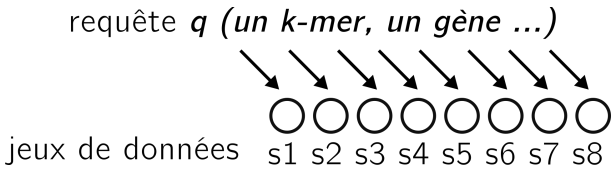
Structures de données
Hachage

Master 1 MISO

Camille Marchet
CNRS, CRIStAL Lille, France

camille.marchet@univ-lille.fr

Introduction - Toujours la même question de bioinformatique



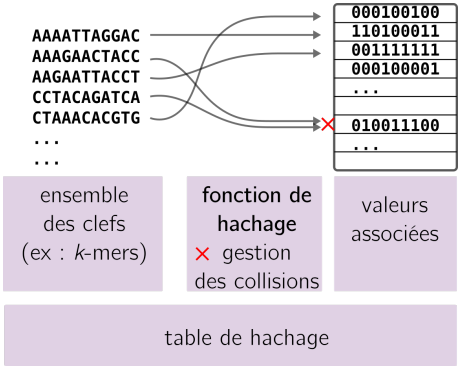
On a évoqué la complexité la requête : $\mathcal{O}(n)$ **si** la recherche est en temps constant dans chaque jeu de donnée.
$\rightarrow$ comment faire une recherche en temps constant et en associant une information (exemple : AATTGA $\rightarrow$ abondance dans s1)

Table de hachage dans la vraie vie

Un annuaire (papier ou Contacts du téléphone):

- nom-prénom écrit à une certaine page de l'annuaire/dans la mémoire du téléphone
- associé à un numéro de téléphone/adresse

Table de hachage - Définitions



- Clef, Valeur
- Association
- Table (structure où on range (*clef*, *valeur*) à une adresse)
- Fonction de hachage
- Stratégie de hachage

Opérations :

- Ajout rapide
- Recherche rapide

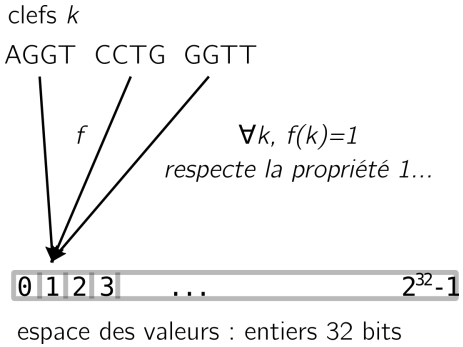
Fonction de hachage

Associer un entier dans `[1...m]` (par exemple l'ensemble des entiers 32 bits) à une clef

- C++ `std::hash`,
- Python `hash()`,
- Java `Object.hashCode()`

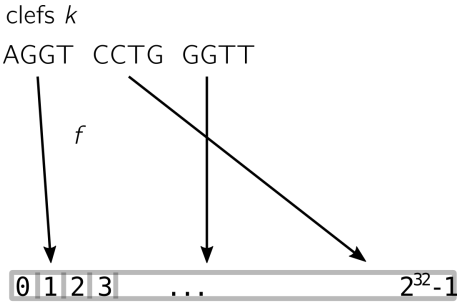
Fonction de hachage

1. Propriété de distribution : chaque élément haché dans `[1,m]` devrait avoir la même probabilité d'être attribué à chaque entier



Fonction de hachage

2. Propriété d'indépendance : l'entier attribué à un élément ne devrait pas influencer l'attribution pour les autres éléments



Adressage dans une table

- **Adressage direct** : la clef == l'adresse
- **Adressage indirect** : l'adresse est une fonction de la clef (on utilise une fonction de hachage)

Adressage - Exercice

Exemple d'adressage direct par l'encodage de k-mers.
Remarquons qu'on peut coder chaque base de cette manière:

`A:00, C:01, G:10, T:11`

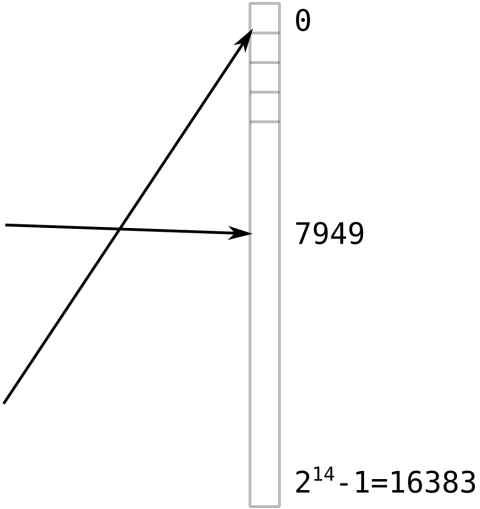
Comment faire un adressage direct de ces k-mers de taille 7: `ATTGATC, GGTACCC, GTCCCAA, CCTGAGA, TTACCGA, AAAAAAA`

Adressage - Exercice

A:00, C:01, G:10, T:11

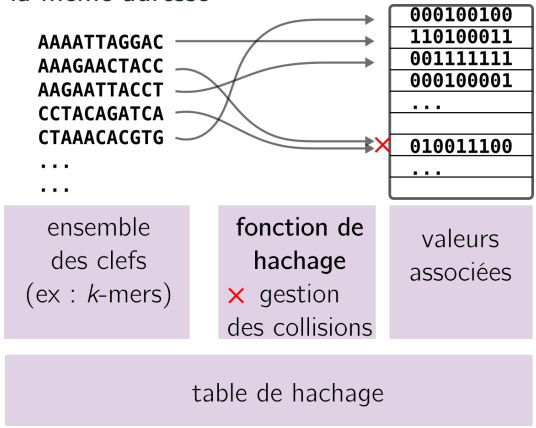
7x2=14 bits

ATTGATC : 00111110001101
GGTACCC : 10101100010101
GTCCCAA : 10110101010000
CCTGAGA : 01011110001000
TTACCGA : 11110001011000
AAAAAAA : 00000000000000

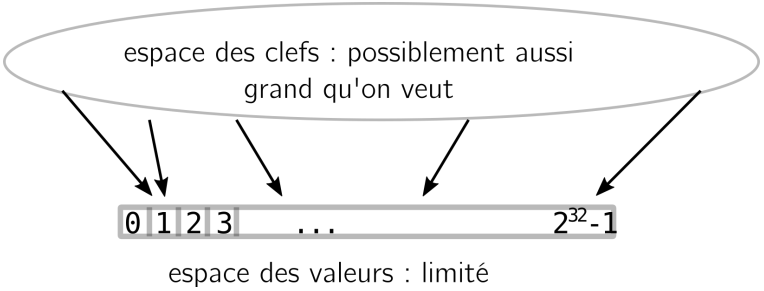


Fonction de hachage - collisions

Collision : >1 clefs à la même adresse

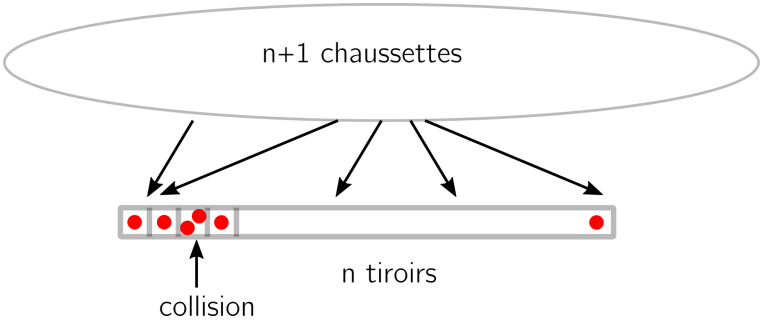


Fonction de hachage - collisions



Fonction de hachage - collisions

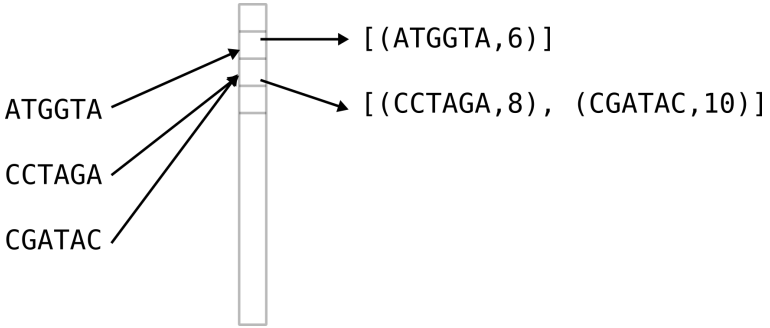
Le principe des tiroirs de Dirichlet



Tables de hachage - gestion des collisions

- Chainage (pointeur vers une liste)
- Adressage ouvert
  - linear probing
  - random probing
  - ...

Tables de hachage - Chainage



Stratégie de hachage - Linear probing

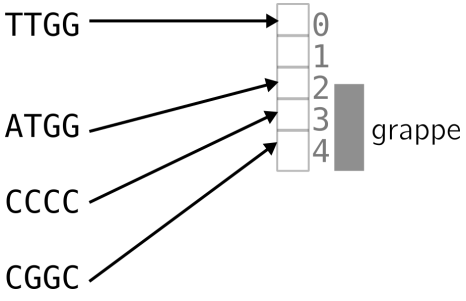
$adresse = (h(c) + i)\%T$
pour c une clef, T la taille de la table et i le nombre d'essais - 1

Exemple pour $T = 5$, adresses en gras :

c	$h(c)$	$(h(c)+i)\%5$		
		i=0	i=1	i=2
ATGG	62	**2**		
CCCC	2	2	**3**	
CGGC	37	2	3	**4**
TTGG	20	**0**		

Stratégie de hachage - Linear probing

c	$h(c)$	$(h(c)+i)\%5$		
		i=0	i=1	i=2
ATGG	62	**2**		
CCCC	2	2	**3**	
CGGC	37	2	3	**4**
TTGG	20	**0**		



Stratégie de hachage - quadratic probing, double hachage

Améliorent la problématique des grappes, mais plus difficiles à construire

- Quadratique : $adresse = (h(c) + a \times i + b \times i^2 \% T$, a, b paramètres
- Double : $adresse = (h_1(c) + i \times h_2(c)) \% M$

Tables de hachage - gestion des collisions

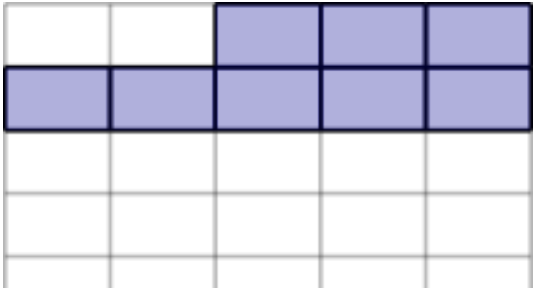
- Adressage ouvert
  - random probing
  - linear probing
  - quadratic, double, ...
- Chainage (pointeur vers une liste)

Baisse des performances: moins bonne localité de l'adressage ouvert (impacte la vitesse), listes baissent les performances en espace.

Et les tables de hachage en Python ?

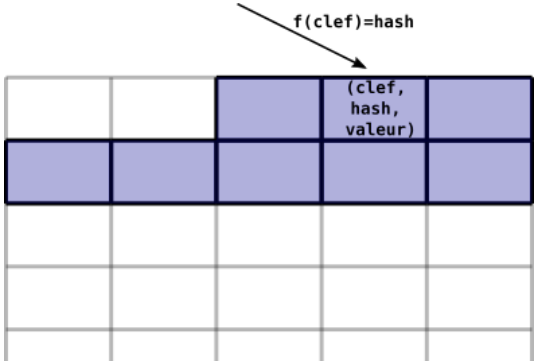
Dictionnaires en Python- implémentation

Allocation d'une zone mémoire contigüe (8 cellules pour un dictionnaire vide)



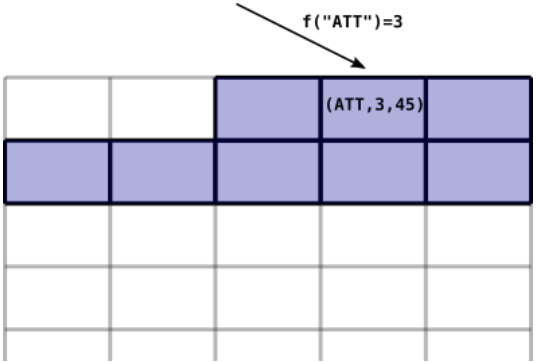
Dictionnaires - implémentation

On déduit du hash l'adresse où stocker (*clef*, *valeur*)



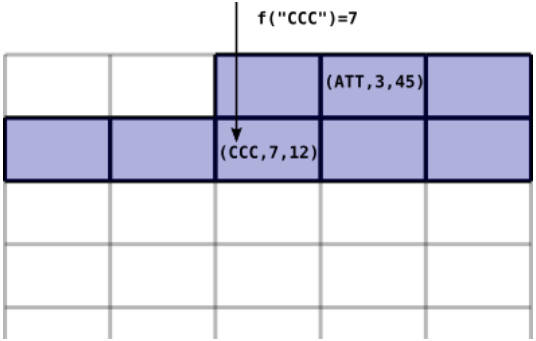
Dictionnaires - implémentation

On déduit du hash l'adresse où stocker (*clef*, *valeur*)



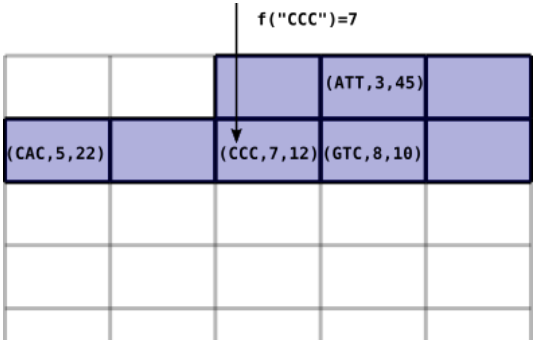
Dictionnaires - implémentation

On déduit du hash l'adresse où stocker (*clef*, *valeur*)



Dictionnaires - implémentation

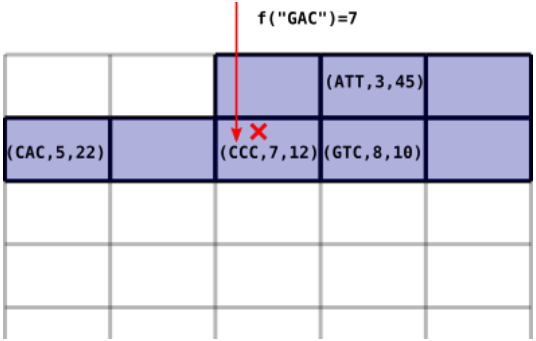
On déduit du hash l'adresse où stocker (*clef*, *valeur*)



→Un élément n'apparait qu'une seule fois

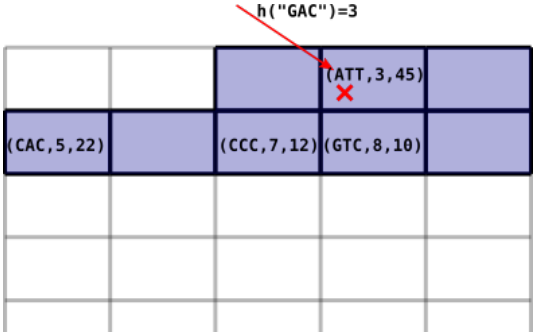
Dictionnaires - implémentation

Collisions : demandent une stratégie de hachage



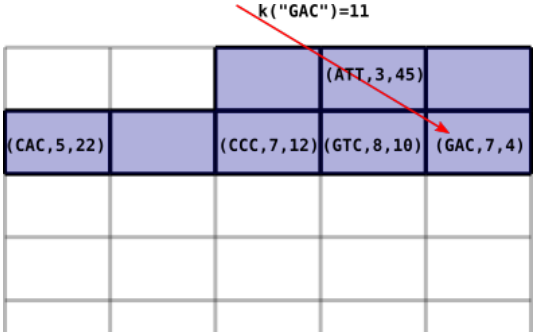
Dictionnaires - implémentation

Stratégie pour gérer les collisions (*random probing*)



Dictionnaires - implémentation

Stratégie pour gérer les collisions (*random probing*)



Fonction de hachage - Complexités

Que peut-on dire sur les complexités de l'insertion et de la recherche ?

Tables de hachage en bioinformatique

Quelques outils utilisant/produisant des tables de hachage :

- VG [Garrison et aL. 2018], méthode de construction de graphe de pangénome
- SPAdes [Bankevich et al. 2012], assembleur de short reads
- Salmon [Patro et al. 2017], méthode de quantification de RNA-seq
- Jellyfish [Marçais & Kingsford 2012], compteur de k-mers
- ...

Utilisation du hachage (au delà des tables) :

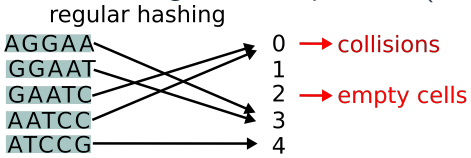
- Méthodes de *sketching* et de sous-échantillonage : comparaison rapide de séquences ou d'ensembles sans alignement (Mash [Ondov et al. 2016], Minimap [Li et al. 2016], Dashing [Baker et al. 2019])
- Filtres de Bloom (cf TP)

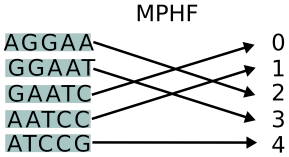
Tables de hachage - en bioinformatique

- La dynamicité (pouvoir ajouter/retirer des éléments) est coûteuse
- Globalement créer des tables de hachages pour un grand nombre de k-mers est coûteux
- Solutions :
  - se spécialiser sur un type de valeur particulière (par exemple les comptages)
  - pré-calculer une fonction de hachage spécialisée sur l'ensemble des clefs en entrée, mais pas d'ajout
  - stocker les clefs de manière plus compacte/y accéder de manière rapide

Stratégies avancées de hachage

Fonctions de hachage minimal parfaites (MPHF)



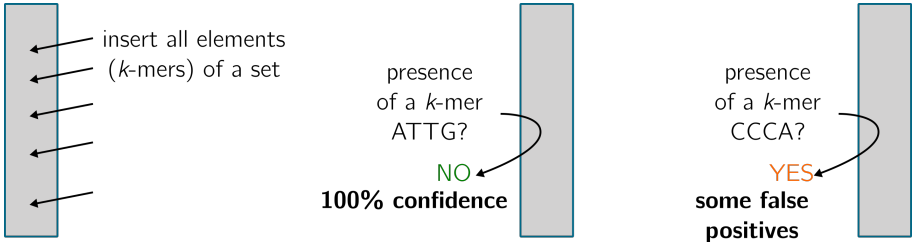


- Elles sont construites sur un ensemble statique de clefs
- Borne inférieure théorique du coût de ces fonctions: 1.44 bits par clef

En bioinformatique on utilise des stratégies non-optimal en coût par clef ($\sim$ 4 bits par clef) mais rapides à requêter/construire :

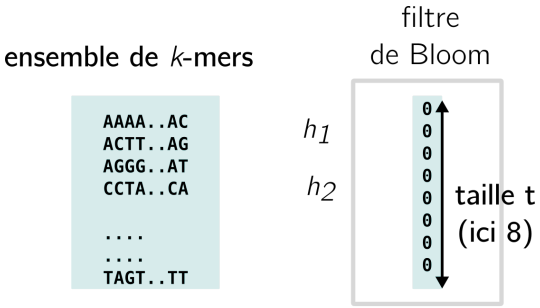
- BBHASH [Limasset et al. 2014], PTHash [Pibiri et al. 2021] $\leftarrow$ problème ouvert

TP filtres de Bloom - intuition

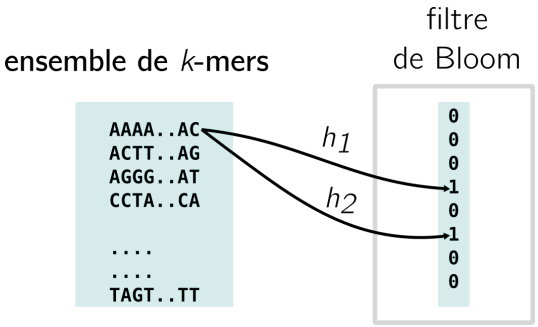


[Bloom 1970]

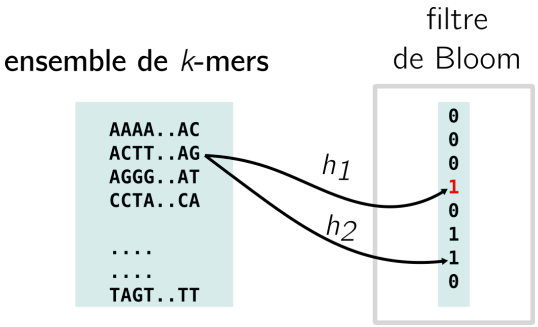
TP filtres de Bloom - construction



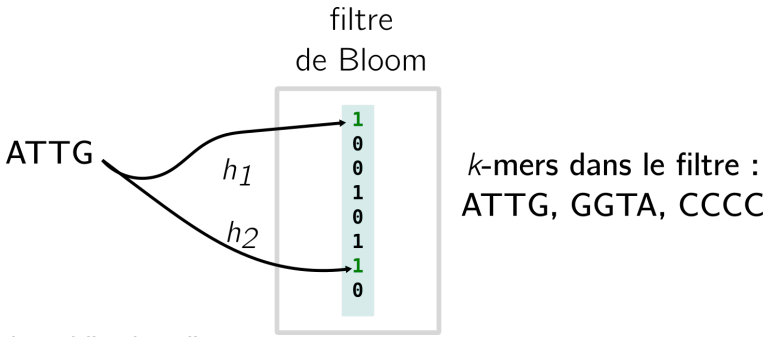
TP filtres de Bloom - construction



TP filtres de Bloom - construction

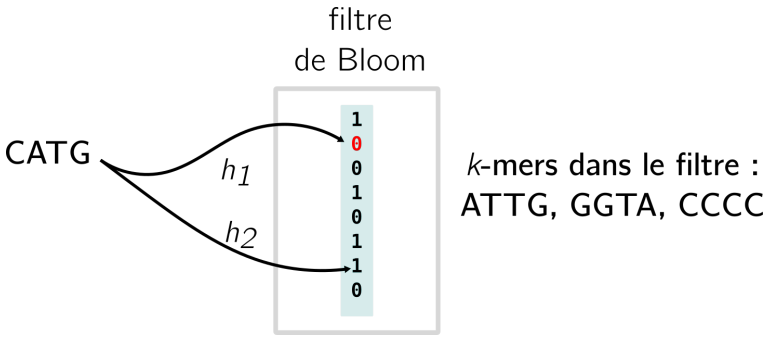


TP filtres de Bloom - requête



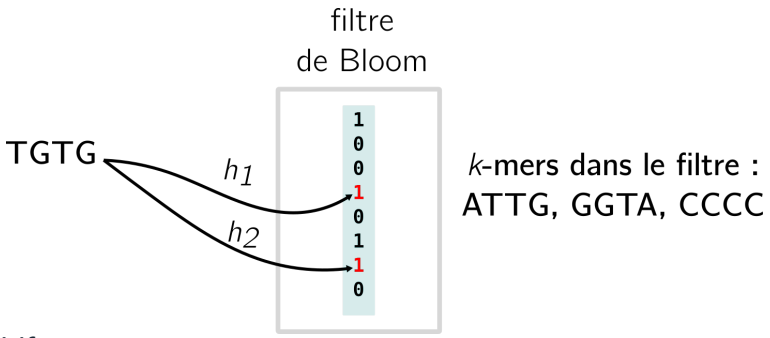
→ le filtre répond "présent"

TP filtres de Bloom - requête



$\rightarrow$ le filtre répond "absent"

TP filtres de Bloom - requête



→ faux positif