Devoir Maison d'ITC N°3
pour la semaine du 04/03/2024
à rendre en séance de TP

Consignes

- La qualité, la lisibilité et l'efficacité du code entreront pour une part importante dans l'appréciation de la copie. L'indentation du code doit figurer clairement sur la copie, il est fortement conseillé d'y ajouter une barre verticale sur la gauche afin de préciser clairement les portions de code ayant la même indentation.
- **Le crayon à papier ne sera pas corrigé.**
- **Il est rappelé également que recopier une correction sur internet est complètement inutile pour tout le monde.**

Toutes les fonctions devront avoir *a minima* une signature.

Problème **Table de hachage** [Sol] Un *dictionnaire* est un ensemble d'objets `<clé>:<valeur>`. Nous supposerons que les clefs sont des chaînes de caractères formées à partir de l'alphabet ASCII, soit 256 caractères.

Le *hachage* consiste à associer un entier à chacune des clés d'un dictionnaire et ce de manière bijective. L'ensemble de ces entiers s'appelle une *table de hachage* et permet de manipuler plus facilement les éléments d'un dictionnaire.

À chaque chaîne de caractère, on associe l'entier correspondant à la décomposition de la clé en base 256. Par exemple, à la clé `"pouet"`, constituée des caractères `p`, `o`, `u`, `e`, `t` dont les valeurs ASCII sont 112,111,117,101,116, on associe l'entier :

$$n = 112\times256^4 + 111\times256^3 + 117\times256^2 + 101\times256^1 + 116\times256^0 = 482906301812.$$

On rappelle que :

- Si `c` est un caractère de l'alphabet ASCII, l'instruction `ord(c)` renvoie l'entier appartenant à $\{0,1,..,255\}$ correspondant. Si `n` est un entier, l'instruction `chr(n)` renvoie le caractère de l'alphabet `ASCII` correspondant. Par exemple,

```
>>> ord("a")
97
>>> ord("à")
224
>>> chr(97)
'a'
>>> chr(255)
'ÿ'
```

- Pour $(a,b)\in\mathbb{N}^2$, $a\%b$ (*resp.* $a//b$) est le reste (*resp.* quotient) de la division euclidienne de a par b.

1. **1.1)** Écrire une fonction `ch_int(ch)` d'argument `ch` une chaîne de caractère et qui retourne l'entier construit selon le principe décrit ci-dessus. *Le nombre de multiplications devra être en* $\mathrm{O}(\mathrm{N})$ *où* N *désigne la longueur de la chaîne* `ch`. *On vérifiera cela une fois la fonction écrite.*

1.2) On rappelle que l'algorithme de HÖRNER repose sur la factorisation suivante d'un polynôme :

$$\begin{aligned}&a_0 + a_1x + a_2x^2 + .. + a_nx^n\\ &= ((((a_nx + a_{n-1})x + a_{n-2})x + \ldots)x + a_1)x + a_0.\end{aligned}$$

Écrire une fonction `ch_int_hor` d'argument ch une chaîne de caractère et qui réalise la même action que `ch_int(ch)`, mais qui utilise l'algorithme de HÖRNER.

1.3) Écrire une fonction `int_ch` d'argument un entier `n`, et qui réalise l'opération inverse de celle réalisée par `ch_int`.

2. Le procédé de fabrication de la table de hachage donné dans la question précédente génère des entiers « longs ». Pour éviter ce problème, on utilise une fonction de hachage qui « raccourcit » les entiers de la table de hachage. Dans la suite, on utilise la fonction de hachage *modulo 255* définie par :

$$\forall n\in\mathbb{N},\quad h(n) = n\%255.$$

2.1) Écrire une fonction `hach(ch)` qui prend en argument une chaîne de caractère `ch` et qui retourne $h(n)$ où n est l'entier de la table de hachage correspondant à la chaîne de caractère `ch`.

2.2) Prouver que si `ch1` est un anagramme (*i.e.* les deux chaînes sont identiques à permutation des lettres près) de la chaîne de caractère `ch`, on a alors `hach(ch) = hach(ch1)`.

2.3) Écrire une fonction `doub(L)`, d'argument `L` une liste d'entiers et qui retourne `True` si `L` contient au moins deux éléments identiques. *La fonction devra éviter l'arrêt de boucle par* `return`.

2.4) Écrire une fonction `dicovalid(dict)` d'argument un dictionnaire et qui renvoie `True` si la table de hachage construite à l'aide de la fonction `hach` ne contient pas de doublons, `False` dans le cas contraire.

Correction
du Devoir Maison d'ITC N°1

Solution

1. 1.1)

```
def ch_int(ch:str)->int:
    N = len(ch)
    n = 0
    p = 1
    for k in range(N):
        n += ord(ch[N-1-k])*p
        p *= 256
    return n
>>> ch_int("pouet")
482906301812
```

On a deux multiplications par itération, donc au total $\sum_{k=0}^{N-1} 2 = \boxed{2N}$ multiplications.

1.2)

```
def ch_int_hor(ch:str)->int:
    N = len(ch)
    n = 0
    for k in range(N):
        n = n*256 + ord(ch[k])
    return n
>>> ch_int_hor("abc")
6382179
>>> ch_int_hor("pouet")
482906301812
```

1.3)

```
def int_ch(n:int)->str:
    k = n
    ch = ""
    r = k%256 # chiffre le plus à droite
    while r != 0:
        ch = chr(r) + ch # ajout au début
        k = (k-r)//256
        r = k%256
    return ch
>>> int_ch(482906301812)
'pouet'
```

2. 2.1)

```
def hach(ch:str)->int:
    n = ch_int(ch)
    return n%255
>>> hach("z")
122
```

2.2) Notons ch = "$c_{N-1}c_{N-2}\dots c_0$" où N désigne `len(ch)`. Alors l'entier associé est par hypothèse :

$$\begin{aligned} n &= \sum_{i=0}^{N-1} \mathrm{ord}(c_i) \times 256^i = \sum_{i=0}^{N-1} \mathrm{ord}(c_i) \times (255+1)^i \\ &= \sum_{i=0}^{N-1} \mathrm{ord}(c_i) \sum_{k=0}^{i} \binom{i}{k} 255^k \\ &\equiv \sum_{i=0}^{N-1} \mathrm{ord}(c_i)\ [255], \end{aligned}$$

modulo 255

puisque tous les $255^k, k \geqslant 1$, sont congrus à 0 modulo 255. Mais si ch1 = "$d_{N-1}d_{N-2}\dots d_0$" où N désigne `len(ch)`, qui est aussi `len(ch1)` lorsque ch1 et ch2 sont des anagrammes, alors l'entier n_1 associé à ch1 vérifie :

$$n_1 \equiv \sum_{i=0}^{N-1} \mathrm{ord}(d_i)\ [255].$$

Puisque $\sum_{i=0}^{N-1} \mathrm{ord}(d_i) = \sum_{i=0}^{N-1} \mathrm{ord}(c_i)$ car ch est une anagramme de ch2, on déduit que : $\boxed{\texttt{hach(ch) = hach(ch1)}}$.

2.3)

```
def doub(L:list)->bool:
    n = len(L)
    doublon = False
    i = 0
    while i < n-1 and not doublon:
        x = L[i]
        # recherche de la présence de x dans L[i+1:]
        j = i+1
        while j < n and not doublon:
            if x == L[j]:
                doublon = True
            else:
                j += 1
        i += 1
    return doublon
```

```
>>> doub([1, 2, 3])
False
>>> doub([1, 2, 2])
True
```

2.4)

```
def dicovalid(D:dict)->bool:
    table = []
    for c in D:
        table.append(hach(c))
    return not doub(table)
```

```
>>> D = {"a":0, "b":1, "c":2}
>>> dicovalid(D)
True
>>> D = {"romain":0, "manoir":1}
>>> dicovalid(D)
False
```