INFO-H-304 2022-2023

Compléments de Programmation et d'Algorithmique

Projet

Alignement de séquences de protéines avec l'algorithme de Smith-Waterman

1 Introduction

L'alignement de séquences est l'un des problèmes les plus communs en bioinformatique. Le but de l'alignement de séquences est de comparer deux séquences (ou plus), typiquement une séquence inconnue et une autre provenant d'une base de données, au moyen d'une certaine mesure de similarité. Les séquences à aligner peuvent être soit des séquences d'acides nucléiques (pour l'ADN et l'ARN), soit des séquences de peptides (pour les protéines). La principale différence entre ces deux types est le nombre de molécules différentes. Une séquence d'acides nucléiques pour l'ADN ou l'ARN consiste principalement en 4 types de nucléobases: cytosine (C), guanine (G), adénine (A) pour l'ADN et l'ARN, ainsi que le thymine (T) pour l'ADN ou l'uracile (U) pour l'ARN. Par contre, une séquence de peptides pour une protéine consiste en environ 20 types différents d'acides aminés (voir [Wik25b] pour une liste complète). Dans ce projet, nous considérerons seulement des séquences de peptides (protéines).

Depuis quelques dizaines d'années, de grandes bases de données contenant des séquences d'acides nucléiques ou de peptides de nombreux organismes vivants connus ont été constituées. Ces bases de données sont utilisées par des chercheurs ou des médecins pour déterminer l'origine ou les propriétés d'organismes inconnus, après les avoir séquencés. Puisque le séquençage produit souvent des ambiguïtés, rechercher dans la base de données ne se résume pas à trouver une correspondance parfaite avec la séquence recherchée, mais plutôt à mesurer la similarité de chaque protéine de la base de données avec la séquence recherchée, au moyen d'un score qui tient compte des insertions, suppressions et mutations nécessaires pour passer d'une séquence à l'autre. Vu la possibilité d'insertion et de suppression de sous-séquences, il faut trouver l'alignement optimal menant au score de similarité le plus élevé. L'algorithme de Smith-Waterman utilise la programmation dynamique pour réaliser l'alignement de séquences, et est un des algorithmes les plus utilisés pour ce problème.

2 But du projet et cahier des charges

Le but général du projet est de produire un programme recevant en entrée une séquence de protéine à aligner avec une grande base de données de séquences, en utilisant l'algorithme de Smith-Waterman. Plus précisément, le programme devrait satisfaire le cahier des charges suivant:

Langage	• C++ et sa *Standard Template Library* • Aucune utilisation de paquets tiers, si ce n'est éventuellement pour la version optimisée optionnelle • Pour la version optimisée optionnelle, veuillez préciser les éventuels paquets à installer dans la machine virtuelle du cours pour pouvoir la compiler
Makefile	• Présence d'un Makefile **à la racine** de votre dépôt git pour faciliter la compilation des différentes version de votre projet: – `projetprelim`: version préliminaire – `projet`: version finale standard, n'utilisant aucun paquet tiers – (optionnel) `projetopt`: version finale optimisée • Le Makefile devrait donc contenir une cible par version du projet, de sorte que les commandes `make projetprelim`, `make projet` et `make projetopt` génèrent les exécutables correspondant **à la racine** de votre dépôt git.
Paramètres de la ligne de commande	• Un fichier contenant une séquence de protéine dans le format FASTA format (séquence de requête) • Un fichier contenant une base de données de protéines dans le format binaire BLAST • Un fichier contenant une matrice de score BLOSUM • Un entier indiquant le *gap open penalty* • Un entier indiquant le *gap extension penalty*
Output	• L'output sera affiché directement dans le terminal, en suivant un format bien déterminé (voir Section 5 pour plus de détails) • Celui-ci consistera en 20 lignes correspondant aux 20 séquences de la base de données qui sont les plus similaires à la séquence de requête, classées par ordre décroissant de score de similarité. • Chaque ligne reprendra l'identifiant unique de la séquence, suivi du score de similarité • Pour l'échéance intermédiaire, l'output consistera en une seule ligne, correspondant à l'identifiant unique de la séquence de la base de données qui est en correspondance exacte avec la séquence de requête.
Remise du code	• Le code source du projet devra être remis via un dépôt git créé dès le début du projet et mis à jour tout au long de son développement • Votre dépôt git devrait être créé en tant que fork du dépôt `https://gitlab.ulb.be/jroland/infoh304-projet` • Le code considéré comme soumis sera celui présent sur le dépôt git au moment de l'échéance (intermédiaire ou finale, voir Section 6)
Rapport	• En plus de votre programme, vous devrez également fournir via le dépôt git un court rapport expliquant le fonctionnement de votre programme, y compris **au moins** les éléments suivants: – Description de la structure du programme en classes, fonctions, etc. (il est également conseillé d'utiliser des commentaires directement dans le code pour le rendre plus lisible) – Éléments de réflexion et choix en résultant pour votre structure de données et éventuellement l'optimisation de votre programme – Utilisation de l'intelligence artificielle (voir Section 8)

3 Dépôt git du projet

La première opération à effectuer pour commencer à travailler sur le projet est de forker et cloner le dépôt git du projet en suivant les instructions mentionnées dans le README de ce dépôt. Ce dépôt sera utilisé par tous les membres du groupe de projet tout au long de son développement. Veuillez dès lors faire des commits réguliers au fur et à mesure de votre travail, la bonne utilisation de ce dépôt git faisant partie des critères d'évaluation du projet. Une guidance sera organisée pour vous aider à vous lancer dans l'utilisation de git.

Note: La bonne utilisation du dépôt git étant un des critères d'évaluation, veillez bien à le tenir à jour continuellement, mais aussi à ne pas y stocker tout et n'importe quoi. En résumé, vous ne devriez ajouter sur le dépôt git que vos fichiers source et Makefile, ainsi que votre rapport en format PDF. En particulier, n'y ajoutez pas de fichiers compilés ou de fichiers de base de données (cela peut d'ailleurs impacter négativement les tests en générant des erreurs, en cas de fichiers compilés pour une autre architecture, ou ralentir fortement le clonage du dépôt git, en cas de stockage de gros fichiers de base de données).

4 Comment réaliser un vrai alignement de séquences de protéines

En vue de mieux comprendre comment votre programme devrait fonctionner, nous allons d'abord utiliser des outils existants pour réaliser un vrai alignement de protéines. Pour cela, nous allons devoir télécharger une base de données de protéines, une protéine de requête (qui devra être alignée avec toutes les protéines de la base de données en vue de mesurer sa similarité avec les protéines connues), ainsi qu'un programme permettant de réaliser cet alignement. Le but du projet sera de réaliser un programme résolvant le même problème.

4.1 La base de données de protéines

Une base de données de protéines contient toutes les séquences de protéines connues. Nous allons utiliser une des bases de données les mieux maintenues, appelée Swiss-Port. Cette base de données consiste en un fichier unique pouvant être téléchargé sur le site web UniProt.

1. Téléchargez le fichier uniprot_sprot.fasta.gz
2. Décomprimez-le au moyen de Gzip (il suffit généralement de double-cliquer sur le fichier) pour obtenir le fichier `uniprot_sprot.fasta`.
3. Copiez ce fichier dans le sous-dossier `database` du dossier où vous avez cloné le dépôt git de votre projet.

Le fichier `uniprot_sprot.fasta` est un simple (mais gros !) fichier texte, qui contient pour chaque protéine une ligne d'entête commençant par le symbole ">" reprenant quelques données identifiant la protéine (en particulier un identifiant unique commençant par "`sp|`", ainsi que le nom de la protéine en anglais), puis quelques lignes reprenant la séquence d'acides aminés constituant la protéine.

Note: Ce fichier étant un simple fichier de texte, il peut en principe être ouvert avec n'importe quel éditeur de texte, mais en raison de sa grande taille, tenter de l'ouvrir avec un éditeur de texte mal optimisé peut rendre votre ordinateur peu réactif.

4.2 La protéine de requête

Lorsque des biologistes veulent étudier une nouvelle protéine, ils en effectuent le séquençage pour obtenir la séquence d'acides aminés qui la constitue, puis ils utilisent cette séquence en tant que protéine de requête pour le problème d'alignement de séquences de protéines, c-à-d qu'ils utilisent un programme pour aligner cette nouvelle séquence avec chacune des séquences dans la base de données en vue de mesurer la similarité avec les protéines connues (des séquences similaires ayant généralement des propriétés biologiques communes).

Puisqu'il n'est pas réaliste pour un projet informatique de partir d'une protéine inconnue, on utilisera à la place des protéines connues, déjà présentes dans la base de données, comme protéines de requête. Le site web UniProt propose une interface facile à utiliser permettant de rapidement trouver toute protéine connue au moyen d'un identifiant. Pour obtenir une première séquence à utiliser comme protéine de requête, suivez les étapes suivantes:

1. Visitez le site web `http://www.uniprot.org/` et entrez l'identifiant `P00533` dans le champ de recherche (à côté de **UniProtKB**).

2. Vous serez redirigé vers une page de résultat pour la protéine correspondant à cet identifiant. Vous verrez que cette protéine est appelée *human epidermal growth factor receptor*. En cliquant sur **Disease & Variants** dans le menu de gauche, vous verrez que cette protéine est impliquée dans certains cancers du poumon et des maladies inflammatoires de l'intestin et de la peau.

3. Pour télécharger la séquence de cette protéine, cliquez sur le bouton **Download** en haut de la page et choisissez le format **FASTA (canonical)**. En fonction de votre navigateur, soit le fichier `P00533.fasta` sera directement téléchargé, soit son contenu sera affiché dans le navigateur, auquel cas il suffit de sauvegarder ce contenu dans un fichier texte nommé `P00533.fasta`.

4. Copiez ce fichier dans le sous-dossier `query` du dossier où vous avez cloné le dépôt git de votre projet.

4.3 Convertir la base de données en format binaire

La base de données que vous avez téléchargée est dans le format FASTA, qui est un simple format texte. En raison des surcoûts dû à l'encodage du texte et aux accès au système de fichiers, utiliser directement un fichier texte pour aligner les séquences serait beaucoup trop lourd. Dès lors, il est nécessaire de convertir le fichier FASTA vers un format binaire nommé BLAST, initialement développé par le *US National Center for Biotechnology Information (NCBI)*.

Voici les instructions pour convertir la base de données dans le format binaire BLAST:

1. Téléchargez l'outil NCBI BLAST+ (Linux / Linux ARM / Windows / Mac ARM / Mac Intel).

2. Extrayez les fichiers de l'archive au moyen de Gzip (il suffit généralement de double-cliquer sur le fichier), et copiez le fichier `bin/makeblastdb` à la racine du dossier où vous avez cloné le dépôt git de votre projet.

3. Depuis ce dossier, exécutez la commande

```
./makeblastdb -in database/uniprot_sprot.fasta -dbtype prot
   -blastdb_version 4
```

   Note: Si vous avez un Mac, il est possible que MacOS refuse de lancer l'exécutable makeblastdb. Dans ce cas, ouvrez les "Réglages système", cliquez sur "Confidentialité et sécurité", puis sur le bouton "Autoriser" qui devrait apparaître dans la section "Sécurité" à côté d'un message indiquant que "makeblastdb" a été bloqué.

4. Quand la conversion sera terminée, vous trouverez dans le dossier `database` des nouveaux fichiers avec des extensions telles que `.pin`, `.phr` et `.phq`. Il s'agit des fichiers de la base de données en format binaire BLAST.

4.4 Matrices BLOSUM

Dans l'algorithme de Smith-Waterman que vous allez implémenter pour effectuer l'alignement de protéines [SW81, Got82], la mesure de la similarité entre deux séquences de protéines se base sur des matrices de scores appelées BLOSUM et standardisées par le NCBI. Pour chaque acide aminé qui diffère entre la protéine de requête et la protéine de la base de données avec laquelle elle est alignée, la matrice BLOSUM affecte une pénalité dépendant de la probabilité qu'une mutation ait eu lieu d'un

acide aminé vers l'autre. Dès lors, l'algorithme aura besoin d'une telle matrice pour calculer les scores de similarité. Le programme Swipe proposé ci-dessous utilise par défaut la matrice `BLOSUM62` que vous pouvez télécharger sur le site du NCBI.

1. Visitez l'adresse `https://ftp.ncbi.nih.gov/blast/matrices/` et téléchargez le fichier `BLOSUM62`.
2. Copiez ce fichier dans le sous-dossier `blosum` du dossier où vous avez cloné le dépôt git de votre projet.

4.5 Le programme Swipe

Pour réaliser l'alignement de la protéine de requête `P00533` avec toutes les protéines présentes dans la base de données, et ce avant de créer votre propre programme, nous allons utiliser un programme existant, nommé Swipe, qui se base sur le même algorithme que celui que vous devrez implémenter, à savoir l'algorithme de Smith-Waterman [SW81, Got82]. La description détaillée et l'analyse de performance de Swipe sont détaillées dans l'article scientifique [Rog11].

Note: Le programme Swipe est incompatible avec l'architecture ARM, et donc en particulier avec les Mac récents équipés de processeurs Apple Silicon (M1/M2/M3). Une solution alternative si vous disposez d'un tel Mac est d'utiliser le programme `SSW` (voir ci-dessous).

Pour tester Swipe, suivez les étapes suivantes:

1. Clonez le dépôt git de Swipe.
2. Compilez Swipe avec la commande habituelle `make swipe`.
3. Copiez l'exécutable `swipe` dans la racine du dossier où vous avez cloné le dépôt git de votre projet.
4. Vous disposez maintenant de toutes les éléments nécessaires pour aligner la protéine de requête `P00533` avec toutes les protéines présentes dans la base de données. Pour ce faire, exécutez la commande suivante depuis la racine du dossier où vous avez cloné le dépôt git de votre projet.

```
./swipe -i query/P00533.fasta -d database/uniprot_sprot.fasta -M
   blosum/BLOSUM62 -G 11 -E 1 > results.txt
```

5. Voici une rapide explication des différents paramètres de cette commande:
   - `-i query/P00533.fasta`: protéine de requête
   - `-d database/uniprot_sprot.fasta`: base de données
   - `-M blosum/BLOSUM62`: matrice de score BLOSUM
   - `-G 11`: *gap open penalty*, soit une pénalité à appliquer lorsqu'une sous-séquence a été insérée ou supprimée par rapport à la protéine de requête
   - `-E 1`: *gap extension penalty*, soit une pénalité à additionner au *gap open penalty* pour chaque acide aminé supplémentaire inséré ou supprimé par rapport à la protéine de requête
   - `> results.txt`: redirection de l'output vers le fichier `results.txt` (sinon l'output serait affiché dans le terminal). Ceci n'est pas spécifique à `swipe` et pourra donc être utilisé par votre propre programme, ce qui sera utile pour sauvegarder les résultats de vos tests.
6. Vous pouvez ouvrir le fichier `results.txt` avec n'importe quel éditeur de texte pour visionner le résultat. Celui-ci contient d'abord une trentaine de lignes reprenant différentes informations techniques sur votre requête et sur la base de données, ainsi que sur le temps qui a été nécessaire pour réaliser l'alignement. Les lignes suivantes reprennent une par une les protéines de la base de données les mieux alignées avec la protéine de requête, par ordre décroissant de score de similarité:

```
Sequences producing significant alignments:                          Score

gnl|BL_ORD_ID|119556 sp|P00533|EGFR_HUMAN Epidermal growth facto...  6525
gnl|BL_ORD_ID|119557 sp|P55245|EGFR_MACMU Epidermal growth facto...  6487
gnl|BL_ORD_ID|119558 sp|Q01279|EGFR_MOUSE Epidermal growth facto...  5957
gnl|BL_ORD_ID|524935 sp|P13388|XMRK_XIPMA Melanoma receptor tyro...  3327
gnl|BL_ORD_ID|124804 sp|Q15303|ERBB4_HUMAN Receptor tyrosine-pro...  3269
...
```

7. La première ligne correspond à la protéine de requête elle-même vu que pour notre test, celle-ci est déjà présente dans la base de données, et c'est quand on aligne la protéine avec elle-même qu'on obtient le score de similarité le plus élevé. Il est néanmoins intéressant de noter que la protéine correspondante chez le macaque, et dans une moindre mesure chez la souris, mène également à des scores de similarité élevés.

4.6 Alternative à Swipe: le programme SSW

Si vous possédez un Mac récent avec processeur Apple Silicon (M1/M2/M3), vous ne pourrez pas utiliser le programme Swipe qui est incompatible avec les processeurs ARM, mais vous pouvez utiliser à la place le programme SSW (celui-ci est d'ailleurs également compatible avec les processeurs Intel mais est un peu moins pratique à utiliser, voir les différences avec Swipe listées ci-dessous).

1. Clonez le dépôt git de SSW.
2. Compilez SSW avec la commande habituelle `make`, lancée dans le dossier `src` du dépôt.
3. Copiez l'exécutable `ssw_test` dans la racine du dossier où vous avez cloné le dépôt git de votre projet.
4. Pour réaliser l'alignement de la protéine de requête avec toutes les protéines de la base de données, utilisez la commande

```
./ssw_test -p -f 3000 -a blosum/BLOSUM62 -o 12 -e 1
   database/uniprot_sprot.fasta query/P00533.fasta > results.txt
```

   Vous reconnaîtrez dans l'ensemble les mêmes paramètres que pour Swipe, mais il y a quelques différences importantes:

   - Par défaut, l'output reprend toutes les séquences de la base de données, et contrairement à Swipe celles-ci ne sont pas triées par ordre décroissant des scores de similarité. L'option "`-f 3000`" permet de n'afficher que les protéines dont le score est plus élevé que 3000, ce qui rend l'output plus digeste (vous pourriez avoir à ajuster ce paramètre pour l'alignement d'autres protéines).
   - Le programme SSW utilise une autre convention pour le paramètre "`-o 12`" qui est l'équivalent de la somme des *gap open penalty* ("`-G 11`") et *gap extension penalty* (" `-E 1`") pour Swipe. Pour pouvoir comparer les scores il faut donc bien ajuster ces paramètres. Noter que le paramètre "`-e 1`" correspond pour sa part exactement au *gap extension penalty* (" `-E 1`") pour Swipe.
   - Alors que Swipe utilise les fichiers de la base de données dans le format binaire BLAST, SSW repart du format texte FASTA (et convertit lui-même ce fichier en un format plus efficace). Cela ne change rien pour le calcul des scores, mais votre propre algorithme devrait, tout comme Swipe, utiliser le format binaire BLAST (l'utilisation telle quelle du format FASTA rendrait votre algorithme nettement moins efficace, et demander à votre programme de réaliser en interne la conversion vers un format binaire compliquerait votre projet).

5. A ces quelques différences près, vous pouvez utiliser le programme SSW à la place de Swipe pour vérifier que votre projet calcule correctement les scores de similarité (qui devraient correspondre aux *optimal_alignment_scores* calculés par SSW).

5 Votre projet

Le but du projet est de réaliser un programme similaire à Swipe, utilisant l'algorithme de Smith-Waterman pour aligner les séquences de protéines. Pour ce faire, il faudra suivre les étapes suivantes.

1. **Comprendre comment manipuler la base de données et les fichiers de requête**. Comme votre programme utilisera les formats très répandus FASTA et BLAST, il est essentiel d'arriver à lire et manipuler les données dans ces fichiers avant de commencer à implémenter votre algorithme. Le format FASTA est bien expliqué sur Wikipedia [Wik25a], et un document PDF expliquant le format binaire BLAST pour la base de données est disponible sur le dépôt git du projet [Far10]. Finalement, vous pouvez facilement trouver diverses ressources en ligne expliquant comment un programme C++ peut lire un fichier binaire.

2. **Créer une version préliminaire de votre projet capable de retrouver la protéine de requête dans la base de données**. Comme étape préliminaire, écrivez un programme capable de trouver dans la base de données une séquence en correspondance parfaite avec la protéine de requête. Le programme devrait se comporter comme Swipe, mais renvoyer en output une seule protéine de la base de données, correspondant à la séquence exacte de la protéine de requête (le fichier FASTA de la protéine de requête pourrait contenir un faux nom et une fausse référence pour rendre la recherche plus intéressante, mais la séquence exacte devrait se trouver dans la base de données). Ce programme préliminaire nécessitera donc d'implémenter les fonctionnalités d'input/output du programme final (y compris la manipulation des fichiers de la protéine de requête FASTA et de la base de données BLAST), mais plutôt qu'implémenter l'algorithme de Smith-Waterman pour chaque séquence dans la base de données, le programme devra effectuer un simple test d'égalité avec la séquence de la protéine de requête. L'exécutable `projetprelim` devrait respecter la syntaxe suivante

   ```
   ./projetprelim query/P00533.fasta database/uniprot_sprot.fasta
   ```

   avec les paramètres suivants, dans l'ordre:

   - `query/P00533.fasta`: protéine de requête
   - `database/uniprot_sprot.fasta`: base de données (notez que même si on donne le nom de la base de données avec l'extension `.fasta`, le programme devrait, comme Swipe, lire les informations nécessaires dans les fichiers binaires `.fasta.pin`, `.fasta.psq` et `.fasta.phr`)

   Par défaut, l'output devrait être envoyé dans le terminal (mais il peut donc être redirigé vers un fichier `results.txt` en ajoutant "`> results.txt`" à la fin de la commande), et ne contenir qu'une seule ligne reprenant l'identifiant unique de la protéine de la base de données en correspondance parfaite, soit pour l'exemple ci-dessus:

   ```
   sp|P00533|EGFR_HUMAN
   ```

   Cette version préliminaire du programme devrait être livrée à l'échéance intermédiaire du projet (voir date ci-dessous).

   Aide : La principale difficulté technique pour cette échéance intermédiaire est d'arriver à lire les fichiers binaires au format BLAST. Vous trouverez quelques conseils utiles pour réaliser cette tâche dans l'Annexe A de ce document.

3. **Comprendre l'algorithme de Smith-Waterman**. L'algorithme de Smith-Waterman a été initialement proposé dans l'article [SW81], puis a été modifié par Gotoh dans l'article [Got82]. Pour bien comprendre l'algorithme, vous pouvez commencer par lire attentivement ces articles, ainsi que l'article sur le programme Swipe [Rog11], voire d'autres références que vous pouvez trouver en ligne. Ces articles clarifieront la définition du problème d'alignement de séquences de protéines, ainsi que la signification des différents paramètres comme la matrice de score BLOSUM, le *gap open penalty* et le *gap extension penalty*.

4. **Implémenter l'algorithme de Smith-Waterman**. Vous pouvez maintenant procéder à votre propre implémentation de l'algorithme de Smith-Waterman. Votre programme devrait accepter les mêmes paramètres que le programme Swipe. Par souci de simplification, ces paramètres seront toujours donnés dans le même ordre, et on omettra donc l'utilisation des "flags" `-i`, `-d`, `-M`, `-G` et `-E`. Dès lors, voici la syntaxe permettant d'exécuter votre programme avec les mêmes paramètres que notre exemple pour Swipe donné plus haut:

```
./projet query/P00533.fasta database/uniprot_sprot.fasta
   blosum/BLOSUM62 11 1
```

   On a donc les trois paramètres additionnels suivants par rapport à la version préliminaire:

   - `blosum/BLOSUM62`: matrice de score BLOSUM
   - `11`: *gap open penalty*
   - `1`: *gap extension penalty*

   L'output du programme, toujours affiché par défaut dans le terminal, devrait maintenant être constitué de 20 lignes reprenant les identifiants uniques des 20 protéines les mieux alignées avec la protéine de requête, suivis des scores de similarités correspondants (par ordre décroissant). Pour l'exemple ci-dessus, l'output devrait donc être le suivant:

```
sp|P00533|EGFR_HUMAN 6525
sp|P55245|EGFR_MACMU 6487
sp|Q01279|EGFR_MOUSE 5957
sp|P13388|XMRK_XIPMA 3327
sp|Q15303|ERBB4_HUMAN 3269
...
```

   Pour tester votre code, qui avant optimisation risque d'être fort lent, vous pouvez considérez des séquences de requête de plus en plus longues, voici par exemple trois possibilités de protéines de requête (une courte, une moyenne et une longue).

   - `P07327`: Human alcohol dehydrogenase 1A
   - `P00533`: Human epidermal growth factor receptor
   - `Q9Y6V0`: Human piccolo

5. **(Optionnel) Optimiser votre code**. La première version de votre projet risque de ne pouvoir traiter en un temps raisonnable que des séquences relativement courtes. Vous pouvez néanmoins essayer d'optimiser votre code de différentes manières, de la simple parallélisation (multi-threading), à des méthodes d'accélération plus sophistiquées utilisant par exemple les instructions vectorielles de votre CPU, voire le calcul parallèle sur votre carte graphique via OpenCL ou d'autres outils de calculs GPU. Contrairement aux versions précédentes de votre projet, cette version optimisée (exécutable `projetopt`) peut faire appel à des bibliothèques tierces (dans ce cas, vous devriez préciser dans votre rapport quels paquets doivent être installés sur la machine virtuelle du cours pour permettre la compilation de votre programme).

6 Evaluation

Les critères d'évaluation suivants seront utilisés:

1. Respect de toutes les contraintes du cahier des charges
2. Qualité générale du code, y compris les aspects orientés objets (un code bien commenté est nécessaire pour pouvoir vérifier ce critère)
3. Qualité du rapport (fond et forme)
4. Rapidité de votre programme
5. Utilisation mémoire de votre programme

6. Bonne utilisation de git: mise en place dès le début du projet, commits réguliers, pertinence des fichiers déposés sur le git (code source uniquement, pas de fichiers compilés, pas de programmes externes comme Swipe, pas de fichiers de base de données, etc.)

Le premier critère est indispensable pour réussir le projet (note supérieure à la moitié), les critères suivants permettant d'améliorer la note davantage. Des scripts de test seront fournis pour vérifier que votre projet satisfait

7 Plagiat

Comme tout travail à l'université, le plagiat sera sévèrement sanctionné. Dans le cas d'un programme informatique, toute utilisation de code écrit par quelqu'un d'autre sans mention explicite sera considérée comme du plagiat. Cela inclut également le code écrit par d'autres étudiants: vous ne pouvez pas récupérer du code écrit par un membre d'un autre groupe de projet.

En pratique, vous n'avez pas le droit de

- copier-coller (ou copier manuellement) la moindre ligne de code qui vous n'auriez pas écrite vous-même (en particulier du code trouvé sur internet ou écrit par un étudiant hors de votre groupe), sauf exception indiquée ci-dessous;
- demander à un autre étudiant de manipuler directement tout ou partie de votre code.

Vous avez par contre le droit de

- discuter de votre projet avec d'autres étudiants ou demander oralement de l'aide pour résoudre des problèmes que vous pourriez rencontrer;
- rechercher de l'aide sur internet sur un problème générique de programmation (syntaxe ou utilisation d'une fonction);
- (éventuellement) réutiliser des bouts de code trouvés sur internet pour réaliser des tâches génériques non directement liées au sujet du projet, à condition de bien mentionner vos sources et de préciser les lignes de code concernées, étant entendu que cela ne peut concerner qu'une partie négligeable du code.

IMPORTANT: votre programme sera systématiquement testé avec un logiciel détecteur de plagiat qui a déjà fait ses preuves. Toute détection de plagiat sera sanctionnée d'une note de 0, en accord avec l'article 83 du Règlement général des études, et un rapport sera envoyé au Doyen qui pourra appliquer des sanctions disciplinaires supplémentaires, comme précisé dans l'article 23 du Règlement de discipline relatif aux étudiants.

8 Utilisation de l'intelligence artificielle

L'utilisation de l'intelligence artificielle est autorisée mais de manière strictement encadrée. En cas d'utilisation d'outils d'intelligence artificielle, celle-ci doit être explicitée et justifiée, conformément à l'article 40 du Règlement général des études. De manière générale, tout code généré par intelligence artificielle sera considéré comme du code écrit par autrui et pourra donc être sanctionné de plagiat s'il ne respecte pas les conditions explicitées dans la section "plagiat".

En pratique, vous n'avez pas le droit d'utiliser du code généré par intelligence articifielle pour résoudre directement une des tâches décrites dans ce document, par exemple extraire des données de fichiers au format FASTA et BLAST ou implémenter l'algorithme de Smith-Waterman.

Vous avez pas contre le droit de vous aider d'outils d'intelligence artificielle pour débugger votre code (à condition que ceci soit mentionné explicitement, en précisant les lignes de code concernées, étant entendu que cela ne peut concerner qu'une partie négligeable du code) ou pour vous aider à améliorer la forme de votre rapport (notamment l'orthographe), mais pas pour générer le contenu proprement dit.

IMPORTANT: Comme indiqué dans le cahier des charges, votre rapport devra obligatoirement contenir une section détaillant l'utilisation de l'intelligence artificielle dans votre projet, y compris

si vous n'y avez pas eu recours (dans ce cas la section se résumera à une phrase certifiant qu'aucun outil d'intelligence artificielle n'a été utilisé).

9 Echéances

La remise du projet se fera en trois étapes:

- Dès le début de votre travail (et au plus tard pour le mercredi 29 octobre): communiquez le lien de votre dépôt git via le formulaire "Lancement du projet"
- Pour le mercredi 12 novembre à 18h - Echéance intermédiaire : validez la version intermédiaire du programme, telle que décrite au point 2 de la section 5 de ce document, avec le script de test `testprelim` qui sera fourni, et partagez les résultats via le formulaire "Remise intermédiaire"
- Pour le mercredi 17 décembre à 18h - Echéance finale : validez la version finale du programme, implémentant l'algorithme de Smith-Waterman, avec le script de test `testfinal` qui sera fourni, et partagez les résultats via le formulaire "Remise finale"

Pour chaque échéance, il n'est pas nécessaire d'envoyer les codes, ceux-ci seront récupérés directement sur votre dépôt git (la version considérée comme soumise sera celle présente sur le dépôt git à la date et heure de l'échéance). De même, pour l'échéance finale, le rapport de projet devra être mis en ligne sur votre dépôt git.

IMPORTANT: Les scripts de test qui seront fournis pour chaque échéance permettent de vérifier que votre programme satisfait certains éléments de base du cahier des charges. Veillez à les utiliser correctement: si votre programme ne passe pas certains tests de base, il pourrait ne pas être évalué et se voir directement attribuer une note de 0.

Liens

- Dépôt git du projet: https://gitlab.ulb.be/jroland/infoh304-projet
- Uniprot protein search: http://www.uniprot.org/
- Swiss-Prot database: https://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/uniprot_sprot.fasta.gz
- NCBI BLAST+: Linux / Linux ARM / Windows / Mac ARM / Mac Intel
- BLOSUM matrices: http://ftp.ncbi.nih.gov/blast/matrices/
- Swipe, Smith-Waterman database searches with inter-sequence SIMD parallelisation: https://github.com/torognes/swipe
- SSW Library, An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications: https://github.com/mengyao/Complete-Striped-Smith-Waterman-Library

Références (articles disponibles sur le dépôt git du projet)

[Far10] Michael S. Farrar, *NCBI BLAST Database Format*, 2010.

[Got82] Osamu Gotoh, *An improved algorithm for matching biological sequences*, Journal of Molecular Biology **162** (1982), no. 3, 705 – 708.

[Rog11] Torbjørn Rognes, *Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation*, BMC Bioinformatics **12** (2011), no. 1, 221.

[SW81] Temple F. Smith and Michael S. Waterman, *Identification of common molecular subsequences*, Journal of Molecular Biology **147** (1981), no. 1, 195 – 197.

[Wik25a] Wikipedia, *Fasta format*, https://en.wikipedia.org/wiki/FASTA_format, 2025.

[Wik25b] ———, *Proteinogenic amino acid*, http://en.wikipedia.org/wiki/Proteinogenic_amino_acid, 2025.

A Lecture des fichiers binaires au format BLAST

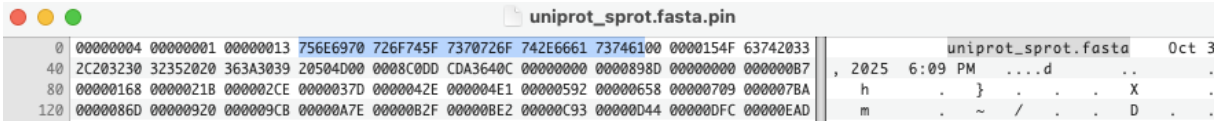
A.1 Endianness

Une erreur typique lors de décodage de nombres entiers dans un fichier binaire est d'obtenir les bons chiffres, mais en ordre inverse (par exemple 312 à la place de 213). Dans ce cas, il est fort probable qu'il y ait une erreur d'endianness (voir par exemple Wikipedia).

En résumé, les octets ont sans doute été décodés dans le mauvais ordre. Comme indiqué à la page 2 du document explicitant le format BLAST [Far10], les entiers sont encodés par 32 bits (4 octets) avec le format "big endian" (sauf le nombre total de résidus qui est encodés en 64 bits avec le format "little endian", mais en principe vous n'en ayez pas besoin).

A.2 Visionner des fichiers binaires

Pour mieux comprendre la structure des fichiers binaires, vous pouvez les ouvrir avec un éditeur de fichiers hexadécimaux (le screenshot ci-dessous utilise "Hex Fiend" sous MacOS, mais vous pouvez facilement trouver des applications pour Windows ou Linux en recherchant "hex editor"). Voici par exemple ce que cela donne si vous ouvrez le fichier `.pin` avec un tel éditeur :



Vous voyez que les 2 premiers groupes de 8 chiffres hexadécimaux (correspondant donc à 2 nombres codés sur 32 bits) encodent bien les entiers 4 et 1, c-à-d la version 4 et le type 1 (protéines). Le groupe suivant vaut 13 en hexadécimal, soit 19 en décimal (1x16+3x1), qui correspond à la longueur du titre, et les 38 chiffres hexadécimaux suivants (surlignés en bleu) encodent en ASCII (1 octet=2 chiffres hexadécimaux par caractère) les 19 caractères du titre `uniprot_sprot.fasta` (voir partie droite du screenshot, qui correspond au décodage en ASCII du fichier).

A.3 Décodage du fichier header `.phr`

Le fichier le plus délicat à lire est sans doute le fichier header `.phr`, mais celui-ci n'est utile que pour l'affichage des résultats. Voici un peu d'aide pour vous aider à l'utiliser:

- Une fois que votre programme a retrouvé la protéine recherchée dans la base de données, vous devriez avoir identifié l'indice i correspondant à la position de cette protéine dans la base de données (avec $0 <= i < N$, où N est le nombre de protéines dans la base de données).
- Sur base de cet indice i, vous pouvez trouver dans la "header offset table" du fichier index `.pin` la position des informations relatives à cette protéine dans le fichier header `.phr`
- Vous pourriez entièrement *parser* le fichier `.phr` à partir de cette position, mais c'est inutilement compliqué, car l'information dont vous avez besoin est en fait contenue dans la première chaîne de caractères (`VisibleString`, voir [Far10]). Comme les chaînes de caractères sont précédées du code hexadécimal 1A, il suffit de rechercher le premier octet prenant la valeur 1A à partir de la position où se trouve les informations de la protéine.
- Vous pouvez ensuite lire la chaîne de caractères dans les octets suivants (attention, le(s) premier(s) octet(s) encode(nt) la longueur de la chaîne, comme indiqué dans le document de référence).
- Vous n'avez besoin que du premier "mot" de cette chaîne de caractères (par exemple, `sp|P00533|EGFR_HUMAN`), vous pouvez donc soit vous arrêter de lire au premier espace, soit lire toute la chaîne puis la tronquer après le premier mot).

A.4 Décodage des acides aminés

Pour le fichier `.psq`, notez que chaque acide aminé est codé par un octet (8 bits), mais l'encodage n'est pas celui de la table ASCII (correspondant au type char en C ou C++), mais un autre encodage précisé à la page 3 du document [Far10]. Il vous faudra donc utiliser la table dans ce document pour le décodage.