Projet Phylogénétique

2023/2024

Enseignant : George Marchment

Courriel : george.marchment@lisn.fr

Date : 17/11/2023

Organisation du projet :

Le projet va être divisé en 2 parties :

- Programmation du projet en **groupe** (70% de la note).
- Soutenance **individuelle** en janvier (30% de la note).

Partie Programmation :

Merci de m'envoyer un email avec les membres du groupe (2 ou 3 au maximum) et soit un fichier .zip contenant le projet, soit un lien Github permettant d'accéder à votre projet avant le **08/01/2024 à 23h59**.

Il n'est pas obligatoire d'utiliser Github pour réaliser le projet, cependant, il est fortement recommandé. Si vous choisissez d'utiliser Github, il est important de créer un fichier `README.md` *ainsi qu'une description complète et visible avec des instructions permettant de lancer le projet.*

Soutenance :

Plus de précisions vous seront envoyées sur la soutenance bientôt (date, lieu, organisation, etc.).

Projet inspiré du projet phylogénétique proposé par Théophile Sanchez pour l'UE de bioinformatique.

Rappel :

Le but de ce projet est de mettre à l'épreuve vos connaissances et vos compétences en programmation en C. Je n'ai aucun doute que ChatGPT ou d'autres outils peuvent proposer des solutions adaptées à ces problèmes. Veuillez donc limiter votre recours à ce type d'outils. Toute occurrence d'utilisation d'IA générative doit être très clairement mentionnée avec le prompt utilisé et le code concerné.

Présentation de ce projet :

Au cours de ce projet, vous étudierez trois espèces disparues de félins qui vivaient autrefois sur le continent américain. Ces trois espèces, le smilodon (tigre à dents de sabre), l'homotherium (scimitar-toothed tigers) et Miracinonyx trumani (guépard américain) se sont éteintes il y a environ 13 000 ans, à la fin de la dernière période glaciaire. Des séquences d'ADN anciennes de la protéine cytochrome b de ces espèces ont pu être séquencées et vont vous permettre de retrouver les liens de parenté entre ces espèces et des espèces de félins contemporaines : le chat domestique, le lion, le léopard, le tigre, le puma, le guépard et les chats sauvages africains, chinois et européens. Des séquences issues d'espèces extérieures aux félins sont également présentes dans le jeu de données.

Afin de reconstruire l'arbre phylogénétique de ces espèces, vous utiliserez une méthode basée sur le calcul des distances évolutives entre les séquences d'ADN des protéines. Notez qu'une démarche similaire peut être appliquée aux séquences d'acides aminés.

Dans ce projet, vous allez implémenter diverses méthodes pour parser un fichier `fasta`, évaluer et effectuer un alignement de séquence pairwise, ainsi que construire les arbres phylogénétiques grâce à divers algorithmes. Les différentes étapes qui vous permettront de construire l'arbre seront détaillées dans la suite.

La structure, le main ainsi que le makefile permettant de construire et exécuter le projet vous ont déjà été donnés. En conséquence, pour faire le projet, il faut simplement remplir les "TODO" précisés dans les divers fichiers "`.c`", pour implémenter les différentes fonctions et les fonctions intermédiaires (pour chaque fonction, une description est donnée avant sa déclaration). Il n'est pas obligatoire de modifier la

structure ou même d'ajouter de nouvelles fonctions, cependant, rien ne vous empêche de le faire.

Vous pouvez utiliser des fonctions définies entre fichiers sans les préciser dans le fichier “**utils.h**”. Dans “la vraie vie”, il n’est pas conseillé de procéder ainsi, mais pour simplifier le projet, on ignore les warnings.

Compiler et exécuter le projet :

La structure et le makefile vous sont fournis pour compiler et exécuter le projet. Dans le fichier main, il y a 5 entiers qui sont définis (**SEQUENCES**, **ALIGNEMENT**, **MATRICE_DISTANCE**, **UPGMA**, **NJ**), qui représentent des booléens (**1** ou **0**). Lorsque l'une de ces variables a la valeur **1**, cela signifie que nous allons exécuter cette partie. À la fin de ce projet, les 5 variables doivent avoir la valeur **1**. Un exemple de la sortie du terminal attendue vous est donné dans les fichiers **exemple_sortie_1.txt** et **exemple_sortie_2.txt** dans le dossier “**Exemples**”.

Pour compiler le projet, effectuez les commandes suivantes :

```
$ make
$ ./phylo
```

Partie 1 : Lecture d'un fichier fasta et importation des séquences

Le format FASTA permet de stocker plusieurs séquences (ADN, ARN ou peptidiques) dans un fichier. Les séquences que vous allez étudier ont été regroupées dans le fichier "`cat_dna.fasta`".

> Dans cette partie, il vous est demandé de remplir les « `TODO` » dans le fichier "`sequences.c`", permettant de parser un fichier FASTA et ensuite d'afficher les séquences une par une. Consultez les fichiers "`exemple_sortie_1.txt`" et "`exemple_sortie_2.txt`" dans le dossier "`Exemples`" pour voir un exemple de sortie souhaitée.

Aide : Utilisez la structure "`Sequence`" définie dans le fichier "`utils.h`".

Partie 2 : Alignement des séquences

La méthode que vous utiliserez pour calculer l'arbre phylogénétique nécessite de calculer la distance évolutive entre les séquences. Avant de pouvoir les calculer, il faut d'abord aligner les séquences en considérant trois types de mutations :

- les substitutions (un nucléotide est remplacé par un autre)
- les insertions
- les délétions

Par exemple, les séquences "`ACTCCTGA`" et "`ATCTCGTGA`" ont plusieurs alignements possibles (il en existe d'autres) :

A_1	A_2	A_3
`-ACTCCTGA` `ATCTCGTGA`	`A-CTCCTGA` `ATCTCGTGA`	`AC-TCCTGA` `ATCTCGTGA`

Il est aussi possible qu'une délétion ne soit pas une mutation, mais en réalité dû à un mauvais séquençage ou à la dégradation d'ADN comme c'est souvent le cas pour le séquençage d'ADN d'espèces disparues.

Le "`-`" désigne un gap, c'est-à-dire un "trou" dans l'alignement qui a été causé par une insertion ou une délétion. On regroupe ces deux types de mutations sous le terme indel.

Ces alignements correspondent à une multitude d'histoires phylogénétiques différentes. Pour sélectionner le meilleur alignement il faut donc introduire l'hypothèse du maximum de parcimonie qui privilégie l'histoire phylogénétique qui implique le moins d'hypothèses et donc, le moins de changements évolutifs. Par exemple, parmis les trois alignements ci-dessus on préférera l'alignement 2 car il correspond au scénario avec le moins de mutations:

- l'alignement 1 implique au minimum 1 indel et 2 substitutions
- l'alignement 2 implique au minimum 1 indel et 1 substitutions

- l'alignement 3 implique au minimum 1 indel et 2 substitutions

On peut maintenant définir un score d'identité que l'on va augmenter de 1 lorsque qu'il n'y pas eu de mutation et ainsi obtenir la matrice suivante :

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	1

Cette matrice correspond au modèle d'évolution de l'ADN défini par Jukes et Cantor qui fait l'hypothèse d'un taux de mutation équivalent pour chacun des nucléotides. Cependant, en réalité ces taux ne sont pas les mêmes partout, on sait par exemple que le taux de transition (substitution A➜G ou C➜T) est différent du taux de transversions (substitution A➜T, C➜G, C➜A ou G➜T) et que d'autres facteurs devrait être pris en compte comme la fréquence du nucléotide dans l'ADN. C'est pour cette raison qu'il existe beaucoup de modèles différents décrivant l'évolution de l'ADN. Dans la suite de ce projet nous utiliserons la matrice de similarité **S** suivante :

	A	C	G	T	-
A	10	-1	-3	-4	-5
C	-1	7	-5	-3	-5
G	-3	-5	9	0	-5
T	-4	-3	0	8	-5
-	-5	-5	-5	-5	0

Implémentez les fonctions suivantes dans le fichier "**`alignement.c`**" : **`get_val_base`**, **`similarity_score`**, **`score_alignement`**, et **`print_quality_alignement`**. Ces fonctions ont pour objectif de calculer le score d'alignement entre deux séquences

Partie 3 : Algorithme de Needleman-Wunsch

Maintenant que vous avez vu ce qu'est une matrice de similarité et comment calculer le score de similarité d'un alignement, vous allez devoir implémenter un algorithme permettant de trouver le meilleur alignement global entre deux séquences. Avec deux séquences à aligner de taille n et m, la première étape consiste à initialiser deux matrices de taille $(n + 1) \times (m + 1)$. La première est la matrice de score `M` et la seconde sera la matrice de *traceback* `T`.

Par exemple, avec la matrice `S` et les séquences `A = "ACTCCTGA"` et `B = "ATCTCGTGA"`, on initialise `M` comme si l'on ajoutait des *gaps* partout :

	-	A	T	C	T	C	G	T	G	A
-	0	-5	-10	-15	-20	-25	-30	-35	-40	-45
A	-5									
C	-10									
T	-15									
C	-20									
C	-25									
T	-30									
G	-35									
A	-40									

Puis on initialise `T` :

	-	A	T	C	T	C	G	T	G	A
-	o	l	l	l	l	l	l	l	l	l
A	u									
C	u									
T	u									
C	u									
C	u									
T	u									
G	u									
A	u									

Il faut ensuite remplir la matrice M en suivant la formule $M_{i,j} = max(M_{i-1,j-1} + s(A_i, B_j), M_{i,j-1} + s(A_i, gap), M_{i-1,j} + s(B_j, gap))$ avec $i \in 2,\dots,n+1$, $j \in 2,\dots,m+1$ et la fonction s qui calcule le score de similarité entre deux nucléotides. Pour chaque case de T on remplie par :

- 'd' (*diagonal*) si $M_{i,j}$ a été calculé en utilisant la diagonale $M_{i-1,j-1}$,
- 'l' (*left*) si $M_{i,j}$ a été calculé en utilisant la case de gauche $M_{i,j-1}$,
- 'u' (*up*) si $M_{i,j}$ a été calculé en utilisant la case du haut $M_{i-1,j}$.

On obtient alors les matrices suivantes M et T :

	-	A	T	C	T	C	G	T	G	A
-	0	-5	-10	-15	-20	-25	-30	-35	-40	-45
A	-5	10	5	0	-5	-10	-15	-20	-25	-30
C	-10	5	7	12	7	2	-3	-8	-13	-18
T	-15	0	13	8	20	15	10	5	0	-5
C	-20	-5	8	20	15	27	22	17	12	7
C	-25	-10	3	15	17	22	22	19	14	11
T	-30	-15	-2	10	23	18	22	30	25	20
G	-35	-20	-7	5	18	18	27	25	39	34
A	-40	-25	-12	0	13	17	22	23	34	49

	-	A	T	C	T	C	G	T	G	A
-	o	l	l	l	l	l	l	l	l	l
A	u	d	l	l	l	l	l	l	l	d
C	u	u	d	d	l	d	l	l	l	l
T	u	u	d	l	d	l	l	d	l	l
C	u	u	u	d	l	d	l	l	l	l
C	u	u	u	d	d	d	d	d	l	d
T	u	u	d	u	d	l	d	d	l	l
G	u	u	u	u	u	d	d	u	d	l
A	u	d	u	u	u	d	u	d	u	d

Il suffit maintenant de regarder le dernier élément $M_{nm} = 49$ pour avoir le score de l'alignement. Pour avoir l'alignement lui-même, il faut partir de T_{nm} et remonter la "trace" jusqu'à arriver au '**o**'. Un '**d**' correspond à un *match* entre les deux séquences, '**l**' à un *gap*

dans la séquence `A` et '`u`' à un *gap* dans la séquence `B`. En revenant à l'exemple précédent on obtient la trace suivante :

	-	A	T	C	T	C	G	T	G	A
-	o									
A		d	l							
C				d						
T					d					
C						d				
C							d			
T								d		
G									d	
A										d

Elle correspond à l'alignement :

```
A-CTCCTGA
ATCTCGTGA
```

Implémentez les fonctions nécessaires pour appliquer l'algorithme de Needleman-Wunsch dans le fichier "**`alignement.c`**".

Partie 4 : Matrice de distance

Pour construire des arbres phylogénétiques, nous avons besoin d'un ensemble de séquences alignées (alignement multiple). Par conséquent, nous ne pouvons pas utiliser l'algorithme de Needleman-Wunsch, car il permet uniquement d'effectuer un alignement pairwise (entre deux séquences). En conséquence, le fichier "**`cat_dna_aligne.fasta`**" contient les séquences déjà alignées.

Dans le cas de séquences très proches (ce qui est le cas ici), on estime que la distance évolutive réelle entre les séquences est proche de la $p_{distance}$ qui est simplement le nombre de substitution dans l'alignement sur le nombre total de nucléotides. Pour simplifier, on ignore les positions alignées à des gaps. On applique ensuite la correction de Jukes-Cantor afin de prendre en compte le phénomène de saturation (un même site peut muter plusieurs fois au cours du temps). Sa formule est $-\left(\frac{3}{4}\right) \times ln\left(1 - \left(\frac{4}{3}\right) \times p_{distance}\right)$.

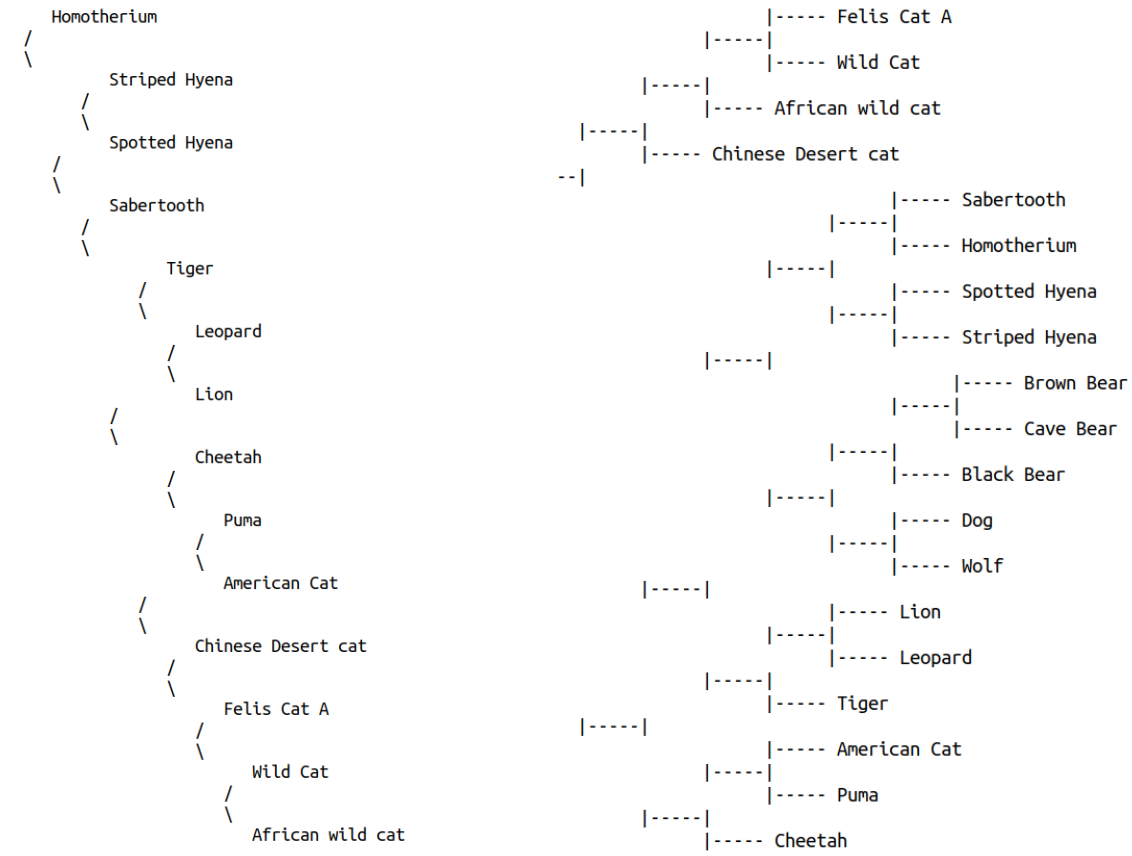
Implémentez les fonctions dans le fichier “**`matrice_distance.c`**” permettant de calculer la matrice de distance (matrice inférieure) avec la fonction décrite ci-dessus à partir des séquences trouvées dans le fichier "**`cat_dna_aligne.fasta`**".

Partie 5 : Affichage d’un arbre

Dans la suite, nous allons construire des arbres phylogénétiques avec deux algorithmes (voir ci-dessous). Il serait donc utile de pouvoir visualiser ces arbres pour les étudier. Dans le fichier "**`tree.c`**", la procédure "**`afficher_arbre_plat`**" prend en entrée un objet de type **`Arbre`** (voir le fichier "**`utils.h`**") et affiche l'arbre d'une manière "plate" (voir exemple ci-dessous).

```
((Homotherium, ((Striped Hyena, Spotted Hyena), (Sabertooth, ((Tiger, (Leopard, Lion)), ((Cheetah, (Puma, American Cat)), (Chin
ese Desert cat, (Felis Cat A, (Wild Cat, African wild cat)))))))), ((Wolf, Dog), (Black Bear, (Cave Bear, Brown Bear))))
```

Exemple d'un affichage d’un arbre plat.



Exemples d'affichage d’arbres.

En vous basant sur cette fonction, proposez une nouvelle fonction d’affichage d’arbre (vous pouvez vous inspirer des exemples ci-dessus).

Aide :

- Isolez cette fonction et créez un arbre de type joué par exemple “`((A, B), (C, (D, E)))`” pour tester votre fonction.
- Étudiez bien les structures "**`Arbre`**" et "**`Noeud`**" définies dans "**`utils.h`**".

Partie 6 : Construction d'un arbre avec UPGMA

Grâce aux mesures de distances entre les séquences, on peut maintenant construire l'arbre phylogénétique de la protéine cytochrome b. Vous allez devoir pour cela implémenter l'algorithme UPGMA (unweighted pair group method with arithmetic mean) qui, malgré son nom compliqué, est l'une des méthodes les plus simples pour la construction d'arbre.

L'algorithme UPGMA se base sur la matrice de distance entre les séquences. À chaque itération, les séquences avec la distance la plus faible sont regroupées puis une nouvelle matrice de distance est calculée avec le nouveau groupe. Cette étape est répétée jusqu'à n'avoir plus qu'un seul groupe. Par exemple, avec la matrice de distance entre les séquences `A`, `B`, `C` et `D` suivante :

	A	B	C	D
A				
B	4			
C	8	8		
D	2	4	8	

Les séquences `A` et `D` sont les plus proches (`distance(A,D)=2`). On les regroupe et on met à jour la matrice :

	(A,D)	B	C
(A,D)			
B	4		
C	8	8	

On regroupe maintenant `(A,D)` et `B` (`distance((A,D),B) = 4`) :

	((A,D),B)	C
((A,D),B)		
C	8	

Important : les nouvelles distances sont calculées en moyennant les distances entre les membres du nouveau groupe et des groupes non modifiés pondérés par le nombre d'UTOs dans chaque groupe. Avec **i** et **j** les deux membres du groupe nouvellement formé et k les groupes restant : $d_{ij,k} = \frac{n_i d_{ik}}{n_i+n_j} + \frac{n_j d_{jk}}{n_i+n_j}$. Par exemple avec la distance entre `((A, D), B)` et `C` :

`distance(((A, D), B), C) = (distance((A, D), C)*2 + distance(B, C))/ 3 = (8*2 + 8)/ 3 = 8`.

L'arbre final est : `((A, D), B), C)`

Dans cette partie, il vous est demandé de remplir les "**TODO**" dans le fichier "**`tree.c`**" pour implémenter l'algorithme UPGMA.

Remarque : Pour simplifier la vie lors de la construction de notre arbre, nous ignorons la distance entre les branches. Nous nous intéressons uniquement aux liens de parenté.

Partie 7 : Construction d'un arbre avec Neighbor-joining

Le neighbor-joining est un autre algorithme permettant de calculer un arbre phylogénétique à partir d'une matrice de distance. Il a l'avantage de faire moins d'hypothèse que UPGMA sur les données (elles ne sont plus forcément ultramétriques) et il donne donc de meilleurs arbres dans presque tous les cas. Vous trouverez un exemple d'application de cet algorithme ici.

Dans cette partie, il vous est demandé de remplir les "`TODO`" dans le fichier "**`tree.c`**" pour implémenter l'algorithme Neighbor-Joining.

Remarque : Pour simplifier la vie lors de la construction de notre arbre, nous ignorons la distance entre les branches. Nous nous intéressons uniquement aux liens de parenté.