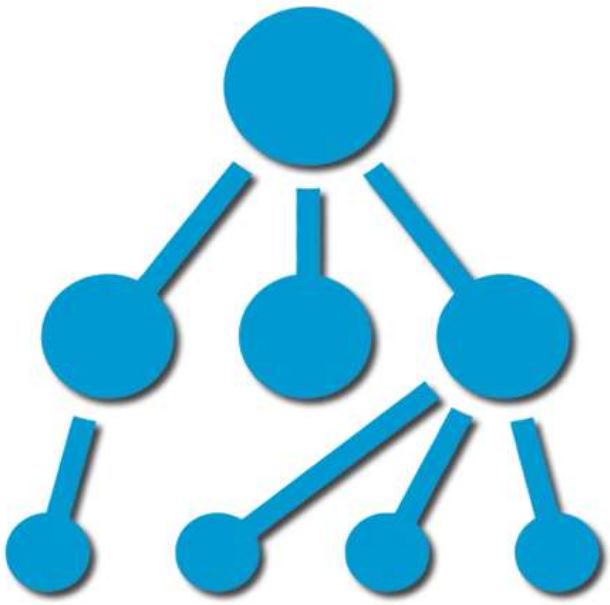
STRUCTURES DE DONNÉES

NSI 02



LES DICTIONNAIRES

[Frédéric Peurière – Marion Szpieg]

1. Quelques rappels

1.1. Qu'est-ce que c'est ?

Un **dictionnaire** est une structure de données dans laquelle un ensemble de clés est associé à un ensemble de valeurs, chaque clé étant associée à une seule valeur.

On peut assimiler un dictionnaire à un ensemble de couples (cle, valeur), les clés étant le plus souvent des nombres ou des chaînes de caractères. On peut faire le parallèle avec un dictionnaire classique où les mots seraient les différentes clés et les définitions la valeur de chaque clé.

Pour retrouver une valeur dans un dictionnaire, il faut connaître sa clé : il n'y a donc pas de notion d'ordre (contrairement aux tableaux où l'ordre est donné par l'indice de chaque élément du tableau).

1.2. Les dictionnaires sous Python

En Python, la structure de donnée « dictionnaire » existe : « dict ». Voici quelques rappels de première :

Commande	Fonction
`dico = {}`	crée un dictionnaire vide
`dico[c] = v`	ajoute le couple (c, v) si la clé « `c` » n'est pas présente dans le dictionnaire. Dans le cas contraire, remplacer la valeur de la clé « `c` » par « `v` »
`c in dico`	teste si `c` est une des clés du dictionnaire
`dico[c]`	renvoie la valeur associée à clé « `c` » si elle existe, une erreur sinon
`for c in dico.keys() :`	parcourt sur les clés
`for v in dico.values():`	parcourt sur les valeurs
`for c, v in dico.items():`	parcourt sur les couples (clé, valeur)

2. Interface d'un dictionnaire

Voici l'interface minimale d'un dictionnaire (à connaître !!) :

- `creer_d()` : retourne un dictionnaire vide
- `est_vide(D)` : retourne vrai si le dictionnaire `D` est vide et faux sinon
- `ajouter(D, cle, valeur)` : ajoute le couple (`cle`, `valeur`) au dictionnaire `D`, puis retourne le nouveau dictionnaire. Si la clé « `cle` » existe déjà, actualise la valeur.

On peut lui ajouter les fonctions suivantes souvent utilisées :

- `chercher_cle(D, cle)` : retourne vrai si `cle` existe dans les clés du dictionnaire `D` et faux sinon
- `chercher_valeur(D, cle)` : retourne la valeur associée à `cle` si le couple (`cle`, `valeur`) existe dans le dictionnaire `D`.
- `supprimer(D, cle)` : supprime le couple (`cle`, `valeur`), puis retourne le nouveau dictionnaire

3. Un exemple d'implémentation d'un dictionnaire : avec une table de hachage.

Le principe : on va construire un dictionnaire à l'aide :

- d'un tableau de taille fixe prédéfinie
- d'une fonction de hachage

Regarder le vidéo suivante : https://www.youtube.com/watch?v=CkLctGYWFPA

Application/exercice : on va implémenter un dictionnaire avec le fonctionnement de la table de hachage décrit dans la vidéo. Toutes les fonctions ci-dessous sont à créer dans un seul et même fichier :

1. Nombre premier choisi pour tout le monde : **53** (à stoker dans une variable globale `T`)

2. a. Créer une fonction de hachage comme décrite dans la vidéo de 9'15'' à 10'00'' : on fait la somme des valeurs des codes ASCII de chaque caractère de la clé que l'on modère par la place dans le mot (donc la première lettre est à la place n°1!). Puis on garde le reste de la division euclidienne par `T`.
   b. Calculer les empreintes des mots suivants : 'Youssef', 'Mohamed', 'Francisco', 'Lucas', 'Rafael', 'Frédéric' et 'Marion' (en respectant bien les majuscules et les accents !)
   c. Quelle remarque peut-on faire ?

Rappel : pour accéder à l'encodage ASCII d'un caractère : `ord(caractère)`*, ou caractère est un caractère de type* `String`*.*

3. Créer la fonction `creer_d()`.

4. Créer la fonction `est_vide(D)`.

5. a. Créer la fonction `ajouter(D, cle, valeur)` qui fonctionne comme expliqué dans la vidéo entre 1'45'' et 4'40''. Le couple (`cle` , `valeur`) sera stocké à l'indice `hachage(cle)`. Attention : il faudra prendre en compte le cas de la où la clé existe déjà ainsi que le cas de la collision.

   b. Tester que le cas où la clé existe déjà est bien géré avec les instructions suivantes :

```
dic=creer_d()
dic=ajouter(dic,'Francisco','lfclt33')
print(dic)
dic=ajouter(dic,'Francisco','lfclt3')
print(dic)
```

   c.Tester que le cas où les collisions sont bien gérées avec les instructions suivantes :

```
dic=creer_d()
dic=ajouter(dic,'Lucas','lfclt5')
print(dic)
dic=ajouter(dic,'Rafael','lfclt2')
print(dic)
dic=ajouter(dic,'Frédéric','lfclt5')
print(dic)
```

6. Créer la fonction `chercher_cle(D,cle)`. La tester

7. Créer la fonction `chercher_valeur(D,cle)`. (qui pourra se servir de la fonction précédente!!) La tester

```python
T=53

def hachage(mot):
    emp=0
    cpt=1
    for i in mot:
        emp=emp+ord(i)*cpt
        cpt+=1
    return emp%T

def creer_d():
    return [None]*T

def est_vide(D):
    return D==[None]*T

def ajouter(D,cle,valeur):
    i=hachage(cle)
    if D[i]==None:
        D[i]=(cle,valeur)
    else:
        while D[i]!=None:
            if cle==D[i][0]:
                D[i]=(cle,valeur)
                return D
            else:
                if i==T-1:
                    i=0
                else:
                    i+=1
        D[i]=(cle,valeur)
    return D

def chercher_cle(D,cle):
    i=hachage(cle)
    while D[i]!=None:
        if D[i][0]==cle:
            return True
        else:
            i+=1
        if i==T:
            i=0
    return False
```