Structures de données

Thibaut Cantaluppi

September 15, 2024

Mutabilité

Un objet est dit **mutable** si l'on peut changer sa valeur après sa création sans nouvelle affectation. Il est dit **immutable** dans le cas contraire.

Mutabilité

Un objet est dit **mutable** si l'on peut changer sa valeur après sa création sans nouvelle affectation. Il est dit **immutable** dans le cas contraire.

Objets immutables:

- entiers
- flottants
- booléens
- chaines de caractères
- tuples

Mutabilité

Un objet est dit **mutable** si l'on peut changer sa valeur après sa création sans nouvelle affectation. Il est dit **immutable** dans le cas contraire.

Objets immutables:

- entiers
- flottants
- booléens
- chaines de caractères
- tuples

Objets mutables:

- listes
- dictionnaires
- sets

Les objets muables sont passés par valeur

Dictionnaire

Un **dictionnaire** est une structure de données qui à chaque **clé** associe une **valeur**. Il possède les opérations suivantes :

- Ajouter une association (clé, valeur).
- Supprimer une association (clé, valeur).
- Obtenir les valeurs associées à une clé donnée.

Dictionnaire

Un **dictionnaire** est une structure de données qui à chaque **clé** associe une **valeur**. Il possède les opérations suivantes :

- Ajouter une association (clé, valeur).
- Supprimer une association (clé, valeur).
- Obtenir les valeurs associées à une clé donnée.

Exemples :

- Associer à chaque utilisateur (clé) son mot de passe (valeur).
- Associer à chaque couleur sous forme de texte (clé) son code RGB (valeur).

Dictionnaire : Définition

Définition d'un dictionnaire en Python, de type `dict` :

```
d = {} # dictionnaire vide
d = dict() # autre façon de définir un dictionnaire vide

# dictionnaire avec 2 associations
mots_de_passe = { "jean-michel" : "azerty", "admin": "1234" }

# dictionnaire avec 3 associations
couleurs = {
    "rouge" : (255, 0, 0),
    "jaune" : (255, 255, 0),
    "blanc" : (255, 255, 255)
}
```

Dictionnaire : Définition

Définition d'un dictionnaire en Python, de type `dict` :

```
d = {} # dictionnaire vide
d = dict() # autre façon de définir un dictionnaire vide

# dictionnaire avec 2 associations
mots_de_passe = { "jean-michel" : "azerty", "admin": "1234" }

# dictionnaire avec 3 associations
couleurs = {
    "rouge" : (255, 0, 0),
    "jaune" : (255, 255, 0),
    "blanc" : (255, 255, 255)
}
```

Dans `mots_de_passe`, les clés et les valeurs sont des chaînes de caractères.
Dans `couleurs`, les clés sont des chaînes de caractères et les valeurs sont des triplets d'entiers.

Dictionnaire : Accès

`d[k]` donne la valeur associée à la clé `k` dans le dictionnaire `d` :

```
mots_de_passe["jean-michel"] # renvoie "azerty"

couleurs["jaune"] # renvoie (255, 255, 0)

couleurs["bleu"] # erreur : la clé n'existe pas
```

Dictionnaire : Modification

On modifie la valeur associée à la clé `k` avec `d[k] = ...`.
Si la clé n'existait pas, elle est ajoutée.

```
mots_de_passe["admin"] = "zpdOQ64n"
# change la valeur associée à "admin"

couleurs["bleu"] = (0, 0, 255)
# ajoute la clé "bleu" avec la valeur (0, 0, 255)
```

Dictionnaire : Test d'appartenance

On peut tester si une clé `k` appartient à un dictionnaire `d` avec `k in d` (ou n'appartient pas, avec `k not in d`) :

Dictionnaire : Test d'appartenance

On peut tester si une clé `k` appartient à un dictionnaire `d` avec `k in d` (ou n'appartient pas, avec `k not in d`) :

```
"jean-michel" in mots_de_passe # renvoie True
"violet" in couleurs # renvoie False
```

Utile pour éviter une erreur en accédant à une clé qui n'existe pas.

Exemple :

```
def login(nom, mdp):
    if nom in mots_de_passe:
        if mdp == mots_de_passe[nom]:
            return True
    return False
```

Dictionnaire : Parcours d'un dictionnaire

On peut obtenir les clés d'un dictionnaire avec `d.keys()` et ses valeurs avec `d.values()` :

```
mots_de_passe.keys() # renvoie ["jean-michel", "admin"]

mots_de_passe.values() # renvoie ["azerty", "1234"]
```

On peut aussi connaître le nombre de clés d'un dictionnaire avec `len(d)`.

Dictionnaire : Parcours d'un dictionnaire

On peut parcourir toutes les clés (et donc aussi toutes les valeurs) :

```
for k in mots_de_passe:
    print(k + " a le mot de passe " + mots_de_passe[k])
```

Dictionnaire : Complexité

Python	**Description**	**Complexité**
`d[k] = v`	Ajout (ou modification) d'une association de `k` à `v`	O(1) en moyenne
`d[k]`	Accès à la valeur de clé `k`	O(1) en moyenne
`len(d)`	Nombre de clés de `d`	O(1) en moyenne
`for k in d:`	Parcourir les clés `k` de `d`	O(n) où n est le nombre de clés

Dictionnaire : Exercices

Exercice

Écrire une fonction `inverse(d)` renvoyant un dictionnaire `d2` tel que :

$$\texttt{d[k]} = \texttt{v} \iff \texttt{d2[v]} = \texttt{k}$$

où on suppose que `d` ne contient pas deux fois la même valeur `v`.

Dictionnaire : Exercices

Exercice

Écrire une fonction `inverse(d)` renvoyant un dictionnaire `d2` tel que :

$$\texttt{d[k]} = \texttt{v} \iff \texttt{d2[v]} = \texttt{k}$$

où on suppose que `d` ne contient pas deux fois la même valeur `v`.

Exercice

Écrire une fonction `frequent(L)` renvoyant l'élément le plus fréquent dans une liste `L` de taille n, en complexité $O(n)$.

Applications : Dictionnaire de variables

Python conserve en mémoire deux dictionnaires :

- `globals` : fonction qui renvoie un dictionnaire contenant les **variables globales**.
- `locals` : idem mais pour les **variables locales** (accessibles uniquement dans un bloc de code, par exemple dans une fonction).

```
def f(x):
    y = 2
    print(locals())
f(1) # affiche {'x': 1, 'y': 2}
```

Applications : Représentation de graphe/arbre

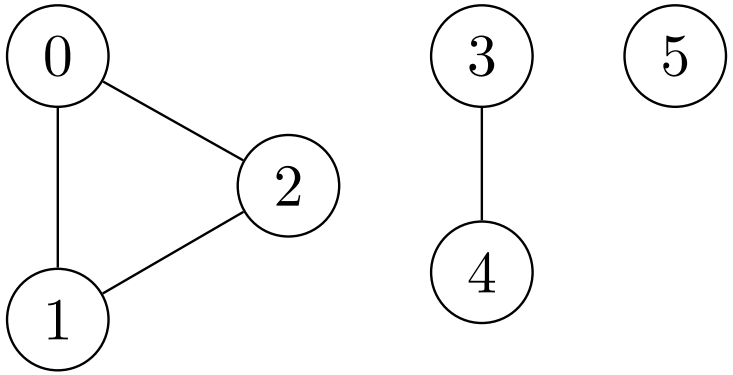
Au lieu de la représentation par matrice/liste d'un graphe G, on peut utiliser une représentation par un dictionnaire `d`, où `d[v]` est la liste (ou l'ensemble) des voisins du sommet `v`.

Applications : Représentation de graphe/arbre

Au lieu de la représentation par matrice/liste d'un graphe G, on peut utiliser une représentation par un dictionnaire `d`, où `d[v]` est la liste (ou l'ensemble) des voisins du sommet `v`.

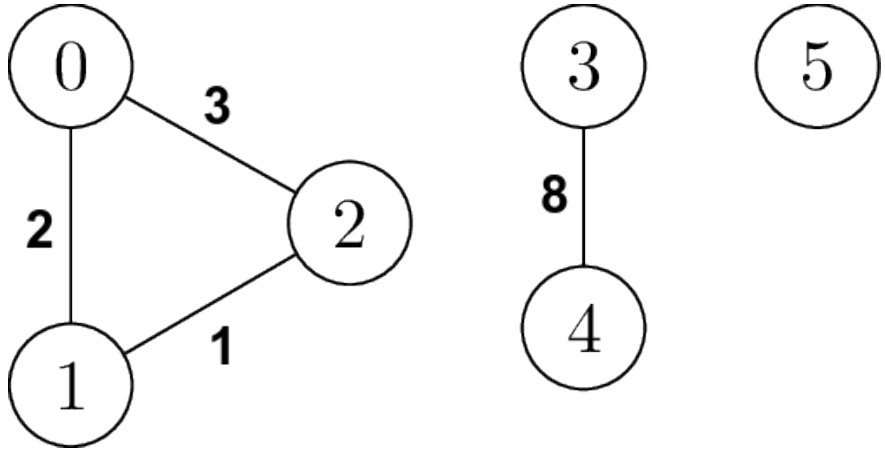
Exercice

Représenter le graphe ci-dessous avec un dictionnaire.



Exercice

Représenter le graphe pondéré ci-dessous avec un dictionnaire.



Implémentation d'un dictionnaire par table de hachage

Il existe (au moins) deux façons de créer un dictionnaire :

- Avec **table de hachage** (utilisé par Python).
- (Pour MP/MP* option info seulement) Avec un **arbre binaire de recherche** : chaque noeud est un couple (clé, valeur) et on compare les clés seulement.

Implémentation d'un dictionnaire par table de hachage

Définition

Une **table de hachage** est composée de :

- Un **tableau** contenant les valeurs.
- Une **fonction de hachage** h telle que, si k est une clé, $h(k)$ est l'indice du tableau où se trouve la valeur associée à k.

Implémentation d'un dictionnaire par table de hachage

Définition

Une **table de hachage** est composée de :

- Un **tableau** contenant les valeurs.
- Une **fonction de hachage** h telle que, si k est une clé, $h(k)$ est l'indice du tableau où se trouve la valeur associée à k.

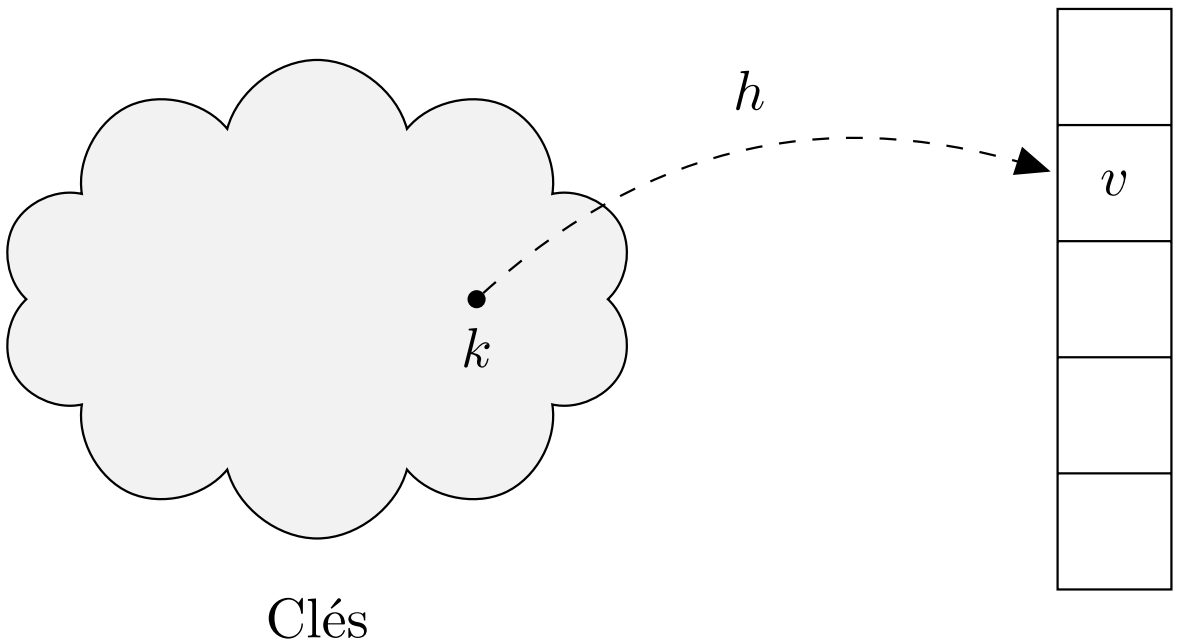
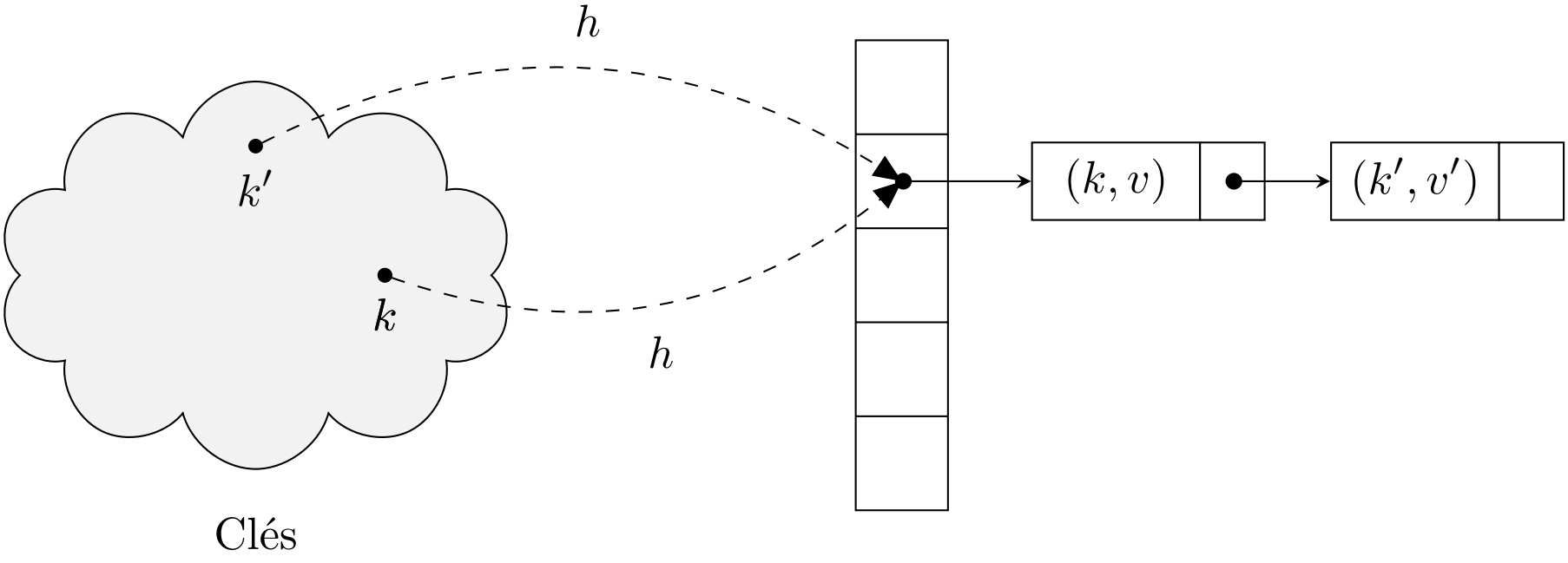


Table de hachage $\approx$ tableau dont les indices (clés) ne sont pas forcément des entiers consécutifs.

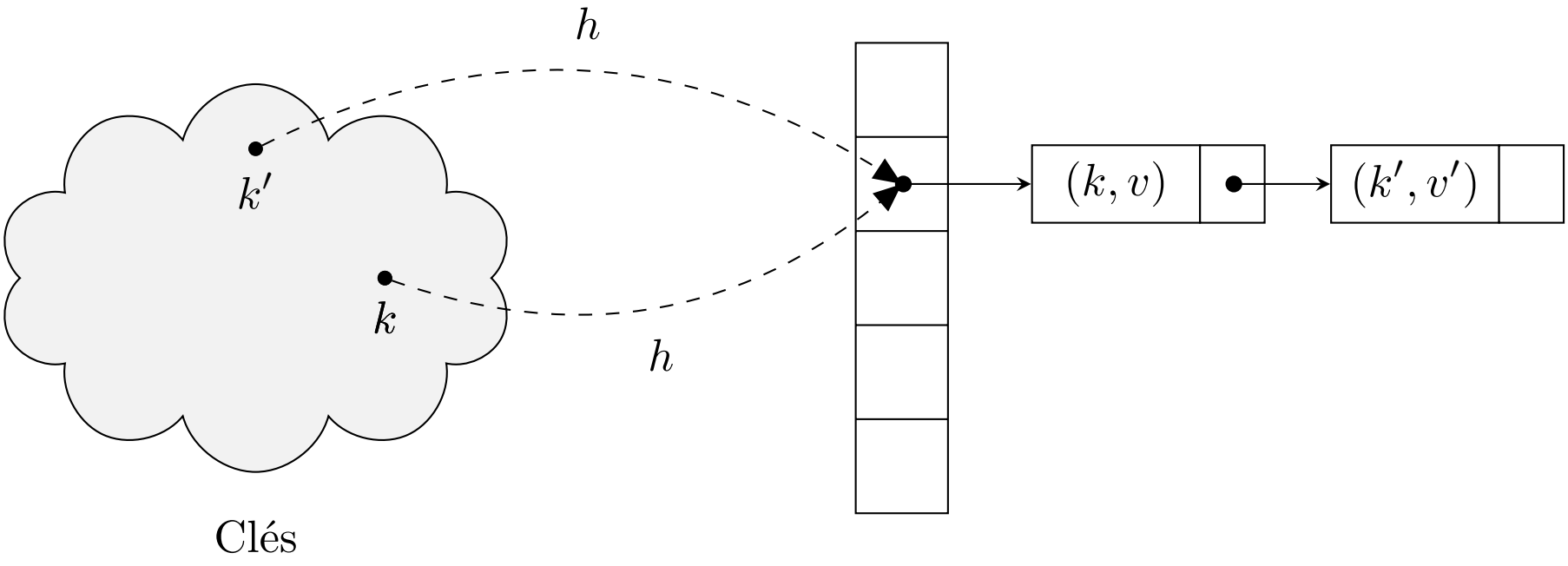
Implémentation d'un dictionnaire par table de hachage

Souvent, il y a beaucoup plus de clés possibles que de cases du tableau, ce qui conduit à des **collisions** : plusieurs clés ayant la même image par h. On peut résoudre ces collisions par **chaînage**, en stockant une liste à chaque position de la table de hachage :



Implémentation d'un dictionnaire par table de hachage

Souvent, il y a beaucoup plus de clés possibles que de cases du tableau, ce qui conduit à des **collisions** : plusieurs clés ayant la même image par h. On peut résoudre ces collisions par **chaînage**, en stockant une liste à chaque position de la table de hachage :



Autre possibilité de résolution de collisions : **adressage ouvert**.

Implémentation d'un dictionnaire par table de hachage

Les dictionnaires en Python sont des tables de hachages qui utilisent une fonction `hash`.

`hash(e)` n'est définit que si `e` est immutable (ou : persistant), c'est-à-dire non modifiable. Il est en effet fortement déconseillé d'utiliser une clé qui puisse être modifiée (ex : liste) puisque cela changerait l'image par la fonction de hachage.

Implémentation d'un dictionnaire par table de hachage

Implémentation d'un dictionnaire par table de hachage, avec résolution par chaînage :

```
T = [[] for i in range(10)] # liste contenant 10 listes vides

def h(k):
    return (k**2) % 10 # exemple de fonction de hachage

def add(k, v): # ajoute l'association de clé k et de valeur v
    T[h(k)].append((k, v))

def get(k): # donne la valeur associée à la clé k
    for k_, v in T[h(k)]:
        if k == k_:
            return v
    raise Exception("clé non trouvée")
```

`T[i]` contient la liste des associations (k, v) telles que $h(k) =$ `i`.

Ensemble (HP)

Un ensemble (`set` en Python) est l'analogue d'un ensemble mathématique :

```
s = {2, 3} # définition d'un ensemble contenant 2 et 3
s.add(5) # ajout de 5
3 in s # renvoie True
for e in s: # affiche tous les éléments de s
    print(e)
```

Ensemble (HP)

Un ensemble (`set` en Python) est l'analogue d'un ensemble mathématique :

```
s = {2, 3} # définition d'un ensemble contenant 2 et 3
s.add(5) # ajout de 5
3 in s # renvoie True
for e in s: # affiche tous les éléments de s
    print(e)
```

`set` est implémenté par table de hachage, comme `dict` $\longrightarrow$ add et `in` sont en $O(1)$.

Ensemble (HP)

Exercice

Écrire une fonction `eratosthene(n)` renvoyant l'ensemble des nombres premiers inférieurs à `n`, en utilisant le crible d'Eratosthène.