td1a_correction_session3

July 1, 2022

1 1A.1 - Dictionnaires, fonctions, code de Vigenère (correction)

Le notebook ne fait que crypter et décrypter un message sachant le code connu. Casser le code requiert quelques astuces décrites dnas ce notebook : casser le code de Vigenère.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

1.0.1 Exercice 1

[2]:
```
def lettre_suivante(lettre) :
    c = ord(lettre) - ord('a')
    c = (c + 1) % 26
    return chr (c + ord('a'))

print (lettre_suivante('m'), lettre_suivante('z'))
```

```
n a
```

1.0.2 Exercice 2

[3]:
```
mots = ['eddard', 'catelyn', 'robb', 'sansa', 'arya', 'brandon',
        'rickon', 'theon', 'rorbert', 'cersei', 'tywin', 'jaime',
        'tyrion', 'shae', 'bronn', 'lancel', 'joffrey', 'sandor',
        'varys', 'renly', 'a' ]

def mots_lettre_position (liste, lettre, position) :
    res = [ ]
    for mot in liste :
        if position < len(mot) and mot[position] == lettre :
            res.append (mot)
    return res

r = mots_lettre_position ( mots, 'y', 1)
print (r)
```

```
['tywin', 'tyrion']
```

1.0.3 Exercice 3 : utilisation d'un dictionnaire

L'énoncé suggère d'utiliser comme clé de dictionnaire le couple `(position, lettre)` et la fonction doit retourne la liste des mots qui ont tous la même lettre à la même position. Le dictionnaire `dictionnaire_bien_choisi` de l'énoncé doit avoir pour clés des couples `(position, lettre)` et pour valeurs des listes de prénoms.

[4]:
```
def dictionnaire_choisi (liste) :
    d = { }
    for mot in liste :
        for i,c in enumerate(mot) :
            d [i,c] = d.get ((i,c), []) + [ mot ]
    return d

def mots_lettre_position (d, lettre, position) :
    return d.get ( (position, lettre), [] )

d = dictionnaire_choisi(mots)
r = mots_lettre_position ( d, 'y', 1)
print ("résultat=",r)
print ("dictionnaire=",d)
```

```
résultat= ['tywin', 'tyrion']
dictionnaire= {(0, 'l'): ['lancel'], (4, 'y'): ['renly'], (0, 'e'): ['eddard'],
(4, 'n'): ['theon', 'tywin', 'bronn'], (1, 'd'): ['eddard'], (4, 'l'):
['catelyn'], (3, 'd'): ['sandor'], (1, 'r'): ['arya', 'brandon', 'bronn'], (2,
'o'): ['bronn'], (3, 'k'): ['rickon'], (2, 'y'): ['arya'], (2, 'e'): ['theon'],
(1, 'y'): ['tywin', 'tyrion'], (1, 'h'): ['theon', 'shae'], (2, 'r'):
['rorbert', 'cersei', 'tyrion', 'varys'], (3, 'y'): ['varys'], (3, 'o'):
['theon'], (2, 'd'): ['eddard'], (4, 'd'): ['brandon'], (1, 'e'): ['cersei',
'renly'], (2, 'w'): ['tywin'], (0, 't'): ['theon', 'tywin', 'tyrion'], (6, 't'):
['rorbert'], (5, 'l'): ['lancel'], (6, 'n'): ['catelyn', 'brandon'], (5, 'e'):
['joffrey'], (0, 's'): ['sansa', 'shae', 'sandor'], (0, 'r'): ['robb', 'rickon',
'rorbert', 'renly'], (2, 'c'): ['rickon'], (0, 'j'): ['jaime', 'joffrey'], (3,
'n'): ['brandon', 'bronn'], (3, 'i'): ['tywin', 'tyrion'], (0, 'c'): ['catelyn',
'cersei'], (1, 'o'): ['robb', 'rorbert', 'joffrey'], (4, 'e'): ['rorbert',
'cersei', 'jaime', 'lancel'], (5, 'd'): ['eddard'], (4, 'r'): ['eddard',
'joffrey'], (3, 'b'): ['robb', 'rorbert'], (2, 'a'): ['brandon', 'shae'], (5,
'o'): ['brandon'], (4, 'a'): ['sansa'], (5, 'i'): ['cersei'], (2, 't'):
['catelyn'], (3, 'f'): ['joffrey'], (3, 'c'): ['lancel'], (6, 'y'): ['joffrey'],
(3, 'm'): ['jaime'], (2, 'f'): ['joffrey'], (5, 'r'): ['rorbert', 'sandor'], (0,
'v'): ['varys'], (1, 'a'): ['catelyn', 'sansa', 'jaime', 'lancel', 'sandor',
'varys'], (2, 'i'): ['jaime'], (3, 'a'): ['eddard', 'arya'], (0, 'b'):
['brandon', 'bronn'], (2, 'b'): ['robb'], (5, 'n'): ['rickon', 'tyrion'], (4,
's'): ['varys'], (5, 'y'): ['catelyn'], (4, 'o'): ['rickon', 'tyrion',
'sandor'], (3, 'e'): ['catelyn', 'shae'], (2, 'n'): ['sansa', 'lancel',
'sandor', 'renly'], (3, 'l'): ['renly'], (3, 's'): ['sansa', 'cersei'], (0,
'a'): ['arya', 'a'], (1, 'i'): ['rickon']}
```

S'il permet d'aller beaucoup plus vite pour effectuer une recherche, le dictionnaire `d` contient beaucoup plus de mots que la liste initiale. Si on suppose que tous les mots sont uniques, il en contient exactement autant que la somme des longueurs de chaque mot.

A quoi ça sert ? Tout dépend du nombre de fois qu'on n'effectue ce type de **recherche**. Il faut d'abord décomposer les deux méthodes en coût fixe (préparation du dictionnaire) et coût recherche puis regarder la page Time Complexity. On obtient :

- liste de l'exercice 2 : coût fixe = 0, coût variable $\sim O(N)$
- dictionaire de l'exercice 3 : coût fixe $\sim O(L)$, coût variable $\sim O(1)$

Où :

- N est le nombre de mots,
- L est la somme des nombres de lettres de chaque mot,
- M est la longueur maximale d'un mot.

Les dictionnaires en Python utilisent une table de hashage pour stocker les clés. L'objet `map` de Python ne rapproche plus de l'objet `unordered_map` de C++ que de l'objet `map`. Ce dernier (C++ uniquement) est un tableau trié. L'accès à chaque élément se fait par dichotomie en $O(\ln_2 n)$ (voir Standard C++ Containers. Le coût dans ce cas serait (toujours en C++) :

- dictionaire de l'exercice 3 : coût fixe $\sim O(L\, ln_2(26 * M))$, coût variable $\sim O(ln_2(26 * M))$

Si on effectue cette recherche un grand nombre de fois, l'utilisation d'un dictionnaire permet d'être beaucoup plus rapide même si on doit créer une structure intermédiaire. Ce schéma revient régulièrement : **représenter autrement les données pour accélérer un traitement effectué un grand nombre de fois**.
Vous pouvez lire également :

- hash
- STL Container Performance
- C++11: unordered_map vs map
- AVL tree
- List of data structures
- Time complexity of accessing a Python dict
- Hash Table Performance Tests
- How to implement a good **hash** function in python

1.0.4 Exercice 4 : crypter et décrypter selon Vigenère

Tout d'abord le code de César :

[5]:
```
def code_cesar(m):
    s = "".join( [ chr((ord(l)-65+3)%26+65) for l in m ] )
    return s

m = "JENESUISPASCODE"
print(code_cesar(m))
```

```
MHQHVXLVSDVFRGH
```

Et le code de Vigenère :

[6]:
```
def code_vigenere ( message, cle) :
    message_code = ""
    for i,c in enumerate(message) :
        d = cle[ i % len(cle) ]
        d = ord(d) - 65
        message_code += chr((ord(c)-65+d)%26+65)
    return message_code

m = "JENESUISPASCODE"
print ( code_vigenere (m, "DOP") )
```

```
MSCHGJLGEDGRRRT
```

Et le décryptage du code de Vigenère pour lequel on modifie la fonction précédente qui pourra alors coder et décoder.

[7]:
```
def code_vigenere ( message, cle, decode = False) :       # ligne changée
    message_code = ""
    for i,c in enumerate(message) :
        d = cle[ i % len(cle) ]
        d = ord(d) - 65
        if decode : d = 26 - d                            # ligne ajoutée
        message_code += chr((ord(c)-65+d)%26+65)
    return message_code

m = "JENESUISPASCODE"
c = code_vigenere (m, "DOP")
d = code_vigenere (c, "DOP", True)
print(c,d)
```

```
MSCHGJLGEDGRRRT JENESUISPASCODE
```

Pour retrouver le code de César, il suffit de choisir une clé d'une seule lettre :

[8]:
```
c = code_vigenere (m, "D")
print(c)
```

```
MHQHVXLVSDVFRGH
```

On peut casser le code de Vigenère. Vous trouverez la solution ici : casser le code de Vigenère.

[9]:
```

```