Méthodes de recherche de similarités dans les séquences biologiques

Module MADG

Eric Tannier, Master 1 Bioinfo Univ Lyon 1, 2022-2023
Eric.Tannier@univ-lyon1.fr

Exam à la fin

Support (ce document) disponible à partir de ma page internet

Plan du cours

I/ Recherche de séquences exactes

II/ Recherche de séquences approchées, alignement

III/ Méthodes et logiciels actuels

IV/ Limites et effets rebonds

Problème de la recherche approchée

Étant donné un texte T de taille n, un mot M de taille m, donner toutes les positions i,j telles que d(T[i..j],M) <= e

Il faut définir une distance entre mots :

Par exemple, nombre minimum d'éditions à faire pour transformer un mot en un autre. Une édition est un changement d'un caractère en un autre, l'insertion ou la délétion d'un caractère (ce qui mime des mutations).

La définition de la distance, en particulier le choix de mutations autorisées et surtout, le choix des mutations interdites (inversions, duplications), est dictée par les techniques mises en oeuvre dans la solution.

Premier principe : voisinage

A partir d’un mot M, on peut générer tous les mots M’ tels que d(M,M’) <= e

Puis faire la recherche exacte sur chaque M’

Le nombre de mots à rechercher grandit avec e, m, et a la taille de l’alphabet, il peut dépasser les capacités de calcul

Deuxième principe : découpage

Pour faire diminuer m et e, on peut appliquer le principe des cages à pigeons :

S'il y a neuf pigeons pour dix cages, au moins une sera vide.

Si un sous-mot du texte T a neuf différences avec M, alors si on coupe M en dix parties, au moins une de ces parties n'a aucune différences.

Si on cherche un sous-mot de T avec e différences, alors nécessairement en coupant M en m/k morceaux de taille k, un morceau au moins a moins de ek/m différences.

On peut donc chercher des mots de taille k avec ek/m différences.

Combinaison des deux principes

On peut donc chercher des mots de taille k avec moins de ek/m différences.

Ces mots peuvent servir d'ancres (seed) pour une recherche de m entier. Le choix de k est un compromis entre la complexité de la recherche (on voudrait un k petit) et la probabilité de tomber sur une similarité par hasard (on voudrait un k grand).

Un compromis habituel (arguments théoriques et empiriques) est de prendre k = log_a n, ou k=11, ou k=7, dépendant de l'alphabet et des techniques de recherche.

Extension

Une fois qu’on a trouvé une ancre, il faut “étendre” de part et d’autre, c’est à dire étant donné deux petits mots M1 et M2=T[i,j], savoir si la distance est au plus e

Par exemple,

d(CTAGTCAGTACGTAGCATCG,CTAGTCAGTACGTAGCATCG) = 0
d(CTAGTCAGTACGTAGCATCG,CTACTCAGTACGTAGCATCG) = 1
d(CTAGTCAGTACGTAGCATCG,CCGATACTTGCGGCTCAGAT) = ?

Alignement

La distance définit un sous-problème :

Étant donnés deux mots A et B, calculer d(A,B)

C’est le problème de l’alignement.

Il en existe plusieurs variantes, selon les coûts associés aux opérations d’édition.

Variante 1 : Plus longue sous-séquence commune

Deux séquences à comparer, M et T, de taille m et n

Une sous-séquence de M ou T est une séquence obtenue à partir de M ou T en enlevant des caractères.

Par exemple, AAA est une sous-séquence de GTAGGGGATCA.

Une sous-séquence commune de M et T est une sous-séquence de M et T

Problème : trouver la sous-séquence commune de plus grande taille. C'est le problème de l'alignement où les insertions/délétions ont un coût nul. C'est un problème utile en soi, par exemple pour la fonction `diff` de Linux

Exemple : trouver la plus grande sous-séquence de AGTGGCAAT et ACCTGCTGACA

Variante 1 : Plus longue sous-séquence commune

On note
S[i] l'élément en position i de S (début en 1)
S[i..j] le sous-mot entre les indices i et j

L(i,j) la taille de la plus longue sous-séquence commune de M[1..i] et T[1..j]

Variante 1 : Plus longue sous-séquence commune

L(i,j) la taille de la plus longue sous-séquence commune de M[1..i] et T[1..j]

L(0,i) = L(i,0) = 0 pour tout i

Si M[i]=T[j] alors L(i,j) = 1+L(i-1,j-1)
Sinon L(i,j) = max(L(i-1,j),L(i,j-1))

Principe de la programmation dynamique (ne cherchez pas d'explication rationnelle à ce nom) :

- équation de récurrence qui résout un problème à partir de sa solution sur des problèmes plus petits
- transformation en algorithme itératif

Variante 1 : Plus longue sous-séquence commune

```
Pour i de 0 à m :      l[i,0] = 0
Pour j de 0 à t :      l[0,j] = 0
Pour i de 1 à m, j de 1 à t :
    Si M[i] = T[j] :
        l[i,j] = 1 + l(i-1,j-1)
        chemin = i et j
    Sinon, si l[i-1,j] < l[i,j-1] :
        l[i,j] = l[i,j-1]
        chemin = j
    Sinon :
        l[i,j] = l[i-1,j]
        chemin = i
```

Variante 1 : Plus longue sous-séquence commune

Retour :

```
i = m, j = t
Tant que i > 0 et j > 0 :
    Si chemin = i et j :
        Afficher M[i]
        i = i - 1
        j = j - 1
    Si chemin = i
        i = i - 1
    Si chemin = j
        j = j - 1
```

Exercice : calculer la complexité de l'algorithme

Variante 2 : Alignement global de deux séquences

Présupposé : les deux séquences sont homologues sur toutes leur longueur. En conséquence toutes les différences doivent être expliquées par des évènements évolutifs.

On doit donc aligner les deux séquences : déterminer les sites homologues et les insertions/délétions

Par exemple,
`AGACTAGTTAC`
et
`CGAGACGT`

A	G	A	C	T	A	G	T	T	A	C
C	G	A				G	A	C	G	T

Variante 2 : Alignement global de deux séquences

Score d'un alignement :
Correspondance : 2
Absence de correspondance : -1
Insertion/délétion : -1

Ici, score total -2

Meilleur score?

A	G	A	C	T	A	G	T	T	A	C
C	G	A				G	A	C	G	T

Variante 2 : Alignement global de deux séquences

Meilleur score d'un alignement entre M et T
(Algorithme de Needleman-Wunch)

a(i,j) = meilleur score d'un alignement entre M[1..i] et T[1..j]

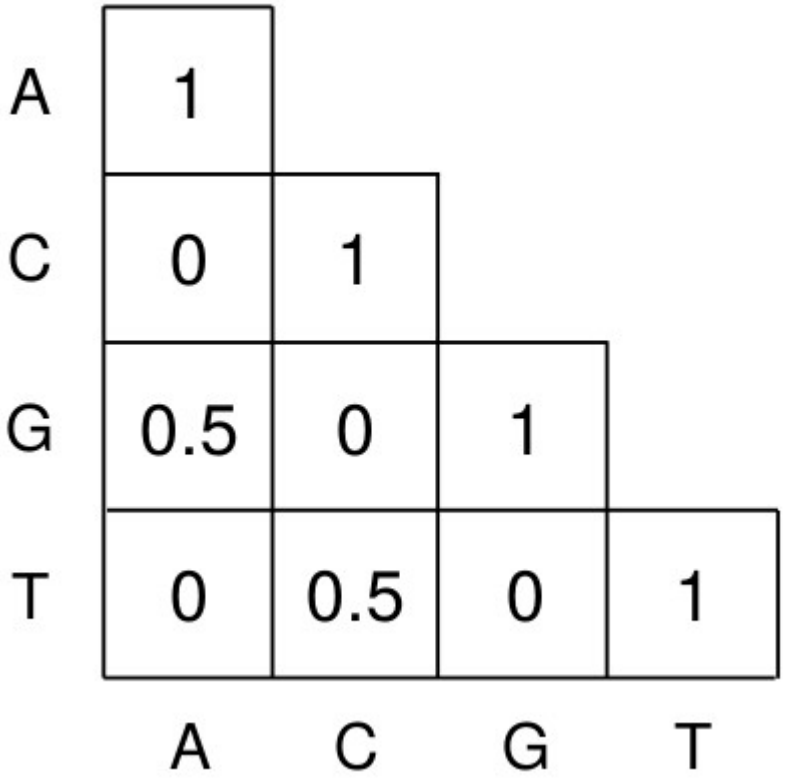
a(i,j) = max(a(i-1,j-1) + score(m[i],t[j]),
a(i,j-1) + score(indel)
a(i-1,j) + score(indel))

a(0,j) = j * score(indel)
a(i,0) = i * score(indel)

A	G	A	C	T	A	G	T	T	A	C
C	G	A				G	A	C	G	T

Variante 2 : Alignement global de deux séquences

a(i,j) = max(a(i-1,j-1) + score(m[i],t[j]),
a(i,j-1) + score(indel)
a(i-1,j) + score(indel))



score(m[i],t[j]) est donné par une matrice de substitutions

Variante 3 : Alignement local de deux séquences

On ne présuppose plus que les séquences sont homologues sur toute leur longueur. Les sites sont supposés homologues entre le premier et le dernier site alignés. Au-delà et en-deçà, les insertions/délétions sont gratuites comme dans la plus longue sous-séquence commune.

A	G	A	C	T	A	G	T	T	A	C
C	G	A				G	A	C	G	T

Variante 3 : Alignement local de deux séquences

Meilleur score d'un alignement LOCAL entre M et T
(algorithme de Smith-Waterman)

a(i,j) = max(a(i-1,j-1) + score(m[i],t[j]),
a(i,j-1) + score(indel),
a(i-1,j) + score(indel),
0)

a(0,j) = 0
a(i,0) = 0

A	G	A	C	T	A	G	T	T	A	C
C	G	A				G	A	C	G	T

Alignements

- Utilisation de la programmation dynamique:
    Une équation de récurrence => Un algorithme
    Remplissage d'une matrice + chemin du retour

- Complexité de la comparaison de deux séquence O(nm)
    Impraticable pour des données génomiques

- Principe théoriquement extensible à 3 séquences O(mnl), ou à k séquences $O(n^k)$

- Baisse de la complexité possible en supposant qu'on ne cherche que les solutions meilleures qu'un certain score (comme le problème de départ : recherche de mots approchés à une distance au plus e).

Alignements

- Ces algorithmes font partie de la boîte à outil standard de la bioinformatique. Ils sont disponibles dans plusieurs logiciels, et utilisés pour des comparaisons fines entre séquences pas trop longues

- Ils sont aussi utilisés comme brique dans des outils de recherche plus rapides

- Le principe de la programmation dynamique se retrouve souvent dans d'autres algorithmes, recherches de chemin dans les graphes, parcours d'arbres phylogénétiques,...

Plan du cours

I/ Recherche de séquences exactes

II/ Recherche de séquences approchées, alignement

III/ Méthodes et logiciels actuels

BLAST (1994)

Recherche de mots exacts (ancres) avec tables de suffixes

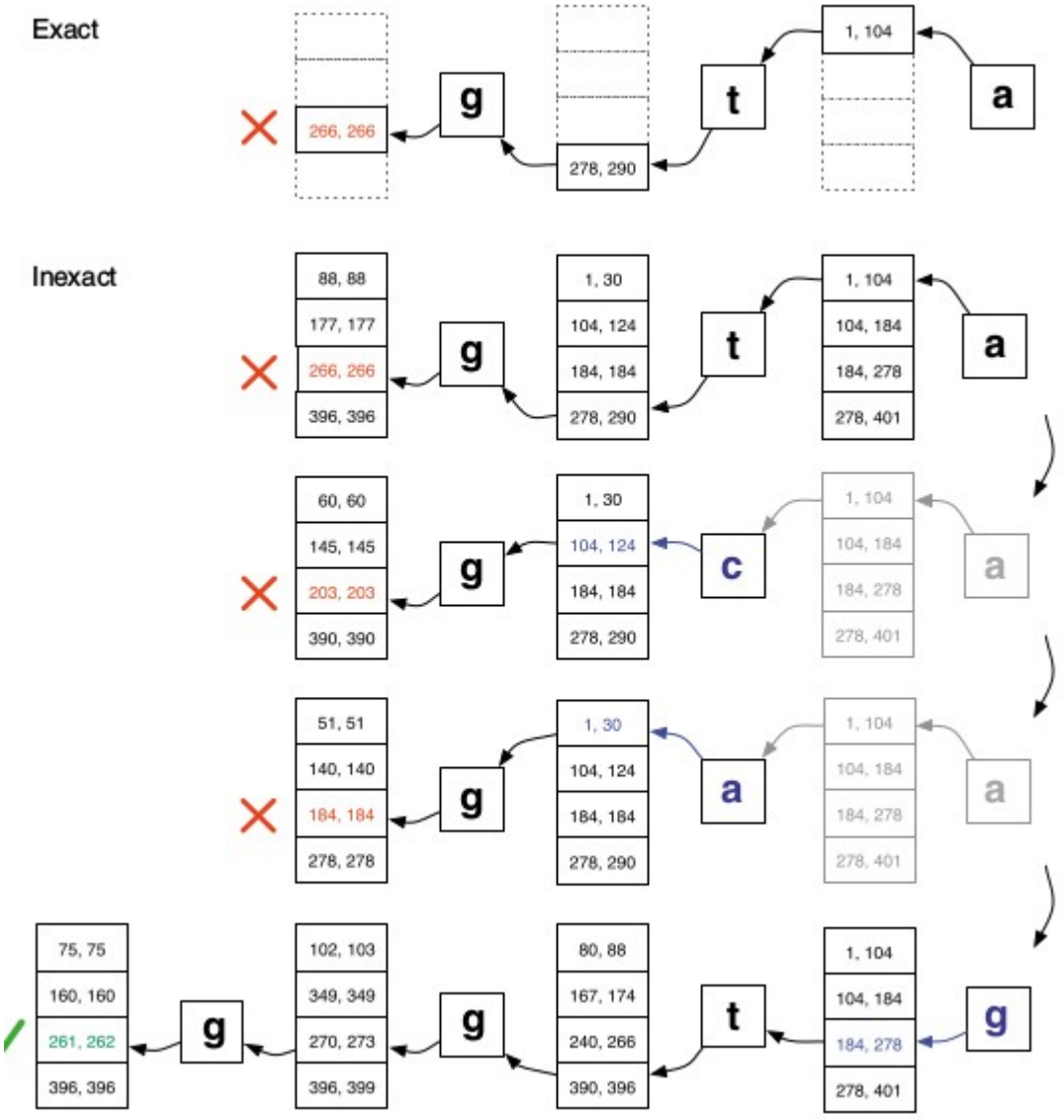
Extension avec Needleman-Wunch tant que le score ne diminue pas

Calcul d’une e-value

Bowtie:

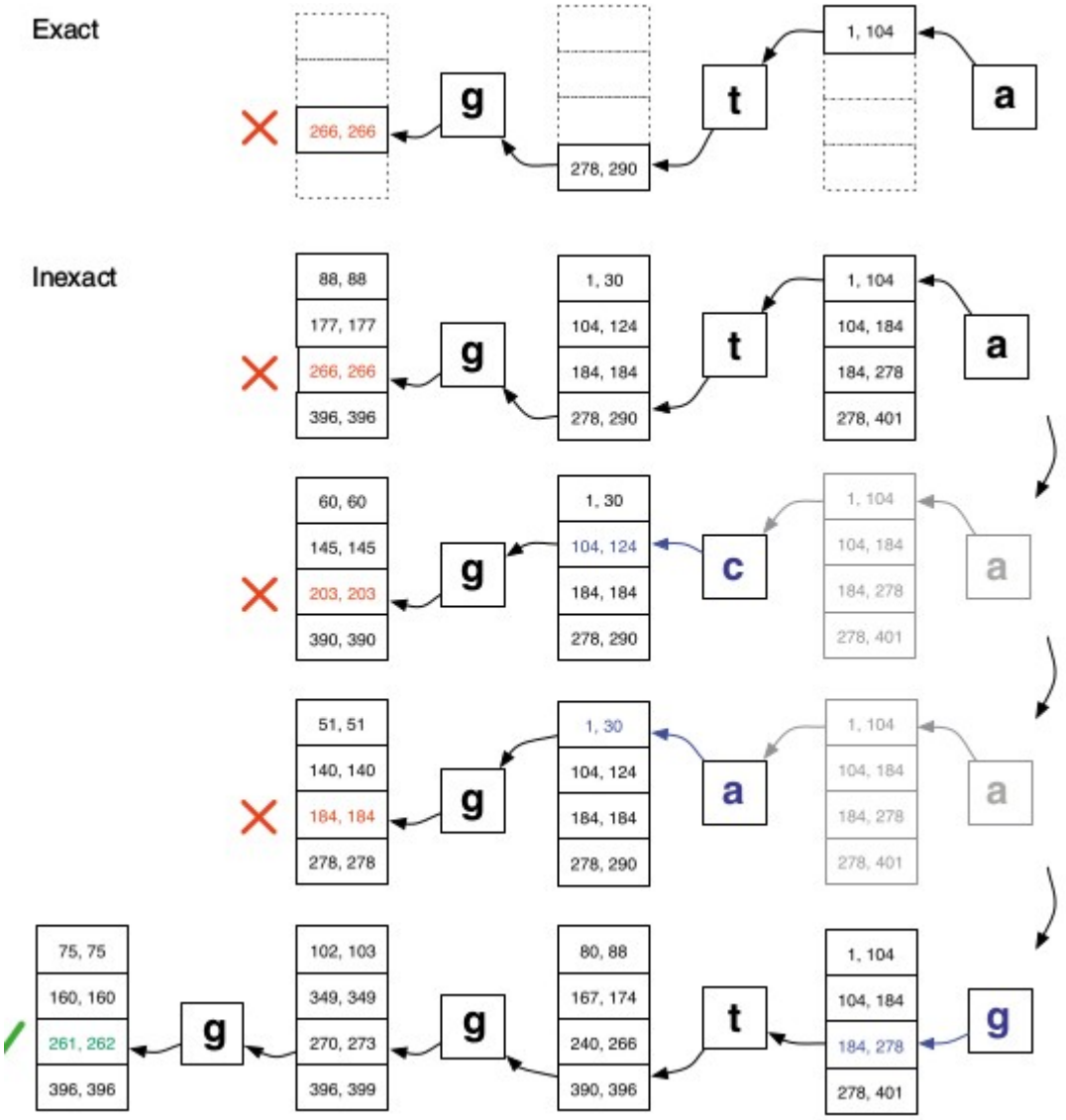
Recherche exacte

En cas d’incuccès, tentatives de recherche dans le voisinage des morceaux alignés



BWA

Même principe mais autorise les indels



Plusieurs logiciels parus depuis (BLAT, USEARCH, RAPSearch2, PAUDA….)

Parmi les plus marquants:

Diamond, 2015:

- double index : le mot et le texte sont indexés par un index du type BWT
- utilisation d'un voisinage partiel : les graines, de taille typique 11-16, sont transformées avec des filtres:

(a)
```
111101011101111
111011001100101111
11110010010100010010111
11110010100001001001011
```

(b)
```
Reference      SLWAKKRTVDGQPKWLPLVAHLVDASNVSRMLFNQWLSD
Spaced seed      111101011101111
Query          FWAKKRTNDGQQKWLPLTQHLEDASNVSR
```

- alphabets simplifiés : 4 ou 20 lettres ne sont pas nécessairement optimal pour la recherche de séquence. On peut adapter l'alphabet, ici 11 lettres
[KREDQN] [C] [G] [H] [ILV] [M] [F] [Y] [W] [P] [STA]

Kallisto, Sailfish, Salmon (2015-2016)

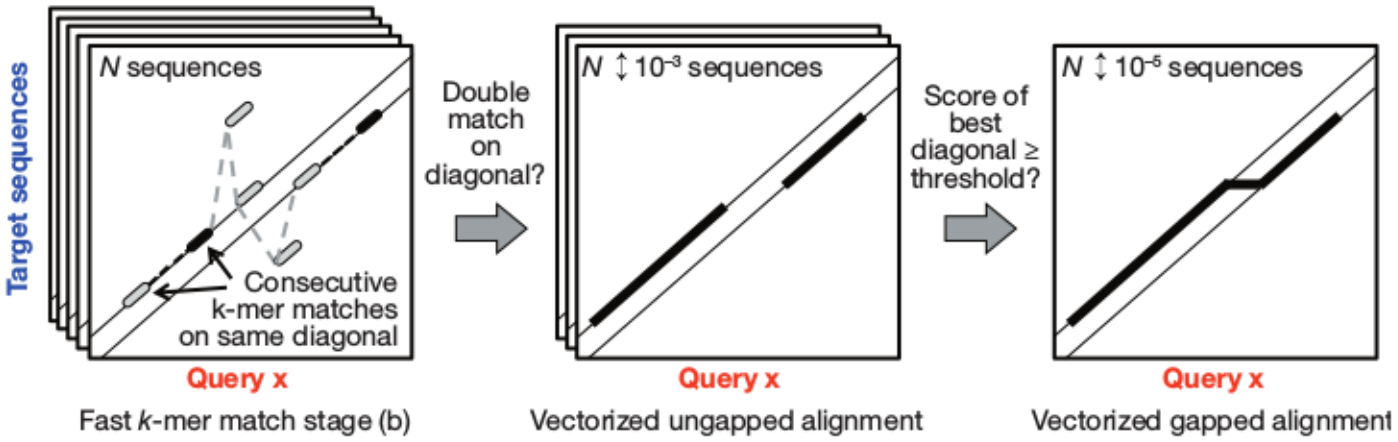
“quasi-mapping”, “pseudo-alignment”

Indexer la présence de “k-mer”, mots courts de taille fixe
Recherche de k-mer en temps constant avec table de hachage

Décider de la présence d’un mot approché sur des bases statistiques de présences de k-mer plutôt que sur un alignement.

MMseq2, 2017

k-mers avec 600 à 60000 voisins générés au hasard, taille typique 7
Regroupement des k-mers qui ont une correspondance dans le texte avec un principe de “diagonale”



Exercices

1/ Ecrire un algorithme efficace qui calcule les nombres selon les formules:

- f(1) = 1 et f(n) = n * f(n-1)
- f(n,0) = 1, f(n,n) = 1, f(i,j) = f(i-1,j-1) + f(i-1,j) pour i <= j

2/ Ecrire un algorithme qui énumère le voisinage d'un mot (tous les mots à distance e). Donnez une estimation du nombre de mots à distance e, en fonction de la taille du mot, de la taille de l'alphabet, et de e

Exam 2018

Algorithme pour générer le voisinage d'un mot

Le but de cet exercice est d'écrire un algorithme qui prenne en entrée une séquence S de taille n dans un alphabet nucléotidique $\{A, C, G, T\}$, et qui renvoie la liste de tous les mots obtenus à partir de S en appliquant au plus un nombre e de substitutions ou délétions de caractères. Cette liste est appelée le *voisinage* de S à distance au plus e.

Notez que différentes mutations peuvent aboutir à une même séquence. Par exemple si $S = TT$, la délétion du premier ou du deuxième caractère résulte en la séquence T. Pour cet exercice on suppose que cette même séquence est alors comptée plusieurs fois dans le voisinage.

1. Rappelez l'utilité de la génération d'un voisinage dans le contexte de la recherche d'un mot M dans un texte T, si on veut détecter les occurrences approchées de M dans T.
2. Donner la liste complète du voisinage pour $e = 1$ et $S = CGT$.

Exam 2018

3. Combien y a-t-il de séquences dans un voisinage pour $e = 1$?
4. On définit $V(i, e)$ comme le voisinage de $S[1..i]$ à distance exactement e. Par convention si $i = 0$ alors $S[1..i]$ est une séquence vide. Si $1 \leq i \leq n$, alors $S[1..i]$ contient les i premiers caractères de S. Que vaut $V(i, 0)$? Que vaut $V(0, e)$? Que vaut $V(i, e)$ si $i < e$?
5. Expliquez comment on peut construire $V(i, e)$ à partir de $V(i - 1, e)$ et $V(i - 1, e - 1)$.
6. À partir de l'initialisation de la question 4 et de la récurrence définie à la question 5, écrivez un algorithme qui génère $V(i, e)$.
7. En pratique le voisinage complet, même en réduisant la redondance de la liste, est de taille trop importante pour le générer en entier. Donnez brièvement un exemple de solution pour réduire cette taille employée par les logiciels de recherche de similarité dans les séquences.

Exam 2020

1. Quelles sont les contraintes sous lesquelles on conçoit habituellement des algorithmes pour la bioinformatique ?
   - Le réalisme biologique des processus modélisés
   - La complexité des problèmes algorithmiques
   - Les capacités des ordinateurs
   - Les ressources naturelles
   - Le carbone rejeté dans l'atmosphère
   - Le temps d'une vie humaine
   - La taille et la quantité des données biologiques disponibles

2. Que contient une table des suffixes construite sur un mot de n lettres ?
   - n lettres
   - n^2 lettres
   - n nombres
   - n mots
   - n suffixes
   - n indices

3. Que contient la transformée de Burrows-Wheeler construite sur un mot de n lettres ?
   - n lettres
   - n^2 lettres
   - n nombres
   - n mots
   - n suffixes
   - n indices

4. Quelle est la table des suffixes associée au mot EXAM ?
   - AEMX
   - 3 1 4 2
   - EXAM XAM AM M
   - $AEMX
   - MX$AE

5. Quelle est la transformée de Burrows-Wheeler associée au mot EXAM ?
   - AEMX
   - 3 1 4 2
   - EXAM XAM AM M
   - $AEMX
   - MX$AE

6. Quel est le mot associé à la transformée de Burrows-Wheeler BBB$AAAA ?
   - AAAA$BBB
   - ABAABAB
   - $AAAABBB
   - ABABABA$
   - ABAABAB$

7. Combien y a-t-il de préfixes dans un mot de taille n ?
   - n
   - n+1
   - n^2
   - n*(n+1)/2
   - 2n
   - 2^n

8. Vous venez de séquencer un nouveau gène, et vous voulez connaître les homologues connus dans une base de données. Quel usage faites-vous des algorithmes de recherche exacte d'un mot dans un texte ?

- Aucun, les homologues n'auront pas exactement la même séquence
- Aucun, j'utilise un logiciel de recherche de séquence similaire
- Je recherche des correspondances exactes entre des séquences de la base et des morceaux de mon gène
- Je génère des séquences similaires à mon gène et je recherche ces séquences dans la base.
- J'utilise un logiciel de recherche de séquence similaire qui fera une recherche exacte sur des parties de mon gène, ou sur des séquences similaires générées automatiquement.

9. Que contient l[m,t] à la fin de cet algorithme ?

```
Entrée : Deux séquences M et T, de tailles respectives m et t
Pour i de 0 à m :
        l[i,0] = 0
Pour j de 0 à t :
        l[0,j] = 0
Pour i de 1 à m, j de 1 à t :
        Si M[i] = T[j] :
                l[i,j] = 1 + l[i-1,j-1]
        Sinon, si l[i-1,j] < l[i,j-1] :
                l[i,j] = 1 + l[i,j-1]
        Sinon :
                l[i,j] = 1 + l[i-1,j]
```

- La plus longue sous-séquence commune
- La taille de la plus longue sous-séquence commune
- La taille de la plus courte séquence, dont M et T sont des sous-séquences
- m+t
- (m+t) - la taille de la plus longue sous-séquence commune

Patchwork

10. Quelle est la complexité de l'algorithme de la question 9 ?
    - O(m+t)
    - O(m*t)
    - O(2^(m+t))
    - O(m*log(t))
    - O(l[m,t])

11. Pour un mot M de taille m, un e-voisin de M est un mot obtenu à partir de M en appliquant au plus e substitutions. Combien AAAA a-t-il de 1-voisins dans un alphabet nucléotidique ? Combien un mot de m lettres a-t-il de 1-voisins dans un alphabet nucléotidique ?

4. De quelle séquence ERMESRAFTON$ est-elle la transformée de Burrows-Wheeler ?
5. Si on cherche une occurrence d'une séquence M dans une séquence T, applique-t-on la transformée à M, à T, ou aux deux ?
6. Si on a trouvé une sous-séquence M' de T très similaire à M, peut-on conclure que M et T sont très homologues ? Pourquoi ?

Exam 2021

Examen sur la comparaison de séquences

1. A quoi sert l'algorithme suivant ?

```
Entrees :
    - une liste L de nombres
    - un nombre x
i = 0
tant que i < len(L) et x > L[i]:
    i = i + 1
Retourner L[:i]+[x]+L[i:]
```

(concaténation de L[1 à i-1], de x, et de L[i à la fin de L] en une seule liste)

2. Quelle est sa complexité ?
3. Quelle est la propriété attendue en sortie, si L est triée par ordre croissant en entrée ?
4. En supposant que L est triée par ordre croissant en entrée, proposez un algorithme de meilleure complexité pour la même tâche.
5. Donnez un algorithme qui utilise cette fonction pour trier par ordre croissant une liste non triée.
6. Dites pourquoi le tri est une étape cruciale d'une méthode de recherche de séquences similaires à une séquence en entrée dans des bases de données génomiques.
7. Si T est un texte, proposez un algorithme qui calcule la plus grande répétition dans T, c'est à dire les deux indices i et j tels que T[i,i+k] == T[j,j+k] et k est maximum sur tous les couples i,j possibles. Vous pouvez supposer que vous avez accès à une table des suffixes de T.