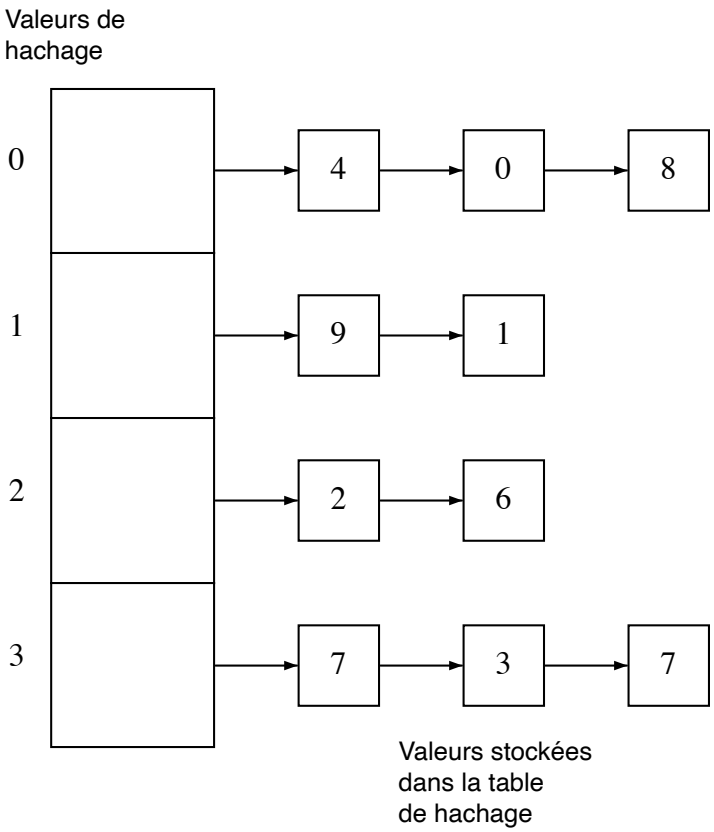
Soutien en algorithmique et programmation

Séance 8 : table de hachage

Introduction

On va travailler sur une structure de données classique : une table de hachage. Une table de hachage peut-être implantée de façon très simple sous la forme d'un tableau de listes chainées. On doit également fournir une fonction de hachage dont le rôle est de déterminer dans quelle liste une valeur se trouve ou doit être insérée. Elle prend en argument la valeur et renvoie l'indice de la case du tableau contenant la liste appropriée. La fonction de hachage dans cet exercice calculera simplement le modulo de la valeur par la taille du tableau.

Le schéma ci-dessous illustre une table de hachage classique, pour laquelle la fonction de hachage sera simplement `valeur % taille_tableau` (`%` est l'opérateur modulo en Python) :



Implantation des listes

Commencer par implanter une classe `Cellule` avec comme champs une valeur et un suivant. On pourra reprendre la classe `Cellule` utilisée les séances précédentes.

Implanter ensuite une classe `Liste` : les listes seront simplement chainées, avec un élément fictif en tête et un compteur d'éléments. Le constructeur `__init__` de la classe `Liste` devra donc initialiser deux champs : un champ `tete` qui pointe sur la cellule fictive en tête qu'on allouera directement dans le constructeur, et

un champ `nbr_elem` qui vaudra zéro initialement (la cellule fictive ne compte pas dans le nombre d'éléments de la liste). On pourra aussi implanter un afficheur `__str__` pour aider à mettre au point le code.

Implantation de la table

On va implanter la table de hachage sous la forme d'un tableau de 2 cases contenant respectivement le nombre d'éléments dans la table et le tableau de listes. On définira deux constantes pour accéder à ces cases proprement :

```
IX_NBR = 0
IX_TAB = 1
```

On pourra tester les fonctions écrites grace au programme principal suivant, à copier-coller dans un fichier `tablehash.py`. On fournit aussi une fonction de création et une fonction d'affichage de la table, en plus de la fonction de hachage nécessaire.

```
def hachage(table, val):
    """
      Methode de hachage tres naive
    """
    return val % len(table[IX_TAB])

def creer_tablehash(taille):
    """
      Cree une table de hachage vide d'une taille donnee
    """
    return [0, [Liste() for _ in range(taille)]]

def afficher(table):
    """
      Affiche le contenu de la table
    """
    print(f"Table contenant {table[IX_NBR]} elements :")
    for idx, lst in enumerate(table[IX_TAB]):
        print(f"[{idx}] {lst}")

def main():
    """
      Fonction principale
    """
    table = creer_tablehash(4)
    afficher(table)
    for _ in range(10):
        inserer(table, randint(0, 9))
        print()
        afficher(table)
    print()
    for val in range(5):
        if est_presente(table, val):
            print(f"{val} est presente dans table")
        else:
            print(f"{val} est absente de la table")
    print()
    afficher(table)
    while table[IX_NBR] > 0:
        val = randint(0, 9)
        if supprimer(table, val):
            print("\nSuppression de la valeur", val)
            afficher(table)
```

Implanter une fonction d'insertion d'une valeur dans la table `inserer(table, val)` qui ne renvoie rien, une fonction `est_presente(table, val)` testant si une valeur donnée est présente dans la table et qui renvoie donc un booléen, ainsi qu'une fonction de suppression `supprimer(table, val)` qui supprime une occurence de la valeur dans la table et renvoie `True` si une suppression a bien eu lieu et `False` si la valeur n'était pas présente dans la table.