Informatique tronc commun

Programmes usuels sur les listes et les dictionnaires

Exercices à compléter

Ce document est à traiter en travail personnel.

Il est essentiel de savoir écrire de courts programmes sur les listes et dictionnaires.

Voici quelques petits exercices (à travailler à votre rythme).

Les **tris**, qui sont aussi des programmes usuels, ne sont pas repris ici : voir le cours de première année ou celui de **M. Vandaele** : https://pvdmaths.fr/contenuWeb/itc.php, mot de passe : _____.

Les opérations autorisées sur les listes et les dictionnaires, avec leurs complexités, sont rappelées ci-dessous (n et m désignent les longueurs des listes `L` et `M`, ou le nombre de clés du dictionnaire `d`).

Listes.

Opération	Coût	Remarque
création vide `[]`	$O(1)$	
création `[x] * n`	$O(n)$	les n cases référencent le même `x`
compréhension `[f(i) for i in range(n)]`	$O(n)$	si chaque `f(i)` coûte $O(1)$
longueur `len(L)`	$O(1)$	la taille est stockée
accès `L[i]`, modification `L[i] = x`	$O(1)$	calcul d'adresse
ajout en fin `L.append(x)`	$O(1)$ amorti	tableau dynamique
retrait en fin `L.pop()`	$O(1)$	
tranche `L[i:j]`	$O(j-i)$	copie des éléments
concaténation `L + M`	$O(n+m)$	nouvelle liste
copie `L.copy()`	$O(n)$	copie superficielle
parcours `for x in L`	$O(n)$	hors coût du corps de boucle

Attention. Le test d'appartenance `x in L` sur une **liste** est **interdit** dans ces exercices : il coûte $O(n)$ (c'est un parcours caché), les énoncés l'interdisent souvent, et il ne figure pas dans l'annexe du programme. On écrit la boucle.

Dictionnaires. Les clés doivent être hachables (non modifiables) : entiers, chaînes, tuples d'éléments hachables… mais pas listes.

Opération	Coût	Remarque
création vide `{}`	$O(1)$	
création `{c1: v1, c2: v2}`	$O(n)$	n couples clé-valeur
accès `d[k]`	$O(1)$ en moyenne	erreur si `k` n'est pas une clé
ajout ou modification `d[k] = v`	$O(1)$ en moyenne	amorti (redimensionnement)
appartenance `k in d`	$O(1)$ en moyenne	c'est tout l'intérêt du dictionnaire
nombre de clés `len(d)`	$O(1)$	
copie `d.copy()`	$O(n)$	copie superficielle
parcours des clés `for k in d`, ou `for k in d.keys()`	$O(n)$	hors coût du corps de boucle
parcours des couples `for k, v in d.items()`	$O(n)$	hors coût du corps de boucle

> Les coûts « en moyenne » sont ceux du hachage (voir le cours) ; ils supposent que le calcul du haché d'une clé coûte $O(1)$, ce qui est faux pour une longue chaîne ou un long tuple (coût proportionnel à la longueur). Les chaînes de caractères et les tuples, non modifiables, se comportent comme des listes pour `len`, l'accès `s[i]`, les tranches, la concaténation et le parcours.

Le type `set`, les méthodes `sort`, `index`, `count`, `insert`, `pop(i)`, l'instruction `del` et les fonctions `sorted`, `min`, `max`, `sum` ne sont pas au programme : on ne les utilise pas ci-dessous, on écrit les boucles. Pour représenter un ensemble, on utilise un dictionnaire dont toutes les valeurs valent `True`.

1 Compter des occurrences

Question. Écrire une fonction `occurrences(L)` prenant en argument une liste `L` d'éléments hachables et renvoyant le dictionnaire associant à chaque élément de `L` son nombre d'occurrences, en un seul parcours de `L`. Quelle est sa complexité ?

> *« il faut penser à traiter à part le cas où le dictionnaire ne comporte pas encore d'entrée pour la valeur à ajouter »* (X-ENS 2018)

2 Maximum et position du maximum

Question. Écrire une fonction `max_argmax(L)` prenant en argument une liste non vide `L` de nombres et renvoyant le couple `(m, i)` formé du maximum `m` de `L` et du **premier** indice `i` où il est atteint. Quelle est sa complexité ?

Question. Écrire une fonction `deux_max(L)` prenant en argument une liste `L` de nombres, de longueur au moins 2, et renvoyant le couple `(m1, m2)` de ses deux plus grandes valeurs, avec `m1 >= m2`, comptées avec multiplicité (`[5, 5, 3]` donne `(5, 5)`), en un seul parcours de `L`. Quelle est sa complexité ?

Question. Écrire une fonction `point_le_plus_bas(P)` prenant en argument une liste non vide `P` de couples `(x, y)` de nombres (les points d'un nuage) et renvoyant le point de plus petite ordonnée et, s'il y en a plusieurs, celui de plus petite abscisse. Quelle est sa complexité ?

« Parmi les erreurs les plus courantes, nous pouvons noter : la non-prise en compte de la directive de plus faible abscisse, dans le cas où plusieurs points ont l'ordonnée minimale ; un test simultané sur abscisse et ordonnée : un tel test simultané ne fonctionne en général pas ; calcul de l'ordonnée maximale et non minimale » (X-ENS 2015)

Question. Écrire une fonction `plus_frequent(L)` prenant en argument une liste non vide `L` d'éléments hachables et renvoyant un couple `(x, k)` formé d'un élément `x` le plus fréquent de `L` et de son nombre d'occurrences `k`. On pourra utiliser `occurrences`. Quelle est sa complexité ?

3 Valeurs distinctes, doublons

Question. Écrire une fonction `distincts(L)` prenant en argument une liste `L` d'éléments hachables et renvoyant la liste des valeurs distinctes de `L`, dans l'ordre de première apparition. Quelle est sa complexité ? La comparer avec celle d'une version qui, pour chaque élément, parcourt la liste résultat pour savoir s'il y figure déjà.

> *« la meilleure approche n'était pas d'enlever des éléments de la table à l'aide de del ou de pop(i) […] (del et pop(i) de Python sont en O(len(L))), mais de reconstruire une table exempte de doublons »* (X-ENS 2018)

Question. Écrire une fonction `a_un_doublon(L)` prenant en argument une liste `L` d'éléments hachables et renvoyant `True` si une valeur apparaît au moins deux fois dans `L`, `False` sinon. Quelle est sa complexité ?

Question. Écrire une fonction `paire_de_somme(L, s)` prenant en argument une liste `L` d'entiers et un entier `s`, et renvoyant `True` s'il existe deux indices $i < j$ tels que `L[i] + L[j] == s`, `False` sinon. Quelle est sa complexité ? La comparer avec celle d'une double boucle.

Question. Une marche sur la grille $\mathbb{Z}^2$ est donnée par la liste `chemin` des positions visitées successivement, chaque position étant une liste `[x, y]` d'entiers (comme dans CCMP 2021). Écrire une fonction `est_auto_evitante(chemin)` prenant en argument une telle liste et renvoyant `True` si la marche ne repasse jamais par une même position, `False` sinon. Justifier que sa complexité est $O(n)$ en moyenne, où n est la longueur de `chemin`.

4 Adressage direct : tableau de booléens

Quand les valeurs sont des entiers d'un petit intervalle $[\![0, m-1]\!]$, une liste de m booléens indexée par ces entiers remplace avantageusement le dictionnaire de `True` : coût $O(1)$ au pire par opération (et non plus en moyenne), mais $O(m)$ pour la création.

Question. Écrire une fonction `plus_petit_absent(L)` prenant en argument une liste `L` de n entiers naturels (quelconques, éventuellement très grands) et renvoyant le plus petit entier naturel qui n'apparaît pas dans `L`. On utilisera une liste de booléens de longueur $n+1$. Quelle est sa complexité ?

5 Suites récurrentes : détecter un cycle

Soit f une fonction d'un ensemble fini E dans lui-même et $u_0 \in E$. La suite définie par $u_{n+1} = f(u_n)$ prend un nombre fini de valeurs, donc finit par boucler : il existe un plus petit indice μ tel que u_μ réapparaisse plus loin, et un plus petit $\lambda \geq 1$ tel que $u_{\mu+\lambda} = u_\mu$; la suite est alors périodique de période λ à partir du rang μ. Par exemple, pour un générateur pseudo-aléatoire $u_{n+1} = (au_n + c) \,\%\, m$, une période λ courte est rédhibitoire.

Question. Écrire une fonction `cycle(f, u0)` prenant en argument une fonction `f` (de E dans E, les éléments de E étant hachables) et un élément `u0` de E, et renvoyant le couple `(mu, lam)` défini ci-dessus. Quelle est sa complexité, en supposant que chaque appel à `f` coûte $O(1)$?

6 Intersection, listes disjointes

Question. Écrire une fonction `intersection(L1, L2)` prenant en argument deux listes `L1` et `L2` (de longueurs n_1 et n_2) d'éléments hachables et renvoyant la liste des éléments distincts de `L1` qui sont aussi dans `L2`. Justifier que sa complexité est $O(n_1 + n_2)$ en moyenne.

Le même schéma (on range l'une des listes dans un dictionnaire, puis on parcourt l'autre) donne la différence de deux listes, ou le test « les deux listes n'ont aucun élément commun » (X-ENS 2023).

> *« Certains candidats construisent séparément le dictionnaire des caractères contenus dans texte1 et celui des caractères contenus dans texte2. C'est inutile. »* (X-ENS 2023)

7 Index et regroupement

Question. Écrire une fonction `positions(L)` prenant en argument une liste `L` d'éléments hachables et renvoyant le dictionnaire associant à chaque valeur de `L` la liste croissante des indices où elle apparaît. Quelle est sa complexité ?

« il était plus judicieux d'itérer sur les indices que sur les éléments » (X-ENS 2018)

Question. Écrire une fonction `inverse(d)` prenant en argument un dictionnaire `d` dont les valeurs sont hachables et renvoyant le dictionnaire associant à chaque valeur `v` de `d` la liste des clés `k` telles que `d[k] == v`. Quelle est sa complexité ?

Question. Deux mots sont **anagrammes** s'ils sont formés des mêmes lettres avec les mêmes multiplicités (« chien », « niche », « chine »). Écrire une fonction `anagrammes(mots)` prenant en argument une liste `mots` de chaînes de lettres minuscules non accentuées et renvoyant la liste des groupes de mots anagrammes les uns des autres (chaque groupe étant une liste de mots ; un mot sans autre anagramme forme un groupe à lui seul, et un mot répété figure autant de fois dans son groupe). On associera à chaque mot une clé : la chaîne de ses lettres rangées dans l'ordre alphabétique, construite sans trier, en comptant ses lettres dans une liste de 26 entiers. Quelle est sa complexité ?

Un tel index permet aussi de faire une **jointure** (au sens des bases de données) sans comparer toutes les paires de lignes : c'est l'objet de la partie III du sujet X-ENS 2018.

Question. Deux tables sont données par des listes de tuples `T1` (de longueur n_1) et `T2` (de longueur n_2). Écrire une fonction `jointure(T1, T2, a, b)` prenant en argument ces deux listes et deux indices de colonnes `a` et `b`, et renvoyant la liste des couples `(l1, l2)` de lignes de `T1` et de `T2` telles que `l1[a] == l2[b]`. Quelle est sa complexité ? La comparer avec celle d'une double boucle.

> *« Il était intéressant de noter que la longueur des listes renvoyées par le dictionnaire est en général bien plus petite que la longueur de la table, apportant donc un gain appréciable. »* (X-ENS 2018)

8 Graphes : de la liste d'arêtes au dictionnaire d'adjacence

Un graphe non orienté, simple et sans boucle, est donné par la liste `sommets` de ses n sommets (hachables) et la liste `aretes` de ses a arêtes (couples de sommets).

Question. Écrire une fonction `adjacence(sommets, aretes)` prenant en argument ces deux listes et renvoyant le dictionnaire associant à chaque sommet la liste de ses voisins (le graphe étant simple et sans boucle, chaque arête `(s, t)` vérifie `s != t` et n'apparaît qu'une fois, dans un seul sens). Quelle est sa complexité en fonction de n et a ?

Question. Écrire une fonction `degres(g)` prenant en argument le dictionnaire d'adjacence `g` d'un graphe simple, sans boucle, et renvoyant le dictionnaire associant à chaque sommet son degré. Quelle est sa complexité ?

Question. Écrire une fonction `voisins(s, sommets, aretes)` prenant en argument un sommet `s` et les listes `sommets` et `aretes` d'un graphe simple, sans boucle, et renvoyant la liste des voisins de `s`, sans construire le dictionnaire d'adjacence. Un élève parcourt la liste `sommets` et, pour chaque sommet `t`, parcourt la liste `aretes` pour savoir si `(s, t)` ou `(t, s)` en fait partie : quelle est la complexité de sa méthode ? Et celle de la vôtre ?

> *« la première consiste à faire une boucle sur les n individus, et appeler la fonction précédente sontAmis() : l'inconvénient de cette approche est qu'elle est de complexité $O(n \times m)$ […] la seconde consiste à faire une boucle sur les m liens […] : cette approche est effectivement de complexité $O(m)$, comme demandé. »* (X-ENS 2016)

9 Agréger par catégorie

Question. Écrire une fonction `moyenne_par_categorie(couples)` prenant en argument une liste `couples` de couples `(c, v)`, formés d'une catégorie `c` (hachable) et d'une valeur numérique `v`, et renvoyant le dictionnaire associant à chaque catégorie la moyenne des valeurs correspondantes (c'est l'équivalent d'un `GROUP BY` avec `AVG` en SQL). Quelle est sa complexité ?

10 Polynômes creux

Un polynôme creux (avec peu de coefficients non nuls, comme $X^{1000} + 3X^2 - 1$) se représente par le dictionnaire degré → coefficient non nul : `{1000: 1, 2: 3, 0: -1}`. On note $|p|$ le nombre de clés de `p`.

Question. Écrire une fonction `somme_poly(p, q)` prenant en argument deux polynômes creux `p` et `q` et renvoyant le polynôme creux $p + q$, sans modifier `p` ni `q`, et sans garder de coefficient nul. Quelle est sa complexité ?

Question. Écrire une fonction `produit_poly(p, q)` prenant en argument deux polynômes creux `p` et `q` et renvoyant le polynôme creux pq, sans coefficient nul. Quelle est sa complexité ?

11 Décoder avec une table de correspondance

Question. Un code préfixe est donné par un dictionnaire `code` qui associe à chaque caractère une chaîne de `'0'` et de `'1'` (par exemple `{'a': '0', 'b': '10', 'c': '11'}`), aucun mot de code n'étant le début d'un autre. Écrire une fonction `decoder(bits, code)` prenant en argument une chaîne de bits `bits`, supposée valide (c'est une suite de mots de code), et un tel dictionnaire `code`, et renvoyant la chaîne décodée. Quelle est sa complexité ?