Programmation en temps limité

2nd semestre 2023/2024

1 Évaluation

Un sujet de programmation de 4h vous sera proposé en fin de semestre. L'épreuve aura lieu le 6 mai à 13h30. Des exercices d'entraînement vous seront proposés tout au long du semestre.

L'épreuve sera simultanée pour tous et sans connexion internet. Le sujet sera constitué d'une série de questions qui pourront largement se traiter de manière indépendante. Il vous sera demandé d'implémenter des algorithmes et manipulations de structures de données simples, pour produire les résultats demandés par le sujet. Vos programmes devront donc être exécutables et fonctionnels ! Vous rendrez également votre code en fin d'épreuve en complément de vos réponses.

Le langage imposé est python. Vous constaterez au travers des exercices d'entraînement proposés que tout est réalisable avec des notions de base.

Durant le semestre, vous pourrez poser vos questions indifféremment à Mathieu Chapelle (mathieu.chapelle@univ-orleans.fr) ou Martin Delacourt (martin.delacourt@univ-orleans.fr).

2 Pour s'entraîner

2.1 À propos de nombres :

- pour un entier n et un entier k, retourner la chaîne de caractères représentant n en base k
- pour un entier k et une chaîne de caractères s, tester si la chaîne s ne contient que des chiffres entre 0 et $k-1$
- pour un entier k et une chaîne de caractères s, retourner (si la chaîne est valide) la valeur de l'entier représenté par s en base k
- pour un entier n, tester si n est premier
- pour deux entiers m et n, tester si n est une puissance de m

2.2 À propos de listes :

- pour une liste donnée en paramètre, retourner la somme des nombres contenus dans la liste
- pour une liste l et un entier k donnés en paramètre, retourner une liste contenant tous les éléments de l multipliés par k
- pour une liste de chaînes de caractères, retourner une liste contenant en position i la longueur de la chaîne en position i
- pour une liste l, retourner une liste contenant chaque élément de l une seule fois (en enlevant les répétitions)
- pour une liste l, retourner l'élément le plus présent dans la liste et son nombre d'occurrences (en cas d'égalité, retourner le premier à apparaître dans l)
- pour une liste l, dire si la liste contient un doublon
- pour deux paramètres k et n, générer une liste de taille k contenant des entiers tirés aléatoirement avec `random.randint` entre 1 et n, puis retourner 1 si cette liste contient au moins un doublon, 0 sinon

- pour $k = 23$ et $n = 365$, faire 1000 fois l'expérience précédente et calculer la moyenne des résultats (0 ou 1). Que constate-t-on en faisant varier k ?

2.3 À propos de dictionnaires :

- pour un dictionnaire d, retourner une liste contenant les couples `(clé,valeur)` présents dans d
- pour un dictionnaire d contenant des couples `(joueur, score)` et un seuil k, retourner la liste des joueurs dont le score est inférieur à k
- pour un dictionnaire d contenant des couples `(joueur, liste de scores)`, modifier d pour qu'il contienne la somme des scores au lieu de la liste, la fonction ne retourne rien
- pour une liste d'étudiants, retourner un dictionnaire d contenant les couples `(étudiant, note)` où les étudiants sont tous ceux de la liste et les notes sont générées aléatoirement avec `random.randint`

2.4 À propos de graphes :

On suppose qu'un graphe est donné par une liste de sommets et une liste d'arêtes représentées par des couples de sommets. Si le graphe est pondéré, les arêtes sont des triplets contenant aussi le poids.

- pour un graphe G, retourner `True` si au moins un sommet n'est relié à aucun autre, `False` sinon
- pour un graphe G, retourner le degré maximal (nombre de voisins d'un sommet)
- pour un graphe G et un sommet x, retourner la composante connexe de x (la liste des sommets y tels qu'un chemin entre x et y existe)
- pour un graphe G, tester la connectivité de G (il existe un chemin de n'importe quel sommet vers n'importe quel autre)
- pour un graphe pondéré G et un sommet x, retourner la liste des couples `(sommet,distance)` pour les sommets dans la composante connexe de x (algorithme de Dijkstra)

2.5 À propos d'automates finis :

- proposer une structure de données représentant un automate fini
- pour un automate fini A et un mot u sur son alphabet, tester si u est dans le langage de A
- pour un automate fini A, tester si le langage de A est vide
- implémenter l'algorithme de minimisation d'un automate fini

3 Pour consolider

3.1 À propos de mots :

- retourner `True` si le mot u donné en entrée contient plus de a que de b
- générer la liste de tous les mots périodiques de longueur 30 et période 3 sur l'alphabet $\{a, b\}$
- pour un mot v, retourner le plus grand mot u tel que u est un préfixe de v et le miroir de u (noté $\tilde{u}$) est un suffixe de v

- pour deux mots u et v de même longueur, retourner le mot contenant les caractères de u dans l'ordre aux positions impaires et ceux de v aux positions paires
- pour un mot u, retourner la liste ordonnée lexicographiquement des caractères sans doublon
- pour un mot u sur l'alphabet $\{a, b\}$, retourner le motif de longueur 3 le plus présent dans u. Par exemple, *aba* est présent 3 fois dans *abababbaaba*.

3.2 En dimension 2 :

On considère ici une matrice donnée par la liste des lignes, chaque ligne étant donnée par une liste. On suppose les lignes de même taille.

- retourner `True` si la matrice donnée en entrée est carrée (même nombre de lignes et de colonnes)
- pour une matrice M et un entier k, retourner le nombre de colonnes de M contenant au moins k fois le caractère `'a'`
- pour une matrice de mots, vérifier que pour chaque $i > 0$ et $j > 0$, le mot en position (i, j) a pour préfixe le mot en position $(i - 1, j - 1)$
- pour une matrice d'entiers, retourner i tel que la somme des éléments de la ligne i soit la plus grande parmi toutes les lignes
- pour n en entrée, retourner la matrice de taille n sur n contenant en position (i, j) le coefficient binomial $\begin{pmatrix} i+j \\ i \end{pmatrix}$. On pourra s'inspirer de `https://fr.wikipedia.org/wiki/Triangle_de_Pascal`.
- pour une matrice M de mots, retourner la matrice de même taille contenant en position (i, j) la taille du mot en position (i, j) de M

3.3 Et des suites :

- calculer le n-ième terme de la suite de Fibonacci donnée par $u_0 = 0$, $u_1 = 1$ et $u_{k+2} = u_{k+1} + u_k$
- on note f la fonction qui à un entier pair k associe $k/2$ et à un entier impair k associe $3 * k + 1$. On appelle suite de Syracuse de k la suite $k, f(k), f(f(k)), f(f(f(k))), \ldots$. On arrête la suite lorsqu'un des termes vaut 1.
  1. Calculer la suite de Syracuse de 13.
  2. Pour un entier n, calculer le plus petit entier k tel que la suite de Syracuse de k est de longueur au moins n.
  3. Montrer que pour tout $k \leq 50$, la suite de Syracuse de k est finie (on tombe sur 1 après un nombre fini d'étapes). Remarque : Savoir si toute suite de Syracuse est finie est une célèbre question ouverte.
- pour un mot u sur l'alphabet $\{a, b, c, \ldots, z\}$, remplacer en partant de la première lettre toute occurrence de deux lettres consécutives de l'alphabet par deux fois la lettre suivante. Exemple : `'efrhabdj'` devient `'ggrhccdj'`.
- pour un mot u appliquer la transformation précédente sur u puis sur l'image de u, etc..., jusqu'à obtenir un mot égal à son image. Exemple : `'efrhabdj'` devient `'ggrhccdj'` qui devient `'ggrhceej'`.