Projet noté – Logiciel d'analyse de séquences

1 Description du projet

Ce projet vise la mise en œuvre d'un logiciel d'analyse de séquences génomiques et protéiques eucaryotes, mettant à disposition de l'utilisateur des modules complémentaires pouvant s'enchaîner à la manière d'un pipeline :

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

Chaque module peut être considéré comme un sous-projet indépendant, vous pouvez les réaliser dans l'ordre que vous voulez. Le niveau de difficulté de chaque module est symbolisé par un numéro de niveau, augmentant avec la difficulté. Une bonne réalisation des modules de niveau 1 et 2 vous assurera la moyenne. Toute créativité complétant avantageusement le logiciel (affichage amélioré, fonctionnalités supplémentaires, ...) sera prise en compte dans la notation.

Le logiciel doit accueillir l'utilisateur avec un menu présentant les différents modules possibles, afin de lui permettre d'en choisir un.

1.1 Modalités de rendu

Ce projet est un travail en binôme. La date limite du rendu du projet est fixée au dimanche 9 janvier 2022 inclus.
Votre projet doit pouvoir être compilé et exécuté sous Unix. Vous me l'enverrez par mail dans archive `nom1-nom2.zip` contenant :

- les fichiers .c
- les fichiers .h
- un makefile pour compiler votre projet
- un rapport pour expliciter quelles sont les parties du sujet que vous avez traitées, les potentiels choix et difficultés rencontrées...

Pour ceux qui connaissent ou souhaitent se former à Git (gestionnaire de version) permettant un travail collaboratif, vous pouvez utiliser GitHub ou GitLab et m'adresser pour rendu le lien vers celui-ci au lieu de l'archive. Vous obtenez un bonus important si Git a été utilisé pertinemment, c'est-à-dire en tant que gestionnaire de version collaboratif pendant le développement et non simplement comme dépôt final de votre projet.

Les points suivants seront pris en compte dans la notation :

- Le code doit être lisible :
  — indentation correcte, code "aéré"
  — commentaires pour introduire les fonctions, expliquer des conditions... toute personne reprenant votre code doit pouvoir comprendre le raisonnement suivi par vos algorithmes
  — noms explicites de variables, fonctions, structures et constantes (et respectant les conventions de nommages)
- Chaque fonction doit se charger d'une tâche bien définie et spécifiée sans ambiguïté dans le commentaire qui précède la fonction. Ce commentaire doit également inclure une description des arguments et du renvoi de la fonction.
- Le découpage modulaire doit être pertinent, regroupant les fonctions par thématique.
- Vous montrez une utilisation pertinente des constantes.
- Votre makefile invoque l'option -Wall pour le compilateur gcc. Le programme rendu doit compiler sans afficher d'avertissements ("*warning*").

1.2 Cahier des charges par un exemple d'utilisation du logiciel :

Un utilisateur souhaite étudier la fonction du *gène P2RX7 humain*. Ce gène étant jusqu'à présent très peu étudié chez l'Homme, il débute avec pour seule donnée la séquence ADN de ce gène, au format *FASTA*. Étape de son analyse :

1. **Obtenir la séquence transcrite puis la séquence traduite.**
   Pour cela, implémentez :
   — une recherche de la plus grande *séquence codante* contenue dans une séquence ADN
   — la transcription d'une séquence ADN, afin de connaître la séquence du potentiel ARN produit
   — la traduction d'une séquence ARN, afin de connaître la séquence de la potentielle protéine produite
2. **Comparer la séquence protéique avec les séquences d'autres protéines**, afin d'étudier leur similarité.
   Pour cela, implémentez :
   — le calcul d'un score d'identité entre séquences protéiques ou nucléotidiques
   (% d'acides aminés ou nucléotides identiques à la même place dans la séquence)
   — le calcul d'un score de similarité de polarité entre séquences protéiques
   (% d'acides aminés de même polarité à la même place dans la séquence)
   — la recherche de la plus grande sous-chaîne de la séquence protéique d'intérêt pour laquelle il existe une séquence de polarité similaire au sein d'une seconde protéine
3. **Identifier les régions les plus conservées entre plusieurs protéines de la même famille**, dans l'idée de mettre en évidence les zones indispensables à la fonction de la protéine (zones de liaison avec un ligand par exemple).
   Pour cela, implémentez :
   — la recherche de la séquence consensus issues d'un alignement multiple fourni par l'utilisateur

2 Mise en oeuvre

Codez les fonctions propres aux différents modules dans des fichiers `.c` séparés (par exemple les fonctions correspondant au module de transcription sont à mettre dans un fichier appelé `module_transcription.c`, et les prototypes de ses fonctions dans un fichier `module_transcription.h`).

Pour chaque module, je vous conseille de créer une fonction principale orchestrant le déroulement du module et qui est la fonction appelée depuis le `main` lorsque l'utilisateur choisit ce module. Cette fonction principale ne prendra aucun argument, elle a donc un prototype de format : `void module_nomexplicitemodule()`
par exemple : `void module_transcription()`

Vous mettrez dans un fichier nommé `utils.c` des fonctions utiles à tous les modules et qui pourront donc être appelées depuis chacun d'entre eux. Je vous conseille de créer les fonctions suivantes :

- `void get_path_from_user(char* path_input)` qui stocke dans `path_input` un nom de fichier saisi par l'utilisateur (tous les modules nécessitent la lecture d'un fichier fournit par l'utilisateur)
- `void extract_sequence(const char* path_input, char* sequence)` qui stocke dans la variable `sequence` une séquence contenue dans un fichier au format FASTA accessible via le chemin précisé en `path_input` (pour vous aider, vous pouvez considérer des fichiers FASTA dont vous retirez manuellement la première ligne)
- `void save_sequence(const char* path_output, char* sequence)` qui écrit dans un fichier la séquence contenue dans la variable `sequence` en renvoyant à la ligne tous les 80 caractères, comme dans le format FASTA, afin de faciliter la lecture (la majorité des modules créeront un nouveau fichier stockant une séquence).

Commencez par créer ces trois fonctions dont vous aurez besoin dans pratiquement tous les modules.

2.1 Recherche de la séquence codante de taille maximale (niv.3)

Ce module demande en entrée à l'utilisateur le nom d'un fichier contenant une séquence ADN au format FASTA (dont vous pouvez avoir retiré la 1ère ligne si cela vous aide). Au sein du module, le fichier indiqué sera donc ouvert et lu (voir fonction `void extract_sequence(const char* path_input, char* sequence)`), et la séquence nucléotidique qu'il contient sera stockée dans une variable. Ne connaissant pas par avance sa taille, considérez que la longueur maximale sera de 10000 bases. Le module recherche ensuite dans cette séquence d'ADN la plus grande séquence commençant par un codon d'initiation ATG et se terminant par l'un des 3 codon-stop possibles TAA/TAG/TGA.

Ainsi, vous devez rechercher la présence d'un codon d'initiation (ATG) puis d'un codon-stop (TAA/TAG/TGA), séparés par une longueur divisible par 3. La recherche devra porter sur le brin sens comme sur le brin anti-sens. En effet, la séquence codante peut être issue de la traduction du brin anti-sens de celui dont on possède la séquence qui, pour rappel, correspond à sa lecture de "droite à gauche" et est constitué des bases complémentaires (A↔T, C↔G).

Le module crée un nouveau fichier dans lequel il écrit le résultat obtenu, à savoir la séquence codante de taille maximale. La fonction `void save_sequence(const char* path_output, char* sequence)` vous sera utile pour cette tâche (vous pouvez appeler le fichier *output_ seqcodante.txt* par exemple).

2.2 Transcription (niv. 1)

Ce module demande en entrée à l'utilisateur un fichier contenant une séquence ADN codante (la séquence doit commencer par un codon d'initiation et avoir une longueur divisible par 3, sinon l'entrée sera redemandée à l'utilisateur). En sortie, le module écrit dans un nouveau fichier cette même séquence mais en remplaçant tous les 'T' par des 'U' et en respectant un retour à la ligne tous les 80 caractères comme l'exige le format FASTA.

2.3 Traduction (niv.2)

Dans ce module, vous demandez à l'utilisateur de fournir un fichier contenant une séquence ARN, et vous traduisez cette séquence ARN en séquence protéique suivant le code génétique rappelé en figure 1.

Vous écrivez la séquence protéique dans un nouveau fichier, en respectant le retour à la ligne tous les 80 caractères.

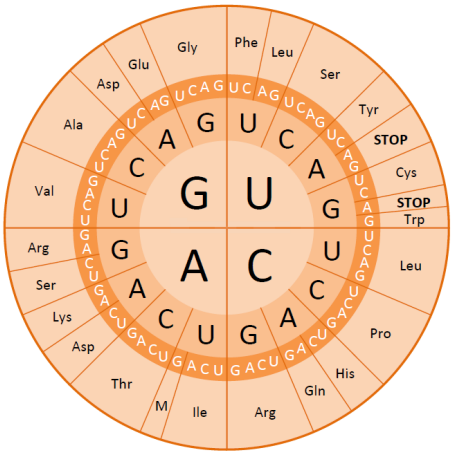

Figure 1 – Le code génétique

2.4 Calcul du score d'identité entre deux séquences (niv.1)

Ce module demande à l'utilisateur deux fichiers contenant chacun une séquence. Ces séquences correspondent au résultat d'un alignement et sont donc de taille identique (elles peuvent contenir des gaps, symbolisés par des tirets). L'objectif de ce module est de compter le nombre de caractères identiques à la même place dans les deux séquences.

Affichez à l'écran le résultat, sous cette forme (ici séquences nucléotidiques, mais peuvent être protéiques) :

```
Identité de séquence: 62/70, soit 88.6%
seq1 ATGCCGGCTTGCTGCAGCTGGAATGAT-TTTTCCAGTATGAGACAAACAAAATCATCTGGATCCAAAGCA
seq2 ATGACGGCCTGCTGCACCTGTAACGATATTTTCCAGTATGAGACAAACAAAATCATCCGGATCCA-AGCA
-id- ATG-CGGC-TGCTGCA-CTG-AA-GAT-TTTTCCAGTATGAGACAAACAAAATCATC-GGATCCA-AGCA
```

2.5 Calcul du score de similarité de polarité entre deux séquences protéiques (niv.2)

Ce module demande à l'utilisateur deux fichiers contenant chacun une séquence protéique (de même taille, pouvant être issues d'un alignement et donc contenir des gaps) et compte le nombre d'acides aminés de polarité identiques à la même place dans les deux séquences. Affichez à l'écran le résultat sous une forme similaire au score d'identité.

Pour réaliser cela, il est fortement conseillé de créer une structure modélisant les acides aminés (la structure sera réutilisée dans le module n°7).

Informations :

liste des AA apolaires (hydrophobes) :
Phénylalanine, Alanine, Leucine, Isoleucine, Méthionine, Tryptophane, Proline, Glycine, Valine

liste des AA polaires (hydrophiles) :
Cystéine, Tyrosine, Thréonine, Sérine, Asparagine, Glutamine, Histidine, Lysine, Arginine, Aspartate, Glutamate

Idée d'amélioration :
Ajoutez un affichage mettant en évidence les zones respectant une similarité, tel qu'illustré dans l'exemple ci-dessous :

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

2.6 Séquence consensus issue d'un résultat d'alignement multiple (niv.3)

Dans ce module, vous demandez à l'utilisateur d'indiquer le nom d'un fichier contenant le résultat d'un alignement multiple. Le fichier doit respecter le format suivant : il ne contient rien d'autre que les séquences alignées, avec une séquence par ligne. Ainsi le fichier suivant est un fichier d'alignement multiple :

```
CGCTGTGTCCTGAGTATCCCAGGCGCGGTGCACAGTGCTCTTCTGACCGGCGTTGTAAAA
GGTTCTGTCCTGAG--TCCCACCCGCAGGACACTCTGTTCCAACGACTGGGGCTGTAA-A
GGTTGTGTCCCGAGTATCCCACCCGCAGGACGCTCTGTTCC-----CCGAGGTTG-AAAA
----GTGTCCCGAGTACCCCACCC--AGGACGCTCTGTTCC-----TCGGGGTTGTAAAA
```

Pensez votre code afin de pouvoir traiter au maximum un alignement multiple de 30 séquences. Créez pour cela un tableau de `char` volontairement grand (de dimension 10001 sur 30, afin de pouvoir traiter jusqu'à 30 séquences de maximum 10000 caractères). Stockez dans ce tableau le contenu du fichier.

Si pour une position donnée vous avez une stricte identité de séquence (même caractère dans chacune des séquences), alors le caractère affecté à cette position dans la séquence consensus sera le caractère conservé. Si un caractère est présent dans au moins 80% des séquences alignées, alors le caractère reporté dans la séquence consensus sera une étoile ($*$). Enfin, si un caractère est présent dans au moins 60% des séquences alignées, alors le caractère reporté dans la séquence consensus sera un tiret ($-$). Sinon ce sera un espace.

Le module stocke dans un fichier la séquence consensus, dont voici un exemple :

```
 - --TGTCC GAG---CCCA--C-**G-- - - TGT*C----  -C------- A--A
```

2.7 Recherche de la plus grande sous-chaîne de polarité commune à 2 séquences (niv.4)

Ce module demande à l'utilisateur deux fichiers contenant chacun une séquence. Dans ce module, il n'est pas important que les séquences soient de même taille. L'objectif est de calculer la plus grande sous-chaîne de la première séquence telle que la deuxième séquence contient une chaîne de même taille et de polarité identique.

Différentes stratégies sont possibles pour trouver une telle séquence. Si vous trouvez plusieurs sous-séquences maximales de taille identique, vous pourrez en afficher une unique selon le critère de votre choix. Pour arriver à coder un tel algorithme, vous pourrez vous inspirer du problème algorithmique connu sous le nom de *plus longue sous-chaîne commune* : `https://fr.wikipedia.org/wiki/Plus_longue_sous-chaîne_commune`. Un algorithme de programmation dynamique est présenté sur le lien. Une légère modification de cet l'algorithme permet de passer de l'identité de lettre à l'identité de polarité. L'algorithme étant gourmand en espace mémoire, on se limitera à des séquences protéiques de taille au plus 600. Il est aussi conseillé d'utiliser une matrice *de* ***char*** *plutôt que de* ***int***, pour économiser de la mémoire.

Vous stockerez le résultat dans un nouveau fichier contenant les séquences input ainsi que le résultat calculé sous la forme de code à une lettre, code à trois lettres, et polarité. Voici ci-dessous un exemple d'output attendu pour ce module :

```
sequence1 MIITFHFVGEQITTATIQAVFSFLFFSFFFLDGVTLCLPRLECSGMISAHCNLCLLGPSD
sequence2 HCAWPKEGFPYTKHSPSPSLNLPTTLEGRYCFWKMICPRPTADSTC

Recherche de la sous-chaîne 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
```