DS 5

Informatique de tronc commun, première année

Julien Reichert

Exercice 1 : Écrire une fonction prenant en entrée une liste de listes `ll` et qui retourne la liste des couples d'indices autorisés dans cette liste. L'ordre n'est pas important.

Par exemple, la fonction retourne `[(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (2, 0)]` pour la liste `[[2, 5, 4, 1], [6, 4], [3]]`.

Exercice 2 : Écrire une fonction prenant en entrée une liste et qui détermine si elle est monotone. **Il est interdit de trier la liste, la complexité doit être linéaire (pas besoin de le prouver).**

Par exemple, la fonction retourne `True` pour les listes `[2, 3, 6]` et `[6, 4, 1]`, ainsi que pour les listes constantes, mais `False` pour les listes `[1, 5, 3]` et `[5, 3, 6]`.

Exercice 3 : Écrire une fonction prenant en entrée une chaîne de caractères et qui détermine le caractère le plus fréquent. La complexité peut être quelconque **mais il faudra la calculer**. En cas d'égalité le comportement est au choix parmi renvoyer tous les caractères à égalité pour le nombre maximal d'occurrences, le premier rencontré, le plus petit dans l'ordre lexicographique, voire n'importe lequel d'entre eux.

Par exemple, la fonction retourne `'o'` pour la chaîne `"Bonjour le monde"`. Noter qu'une espace peut être le caractère le plus fréquent, de même que des caractères non alphanumériques plus généralement. Il faut bien entendu considérer les lettres capitales et les lettres minuscules comme des caractères différents.

Exercice 4 : Écrire une fonction prenant en entrée une liste de nombres et qui détermine le plus petit écart en valeur absolue entre deux nombres consécutifs de la liste.

Par exemple, la fonction retourne `5` pour la liste `[1, 20, 42, 73, 68, 57, 1]`.

Exercice 5 : Autour de l'algorithme de Dijkstra...

Nous allons considérer ici un problème de recherche de chemin optimal en poids dans un graphe, avec la contrainte supplémentaire qu'en cas d'égalité de poids le chemin utilisant le moins d'arcs sera privilégié. En cas de nouvelle égalité, n'importe quel chemin ainsi optimal pourra être retenu.

Les graphes seront représentés par liste d'adjacence et les sommets seront les entiers naturels de `0` au nombre de sommets moins un. Ainsi, pour un tel graphe `g`, si la liste `g[2]` contient un couple `(4, 6)`, c'est qu'il existe un arc depuis le sommet appelé `2` vers le sommet appelé `4`, et que cet arc est de poids `6`.

Il est important de noter que le couple en question peut être à n'importe quel indice de la liste d'adjacence `g[2]`, et on ne devra pas supposer qu'elle est croissante, ni selon les sommets ni selon les poids.

Une première possibilité pour répondre au problème posé en introduction est d'utiliser une comparaison naturelle sur les couples, les distances au sommet de références étant alors stockées en tant que couples `(d, l)` où `d` est le poids minimal actuellement recensé vers le sommet concerné et `l` la longueur minimale d'un chemin ayant ce poids. Mais pour la beauté des raisonnements informatiques et des encodages de plusieurs informations en une valeur, une autre possibilité sera explorée ici.

Nous allons donc transformer le graphe pour que les poids soient remplacés par une valeur hybride égale à dix puissance `k` fois le poids d'origine plus un, de sorte que dix puissance `k` soit strictement supérieur au nombre de sommets. Ainsi, le nouveau poids d'un chemin de taille limitée au nombre de sommets sera dix puissance `k` fois le poids d'origine du chemin plus le nombre d'arcs empruntés, permettant de distinguer par ce critère les chemins de même poids dans le graphe de base, tout en évitant des valeurs exceptionnelles dues à une retenue car s'il existe un chemin optimal en poids, il en existe aussi un qui ne passe au plus qu'une fois par un même sommet.

Question 5.1 : Écrire une fonction prenant en entrée un entier positif `n` et qui détermine la plus petite puissance de dix strictement supérieure à `n`.

Question 5.2 : Écrire une fonction prenant en entrée un graphe pondéré `g` donné par liste d'adjacence et qui renvoie le graphe `g2` obtenu en modifiant tous les poids selon le principe évoqué précédemment.

Question 5.3 : Écrire en Python l'algorithme de Dijkstra pour un graphe quelconque (qui peut être aussi bien `g2` que `g` en pratique). **Il n'y a aucun objectif de complexité mais il faudra calculer celle-ci.** En particulier, il est recommandé de chercher par exploration intelligente d'une liste le sommet non encore traité à distance minimale de l'origine (en argument, et pas nécessairement le sommet `0`), et toute tentative d'utilisation d'un module *ad hoc* nécessitera d'une part qu'il n'y ait pas la moindre erreur d'utilisation du module et d'autre part que la complexité des fonctions appelées soit donnée et exacte (cela devrait être suffisamment décourageant). De même, toute utilisation de file de priorité nécessitera d'écrire soi-même les fonctions associées (c'est suffisamment décourageant).

Question 5.4 : Écrire alors une fonction prenant en entrée un graphe `g` et deux sommets `s` et `t` et qui détermine le poids minimal d'un chemin de `s` à `t` ainsi que la longueur minimale d'un chemin ayant ce poids.

Question 5.5 : Expliquer les adaptations qu'il faudra faire au niveau des programmes précédents si on avait voulu le chemin le plus court, et à égalité de longueur celui de poids moindre.