



MiNi Info I

Rapport Projet C

GOULET Lindsay
BAUMANN Ambre
M1 BIBS 2021-2022

Table des matières

1	Module utiles.c	4
2	Module 1 : Recherche de la séquence codante de taille maximale	5
3	Module 2 : Transcription	6
4	Module 3 : Traduction	6
5	Module 4 : Calcul du score d'identité entre deux séquences	7
6	Module 5 : Calcul du score de similarité de polarité entre deux séquences protéiques	7
7	Module 6 : Séquence consensus issue d'un résultat d'alignement multiple	8
8	Module 7 : Recherche de la plus grande sous-chaîne de polarité commune à 2 séquences	9
9	Fonction principale du logiciel : main.c	10
10	Comparaison des résultats de notre logiciel	11
10.1	Module 1	11
10.2	Module 2	11
10.3	Module 3	12
10.4	Module 4	12
10.5	Module 5	13
10.6	Module 6	13
10.7	Module 7	13
	Références	15

Introduction

Le but de ce projet est de créer un logiciel d'analyse de séquences génomiques et protéiques eucaryotes. Il est composé de 7 modules qui sont les suivants :

1. Recherche de la séquence codante de taille maximale.
2. Transcription d'une séquence ADN en séquence ARN.
3. Traduction d'une séquence codante en séquence protéique.
4. Calcul du score d'identité entre deux séquences.
5. Calcul du score de similarité de polarité entre deux séquences protéiques.
6. Recherche d'une séquence consensus à partir d'un alignement multiple.
7. Recherche de la plus grande sous-chaîne de polarité commune à 2 séquences protéiques.

Le logiciel accueille l'utilisateur sur un menu et lui demande le module qu'il souhaite utiliser. Lorsque le logiciel demande un fichier contenant une séquence à l'utilisateur, celui-ci doit fournir un fichier FASTA. Un fichier FASTA est composé :

- d'une première ligne commençant par '>' et qui décrit la séquence (identifiant puis commentaire par exemple) ;
- d'une ou plusieurs lignes contenant la séquence d'acides nucléiques ou aminés (maximum 80 caractères par ligne).

Dans le fichier README.md des instructions d'utilisations du logiciel sont écrites.

Nous avons créé un fichier nommé *utiles.c* dans lequel nous avons placé toutes les fonctions utiles à au moins deux modules différents. Nous y avons placé les fonctions/procédures suivantes :

- void `get_path_from_user(char* path_input)` qui stocke dans *path_input* un nom de fichier saisi par l'utilisateur.
- void `get_module_number_from_user(char* module_number)` qui affiche dans le terminal les différents choix qui s'offrent à l'utilisateur et qui permet à l'utilisateur de choisir l'une des options possibles.
- void `supp_premiere_ligne(const char* path_input)` qui supprime la ligne d'informations du fichier FASTA si celle-ci est présente.
- void `extract_sequence(const char* path_input, char* sequence)` qui extrait la séquence du fichier *path_input* et la stocke dans une variable *sequence*.
- void `save_sequence(const char* path_output, char* sequence)` qui enregistre une séquence dans un fichier *path_output* au format FASTA.
- int `longueur_sequence(const char* path_input)` qui retourne la longueur de la séquence (afin de créer la variable séquence de la bonne taille).
- int `polarite(char aa)` qui retourne 0 si l'acide aminé est polaire, 1 sinon.
- int `verification_sequence(char* path_input, char* sequence, int taille_seq)` qui vérifie si la séquence est codante, c'est-à-dire si elle commence par ATG (ou AUG si c'est de l'ARN), est constituée d'un nombre de nucléotides multiples de 3 et contient un codon STOP. Elle renvoie 1 si c'est le cas, 0 sinon.

2

Module 1 : Recherche de la séquence codante de taille maximale

Ce premier module comporte sept fonctions/procédures différentes :

- `void codons_start_stop_brin_sens(char* sequence, int taille_seq, int* start, int* stop, int* longueur)` qui permet de trouver un codon START puis le STOP le plus proche dans le brin sens. Cette procédure modifie les variables *start* et *stop* en leur affectant la position du premier nucléotide du codon (la position du A pour ATG par exemple).
- `void codons_start_stop_brin_antisens(char* sequence, int taille_seq, int* start, int* stop, int* longueur)` qui permet de trouver un codon START puis le STOP le plus proche dans le brin antisens. Cette procédure recherche un codon START sur le brin antisens, elle le parcourt donc en partant de la fin et recherche le motif TAC (complémentaire de ATG), puis le codon STOP le plus proche. Par exemple :

3' **ATTCTGCTGATTCAT** 5'

← sens de lecture

Cette procédure modifie les variables *start* et *stop* en leur affectant la position du premier nucléotide du codon.

- `void sequence_complementaire(char* sequence, int taille_seq)` qui permet de créer la séquence complémentaire.
- `int codons_start_stop_ORF_max(char* sequence, int taille_seq, int* start, int* stop, int* taille)` qui renvoie 2 si la séquence est codante sur le brin sens, 1 si c'est sur le brin antisens, 0 si elle n'est pas codante. Elle utilise ensuite les 2 procédures suivantes pour stocker la séquence dans une variable suivant qu'elle soit sur le brin sens ou antisens.
- `void creation_sequence_ORF_sens(int taille_seq_codante, char* sequence, int start, int stop, char* seq_cod)` qui crée un tableau de caractères contenant la séquence codante dans le brin sens.
- `void creation_sequence_ORF_antisens(int taille_seq_codante, char* sequence, int start, int stop, char* seq_cod)` qui crée un tableau de caractères contenant la séquence codante dans le brin antisens. La séquence est stockée dans la variable *sequence* dans le sens 5' → 3'. Par exemple, si la CDS maximale sur trouve sur le brin antisens du précédent exemple, la séquence qui sera stockée dans *sequence* sera celle-ci :

5' **ATGAATCAGCAGAAT** 3'.

- `void module_recherche_sequence_codante_maximale()` qui cherche le couple de codons START et STOP avec la longueur maximale. La séquence est parcourue, si un START est trouvé, le STOP le plus proche est cherché. La longueur entre les 2 codons est calculée. Puis ceci est répété, et les longueurs des deux CDSs sont comparées. Les positions des START et STOP correspondant à la longueur maximale sont conservées. A la fin de la séquence, on obtient alors les positions du START et STOP, ainsi que la longueur de la CDS. Si aucune CDS n'est trouvée, un message est affiché.

Dans les procédures *codons_start_stop_brin_sens* et *codons_start_stop_brin_antisens* la condition suivante nous permet de choisir la CDS la plus longue mais dans le cas de deux CDS de même taille la première sera choisie.

```

if ((l = abs(st-sp)+3) > longueur_maximale) {
    longueur_maximale = l; //On prend le couple start-stop ayant la plus grande longueur
    *start = st; //On modifie les valeurs de start, stop et longueur_maximale
    *stop = sp;
    *longueur = longueur_maximale;
}

```

FIGURE 1 – Extrait de l'une des procédures du module 1

3

Module 2 : Transcription

Ce module contient les procédures suivantes :

- void transcription(char* sequence, int taille_seq) qui transcrit la séquence en remplaçant les T par des U.
- void module_transcription_sequence() qui enregistre la séquence transcrite dans un fichier FASTA. Cette procédure vérifie en amont si la séquence est codante. Si elle ne l'est pas, elle redemande à l'utilisateur un autre fichier.

4

Module 3 : Traduction

Ce module contient les procédures suivantes :

- void traduction(int taille_seq, char* seq_arn, char* seq_aa) qui traduit la séquence en acides aminés. Nous avons choisi de partir des différents codons et de faire des conditions imbriquées.
On regarde le premier nucléotide, puis le second et le troisième si besoin pour avoir l'acide aminé correspondant et donc traduire la séquence (qui a été transcrite précédemment).
- void module_traduction_sequence() qui enregistre la séquence traduite dans un fichier FASTA. Cette procédure vérifie en amont si la séquence est codante. Si elle ne l'est pas, elle redemande à l'utilisateur un autre fichier.

5

Module 4 : Calcul du score d'identité entre deux séquences

Dans ce module, nous comparons deux séquences de même taille afin d'obtenir le score d'identité ainsi que l'alignement des séquences (avec l'affichage d'une séquence permettant de voir les différences entre les deux séquences).

Nous avons choisi de faire ce module avec trois fonctions/procédures :

- `int nb_nucleotide_ident(char* sequence1, char* sequence2, int taille_seq)` qui permet de compter les nucléotides ou les acides aminés identiques entre nos deux séquences.
- `void seq_id(int taille_seq, char* seq1, char* seq2, char* id)` qui permet d'écrire dans le terminal les nucléotides identiques entre nos deux séquences et met un - lorsqu'ils sont différents.
- `void module_score_id()` qui est notre procédure principale, elle parcourt les deux fichiers (séquences) donnés par l'utilisateur et renvoie dans le terminal le score d'identité (en fraction et en pourcentage), les deux séquences alignées et leurs différences (marquées par un -).

6

Module 5 : Calcul du score de similarité de polarité entre deux séquences protéiques

Dans ce module, nous comparons deux séquences de même taille afin d'obtenir le score de polarité ainsi que l'alignement des séquences afin de pouvoir comparer tout le long des séquences les différences de polarité.

Nous avons choisi de faire ce module avec trois fonctions/procédure :

- `int nb_a_polarite(int taille_seq, char* sequence1, char* sequence2)` qui permet de compter les acides aminés de même polarité entre nos deux séquences. On utilise pour cela la fonction `polarite` du fichier `utiles.c`.
- `void seq_id_pol(int taille_seq, char* sequence1, char* sequence2, char* id)` qui permet d'écrire dans le terminal la comparaison de polarité des deux séquences.
- `void module_score_pol()` qui est notre procédure principale, elle parcourt les deux fichiers (séquences) donnés par l'utilisateur et renvoie dans le terminal le score de polarité (en fraction et en pourcentage), les deux séquences alignées et leur comparaison de polarité (1 si hydrophile, 0 si hydrophobe et - si la polarité est différente).

Pour ce module ainsi que le module 7, il était conseillé de créer une structure Acide Aminé. Nous avons préféré créer une fonction qui retourne 0 ou 1 suivant la polarité de l'acide aminé, ainsi qu'une fonction qui donne le code 3 lettres d'un acide aminé donné.

7

Module 6 : Séquence consensus issue d'un résultat d'alignement multiple

Ce module permet d'obtenir la séquence consensus d'un alignement multiple pour chaque position.

Nous avons choisi de faire ce module avec sept fonctions/procédures différentes :

- `int longueur_seq(const char* path_input)` qui permet d'avoir la longueur des séquences issues de l'alignement multiple. Dans le fichier `utiles.c` il y avait une fonction qui retournait la longueur de la séquence mais seulement quand la séquence était seule dans le fichier FASTA. Nous avons donc créé une nouvelle fonction qui calcule la taille d'une des séquences de l'alignement.
- `int nombre_sequences(const char* path_input)` qui retourne le nombre de séquences issues de l'alignement multiple.
- `void extract_sequences(const char* path_input, int long_seq, int nb_seq, char sequences[nb_seq][long_seq])` qui permet d'extraire les séquences contenues dans le fichier et de les mettre dans un tableau de caractères. Les lignes correspondent aux séquences et les colonnes à chaque position.
- `pourcentage(int nb_seq, double nb_nucleo)` qui retourne un chiffre, pour un nucléotide donné, en fonction du pourcentage d'alignement des séquences.
 - 3 si 100%
 - 2 si > 80%
 - 1 si > 60%
 - 0 sinon (< 60%)
- `void seq_consensus_par_position(char* seq_consensus, int position, int nb_seq, int nbA, int nbT, int nbG, int nbC)` qui permet d'afficher un caractère donné en fonction du pourcentage d'alignement pour une position donnée.
 - Le nucléotide correspondant si 100% (3)
 - * si > 80% (2)
 - - si > 60% (1)
 - ' ' sinon (< 60%) (0)
- `void sequence_consensus(int long_seq, int nb_seq, char sequences[nb_seq][long_seq], char* seq_consensus)` qui permet de construire et d'afficher dans le nouveau fichier la séquence consensus en utilisant la procédure précédente.
- `void module_recherche_sequence_consensus()` qui est la procédure principale de notre module. Elle demande à l'utilisateur le fichier d'entrée, et le nom du fichier de sortie. Dans le fichier de sortie, la séquence consensus sera affichée.

8

Module 7 : Recherche de la plus grande sous-chaîne de polarité commune à 2 séquences

Ce module recherche la sous-chaîne maximale de la séquence 1 telle que la séquence 2 contient une série d'acides aminés de polarité similaire.

Nous avons utilisé les fonctions/procédures suivantes :

- void `matrice_alignement(char* sequence1, char* sequence2, int taille_seq1, int taille_seq2, char mat_alignement[taille_seq2+2][taille_seq1+2])` qui crée une matrice d'alignement suivant la polarité et le schéma suivant :

$$\forall p, q > 1, M[p][q] = \begin{cases} M[p-1][q-1] & \text{si } M[0][q] = M[p][0] \\ 0 & \text{sinon.} \end{cases}$$

Par exemple, si nous avons les 2 séquences suivantes :

seq1 MASWLCE de polarité 1101100
seq2 MVWWDCE de polarité 1111000

Cette procédure va créer la matrice suivante :

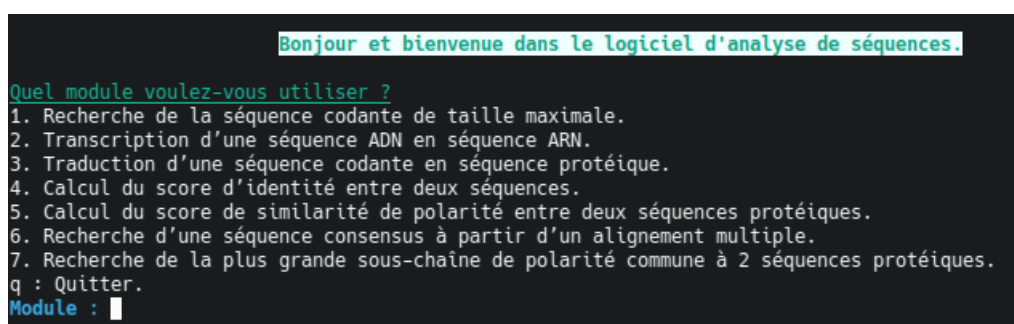
$$\begin{pmatrix} & & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 0 \\ 1 & 0 & 1 & 2 & 0 & \mathbf{1} & 2 & 0 & 0 \\ 1 & 0 & 1 & 2 & 0 & 1 & \mathbf{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & \mathbf{3} & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \mathbf{4} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 2 \end{pmatrix}$$

- void `trouver_souschaîne_maximale(int taille_seq1, int taille_seq2, char mat_alignement[taille_seq2+2][taille_seq1+2], int* taille_souschaîne, int* debut_souschaîne)` qui permet de récupérer la position de début et de fin de la plus grande sous-chaîne. Cette procédure parcourt la matrice et récupère le chiffre le plus élevé ; celui-ci correspond à la taille de la sous-chaîne maximale.
- void `conversion_3_lettres(char aa, char* code_3lettres)` qui permet de convertir la sous-chaîne d'acides aminés code 1 lettre en code 3 lettres.
- void `creation_sous_chaine(char* sequence1, char* sous_chaine_1lettre, char* sous_chaine_polarite, char* sous_chaine_3lettres, int taille_souschaîne, int debut_souschaîne)` qui permet de stocker dans les différentes variable la sous-chaîne maximale sous forme du code 3 lettres, 1 lettre et suivant la polarité des acides aminés. Dans l'exemple précédent, cette procédure va créer les variables contenant les séquences suivantes :
 - Trp-Leu-Cys-Glu pour la sous-chaîne code 3 lettres
 - WLCE pour la sous-chaîne code 1 lettres
 - 1100 pour la sous-chaîne polarité

- void save_output(int taille_sous_chaine, char* sequence1, char* sequence2, char* sous_chaine_1lettre, char* sous_chaine_polarite, char* sous_chaine_3lettres) qui permet de stocker les différentes sous-chaînes dans un fichier.
- void module_sous_chaine_polarite_maximale() qui est la procédure principale du module. Elle demande à l'utilisateur les fichiers contenant les 2 séquences ainsi que le nom du fichier de sortie.

9

Fonction principale du logiciel : main.c



```

    Bonjour et bienvenue dans le logiciel d'analyse de séquences.
    Quel module voulez-vous utiliser ?
    1. Recherche de la séquence codante de taille maximale.
    2. Transcription d'une séquence ADN en séquence ARN.
    3. Traduction d'une séquence codante en séquence protéique.
    4. Calcul du score d'identité entre deux séquences.
    5. Calcul du score de similarité de polarité entre deux séquences protéiques.
    6. Recherche d'une séquence consensus à partir d'un alignement multiple.
    7. Recherche de la plus grande sous-chaîne de polarité commune à 2 séquences protéiques.
    q : Quitter.
    Module : █
  
```

FIGURE 2 – Affichage lors du démarrage de notre logiciel

Ce fichier est composé de différentes parties. En premier, nous avons les importations de tous nos modules en .h, puis les define afin de changer l'affichage dans le terminal. Pour terminer, le main() qui permet de choisir le module grâce à la procédure *get_module_number_from_user* et grâce à l'instruction switch case.

Nous avons choisi de rajouter le cas où l'utilisateur veut quitter l'application avec 'q'. Quand l'utilisation d'un module est terminée, il y a aussi la possibilité de continuer à utiliser le logiciel.

10

Comparaison des résultats de notre logiciel

Cette partie va nous permettre de montrer les résultats d'exécution de notre code et de le comparer avec d'autres logiciels déjà existants (que nous avons déjà utilisé en cours).

10.1 Module 1

Pour comparer notre algorithme, nous avons choisi d'utiliser la séquence du récepteur P2RX7 (chromosome 12) humain de notre énoncé de projet.

Après exécution de notre code nous obtenons pour notre séquence :

```
ATGATCATAACATTCCATTTTGTAGGTGAACAAATAACAACCTGCTACAATTGAGGAGTGTTCCTTTCTTTCTTTCTTTTC
TTTTTTTTTTTATAGATGGAGTCACACTCTGCTTGCCAGGCTGGAGTGCAAGTGGATGATCTCAGCTCACTGCAACCTCT
GCCTCCTAGGTCCAAGCGATCTCCACCTCCCAAGTTTCTGGGACCAAGGATGTGCCACCAACCCAGCTAATTTTT
GTATATTCAGTAGAGATGGGGTTTCACTGTGTTGGCCAGTCTGGTCTCGAACTCTTGACCTCAAGTGATCTTCCCGCCTT
GGCTTCCCAAGTGTAGGATTACAGGCATGAGCCACTGTGCTGGCCCAAGGAGGGTTTCCATATACCAAGCACTCCC
CGTCGCCATCCCTAAATCTCCCAACACCTGGAAGGAAGATATTGTTCTGGAAGATGATTGGCCCAAGACCCACAGCT
GATAGTACATGTTGCATAATTCTAACCACGTCACCTGACCCACACTCACACTCCATCCCTTCCCTCCCATCTCATGA
TTTTCTCACCTACGCTCCATGATTGAATATTGAGTTGCTTCCAGTTTTCTATTACAAGTAA
```

Nous avons comparé nos résultats avec ceux que nous obtenons sur ORFfinder [2] qui permet de trouver toutes les ORF d'une séquence. Il faut juste avant de lancer la recherche mettre comme option "ATG only" pour "ORF start codon to use :" afin d'être dans les mêmes conditions que dans notre algorithme.

ORF177 (624 nt)
[Display ORF as...](#)
[Mark](#)

```
>1c1|ORF177 CDS
ATGATCATAACATTCCATTTTGTAGGTGAACAAATAACAACCTGCTACAAT
TCAGGAGTGTTCCTTTCTTTCTTTCTTTCTTTTATAGTGAAG
TCACACTCTGCTTGCCAGGCTGGAGTGCAAGTGGATGATCTCAGCTCAC
TGCAACCTCTGCTCCTAGGTCCAAGCGATCTCCACCTCCCAAGTTTC
TGGGACCAAGGATGTGCCACCAACCCAGCTAATTTTGTATATTGAG
TAGAGATGGGGTTTCACTGTGTTGGCCAGTCTGGTCTCGAACTCTTGACC
TCAAGTGATCTTCCCGCTTGGCTTCCCAAGTGTAGGATTACAGGCAT
GAGCCACTGTGCTGGCCCAAGGAGGGTTTCCATATACCAAGCACTCCC
CGTCGCCATCCCTAAATCTCCCAACACCTGGAAGGAAGATATTGTTTC
TGGAAAGATGATTGGCCCAAGACCCACAGCTGATAGTACATGTTGCATAAT
TCTAACCACGTCACCTGACCCACACTCACACTCCATCCCTTCCCTTCC
CATCTCATGATTTTCTCAGCTACGCTCCATGATTGAATATTGAGTTGC
TTCCAGTTTTCTATTACAAGTAA
```

ORF177
[SmartBLAST](#)
[BLAST](#)

Marked set (0)
[SmartBLAST best hit titles...](#)
[BLAST](#)

BLAST Database:
UniProtKB/Swiss-Prot (swissprot)

[Mark subset...](#)
Marked: 0
[Download marked set](#)
as **Protein FASTA**

Label	Strand	Frame	Start	Stop	Length (nt aa)
ORF177	+	3	35304	35927	624 207
ORF8	+	1	6106	6663	558 185
ORF198	-	1	51311	50781	531 176
ORF191	+	3	51375	51905	531 176
ORF189	+	3	44595	45098	504 167
ORF223	-	1	29972	29490	483 160
ORF62	+	1	51457	51933	477 158
ORF315	-	2	6835	6362	474 157
ORF281	-	2	40027	39566	462 153
ORF336	-	3	46107	45673	435 144

La capture précédente représente l'ORF maximale trouvée par ORFfinder. Nous pouvons voir que nous obtenons la même séquence qu'avec notre logiciel.

Ccl : Nos résultats correspondent donc bien à ce qui est attendu.

10.2 Module 2

Nous avons transcrit une séquence codante avec notre logiciel et avec le logiciel GénieGen [3]. Nous avons utilisé la CDS obtenue avec le module 1.

A gauche le résultat de notre logiciel et à droite celui de GénieGen.

Ccl : Nous pouvons voir que nous avons les mêmes séquences, nous observons simplement que

notre logiciel renvoie un format FASTA avec 80 caractères par ligne et le logiciel GénieGen un format FASTA mais avec 70 caractères par lignes.

```

AUGAUCUAUAACAUUCCAUUUUUGAGGUGAACAAUUAACACUGCUACAAUUCAGGCAGUGUUUUCUUUUUUUUUUU
UUUUUUUUUUUAGAUUGGAGUCACACUCUGCCAGGUGGAGUGGCAUGGCAUGAUCACAGCUCACUGCAACUCU
GCCUCCUAGGUCCAAGCGAUCCACUCUCCCAAGUUUCUGGACCCACAGGCAUGUGCCACACCCAGCUAAUUUUU
GUAAUUUCAGUAGAGUAGGUGUUUACUGUGUUGGCCAGUCUGGUCUGAACUUGACCUCAAGUGAUUUCCCGCUU
GGCUUCCCAAAGUGCUAGGAUACAGGCAUGAGCCACUGUGCCUAGGAGGUGUUUCCAUUACCAAGCACUCC
CGUGCCAUCCCUAAAUUCCCAACCCUGGAGGAAGAUUUUUGGGAAGAUUUUCCCAAGACCCACAGCU
GAUAGUACAUUGCUAAAUUUAACCCAGUCACUCUGACCCACACUCACAUCCUCCUCCUCCAUCAUGA
UUUUUCACCCUACGCCUCCAUUGAUUUAUUGAGUUGCUCCAGUUUUUUAUUUAAGUAA
AUGAUCUAUAACAUUCCAUUUUUGAGGUGAACAAUUAACACUGCUACAAUUCAGGCAGUGUUUUCUUUUUUUUUUU
UUUUUUUUUUUAGAUUGGAGUCACACUCUGCCAGGUGGAGUGGCAUGGCAUGAUCACAGCUCACUGCAACUCU
GGCAUGUGCCACACACCCAGCUAAUUUUUUGAUUUAUUCAGUAGAGUAGGUGUUUACUGUGUUGGCCAGU
CUGGUCUCGAACUCUUGACCUCAAGUGAUUUCCCGCUUUGGCUUCCAAAGUGCUAGGAUUAACAGGCAU
GAGCCACUGUGCCUGGCCAAGGAGGUGUUUCCAUUUAACCAAGCACUCCCGUGCCAUCCUAAAUUUC
CCAAACACCCUGGAGGAAGAUUUUUGGGAAGAUUUUCCCAAGACCCACAGCUGAUAGUACAU
GUUGCAUAAUUAACCCAGUCACUCUGACCCACACUCACAUCCUCCUCCUCCAUCAUGA
UUUUUCACCCUACGCCUCCAUUGAUUUAUUGAGUUGCUCCAGUUUUUUAUUUAAGUAA

```

10.3 Module 3

Concernant la traduction, nous avons utilisé le même logiciel de comparaison que pour la transcription (GénieGen [3]). Nous avons traduit la séquence que nous avons transcrit précédemment.

A gauche le résultat de notre logiciel et à droite celui de GénieGen.

Ccl : Nous pouvons voir comme pour la transcription que le nombre de caractères par ligne des formats FASTA n'est pas le même. Nous avons les mêmes séquences protéiques. Cependant avec notre logiciel le codon STOP est représenté par '*', tandis qu'avec l'autre logiciel, il n'est pas représenté en fin de séquence.

```

MIITHFVGEQITTATIAVFSFLFFSFFFLDGVTLCLPRLECSGMISAHCLLGLPSDPPTSQVSSTTGMCHHTQLIF
VYSVEMGFHCVGSGLELLTSSDLPALASQASARITGMSHCANPKGEFPYTKHSPSPSLDLPTTLEGRYCFWKMICRPTA
DSTCCIIILHTVLTTPHSHSIPSPSHDFTLYASMIIEYLSCFQFFYYK*
MIITHFVGEQITTATIAVFSFLFFSFFFLDGVTLCLPRLECSGMISAHCLLGLPSDPPTSQVSSTTGMCHHTQLIF
VYSVEMGFHCVGSGLELLTSSDLPALASQASARITGMSHCANPKGEFPYTKHSPSPSLDLPTTLEGRYCFWKMICRPTA
PTTLEGRYCFWKMICRPTADSTCCIIILHTVLTTPHSHSIPSPSHDFTLYASMIIEYLSCFQFFYYK

```

10.4 Module 4

Afin de tester ce module, nous avons calculé le score d'identité de notre CDS précédente avec la CDS suivante (obtenue en utilisant BLAST) après un alignement des séquences.

Nous avons comparé notre résultat avec celui obtenu avec Blast [4]. En haut le résultat de notre logiciel et en bas celui de Blast.

Ccl : Nos résultats correspondent à ceux obtenus avec Blast.

```

Identité de séquence: 246/310, soit 79.00 %.
Query 1 ttttcttttctttt-ttttttAGATGGAGTCACACTCTGCTTGCCAGGCTGGAGTGCAGTGGCATGATCTCAGCTCACTGCAACCTCTGCCTCTAG-GTCCAAGCGATCCT-...-CC-CA-CCTCCCAAG
TTTCTGGGACCCAGGCGATGTGCCACACACCCAGCTAATTTTG-TATATTAGTAGAGATGGGTTTCACTGTGTTGCCAGTCTGGTCTGAACTCTTGACCTCAAGTGATCTTCCCGCTTGCTTCCCAAGTGCTA
GGATTACAGGCGATGAGCACTGTGCTGGCCCAA
Sbjct 1 tttt-ttttttAAATTTTAGAGACAGTCTTGCTCTG-TGCCACAGGCTGGAGTGCAGTGGCATGATCTTGCTCACTACAATCTCTGCCTC-TAGAGTTCGGGCGATTTCTTGCTCAGCTCTCTAG
TAGTGGGATTACAGGCGTGCCACACACACCCAGTAAATTTTCTATTTTAGTACAGACAGGGTTTGCTATGTTAGCCAGGCTGGTCTTGAACTCTGGCTCAAGTGATCTGCCTGCTTGGCTCCCAAGTGCTG
GGATCACAGGCGTGAGCCACTGCGCTGGCTAA
Query 2 tttt-tttt-tttt-tttt-AGA--GAGTC--CTCTG-T-GCCACAGGCTGGAGTGCAGTGGCATGATCT--GCTCACT-CAA-CTCTGCCTC-TAG-TG-C--GGCAT-CT-...-CC-CA-CCTC-C-AG
T--TGGGA--ACAGGC-TG--CCACACACCCAG-TAATTTT--TAT-TT-AGTA-AGA--GGGTTT--CT-TGTT-GCCAG-CTGGTCT-GAATC-TG-CCTCAAGTGATCT-CC-GC-TTGGC-TGCCAAAGTGCT-
GGAT-ACAGGC-TGAGCCACTG-GCTGGCC-AA

```

Score	Expect	Identities	Gaps	Strand
207 bits(112)	9e-49	246/310(79%)	12/310(3%)	Plus/Minus
Query 71	ttttcttttctttt-ttttttAGATGGAGTCACACTCTGCTTGCCAGGCTGGAGTGC	129		
Sbjct 59888	TTTT-TTTTTTTTAAATTTTAGAGACAGATCTTGCTCTG-TGCCACAGGCTGGAGTGC	59831		
Query 130	AGTGGCATGATCTCAGCTCACTGCAACCTCTGCCTCTAG-GTCCAAGCGATCCT-...-C	184		
Sbjct 59830	AGTGGCATGATCTGGCTCACTACAATCTCTGCCTC-TAGAGTTCGGCGATTTCTGTGC	59772		
Query 185	C-CA-CCTCCCAAGTTTCTGGGACCAAGGATGTGCCACACACCCAGCTAATTTTTT	241		
Sbjct 59771	CTCAGCTCTCTAGTAGTGGGATTACAGGCGTGACACACACCCAGTAAATTTTCT	59712		
Query 242	TATATTAGTAGAGATGGGTTTCACTGTGTTGGCAGTCTGGTCTGAACTCTTGACCT	381		
Sbjct 59711	TATTTTAGTACAGACAGGGTTTGCTATGTTAGCCAGGCTGGTCTTGAACTCTGGCT	59652		
Query 382	CAAGTGATCTTCCCGCTTGCTTCCCAAGTGCTAGGATTACAGGATGAGCCACTGTG	361		
Sbjct 59651	CAAGTGATCTGCTGCTTTGGCTCCCAAGTGCTGGGATACAGGCGTGAGCCACTGCG	59592		
Query 362	CCTGGGCCAA	371		
Sbjct 59591	CCTGGCCTA	59582		

10.5 Module 5

Pour ce module, nous n'avons pas trouvé de logiciel en ligne permettant de calcul le score de similarité de polarité de 2 séquences. Nous avons donc comparé nos résultats à l'exemple de l'énoncé du projet.

A gauche les résultats de l'énoncé, à droite ceux de notre logiciel.

Ccl : Nos résultats correspondent à ceux de l'énoncé.

```
0:hydrophiles , 1:hydrophobes , -:différents
LCLLGPSDPPTSQVSTGMCHHPSLNL
MLLLEPRGSPTADETQGLHKAASLRAP
1-11-10--10-0-001100-101011
```

```
Similarité de polarité: 20/27, soit 74.00 %.
0 : hydrophiles, 1 : hydrophobes, - : différents
seq1 LCLLGPSDPPTSQVSTGMCHHPSLNL
seq2 MLLLEPRGSPTADETQGLHKAASLRAP
ld 1-11-10--10-0-001100-101011
```

10.6 Module 6

Lors de l'exécution de ce module, nous avons choisi d'utiliser l'alignement multiple proposé dans le sujet du projet.

```
CGCTGTGTCTGAGTATCCCAGGCGCGGTGCACAGTGCTCTTCTGACCGGCGTTGTAAAA
GGTTCTGTCTGAG--TCCCACCCGAGGACACTCTGTTCCAACGACTGGGGCTGTAA-A
GGTTGTGTCCCGAGTATCCCACCCGAGGACGCTCTGTTCC----CCGAGGTTG-AAAA
----GTGTCCCGAGTACCCACCC--AGGACGCTCTGTTCC----TCGGGGTTGTAAAA
```

Après exécution de notre code nous obtenons un nouveau fichier contenant la séquence consensus.

```
- - -TGTC GAG--CCCA--C--G--C C--TG-TC- --G--G-TG-AA-A
```

Nous avons choisi de comparer notre module à l'algorithme de ClustalW [1] qui permet de réaliser un alignement de séquence mais aussi d'observer les positions conservées.

```
GGTTGTGTCCCGAGTATCCCACCCGAGGACGCTCTGTTCC----CCGAGGTTG-AAAA
---GTGTCCCGAGTACCCACCC--AGGACGCTCTGTTCC----TCGGGGTTGTAAAA
GGTTCTGTCTGAG--TCCCACCCGAGGACACTCTGTTCCAACGACTGGGGCTGTAA-A
CGCTGTGTCTGAGTATCCCAGGCGCGGTGCACAGTGCTCTTCTGACCGGCGTTGTAAAA
***** ** * * * * * * * * * * * * *
```

Ccl : Nous pouvons remarquer que les positions conservées sont les mêmes pour notre logiciel et l'algorithme ClustalW. Cependant, notre résultat nous donne plus de précisions sur les autres positions (comme décrit plus tôt).

10.7 Module 7

Comme pour le module 5, nous n'avons pas trouvé de logiciel en ligne permettant de trouver la sous-chaîne commune maximale entre 2 séquences. Nous avons donc comparé nos résultats à l'exemple de l'énoncé du projet.

En haut les résultats de l'énoncé, en bas ceux de notre logiciel.

Ccl : Nos résultats correspondent à ceux de l'énoncé.

```
sequence1 MIITFHFVGEQITTATIQAVFSFLFFSFFFLDGVTLCLPRLECSGMISAHCNLCLLGPSD
sequence2 HCAWPKEGFPYTKHSPSPSLNLPTTLEGRYCFWKMICPRPTADSTC
```

Recherche de la sous-chaine maximale de la séquence 1 telle que la séquence 2 contient une série d'acides aminés de polarité identique: (0:hydrophile, 1:hydrophobe)

Phe-Leu-Asp-Gly-Val-Thr-Leu-Cys-Leu (longueur=9)

FLDGVTLCL

110110101

```
sequence1      MIITFHFVGEQITTATIQAVFSFLFFSFFFLDGVTLCLPRLECSGMISAHCNLCLLGPSD
sequence2      HCAWPKEGFPYTKHSPSPSLNLPTTLEGRYCFWKMICPRPTADSTC
```

Recherche de la sous-chaine maximale de la sequence 1 telle que la sequence 2 contient une serie d'acides aminés de polarite identique: (0:hydrophile, 1:hydrophobe)

Phe-Leu-Asp-Gly-Val-Thr-Leu-Cys-Leu (longueur = 9)

FLDGVTLCL

110110101

Références

- [1] GenomeNet, <https://www.genome.jp/tools-bin/clustalw>
- [2] ORFfinder, NCBI <https://www.ncbi.nlm.nih.gov/orffinder/>
- [3] ENS Lyon, GénieGen <https://www.pedagogie.ac-nice.fr/svt/productions/geniegen2/>
- [4] Blast, NCBI <https://blast.ncbi.nlm.nih.gov/Blast.cgi>