tp_ITC_dictionnaire_2022

November 5, 2024

1 Collisions

La fonction `hash` est utilisée par `Python` pour calculer les valeurs de hachage des clés de dictionnaires.

Nous savons que les capacités des dictionnaires sont des puissances de 2, et que la capacité est toujours au moins 1.5 fois plus grande que le nombre de clés utilisées. Ainsi un dictionnaire de 5 clés et moins a une capacité de 8.

Dans cet exercice, on suppose que **le tableau sous-jacent est de taille 8**.

Dans un tel dictionnaire, la position du tuple (clé,valeur) est déterminée par `hash(clé)%8`. Le risque de collision est alors fréquent en particulier, si les clés sont prises parmi les caractères de `"a"` à `"z"`.

- Q1. On cherche à déterminer toutes les collisions possibles parmi les clés constituées d'une seule lettre.Ecrire une fonction `doublons()` qui retourne la liste des tuples (x, y) où x, y sont deux lettres minuscules telles que $x < y$ et vérifiant que le hachage de x soit égal au hachge de y.

Voici deux fonctions Python très utiles pour cet exercice :

[30]:
```
ord("a"),ord("p"), chr(98), chr(ord("a"))
```

[30]:
```
(97, 112, 'b', 'a')
```

[7]:
```
ord("Z"), ord("a")
```

[7]:
```
(90, 97)
```

[1]:
```
#*****************
def doublons():
    l = []
    for i in range(ord("a"),ord("z")+1):
        for j in range(i+1,ord("z")+1):
            if hash(chr(i))%8==hash(chr(j))%8:
                l.append((chr(i),chr(j)))
    return l
```

[2]:
```
print(doublons())
```

```
[('a', 'g'), ('a', 'm'), ('a', 'r'), ('a', 's'), ('b', 'l'), ('b', 'x'), ('c',
'e'), ('c', 'f'), ('c', 'h'), ('d', 'w'), ('d', 'z'), ('e', 'f'), ('e', 'h'),
('f', 'h'), ('g', 'm'), ('g', 'r'), ('g', 's'), ('i', 'j'), ('i', 'n'), ('i',
'q'), ('i', 't'), ('i', 'v'), ('j', 'n'), ('j', 'q'), ('j', 't'), ('j', 'v'),
('k', 'p'), ('l', 'x'), ('m', 'r'), ('m', 's'), ('n', 'q'), ('n', 't'), ('n',
'v'), ('q', 't'), ('q', 'v'), ('r', 's'), ('t', 'v'), ('u', 'y'), ('w', 'z')]
```

Attention, la fonction `hash` est réinitialisée à chaque session, ce qui fait que mes valeurs et les vôtres seront sans doute différente.

2 Extrêmums

- Q1. Ecrire une fonction `min_max` qui prend en paramètres une liste de nombres non vide et renvoie un dictionnaire dont les clés sont les chaînes de caractères `"min"` et `"max"`. les valeurs correspoondantes sont bien sûr le minimum et le maximum.

[1]:
```python
# ***********
def min_max(t):
    d={"min":t[0],"max":t[0]}
    for e in t:
        if e<d["min"]:
            d["min"]=e
        if e>d["max"]:
            d["max"]=e
    return d
```

3 Comparer

On veut comparer deux listes de nombres de même type (`int` ou `float`)

1. Ecrire une fonction `occurrences(t)` qui prend en paramètre une liste de nombre et retourne le dictionnaire des nombres d'occurrences.

[6]:
```python
# *****************
def comparer_naif(l1, l2):
    if len(l1)!=len(l2):
        return False
    l1 = sorted(l1)#O(n log n)
    l2 = sorted(l2)
    for i in range(len(l1)):
        if l1[i]!=l2[i]:
            return False
    return True # pseudo linéaire
```

[7]:
```
# ********
def occurrences(t):#O(n)
    d={}
    for e in t:
        if e in d:
            d[e]+=1
        else :
            d[e]=1
    return d
```

[8]:
```
occurrences([-1,3,-1,2,5,3,-1,2])
```

[8]:
```
{-1: 3, 3: 2, 2: 2, 5: 1}
```

2. Ecrire une fonction `taille(d)` qui prend en paramètres un dictionnaire comme celui obtenu par `occurrences` et retourne le nombre d'éléments de la liste.

[9]:
```
#***********
def taille(d):
    s=0
    for e in d:
        s+=d[e]
    return s
```

[10]:
```
d={-1: 3, 3: 2, 2: 2, 5: 1}
taille(d)
```

[10]:
```
8
```

3. Ecrire une fonction `cmp(t1,t2)` qui prend en paramètres deux listes de nombres et retourne `True` si les deux listes ont les mêmes éléments (dans l'ordre ou pas). Estimer la complexité.

****

Complexité linéaire en $O(|t_1|+|t_2|)$ pour construire les dicos des occurrences. Le parcours des deux dictionnaires est en temps lineaire. Donc complexité en $O((|t_1|)$ (puisque les listes ont même taille, sinon on n'explore pas leur contenu).

[11]:
```
#**********
def cmp(t1,t2):
    if len(t1)!=len(t2):
        return False
    d1=occurrences(t1)
    d2=occurrences(t2)
    for e in d1:
        #if e not in d2 or d2[e] != d1[e]):
        if not (e in d2 and d2[e]==d1[e]):
```

```
            return False
    return True

def cmp2(t1,t2):
    """version plus courte utilisant
    les capacités de python à comparer des
    dicos
    """
    if len(t1)!=len(t2):
        return False
    d1=occurrences(t1)
    d2=occurrences(t2)
    return d1==d2
```

[12]:
```
d1={1:2,2:6}
d2={2:6,1:2}
d1==d2
```

[12]: True

[13]:
```
cmp([-1,3,-1,2,5,3,-1,2],[-1,-1,-1,2,2,3,3,5])
```

[13]: True

[14]:
```
cmp([-1,3,-1,2,5,3,-1,2],[-1,-1,-1,2,2,3,3,5,6])
```

[14]: False

[15]:
```
cmp([-1,3,-1,2,5,3,-1,2],[-1,-1,-1,1,2,2,3,3])
```

[15]: False

4 Simuler la fonction hash

4.0.1 Q1

Importer le module `sys` et inspecter le retour de `sys.hash_info` pour trouver la valeur de `modulus`. Ce nombre, noté M dans le cours, est utile notamment pour le hachage des flottants.

[2]:
```
#*****
import sys
print("voici ce que ça donne sur ma machine")
sys.hash_info
```

```
voici ce que ça donne sur ma machine
```

[2]:
```
sys.hash_info(width=64, modulus=2305843009213693951, inf=314159, nan=0,
imag=1000003, algorithm='siphash13', hash_bits=64, seed_bits=128, cutoff=0)
```

[3]:
```
#*********
sys.hash_info.modulus
```

[3]: 2305843009213693951

[4]:
```
#**************
import numpy as np
np.log2(sys.hash_info.modulus)
```

[4]: 61.0

[5]:
```
"""
La valeur du modulus donnée en cours :
il faut la retrouver en inspectant sys.hash_info
"""
2**61-1
```

[5]: 2305843009213693951

- Q2.Trouver par un programme le nombre n tel que `modulus` vaut $2^n - 1$ (on sait que c'est 61, on demande de faire comme si on ne savait pas).

[6]:
```
#******
exp, e = 1, 0
while exp-1!=sys.hash_info.modulus:
    exp, e=2*exp, e+1
print(e,exp)
```

```
61 2305843009213693952
```

[2]:
```
import sys
exp, e = 1, 0
while hash(exp)==exp :
    exp, e=2*exp, e+1
print(e,exp,exp-1)
```

```
61 2305843009213693952 2305843009213693951
```

[7]:
```
#***********
import numpy as np
np.log2(sys.hash_info.modulus)
```

[7]: 61.0

- Q3.

Poser `M=sys.hash_info.modulus`.

Ecrire une fonction `my_hash(p,q)` qui prend en paramètres deux entiers avec q non nul, ces entiers représentant le flottant $f = \frac{p}{q}$. La fonction renvoie `int(abs(p)*M/abs(q))%M` si $p \geq 0$ et l'opposé

de cette valeur sinon.

Comparer les résultats fournis par `my_hash` et ceux de `hash`:

```
my_hash(3,2), hash(3/2), hash(1.5)
my_hash(115,100), hash(1.15)
my_hash(-7,3), my_hash(7,-3), hash(-7/3)
```

[8]:
```
# **********

M=sys.hash_info.modulus

def my_hash(p,q):
    assert q!=0 and isinstance(q,int) and isinstance(p,int)
    if p/q >0:
        return int(abs(p)*M/abs(q))%M
    if p/q==-1:
        return -2
    return -(int(abs(p)*M/abs(q))%M)
```

[19]:
```
hash(M-1),hash(M+1)
```

[19]:
```
(2305843009213693950, 1)
```

[9]:
```
my_hash(3,2), hash(3/2), hash(1.5)
```

[9]:
```
(1152921504606846977, 1152921504606846977, 1152921504606846977)
```

[10]:
```
my_hash(115,100), hash(1.15)
```

[10]:
```
(345876451382053889, 345876451382053889)
```

[11]:
```
my_hash(-7,3), my_hash(7,-3), hash(-7/3)
```

[11]:
```
(-768614336404564994, -768614336404564994, -768614336404564994)
```

[15]:
```
hash(-1), my_hash(-1,1)
```

[15]:
```
(-2, -2)
```

5 Gestion des collisions

5.1 Hachage de polynômes

On considère des polynômes non nuls à coefficients entiers de degré quelconque mais ne comportant pas plus de 5 monômes non nuls. Pour simuler un dictionnaire, on stocke les couples (degré, coeff) dans un tableau de longueur 8 (on peut y stocker jusqu'à 8 couples). Les clés sont les degrés.

Le tableau de tuples est initialisé avec la valeur -1.

La fonction de hachage utilisée est l'identité. Par exemple, si on considère le polynôme $3x^{11}$, la clé est 11, sa valeur de hachage est $h(11) = 11$ et on stocke le tuple (11,3) à l'indice 11%8.

Voici un exemple de stockage pour le polynôme $8 + 3x^{10} - 5x^{12}$

indice	degré	coeff
0	0	8
1	−1	−1
2	10	3
3	−1	−1
4	12	−5
5	−1	−1
...	...	...

- Q1. Donner le stockage correspondant au polynôme de $2x^5 - 3x^{34} + 4x^{105}$
- Q2. Quel est le problème avec $8 - 5x^2 + 3x^{10}$?
- Q3. En cas de collision, on décide d'utiliser la première place libre suivante (sondage linéaire). Les monômes sont entrés suivant l'ordre de lecture. Donner un polynôme de degré minimum à 5 monômes, donné par degrés croissants de la gauche vers la droite) qui engendre une collision à chaque insertion sauf pour la première. Parmi les nombreuses réponses possibles, choisir celle dont la somme des degrés est minimale.
- Q3.bis Même question si l'ordre des degrés est arbitraire.
- Q4. Pour stocker les polynômes, on envisage maintenant une stratégie à deux tableaux. On stocke, dans le 1er tableau, les tuples (degré,coeff) dans l'ordre où on les trouve. Et on complète le tableau avec des -1. Dans le second tableau, à l'indice `degré%8`, on stocke l'indice où on a mis le tuple (degré,coeff) dans le premier tableau. Et on complète le tableau avec des -1. Donner les deux tableaux correspondant au stockage du polynôme $3x^5 - x^{18} + 7x^{20}$.
- Q5. On suppose stockés les tableaux de la question Q4. Pour connaître le coefficient du monôme de degré i, l'utilisateur ne peut interroger que le second tableau. Indiquer comment, grâce au hachage, il peut déterminer le coefficient du monôme de degré 18.

****

- Q1.

indice	degré	coeff
0	−1	−1
1	105	8
2	34	−3
3	−1	−1
4	−1	−1
5	5	2
...	...	...

[16]:
```
#*********
34%8, 105%8
```

[16]: (2, 1)

- Q2.

2 et 10 ont le même modulo 8 : collision

[18]:
```
#************
10%8
```

[18]: 2

********** - Q3.

$1 + x^8 + x^9 + x^{10} + x^{11}$

puis

$1 + x^8 + x^1 + x^2 + x^3$

********** - Q4.

$3x^5 - x^{18} + 7x^{20}$

1er tableau

indice	degré	coeff
0	5	3
1	18	−1
2	20	7
3	−1	−1
4	−1	−1
5	−1	−
...	...	...

2nd tableau

indice	pos. dans 1er tableau
0	−1
1	−1
2	1
3	−1
4	2
5	0
...	...

- Q5.Pour savoir le coeff de x^{18}, on calcule 18%8 soit 2. On trouve dans le second tableau que 2 correspond à la position 1. Dans le 1er tableau, on trouve le coeff de x^{18}, soit -1

[5]:
```
#**********
18%8,20%8
```

[5]: (2, 4)

5.2 Implémentation

Dans cet exercice, on s'intéresse au tableau sous-jacent d'un dictionnaire. On le suppose de taille 8 (il contient donc au plus 5 éléments avant redimensionnement). On veut simuler le stockage des tuples (clé,valeur) dans le tableau sous-jacent. Comme fil conducteur, on considère le dictionnaire théorique $d = \{"i" \mapsto 19, "n" \mapsto 14, "f" \mapsto 6, "o" \mapsto 15\}$. Le hachage utilisé est la fonction `hash` de Python.

- Q1. Ecrire la fonction `dic(couples)` qui prend en paramètre une liste comme `L=[("i",19), ("n",14), ("f",6),("o",15)]`. Elle insère les tuples dans un dictionnaire dont elle retourne le tableau sous-jacent. Les cases vides du tableau sous-jacent sont initialisées à `None`. On ne demande pas de gérer les collisions (si deux clés ont la même valeur de hachage, la plus récente prend la place de la plus ancienne). Appliquer la fonction à la liste `L`. Constater la présence d'une collision.

[1]:
```
#*****
def dic(couples):
    liste = 8*[None]
    for c,v in couples:
        liste[hash(c)%8] = c,v
    return liste

L=[("i",19), ("n",14), ("f",6),("o",15)]
dic(L)
```

[1]:
```
[('o', 15), None, None, ('f', 6), None, ('i', 19), None, None]
```

- Q2. Ecrire la fonction `dic2(couples)` qui gère les collisions par adressage ouvert. La fonction de sondage est ici linéaire. On s'inspire de celle de Python, qui, en cas de collision pour de petits indices (inférieurs à 32) à la case i, se positionne à la case $(5 \times i + 1)\%8$. Appliquer à la liste `L`. Constater que le problème de collision a été réglé.

[1]:
```
#**********
def dic2(couples):
    liste = 8*[None]
    for c,v in couples:
        i=hash(c)%8
        print("c={},d={},i={}".format(c,v,i))
        while liste[i] is not None:
            i=(5*i+1)%8
        print("  i choisi:{}".format(i))
        liste[i] = c,v
    return liste

L=[("i",19), ("n",14), ("f",6),("o",15)]
dic2(L)
```

```
c=i,d=19,i=2
  i choisi:2
c=n,d=14,i=7
```

```
  i choisi:7
c=f,d=6,i=2
  i choisi:3
c=o,d=15,i=6
  i choisi:6
```

[1]:
```
[None, None, ('i', 19), ('f', 6), None, None, ('o', 15), ('n', 14)]
```

6 KWARGS

Importer le sous-module `pyplot` de `matplotlib` avec le préfixe `plt`. Puis examiner le manuel de la fonction `plt.plot`. Que signifie le mystérieux `kwargs` à la fin du prototype de la fonction ?

[3]:
```
#********
import matplotlib.pyplot as plt
```

[5]:
```
#***********
help(plt.plot)
```