Informatique du Tronc Commun

TD n°2 Les dictionnaires – *Corrigé*

Parcours possibles

♪ Si vous avez des difficultés en python, notamment sur les dictionnaires : exercices n°1, n°2, n°3, n°4.

♪ ♪ Si vous vous sentez moyennement à l'aise, mais pas en difficulté : exercices n°1, n°2 (Q2), n°4, n°6

♪ ♪ ♪ Si vous êtes à l'aise : exercices n°4, n°6, n°5, n°7.

I Exercices fondamentaux sur les dictionnaires

Exercice n°1 Min-Max dans un dictionnaire ♪♥

On pourra utiliser la fonction `float('inf')` qui permet de coder le plus grand flottant possible (il n'a pas de valeur, mais il est supérieur à tous les flottants codables par python).

R1. Écrire une fonction `max_dico(D:dict)->float` qui prend en entrée le dictionnaire `D` et qui renvoie la valeur maximale.

Indice : les valeurs dans le dictionnaire peuvent être positives ou négatives, et il n'y a pas d'ordre dans un dictionnaire, on ne peut donc pas connaître la « première » valeur (comme on ferait avec les listes). Il faut donc initialiser le maximum à une valeur qu'on sait nécessairement ne pas être plus grande qu'une des valeurs du dictionnaire.

Solution:

```
def max_dico(D):
    M=-float('inf')
    for c in D:
        if D[c]>M:
            M=D[c]
    return M
```

R2. Écrire une fonction `cle_min_dico(D:dict)->int` qui prend en entrée le dictionnaire `D` et qui renvoie la clé de valeur minimale.

Solution:

```
def cle_min_dico(D):
    m=float('inf')
    for c in D:
        if D[c]<m:
            m=D[c]
            cmin=c
    return cmin
```

R3. Écrire une fonction `min_max(L:list)->dict` qui prend en argument une liste de nombres non vide et renvoie un dictionnaire dont les clés sont les chaînes `"min"` et `"max"` avec pour valeurs respectives le minimum et le maximum des nombres de la liste.
Par exemple : `min_max([8,5,9,3,1,7])` renverra `{"min":1,"max":9}`.

Solution:

```
def min_max(L):
    m,M=L[0],L[0]
    for i in range(1,len(L)):
        if m<L[i]:
            m=L[i]
        if M>L[i]:
            M=L[i]
    return {"min":m,"max":M}
```

Exercice n°2 Occurrences ♪❤

R1. Écrire une fonction `occurrences(L:list)->dict` qui prend en argument une liste de nombres et renvoie un dictionnaire dont les clés sont les différents nombres de la liste avec pour valeur le nombre d'occurrences de chaque nombre.

Par exemple `occurrences([3,5,-2,3,3,-2])` renvoie `{-2:2, 3:3, 5:1}`.

Solution:

```
def occurrences(L):
    d={}
    for x in L:
        if x not in d: # x n'a pas été encore rencontré
            d[x]=1
        else: # x apparaît une fois de plus
            d[x]=d[x]+1
    return d
```

R2. Écrire une fonction `occurrences(ch:str)->dict` qui prend en argument une chaîne de caractères et renvoie un dictionnaire dont les clés sont les différents caractères de la chaîne avec pour valeur le nombre d'occurrences de chaque nombre.

Solution:

```
def occurrences(ch):
    d={}
    for x in ch:
        if x not in d: # x n'a pas été encore rencontré
            d[x]=1
        else: # x apparaît une fois de plus
            d[x]=d[x]+1
    return d
```

Exercice n°3 Température ♪

On dispose de noms de villes avec les températures moyennes relevées à une date donnée. Les données sont enregistrées dans une liste de listes comme `[["Paris",12],["Lyon",14],["Marseille",21],...]` On note n la longueur de la liste.

R1. On souhaite accéder à une donnée, la modifier, ou en ajouter une. Quelle est en fonction de n la complexité en temps de chacune de ces opérations ?

Solution: Ces opérations sont de complexité linéaire en la longueur de la liste.

R2. Afin de trouver la température d'une ville, on trie cette liste, à l'aide d'un algorithme de tri rapide (rappel : il est de complexité $O(n\log_2(n))$), suivant les noms des villes en utilisant l'ordre lexicographique. Puis on effectue une recherche dichotomique du nom de la ville. Quelle est la complexité de cette recherche de température ?

Solution: La recherche dichotomique est de complexité logarithmique.

On souhaite stocker ces données dans un dictionnaire dont les clés sont les noms des villes et les valeurs les températures.

R3. Écrire une fonction `convert(L:list[list])->dict` prenant en paramètre une liste comme ci-dessus et renvoyant le dictionnaire correspondant.

Solution:

```
def convert(L):
    d={}
    for li in L:
        d[li[0]]=li[1] # le nom de la ville (élément 0 de sous liste li) est la clé, et la température la valeur
    return d
```

On suppose posséder un dictionnaire `d` dont les clés sont les villes et les valeurs associées de la température moyenne relevée à une date donnée.

R4. Écrire une fonction `temperature(d:dict,ville:str)->float` qui prend en argument un tel dictionnaire et un nom de ville et renvoie la température correspondant à la ville.

Solution:

```
def temperature(d,ville):
    return d[ville]
```

R5. Quelle est la complexité en temps d'une recherche ou d'une modification de température ?

Solution: C'est une recherche ou modification en temps constant, indépendant de la longueur du dictionnaire.

R6. Écrire une fonction `moyenne(d:dict)->float` qui prend en argument un tel dictionnaire et qui renvoie la température moyenne en France ce jour-là.

Solution:

```
def temperature(d):
    S=0 # somme
    for ville in d:
        S=S+d[ville] # on somme les températures
    return S/len(d) # moyenne
```

Exercice n°4 Inversion ♪ ♪ ♥

Un graphe orienté est représenté par un dictionnaire. Les clés sont les sommets du graphe. La valeur associée à une clé `s` est la liste des sommets extrémités des arêtes partant de `s`. On souhaite créer le graphe obtenu à partir d'un graphe initial en inversant le sens des arêtes.

Écrire une fonction `inverse(G:dict)->dict` qui prend un graphe (représenté par le dictionnaire d'adjacence) en paramètre et renvoie le graphe obtenu par l'inversion du sens des arêtes.

Par exemple :

```
>>> G = {'a':['b','c','e'] , 'b':['d'] , 'c':['e'] , 'd':['c','e'] , 'e':[]}
>>> inverse(G)
{'a':[] , 'b':['a'] , 'c':['a','d'] , 'd':['b'] , 'e':['a','c','d'] }
```

Solution:

```
def inverse(G):
    Ginv={}
    for c in G:
        for x in G[c]: # parcours des successeurs de c
            if x not in Ginv: # si x n'est pas encore dans Ginv
                Ginv[x]=[c] # on l'ajoute avec comme valeur associée la liste qui contient c qui est un prédécesseur de x
            else:
                Ginv[x].append(c) # si x est déjà dans Ginv, il suffit d'ajouter c dans la liste des prédécesseurs de x
    return Ginv
```

II Exercices d'approfondissements

Exercice n°5 Matrice ♪ ♪ ♪

On s'intéresse ici aux matrices parcimonieuses, c'est-à-dire dont la plupart des coefficients sont nuls.

Une telle matrice M de dimensions (n, p) pourra être codée par un dictionnaire ayant pour couples clefs/valeurs :

- `'dim' : (n,p)` , qui donne les dimensions de la matrice ;
- `(i,j) : M_{ij}` , qui donne pour chaque couple (i, j), la valeur de l'élément $M_{i,j} \neq 0$ de la matrice.

R1. Donner le dictionnaire qui code la matrice : $\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 \end{pmatrix}$

Solution:

```
M={'dim': (2, 4), (1, 2): 4}
```

R2. Proposer une fonction d'addition `somme_mat(M1:dict,M2:dict)->dict` de deux matrices parcimonieuses.

Solution:

```
def somme_mat(M1,M2):
    assert M1['dim']==M2['dim'] , "les deux matrices doivent être de même dimension"
    M={'dim':M1['dim']}
    for c1 in M1:
        if c1!="dim":
```

```
            if c1 in M2:
                M[c1]=M1[c1]+M2[c1]
            else:
                M[c1]=M1[c1]
    for c2 in M2:
        if c2!="dim" :
            if c2 not in M:
                M[c2]=M2[c2]
    return M
```

R3. Si la première matrice contient c coefficients non nuls, et la seconde c', quelle est la complexité temporelle de cet algorithme ?

Solution: en $O(c, c')$. Sinon, pour une matrice écrite avec une liste de liste c'est en $O(np)$ (deux boucles for imbriquées.

Exercice n°6 Double hachage ♪ ♪

Le double hachage est l'une des meilleures méthodes connues pour l'adressage ouvert. Il utilise une fonction de hachage de la forme :

$$\begin{array}{rccc} h: & \mathbb{N}\times\mathbb{N} & \longrightarrow & [\![0, m-1]\!] \\ & (k,i) & \mapsto & \Big(h_1(k) + i \times h_2(k)\Big) \bmod m \end{array}$$

où h_1 et h_2 sont des fonctions de hachages.
i prend les valeurs suivantes :

— Par défaut, $i = 0$.

— S'il y a collision, on incrémente i de 1, jusqu'à ne plus avoir de collision pour la clé considérée.

— Il reprend la valeur 0 pour la clé suivantes ...

R1. Insérer les clés : 5, 28, 19, 15, 20, 33, 12, 17, 10 dans un tableau de taille $m = 13$ avec $h_1(k) = k \bmod 13$ et $h_2(k) = 1 + \Big(k \bmod 12\Big)$.

Solution:

- 5 : $h_1(5) = 5\%13 = 5$; $h_2(5) = 1 + (5\%12) = 6$; $h(5) = (5 + 0 \times 6)\%13 = 5$
- 28 : $h_1(28) = 28\%13 = 2$; $h_2(28) = 1 + (28\%12) = 5$; $h(28) = (2 + 0 \times 5)\%13 = 2$
- 19 : $h_1(19) = 19\%13 = 6$; $h_2(19) = 1 + (19\%12) = 8$; $h(19) = (6 + 0 \times 8)\%13 = 6$
- 15 : $h_1(15) = 15\%13 = 2$; $h_2(15) = 1 + (15\%12) = 4$; $h(15) = (2 + 0 \times 4)\%13 = 2$.
  Il y a collision, on incrémente i de 1, $i = 1$.
  $h(15) = (2 + 1 \times 4)\%13 = 6$, il y a collision, on incrémente i de 1 : $i = 2$
  $h(15) = (2 + 2 \times 4)\%13 = 10$, ok
- 20 : $h_1(20) = 20\%13 = 7$; $h_2(20) = 1 + (20\%12) = 9$; $h(20) = (7 + 0 \times 9)\%13 = 7$
- 33 : $h_1(33) = 33\%13 = 7$; $h_2(33) = 1 + (33\%12) = 10$; $h(33) = (7 + 0 \times 10)\%13 = 7$
  Il y a collision, on incrémente i de 1, $i = 1$.
  $h(33) = (7 + 1 \times 10)\%13 = 4$.
- 12 : $h_1(12) = 12\%13 = 12$; $h_2(12) = 1 + (12\%12) = 1$; $h(12) = (12 + 0 \times 1)\%13 = 12$
- 17 : $h_1(17) = 17\%13 = 4$; $h_2(17) = 1 + (17\%12) = 6$; $h(17) = (4 + 0 \times 6)\%13 = 4$
  Il y a collision, on incrémente i de 1, $i = 1$.
  $h(17) = (4 + 1 \times 6)\%13 = 10$, il y a collision, on incrémente de 1 : $i = 2$
  $h(17) = (4 + 2 \times 6)\%13 = 3$

- 10 : $h_1(10) = 10\%13 = 10$; $h_2(10) = 1 + (10\%12) = 11$; $h(10) = (10 + 0 \times 11)\%13 = 10$
  Il y a collision, on incrémente i de 1, $i = 1$.
  $h(10) = (10 + 1 \times 11)\%13 = 8$.

indice 0	
indice 1	
indice 2	28
indice 3	17
indice 4	33
indice 5	5
indice 6	19
indice 7	20
indice 8	10
indice 9	
indice 10	15
indice 11	
indice 12	12

R2. Proposer une fonction en `Python` qui prend en argument une clé `c` (entier) et la taille `m` de la table, et renvoie la valeur de $h(c)$.

Solution:

```
import numpy as np
m=13
tab_h=np.zeros(m) # table de hachage initialisée

def double_hachage(c,m):
    if c==0:
        tab_h[0]=c
        return 0
    i=0
    h1=c%m
    h2=1+(c%(m-1))
    h=(h1+i*h2)%m
    while tab_h[h]!=0: # si coefficient non nul dans le tableau, il y a collision
        i=i+1 # incrémente i de un
        h=(h1+i*h2)%m # on calcule la nouvelle valeur de hachage
    tab_h[h]=c # ajout de c dans la case h
    return h

L=[5,28,19,15,20,33,12,17,10]
for x in L:
    double_hachage(x,13)
>>> tab_h
array([ 0.,  0., 28., 17., 33.,  5., 19., 20., 10.,  0., 15.,  0.,
   12.])
```

Exercice n°7 Polynôme ♪ ♪ ♪

On considère des polynômes non nuls à coefficients entiers de degré quelconque mais qui ne contiennent pas plus de cinq monômes. On utilise un tableau de longueur $16 = 8 \times 2$ pour stocker les couples (degré, coefficient) dans lequel on pourrait stocker au maximum huit couples. Les places non occupées contiennent la valeur -1.

La fonction de hachage h est la fonction identité : pour tout $n \in \mathbb{N}$, $h(n) = n$. Donc à un degré qui vaut 10, on associe le nombre 10, soit $h(10) = 10$.
Ensuite, on écrit le degré (la clé), suivi du coefficient(la valeur) à l'indice $(10 \bmod 8) = 2$.
Par exemple, le polynôme $8 + 3x^{10} - 5x^{12}$ est stocké dans un tableau de la forme :

indice 0	0	8
indice 1	-1	-1
indice 2	10	3
indice 3	-1	-1
indice 4	12	-5
indice ...	...	...

R1. Donner le tableau correspondant au stockage du polynôme $2x^5 - 3x^{34} + 4x^{105}$.

Solution:
degré 5 : $h(5) = 5$, puis `5%8=5`
degré 34 : $h(34) = 34$, puis `34%8=2`
degré 105 : $h(105) = 105$, puis `105%8=1`

indice 0	-1	-1
indice 1	105	4
indice 2	34	-3
indice 3	-1	-1
indice 4	-1	-1
indice 5	5	2

R2. Quel est le problème avec par exemple le polynôme $8 - 5x^2 + 3x^{10}$?

Solution:
degré 0 : `0%8=0`
degré 2 : `2%8=2`
degré 10 : `10%8=2` => collision !

R3. En cas de collision, on décide d'utiliser la première place libre suivante. Les monômes sont entrés dans le tableau suivant l'ordre de lecture. Donner un exemple de polynôme de degré minimum qui génère une collision pour chaque monôme excepté le premier.

Solution: Il y a collision, si tous les restes des divisions euclidiennes des degrés sont égaux.
Par exemple : $x + x^2 + x^{10} + x^{26} + x^{34}$

R4. On envisage une autre possibilité de stockage avec deux tableaux, un tableau pour les couples (degré, coefficient) et un tableau pour les indices, les deux tableaux ayant pour capacité 8.
Avec le polynôme $4x^3 - 2x^5 + 4x^9$, on obtient les deux tableaux de la manière suivante :

- dans le premier tableau, on écrit chaque degré avec le coefficient correspondant suivant l'ordre des degrés et on complète le tableau avec des 0 ;
- dans le second tableau, on calcule $(d \bmod 8)$ où d est un degré et on place à l'indice trouvé l'indice où on trouve le couple (degré,coefficient) dans le premier tableau. On complète le tableau avec des -1. Extraits des tableaux :

indice 0	3	4
indice 1	5	-2
indice 2	9	4
indice 3	0	0
indice ...	...	...

indice 0	-1
indice 1	2
indice 2	-1
indice 3	0
indice 4	-1
indice 5	1

Donner les deux tableaux correspondant au stockage du polynôme $3x^5 - x^{18} + 7x^{20}$.

Solution:

indice 0	5	3
indice 1	18	-1
indice 2	20	7
indice 3	0	0
indice 4	0	0
indice 5	0	0
indice 6	0	0
indice 7	0	0

indice 0	-1
indice 1	-1
indice 2	1
indice 3	0
indice 4	2
indice 5	-1
indice 6	-1
indice 7	-1

`5%8=3 18%8=2 20%8=4`