```
# Code écrit par Morgane Farez L1 CMI Informatique USMB
# Hash Table version 3
#dual hashing : 2 fonctions de hachage

collision = [0]
tab_nb_elements = [0]

#taille des listes de collisions
tab_liste_collisions = [0]

### DEBUT ###

### FONCTION HACHAGE ###
def hashFunction1(cle):
    """
    Associe une case a un element donne.
    Parameters
    ----------
    cle : entier, chaine de caracteres
    table : table de hachage
    Returns
    -------
    pos : entier
        indice de l'element dans la table de hashage.

    """
    if type(cle) == int:
        pos = cle
    else :
        pos = 0
        for i in range(len(cle)):
            pos = pos*11 + ord(cle[i])*73#i #ord() permet de recuperer le numero ascii du caractere
    return pos%(2**64)

def hashFunction2(cle):
    """
    Associe une case a un element donne.
    Parameters
    ----------
    cle : entier, chaine de caracteres
    table : table de hachage
    Returns
    -------
    pos : entier
        indice de l'element dans la table de hashage.

    """
    if type(cle) == int:
```

```
        pos = cle*3
    else :
        pos = 0
        for i in range(len(cle)):
            pos = pos*17 + ord(cle[i])*97#i #ord() permet de recuperer le numero ascii du caractere
    return pos%(2**64)


### TABLE DE HACHAGE ###
class HashTable:
    def __init__(self, size):
        '''
        creer un tableau de tableaux de la taille demandee a l'initialisation
        Parameters
        ----------
        size : entier
        Returns
        -------
        None.
        '''
        self.table = []
        self.nb_collisions = 0
        self.nb_case_remplie = 0 # compte le nombre de cases remplies dans la table
        self.nb_elements = 0 # compte le nombre de couples cle/valeur inseres dans la table
        self.taille = size
        for i in range(self.taille):
            self.table.append([])

    def __str__(self):
        """
        Permet un affichage de la table plus agréable et ressemblant à celui de python
        Returns
        -------
        affichage : chaine de caractères
        """
        affichage = '{'
        for i in range(len(self.table)):
            if len(self.table[i]) != 0 :
                for j in range(len(self.table[i])):
                    affichage = affichage +  f'{self.table[i][j][0]} : {self.table[i][j][1]}, \n '
        affichage += '}'
        return affichage

    def ajouter_valeur(self, cle, valeur):
        """
        Ajoute l'element à la position indique de la table
        Parameters
        ----------
```

```
    position : entier (< len(table))
    valeur : pas de type particulier
    Returns
    -------
    None.
    """
    position1 = (hashFunction1(cle)) %  len(self.table)
    position2 = (hashFunction2(cle)) %  len(self.table)

    # si l'éléments n'est pas encore present dans la table
    if self.recherche_valeur(cle)==None:
        # récupère la position où il y a la liste est la plus petite
        position = self.quelle_pos(position1, position2)
        if self.nb_elements ==100 or self.nb_elements ==107:
            tab_liste_collisions.append(len(self.table[position])+1)
            # complete tableau pour le nuage de points des tests
            tab_nb_elements.append(self.nb_elements)
            collision.append(self.nb_collisions)
        if len(self.table[position])==0:
            self.nb_case_remplie += 1
        else:
            self.nb_collisions += 1
            tab_liste_collisions.append(len(self.table[position])+1)
            # complete tableau pour le nuage de points des tests
            tab_nb_elements.append(self.nb_elements)
            collision.append(self.nb_collisions)

        self.table[position].append((cle, valeur)) #ajoute l'élément à cette liste
        self.nb_elements += 1

    else: #si l'element est deja dans la table
        maj = False
        for i in range(len(self.table[position1])):
            if self.table[position1][i][0] == cle:
                self.table[position1][i] = (cle, valeur)
                maj = True
        if maj == False:
            for j in range(len(self.table[position2])):
                if self.table[position2][j][0] == cle:
                    self.table[position2][j] = (cle, valeur)


    self.agrandir_table() # agrandit la table si necessaire



def __setitem__(self, cle, valeur):
    '''
```

```
        permet d'ajouter un couple cle/valeur a la table avec de la facon
        suivante:
            t['r']='t'
        '''
        self.ajouter_valeur(cle, valeur)

    def quelle_pos(self, position1, position2):
        '''
        renvoie a quelle position il faut inserer l'element
        Parameters
        ----------
        position1 : entier
        position2 : entier
        Returns
        -------
        position : entier
        '''
        position = position1
        if len(self.table[position1]) > len(self.table[position2]):
            position = position2
        return position

    def recherche_valeur(self, cle):
        '''
        Renvoie la valeur correspondante a la cle
        Parameters
        ----------
        cle : entier ou chaine de caracteres
        Returns
        -------
        valeur : pas de type particulier
        '''
        # decommenter pour verifier l'invariant de boucle
        valeur = None
       # ind1 = -1
       # ind2 = -1

        # test si l'element est place dans la table a l'aide de la fonction hash1
        position1 = (hashFunction1(cle))%len(self.table)
        for i in range(len(self.table[position1])):
            if self.table[position1][i][0] == cle:
                #ind1 = i
                valeur = self.table[position1][i][1]
                return valeur #break

        # test si l'element est place dans la table a l'aide de la fonction hash2
        position2 = (hashFunction2(cle))%len(self.table)
        for j in range(len(self.table[position2])):
```

```
        if self.table[position2][j][0] == cle:
            valeur = self.table[position2][j][1]
            #ind2 = j
            break

    #invariant: un element place avec hash1 ne peut pas aussi etre place avec hash2 et inversement
    # if valeur!= None:
    #    assert ind1!=-1 or ind2!=-1, "Element en double"

    return valeur


def __getitem__(self, cle):
    '''
    permet de faire une recherche d'une valeur dans une table T de la facon
    suivante:
        T[cle]
    Parameters
    ----------
    cle : eniter ou chaine de caracteres
    Returns
    -------
    renvoie la valeur associee a la cle
    '''
    return self.recherche_valeur(cle)

def supprimer_tuple(self, cle):
    '''
    Supprime le couple clé-valeur de la clé indiquée
    Parameters
    ----------
    cle : entier ou chaine de caractères
    Returns
    -------
    None.
    '''
    position1 = (hashFunction1(cle))%len(self.table)
    for i in range(len(self.table[position1])):
        if self.table[position1][i][0] == cle:
            del self.table[position1][i]
            self.nb_elements -=1
            if len(self.table[position1])==0:
                self.nb_case_remplie -= 1
            return #break

    position2 = (hashFunction2(cle))%len(self.table)
    for i in range(len(self.table[position2])):
        if self.table[position2][i][0] == cle:
```

```
            del self.table[position2][i]
            self.nb_elements -=1
            if len(self.table[position2])==0:
                self.nb_case_remplie -= 1
            break

    def __delitem__(self, cle):
        '''
        permet de supprimer un element d'une table t de la façon suivante :
            del t['em']
        '''
        return self.supprimer_tuple(cle)

    def agrandir_table(self):
        """
        test si la table est remplie a plus de 2/3 de sa taille et double sa
        taille si c'est le cas
        Returns
        -------
        None
        """
        if self.nb_case_remplie >= ((2/3)*self.taille):
            ancienneTable = self.table
            self.__init__(self.taille*2)
            for i in range(len(ancienneTable)):
                for j in range(len(ancienneTable[i])):
                    self.ajouter_valeur(ancienneTable[i][j][0], ancienneTable[i][j][1])
```