TP n° 5 : corrigé.

MP Lycée Camille Jullian

22-29 janvier 2022

Exercice 1 : Quelques tests de complexité.

1. Bien entendu, le programme intelligent va plus vite que l'autre, mais il faut donner de très grandes valeurs à n pour mieux voir la différence (sur mon exemple avec $n = 42$, un facteur 5 environ entre les deux versions).
2. Le programme que j'ai écrit met un peu de temps à renvoyer les valeurs. Sur mon ordinateur, on observe que les quotients sont relativement proches, entre 1.5 et 2 en gros. La valeur théorique devrait être d'environ 1.6 (la valeur du nombre d'or).

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

1. Rien à signaler, le programme est vraiment très basique.
2. J'ai ici écrit un programme vraiment très rapide qui consiste tout simplement à exploiter que tout nombre pair terminera par un 0 et tout nombre impair par un 1 en écriture binaire, puis à calculer les chiffres précédants ce 0 ou 1 final en appliquant récursivement l'algorithme à $n//2$, ce qui donne bien le résultat correct. Les multiplications par 10 servent juste à écrire les chiffres correspondants avant le chiffre final. En fait, j'affiche des entiers écrits en base 10 tout à fait classiquement, mais qui ne contiendront par construction que des 0 ou des 1.
3. J'ai écrit deux programmes, l'un stupide (son nom est assez éclairant) et l'autre malin, qui évite de refaire en permanence des calculs déjà effectués (pour ce dernier, j'ai exploité une astuce de programmation classique consistant à écrire une fonction auxilaire qui calcule la liste de tous les nombres jusqu'à c_n, pour finalement n'afficher que le dernier d'entre eux). La comparaison entre les deux est impressionnante : on a l'impression qu'ils font les mêmes calculs, mais la première version est incapable de calculer au-delà de c_{15} sans que ça ne prenne beaucoup de temps, alors que l'autre version affiche c_{900} sans grande difficulté (on ne peut pas aller beaucoup au-delà à cause de la limite d'étapes de récursion).

Exercice 3 : Un classique amusant.

1. Si on compte tous les chemins, on peut les séparer en deux catégories disjointes : ceux qui se terminent par un déplacement vers le haut, et qui arrivent donc en position $(n, p-1)$ avant ce dernier déplacement (il y en a donc $c(n, p-1)$ puisque chaque chemin menant du point de départ au point de coordonnées $(n, p-1)$ peut être complété de façon unique par un déplacement vers le haut pour en faire un chemin convenable), et ceux qui terminent par un déplacement vers la droite, et arrivent donc en position $(n-1, p)$ avant ce dernier déplacement (il y en a $c(n-1, p)$ pour la même raison). On obient la formule demandée en additionnant simplement les deux valeurs obtenues. Bien sûr, si $n = 0$ ou $p = 0$, il n'y a qu'un seul chemin rectiligne possible.
2. Il s'agit du programme cheminsbete du corrigé. On commence à attendre un certain temps à partir de $n = p = 12$ (peut-être un peu plus si vous avez un meilleur ordi que le mien). Le programme est inutilisable avant d'atteindre $n = p = 20$.

3. On commence par exemple par compléter la dernière colonne jusqu'à hauteur n, donc par calculer $c(n+1,j)$ pour $j \leqslant n$, à l'aide de la formule obtenue à la première question (en les calculant par ordre croissant de la valeur de j, on a toujours les deux valeurs nécessaires à disposition). Ensuite (ou avant, peu importe), on calcule les $c(i,n+1)$ de la même façon en débutant à $i=0$. Cette fois-ci, on peut aller jusqu'au calcul de $c(n+1,n+1)$ sans difficulté.
4. Le programme matchemin effectue exactement ce qui est demandé. J'ai ensuite programmé une fonction supplémentaire qui ne renvoie que la valeur de $c(n,n)$ pour tester sans avoir un affichage super lourd. Avec ce nouveau programme, on obtient l'affichage de $c(500,500)$ quasiment instananément. Le temps d'exécution est un tout peu plus long pour $n=900$ mais reste nettement en-dessous de la seconde. On ne peut pas aller beaucoup plus loin car on dépasse ensuite la taille de la Pile de récursivité de Python.
5. Un chemin menant du point de départ au point d'arrivée est toujours constitué d'une liste de $n+p$ déplacements : n déplacements vers la droite et p déplacement vers le haut. Il suffit donc de choisir la position des n déplacements vers la droite (ou celle des p déplacements vers le haut) dans cette liste pour décrire un déplacement. Il y a donc au total $\binom{n+p}{n} = \frac{(n+p)!}{n!p!}$ chemins possibles.

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

1. On obtient sans surprise une valeur moyenne se rapprochant de 3.5 (une silation effectuée avec dix millions de lancers me donne par exemple une moyenne de 3.499662).
2. On exploite bien sûr un while ici. Attention à initialiser correctement pour renvoyer la valeur 1 et pas 0 quand on obtient immédiatement un 6.
3. J'ai écrit un programme ne stockant volontairement que les valeurs inférieures à 20, mais on peut bien sûr le modifier pour aller plus loin. Par contre, si on veut que la liste adapte automatiquement sa longueur à la plus grande valeur obtenue lors des simulations, c'est un peu plus technique à programmer. On devrait obtenir la valeur 1 un sixième du temps, ce qui est confirmé par les simulations.
4. Il existe une fonction matplotlib permettant de tracer directement un histogramme à partir de la liste des valeurs obtenues (et non la liste des fréquences telle que calculée à la question précédente), du coup j'ai écrit une nouvelle version du programme. On obtient de beaux histogrammes « exponentiels décroissants » comme la théorie le prévoit.
5. On écrit donc un programme effectuant une simulation de cette expérience puis on trace l'histogramme correspondant. Le paramètre n représente le nombre de lancers effectués à chaque simulation, le paramètre p le nombre de simulations effectuées.

Exercice 5 : Marche aléatoire en deux dimensions.

Le programme est assez facile à écrire : à chaque étape, on tire un entier aléatoire entre 1 et 4, qui donne la direction du prochain déplacement. Dans mon programme, les listes a et o représentent l'abscisse et l'ordonnée des points atteints. J'ajoute un compteur c pour déterminer le nombre de passages par l'origine. En pratique, on repasse très rarement par le point de départ (une ou deux fois sur une marche de 1000 pas, typiquement).

Exercice 6 : Un classique perturbant.

1. La variable X peut prendre toutes les valeurs entières non nulles. On a clairement $\mathbb{P}(X=1) = \frac{1}{2}$ (initialement, il y a deux boules dans l'urne, dont une seule rouge), puis $\mathbb{P}(X=2) = \frac{1}{2} \times \frac{1}{3} = \frac{1}{6}$ (on doit tirer la boule bleue au premier tirage, puis la rouge au deuxième,

mais dans une urne contenant désormais deux boules bleues et une boule bleue). De façon générale, si on note B_n et R_n les évènements « on tire une boule bleue au n-ème tirage » et « on tire une boule rouge au n-ème tirage », on aura $\mathbb{P}(B_n) = \dfrac{n}{n+1}$ (on a déjà ajouté $n-1$ boules bleues dans l'urne, qui contient donc désormais une boule rouge et n boules bleues), et $\mathbb{P}(R_n) = \dfrac{1}{n+1}$. L'évènement $X = n$ étant égal à $\left(\bigcap_{i=1}^{n-1} B_i\right) \cap R_n$, donc $\mathbb{P}(X = n) = \prod_{k=1}^{n-1} \dfrac{k}{k+1} \times \dfrac{1}{n+1} = \dfrac{(n-1)!}{(n+1)!} = \dfrac{1}{n(n+1)}$. La variable X admet donc une espérance si la série de terme général $n \times P(X = n) = \dfrac{1}{n+1}$ converge, ce qui n'est pas le cas.

2. On devrait obtenir 1 une fois sur deux, ce qui est évidemment vérifié en pratique (498 fois sur 1 000 tests à mon premier essai).
3. Bien sûr, on obtient toujours une moyenne finie. Pire, on n'a pas vraiment l'impression que cette moyenne augmente régulièrement avec le nombre n d'essais effectués. Par contre, on note des fluctuations surprenantes, ayant de temps à autre une moyenne nettement plus élevée qui apparait (pour une même valeur de n). C'est d'ailleurs le cas si on affiche la liste des résultats des différentes simulations : beaucoup de valeurs petites, et une fois de temps à autres, une valeur exceptionnellement grande. Dans une liste de 100 simulations, j'obtiens pas exemple onze valeurs supérieures à 10 : deus fois 11, trois fois 14, une fois 24, une fois 26, une fois 34, une fois 47, une fois 97 et enfin un spectaculaire 4 145.

Exercice 7 : Tracé de fractales.

Je vous laisse admirer mon superbe programme. La première fonction découpe le segment $[a, b]$, en créant trois nouveaux points (le point c est au tiers du segment, le point e aux deux tiers du segment, et d est le sommet d'un triangle équilatéral dont la base est le segment $[c, e]$, les calculs ne devraient pas être très compliqués à comprendre même si l'emploi d'affixes complexes simplifierait la chose). Ensuite, la deuxième fonction applique simplement la précédente à trois segments reliant les sommets d'un triangle équilatéral. L'option imposée sur les axes sert à tracer un dessin homogène en termes d'échelle. Attention, la valeur à donner à n ne doit pas être grande, la complexité est exponentielle ! À partir de $n = 7$ ça commence à prendre beaucoup de temps (et de toute daçon on ne voit plus grand chose).