Hashing et bio-informatique

- Tous les langages de programmation modernes ont déjà des fonctions de hachage. La, on va les coder nous-même.

Exercice 1 : Codage

Un **k-mer** est un mot de taille **k**.

En bio-informatique, les mots sont souvent des morceaux d'ADN composés uniquement des lettres **A**, **C**, **G**, **T**.

- Dans l'exemple j'ai les lettres : A, C, G, T.
- Je peux avoir 2 chaines :

« ACCGGTTACC … »

« CCAGTTGAC … »

- Je me pose la question quelle est la distance, sachant que l'utilisation de **la distance de Levensthein** de semaine dernière ne suffira pas car les chaines peuvent être très long.

La distance de Levensthein (ou distance d'édition), mesure le nombre minimal de modifications nécessaires pour transformer une chaîne en une autre, grâce à des opérations d'insertion, de suppression ou de substitution. Par exemple, la distance entre A = "banane" et B = "banana" est de 1 car il suffit de substituer la dernière lettre de A par un a.

(1) Comment transformer un **k-mer** en un entier de taille **2k** bits ?

- Je prends un k-mer : un mot de taille k.
- Par exemple, on peut lister tous les 3-mers :

CCAGTTGAC ...

CCA

CAG

AGT

• • •

- On va coder A comme 0 (choix arbitraire) :
  - A = 0
  - C = 1

- G = 2
- T = 3

- En binaire :

A	0	00
C	1	01
G	2	10
T	3	11

- Donc, CCA va correspondre en binaire a :

CCA = 010100

$= 0 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0$

$= 1 * 2^4 + 1 * 2^2 = 16 + 4 = 20.$

CAG = 010010

$= 0 * 2^5 + 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$

$= 1 * 2^4 + 1 * 2^1 = 16 + 2 = 18.$

AGT = 001011

$= 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$

$= 1 * 2^3 + 1 * 2^1 + 1 * 2^0 = 8 + 2 + 1 = 11.$

Et ainsi de suite…

Donc, quelle est l'intervalle des valeurs finales **(en orange)** ?

$2^7 - 1 = 63 \longrightarrow$ La valeur maximale est 63.

(2) Écrire un algorithme qui, à partir d'une séquence ADN de taille **n**, écrit toutes les paires k-mer/valeur **($k \leq n$)**. Quelle est sa complexité ?

CCAGTTGAC …

CAA = 20

CAG = 18

AGT = 11

…

On doit écrire un algo qui fait ça…

Il y a un algorithme naïf et algo plus intelligent.

Algo naïf

Evidemment, on peut écrire simplement une fonction qui prend 3 lettres et nous donne le code : une fonction **code3(s)** qui étant donné une chaine **s** de taille 3 renvoie son code. On peut également écrire une fonction plus simple qui nous donne le code pour chaque caractère…

On va écrire une boucle qui prend d'abord les 3 premières caractères du code, et ainsi de suite.

```
def code3(s) : # s : une chaine de taille 3.
    return ( 16*code1(s[0]) + 4*code1(s[1]) + code1(s[2]) ) # 16 -> faire un
shift, code1() c'est pour une lettre

# Une fonction qui imprime une chaine et son code.
def code(s) :
    for i in range( len(s) - 3 )
        print( s[i:i+3], code3(s[i:i+3]) ) # (i+3) -> on python c'est exclue...
```

Dans ce code il y a des choses inutiles : Je calcule le code de « CCA », ensuite, je calcule le code de « CAG »... donc pour chaque lettre je vais appeler la fonction **code3(s)** 3 fois. Ce n'est pas efficace. Est-ce qu'on peut calculer une fois le code de chaque lettre ?

Par exemple, comment passer du code de CCA (= 20) au code de CAG (= 18) ? Pour CCA on a en binaire : 010100. Maintenant, quoi faire pour CAG ?

On doit décaler de 2 a gauche. « Décaler de 2 à gauche », ça veut dire quoi en terme binaire ? Dans tous les langages de programmation on a un « Shift » pour ça.

CCA	010100
	shift
	01010000
	On ajoute G
	01010010
	On enlève le début et ça nous donne on veut enlever le « 01 » au début, c.-à-d. : enlever le bit numéro 7 et le bit numéro 8
CAG	010010

Du coup, on n'a pas besoin de calculer les lettres qu'on a déjà calculé. Ainsi le fait de manière plus efficace.

Pour shifter en Python : <<

Comment faire pour enlever les 2 premières bits : Par exemple, on pourrait faire un ET bit par bit …. (à compléter).

```
def codechar(c):
    if c == 'A': return 0
    if c == 'C': return 1
    if c == 'G': return 2
    if c == 'T': return 3

def codekmer(s):
    # s composé de 'A' 'C' 'G' 'T'
    res = 0
    for c in s: # boucle sur tous les caracteres de la chaine s.
        res = (res << 2) + codechar(c)
    return res
```

Exercice 2 : Tables de hachage

On veut maintenant comparer le contenu de deux séquences

(et donc deux ensembles de **k-mers**).

Pour cela, on peut commencer par énumérer tous les **k-mers** d'une des deux séquences et les stocker dans une table de hachage.

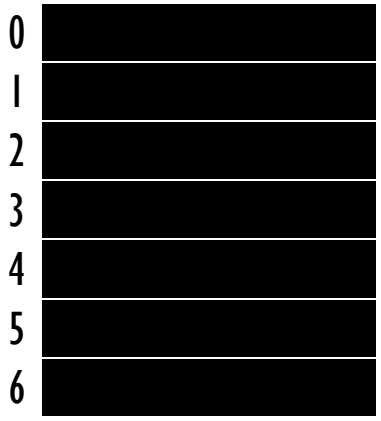
Ensuite, on énumère tous les **k-mers** de la seconde séquence et on compte ceux qui entrent en collision avec ceux indexés dans la table.

Commençons par créer une table de hachage pour recevoir les **k-mers** de la première séquence.

Pour simplifier, nous supposerons que tous les **k-mers** d'une séquence sont différents

Une table de hachage

- Un tableau d'une certaine taille.
- La taille est le paramètre de la table de Hachage.
- Par exemple, en python, les dictionnaires sont faits avec des tables de hachages.
- Donc, une table de hachage c'est juste un tableau, ici on va utiliser une table de taille 7 :



Je veux calculer un indice pour « ACT » :

$$\mathbf{h}(\text{"ACT"}) = \underbrace{\mathbf{encode}(\text{"ACT"})}_{\textit{La fonction code3}}$$

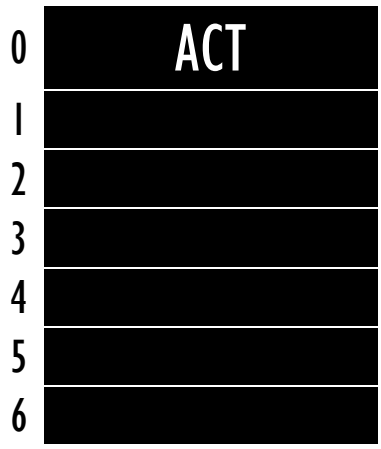
On a 7, ce qui est plus que 6. Donc, on va faire « modulo » :

$\mathbf{encode}(\text{"ACT"}) \bmod \mathbf{7}$

On fait $\mathbf{7} \bmod \mathbf{7}$

Ca nous donne 0 donc on place « ACT » a la place 0 de la table de hachage.

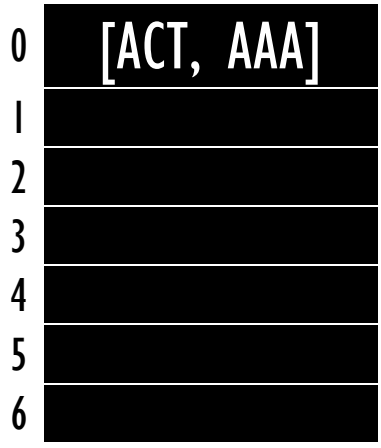
En python le contenu initial de la table est **None**, mais après ?



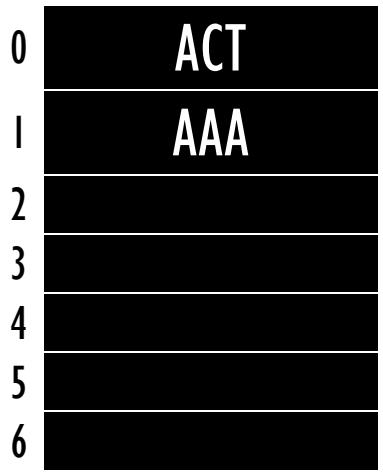
En si on a un autre mot avec le même code ? « AAA » par exemple :

⟹ collision : 2 chaines avec la même valeur de **h**()

Solution 1 : chaque case du tableau va être une liste



Solution 2 : On pose « AAA » dans le case au-dessous

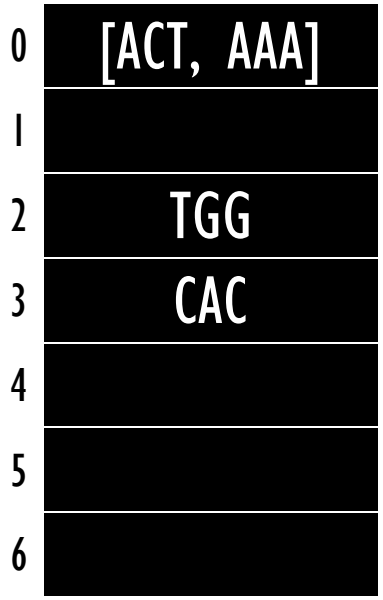


Maintenant, si on veut ajouter quelque chose avec le code 1 ?

Ça va être compliquer de trouver les éléments qu'on cherche puisqu'ils ne sont pas forcément là où on les attend.

TGG, CAC : ça va être où ?

TGG : 58 modulo 7 = 2
CAC : 17 modulo 7 = 3



(3) Que se passe-t-il lorsque l'on cherche à insérer le **k-mer CAA** ?

Pour résoudre ce problème on peut choisir
de ne pas avoir un seul élément par case mais une liste.

Le code de CAA c'est 16 modulo … = 2.
Donc, on peut l'ajouter dans une liste.

0	[ACT, AAA]
1	
2	[TGG, CAA]
3	CAC
4	
5	
6	

(4) Quel impact cela a sur les complexités d'insertion
et de requête de la table ?

- L'ajout de listes augmente la complexité lorsque on veut chercher un mot dans la table.

L'autre méthode en cas de collision : mètre le mot une case plus bas.

- Si il y a beaucoup de collisions a un endroit X et si je continue d'ajouter des choses je risque de tomber dans une zone déjà occupée, du coup : j'aurai des places très chargé et des places très vide.

Autres solutions (cf. internet, si ça nous intéresse) :

1. “Sondage quadroitique”
2. Avoir 2 fonctions de hachage : le double hachage. En cas de conflit, on applique la 2em.

(5) Écrire la fonction de requête

Méthode a privilégier : celle avec les listes.

Une fois qu’on a cette structure de données, comment savoir combien de sous-séquence en commun ?

- Je prends la première chaine, je la découpe en sous-séquences, je pose dans la table de hachage.
- Ensuite, pour la chaine 2, je regarde pour chaque sous-séquence, est-ce qu’elle est dans la table ou pas ?

Donc, dans le code on aura les fonctions suivantes :

- Créer une table.
- Insérer un élément dans la table de hachage
- Savoir si quelque chose se trouve dans la table

….

Exercice 3 : Min-sketch

Quand l'ensemble de données est vraiment très grand,
il n'est plus raisonnable de stocker tous les hash.

En sous-échantillonnant notre ensemble de **k-mers** on peut tout de même obtenir une bonne estimation de la similarité entre nos deux séquences :

On ne compare plus les hash de l'ensemble **A** à l'ensemble **B**

mais un sous ensemble **x** de **A** et un sous ensemble **y** de **B**.

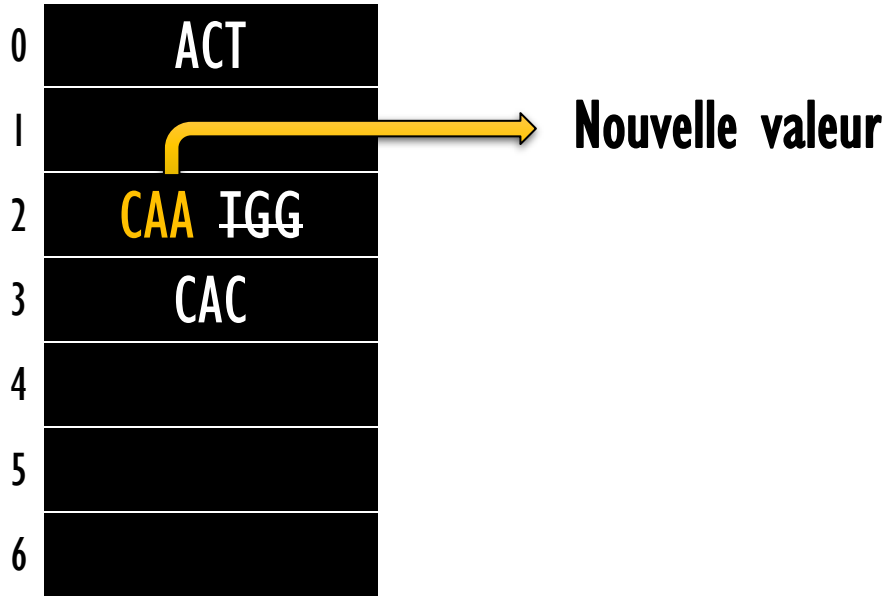
Une méthode de sous-échantillonnage serait de ne pas gérer les collisions,
mais en cas de conflit de juste remplacer la valeur précédente dans la table de hachage par la nouvelle valeur

Ce qui nous embête ce sont les collisions. Om va imprimer une empreinte de chaine, si on va voir que dans la table de hachage il y a déjà quelque chose dans la case de la table, on va remplacer cette valeur par une nouvelle.

J'ai

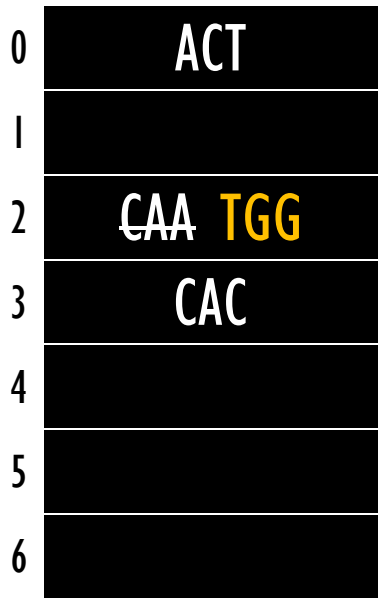
ACT
TGG
CAC
CAA

On veut les insérer dans une table de hachage.



Et si on a

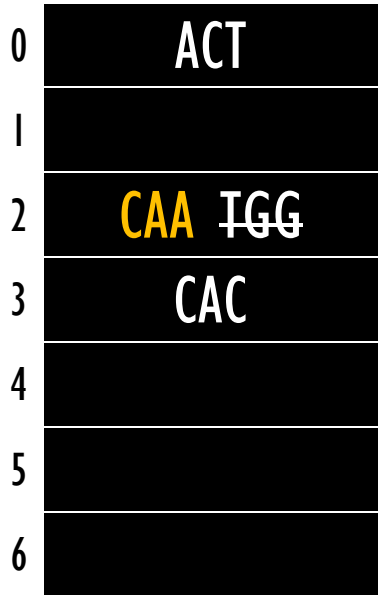
CAA
ACT
TGG
CAC



Bien qu'on ait le même ensemble de sous-séquences, on obtient 2 résultats différent selon l'ordre.

Comment faire pour avoir le même résultat ?

En cas de conflit : garder celui avec le code le plus petit.



Ça s'appelle : le calcul d'un **min-sketsh**.

```
def inserttable2(t, size, s): # une fonction pour inserer
    codek = codekmer(s)
    code = codekmer(s) % size
    if t[code]: # Si la table a cette position est pas vide
        if codek < codekmer(t[code]):
            t[code] = s
    else:
        t[code] = s
```

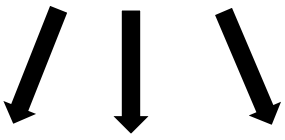
(2) Grâce à cette méthode on peut calculer
pour n'importe qu'elle séquence un Min-sketch, une table de hash d'une taille fixe
qui contient un sous ensemble des k-mers présents dans une séquence.
Écrivez une fonction qui réalise cette opération.

Paramètre 1 : La taille de la table de hachage.

Paramètre 2 : k : le k du « k-mer » (taille de sous-séquence)

- Pour une séquence donnée : calculer un Min-sketch à partir de cette séquence.
- **Min-sketch** : table de hachage avec une valeur max par case.
- On calcule le Min-sketch de la premier séquence $\Longrightarrow$ table de hachage, on calcule le Min-sketch de la 2eme séquence $\Longrightarrow$ table de hachage. On compare.
- Je fais une approximation du degré d'égalité entre les 2 chaines.
- 2 chaines sont similaires si leur Min-sketch est similaire.
- Donc, étant donné une séquence d'entrée, on veut une fonction qui calcule le Min-sketch de la chaine.

minSketch(s, k, size)



Chaine de caractères — k-mer — Taille du tableau de hachage

- **Return :** un tableau de taille **size** (initialisé avec None…)

```python
def table(size): # une fonction qui cree la table
    return size*[None]

def minsketch(s,k,size):
    t = table(size)
```

(3) Écrivez une fonction qui estime la distance entre deux séquences grâce à un Min-sketch.

- Ecrire un code qui compare 2 tableaux : combien d'entrés sont les même (en pourcentages par exemple).
- L'idée est que si j'ai des chaines énormes, je calcule l'empreinte de cette chaine, et je considère que 2 chaines sont pareilles si elles ont la même empreinte.
- On a besoin d'abord du code qui calcule le d'une sous-séquence.

```python
def compare(s1,s2,k,size):
    t1 = minsketch(s1,k,size)
    t2 = minsketch(s2,k,size)
    count = 0
    for i in range( len(t1) ):
        if t1[i] == t2[i]:
            count += 1
    return count/len(t1)
```

(4) Comparez son résultat avec la fonction de l'exercice précédant.

Est-ce qu'on obtient les même résultats ?

D'où vient l'erreur ?

Comment la réduire ?

Est-ce qu'on peut estimer ce taux d'erreur ?