TP 1 : Structures linéaires

Exercice 1. 1. Écrire une fonction `touille` d'argument une liste `L` renvoyant une liste commençant par le dernier élément de `L`, puis le premier, puis l'avant-dernier, etc. Ainsi,

`touille([1,6,5,3,8]) = [8,1,3,6,5])` et `touille([1,6,5,4,3,8]) = [8,1,3,6,4,5])`.

2. Montrer qu'après six itérations de `touille` appliquées à la liste `[1,2,3,4,5,6]`, on retombe sur [1,2,3,4,5,6].

3. Écrire une fonction `ordre` d'argument n renvoyant le minimum d'itérations de `touille` pour retomber sur L en partant de la liste contenant les entiers entre 1 et n rangés par ordre croissant. La tester pour $n = 7$.

4. Quels sont les entiers n entre 1 et 100 tels qu'en partant de la liste contenant les entiers entre 1 et n rangés par ordre croissant, on retombe sur L après n itérations ?

5. Représenter graphiquement sous forme ponctuelle $\texttt{ordre}(n)/n$ en fonction de `n` pour $1 \leqslant n \leqslant 200$.

Exercice 2. Un numismate collectionne des pièces romaines. Il y a N pièces différentes, d'égale rareté (modèle hautement réaliste).

1. Le collectionneur reçoit n pièces au hasard. Écrire une fonction de paramètres n et N retournant les types de pièces qu'il a reçues sans comptabiliser les doublons.

2. Calculer, pour dix mille essais et $N = n = 10$, la proportion de collections complètes. La comparer avec la valeur théorique.

3. Écrire un fonction d'argument N renvoyant la plus petite valeur de n assurant d'obtenir une collection complète avec une probabilité de 0,9, puis tester cette fonction pour $N = 5$, $N = 10$ et $N = 20$.

a. En utilisant un test sur 1000 tirages.

b. En utilisant la formule théorique donnant le nombre de surjections $S_{n,p}$ d'un ensemble à p éléments sur un ensemble à N éléments :

$$S_{n,p} = \sum_{k=1}^{N} (-1)^{N-k} \binom{N}{k} k^n.$$

Exercice 3. 1. Lire le fichier `pi5.txt`, qui contient des décimales de π.

2. Lire les dix premières décimales données, ainsi que les dix dernières. Combien de décimales contient le fichier ?

3. Écrire une fonction testant si une suite de décimales donnée contient une séquence donnée (par exemple, si 567341 contient 34, ce qui est le cas). Tester la fonction avec la séquence 2015 dans la liste des décimales de π fournie.

4. Déterminer la fréquence de la séquence de deux chiffres la plus élevée et celle qui est la plus faible, ainsi que les séquences correspondantes. On étudie ici la suite des décimales de π.

Exercice 4. Cet exercice présente le test de Miller-Rabin. Ce test probabiliste permet de déterminer avec une probabilité très proche de 1 si un nombre entier donné est premier. Il est utilisé en cryptographie.

1. Écrire une fonction qui prend un entier n en entrée et renvoie le couple d'entiers (s, d) tel que $p - 1 = 2^s d$, s étant la plus grande valeur possible.

On peut démontrer que si n est premier, alors, avec les valeurs s et d renvoyées par la fonction précédente, pour tout $a \in [\![1, n-1]\!]$:

– ou bien $a^d - 1$ est multiple de n ;

– ou bien il existe $r \in [\![0, s-1]\!]$ tel que $a^{2^r d} + 1$ soit multiple de n.

2. Écrire une fonction prenant comme arguments a et n et testant si la propriété ci-dessus est vérifiée ou non. Cette fonction renvoie un booléen. Estimer la complexité de cet algorithme.

3. Écrire une fonction `MillerRabin(n)` qui choisit au hasard six valeurs de a entre 1 et $n-2$ et qui applique le test précédent.

On considère que si n passe ce test, il est premier avec une marge d'erreur probabiliste de l'ordre de 10^{-80}. On conjecture que si l'on remplace 6 valeurs au hasard par tous les entiers inférieurs à $2(\ln n)^2$, alors il est vraiment premier.

Exercice 5. Les suites de Syracuse sont définies de manière récurrente à partir de $u_0 \in \mathbb{N}^*$ par la formule $u_{n+1} = \begin{cases} u_n/2 & \text{si } n \text{ est pair} \\ 3u_n + 1 & \text{sinon.} \end{cases}$

1. Définir une fonction `prochain` d'argument m, qui renvoie l'image de m par la fonction ci-dessus (autrement dit, le terme suivant m d'une suite de Syracuse).

2. Écrire une fonction d'arguments u_0 et m qui affiche successivement les éléments u_k pour $0 \leqslant k \leqslant m$. Tester cette fonction pour quelques valeurs.

Selon une célèbre conjecture, pour toute valeur de u_0, la suite $(u_n)_n$ est ultimement périodique de la forme $(u_0, u_1, \ldots, u_{n_0-1}, \overline{1, 4, 2}, \ldots)$. Ici, n_0 est le plus petit indice n tel que $u_n = 1$. On admettra la validité de cette conjecture dans la suite.

3. Écrire une fonction `vol` d'argument u_0 affichant un tuple de longueur 2 dont le premier élément est, avec les notations ci-dessus, la liste $[u_0, u_1, \ldots, u_{n_0-1}, 1]$ (dite *orbite* de la suite) et le deuxième, l'entier n_0, appelé *temps de vol* de la suite.

4. Afficher le graphe des temps de vol des suites $(u_n)_{n \geqslant 0}$ pour $u_0 \in [\![1, 300]\!]$.

5. Dresser un histogramme des valeurs de u_n pour $u_0 \in [\![1, 25]\!]$.

6. Créer un dictionnaire dont les clefs sont les entiers u_0 et les valeurs les orbites correspondantes calculées par la fonction `vol` et le remplir bêtement pour $1 \leqslant u_0 \leqslant 25$ (il s'agit d'une unique opération).

7. Expliquer le terme « bêtement » de la question précédente s'il s'agissait de remplir le dictionnaire pour, disons, 10^3 entiers. La suite de la question s'attache à construire le même dictionnaire « intelligemment ».

Écrire une fonction permettant de calculer l'orbite de u_0 et utilisant le dictionnaire supposé rempli jusqu'à la valeur initiale $u_0 - 1$. Remplir le dictionnaire pour $u_0 \leqslant 10^3$. Est-ce si intelligent que ça ?

8. En utilisant le dictionnaire précédent, déterminer le plus long temps de vol enregistré et pour quelle valeur de u_0 il se produit.

9. Créer un nouveau dictionnaire ayant toujours les u_0 comme clefs et les temps de vol comme valeurs. Remplir le dictionnaire pour $u_0 \leqslant 10^5$.

Rappel de quelques commandes utiles :

`import matplotlib.pyplot as plt` charge la librairie graphique.
`plt.scatter(X,Y)` trace les points de coordonnées (x_i, y_i).

`plt.plot(X,Y)` trace la courbe reliant les points de coordonnées (x_i, y_i).

`import random as rand` charge la librairie de probas.
`rand.random()` génère un nombre flottant aléatoire dans $[0, 1]$.
`rand.randint(a,b)` génère un nombre entier aléatoire dans $[\![a, b]\!]$.

`from sympy import binomial` pour calculer les coefficients du binôme : `binomial(n,p)`$=\binom{n}{p}$