DS 1

CPES « Sciences des données, arts et cultures » : Introduction à l'algorithmique

Antoine GAUQUIER antoine.gauquier@ens.fr
Pierre SENELLART pierre@senellart.com

14 novembre 2023

L'examen dure deux heures, comporte 2 exercices et 10 questions. Les documents sont autorisés, mais limités à 10 feuilles recto-verso par personne. Aucun appareil numérique n'est autorisé, donc, de façon non-exhaustive : pas d'ordinateur, pas de téléphone, pas de montre connectée, etc. Pas de calculatrice non plus. Pour les questions contenant du code Python, les erreurs mineures de syntaxe Python ne seront pas pénalisées, tant que le code écrit ressemble à un programme Python. Veiller à indiquer votre nom sur l'ensemble des copies rendues, et à les numéroter le cas échéant.

Dans tout le sujet, quand on parlera de complexité, il s'agira de complexité algorithmique en temps, donnée avec un $\Theta()$ si possible ou un $O()$ si on a seulement une borne supérieure.

A. Mot le plus fréquent

Question 1. Rappeler la complexité d'accès à un élément arbitraire d'un dictionnaire Python.

Question 2. Écrire une fonction Python `clef_plus_grande_valeur` prenant en entrée un dictionnaire Python `d` et retournant une clef de `d` associée à une valeur maximale, c'est-à-dire une clef `x` telle que `d[x]` est supérieure ou égale à tous les `d[y]` pour chaque `y` clef de `d`. On n'utilisera aucune fonction avancée des dictionnaires Python, en particulier aucune fonction de tri.

Question 3. Quelle est la complexité de cette fonction `clef_plus_grande_valeur` ? Justifier.

Question 4. On s'intéresse au problème de déterminer le mot le plus fréquent dans un texte (par exemple, le contenu d'un roman). On suppose que le texte est fourni sous la forme d'une liste Python de la suite de tous ses mots, en minuscule. En utilisant la fonction `clef_plus_grande_valeur` écrire une fonction Python `mot_plus_frequent` qui prend en argument la liste Python des mots du texte et retourne en sortie le mot le plus fréquent de la liste.

Question 5. Quelle est la complexité de cette fonction `mot_plus_frequent` ? Justifier.

B. Chaînes bien parenthésées

On considère les chaînes de caractères formées uniquement de parenthèses et crochets ouvrants ou fermants, c'est-à-dire des quatre caractères « `(` », « `)` », « `[` », « `]` ».

On dit qu'une telle chaîne de caractères est *bien parenthésée* s'il existe une fonction f associant certaines positions de la chaîne à d'autres positions de la chaîne telle que :

— on peut associer à chaque parenthèse ouvrante « `(` » en position i une parenthèse fermante « `)` » dans une position $f(i) > i$;

— on peut associer à chaque crochet ouvrant « `[` » en position i un crochet fermant « `]` » en position $f(i) > i$;

— si i_1 e i_2 sont les positions de deux caractères ouvrants avec $i_1 < i_2 < f(i_1)$, alors $f(i_2) < f(i_1)$.

Cela correspond à la définition intuitive de bien parenthésée : si on ouvre une seconde parenthèse/crochet avant d'avoir refermé une première parenthèse/crochet, il faudra la refermer avant pour que les parenthèses/crochets soient bien imbriqués.

Par exemple, « `([]())` » est bien parenthésée, comme en témoigne la fonction f définie par : $f(0) = 5$, $f(1) = 2$, $f(3) = 4$.

Question 6. Montrer que la chaîne « `(([])[()])[]` est bien parenthésée en proposant une fonction mathématique f satisfaisant les conditions. On indexera les positions dans la chaîne à partir de 0.

Question 7. On propose l'algorithme suivant, qui utilise une structure de *pile*, initialement vide :

— On parcourt la chaîne de caractères de gauche à droite en lisant les caractères l'un après l'autre.

— Quand on lit un caractère ouvrant, on l'ajoute en haut de la pile.

— Quand on lit un caractère fermant, on vérifie que la pile n'est pas vide et que le caractère en haut de la pile est le caractère ouvrant correspondant ; si oui, on enlève le caractère en haut de la pile ; sinon, on renvoie Faux.

— Si on a fini de parcourir la chaîne, on renvoie Vrai si la pile est vide, Faux sinon.

On admettra que cet algorithme renvoie Vrai si la chaîne est bien parenthésée, Faux sinon. Écrire une fonction Python `est_bien_parenthesee` qui prend en argument une chaîne de parenthèses et crochets et implémente cet algorithme (on choisira une structure de données Python appropriée pour la pile).

Question 8. Quelle est la complexité de votre implémentation ? Justifier.

Question 9. On souhaite créer une classe Python `Chaine` qui comporte les méthodes suivantes :

— un constructeur qui prend en argument, en plus de `self`, une chaîne de caractères Python et la stocke dans une propriété `__c` ;

— une méthode `taille` sans arguments autre que `self` qui renvoie le nombre de caractères de la propriété `__c` ;

— une méthode `est_bien_parenthesee` sans argument autre que `self` qui renvoie si la chaîne est bien parenthésée.

Donner le code Python d'une telle classe ; dans le cas de la méthode `est_bien_parenthesee`, on ne recopiera pas le code de la fonction `est_bien_parenthesee` de la question 7 mais on expliquera comment transformer cette fonction en une méthode.

Question 10. En supposant qu'une chaîne peut contenir uniquement des parenthèses et pas de crochets, proposer un algorithme (décrit, au choix, en français avec suffisamment de détails, ou en pseudo-code, ou en Python) plus simple que le précédent (en particulier n'utilisant pas de structure de pile), qui teste si une telle chaîne est bien parenthésée. Quelle est sa complexité ?