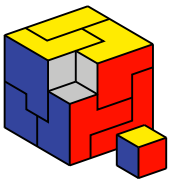
Prologin 2024



Concours national d'informatique

Épreuve écrite d'algorithmique
Lyon Aix Bordeaux

18 février 2024

Bio-login

1 Préambule

Bienvenue à **Prologin**. Ce sujet est l'épreuve écrite d'algorithmique et constitue la première des trois parties de votre épreuve régionale, sa durée est de 3 heures. Pendant cette épreuve, vous passerez un entretien (10 minutes) avec un organisateur. Ensuite, vous aurez une épreuve de programmation sur machine (3 heures et 30 minutes) l'après-midi.

Conseils

— Lisez bien tout le sujet avant de commencer.

— **Soignez la présentation** de votre copie.

— N'hésitez pas à poser des questions.

— Si vous avez fini en avance, relisez bien.

— N'oubliez pas de passer une bonne journée.

Remarques

— Le barème est donné à titre indicatif uniquement.

— Indiquez lisiblement vos nom et prénom, la ville où vous passez l'épreuve et la date en haut de votre copie.

— Lorsqu'un algorithme est demandé, vous pouvez le décrire avec suffisamment de précision, le pseudo-coder ou l'implémenter dans le langage de votre choix. Dans le dernier cas, veuillez néanmoins préciser le langage que vous utilisez.

— Ce sont des humains qui lisent vos copies : laissez une marge, aérez votre code, ajoutez des commentaires (**seulement** lorsqu'ils sont nécessaires) et évitez au maximum les fautes d'orthographe.

— Le barème récompense les algorithmes les plus efficaces : écrivez des fonctions qui trouvent la solution le plus rapidement possible.

— Si vous trouvez le sujet trop simple, relisez-le, réfléchissez bien, puis dites-le-nous, nous pouvons ajouter des questions plus difficiles.

2 À propos du sujet

Ce sujet est composé de 3 parties **indépendantes** :

— Similarité entre séquences ADN

— Recherche de sous-séquences ADN

— Analyse phylogénétique

Si vous vous retrouvez bloqué sur une partie, n'hésitez pas à passer à la suivante ou à demander de l'aide.

3 Introduction

Alice Marchand a retrouvé un viking congelé au nord de la Norvège. Elle trouve une rune où est inscrit "Cigît, Jøsëf Marchand, l'homme ayant prouvé P = NP". Afin d'avoir la réponse à cette conjecture existentielle, elle veut le recréer à partir de son ADN. Pour cela, elle a besoin de plusieurs algorithmes afin de le séquencer.

4 Similarité entre séquences ADN

4.1 Sélection d'ADN

Beaucoup de données d'ADN ont été récupérées, avant de se lancer dans l'analyse des données il faut tout d'abord les pré-traiter. Pour cela, l'ordinateur d'Alice renvoie une liste composée de plusieurs entiers représentant la fiabilité de chaque échantillon. Afin de trouver une suite correcte d'échantillons, Alice s'intéresse à retrouver la suite dont la somme est maximale.

L'algorithme ci-dessous permet de trouver le sous-tableau [1] ayant la plus grande somme de tout le tableau d'entrée.

Algorithme 1 : Somme maximale

```
Entrées : Un tableau A de taille N
Résultat : La somme maximale
somme = 0 ;
maximum = 0 ;
pour i = 1 à N faire
    somme += A[i] ;
    si somme < 0 alors
        somme = 0 ;
    fin
    maximum = max(maximum, somme) ;
fin
retourner maximum
```

Question 1 (2 points)

> Quel est le résultat de l'algorithme sur $A = [-4, 1, -3, 4, -1, 2, 1, -5, 2]$?

Question 2 (1 point)

> Comment modifier l'algorithme pour avoir les résultats de l'algorithme lancé sur chaque préfixe de A sous forme d'un tableau D ? (Tel que $D[i]$ soit la somme maximale d'un sous tableau dans $A[1 : i]$).

Question 3 (2 points)

Supposons que nous ayons un tableau de N éléments tel que $D[i]$ est la valeur de *somme* à la fin de la i-ème itération.

Par exemple, nous obtenons $D = [0, 1, 0, 4, 3, 5, 6, 1, 3]$ pour l'exemple précédent.

> Comment savoir où se termine le sous-tableau maximisant la somme ?

Question 4 (3 points)

> Proposer une méthode pour trouver également le début de ce sous-tableau.

4.2 Distance caractère par caractère

Alice possède maintenant plusieurs échantillons d'ADN, chaque échantillon S_i possède N nucléotides [2] parmi $\{A, T, G, C\}$.

Les échantillons n'étant pas parfaits, elle veut en un premier temps voir quels sont les échantillons se ressemblant le plus.

Pour cela, elle trouve un algorithme permettant de mesurer la similarité entre deux séquences nommé la "distance de Hamming". La distance entre deux chaînes A et B est le nombre total de positions i tel que $A[i] \neq B[i]$ [4].

1. Un sous-tableau S est une partie d'un tableau A que nous pouvons écrire sous forme $A[i : j]$, c'est-à-dire qu'il est constitué des caractères de A à partir de l'indice i à l'indice j dans le même ordre
2. mot compliqué pour dire "caractère" [3]
3. mot compliqué pour dire "lettre"
4. $A[i]$ est le i-ème caractère de A

Question 5 (1 point)

> Calculer la distance entre les échantillons :
>
> 1. "TAC" et "TAG"
> 2. "CATA" et "AGAT"
> 3. "TATATA" et "ATATAT"

Question 6 (2 points)

> Coder un algorithme permettant de calculer cette distance entre deux chaînes A et B de N caractères.

Question 7 (1 point)

> Alice ne trouve pas cet algorithme pertinent pour son utilisation, pourquoi ? Détailler avec au moins deux problèmes liés à cet algorithme.

4.3 Distance d'édition

Alice continue ses recherches, elle trouve un algorithme sur la similarité entre deux séquences prenant en compte l'édition de celles-ci comme dans un éditeur de texte.

La distance entre deux chaînes A et B de tailles N et M respectives est le nombre de fois au minimum que nous devons supprimer ou ajouter un caractère à l'une des chaînes pour la rendre égale à l'autre chaîne.

Par exemple, la distance entre "GATA" et "CAT" est 3 car nous ajoutons / supprimons 3 caractères au minimum pour passer d'une chaîne à l'autre :

1. "**G**ATA" → "ATA" via suppression
2. "AT**A**" → "AT" via suppression
3. "AT" → "**C**AT" via ajout

La distance entre "TAC" et "TAG" est 2, celle entre "TAAC" et "TAC" est 1 et celle entre "AAA" et "TTT" est 6.

Au lieu d'ajouter et de supprimer des caractères à l'une des chaînes, nous pouvons seulement supprimer des caractères aux deux chaînes jusqu'à les rendre égales.

Question 8 (2 points)

> Refaire les étapes de la distance d'édition comme ci-dessus avec "GATA" et "CAT", cette fois-ci en ne faisant que l'action de supprimer un caractère sur n'importe quelle chaîne (à chaque étape la suppression peut se faire sur A ou sur B). En d'autres mots, illustrer les étapes pour obtenir le nombre minimum de suppressions entre les deux chaînes.
>
> Par exemple, la distance entre "AT" et "T" est 1 car il y a une opération de suppression "**A**T" → "T" pour que les deux chaînes deviennent égales.

Question 9 (4 points)

Alice veut à présent implémenter cet algorithme de manière **récursive**. Elle utilise un langage dont le nom fait penser aux animaux ayant deux bosses [5] sur le dos. Ce langage possède deux fonctions sur les chaînes de caractères :

1. **tete** : Permet d'obtenir le premier caractère de la chaîne ($tete("hello") = "h"$)
2. **queue** : Permet d'obtenir la chaîne sans le premier caractère ($queue("hello") = "ello"$)

> En utilisant ces fonctions ainsi que les comparaisons, compléter l'algorithme **editionsimple** calculant la distance d'édition avec ajout / suppression de caractères entre A et B. *L'algorithme doit être complété sur votre feuille et non le sujet.*

5. et non une seule

Algorithme 2 : Distance d'édition par ajout / suppression, fonction **editionsimple**

Entrées : Une chaîne A et une chaîne B
Résultat : La distance d'édition avec ajout / suppression entre A et B
si *A vide ou B vide* **alors**
  retourner taille A + taille B
fin
si *tete* A = *tete* B **alors**
  retourner ??? `// Complétez le code (1)`
fin
sinon
  retourner ??? `// Complétez le code (2)`
fin

Question 10 (2 points)

> Quelle est la classe de complexité temporelle de cet algorithme sachant que les chaînes ont pour longueur respective N et M ?

1. Linéaire
2. Quadratique
3. Exponentielle

Question 11 (1 point)

> Donner un exemple maximisant le nombre d'appels de fonctions avec $N \leq 3$ et $M \leq 3$.

Question 12 (2 points)

Alice trouve que les chaînes "CAT" et "CAG" devraient avoir une distance de 1 et non de 2. En effet, elle voudrait ajouter l'action de changer un caractère. Elle trouve qu'en ajoutant cette action, l'algorithme se nomme "distance de Levenshtein".

> Proposer une modification de l'algorithme au niveau du commentaire "Complétez le code (2)" afin d'y ajouter cette action.

Question 13 (2 points)

Cet algorithme est coûteux, Alice voudrait l'optimiser car elle trouve que certains résultats intermédiaires sont recalculés inutilement. Un résultat intermédiaire représente un appel de fonction, $puissance2(42)$ et $puissance2(40+2)$ peuvent être représentés par un seul résultat intermédiaire d'une valeur de 42^2 par exemple.

> Donner un ordre de grandeur sur le nombre de résultats intermédiaires maximum en fonction de N et M pour calculer le résultat total.

Question 14 (2 points)

Afin d'implémenter l'algorithme optimisé, Alice veut le faire en plusieurs étapes :

1. Initialisation du tableau multi-dimensionnel de résultats intermédiaires
2. Remplissage du tableau
3. Obtenir le résultat global

Le tableau de résultats intermédiaires contient les résultats d'un appel récursif. Par exemple, si nous voulons compter jusqu'à N de manière récursive, nous pouvons avoir un tableau à N éléments tel que :

1. Initialisation : Tableau T d'une dimension de N éléments, le premier élément est 1.
2. Transitions : $T_{i+1} = 1 + T_i$, avec i allant de 1 à $N - 1$
3. Résultat : Le résultat est T_N

> Combien de dimensions doit avoir le tableau D et quelles sont les tailles de chaque dimension ?

Question 15 (1 point)

> Comment initialiser ce tableau ?

Question 16 (4 points)

> Comment obtenir le résultat intermédiaire grâce à un ou plusieurs résultats intermédiaires précédents ? Le calcul doit être de complexité constante. Ce calcul peut être décomposé en deux sous-calculs dans le cas où *tete* A = *tete* B.

Question 17 (1 point)

> Quelle est donc la complexité de cet algorithme optimisé en temps et en mémoire, en fonction de N et M ?

4.4 Alignement de séquences ADN

A présent, Alice se renseigne sur l'algorithme Smith-Waterman pour pouvoir aligner les différentes séquences d'ADN. Cet algorithme permet de trouver comment aligner deux séquences afin de maximiser un score. "Aligner" signifie trouver quels éléments sont associés deux-à-deux dans les deux séquences.

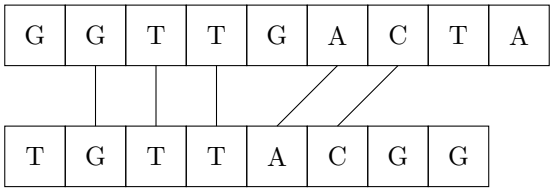
Le score est la somme de deux composantes :

1. **Les points de similarité** : Compare deux éléments des deux séquences et donne des points en fonction du résultat de la comparaison.
2. **La pénalité d'espacement** : Renvoie un score pénalisant si nous ignorons un élément d'une séquence.

Voici un exemple d'utilisation de l'algorithme avec ces fonctions :

1. Similarité : Si les deux caractères sont égaux, nous avons 3 points sinon -3 points.
2. Espacement : Nous avons 2 points en moins par caractère ignoré.

Avec les deux séquences d'entrée "GGTTGACTA" et "TGTTACGG" nous trouvons cet alignement maximisant le score :



Nous notons que nous pouvons ignorer les éléments avant et après l'alignement dans la notation du score.

Il y a 5 alignements donnant 3 points chacun soit 15 points de similarité et 2 points de pénalité d'espacement. Au total le score est donc de 13.

Il est ainsi possible de noter les deux alignements obtenus "GTT-AC" et "GTTAC".

Question 18 (2 points)

> Avec "AGAGCT" et "AACT", quel est le score total et la représentation des deux alignements avec pour similarité 2 ou -2 points si les caractères sont égaux ou non et 1 point en moins par caractère ignoré ?

Question 19 (1 point)

Alice voudrait à présent commencer l'algorithme, elle prévoit d'utiliser un tableau bi-dimensionnel pour stocker tous les scores intermédiaires. Les deux séquences sont A et B et leurs longueurs respectives sont N et M.

Elle initialise alors un tableau à deux dimensions D de $(N+1) \times (M+1)$ cases tel que $D_{i,j}$ = score de l'alignement entre les préfixes [6] $A[:i]$ et $B[:j]$.

6. un préfixe est composé des K $(0 \leq K \leq N)$ premiers caractères de la séquence

> De quels éléments du tableau D la valeur $D_{i,j}$ dépend-elle ?

Question 20 (4 points)

> Montrer comment calculer $D_{i,j}$ en fonction des valeurs énoncées précédemment ainsi que la mesure de similarité et de la pénalité d'espacement.

Question 21 (2 points)

Une fois que toutes les valeurs sont calculées, le score optimal dans le tableau est alors l'élément maximum. Nous supposons qu'il n'y a qu'un seul alignement optimal.

> A quoi correspond la position du score optimal dans le tableau ?

Question 22 (5 points)

> Proposer une manière de reconstruire les deux représentations des alignements résultants. Vous pouvez utiliser un tableau bi-dimensionnel supplémentaire.

5 Recherche de sous-séquences ADN

Alice décide à présent de coder un célèbre algorithme afin de pouvoir chercher efficacement des sous-séquences parmi les échantillons qu'elle possède.

Dans cette section, Alice va construire l'algorithme de Knuth-Morris-Pratt pas à pas. Certaines questions peuvent être difficiles, n'hésitez pas à les passer.

Notations

- Dans cette partie, on appelle un *mot* ou *chaîne de caractères* une séquence (potentiellement vide) de lettres majuscules parmi les 26 lettres de l'alphabet français [7]. On appelle Σ l'ensemble des 26 lettres de l'alphabet, c'est à dire $\Sigma = \{\text{«A»}, \text{«B»}, \ldots, \text{«Z»}\}$.
- On écrira $|w|$ pour représenter la longueur d'une chaîne de caractères w. Par exemple, avec $w =$ «PROLOGIN», on a $|w| = 8$. La chaîne de caractères vide $\varepsilon =$ «» a une longueur de $|\varepsilon| = 0$.
- Les chaînes de caractères sont basées à 0, ainsi dans cette section $w[0]$ représente le premier caractère de w, en « position 0 », et $w[|w|-1]$ représente le dernier caractère de w. Par exemple, avec $w =$ «PROLOGIN», on a $w[0] =$ «P», $w[1] =$ «R», $\ldots$, $w[7] =$ «N».
- On note $w[i..j]$ la sous-chaîne de caractères allant du caractère en position i inclus au caractère en position j exclus. Par exemple, avec $w =$ «PROLOGIN», on a $w[3..6] =$ «LOG».
- On note $u \mid\mid v$ la concaténation des chaînes de caractères u et v. Par exemple, si $u =$ «PRO» et $v =$ «LOGIN», on a $u \mid\mid v =$ «PROLOGIN».
- Dans les blocs de pseudo-code donnés, les deux bornes des boucles `pour` sont toujours incluses.

Question 23 (1 point)

- On dit qu'un mot u est un *préfixe* d'un mot w si w commence par u. Autrement dit, u est un préfixe de w s'il existe un i tel que $w[0..i] = u$.
- Similairement, on dit qu'un mot u est un *suffixe* d'un mot w si w se finit par u. Autrement dit, u est un suffixe de w s'il existe un j tel que $w[j..|w|] = v$.
- Finalement, on dit qu'un mot u est un *facteur* d'un mot w si u est inclus dans w. Autrement dit, u est un facteur de w s'il existe un i et un j tel que $w[i..j] = u$.

Considérez la chaîne de caractères $w =$ «AAABA». On a alors «AAAB» qui est un des préfixes de w, «ABA» un des suffixes de w, et «AAB» un des facteurs de w.

On appelle $\text{Pref}(w)$ l'ensemble des préfixes de w, $\text{Suff}(w)$ l'ensemble des suffixes de w et $\text{Fact}(w)$ l'ensemble des facteurs de w.

Par exemple, on a $\text{Pref}(w) = \{\varepsilon, \text{«A»}, \text{«AA»}, \text{«AAA»}, \text{«AAAB»}, \text{«AAABA»}\}$.

> Combien existe-t-il de facteurs de w ? Indiquez l'ensemble des facteurs de w.

Question 24 (1 point)

> Considérez la chaîne de caractères $w =$ «AAABAACAAAABAA». Indiquez tous les préfixes de w qui sont également des suffixes de w.

Question 25 (2 points)

Pour une chaîne de caractères w, on appelle *indice LPS* la taille du plus long préfixe de w différent de w qui est également un suffixe de w. Par exemple, l'indice LPS de $w =$ «AABAABA» est 4, car le plus long préfixe différent de w qui est également un suffixe est $u =$ «AABA», de longueur 4.

> Pour chaque préfixe $u \neq \varepsilon$ de $w =$ «AAABAACAAAABAAA», indiquez l'indice LPS de u.

7. L'imprimante n'avait malheureusement pas assez d'encre pour inclure également les caractères de l'alphabet runique

Question 26 (2 points)

> Soit une chaîne u dont le préfixe de taille k est un suffixe de u. Soit $w = u \mid\mid u[k]$. Justifiez que le préfixe de taille $k+1$ de w est un suffixe de w.

Question 27 (2 points)

> Soient une chaîne u dont l'indice LPS est égal à k, et x n'importe quel caractère. Justifiez que l'indice LPS de $w = u \mid\mid x$ est inférieur ou égal à l'indice LPS de $u + 1$.

Question 28 (1 point)

> Supposons que nous ayons une chaîne u d'indice LPS égal à k. Quel est l'indice LPS de $w = u \mid\mid u[k]$? Justifiez.

Question 29 (3 points)

> On appelle le tableau LPS de u, noté LPS$[i]$, le tableau indiquant pour tout i allant de 1 à $|u|$ l'indice LPS de $u[0..i]$. La question précédente nous permet de trouver l'indice LPS de $u \mid\mid u[\text{LPS}[|u|]]$. Qu'en est-il de l'indice LPS de $w = u \mid\mid u[\text{LPS}[\text{LPS}[|u|]]]$?

Question 30 (4 points)

> En étendant ce principe, proposez un algorithme qui calcule le tableau LPS d'un mot w, avec une complexité temporelle de $O(|w|)$. Prouvez la complexité de votre algorithme.

Question 31 (2 points)

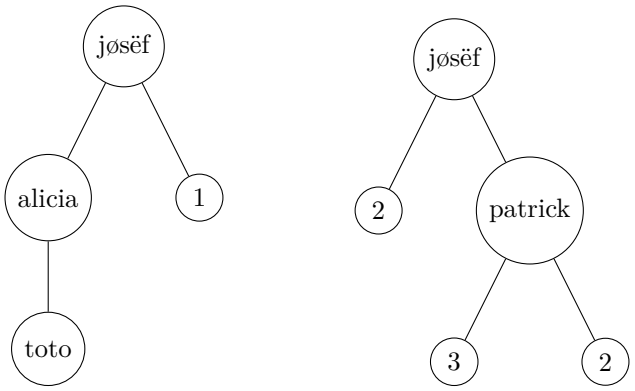
> En supposant que vous pouvez calculer le tableau LPS de n'importe quelle chaîne w avec une complexité temporelle de $O(|w|)$, proposez un algorithme qui permet de trouver toutes les occurrences d'une sous-chaîne u dans une plus grande chaîne v avec une complexité temporelle de $O(|v|)$.

6 Analyse phylogénétique

Plusieurs chercheurs ont abouti à des théories sur l'arbre phylogénétique de Jøsëf Marchand ! Aidez-les à les mettre en commun pour générer l'arbre phylogénétique de Jøsëf.

6.1 Les arbres phylogénétiques

On considère ici des arbres enracinés dont les noeuds sont étiquetés par des noms d'espèces ou des entiers qui représentent des inconnus. Par exemple, voici deux arbres qui correspondent à des résultats de deux chercheurs différents :



Il est à noter que les feuilles ne sont pas nécessairement des inconnues (par exemple la feuille *toto* dans l'arbre 1), mais les noeuds internes seront nécessairement des noms d'espèces. Un nom d'espèce, quelque soit l'arbre, a toujours le même nombre de fils.

On pourra représenter ces arbres sous la forme d'une chaîne de caractères w tel que :

1. La chaîne n'est composée que d'un mot si l'arbre n'est composé que d'un noeud sans fils
2. La chaîne est de la forme $w(x_1, ..., x_n)$ si la racine est étiquetée par w et que chacun des n fils de gauche à droite sont représentés par les chaînes $x_1, ..., x_n$

Ainsi, on a la chaîne `jøsëf(alicia(toto),1)` qui représente le premier arbre.

Question 32 (1 point)

> Donner la chaîne représentant les résultats du second scientifique.

Question 33 (2 points)

> Donner un algorithme qui, à partir d'un arbre, associe la liste de toutes ses inconnues différentes. Par exemple, pour l'entrée représentée par le texte `jøsëf(simple(1,2),2)`, l'algorithme renverra la liste $[1, 2]$.

Question 34 (3 points)

> Donner un algorithme qui, à partir d'une chaîne de caractères, reconstruit l'arbre qu'elle représente. Ainsi `jøsëf(alicia(toto),1)` donnera l'arbre du premier scientifique. On indiquera quelle structure de donnée a été choisie pour représenter les arbres.

Question 35 (3 points)

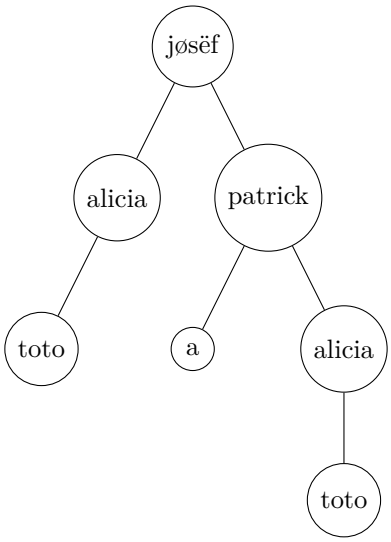
> Mince ! Des chercheurs différents ont utilisé les mêmes inconnues. Proposer un algorithme qui prend en argument une liste d'arbres phylogénétiques et qui renvoie les mêmes arbres où les inconnues ont été modifiées de telle sorte à ce que deux chercheurs différents n'ont pas d'inconnue en commun.
>
> Par exemple, avec la liste [`jøsëf(1,madeline(2,1),3)` , `jøsëf(theo(1),2,2)`], on pourra renvoyer [`jøsëf(1,madeline(2,1),3)` , `jøsëf(theo(4),5,5)`]

6.2 Des systèmes de ré-écriture de termes

Ici, on s'intéresse à trouver les plus petits arbres phylogénétiques $(A_n)_n$ tel qu'en remplaçant toutes les inconnues dans tous les arbres des scientifiques, on arrive à un consensus (si c'est possible). Plus précisément,

on cherche une liste de couples (x, A_x) telle que si on remplace chaque inconnue x par son A_x correspondant dans tous les arbres phylogénétiques de chaque scientifique, on obtient le même arbre phylogénétique. Une telle liste est appelée une *solution du problème d'unification*.

Par exemple, $[(1, \texttt{patrick(a,alicia(toto))}), (2, \texttt{alicia(toto)}), (3, \texttt{a})]$ est une solution au problème d'unification des deux arbres du premier exemple donné en 6.1, car si on remplace l'inconnue 1 par `patrick(a,alicia(toto))`, l'inconnue 2 par `alicia(toto)` et l'inconnue 3 par `a` on obtient l'arbre `jøsëf(alicia(toto),patrick(a,alicia(toto)))` :



L'arbre final ne doit plus contenir d'inconnue.

Question 36 (2 points)

> Donner un algorithme `substitution` qui prend en arguments un arbre A et une liste de couples (x, A_x) et qui renvoie un arbre où toutes les inconnues i sont remplacées par A_i si (i, A_i) est dans la liste.

On appelle *problème d'unification* un couple de deux arbres (A_1, A_2) pour lesquels on cherche une liste de couples (x, A_x) tel que $\texttt{substitution(}A_1\texttt{,}L\texttt{)} = \texttt{substitution(}A_2\texttt{,}L\texttt{)}$

Question 37 (1 point)

> Pour chacune des paires d'arbres suivantes, proposer une solution au problème d'unification si c'est possible :
>
> 1. `a(1,2,2)` et `a(3,3,4)`
> 2. `1` et `a(1)`
> 3. `a(g,f(1,1),f(2,2))` et `a(1,2,3)`
>
> Si deux inconnues ont le même nom, elles doivent alors être remplacées par la même chose.

On essaye maintenant de résoudre le problème d'unification entre deux termes (deux arbres phylogénétiques). On pourra supposer que le noeud `a` n'apparaît dans aucun des arbres d'entrée.

Question 38 (6 points)

> Proposer un algorithme qui prend deux arbres A_1 et A_2 et qui résout le problème d'unification en renvoyant un couple $(\texttt{VRAI}, L)$ avec L la liste des substitutions s'il est résoluble, et $(\texttt{FAUX}, [])$ sinon.

7 Bonus

Question bonus 39 (1 point)

> Quels langages de programmation possèdent la propriété de l'homoïconicité ?

Question bonus 40 (1 point)

> Indiquer le nombre préféré de l'un des organisateurs.

Question bonus 41 (1 point)

> Proposer un algorithme prenant un entier N d'une complexité quadratique permettant de calculer 42. Attention, l'algorithme doit être écrit à l'envers.

Question bonus 42 (1 point)

> Écrire un algorithme sans lever le crayon.