BLAST

BLAST : recherche par hachage + théorie de probabilités pour alignements locaux

Hachage — idée principale :
pour alignement local rapide entre S et T

1. fixer $k > 0$
2. comparer chaque sous-mot de longueur k (k-mer) de S avec ceux de T
3. extension des matches pour obtenir un alignement local entre S et T

$\Rightarrow$ on trouvera rapidement les alignements qui contiennent k matches consécutifs

Hachage

Deux séquences S, T
Fonction de hachage : $h\colon \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}^k \mapsto \mathcal{H}$
hit : (i, j) avec $h(S[i..i+k-1]) = h(T[j..j+k-1])$
Technique : listes $\mathsf{Occ}(u)$ de positions où $h^{-1}(u)$ apparaît

1. **pour** $i \leftarrow 1, \ldots, |S| - k + 1$ **faire**
2. $\quad \mathsf{clé} \leftarrow h(S[i..i+k-1])$
3. $\quad$ ajouter i à la fin de la liste $\mathsf{Occ}(\mathsf{clé})$
4. **pour** $j \leftarrow 1, \ldots, |T| - k + 1$ **faire**
5. $\quad \mathsf{clé} \leftarrow h(T[j..j+k-1])$
6. $\quad$ traitement [extension ?] des *hits* $(i, j)\colon i \in \mathsf{Occ}(\mathsf{clé})$

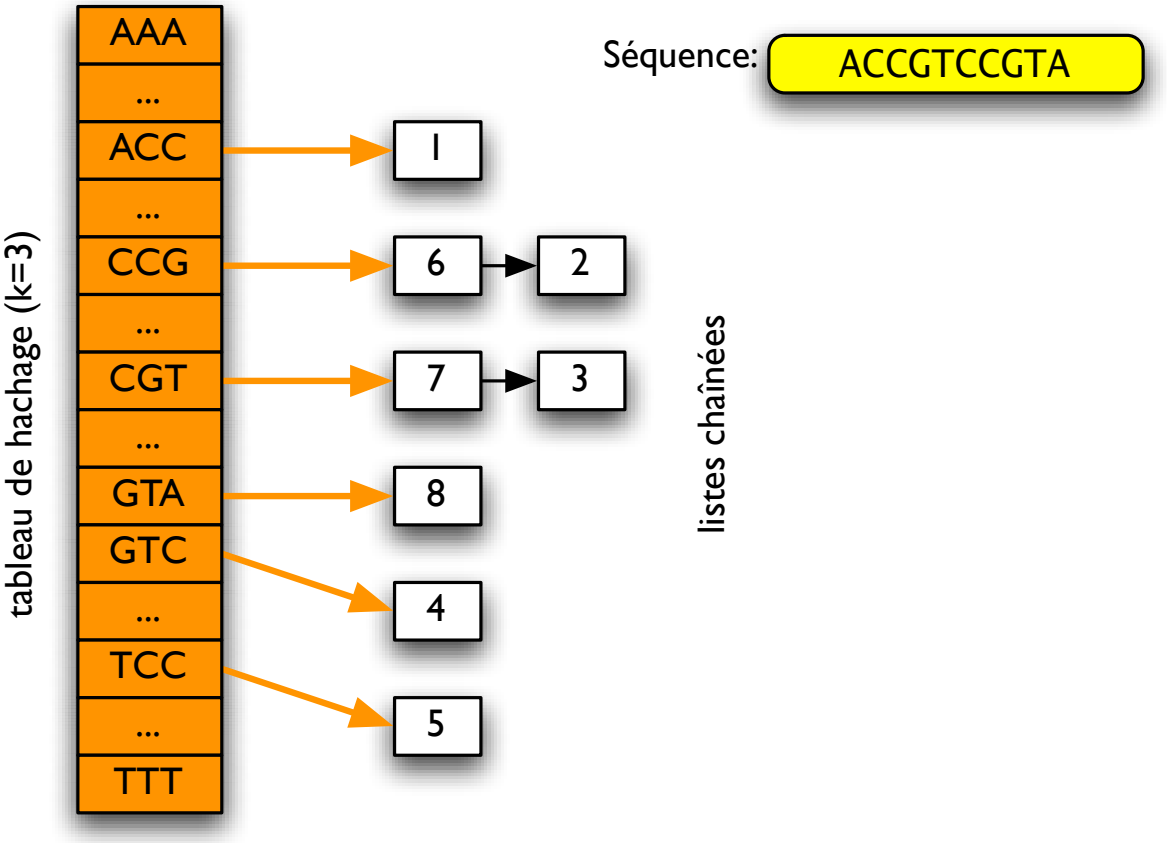
Hachage — structure de données

Trouver les clés partagés : stocker les occurrences (Occ) de tous les clés de S en un tableau de hachage

Implantation facile en Java : on peut utiliser les sous-mots w comme clés directement, `Hashtable` calcule des clés de hachage automatiquement, liste chaînée pour chaque $\mathsf{Occ}(w)$

(utilise plus de mémoire que nécessaire...)

TABLEAU DE k-MERS



Ma, Tromp et Li. *Bioinformatics* 18 :440.

IMPLANTATION

1. Encodage des k-mers en $2k$ bits :
`A` $\rightarrow 00$, `C` $\rightarrow 01$, `G` $\rightarrow 10$, `T` $\rightarrow 11$.

Java `int` : 32 bits ($k \leq 16$) ; `long` : 64 bits ($k \leq 32$).

2. Encodage des listes chaînées :
chaque position de la séquence n'apparaît qu'une fois !
Définir un tableau `int[] successeur` où `successeur`$[i]$ donne la position qui suit i dans une des listes chaînées ou égale à -1 si i est la dernier objet dans une liste.
Tête de chaque liste est trouvée par un tableau `int[] tete` où `tete`$[i]$ donne la première position ou le k-mer encodé par i se trouve.

Mémoire : $(4^k + |S|)$ fois taille de `int` (4 octets).

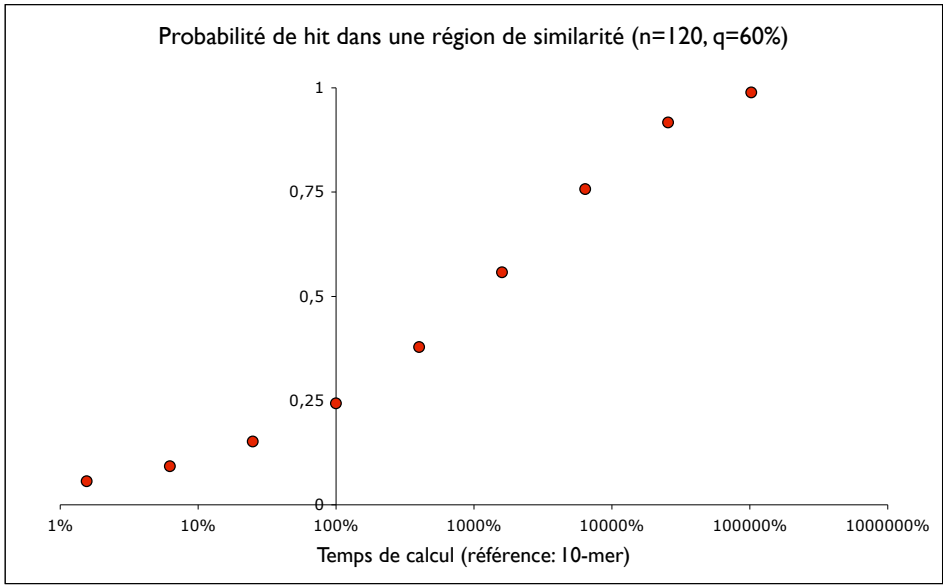
IMPLANTATION — JAVA

```
HT1  int[] tete=new int[1<<(2*k)];
HT2  int[] successeur=new int[S.length()-k+1];
HT3  pour tout i, tete[i] ← −1
HT4  for (int i=0; i<S.length()-k+1; i++){
HT5      calcul de l'encodage w pour le sous-mot S[i..i+k−1];
HT6      successeur[i]=tete[w];
HT7      tete[w]=i;
HT8  }
HT9  for (int j=0; j<T.length()-k+1; j++){
HT10     calcul de l'encodage w pour le sous-mot T[j..j+k−1];
HT11     int i=tete[w];
HT12     while (i != -1){
HT13         extension du hit (i, j)
HT14         i=successeur[i];
HT15     }
HT16 }
```

Hachage — performance

Spécificité : mesurée par nombre de *hits* entre deux séquences sans homologies (p.e., aléatoires)

Sensibilité : mesurée par la probabilité de *hit* dans une région de homologie



Hachage — nombre de *hits*

Modèle : S aléatoire, avec nucléotides iid selon p ; T aléatoire, avec nucléotides iid selon q : $\mathbb{P}\{S[i] = c\} = p_c$ et $\mathbb{P}\{T[j] = c'\} = q_{c'}$.

Fonction de hachage : identité $h(u) = u$, $\mathcal{H} = \Sigma^k$ ($\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$)

Thm. Soit $\beta = \sum_{c \in \Sigma} p_c q_c$. Alors le nombre de *hits* en espérance est $st\beta^k$ où $s = |S| - k + 1$ et $t = |T| - k + 1$.

Détour — nombre de hits

L'espérance de nombre de hits est la même pour tous les sous-mots $w \in \Sigma^k$. Est-ce que la distribution est la même aussi ? Non !

Exemple : «pas tous les mots sont créés égaux»

Exemple : T est une séquence de longueur n «au hasard»
au hasard : chaque caractère de T est 0 ou 1 avec probabilités $\frac{1}{2}$-$\frac{1}{2}$.

Quelle est la probabilité de voir $w = \texttt{00}$ ou $w = \texttt{01}$?