Informatique et Programmation

Cours 5

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-py

Plan

- gestion mémoire et alias
- dictionnaires Python
- recherche en table
- recherche en table par dichotomie
- hachage
- hachage ouvert
- correcteur de français

dès maintenant: télécharger Python 3 en `http://www.python.org`

Tableaux dynamiques — listes

- on peut ajouter, étendre, insérer ou détruire un élément dans une liste

Rappel: les listes Python sont des tableaux dynamiques

```
>>> a = [3, 5, 7, 9]
>>> a.append(11)
>>> a
[3, 5, 7, 9, 11]
>>> a.extend([13, 17, 19])
>>> a
[3, 5, 7, 9, 11, 13, 17, 19]
>>> a.insert (6, 15)
>>> a
[3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> a.insert (1, 5)
>>> a
[3, 5, 5, 7, 9, 11, 13, 15, 17, 19]
>>> del a[1]
>>> a
[3, 5, 7, 9, 11, 13, 15, 17, 19]
```

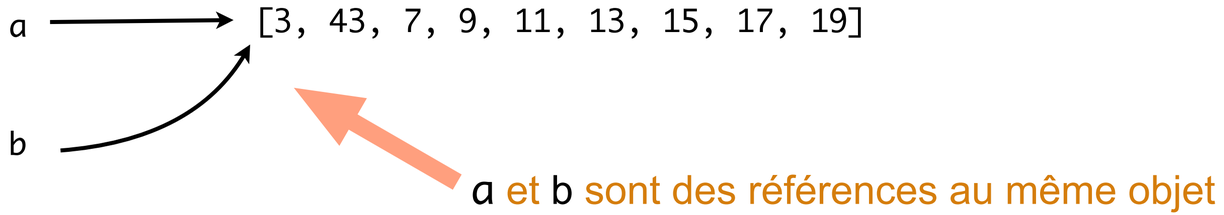
application de la méthode `append` à `a` (on verra plus tard la notation objet)

mot-clé `del` pour suppression (delete) d'un objet

Tableaux dynamiques — listes

- une affectation à une variable ne copie pas le tableau

```
>>> b = a
>>> id(a) == id(b)
True
>>> b[1] = 43
>>> a
[3, 43, 7, 9, 11, 13, 15, 17, 19]
```



- on dit alors que a et b sont des alias

- quand on modifie le tableau b, on modifie aussi le tableau a

DANGER ! DANGER !

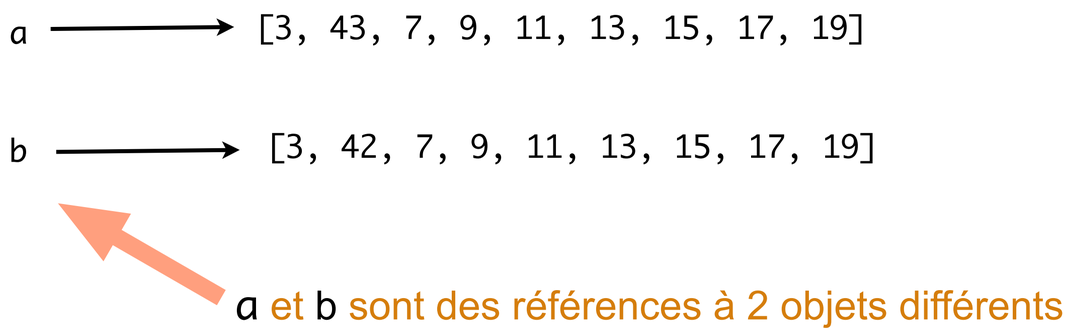
- alias et données modifiables ne font pas bon ménage !!

- alias et données **non modifiables** sont à la base de la **programmation fonctionnelle** (Ocaml, Haskell, ..)

Tableaux dynamiques — listes

- le module `copy` permet de faire une copie de `a`

```
>>> import copy
>>> a
[3, 43, 7, 9, 11, 13, 15, 17, 19]
>>> b = copy.copy(a)
>>> b
[3, 43, 7, 9, 11, 13, 15, 17, 19]
>>> b[1] = 42
>>> b
[3, 42, 7, 9, 11, 13, 15, 17, 19]
>>> a
[3, 43, 7, 9, 11, 13, 15, 17, 19]
```



Dictionnaires

- les dictionnaires de Python sont des listes non ordonnées d'association (clé, valeur)

```
>>> age = {'JJ':35, "marie": 25, "alice": 38, "bob": 29}
>>> print (age)
{'bob': 29, 'marie': 25, 'alice': 38, 'JJ': 35}
>>> age['marie']
25
>>> age['JJ']
35
>>> age['raymond'] = 29
>>> age
{'bob': 29, 'marie': 25, 'alice': 38, 'JJ': 35, 'raymond': 29}
>>> age ['JJ'] = 49
>>> age
{'bob': 29, 'marie': 25, 'alice': 38, 'JJ': 49, 'raymond': 29}
```

```
>>> age.keys()
['bob', 'marie', 'alice', 'JJ', 'raymond']
>>> age.values()
[29, 25, 38, 49, 29]
```

```
>>> age['henry']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'henry'
>>> 'henry' in age
False
>>> 'alice' in age
True
```

Dictionnaires

- les clés (et les valeurs) peuvent être des expressions quelconques

```
>>> adresse = {'JJ': (9, 'heredia', 75007, 'Paris'),
               'marie': (3, 'kleber', 75018, 'Paris'),
               'bob': (14, 'broadway', 3001, 'New York')}
>>> adresse['bob']
(14, 'broadway', 3001, 'New York')
>>> adresse['JJ']
(9, 'heredia', 75007, 'Paris')
```

les clés sont des chaînes de caractères

création avec notation compréhensive

```
>>> squares = {x: x*x for x in range(7)}
>>> squares
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
>>> squares[4]
16
>>> squares[8] = 63
>>> squares
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 8: 63}
>>> squares [7]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 7
>>> squares[8] = 64
>>> squares[7] = 49
```

les clés sont des entiers

Recherche en table

- les dictionnaires permettent d'accéder à une valeur par une clé
- le mécanisme de base est la recherche d'une clé en table
- exemple: recherche d'un numéro de téléphone dans un annuaire

```
def recherche (x, a) :
    for p in a :
        if p[0] == x :
            return p[1]
    return 'None'
```

a

('jj', '0543221800')
('marie', '0728423301')
('raymond', '0924828202')
('bob', '0630204100')
('alice', '0629359210')
('ali', '0330013917')

- il vaut mieux retourner une exception quand la clé n'est pas trouvée

```
def recherche (x, a) :
    for p in a :
        if p[0] == x :
            return p[1]
    raise ValueError
```

type du résultat plus cohérent avec le type des valeurs trouvées

- on verra plus tard comment retourner une exception plus précise

Recherche en table

- la recherche précédente fait n opérations si n est la longeur de a
- si le tableau est trié dans l'ordre croissant des clés, on ramène ce nombre d'opératons à log(n)

```
def recherche_dichotomique (x, a) :
    n = len(a)
    g = 0; d = n
    while g < d :
      i = (g + d) // 2
      p = a[i]
      if x == p[0] :
          return p[1]
      elif x < p[0] :
          d = i
      else :
          g = i + 1
    raise ValueError
```

a

('ali', '0330013917')
('alice', '0629359210')
('bob', '0630204100')
('jj', '0543221800')
('marie', '0728423301')
('raymond', 0924828202')

n = 230000 donne log (230000) ~ 18 opérations

Lecture de fichier

- pour lire un fichier, on l'ouvre en mode lecture seule ‘r’ (read)

  et on le lit en séparant les lignes

```
def lire_lignes (nom) :
    f = open (nom, 'r')
    return f.read().splitlines()
```

- dans le répertoire courant, il y a un dictionnaire de français `french-iso de 239210 mots`

```
>>> a = lire_lignes ('french-iso')
>>> len(a)
239210

>>> for i in range (10) :
...     print (a[i])
...
a
abaca
abacule
abaissa
abaissable
abaissai
abaissaient
abaissais
abaissait
abaissant
```

Recherche d’un mot français

- on applique les méthodes de la recherche en table

```
def recherche (x, a) :
    for p in a :
        if p == x :
            return True
    return False
```

239210 **opérations**

```
def recherche_dichotomique (x, a) :
    n = len(a)
    g = 0; d = n
    while g < d :
      i = (g + d) // 2
      p = a[i]
      if x == p :
          return True
      elif x < p :
          d = i
      else :
          g = i + 1
    return False
```

17.9 **opérations**

```
>>> tri_Fusion (a)
>>> recherche_dichotomique ('jouer', a)
True
>>> recherche_dichotomique ('jouera', a)
True
>>> recherche_dichotomique ('jouerra', a)
False
```

Recherche par interpolation

- on a une connaissance de la distribution des clés (par exemple uniforme)
- au lieu de comparer à la clé du milieu, on peut comparer à une clé plus proche de la distribution

```
def recherche_interpolation (x, a) :
    g = 0; d = len(a); lo = a[g]; hi = a[d-1]
    while g < d :
        if not lo <= x <= hi :
            return False
        if hi == lo:
           i = g
        else :
           i = (g * (hi - x) + (d-1) * (x - lo) ) // (hi - lo)
        p = a[i]
        if x == p :
            return True
        elif x < p :
            d = i
        else :
            g = i + 1
    return False
```

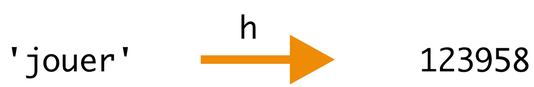
- pour une distribution uniforme, cela donne de l'ordre de `log(log(n))` opérations

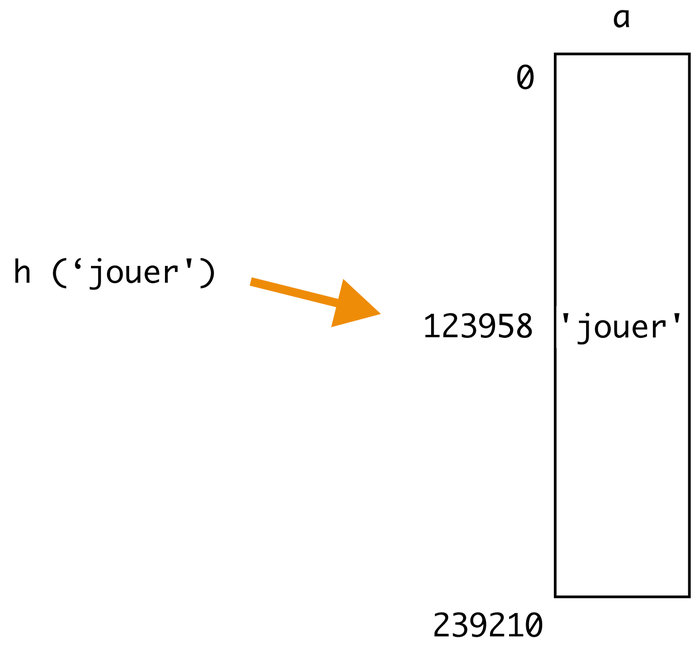
`4.16` **opérations pour table de** `239210` **entrées !!**

Exercice écrire la recherche par interpolation pour le dictionnaire de français

Hachage

• une fonction de hachage donne l'entrée dans la table en fonction de la clé





• comment réaliser une telle fonction ?

• 2 types de fonctions de hachage:

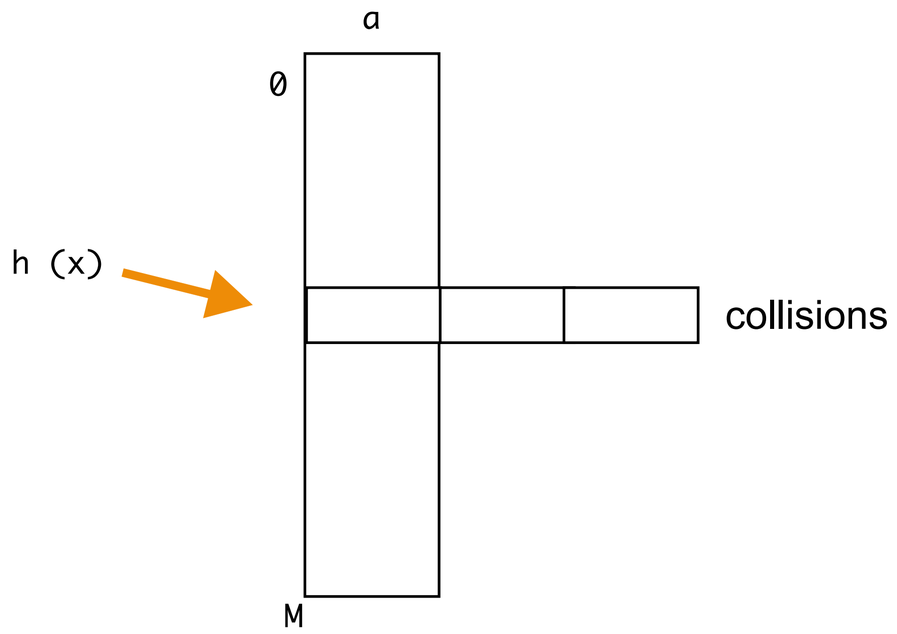
- pour la cryptographie: md5, sha1, sha-512 (on veut distinguer les valeurs avec de longues empreintes)
- pour les tables de hachage (*hash tables*) (on veut rendre uniforme les valeurs de hachage dans un petit ensemble)

Hachage

- on réorganise la table en créant des classes d'équivalences (collisions)

```
def hash (x, l, M) :
    s = 0
    for i in range (l) :
        c = ord (x[i]) if i < len (x) else 0
        s = (256 * s + c) % M
    return s


def recherche (x, a) :
     i = hash (x, 4, 61)
     for p in a[i] :
         if p == x :
             return True
     return False
```



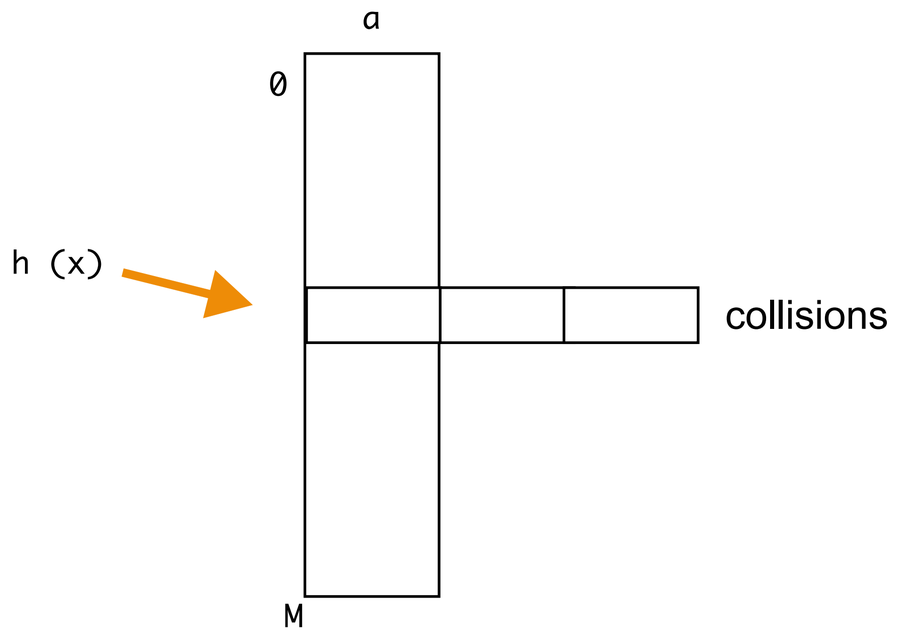
- on doit organiser la table autour de la fonction de hachage !!

Hachage

- on réorganise la table en créant des classes d'équivalences (collisions)

```
def h (x, l, M) :
    s = 0
    for i in range (l) :
        c = ord (x[i]) if i < len (x) else 0
        s = (256 * s + c) % M
    return s


def recherche (x, a) :
    i = h (x, 4, 61)
    for p in a[i] :
        if p == x :
            return True
    return False
```

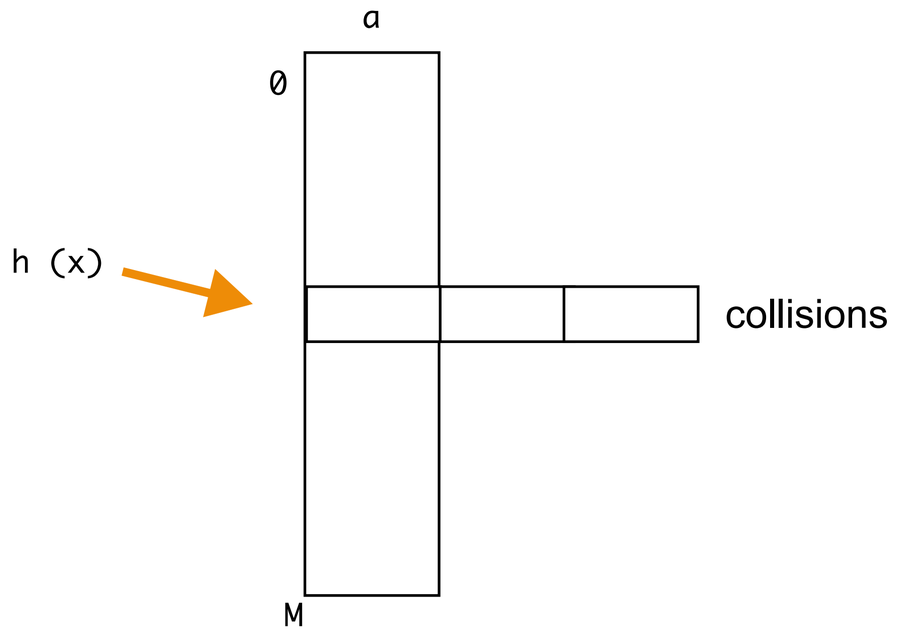


- plus la fonction de hachage est uniforme, moins la recherche fera d'opérations

- on doit aussi avoir organisé la table autour de la fonction de hachage !!

Hachage

Exercice insérer un nouvel élément dans une table de hachage



Exercice imaginer une structure de données pour le hachage sans réorganiser la table

Correcteur de français

- on peut rechercher tous les mots d'un texte en français dans le dictionnaire de français
  et on peut les mots qui n'y sont pas

```
def lire_mots_fichier (nom) :
    f = open (nom, 'r')
    return f.read().split()
```

Exercice écrire un vérificateur d'orthographe

- pour un correcteur d'orthographe, on peut essayer de changer une lettre,
  ou de l'insérer, ou de la supprimer, ou de changer l'ordre de 2 lettres consécutives

Exercice écrire un correcteur d'orthographe

- en 1980, la taille de la mémoire des ordinateurs était restreinte. McIlroy a fait tenir un dictionnaire de 300000 mots
  (programme `spell`) en 30k octets en ne stockant que les bits de 10 fonctions de hachage pour chaque mot
  du dictionnaire