > 0 un entier et b > 0 un entier
    Résultat: ......."""
    t = [] # tableau vide
    while n > 0:
        c = n % b
        t.append(c)
        n = n // b
    return t
```

On rappelle que la méthode `append` rajoute un élément en fin de liste. Si l'on choisit par exemple `t = [4,5,6]`, alors, après avoir exécuté `t.append(12)`, la liste `t` a pour valeur `[4,5,6,12]`.

Pour $k \in \mathbb{N}^*$, on note c_k , t_k et n_k les valeurs prises par les variables c, t et n à la sortie de la k-ème itération de la boucle "while".

2. Quelle valeur est renvoyée lorsque l'on exécute `mystere(256, 10)` ?

On recopiera et complétera le tableau suivant, en ajoutant les éventuelles colonnes nécessaires pour tracer entièrement l'exécution.

k	1	2	$\cdots$
c_k			$\cdots$
t_k			$\cdots$
n_k			$\cdots$

3. Soit $n > 0$ un entier. On exécute `mystere(n, 10)`. On pose $n_0 = n$.

a. Justifier la terminaison de la boucle `while`.

b. On note p le nombre d'itérations lors de l'exécution de `mystere(n,10)`. Justifier que pour tout $k \in [\![0, p]\!]$, on a $n_k \leqslant \frac{n}{10^k}$. En déduire, une majoration de p en fonction de n.

4. En s'aidant du script de la fonction `mystere`, écrire une fonction `somme_chiffres` qui prend en argument un entier naturel et renvoie la somme de ses chiffres. Par exemple, `somme_chiffres(256)` devra renvoyer `13`.

5. Écrire une version récursive[1] de la fonction `somme_chiffres`, on la nommera `somme_rec`.

3 *Extrait de CCP 2016 Maths 2 : Algorithme d'Euclide*

Les algorithmes demandés doivent être écrits en Python. On sera très attentif à la rédaction et notamment à l'indentation du code. Cet exercice étudie deux algorithmes permettant le calcul du pgcd (plus grand diviseurs communs) de deux entiers naturels.

1. La récursivité sera vue prochainement...

1. Pour calculer le pgcd de 3705 et 513, on peut passer en revue tous les entiers $1,2,3,\ldots,512,513$ puis renvoyer parmi ces entiers le dernier qui divise à la fois 3705 et 513. Il sera alors bien le plus grand des diviseurs commun à 3705 et 513. Écrire une fonction `gcd` qui renvoie le pgcd de deux entiers naturels non nuls, selon la méthode décrite ci-dessus. On pourra éventuellement utiliser librement l'instruction `min(a,b)` qui calcule le minimum de a et b. Par exemple `gcd(3705 ,513)` renverra 57.
2. L'algorithme d'Euclide permet aussi de calculer le pgcd. Voici une fonction Python nommée `euclide` qui implémente l'algorithme d'Euclide.

```
def euclide(a,b):
    """Données: aet b deux entiers naturels
    Résultat: le pgcd de a et b, calculé par l'algorithme d'Euclide"""
    u = a
    v = b
    while v != 0:
        r = u % v
        u = v
        v = r
    return u
```

   Écrire une fonction « récursive [1] » `euclide_rec` qui calcule le pgcd de deux entiers naturels selon l'algorithme d'Euclide.
3. On note $(F_n)_{n\in\mathbb{N}}$ la suite des nombres de Fibonacci définie par :

$$F_0 = 0, F_1 = 1, \quad \forall n \in \mathbb{N},\ F_{n+2} = F_{n+1} + F_n.$$

   3.a. Écrire les divisions euclidiennes successivement effectuées lorsque l'on calcule le pgcd de $F_6 = 8$ et $F_5 = 5$ avec la fonction `euclide`.

   3.b. Soit $n \geqslant 2$ un entier. Quel est le reste de la division euclidienne de F_{n+2} par F_{n+1} ? On pourra utiliser librement que la suite $(F_n)_{n\in\mathbb{N}}$ est strictement croissante à partir de $n = 2$. En déduire, sans démonstration, le nombre u_n de divisions euclidiennes effectuées lorsque l'on calcule le pgcd de F_{n+2} et F_{n+1} avec la fonction `euclide`.

   3.c. Comparer pour n au voisinage de $+\infty$, ce nombre u_n, avec le nombre v_n de divisions euclidiennes effectuées pour le calcul du pgcd de F_{n+2} et F_{n+1} par la fonction `gcd`. On pourra utiliser librement que F_n est équivalent au voisinage de $+\infty$, à $\phi^n/\sqrt{5}$ où $\phi = (1+\sqrt{5})/2$ est le nombre d'or.
4. Écrire une fonction `fibo` qui prend en argument un entier naturel n et renvoie le nombre de Fibonacci F_n. Par exemple, `fibo(6)` renverra 8.
5. En utilisant la fonction `euclide`, écrire une fonction `gcd_trois` qui renvoie le pgcd de trois entiers naturels. Par exemple, `gcd_trois(18, 30, 12)` renverra 6.

4 *Extrait de CCP 2017 Maths 2 : Calcul effectif de la probabilité invariante d'une matrice stochastique strictement positive*

Si A est une matrice stochastique strictement positive, on a établi dans la partie précédente la convergence de la suite $(\mu_n)_{n\in\mathbb{N}}$ associée à la matrice A. Ceci fournit un algorithme de calcul de la probabilité invariante par A. On en propose une implémentation

1. La récursivité sera vue prochainement...

en langage Python. On sera très attentif à la rédaction et notamment à l'indentation du code.

Un vecteur x de $\mathbb{R}^p$ sera représenté en Python par une liste de flottants. Par exemple, le vecteur $x=(1,2,3)$ de $\mathbb{R}^3$ sera représenté par la liste `[1,2,3]`. De même, une matrice A sera représentée par une liste dont les éléments sont les lignes de la matrice. Par exemple, la matrice $A=\left(\begin{smallmatrix}1&2&3\\4&5&6\end{smallmatrix}\right)$ sera représentée par la liste `[[1,2,3],[4,5,6]]`.

1. On exécute le script suivant `A=[[1,2,3],[4,5,6],[7,8,9],[10,11,12]]` qui représente la matrice $A=\left(\begin{smallmatrix}1&2&3\\4&5&6\\7&8&9\\10&11&12\end{smallmatrix}\right)$. Donner les valeurs renvoyées lorsque l'on exécute `len(A),A[1]` et `A[2][1]`.
2. Écrire une fonction `difference` qui prend en arguments deux vecteurs x et y de même taille et renvoie le vecteur $x-y$. Par exemple si $x=(5,2)$ et $y=(3,7)$, `difference(x,y)` renverra `[2,-5]`.
3. Écrire une fonction `norme` qui prend en argument un vecteur $x=(x_1,\dots,x_p)$ et renvoie sa norme infinie $\|x\|_\infty=\max\{|x_i| \mid i\in[\![1,p]\!]\}$ (on pourra utiliser librement la fonction `abs` qui renvoie la valeur absolue d'un nombre, mais on s'interdit l'utilisation de la fonction `max` déjà implémentée dans Python).
4. Écrire une fonction `itere` qui prend en arguments un vecteur ligne x et une matrice carrée de même taille que x et qui renvoie le vecteur xA. Par exemple si $x=(1,1)$ et $A=\left(\begin{smallmatrix}1&2\\4&5\end{smallmatrix}\right)$, on a $xA=(5,7)$ et donc `itere(x,A)` renverra `[5,7]`.
5. On a vu, dans la partie IV, que si A est une matrice strictement positive, la suite de vecteurs lignes de $\mathbb{R}^p$ associée $(\mu_n)_{n\in\mathbb{N}}$ définie par la relation : $\forall n\in\mathbb{N},\ \mu_{n+1}=\mu_nA$ convergeait vers un vecteur μ_∞ indépendant du choix de μ_0 vecteur stochastique.

   Écrire une fonction `probaInvariante` qui prend en arguments une matrice stochastique strictement positive A de $\mathcal{M}_p(\mathbb{R})$ et un réel $\varepsilon>0$ et qui renvoie le premier terme μ_k de la suite $(\mu_n)_{n\in\mathbb{N}}$ avec $\mu_0=\left(\frac{1}{p},\frac{1}{p},\dots,\frac{1}{p}\right)$ tel que $\|\mu_k-\mu_{k-1}\|_\infty\leqslant\varepsilon$.

   On ne demandera pas à l'algorithme de vérifier que la matrice passée en argument est bien stochastique et strictement positive.

   Par exemple, si $A=\left(\begin{smallmatrix}1/2&1/2\\1/4&3/4\end{smallmatrix}\right)$ et $\varepsilon=10^{-6}$, `probaInvariante(A,eps)` renverra

   `[0.33333396911621094, 0.6666660308837891].`

5 Écrire une fonction `plusfrequent(T)` qui reçoit un tableau `T` et renvoie un couple `(a, i)` où `a` est l'un des éléments de `T` figurant le plus souvent dans `T` et `i` son nombre d'occurrences. Justifier la correction de la fonction et calculer la complexité temporelle et la complexité spatiale. Proposer une amélioration lorsque l'on suppose que les éléments de `T` sont des entiers naturels.

6 Expliquer les complexités de chaque opération sur les listes python et réécrire chacune d'entre elle avec cette même complexité.