TP noté

PTSI Lycée Eiffel

20 décembre 2019

Ce TP noté se propose de modéliser sommairement et d'étudier quelques propriétés de structures obtenues à partir d'empilements en deux dimensions de petites particules (du type de tas de sable). Certaines fonctions demandées dans le sujet faisant intervenir des comportements aléatoires, on rappelle que l'intégralité des fonctions du module **random** peuvent être importées via la commande suivante :

```
from random import *
```

et qu'elles peuvent ensuite être utilisées sans rappeler le nom de leur module d'origine. Parmi les fonctions disponibles dans ce module, seules les fonctions **random()** (qui renvoie un nombre flottant aléatoire dans l'intervalle $[0, 1]$) ou **randint(a,b)** (qui renvoie un nombre entier aléatoire compris entre a et b, les bornes a et b étant toutes les deux incluses) devraient pouvoir servir.

Sauf mention du contraire, aucune méthode ou fonction permettant de manipuler les listes n'est autorisée, à l'exception de la fonction **len** et de la méthode **append**. En particulier, il est hors de question d'utiliser des commandes comme **sum** ou **sort** qui rendraient totalement immédiate la résolution de certaines questions. Les différentes parties du sujet sont complètement indépendantes (en particulier, les fonctions demandées dans les deux premières parties ne seront pas utilisées par la suite).

I. Quelques fonctions classiques sur les listes d'entiers.

Question 1 : Écrire une fonction Python **croissante** qui prend comme argument une liste de nombres et qui renvoie True ou False selon que la liste est ou non triée dans l'ordre croissant.

Exemple : La commande croissante([1,3,5,5,7]) doit renvoyer True.

Question 2 : Écrire une fonction Python **somme** qui prend comme argument une liste de nombres et qui renvoie la somme de ces nombres.

Exemple : La commande somme([1,4,-2,3]) doit renvoyer 6.

Question 3 : Écrire une fonction Python **indiceszeros** qui prend comme argument une liste de nombres et qui renvoie la liste (éventuellement vide) des indices i pour lesquels $l[i]$ est égal à 0.

Exemple : La commande indiceszeros([1,3,0,5,0,0,7]) doit renvoyer [2,4,5].

II. Quelques fonctions faisant intervenir les probabilités.

Question 4 : Écrire une fonction Python **lancerde** qui prend comme argument un entier non nul n et qui renvoie une simulation de lancer de dé à n faces, c'est-à-dire un entier aléatoire entre 1 et n.

Question 5 : Écrire une fonction Python **simulde** qui prend comme arguments deux entiers n et k et qui effectue k simulations de lancers de dé à n faces, en stockant dans une liste le nombres de fois que chaque face du dé a été obtenue. La fonction renverra bien sûr cette liste.

Exemple : La commande simulde(6,10) peut renvoyer la liste [1,3,0,2,1,3] si lors des dix lancers de dé à six faces, on a obtenu une fois 1, trois fois 2, jamais 3, etc.

Question 6 : Écrire une fonction Python **premiersix** qui ne prend aucun argument et qui effectue des simulations de lancer de dé à six faces jusqu'à obtenir un 6. La fonction renverra le nombre d'essais nécessaires avant d'obtenir ce premier 6.

Question 7 : Écrire une fonction Python **moyennesix** qui prend comme argument un entier k et qui effectue k fois de suite l'expérience de la question précédente (on compte le nombre d'essais nécessaires avant d'obtenir un 6). La fonction doit cette fois renvoyer le nombre moyen de tentatives nécessaires pour obtenir un 6 lors de ces k expériences.

III. Modélisation de tas de sable.

On va dans cette dernière partie modéliser un tas de sable bidimensionnel de façon très rudimentaire par une liste Python : une liste d'entiers naturels modélisera simplement un tas de sable constitué de colonnes de grains de sables empilés verticalement, les entiers de la liste indiquant le nombre de grains dans chaque colonne. Ainsi, la liste [1,2,3,2,1] modélise un tas pyramidal : un seul grain dans la première colonne, deux grains dans la suivante, trois dans la troisième (qui est la plus haute), puis on redescend. On a le droit d'avoir des colonnes vides (donc des 0 à l'intérieur de la liste). Cette modélisation est bien évidemment extrêmement rudimentaire et peu réaliste.

Question 8 : Écrire une fonction Python **graphetas** qui prend comme argument une liste d'entiers l et qui renvoie une représentation graphique du tas de sable correspondant (on utilisera le module **matplotlib.pyplot** et on se contentera de relier les points de coordonnées $(i, l[i])$).

Une même liste peut en fait contenir plusieurs piles de grains de sable si certains de ses éléments sont égaux à 0. Toute suite d'éléments non nuls (éventuellement entourés de 0) à l'intérieur d'une liste représente donc une pile. Un tas de sable est **connexe** s'il est constitué d'une seule pile.

Question 9 : Écrire une fonction Python **connexe** qui prend comme argument une liste d'entiers l et qui renvoie True ou False selon que le tas est connexe ou non.

Exemple : La commande connexe([0,0,1,3,2,4,3,1]) doit renvoyer True.

Question 10 : Écrire une fonction Python **nbrepiles** qui prend comme argument une liste d'entiers l et qui renvoie le nombre de piles de sable contenus dans le tas représenté par cette liste.

Exemple : La commande nbrepiles([1,3,2,1,0,0,1,2,2,0,1]) doit renvoyer la valeur 3.

Un **sommet** à l'intérieur d'un tas de sable correspond à une colonne strictement plus haute que chacune des ses deux voisines. Les colonnes situées complètement à gauche et à droite du tas de sable ne peuvent pas être considérées comme des sommets.

Question 11 : Écrire une fonction Python **nbresommets** qui prend comme argument une liste d'entiers l et qui renvoie le nombre de sommets du tas de sable correspondant.

Un tas de sable est considéré comme **stable** si l'écart du nombre de grains contenu dans deux colonnes successives n'est jamais strictement supérieur à 2. Autrement dit, si par exemple une colonne de notre tas de sable contient 5 grains, les colonnes adjacentes doivent contenir entre 3 et 7 grains de sable pour que le tas soit stable.

Question 12 : Écrire une fonction Python **stable** qui prend comme argument une liste d'entiers l et qui renvoie True ou False selon que le tas est stable ou non.

Exemple : La commande stable([1,2,4,7,6,4,5,2]) doit renvoyer False.

Si un tas de sable n'est pas stable, on peut le stabiliser en appliquant la procédure suivante : on parcourt la liste, et à chaque fois qu'un écart plus grand que 2 est constaté entre les hauteurs de deux colonnes successives, on enlève un grain de la pile la plus haute pour le transférer sur la pile la plus basse. On recommence cette procédure jusqu'à obtenir un tas de sable stable. On admet que cette procédure finira toujours par donner un tas stable !

Question 13 : Écrire une fonction Python **stabilise** qui prend comme argument une liste d'entiers l et qui renvoie la liste représentant le tas de sable stabilisé correspondant

Exemple : La commande stabilise([0,0,10,0,0]) doit renvoyer [1,2,4,2,1].

On souhaite désormais ajouter un grain de sable à un tas de sable déjà existant. Si on tente d'ajouter un grain dans la colonne i du tas de sable, il va se produire les ajustements suivants :

- si la colonne i avait avant l'ajout du grain supplémentaire une hauteur inférieure ou égale à chacune de ses deux voisines (ou à celle de sa voisine s'il s'agit de la première ou de la dernière colonne), le grain supplémentaire est simplement ajouté à la colonne i.
- si la colonne i contient strictement plus de grains qu'une seule de ses voisines, le grain de sable va s'ajouter à la colonne i avec probabilité $\frac{1}{2}$, et « tomber » dans la colonne voisine avec probabilité $\frac{1}{2}$. Attention, on ne s'arrête pas là, le grain peut ensuite à nouveau tomber dans une autre colonne contenant encore moins de grains, et ainsi de suite jusqu'à ce que le grain se stabilise dans une colonne.
- si la colonne i contient strictement plus de grains que chacune de ses deux voisines, le grain supplémentaire peut s'ajouter à la colonne i ou tomber dans chacune des deux voisines avec probabilité $\frac{1}{3}$ à chaque fois. Là encore, le grain peut éventuellement tomber plusieurs fois avant de se stabiliser.

Question 14 : Écrire une fonction Python **ajoutegrain** qui prend comme arguments une liste d'entiers l et un indice i et qui ajoute un grain dans la colonne i du tas de sable en respectant les indications précédentes.

Question 15 : Écrire une fonction Python **simulationtas** qui prend comme arguments deux entiers n et k et qui crée un tas de sable initialement vide de longueur n, puis qui y ajoute k grains de sable dans des colonnes choisies aléatoirement à chaque fois. La fonction tracera l'allure du tas de sable obtenu avant de renvoyer la liste correspondante.