Cette épreuve est constituée de deux problèmes indépendants.

Problème n° 1

Ce problème s'intéresse au processus de séquençage de l'ADN. Dans ce contexte, l'une des étapes importantes du séquençage permet de déterminer quelles sont les parties de la séquence d'ADN recherchée mais dans un ordre indéterminé. Le problème consiste alors à **reconstruire** la séquence complète à partir de ces parties. La suite de ce sujet propose une résolution algorithmique à ce problème.

Pour cela, nous représentons **une séquence** S_n d'ADN de longueur n par une suite de n lettres, chaque lettre appartenant à l'ensemble $\{A, T, G, C\}$.

Exemple : $S_9 = AGGTCAGGT$ est une séquence d'ADN de 9 lettres.

Recherche de mots dans une séquence d'ADN

Dans un premier temps nous allons chercher à déterminer quelles sont les parties de longueur fixe d'une séquence, appelées **mots de la séquence** par la suite.

Formellement, soient S_n une séquence d'ADN et l un entier tel que $0 < l \leq n$. On appelle *mot de S_n* de longueur l toute suite de l lettres contiguës contenue dans la séquence S_n.

Exemples : GTC est un mot de longueur 3 de la séquence $S_9 = AGGTCAGGT$, tandis que ATT n'en est pas un.

1. **Combien de mots de longueur l existe-t-il dans une séquence de longueur n (en comptant les répétitions possibles) ?**

2. **Proposer un algorithme qui, étant donnés un entier n, une séquence d'ADN S_n de longueur n et un entier strictement positif l ($l \leq n$), calcule la liste de tous les mots de longueur l de S_n.**

   Une telle liste peut contenir plusieurs fois le même mot. Vous supposerez que la séquence d'ADN est contenue dans un tableau de caractères. Il n'est pas demandé de décrire les manipulations de listes que vous aurez éventuellement besoin d'effectuer, comme l'ajout d'un élément dans une liste par exemple.

   Exemple : l'algorithme appelé sur la séquence $S_9 = = AGGTCAGGT$ et $l = 3$ doit calculer la liste [AGG,GGT,GTC,TCA,CAG,AGG,GGT].

3. **Quelle est la complexité de votre algorithme en nombre d'opérations, en fonction de n et de l ?**

On s'intéresse à la question de déterminer tous les mots distincts de longueur l dans S_n. Nous cherchons ici à proposer une fonction Python. Une séquence S_n sera alors décrite comme une chaîne de caractères de longueur n.
Exemple : la liste de tous les mots distincts de longueur 3 *dans la séquence* S_9 = *"AGGTCAGGT" où* $n = 9$ *et* $l = 3$ *est ["AGG","GGT","GTC","TCA","CAG"].*

4. **Soient n et l deux entiers ($n \geq l > 0$). Combien de mots distincts de longueur l existe-t-il au plus dans une séquence de longueur n ?**

5. **Proposer une structure de données en Python adaptée pour stocker les mots de la séquence sans répétition.**

6. **Proposer une fonction `liste_mots` en Python qui prend en arguments une séquence S_n, un entier n et un entier l ($n \geq l > 0$), et qui renvoie la liste de tous les mots distincts de longueur l de S_n.**

Reconstruction d'une séquence d'ADN

L'une des difficultés lors du séquençage de l'ADN est qu'il faut reconstruire la séquence complète à partir de l'ensemble des mots de longueur l qui la composent, sans qu'aucune information sur leur position dans la séquence ne soit donnée.
Le reste de cet exercice va se concentrer précisément sur la résolution algorithmique de ce problème.

Soient S_n une séquence d'ADN de longueur n et l un entier tel que $0 < l \leq n$. On appelle spectre(S_n, l) l'ensemble des mots de longueur l distincts qui sont dans la séquence S_n.
Exemple : étant donnée la séquence $\mathcal{S}_9$ = *AGGTCAGGT, spectre(*S_9*,3) = {AGG, CAG, GGT, GTC, TCA}.*
Le problème de la **reconstruction** de la séquence d'ADN consiste, à partir de la connaissance d'un spectre $\mathcal{SP}$ contenant des mots de longueur l, à déterminer une séquence d'ADN S compatible avec $\mathcal{SP}$, c'est-à-dire telle que spectre$(S, l) = \mathcal{SP}$. Plus formellement, on peut formuler le problème **Reconstruction** de la manière suivante :
Données : un entier l ; un ensemble $\mathcal{SP}$ de mots de longueur l.
Sortie : une séquence d'ADN S telle que spectre$(S, l) = \mathcal{SP}$.

Pour résoudre ce problème, nous allons nous intéresser aux recouvrements existant entre les mots d'un spectre.
Soient M un mot de longueur $n > 0$ et k un entier tel que $0 < k \leq n$, on appelle *préfixe* de M de longueur k le mot constitué des k premières lettres de M. De manière similaire, on appelle *suffixe* de M de longueur k le mot constitué des k dernières lettres de M.

Soient deux mots M_1 et M_2 de longueur n_1 et n_2 respectivement et soit $k \leq \min(n_1, n_2)$ un entier strictement positif. Il existe un *recouvrement de longueur k* entre M_1 et M_2 si et seulement si le suffixe de M_1 de longueur k est identique au préfixe de M_2 de longueur k. Nous appellerons également *longueur du recouvrement maximal* entre M_1 et M_2, noté lrmax(M_1, M_2) dans la suite, le plus grand entier k tel qu'il existe un recouvrement de longueur k entre M_1 et M_2. Si aucun recouvrement n'existe entre M_1 et M_2, lrmax$(M_1, M_2) = 0$.
Exemple : lrmax$(ACGG, CTAG) = 0$ *et* lrmax$(ACGG, GGAT) = 2$.

7. **Que valent :**
   - lrmax($ATGC$, $GGTA$) ?
   - lrmax($TGGCGT$, $CGTAAATG$) ?
   - lrmax($GCTAGGCTAA$, $AGGCTAAGTCGAT$) ?
   - lrmax($TCTAGCCAGCTAGC$, $TAGCCAGCTAGCACT$) ?

Première modélisation

Soient $l \geq 2$ un entier et $\mathcal{SP}$ un spectre contenant des mots de longueur l. Nous allons modéliser notre problème par un graphe orienté $G = (V, E)$, dans lequel :

- Chaque sommet de V correspond à un mot du spectre et chaque mot du spectre est représenté par un et un seul sommet.
- L'ensemble E contient un arc entre $v_1 \in V$ et $v_2 \in V$ si et seulement si $\text{lrmax}(v_1, v_2) = l-1$.

8. **Quelle est la modélisation obtenue sous forme de graphe pour les spectres et longueurs suivants :**
   - $\{GTGA, ATGA, GACG, CGTG, ACGT, TGAC\}$ et $l = 4$?
   - $\{TAC, ACC, ACG, CAC, CCG, CGT, CGC, GCA, GTA\}$ et $l = 3$?

9. **Proposer une formulation du problème de reconstruction comme un parcours de graphe. De quel problème classique de théorie des graphes ce problème se rapproche-t-il ?**

10. **Proposer pour chacun des graphes obtenus dans la question 8 une solution au problème Reconstruction.**

Deuxième modélisation

Il est possible de modéliser autrement ce problème, toujours sous forme de graphe.

Nous modélisons maintenant les données ($l \geq 2$ un entier et $\mathcal{SP}$ un spectre contenant des mots de longueur l) par un graphe orienté $G = (V, E)$ où :

- V est l'ensemble de tous les préfixes de longueur $l-1$ et de tous les suffixes de longueur $l-1$ des mots du spectre.
- L'ensemble E contient un arc entre les sommets $v_1 \in V$ et $v_2 \in V$ si et seulement si le spectre $\mathcal{SP}$ contient un mot M de longueur l tel que v_1 est le préfixe de M de longueur $l-1$ et v_2 est le suffixe de M de longueur $l-1$.

11. **Quelle est la modélisation obtenue sous forme de graphe pour les spectres suivants :**
   - $\{GTGA, ATGA, GACG, CGTG, ACGT, TGAC\}$ et $l = 4$?
   - $\{TAC, ACC, ACG, CAC, CCG, CGT, CGC, GCA, GTA\}$ et $l = 3$?

   On rappelle que dans un graphe orienté $G = (V, E)$, un *chemin* d'origine u ($\in V$) et d'extrémité v ($\in V$) est défini par une suite d'arcs consécutifs reliant u à v. Un *circuit* est un chemin dont les deux extrémités sont identiques. Un chemin (resp. circuit) est dit *eulérien* s'il contient une fois et une seule chaque arc de G.

12. **Les graphes construits à l'aide de la modélisation de la question 11 contiennent-ils un circuit eulérien ? un chemin eulérien ?**

13. **En quoi un circuit ou un chemin eulérien dans un graphe construit avec cette modélisation constitue-t-il une solution au problème de la reconstruction de séquence d'ADN ?**

    Dans la suite de l'exercice, nous nous restreignons au circuit eulérien qui est plus simple à déterminer qu'un chemin eulérien.
    Soit $(u, v) \in E$. On dit que l'arc (u, v) est un arc entrant du sommet v et un arc sortant du sommet u. On définit le degré entrant d'un sommet v, noté $d_e(v)$, comme le nombre d'arcs entrants du sommet v. On définit le degré sortant d'un sommet u, noté $d_s(u)$, comme le nombre d'arcs sortants du sommet u.

14. **Montrer qu'un graphe orienté contient un circuit eulérien si et seulement si $\forall v \in V, d_e(v) = d_s(v)$.**

    Dans la suite de l'exercice, un graphe sera représenté en Python par un dictionnaire dont les clefs sont des chaînes de caractères (`str`) représentant les sommets du graphe et les valeurs sont des listes de chaînes de caractères (`list`) contenant les sommets voisins de la clé.

15. **Écrire une fonction `presence_circuit_eulerien` en Python qui prend en arguments un graphe orienté G et qui détermine si G a un circuit eulérien.**

    Nous supposons que les graphes considérés dans les questions 16, 17, 18, 19 et 20 comprennent un circuit eulérien.

16. **Écrire une fonction `construction_circuit` en Python qui prend en arguments un graphe orienté G et un sommet v quelconque de G et qui renvoie un circuit ayant pour origine le sommet v.**

17. **Écrire une fonction `enleve_circuit` en Python qui prend en arguments un graphe orienté G et un circuit C de G et qui supprime de G les arcs appartenant au circuit C.**

18. **Écrire une fonction `sommet_commun` en Python qui prend en arguments un graphe orienté G et un circuit C (qui n'est pas inclus dans G) et qui renvoie un sommet commun de degré non nul appartenant à la fois à G et à C.** Nous supposerons que G et C ont au moins un tel sommet en commun.

19. **Écrire une fonction `fusion_circuits` en Python qui prend en arguments deux circuits $C1$ et $C2$ et v un sommet commun à $C1$ et $C2$ et qui renvoie un circuit composé des arcs de $C1$ et des arcs de $C2$.**

20. **À partir des fonctions Python que vous avez proposées aux questions 16, 17, 18 et 19, écrire une fonction `circuit_eulerien` en Python qui prend un entrée un graphe orienté G et qui renvoie un circuit eulérien de G.**

21. **Quelle est la séquence d'ADN déterminée par la fonction proposée à la question 20 sur le graphe obtenu à la question 11 avec le spectre** $\mathcal{SP} = \{TAC, ACC, ACG, CAC, CCG, CGT, CGC, GCA, GTA\}$ **et** $l = 3$ **?**

22. **Proposer une autre séquence d'ADN compatible avec le spectre** $\mathcal{SP} = \{TAC, ACC, ACG, CAC, CCG, CGT, CGC, GCA, GTA\}$ **et différente de la séquence déterminée à la question 21.**

Problème n° 2

Préambule : les différentes parties de ce problème peuvent être traitées indépendamment les unes des autres, bien que les définitions et notations données au fur et à mesure du sujet soient communes à toutes les parties concernées.

Nous nous intéressons dans ce problème aux systèmes de gestion de base de données (SGBD), c'est-à-dire aux logiciels qui permettent de stocker et manipuler des informations contenues dans les bases de données. Les SGBDs implémentent différents mécanismes pour assurer les bonnes propriétés des bases de données, notamment la persistance des données en cas de panne, l'atomicité et la cohérence des transactions ainsi que la protection des données en fonction des droits des utilisateurs.

1 Interrogation et manipulation de bases de données

Vous disposez d'un site de critiques gastronomiques dans lequel les utilisateurs peuvent évaluer des restaurants. Ce site s'appuie sur un modèle relationnel composé de trois relations.

Les restaurants sont stockés dans la relation *Restaurant*. Chaque restaurant est identifié par son *rID* et a un *nom*, un *type* (de cuisine) et une *adresse*.

La relation *Evaluateur* décrit les évaluateurs qui sont identifiés par leur *eID* et décrits par leur *pseudonyme* et leur date d'inscription (*dateInscription*).

Enfin, la relation *Evaluation* décrit les appréciations faites par les évaluateurs sur les restaurants. Chaque évaluation est identifiée par l'ensemble d'attributs *rID*, *eID*, *dateEval* où *rID* référence un restaurant dans la relation *Restaurant* et *eID* référence un évaluateur dans la relation *Evaluateur*. Une évaluation contient également une *note*.

Ceci donne le schéma relationnel suivant :

- *Restaurant*(*rID*, *nom*, *type*, *adresse*)
- *Evaluateur*(*eID*, *pseudonyme*, *dateInscription*)
- *Evaluation*($rID^{\#}$, $eID^{\#}$, *dateEval*, *note*)

On suppose que les tables, associées à ce schéma relationnel, viennent d'être créées avec :

```
/* Création de la table Restaurant */
CREATE TABLE Restaurant(
rID integer,
nom varchar(50),
type varchar(30),
adresse varchar(50));
```

```
/* Création de la table Evaluateur */
CREATE TABLE Evaluateur(
eID integer,
pseudonyme varchar(30),
dateInscription date);
/* Création de la table Evaluation */
CREATE TABLE Evaluation(
rID integer,
eID integer,
dateEval date,
note integer);
```

1. **Implanter en SQL les contraintes et requêtes suivantes.**
   (a) *rID* est la clé primaire de la relation *Restaurant*.
   (b) *eID* est la clé primaire de la relation *Evaluateur*. Le couple (*pseudonyme*, *dateInscription*) est également clé.
   (c) Déterminer tous les restaurants qui ont été évalués par l'évaluateur dont l'*eID* est 135.
   (d) On souhaite obtenir les paires d'évaluateurs qui ont noté le même restaurant. Retourner le pseudonyme de ces deux évaluateurs en éliminant les duplications ((a, b) et (b, a) représentent la même paire d'évaluateurs).
   (e) Pour tous les cas où la même personne note deux fois le même restaurant et donne une note plus élevée la seconde fois, retourner le pseudonyme de l'évaluateur et le nom du restaurant.
   (f) Trouver le pseudonyme de tous les évaluateurs qui ont fait au moins 3 évaluations.
   (g) Les restaurants (respectivement les évaluateurs) de la relation *Evaluation* doivent être contenus dans la relation *Restaurants* (respectivement *Evaluateur*) : *Evaluation*[*rID*] $\subseteq$ *Restaurant*[*rID*] (respectivement *Evaluation*[*eID*] $\subseteq$ *Evaluateur*[*eID*]).

2. **Donner une définition des propriétés d'atomicité, de cohérence et de persistance énoncées en introduction du sujet.**

2 Inférence de dépendances fonctionnelles

On s'intéresse dans cette partie aux problèmes d'anomalies lors de mises à jour d'une base de données. De tels événements étant difficilement détectables, l'une des solutions possibles consiste à réduire les redondances présentes dans les bases de données.

Pour ce faire, l'une des approches usuelles consiste à étudier les *dépendances fonctionnelles* entre les attributs de la base de données :

Définition 1 (Dépendance fonctionnelle)**.** *Soient R une relation et X et Y deux sous-ensembles de l'ensemble des attributs de R. On dit que, dans une instance de relation r définie sur le schéma R, il existe une dépendance fonctionnelle entre X et Y (ou que X détermine fonctionnellement Y) si et seulement si pour tous n-uplets t_1 et t_2 de r, $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$.*

Une dépendance fonctionnelle entre X et Y est notée $R : X \rightarrow Y$ (ou simplement $X \rightarrow Y$ lorsque le schéma est implicite). La projection d'un n-uplet t sur les attributs X, notée $t[X]$, retourne les valeurs du n-uplet t pour chaque attribut de X.

Intuitivement, une dépendance fonctionnelle entre deux ensembles d'attributs X et Y exprime le fait que si l'on connait la valeur des attributs X alors on peut retrouver sans aucune ambiguïté les valeurs correspondantes sur les attributs Y. Plus simplement, pour une valeur de X il ne correspond qu'une seule valeur de Y possible.

L'intérêt de l'étude des dépendances fonctionnelles réside dans le fait qu'il est possible, à partir de quelques dépendances fonctionnelles explicites, de déterminer par inférence l'ensembles des dépendances fonctionnelles valides dans un schéma relationnel.

Dans le cas général, une règle d'inférence s'exprime sous la forme d'une relation entre les *prémisses* et la *conclusion* de la règle, représentée graphiquement par :

$$\text{Règle d'inférence : } \frac{\text{Formule}_1 \quad ... \quad \text{Formule}_n}{\text{Conclusion}}$$

Si les prémisses Formule$_1$, .., Formule$_n$ sont vrais, alors Conclusion est vraie.

Dans le contexte des dépendances fonctionnelles, on dispose des règles d'inférence suivantes (X, Y, Z, W sont des sous-ensembles de l'ensemble des attributs de R, et pour deux ensembles X et Y, XY désigne par abus de notation l'union de X et Y ($XY = X \cup Y$)) :

$$\text{Réflexivité } (\sigma_{Re}) : \frac{Y \subseteq X}{X \to Y}$$

$$\text{Augmentation } (\sigma_A) : \frac{X \to Y}{WX \to WY}$$

$$\text{Transitivité } (\sigma_T) : \frac{X \to Y \qquad Y \to Z}{X \to Z}$$

Ces trois règles d'inférences forment le système d'inférence d'Armstrong. Le but de cette partie est d'étudier et d'utiliser le système d'inférence d'Armstrong.

3. **Expliquer ce qu'est une anomalie de mises à jour.**

4. **Rappeler ce qu'est une *clé* dans une base de données. En quoi une clé détermine-t-elle fonctionnellement tous les autres attributs d'un schéma relationnel ?**

5. **Montrer que la règle d'inférence suivante est fausse.**

$$\frac{X \to Y \qquad Y \to Z}{Z \to X}$$

   On dit qu'un système d'inférence est correct s'il ne permet pas de produire des dépendances fonctionnelles non valides.

6. **Est-ce qu'un système d'inférence contenant la règle d'inférence de la question 5 est correct ?**

7. **Démontrer que les règles du système d'Armstrong (Réflexivité, Augmentation et Transitivité) sont correctes en exploitant la notion de dépendance fonctionnelle. Conclure sur le système d'inférence d'Armstrong.**

8. **Soit le schéma de relation $R(A, B, C, D, E, F)$ (A, B, C, D, E et F étant les attributs de la relation R). En utilisant le système d'inférence d'Armstrong $\{\sigma_{Re}, \sigma_A, \sigma_T\}$, montrer que** :
   (a) $CE \rightarrow E$.
   (b) La dépendance fonctionnelle $BD \rightarrow C$ peut être inférée à partir de l'ensemble de dépendances fonctionnelles $\{AB \rightarrow BC, D \rightarrow A\}$.
   (c) La dépendance fonctionnelle $AB \rightarrow F$ peut être inférée à partir de à partir de l'ensemble de dépendances fonctionnelles $\{AB \rightarrow C, A \rightarrow D, CD \rightarrow EF\}$.

3 Fermeture transitive

On s'intéresse dans la suite de l'exercice à déterminer l'ensemble des dépendances fonctionnelles que l'on peut dériver à partir des règles d'inférences du système d'Armstrong. Pour cela, on va s'intéresser à la notion de fermeture transitive.

Soient X un ensemble d'attributs et Σ un ensemble de dépendances fonctionnelles sur le schéma de relation R. $\Sigma \models X \rightarrow Y$ signifie que l'ensemble de dépendances fonctionnelles Σ implique la dépendance fonctionnelle $X \rightarrow Y$. La *fermeture transitive* de X par rapport à Σ, notée $X^+_{\Sigma,R}$ (ou simplement X^+ lorsque Σ et R sont évidents) est définie de la façon suivante :

$$X^+_{\Sigma,R} = \{A \ \in R \ tq \ \Sigma \models X \rightarrow A\}.$$

Intuitivement, la fermeture transitive d'un ensemble d'attributs X correspond à tous les attributs qui sont déterminés fonctionnellement par X.

9. Soit l'algorithme 1 qui calcule la fermeture transitive d'un ensemble d'attributs par rapport à un ensemble de dépendances fonctionnelles.

Algorithme 1 : *Fermeture*(Σ, R, X)

Entrées : Σ un ensemble de dépendances fonctionnelles, X un ensemble d'attributs, le schéma R
Sortie : X^+ (la fermeture de X par Σ)

```
1 CI := X;
2 done := false;
3 while (¬done) do
4 |   done := true;
5 |   forall W → Z ∈ Σ do
6 |   |   if W ⊆ CI and Z ⊈ CI then
7 |   |   |   CI := CI ∪ Z;
8 |   |   |   done := false;
9 return CI
```

Etant donnés $\Sigma = \{A \to D; AB \to E; BF \to E; CD \to F; E \to C\}$ et le schéma de relation $R(A, B, C, D, E, F)$ (A, B, C, D, E et F étant les attributs de la relation R), que retourne l'algorithme pour les trois cas suivants ? :

(a) $X = F$

(b) $X = BF$

(c) $X = ABF$

Que peut-on dire de l'ensemble d'attributs ABF ?

10. Soit le lemme suivant : $\Sigma \models X \to Y$ si et seulement si $Y \subseteq X^+$.
    On considère l'ensemble Σ de dépendances fonctionnelles suivant défini sur le schéma de relation $R(A, B, C, D, E, F, G)$:

$$\Sigma = \{A \to BCD; ABE \to G; CD \to E; EG \to ABD; BE \to F; FG \to A\}.$$

    (a) **Montrer que $\Sigma \models A \to F$.**

    (b) **Montrer que BEF n'est pas une clé de R.**

    On s'intéresse maintenant aux propriétés algébriques de la fermeture. En algèbre, on appelle *fermeture* une application $\phi : \wp(E) \to \wp(E)$ où $\wp(E)$ désigne l'ensemble des parties de E, qui est :

    Extensive : $X \subseteq \phi(X)$

    Croissante : $X \subseteq Y \Rightarrow \phi(X) \subseteq \phi(Y)$

    Idempotente : $\phi(\phi(X)) = \phi(X)$

11. **Montrer que la fermeture X^+ est bien une fermeture au sens algébrique du terme.**

4 Equivalence de systèmes d'inférence

On dit qu'un système d'inférence est complet si toutes les dépendances valides peuvent être inférées par ce système. Le système d'inférence d'Armstrong est complet.

L'objectif de cette partie est de trouver un système d'inférence correct et complet de plus petite taille que le système d'Armstrong.

Considérons la nouvelle règle d'inférence suivante :

$$\text{Pseudo-transitivité } (\sigma_P) : \frac{X \to Y \qquad WY \to Z}{WX \to Z}$$

12. **Soit Σ un ensemble de dépendances fonctionnelles, montrer que toute preuve de $\Sigma \models X \to Y$ utilisant la règle σ_P peut être transformée en une preuve n'utilisant que σ_A et σ_T.**

13. **Montrer que toute preuve de $\Sigma \models X \rightarrow Y$ utilisant les règles σ_{Re}, σ_A et σ_T peut être transformée en une preuve n'utilisant que σ_{Re} et σ_P.**

14. **En déduire que le système $\{\sigma_{Re}, \sigma_P\}$ est correct et complet pour l'inférence des dépendances fonctionnelles.**