

TP 2 - CORRECTION

Dictionnaires

1 Dictionnaires

Exercice 1.1. *Le Scrabble est un jeu de lettres où l'on forme des mots et l'on marque des points en faisant la somme des valeurs de chaque lettre composant le mot.*

Dans la version française, les points sont les suivants :

- 1 point pour A, E, I, L, N, O, R, S, T et U ;
- 2 points pour D, G et M ;
- 3 points pour B, C et P ;
- 4 points pour F, H et V ;
- 8 points pour J et Q ;
- 10 points pour K, W, X, Y, Z.

1. Donner la commande permettant de créer un dictionnaire dont les clés sont les lettres (type `str`) et les valeurs sont les points correspondant suivant le barème ci-dessus.
2. Écrire une fonction `score(mot)` qui prend en argument une chaîne de caractères `mot` (en majuscules sans accent) et qui renvoie la valeur de ce mot.
On utilisera bien sûr le dictionnaire défini dans la question précédente.
3. On souhaite créer une fonction de hachage pour un mot, fondée sur le même barème que le Scrabble. On définit la valeur de hachage d'un mot par :

$$h(\text{mot}) = \left(\sum_{\text{lettre} \in \text{mot}} \text{points}[\text{lettre}] \right) \times \text{len}(\text{mot})$$

Autrement dit, on multiplie le score Scrabble du mot par sa longueur. Écrire une fonction `hachage(mot)` qui renvoie cette valeur de hachage.

4. On désire maintenant stocker des mots dans une table de hachage de taille $T = 20$.

On utilise la fonction de hachage :

$$h(\text{mot}) = (\text{score}(\text{mot}) \times \text{len}(\text{mot})) \bmod T$$

- (a) Écrire une fonction **hachage_index(mot, T=20)** qui renvoie l'indice de hachage d'un mot.
- (b) Pour gérer les collisions, on utilisera la méthode du chaînage : chaque case de la table contient une liste de mots ayant le même indice. Écrire une fonction **insérer(table, mot)** qui insère un mot dans la table de hachage.
- (c) Donner un exemple d'utilisation : création d'une table de taille 20 puis insertion des mots **PYTHON**, **LION**, **OURS** et **JEU**.

```
1. POINTS = {'A':1, 'B':3, 'C':3, 'D':2, 'E':1, 'F':4, 'G':2, 'H':4, 'I':1, 'J':8,
            'K':10, 'L':1, 'M':2, 'N':1, 'O':1, 'P':3, 'Q':8, 'R':1, 'S':1,
            'T':1, 'U':1, 'V':4, 'W':10, 'X':10, 'Y':10, 'Z':10}
```

```
2. def score(mot: str) -> int:
    """
    Renvoie la valeur du mot au scrabble.
    """
    total = 0
    for lettre in mot:
        total = total + POINTS[lettre]
    return total
```

3. Fonction de hachage simple :

```
def hachage(mot):
    return score(mot) * len(mot)
```

4. Table de hachage (taille $T = 20$) :

- (a) Fonction **hachage_index** :

```
def hachage_index(mot, T=20):
    return (score(mot) * len(mot)) % T
```

- (b) Insertion dans la table avec chaînage :

```
def inserer(table, mot):
    indice = hachage_index(mot, len(table))
    if table[indice] is None:
        table[indice] = []
    table[indice].append(mot)
```

- (c) Exemple d'utilisation :

```
T = 20
table = [None] * T

for mot in ["PYTHON", "LION", "OURS", "JEU"]:
    inserer(table, mot)

print(table)
```

Exercice 1.2. *Le tiroir des chaussettes est sens dessus dessous : elles ne sont pas appariées et il en manque peut-être certaines pour faire des paires. On représente les chaussettes dans une liste de chaînes de caractères représentant leur couleur, par exemple :*

```
tiroir = ['blanche', 'noire', 'noire', 'bleue', 'noire', 'blanche']
```

*Écrire une fonction **paires(tiroir)** qui renvoie le nombre de paires de chaussettes de même couleur que l'on peut former.*

Par exemple, avec la composition du tiroir donnée précédemment, le résultat sera 2.

En effet : une paire noire et une paire blanche, les deux restantes ne s'apparient pas.

```
def paires(tiroir: [str]) -> int:
    """
    Renvoie le nombre de paires de chaussettes de meme couleur que
    l'on peut former avec celles presentes dans le tiroir.
    """
    # On commence par compter le nombre de chaussettes
    # de chaque couleur
    # (fonction comptage comme exercice 1).
    couleurs = {}
    for chaussette in tiroir:
        if chaussette not in couleurs:
            couleurs[chaussette] = 1
        else:
            couleurs[chaussette] = couleurs[chaussette] + 1
    # Maintenant on compte le nombre de paires que
    # l'on peut former, couleur par couleur.
    nb_paires = 0
    for coloris in couleurs:
        nb_paires = nb_paires + couleurs[coloris]//2
    return nb_paires
```

2 Fonction de hachage

Exercice 2.1. *Des chaînes vers les entiers*

On propose ici un exemple simple de pré-traitement des chaînes de caractères avant hachage.

Nous allons nous servir du codage ASCII¹ qui est donné sous Python par la fonction `ord ( c )` qui renvoie le code ASCII sur 8 bits du caractère `c`.

*Inversement, l'instruction **chr(code)** renvoie le caractère associé à un code ASCII.*

1. *Écrire une fonction **chaîneversentier(ch)** qui prend une chaîne de caractères en argument et renvoie la valeur :*

$$\sum_{k=0}^{n-1} \text{ord}(ch[k])256^k.$$

Cette fonction renvoie des entiers de grande taille, par exemple :

chaîneentiers ('exemple de taille modérée')

renvoie :

639711425003567419095171883946955047862310670182554716698725.

2. On souhaite réserver une table de hachage de longueur m , la fonction de hachage étant la fonction f précédente.
Écrire une fonction **chaîneentiersmodulo**(ch, m) qui prend une chaîne de caractères ainsi qu'un entier m en arguments et renvoie la valeur $g : ch \rightarrow f(ch) \% m$ (reste de la division euclidienne).
3. (a) Générer une liste de 1000 chaînes de caractères de longueur 5 de façon aléatoire.
(b) Pour $m = 12$, vérifier sur les 1000 chaînes générées si la distribution de la fonction de hachage **chaîneentiersmodulo** est uniforme.

```
1. def chaîneversentier(ch):
    n,r=len(ch),0
    for i in range(1,n+1):
        r=256*r+ord(ch[n-i])
    return(r)

    def chaîneversentiermodulo(ch,m):

        return(chaîneversentier(ch)%m)

3. def generation(longueur,nbre):
    L=[]
    for i in range(nbre):
        chaîne=''
        for j in range(longueur):
            chaîne+=chr(rd.randint(0,256))
        L.append(chaîne)
    return(L)

    def distribution(longueur,nbre,m):
        L=generation(longueur, nbre)
        dico={}
        for chaîne in L:
            entier=chaîneversentiermodulo(chaîne, m)
            if entier in dico:
```

```

        dico[entier]+=1
    else:
        dico[entier]=1
    return(dico)

print(distribution(5, 100000, 12))

```

On obtient :

```

{0: 8569, 6: 8311, 9: 8196, 4: 8491, 7: 8172, 1: 8351, 11: 8271, 3: 8430,
 2: 8362, 8: 8472, 10: 8240, 5: 8135}

```

La distribution de la fonction de hachage semble donc bien uniforme.

Exercice 2.2. Table de hachage et gestion des collisions

On considère une table de hachage de taille $T = 17$, représentée en Python par une liste de 17 cases initialisées à **None**.

Pour insérer des éléments, on utilisera l'**adressage ouvert** avec **sondage linéaire** : en cas de collision, si une case est occupée, on essaie la suivante, puis encore la suivante, en revenant au début si nécessaire.

1. On définit la fonction de hachage suivante pour une chaîne de caractères **mot** composée de lettres majuscules :

$$h(\text{mot}) = \left(\sum_{k=0}^{n-1} (\text{ord}(\text{mot}[k]) - 64) \right) \bmod T$$

où n est la longueur du mot.

- (a) Calculer la valeur de hachage des mots "CHAT", "LION" et "TIGRE".
 - (b) Écrire une fonction Python `hachage(mot, T)` correspondant à cette définition.
2. On souhaite insérer un mot dans la table en utilisant le sondage linéaire :

$$i, i + 1, i + 2, \dots \pmod{T}$$

tant qu'une case est occupée.

- (a) Écrire une fonction `insérer(table, mot)` qui effectue cette insertion.
 - (b) Montrer que, dans le pire des cas, le nombre de comparaisons est $O(T)$.
3. On souhaite maintenant insérer successivement les mots :

LION, OURS, TIGRE, LOUP, LYNX

- (a) Donner, à chaque étape, l'état de la table (indices et mots stockés).
 - (b) Indiquer le nombre total de collisions rencontrées.
4. Discuter brièvement les avantages et inconvénients du sondage linéaire par rapport au chaînage.

Correction

1. (a) Calculs :

- **CHAT** : C=3, H=8, A=1, T=20 $\rightarrow$ somme = 32 $\rightarrow$ 32 mod 17 = 15
- **LION** : L=12, I=9, O=15, N=14 $\rightarrow$ somme = 50 $\rightarrow$ 50 mod 17 = 16
- **TIGRE** : T=20, I=9, G=7, R=18, E=5 $\rightarrow$ somme = 59 $\rightarrow$ 59 mod 17 = 8

(b) Fonction Python :

```
def hachage(mot, T):
    s = sum(ord(c) - 64 for c in mot)
    return s % T
```

2. (a) Insertion avec sondage linéaire :

```
def inserer(table, mot):
    T = len(table)
    i = hachage(mot, T)
    start = i
    while table[i] is not None:
        i = (i + 1) % T
        if i == start:
            raise Exception("Table pleine")
    table[i] = mot
```

(b) Analyse :

Dans le pire des cas, on parcourt toute la table avant de trouver une case vide :

$$O(T)$$

3. Insertion des mots successifs :

Mot	$h(mot)$
<i>LION</i>	16
<i>OURS</i>	5
<i>TIGRE</i>	8
<i>LOUP</i>	13
<i>LYNX</i>	7

Aucun indice ne se répète $\rightarrow$ aucune collision.

Table finale :

Indice	Mot
5	<i>OURS</i>
7	<i>LYNX</i>
8	<i>TIGRE</i>
13	<i>LOUP</i>
16	<i>LION</i>

Nombre total de collisions : $**0**$.

4. Comparaison sondage linéaire / chaînage :

- **Sondage linéaire :**

- + Simple, compact.
- – Agglutination, performances chutent quand la table est remplie.

- **Chaînage :**

- + Supporte mieux les fort taux de remplissage.
- – Besoin de listes, allocations supplémentaires.

Exercice 2.3. Multiplication

Une méthode consiste à construire des fonctions de hachage de la façon suivante :

- on se donne un réel (ou un flottant) $\theta \in]0, 1[$ et un entier $m > 0$.
- on définit $h : e \in \mathbb{N}^* \rightarrow \lfloor m \times ((e \times \theta) \bmod 1) \rfloor$ (où $x \bmod 1 = x - \lfloor x \rfloor$).

On se propose d'en explorer quelques propriétés.

1. Quelle sera la taille d'une table de hachage associée à une fonction multiplicative de cette forme ?
2. On décide de tester la répartition des clés dans la table en fonction de θ .

- (a) Définir une fonction hachage (m, θ, c) qui prend en arguments m un entier strictement positif, $0 < \theta < 1$ un flottant, c entier et qui réalise le hachage de c .
- (b) Écrire une fonction **testhachage**(**m**, **theta**) qui calcule pour chaque entier i compris entre 0 et $m - 1$ le nombre de valeurs $c \in \llbracket 0, 10^5 \rrbracket$ telles que $h(c) = i$, soit :

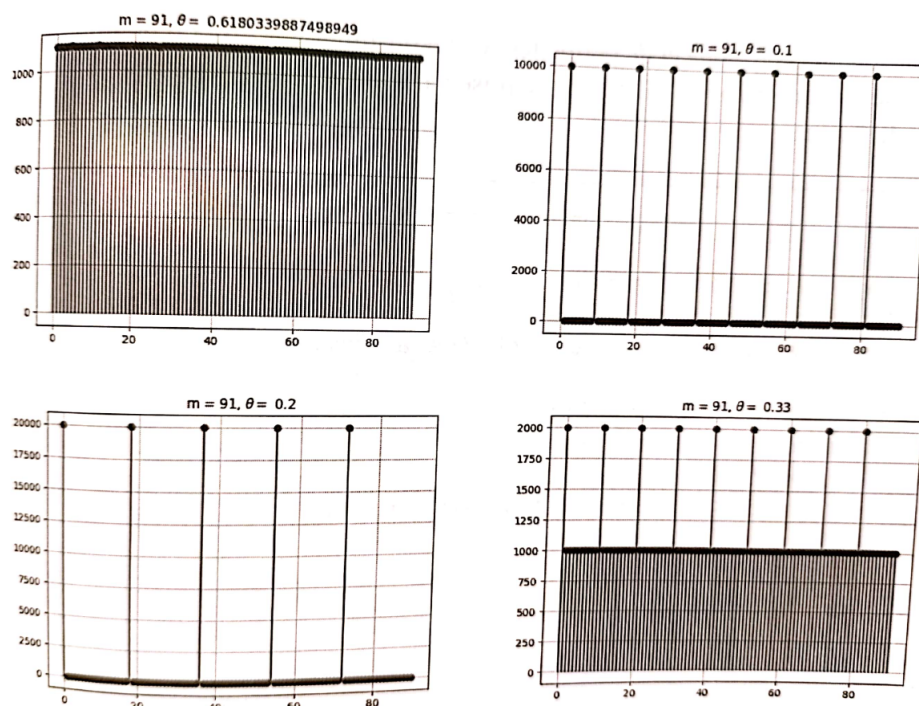
$$d_i = |\{c \in \llbracket 0, 10^5 \rrbracket / h(m, \theta, c) = i\}|.$$

Ces valeurs seront stockées dans un dictionnaire que renverra la fonction.

- (c) Les quatre histogrammes ci-dessous représentent les répartitions obtenues (i en abscisse, et d_i en ordonnée) avec :

$$m = 91 \text{ et } \theta \in \left\{ \frac{\sqrt{5}-1}{2}, 0.1, 0.2, 0.33 \right\}.$$

Expliquer les résultats lorsque $\theta = 0.1, 0.2$.



(d) Justifier que $c \rightarrow h(m, a/b, c)$ prend au plus $\max(m, b)$ valeurs où a et b sont deux entiers tels que $a < b$.
Est-ce cohérent avec ce que l'on observe ? **Conseil :** Il pourra être intéressant d'observer la répartition des collisions lorsque $\theta = 0.33$ et pour différentes valeurs de m (plus petites, plus grandes que 100).

1. Il est clair que $h(e) \in \llbracket 0, m-1 \rrbracket$. On pourra donc réserver un tableau de longueur m .

2. (a)

```
def hachage(m, theta, c):
    return(m*(theta*c%1))

def testhachage(m, theta):
    d={i:0 for i in range(0,m+1)}
    print(d)
    for x in range(0,10**5):
        h=hachage(m, theta, x)
        d[h]+=1
    return(d)
```

(b) On a ici $m = 91$.

- Cas où $\theta = 1/10$. En posant $c = 10q + r$ avec $0 \leq r < 10$, il vient :

$$h(c) = \lfloor 91 \times r/10 \rfloor \in \{0, 8, 18, 27, 36, 40, 54, 63, 72, 81\}.$$

La fonction prend donc 10 valeurs comme on peut le remarquer sur le graphique.

- Le cas $\theta = 1/5$ se traite de la même façon.
- (d) Soit $e = bq + r$ avec $0 \leq r < b$ par division euclidienne de e par b :

$$\theta e = \frac{a}{b}(bq + r) = aq + \frac{r}{b}$$

avec r un entier compris entre 0 et $b - 1$ et $aq \in \mathbb{N}$.

Or $\frac{r}{b} < 1$. La partie décimale de θe est donc $\frac{r}{b}$.

Ce qui, après multiplication par m , nous fait au plus b valeurs distinctes pour $h(e)$.

Et comme par ailleurs, $h(m, a/b, e)$ est un entier compris entre 0 et $m - 1$, il prend au plus m valeurs.

En conclusion, $h(m, a/b, e)$ prend au plus $\min(m, b)$ valeurs distinctes (comprises en 0 et m).

- (e) Une exploration avec différentes valeurs de m et de θ semble montrer que les phénomènes de collisions dépendent de la valeur de θ .

Les cas à éviter sont les valeurs $\theta = \frac{p}{q}$ avec q trop petit.

Dans tous les cas le nombre d'or semble fournir une répartition quasi-uniforme.