Mon Super Titre Personnalisé

September 13, 2026

[3]:
```
import sys
print(sys.executable)
```

```
/usr/bin/python3
```

1 Table de hachage

Pour implémenter un dictionnaire, on crée une liste de taille fixe N puis, à chaque clé, on associe un indice i compris entre 0 et $N-1$. On stocke le tuple (clé,valeur) à l'indice i.

On appelle **fonction de hachage**, la fonction qui, à chaque clé, fait correspondre l'indice de la clé.

Une fonction de hachage n'est pas injective en général. Autrement dit, un même indice peut très bien correspondre à plusieurs clés. On dit alors qu'il y a collision.

Prenons l'exemple où les clés sont des chaînes de caractères. La fonction qui renvoie la longueur est une fonction de hachage. Dans ce cas, des clés de même longueur provoquent des collisions.

Les collisions sont toujours possibles mais doivent être le plus possible évitées. Pour cela, on doit choisir une valeur de N assez grande (en tout cas nettement plus grande que la longueur du dictionnaire) et une "bonne" fonction de hachage.

Lorsqu'il y a collision, on peut par exemple choisir la première case du tableau disponible après l'indice obtenu par la fonction de hachage.

Exercice 1

On considère la fonction de hachage suivante. Pour chaque clé de longueur n, on note $a_0, \dots, a_{n-1}$ le code ASCII de ses lettres de droite à gauche et on considère le polynôme

$$P(X) = \sum_{k=0}^{n-1} a_k X^k$$

Ecrire la fonction de hachage `hachage` qui calcule $P(256)$ modulo N.

Rappels

- le reste et le quotient de la division euclidienne de a par b sont donnés par les opérations `%` et `//`.
- la fonction `ord()` convertit tout caractère alphanumérique en son code ASCII qui est un nombre compris entre 0 et 255.

[1]:
```
print('reste=',19%7)
print('quotient=',19//7)
print('A:',ord('A'),';  B:',ord('B'),';  a:',ord('a'))
```

```
reste= 5
quotient= 2
A: 65 ;  B: 66 ;  a: 97
```

[2]:
```
def hachage(cle,N):
    s=0
    ...
    return s
```

[ ]:
```
hachage('mot',300)
```

Exercice 2

1. Ecrire une foncion `implementation1` qui stocke un dictionnaire dans une liste de taille N en utilisant la fonction hachage de l'exercice 1. La fonction renvoie `False` s'il y a une collision, elle renvoie la liste s'il n'y en a pas. La fonction prend en entrée l'entier N et les listes `Cles`, `Valeurs`.
2. On veut implementer le dictionnaire dont les clés sont les mois en anglais et les valeurs les mois correspondant en français. Trouver le plus petit N qui convient.

[9]:
```
def implementation1(N, Cles, Valeurs):
    ...
    return L
```

[ ]:
```
mois=['janvier','février','mars','avril','mai','juin','juillet',
      'août','septembre','octobre','novembre','décembre']
months=['january','february','march','april','mai','june',
        'july','august','september','october','november','december']
N=12
...
print(N)
```

Exercice 3

Ecrire une fonction `recherche1` dont les paramètres sont une chaîne de caractère `mot` et une liste L obtenue par la fonction `implementation1`.

Cette fonction calcule la valeur hachage de `mot`, vérifie qu'il se trouve bien dans la liste L. La fonction renvoie dans ce cas le mot associé sinon elle renvoie `False`.

[11]:
```
def recherche1(mot,L):
    ...
```

[ ]:
```
L=implementation1(27,months,mois)
recherche1('march',L)
```

Exercice 4

1. Implémenter un dictionnaire, sans utiliser de table de hachage, à l'aide d'une liste dont les éléments sont les tuples `(clé,valeur)`. La fonction prend en entrée les listes *Cles* et *Valeurs*.
2. Ecrire une fonction qui renvoie la valeur d'une clé d'un dictionnaire. La fonction prend en entrée une liste L (contenant le dictionnaire) et une chaîne de caractères `mot`. Si la clé n'est pas dans le dictionnaire, la fonction renvoie `False`.

[15]:
```
def implementation2(Cles,Valeurs):
    ...
```

[24]:
```
def recherche2(L,mot):
    ...
```

[ ]:
```
L=implementation2(months,mois)
recherche2(L,'february')
```

Exercice 5

Calculer la complexité des deux fonctions de recherche. Comparer les deux implémentations (avantages et inconvénients de chaque méthode).

Exercice 6

1. Ecrire une fonction `implementation3` qui améliore la fonction `implementation1`: en cas de collision, la fonction cherche la première place libre après la valeur de hachage (on revient à 0 si on arrive au bout de la liste).
2. Ecrire la fonction de recherche associée à cette implémentation.
3. Tester en prenant $N = 50$

[26]:
```
def implementation3(N,Cles,Valeurs):
    ...
```

[ ]:
```
L=implementation3(50,months,mois)
print(L)
```

[28]:
```
def recherche3(mot,L):
    ...
```

[ ]:
```
recherche3('december',L)
```