TP n° 5 : Encore de la récursivité, et des probas.

MP Lycée Camille Jullian

22-29 janvier 2022

Exercice 1 : Quelques tests de complexité.

On désire vérifier expérimentalement la complexité de certains algorithmes vus en cours. Pour cela, on mesurera le temps d'exécution de nos programmes à l'aide de la commande **time** qu'on peut importer depuis le module **time**. Cette commande mesure très précisément la date au moment où elle est lancée. Pour savoir le temps d'exécution d'un programme, on stocke donc le temps via un a=time(), on lance le programme, puis on affiche print(time()-a) pour savoir le temps qui s'est écoulé entre les deux mesures de la date.

1. Écrire les algorithmes de calcul de puissance évoqués dans le cours (de façon récursive tant qu'à faire), avec une version classique (on effectue $n-1$ multiplications par x), et une version rapide (en exploitant la division par 2 de la puissance), puis tester la rapidité d'exécution des deux programmes (on pourra par exemple comparer le temps d'exécution pour calculer $42^{9}42$, ne pas hésiter à faire tourner l'algorithme beaucoup de fois pour avoir un temps d'exécution à peu près fiable car ça va très vite en pratique, on ne peut pas augmenter suffisamment n pour le premier programme sans dépasser la capacité de la Pile de récursion).
2. Écrire l'horrible programme récursif calculant les termes de la suite de Fibonacci de façon théoriquement exponentielle. Essayer de vérifier que le temps est réellement exponentiel. On évitera d'aller trop haut au niveau des valeurs de n, et on pourra par exemple mesurer les quotients $\dfrac{C(n+1)}{C(n)}$ pour différentes valeurs de n (ici $C(n)$ désignera simplement le temps d'exécution) pour tester l'hypothèse.

Exercice 2 : Un peu d'entrainement supplémentaire sur la récursivité.

Les questions sont indépendantes, et le but est bien sûr d'écrire des programmes récursifs à chaque fois.

1. Écrire un programme permettant de calculer la valeur de u_n pour une suite définie par $u_0 = 1$ et $\forall n \in \mathbb{N}$, $u_{n+1} = \dfrac{1}{2}\left(u_n + \dfrac{2}{u_n}\right)$ (la suite converge très rapidement vers $\sqrt{2}$, ce qui doit permettre de vérifier facilement le programme).
2. Écrire un programme récursif effectuant la conversion en base 2 d'un entier décimal (on essaiera de renvoyer un entier ou une chaîne de caractères et pas seulement la liste des chiffres).
3. Les nombres de Catalan sont définis par la formule suivante : $C_0 = 1$ et $\forall n \in \mathbb{N}$, $C_{n+1} = \displaystyle\sum_{k=0}^{n} C_k C_{n-k}$. Écrire un programme permettant de calculer C_n (on essaiera d'en trouver une version qui ne soit pas de complexité exponentielle, quitte à calculer la liste de tous les nombres de Catalan jusqu'à C_n au lieu de simplement calculer C_n).

Exercice 3 : Un classique amusant.

On souhaite compter le nombre de chemins différents permettant de traverser une grille $n \times p$ en partant du coin inférieur gauche pour arriver au coin supérieur droit, en effectuant à chaque étape un déplacement vers le haut ou vers la droite. Par exemple, pour $n = p = 2$, il y a six possibilités. On notera $c(n, p)$ le nombre de chemins possibles.

1. Expliquer pourquoi on a la relation de récurrence suivante : $c(n, p) = c(n-1, p) + c(n, p-1)$, avec $c(n, 0) = c(0, p) = 1$.
2. Écrire un programme récursif naïf calculant $c(n, p)$ à partir de cette relation. Jusqu'à quelles valeurs est-il fonctionnel (on prendra $n = p$ pour les tests) ?
3. On suppose connues les valeurs de $c(i, j)$ pour tout couple d'entiers i, j tous deux inférieurs ou égaux à n. Expliquer comment calculer $c(n+1, n+1)$ (ainsi que les $c(n+1, j)$ et $c(i, n+1)$ avec $i \leqslant n+1$ ou $j \leqslant n+1$) en effectuant simplement quelques additions (et en précisant surtout l'ordre dans lequel on fera lesdites additions).
4. En déduire un programme récursif renvoyant une matrice contenant toutes les valeurs de $c(i, j)$ pour $i \leqslant n$ et $j \leqslant n$, de complexité quadratique.
5. Trivialiser le calcul de $C(n, p)$ à l'aide d'un raisonnement de dénombrement mathématique (question facultative).

Exercice 4 : Des histoires de lancers de dés ou de pièces.

Dans tout l'exercice on s'intéressa à un dé à six faces, même si avec Python il est tout aussi facile de traiter le cas d'un dé à n faces.

1. Tester la loi des grands nombres en écrivant un programme **moyennede(n)** qui effectue n lancers de dés et affiche la moyenne des résultats obtenus (on utilisera bien sûr les fonctions du module random).
2. Écrire un programme **premiersix()** effectuant des simulations de lancers de dé jusqu'à obtenir un 6.
3. Écrire un programme **geometrique(n)** qui effectue n fois de suite la simulation précédente et stocke dans une liste le nombre de fois où on a obtenu chaque valeur (je vous laisse gérer le plus intelligemment possible les problèmes évidents auxquels vous allez être confrontés). Vérifier en particulier que le nombre de fois où on a obtenu la valeur 1 correspond à peu près à la probabilité théorique quand n est suffisamment grand.
4. Utiliser matplotlib pour tracer un histogramme des valeurs précédentes, et vérifier que cela correspond à ce que vous imaginez pour une loi géométrique.
5. Faire le même genre d'étude pour un lancer de pièces équilibrée où on cherche à représenter le nombre de Piles obtenus sur n lancers.

Exercice 5 : Marche aléatoire en deux dimensions.

On souhaite simuler une marche aléatoire en deux dimensions : un objet se trouve initialement à la position $(0, 0)$ dans le plan et peut avancer dans une des quatre directions (haut, bas, gauche et droite) avec probabilité $\frac{1}{4}$ à chaque étape. Écrire un programme Python simulant cette marche aléatoire. Tracer la trajectoire obtenue à l'aide de matplotlib.pyplot. On pourra ensuite affiner le programme pour qu'il affiche également le nombre de fois où la marche est repassée par son point de départ en cours de route.

Exercice 6 : Un classique perturbant.

On effectue l'expérience aléatoire suivante : une urne contient initialement une boule bleue et une boule rouge. On effectue des tirages successifs dans l'urne jusqu'à tirer la boule rouge, sachant qu'à chaque fois qu'on tire une boule bleue, on la remet dans l'urne et qu'on **ajoute** une boule bleue supplémentaire dans l'urne. On note X la variable aléatoire donnant le nombre de tirages nécessaires avant d'obtenir la boule rouge.

1. Déterminer la loi de X. Quelle est la probabilité de ne jamais tirer la boule rouge ? La variable X admet-elle une espérance ? Oui, je sais, c'est une question de maths, désolé...
2. Écrire un programme Python effectuant une simulation de cette expérience aléatoire, puis répéter cette simulation un certain nombre de fois (on se contentera de stocker sous forme de liste les différentes valeurs obtenues pour X lors des simulations). Vérifier que la fréquence de 1 obtenue pour les valeurs de X correspond à la probabilité théorique.
3. Modifier le programme précédent pour qu'il affiche l'espérance obtenue sur n simulations. Tester pour différentes valeurs de n. Commenter les résultats obtenus.

Exercice 7 : Tracé de fractales.

On termine avec un peu de détente pour ceux qui ont trouvé le reste du TP trop facile. Le flocon de von Koch est un des exemples les plus célèbres de courbes fractales. Il est obtenu en partant d'un triangle équilatéral et en appliquant récursivement la procédure suivante : sur chaque côté du triangle, on efface le tiers du segment situé au milieu, et on le remplace par les deux autres côtés d'un triangle équilatéral dont la base serait le segment effacé (cf illustration). Le but est simplement d'écrire un programme Python permettant de tracer l'allure du flocon obtenu après n étapes de récursion (techniquement, la courbe de von Koch est bien sûr la **limite** obtenue en faisant tendre n vers $+\infty$, mais on ne peut pas vraiment tracer ça avec Python !). Il est fortement conseillé d'écrire un programme récursif utilisant une fonction auxilaire du type **koch(n,a,b)**, où n est le nombre d'étapes de récursion, et a et b les coordonnées des extrémités du segment auquel appliquer les n étapes.