BRICAUD Pierre CSI2

MACÉ Lucien

GROSSMANN—LE MAUGUEN Arthur

Rapport de projet Python 2023

Thématique : Alignement de séquences biologiques et de protéines

Introduction

Notions impliquées

On étudie ici les alignements de séquences biologiques et de protéines, utiles dans les études de virologie et de génétique.

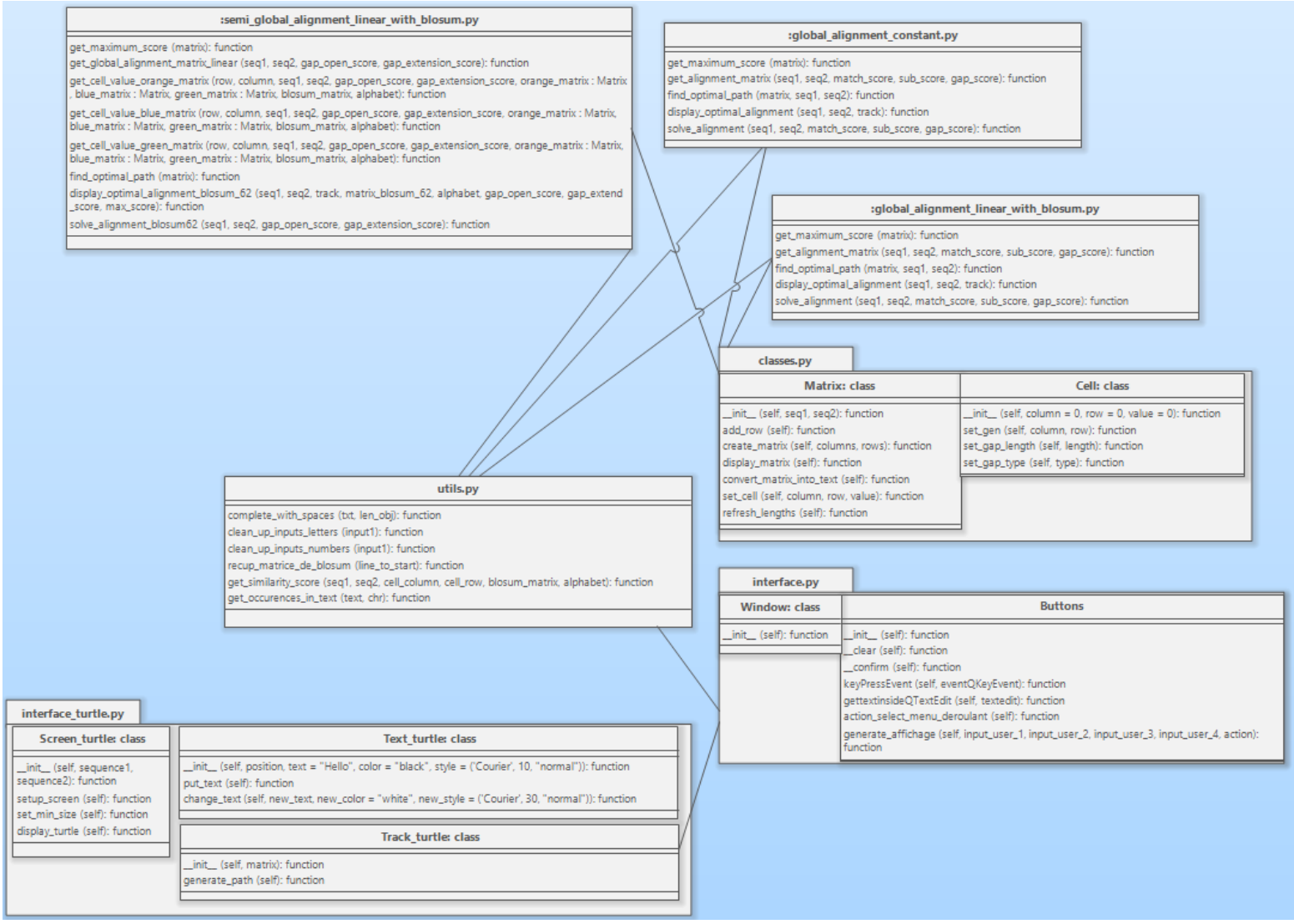
Lexique

Matrice – de l'anglais 'Matrix'

Cellule – de l'anglais 'Cellule'

Plan

Nous allons présenter un schéma UML ('Unified Modeling Language', 'Langage de Modélisation Unifié' en français) afin pouvoir visualiser l'organisation générale de notre code.



Déroulement du projet

Alignement global optimal du Pydéfis

- Principe

Nous avons deux séquences d'acides aminés (une suite de lettres parmi 'A', 'G', 'C', 'T') à aligner avec une série de transformations, au moindre coût. Parmi les différentes transformations, nous avons :

- L'identité : Les deux éléments sont identiques. C'est la transformation avec le score le plus élevé.

- La substitution : Nous modifions le premier élément pour obtenir le second. Cela a un coût (i.e. un score négatif)

- L'insertion : Nous insérons un élément dans la deuxième séquence. Cela a un coût plus élevé, mais un décalage est créé, ce qui peut révéler des avantages par la suite, comme une série d'identités.

- La substitution : Nous supprimons l'élément dans la première séquence. Cette opération a les mêmes conséquences qu'une substitution, mais le décalage crée est dans l'autre sens.

- Démarche

Nous créons une matrice de transformations, qui représente tous les chemins possibles, pour passer de la séquence 1, à la séquence 2. Nous commençons en haut à gauche, et nous cherchons le score maximal entre une substitution (identité, si applicable), une insertion, ou une suppression. Une fois la matrice créée, il faut retracer le trajet, en partant de la fin, pour trouver le chemin optimal.

Alignement global avec similarité (matrice de BLOSUM)

- Matrice de similarité

A la place d'avoir un score constant pour une identité ou une substitution, nous avons un tableau, (ici, une matrice), qui nous donne le score de l'opération pour chaque combinaison impliquant chaque protéine.

- Affichage façon EMBOSS Needle

Cet affichage est très utilisé pour représenter graphiquement l'alignement, et communique également des informations sur le chemin utilisé. Cet affichage peut se révéler pratique pour identifier des similarités entre des protéines, ou pour observer des mutations génétiques.

Modèles de gaps

- Gap linéaire

Le modèle du gap linéaire, le plus simple, consiste à pénaliser les gaps (insertions et suppressions) de la même manière, peu importe la longueur du gap.

- Gap affine

La modèle du gap affine, plus complexe, mais plus représentatif, prend en compte la longueur du gap. Il y a ce que nous appelons un gap d'ouverture, généralement très couteux, et un gap d'extension, qui suit un autre gap. Un gap d'extension a un coût souvent très faible. Ce modèle favorise davantage une série de gaps, dans l'éventualité où il y a une identité d'une partie de séquence, mais sur un intervalle décalé de plusieurs protéines.

Résolution

Classes

- Classe ‘Matrix’ – Matrice de transformation

Cette classe à plusieurs attributs, qui sont créés dans le ‘__init__’ :

- ‘seq1’ : Première séquence de l’alignement
- ‘seq2’ : Deuxième séquence de l’alignement
- ‘value’ : Valeur de la matrice, son type est une liste de listes, avec les
- Colonnes dans les lignes. Exemple : ‘[[1, 5, 9],[2, 7, 3],[4, 6, 8]]’ est la matrice suivante : $\begin{matrix} 1 & 5 & 9 \\ 2 & 7 & 3 \\ 4 & 6 & 8 \end{matrix}$

Visuellement, si nous affichons chaque élément de ‘value’ les uns après les autres, nous obtenons ceci :

[1, 5, 9]

[2, 7, 3]

[4, 6, 8]

Cette image est assez parlante, ce qui explique ce choix.

- ‘rows’ : le nombre de lignes de la matrice
- ‘columns’ : le nombre de colonnes de la matrice

Une matrice appartenant à $M_{i,j}(K)$ se traduit par ‘rows’ = i, et ‘columns’ = j.

La classe ‘Matrix’ a aussi plusieurs méthodes, que voici :

- ‘add_row(rows)’, qui permet d’ajouter le nombre de lignes contenu dans la variable ‘rows’, ce qui se traduit par l’ajout de listes vides dans l’attribut ‘value’
- ‘create_matrix(columns, rows)’, qui crée une matrice $M_{rows,columns}(K)$, remplie de valeurs ‘None’, car nous n’avons pas encore les valeurs des cellules. Elles seront déterminées par la suite, mais le fait de faire ce procédé va nous permettre de remplacer chaque valeur, sans avoir d’erreurs d’indice.
- ‘convert_matrix_into_text’, similaire à la fonction ‘__str__’, qui convertit une matrice en texte, pour pouvoir l’afficher.
- ‘display_matrix’, qui affiche la matrice, à l’aide de la fonction précédente.
- ‘set_cell(column, row, value)’, qui attribuer la valeur de l’attribut ‘value’, aux indices [row][column], la cellule (de classe ‘Cell’) de coordonnées (column, row), qui prend pour valeur, la valeur donnée dans la variable ‘value’.
- ‘refresh_lengths’, qui actualise les attributs de longueur de largeur de la matrice, en fonction de la longueur des deux listes.

- Classe ‘Cell’ – Cellule de la matrice de transformation

Cette classe a plusieurs attributs :

- ‘column’ : coordonnée x (ou n° de colonne) de la cellule
- ‘row’ coordonnée y (ou n° de ligne) de la cellule

Nous notons que ces deux valeurs commencent à 0, elles vont donc jusqu’à leurs taille – 1.

- ‘value’ : valeur de la cellule, de type ‘int’ ou ‘float’
- ‘gen_coords’ : coordonnées de la cellule génératrice de cette cellule. La cellule génératrice sera très utile pour retrouver le chemin de l’alignement. Cela prend un peu plus de mémoire, mais réduit énormément le nombre de calculs.

La classe ‘Cell’ a une seule méthode exceptée l’initialisation :

- ‘set_gen(column, row)’ qui fixe les coordonnées de sa cellule génératrice, sous forme de tuples de taille deux (un couple de coordonnées).

Utilitaires – fichier ‘utils.py’

- Complétion avec des espaces

La fonction ‘complete_with_spaces(txt, len_obj)’ renvoie un texte de longueur égale à la variable donnée ‘len_obj’, et en fonction de la longueur du texte {txt}, nous ajoutons la différence de longueur, le nombre d’espaces à gauche.

Exemples :

‘complete_with_spaces(‘Hello’, 10)’ renvoie ‘ Hello’

Analogie avec la fonction commune, ‘complete_with_zeros(txt, len_obj)’, utile pour uniformiser la table ASCII.

Exemple:

‘complete_with_zeros(42, 3)’ renvoie ‘042’

- Filtres de l’entrée texte

Les fonctions ‘clean_up_inputs_letters’ et ‘clean_up_inputs_letters’ permettent respectivement de retirer les caractères qui ne sont pas de protéines et mettre le résultat en majuscule, et de retirer toutes les lettres quand il faut entrer un nombre, ainsi que de remplacer les virgules par des points.

Exemples:

‘clean_up_inputs_letters("Input1234TesT")’ renvoie "INPTTEST"

‘clean_up_inputs_numbers("Input,1234,Test")’ renvoie ".1234."

- Récupération de la matrice de BLOSUM à partir d’un fichier

La fonction ‘get_blosum_matrix(line_to_start)’ renvoie une liste de liste avec les scores de chaque transformation, et la liste des lettres représentant les protéines.

Exemples:

‘recup_matrice_de_blosum(7)’ renvoie ([[4, -1, -2, -2, 0, -1, -1, 0, -2, -1, -1, -1, -1, -2, -1, 1, 0, -3, -2, 0, -2, -1, 0, -4],[...], [-4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, 1]], ['A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V', 'B', 'Z', 'X', '*'])

- Calcul du score de similarité

La fonction ‘get_similarity_score(seq1, seq2, cell_column, cell_row, blosum_matrix, alphabet)’ donne le score pour l’opération entre les deux lettres présentes aux positions {cell_column, cell_row) de la matrice. Ce score est invoqué depuis la matrice de BLOSUM.

- Calcul d’occurrences d’un caractère dans une chaîne

La fonction ‘get_occurences_in_text (text, chr)’ renvoie le nombre de caractères ‘chr’ présents dans le texte ‘txt’.

Programmes d’alignements

- Alignement global, gap linéaire

Le programme ‘global_alignment_constant’ effectue l’alignement global des deux séquences avec une pénalité de gap linéaire.

- Alignement global avec BLOSUM, gap affine

Le programme ‘global_alignment_linear_with_blosum’ effectue l’alignement global des deux séquences avec une pénalité de gap affine, et avec les scores des transformations présents dans la matrice de BLOSUM.

- Alignement semi-global avec BLOSUM, gap affine

Le programme ‘semi_global_alignment_linear_with_blosum’ effectue l’alignement semi-global des deux séquences avec une pénalité de gap affine, et avec les scores des transformations présents dans la matrice de BLOSUM. Un alignement semi-global nullifie les pénalités des gaps s’ils sont au début ou à la fin de l’alignement.

Fonctions

- Génération de la matrice de transformation – Classique

La fonction ‘get_alignment_matrix(seq1, seq2, match_score, sub_score, gap_score)’ crée matrice de l’alignement de {seq1} sur {seq2} avec les scores renseignés, et une pénalité de gap linéaire {gap_score}

- Génération de la matrice de transformation – BLOSUM

La fonction ‘get_alignment_matrix_linear(seq1, seq2, gap_open_score, gap_extension_score)’ crée matrice de l’alignement de {seq1} sur {seq2} avec les scores renseignés, et une pénalité de gap affine qui vérifie la relation :

Score = {gap_open_score} * (« longueur du gap » - 1) * « gap_extension_score »

Pour créer cette matrice, nous faisons une récursivité en partant du bas, car des valeurs se définissent par le score maximal de valeurs, qui ne sont elles-mêmes pas définies. Le cas de base est la case tout en haut à gauche, qui a pour valeur 0. Il nous faut trois matrices : celle que nous renverrons, puis une pour les alignements se terminant par une insertion, et une pour les alignements se terminant par une suppression.

- Calculer le score maximal

Pour la plupart des cas, calculer le score maximal est trivial, puisqu’il s’agit de la valeur de la dernière case de la matrice, celle qui est tout en bas à droite.

- Trouver le chemin optimal

Pour trouver le chemin optimal, il faut parcourir le chemin à l’envers, en regardant à chaque la case qui a généré la suivante. C’est principalement là que la classe ‘Cell’ va montrer son potentiel.

La fonction ‘find_optimal_path(matrix)’ parcoure la matrice à l’envers, en passant de case en case. Les coordonnées de la cellule précédente sont trouvées dans la cellule génératrice. Nous obtenons donc une liste de tuples de taille deux (couples de coordonnées), qui représente les coordonnées des points, dans l’ordre.

- Afficher le chemin optimal – Classique

La fonction ‘display_optimal_alignment(seq1, seq2, track)’ se sert du chemin trouvé juste avant pour afficher l’alignement. Nous parcourons les chaînes des caractères, et nous associons un caractère (‘|’, ‘v’, ou bien ‘ ’) pour chaque transformation.

- Afficher le chemin optimal – BLOSUM

Nous adoptons ici l’affiche EMBOSS Needle. La fonction ‘display_optimal_alignment_blosum_62(seq1, seq2, track, matrix_blosum,

alphabet, gap_open_score, gap_extension_score, max_score)' affiche le résultat de l'alignement, avec tout d'abord les entrées de l'alignement (scores, séquences, matrice de BLOSUM), puis l'alignement. Nous divisons celui-ci en tranches de 50 pour qu'il rentre dans une page ou dans un document.

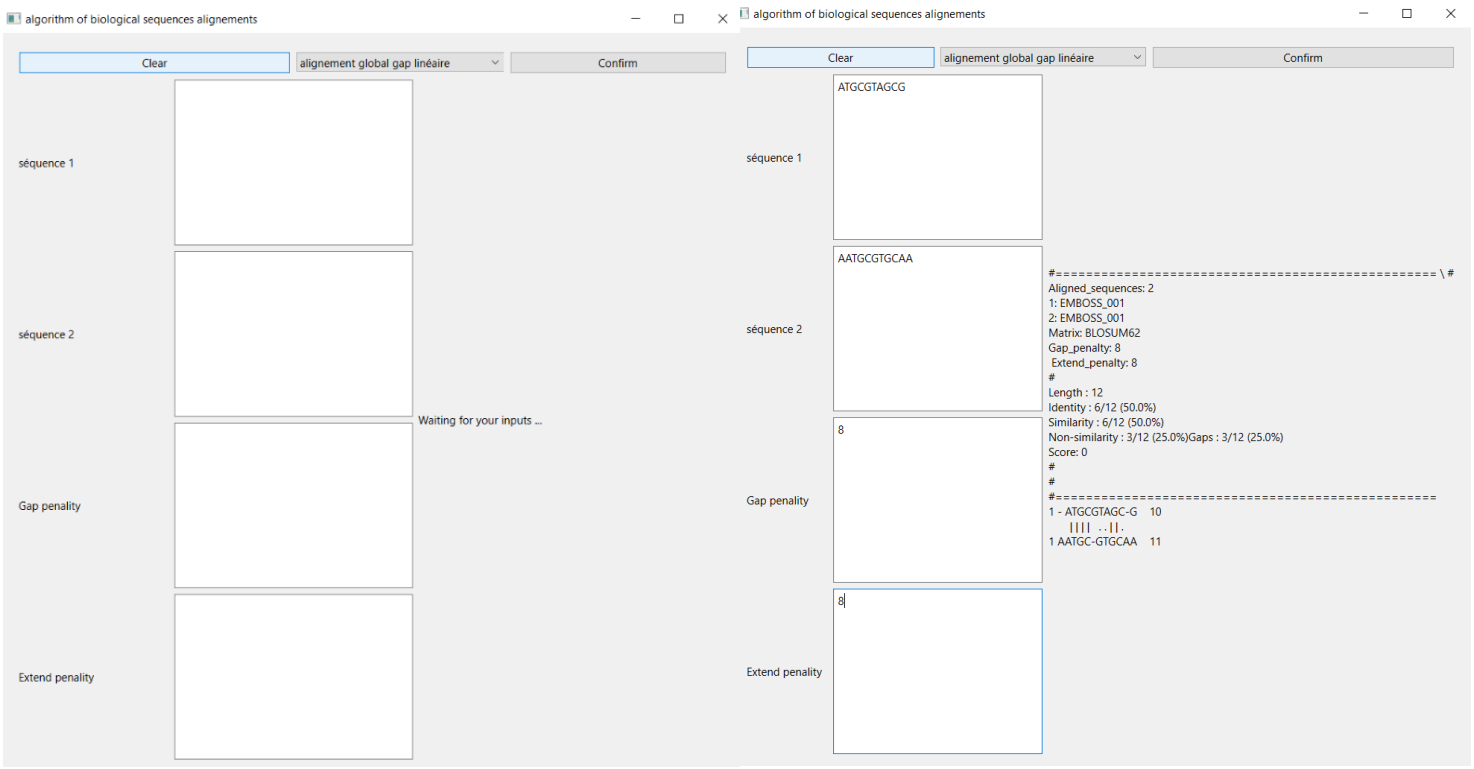
- Résoudre l'alignement

Cette fonction, 'solve_alignement' / 'solve_alignement_blosum_62', regroupe toutes les fonctions, et les exécute dans le bon ordre.

Imagination – appropriation du sujet – pistes d'amélioration

Interface

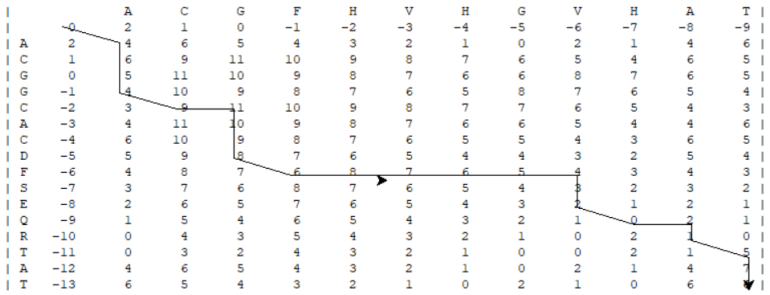
Nous avons souhaité pouvoir rentrer les valeurs des deux séquences ainsi que les valeurs pour la pénalité d'ouverture et d'extension. L'utilisateur peut aussi choisir quelle méthode d'alignement il souhaite appliquer dans un menu déroulant. Lorsque le bouton 'confirm' est appuyé, les valeurs du menu déroulant et des quatre cases remplies. Une fonction génère l'affichage EMBOSS NEEDLE d'après le retour de la méthode choisie puis une autre fenêtre s'ouvre avec l'interface turtle dans laquelle se trouve la matrice et le chemin le plus avantageux mis en évidence.



Interface turtle

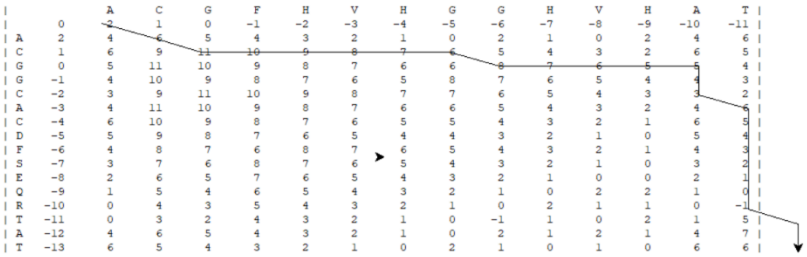
Nous avons choisi turtle pour afficher le meilleur chemin à prendre dans la matrice.

En premier lieu nous avons pensé à tracer le chemin grâce à la fonctionnalité des objets turtle mais comme nous affichons une matrice ayant les séquences marquées sur les côtés, il est impossible de trouver le point de départ du chemin pour n'importe quelle séquence donnée.



Ici nous avons trouvé à tâtons le point de départ le chemin est donc correct

Il y a une flèche à l'emplacement (0,0) et nous observons qu'il n'est pas au centre de la matrice.



Mais en changeant la longueur d'une séquence, le décalage n'est pas le même et

Nous nous retrouvons avec un chemin décalé.

la flèche au centre de la matrice ne s'est pas décalé de la même façon que la matrice s'est agrandi.

Pistes d'amélioration

- Pour la partie turtle une solution au problème serait d'écrire dans une couleur différente le contenu des cellules par lequel passe le chemin optimal.
- Lors de la lecture de la matrice de BLOSUM62 au lieu de renvoyer deux listes (une pour l'alphabet et une liste de liste pour le contenu de la matrice) il serait possible de créer un dictionnaire de dictionnaire afin de simplifier la récupération de la valeur d'une cellule individuelle par la suite.

Blosum= {'A' :{'A'=1,'B'=2, ...}, 'B' :...}

Sources

Lien	Utilisation	Date de dernière utilisation
https://www.youtube.com/@Projetinformatique2023-sg2np/playlists	Compréhension des différents algorithmes	06/06/2023
https://www.ebi.ac.uk/Tools/psa/emboss_needle/	Tests	05/06/2023
https://www.ebi.ac.uk/Tools/psa/emboss_water/	Tests	05/06/2023
https://pydefis.callicode.fr/defis/Genome/txt	Tests	05/06/2023