Banque MP inter-ENS – Session 2022

Rapport relatif à l'épreuve orale d'informatique fondamentale LCR

- Ecoles partageant cette epreuve : ENS Paris-Saclay, ENS Rennes, ENS de Lyon
- Coefficients (en pourcentage du total des points de chaque concours) :
  - ENS Paris-Saclay
    - Concours MP Option MPI : 13.2 %
    - Concours MP Option MP : 23.1 %
  - ENS Rennes
    - Concours MP Option MPI : 8.6 %
    - Concours MP Option MP : 23.1 %
  - ENS de Lyon :
    - Concours MP Option MPI : 14.1 %
    - Concours MP Option MP : 10.8 %
- Membres du jury : A. Pouly, B. Simon, R. Demangeon, S. Le Roux

L'épreuve orale d'informatique fondamentale concerne les candidats aux ENS de Lyon, Paris-Saclay, et Rennes des concours MPI et Informatique.

Cette année 214 candidates et candidats ont participé à l'épreuve. Leurs notes sont comprises entre 4 et 20, avec une moyenne de 13.49 et un écart-type de 3.16. L'histogramme de la figure 1 présente la distribution des notes.

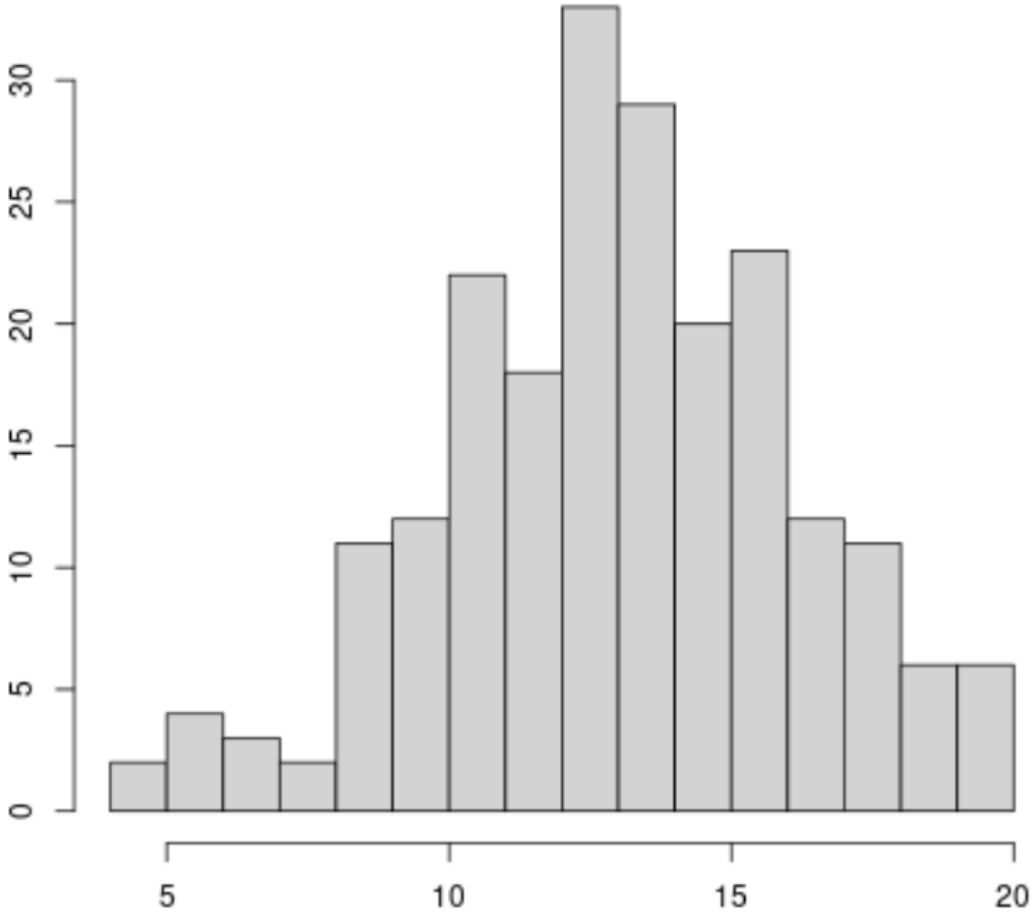

FIGURE 1 – Histogramme des notes de l'épreuve

Après avoir reçu un sujet, les candidats disposaient de 30 minutes de préparation, suivies de 27 minutes d'interrogation devant un des examinateurs. En effet, 3 minutes sont utilisées par l'examinateur pour aller chercher le candidat en salle de préparation et lui rappeler les conditions de l'évaluation : cette évaluation se base principalement sur la progression des candidats dans les questions, mais aussi sur la pédagogie de leur présentation et l'autonomie dont ils ou elles ont fait preuve pendant l'oral.

Le jury a proposé 24 sujets originaux, présentés en annexe. Cela représente une moyenne de 9 candidats sur chaque sujet, ce qui permet aux membres du jury d'effectuer une meilleure harmonisation des évaluations, notamment via une pondération des questions selon leur difficulté.

Chaque sujet débute par un énoncé qui présente un problème d'informatique et introduit ses notations, puis comporte entre six et neuf questions de difficulté globalement croissante. Une ou deux premières questions relativement faciles d'accès permettent aux candidats de vérifier leur compréhension de l'énoncé et de « se lancer » dans l'interrogation orale. Les questions suivantes, progressivement plus difficiles, occupent la majeure partie du temps de préparation et d'interrogation, et visent à permettre de départager les candidats. Enfin, la plupart des sujets terminent par une ou plusieurs questions considérées comme très difficiles. En général, il n'est pas attendu des candidats qu'ils traitent l'ensemble des questions dans le temps imparti.

Les sujets font appel aux différentes compétences nécessaires en science informatique : comprendre des concepts nouveaux, démontrer des résultats théoriques, et construire des solutions techniques telles que des algorithmes. Si l'épreuve d'informatique *fondamentale* met l'accent sur les concepts théoriques de l'informatique, on veille néanmoins à garder à l'esprit le sens des objets que l'on étudie.

Par contraste avec les constats formulés lors de la session précédente, en 2021, le jury a constaté avec satisfaction que la plupart des candidats affichait une bonne maîtrise des concepts mathématiques (dénombrement, distinction de cas, principes d'induction, ...). Au contraire, quelques candidats ont fait montre d'une moins bonne vision algorithmique (invention d'une solution, déduction d'un processus calculatoire à partir d'une équation).

Les énoncés proposés cette année portaient sur des thèmes assez variés : mots et langages, ordres, logique informatique, combinatoire des graphes, sécurité au sens large, etc. Une majorité des sujets présente un aspect algorithmique, de manière dominante ou anecdotique.

Le jury adresse les conseils suivants aux futurs candidats :

— Si le principe même d'un concours nous impose de faire un classement et que l'épreuve orale peut sembler intimidante, nous tenons à rappeler que *tous* les candidats et *toutes* les candidates admissibles ont un excellent niveau et peuvent aborder l'oral avec confiance. Les candidats ne doivent en aucun cas se dévaloriser ou s'auto-censurer sur la base de leur parcours, de leurs notes en classes préparatoires, ou de ce qu'ils estiment être leur position dans le classement !

— L'objectif du jury est d'amener chaque candidat à réussir à traiter au mieux le maximum de questions. Il est donc dans l'intérêt des candidats d'écouter les demandes et les indications de l'examinateur et d'en tenir compte.

— Lors de la préparation, lire l'ensemble du sujet est indispensable : cela permet au candidat de comprendre la direction générale de l'exercice et d'identifier des questions éventuellement plus simples à traiter. Dès le début de l'interrogation, il faut ensuite préciser à l'examinateur quelles questions on a traitées, ne serait-ce que partiellement. Lors de l'interrogation, il ne faut pas non plus hésiter à proposer à l'examinateur de sauter ou remettre à plus tard des questions pour avoir le temps de présenter l'ensemble des résultats préparés. Cette année, plusieurs candidats sont arrivés au terme des 27 minutes imparties avant d'avoir pu présenter toutes leurs solutions.

— Deux écueils à éviter sont donc, d'une part, de traiter de manière trop superficielle les démonstrations et les algorithmes, en passant à côté de points techniques importants, et inversement, de détailler excessivement les réponses de manière trop formelle et manquer de temps pour traiter suffisamment de questions. Nous conseillons aux candidats de soigner la forme des réponses aux premières questions, plus faciles, quitte à prendre progressivement de la hauteur pour se concentrer sur le fond dans les questions difficiles. Dans tous les cas, on prendra soin de présenter brièvement la démarche ou l'intuition suivie avant de se lancer dans une démonstration, un calcul ou un algorithme.

— Lorsque l'on demande au candidat de proposer un algorithme ou une preuve, on attend de lui qu'il nous en présente d'abord la structure générale et les étapes principales, avant de détailler celles-ci si l'examinateur en fait la demande. En particulier, il n'est **jamais** attendu du candidat qu'il commence par écrire un pseudo-code au tableau.

— Les Écoles Normales Supérieures forment à la recherche, et le candidat ne doit donc pas hésiter à chercher au tableau : faire des dessins, traiter des exemples simples que suggèrent les premières questions ou que le candidat aura inventées lui-même, faire des essais qui seront soit concluants, soit porteurs d'une information nouvelle.

— La pédagogie est valorisée : le candidat ne doit pas cacher le tableau à l'examinateur, doit exprimer clairement ses arguments, écrire si besoin les étapes-clés au tableau mais ne pas se perdre dans des détails inutiles afin de montrer que l'argument principal est identifié.

— Certains sujets proposent une ouverture finale sur un nouveau concept, suivie d'une question facile afin de vérifier que le concept est compris avant une question plus intéressante. Avoir fait seulement cette question facile sans avoir réflechi à la question suivante ni aux questions difficiles de la première partie n'est pas valorisé.

En annexe, nous présentons les sujets de cette année :

Robots d'échange

On dispose d'un ensemble d'*objets* abstraits $a, b, c, \ldots$.
Un *robot* $\mathcal{R}$ est un ensemble d'*échanges*, chacun de la forme $(a_1, \ldots, a_n) \triangleright (b_1, \ldots, b_m)$ qu'il faut comprendre par "si on fournit les objets $a_1, \ldots, a_n$ au robot, il nous donne les objets $b_1, \ldots, b_m$". Un échange est non-vide (c'est-à-dire $n + m > 0$). Le même objet peut apparaître plusieurs fois dans un échange.
Un *stock* d'objets est défini par un *multiensemble fini* $\{s_1, \ldots, s_k\}$, c'est-à-dire, par une collection **finie, non ordonnée** d'objets dans laquelle un même objet peut apparaître plusieurs fois.

On peut échanger une partie (ou la totalité) d'un stock avec le robot (selon les échanges dont il est composé), puis recommencer tant que cela est possible.

Formellement, si on a un robot $\mathcal{R} = \{(a_1, \ldots, a_n) \triangleright (b_1, \ldots, b_m)\} \cup \mathcal{R}'$, le stock

$$\mathcal{H} = \{a_1, \ldots, a_n, c_1, \ldots, c_k\}$$

peut *se réduire* vers le stock

$$\mathcal{H}' = \{b_1, \ldots, b_m, c_1, \ldots, c_k\}$$

(on a donné les objets $a_1, \ldots, a_n$ présents dans le stock et récupéré les objets $b_1, \ldots, b_m$, conformément à l'échange).
Dans ce cas, on note $\mathcal{H} \longrightarrow_{\mathcal{R}} \mathcal{H}'$ la relation de réduction.
Quand il est clair de quel $\mathcal{R}$ on parle, on s'autorise à écrire $\mathcal{H} \longrightarrow \mathcal{H}'$.
On note $\longrightarrow^*$ la clotûre reflexive et transitive de $\longrightarrow$ (la plus petite relation réflexive et transivite qui contient $\longrightarrow$).

Question 1. Soit le robot $\mathcal{R} = \{(a) \triangleright (b, b, b), (b) \triangleright (c, c)\}$ et le stock $\mathcal{H} = \{a, b, b, c\}$)

Donner une séquence de réductions depuis $\mathcal{H}$ selon $\mathcal{R}$ jusqu'à obtenir un stock duquel plus aucune réduction n'est possible.

Question 2. Soit le robot $\mathcal{R} = \{(a) \triangleright (a), (a, a) \triangleright (b)\}$ et le stock $\mathcal{H} = \{a, a\}$

Etudier l'ensemble des suites de réductions possibles depuis $\mathcal{H}$ selon $\mathcal{R}$.

Un robot $\mathcal{R}$ est *non-déterministe* quand il existe trois stocks $\mathcal{H}, \mathcal{H}_1, \mathcal{H}_2$ tels que :

- $\mathcal{H} \longrightarrow_{\mathcal{R}} \mathcal{H}_1$
- $\mathcal{H} \longrightarrow_{\mathcal{R}} \mathcal{H}_2$
- $\mathcal{H}_1 \neq \mathcal{H}_2$

Question 3. Donner un exemple de robot non-déterministe, et un exemple de robot déterministe.

Question 4. Trouver une condition nécessaire et suffisante pour qu'un robot qui contient au moins deux échanges soit non-déterministe.

Un robot $\mathcal{R}$ est *concurrent* quand il existe trois stocks $\mathcal{H}, \mathcal{H}_1, \mathcal{H}_2$ tels que :

- $\mathcal{H} \longrightarrow_{\mathcal{R}} \mathcal{H}_1$
- $\mathcal{H} \longrightarrow_{\mathcal{R}} \mathcal{H}_2$
- $\mathcal{H}_1 \neq \mathcal{H}_2$
- il n'existe pas de stock $\mathcal{H}_f$ tel que $\mathcal{H}_1 \longrightarrow^*_{\mathcal{R}} \mathcal{H}_f$ et $\mathcal{H}_2 \longrightarrow^*_{\mathcal{R}} \mathcal{H}_f$

Question 5. Montrer que les robots des questions 1 et 2 ne sont pas concurrents.

Question 6. Montrer qu'un robot $\{(a_i) \triangleright (M_i)\}_i$ tel que les a_i sont différents deux-à-deux est non-concurrent.

Donner un exemple de robot concurrent.

Un robot $\mathcal{R}$ est divergent quand il existe une suite de stocks $(\mathcal{H}_n)_{n\in\mathbb{N}}$ telle que :
$\forall n \in \mathbb{N}, \mathcal{H}_n \longrightarrow_{\mathcal{R}} \mathcal{H}_{n+1}$
Un robot est terminant quand il n'est pas divergent.

Question 7. Montrer que si tous les échanges $(a_1, \ldots, a_n) \triangleright (b_1, \ldots, b_m)$ d'un robot sont tels que $n > m$, alors le robot est terminant.

Un robot de la forme $\{(a_k) \triangleright (b_{(k,1)}, \ldots, b_{(k,m_k)})\}_k$ est appelé *simple*.

Question 8. Soit $\mathcal{R}$ un robot simple. On dispose d'une fonction v qui associe à chaque objet manipulé par $\mathcal{R}$ une valeur entière positive.

Donner un critère sur v permettant de garantir que $\mathcal{R}$ est terminant. Donner la preuve de terminaison.

Question 9. Donner un algorithme qui prend en entrée un robot simple et renvoie une fonction v tel que le critère de la question précédente soit respecté quand c'est possible, et qui échoue sinon.

Sémantique axiomatique

On dispose d'un ensemble infini $\mathcal{C}$ de *nom de cellules mémoires* désignée par $x, y, z, \ldots$ Ces cellules contiennent un entier relatif.
On dispose d'une syntaxe pour des *expressions arithmétiques*, des *propositions sur les cellules* et des *formules* décrivant une mémoire.

$$\begin{array}{rcl} E & ::= & n \mid x \mid E + E \mid E - E \mid E * E \\ C & ::= & E = E \mid E \leq E \mid E < E \\ \varphi & ::= & \mathtt{T} \mid \bot \mid C \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \end{array}$$

avec $n \in \mathbb{N}, x \in \mathcal{C}$
On note $\varphi[E/x]$ pour signifier la formule φ dans laquelle toutes les occurences de la cellule x ont été remplacées par l'expression E

Le but de cet exercice est d'étudier des *triplets* de Hoare pour un langage de programmation impératif. Un triplet $\{\varphi_g\}\ P\ \{\varphi_d\}$ signifie *si la formule φ_g est vraie avant l'exécution du programme P, alors la formule φ_d est vraie après l'exécution de P.*

On commence par étudier deux commandes du langage l'inactivité `pass` et l'affectation $x \leftarrow E$. Les règles associées sont les suivantes :

- Pour tout φ, $\{\varphi\}$ `pass` $\{\varphi\}$
- Pour tout φ, $\{\varphi[E/x]\}\ x \leftarrow E\ \{\varphi\}$.
- Si $\psi \Rightarrow \varphi$ (pour les règles d'implication de la logique propositionnelles) et $\{\varphi\}\ P\ \{\varphi_d\}$, alors $\{\psi\}\ P\ \{\varphi_d\}$
- Si $\varphi \Rightarrow \psi$ et $\{\varphi_g\}\ P\ \{\varphi\}$, alors $\{\varphi_g\}\ P\ \{\psi\}$.

Question 1. Montrer que $\{\mathtt{T}\}\ x \leftarrow 2\ \{0 < x\}$.

Question 2. Montrer que $\{-1 < x\}\ x \leftarrow x + 1\ \{0 < x\}$.

Question 3. Montrer que $\{-1 < x\}$ `pass` $\{0 < x\}$.

Question 4. Montrer que $\{\bot\}\ x \leftarrow x + 1\ \{0 < x\}$.

On ajoute la séquence ; au langage $P_1; P_2$ qui représente l'enchainement, dans l'ordre, des programmes P_1 et P_2. La règle correspondante est

- Si $\{\varphi_g\}\ P_1\ \{\varphi_i\}$ et $\{\varphi_i\}\ P_2\ \{\varphi_d\}$, alors $\{\varphi_g\}\ P_1; P_2\ \{\varphi_d\}$

Question 5. Montrer que $\{x = 0\}\ x \leftarrow x + 1; x \leftarrow 2 * x\ \{x = 4\}$

Question 6. Trouver la formule φ *la plus générale* telle que $\{\varphi\}\ x \leftarrow x + 1; x \leftarrow 2 * x\ \{0 < x\}$

On ajoute le branchement `if` C `then` P_1 `else` P_2 au langage et la règle associée est :

— Si $\{\varphi_1\}\ P_1\ \{\varphi_d\}$, $\{\varphi_2\}\ P_2\ \{\varphi_d\}$, $C \wedge \varphi \Rightarrow \varphi_1$ et $\neg C \wedge \varphi \Rightarrow \varphi_2$ alors $\{\varphi\}$ `if` C `then` P_1 `else` $P_2\ \{\varphi_d\}$

Question 7. Montrer que $\{\mathtt{T}\}$ `if` $0 < x$ `then pass else` $x \leftarrow (0 - x)\ \{0 \leq x\}$

Question 8. Donner une règle pour une construction `while` C `do` $[P]$

Question 9. Montrer que $\{\mathtt{T}\}\ x \leftarrow 0; z \leftarrow y;$ `while` $0 < z$ `do` $[x \leftarrow x + 1; z \leftarrow z - 1]\ \{x = y\}$

Habitants des Types Simples

On étudie les *types* d'un langage fonctionnel, définis (dans un premier temps) par :

$$T ::= \mathbb{N} \mid T \to T$$

On considère ces types comme des *ensembles* et les éléments d'un type sont appelé *ses habitants*.
Ainsi :

- les habitants de $\mathbb{N}$ sont les entiers naturels.
- les habitants de $T_1 \to T_2$ sont les fonctions qui à chaque habitant de T_1 associe un élément de T_2.

Un type est *habité* quand il existe un habitant de ce type.

Pour décrire une fonction du langage, on utilise l'expression fonctionnelle $x \mapsto E$ dénotant la fonction qui à x associe la valeur de l'expression E. On restreindra la syntaxe de E aux expressions obtenues avec l'application d'une fonction à son argument (noté $f(x)$), les expressions fonctionnelles $((x' \mapsto E'))$ et les opérations arithmétiques simples des entiers.

Par exemple $f_0 = x \mapsto (y \mapsto 1 + x(y))$ est la fonction qui prend en entrée un x et renvoie la fonction qui prend en entrée un y et renvoie l'entier obtenu en ajoutant 1 au résultat de x appliquée à y.

Question 1. Donner deux habitants différents de chacun des types suivants :

1. $\mathbb{N} \to \mathbb{N}$
2. $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$
3. $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$

Question 2. Donner deux types habités par f_0.

Question 3. Tous les types sont-ils habités ? Justifier.

Pour ajouter du polymorphisme, on ajoute des variables de types $A, B, C, \ldots$:

$$T ::= A \mid \mathbb{N} \mid T \to T$$

On note $T[T'/A]$ le type obtenu en remplaçant dans T toutes les occurences de la variable A par le type T'.
Le sens qu'on donne aux variables se déduit de la règle suivante :

- Si T contient la variable A, un habitant du type T doit être un habitant du type $T[T'/A]$ pour tout type T'.

Question 4. Le type A est-il habité ? Justifier.

Question 5. Pour chaque type suivant, décider s'ils sont habités ou non (justifier formellement chaque résultat) :

1. $A \to A$
2. $A \to (A \to A)$
3. $(A \to A) \to A$
4. $A \to \mathbb{N}$
5. $\mathbb{N} \to A$
6. $A \to (B \to A)$

Question 6. Soit S, T deux types, montrer que si $S \to T$ est habité, et S est habité, alors T est habité.

Question 7. Soit S, T, deux types, montrer que si "(S est habité) $\Rightarrow$ (T est habité)", alors $S \to T$ est habité.

Question 8. En déduire un algorithme qui décide si un type T est habité ou non.
Etudier sa terminaison, sa correction et sa complexité.

Question 9. Exécuter l'algorithme sur $\mathbf{S} = (A \to (B \to C)) \to (A \to B) \to (A \to C)$

Question 10. Améliorer l'algorithme pour qu'il renvoie un habitant de T quand c'est possible.

Question 11. Exécuter le nouvel algorithme sur $\mathbf{S}$

Entrelacements Complets de Processus

Un *processus* P est une séquence finie d'*actions atomiques abstraites* distinctes $(a_i)_{i\in[\![0;n]\!]}$. On note $P[i]$ l'action d'indice i de P quand elle existe.
Un *système* S est un ensemble de processus $P_1, \ldots, P_k$ dont les actions sont toutes différentes.
Un *entrelacement* d'un système $S = \{P_1, \ldots, P_k\}$ est une séquence finie d'actions $(a_l)_{1\leq l\leq L}$ qui vérifie les conditions suivantes :

- les a_l sont différentes deux à deux,
- pour tout $1 \leq l \leq L$, il existe (i, j) tel que $a_l = P_i[j]$ (chaque action d'un entrelacement est une action d'un processus)
- si $a_{l_1} = P_i[j_1]$, $a_{l_2} = P_i[j_2]$, alors $l_1 \leq l_2 \Rightarrow j_1 \leq j_2$ (les actions d'un même processus apparaissent dans l'ordre dans un entrelacement)

La *taille* d'un processus $|P|$ est le nombre d'action qu'il contient. De manière similaire, la *taille* d'un entrelacement est le nombre d'action qu'il contient.
Un entrelacement est *complet* si toutes les actions de tous les processus du système apparaissent dans l'entrelacement.

Question 1. Soit $P_1 = (a_1, a_2, a_3)$, $P_2 = (b_1, b_2)$, $S = \{P_1, P_2\}$.

Donner deux entrelacements différents de taille 3 pour le système S.

Donner quatre entrelacements complets de S.

Question 2. Quelle est la taille d'un entrelacement complet de $S = \{P_i\}_{1\leq i\leq N}$.

Question 3. Si P_1 est de taille 1 et P_2 de taille n, décrire les entrelacements complets de $\{P_1, P_2\}$.

Question 4. Exprimer le nombre d'entrelacements complets d'un système $S = \{P_1, P_2\}$ en fonction de $|P_1|$ et $|P_2|$.

Question 5. Généraliser à $S = \{P_i\}_{1\leq i\leq N}$.

Question 6. Si le nombre de processus est fixé, et que le nombre total d'actions est fixé, quand le nombre d'entrelacements complets est-il maximal ?

Pour un système $S = \{P_i\}_{1\leq i\leq N}$, on dispose d'une fonction δ_S (on qu'on notera δ quand S est identifié clairement) qui donne le *coût* de passer d'une action atomique à une autre :
$\delta_S(a, b) \geq 0$ est le coût de passer de l'action atomique a à l'action atomique b.
Le coût d'un entrelacement $E = (a_l)_{1\leq l\leq L} = (P_{i_l}[j_l])_{1\leq l\leq L}$ est la somme des coûts de passage d'une action à l'action suivante, c'est-à-dire

$$\delta(E) = \sum_{1\leq l<L} \delta(P_{i_l}[j_l], P_{i_{l+1}}[j_{l+1}])$$

L'objectif est de trouver un entrelacement complet de coût minimal.

Question 7. Donner un algorithme **naïf** qui résout le problème dans le cas général. Etudier sa terminaison, sa correction et sa complexité.

Question 8. Si $S = \{(a_{l,i})_{1\leq i\leq K_l}\}_{1\leq l\leq N}$, on note

$$S - (p_1, \ldots, p_N) = \{(a_{l,i})_{p_i+1\leq i\leq K_l}\}_{1\leq l\leq N}$$

(le système obtenu en enlevant les p_i premières actions des processus P_i à S)

et $\mathcal{E}(s, p_1, \ldots, p_N)$ l'ensemble des entrelacements complets de $S - (p_1, \ldots, p_N)$ qui commencent par a_{p_s+1}

Donner une relation sur $min\{\delta(E) \mid E \in \mathcal{E}(s, p_1, \ldots, p_N)\}$ et en déduire un algorithme qui résout le problème dans le cas général. Etudier sa terminaison, sa correction et sa complexité.

On dispose d'un ensemble de *cellules* $V = x, y, z, \ldots$.
Une *mémoire* est une fonction partielle $\sigma : D \subset V \to \mathbb{N}$ où D est un ensemble fini de cellules.
Une action concrète sur une mémoire est une affectation $x \leftarrow E$ où E est une expression arithmétique simple (addition, multiplication, puissance) faisant éventuellement intervenir des cellules.
On appelle $val(E)_\sigma$ la valeur de E dans la mémoire σ (obtenue selon des règles standards) quand elle existe (si E fait référence à une cellule qui n'est pas dans le domaine de σ, elle n'a pas de valeur).
$\sigma[C]$ est la *mémoire après l'action* C définie par :

- $\sigma[x \leftarrow E](x) = val(E)_\sigma$ si elle existe.
- Pour tout y du domaine de σ, $\sigma[x \leftarrow E](y) = \sigma(x)$ si $y \neq x$

Si $C_1, C_2, \ldots, C_k$ est une séquence d'actions concrètes, $\sigma[C_1, C_2, \ldots, C_k] = \sigma[C_1][C_2]\ldots[C_k]$
On suppose maintenant que les actions abstraites des processus sont liées à des actions concrètes (deux actions abstraites peuvent être liées à la même action concrète), ce qui permet de donner un sens $\sigma[E]$ où E est un entrelacement.
On écrira, par abus de notations $P = \{C_i\}_i$ pour désigner un processus dont les actions abstraites sont liées, dans l'ordre aux actions concrètes C_i.

Question 9. On prend $P_1 = (x \leftarrow x + 1, x \leftarrow x + 1)$, $P_2 = (x \leftarrow 2 * x)$, $S = \{P_1, P_2\}$.

En partant d'une mémoire $\sigma_0 : \{x\} \to \mathbb{N}$ définie par $\sigma_0(x) = 0$, donner toutes les mémoires que l'ont peut obtenir après un entrelacement complet de S.

Question 10. Exprimer le nombre de mémoires différentes que l'on peut obtenir après un entrelacement complet d'un système quelconque.

Logique Temporelle Linéaire

On dispose d'un ensemble $\mathcal{A} = \{a, b, c, \dots\}$ de *propositions atomiques* décrivant un système.
Une *exécution* d'un système E est une fonction de $\mathbb{N}$ dans les parties finies de $\mathcal{A}$. Intuitivement, $a \in E(n)$ signifie "la proposition atomique a est vraie à l'instant n de l'exécution E".
On construit inductivement l'ensemble de formule suivant :

- $\mathtt{T}$ et $\bot$ sont des formules.
- $a \in \mathcal{A}$ est une formule.
- si φ est une formule, $\neg\varphi$, $\circ\varphi$, $\diamond\varphi$, $\Box\varphi$ sont des formules.
- si φ_1 et φ_2 sont des formules, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$ et $\varphi_1 \,\mathcal{U}\, \varphi_2$ sont des formules.

On donne une sémantique à ces formule sous la forme du jugement $n \vDash_E \varphi$ (intuitivement, l'exécution E vérifie φ à l'instant n) de manière inductive. Soit E une exécution (on donne entre parenthèse une intuition sur le sens de la formule) :

- pour tout $n \in \mathbb{N}$, $n \vDash_E \mathtt{T}$ (vrai)
- pour aucun $n \in \mathbb{N}$, $n \vDash_E \bot$ (faux)
- pour tout $n \in \mathbb{N}$, $n \vDash_E a$ si $a \in E(n)$ (proposition atomique)
- pour tout $n \in \mathbb{N}$, $n \vDash_E \neg\varphi$ si on n'a pas $n \vDash_E \varphi$ (négation)
- pour tout $n \in \mathbb{N}$, $n \vDash_E \circ\varphi$ si $(n+1) \vDash_E \varphi$ ("au suivant")
- pour tout $n \in \mathbb{N}$, $n \vDash_E \diamond\varphi$ s'il existe $k \geq n$, $k \vDash_E \varphi$ ("finalement")
- pour tout $n \in \mathbb{N}$, $n \vDash_E \Box\varphi$ si pour tout $k \geq n$, $k \vDash_E \varphi$ ("désormais")
- pour tout $n \in \mathbb{N}$, $n \vDash_E \varphi_1 \wedge \varphi_2$ si $n \vDash_E \varphi_1$ et $n \vDash_E \varphi_2$ (conjonction)
- pour tout $n \in \mathbb{N}$, $n \vDash_E \varphi_1 \vee \varphi_2$ si $n \vDash_E \varphi_1$ ou $n \vDash_E \varphi_2$ (disjonction)
- pour tout $n \in \mathbb{N}$, $n \vDash_E \varphi_1 \,\mathcal{U}\, \varphi_2$ s'il existe $k \geq n$, $k \vDash_E \varphi_2$ et $\forall n \leq k' < k, k' \vDash_E \varphi_1$ ("jusqu'à")

On s'autorise à écrire $n \vDash \varphi$ quand il est clair de quel E on parle.

Question 1. Soit une exécution E définie par : "pour tout $n \in \mathbb{N}$, $E(2.n+1) = \{b\}$ et $E(2.n) = \{a\}$"
Donner, à chaque fois, l'ensemble des entiers j tels que :

$$j \vDash_E a \qquad j \vDash_E \neg a \qquad j \vDash_E \diamond a \qquad j \vDash_E \Box a \qquad j \vDash_E a \vee b \qquad j \vDash_E a \,\mathcal{U}\, b$$

Question 2. Caractériser les exécutions E telles que $0 \vDash_E \Box(\neg a \,\mathcal{U}\, b)$ et $0 \vDash_E \Box\neg(a \wedge b)$.

Question 3. Montrer que pour toute exécution E, toute formule φ et tout entier i, $i \vDash_E \diamond\diamond\varphi$ est équivalent à $i \vDash_E \diamond\varphi$ et que $i \vDash_E \Box\Box\varphi$ est équivalent $i \vDash_E \Box\varphi$.

Question 4. Montrer que pour toute exécution E, toute formule φ et tout entier i, $i \vDash_E \diamond\Box\varphi$ implique $i \vDash_E \Box\diamond\varphi$.
Que dire de l'implication inverse ?

Question 5. Que dire de E_1 et E_2 quand, pour toute formule φ, $0 \vDash_{E_1} \varphi$ est équivalent à $0 \vDash_{E_2} \varphi$.

On définit l'opérateur $\mathcal{R}$ par $\varphi_1 \,\mathcal{R}\, \varphi_2 = \neg((\neg\varphi_1) \,\mathcal{U}\, (\neg\varphi_2))$.

Question 6. Expliciter le sens de $\varphi_1 \,\mathcal{R}\, \varphi_2$.
Comparer le sens des opérateurs $\mathcal{R}$ et $\mathcal{U}$.

Une formule φ est en *NNF* (forme normale de négation) si :

- si $\neg\varphi'$ apparaît dans φ, alors $\varphi' = a$ (les négations apparaissent uniquement devant les propositions atomiques).
- les seuls opérateurs apparaissant dans φ sont $\mathtt{T}, \bot, \wedge, \vee, \neg, \circ, \mathcal{R}, \mathcal{U}$.

Question 7. Montrer que tout formule φ est équivalente à une formule φ' en NNF.

Question 8. Discuter de la taille (en nombre de connecteurs) d'une telle formule.

Question 9. On peut considérer une exécution comme un *mot infini* sur un alphabet particulier.

Expliquer comment généraliser les notions connues sur les automates finis pour décrire un automate lisant des exécutions.

Question 10. Un automate A (lisant des exécutions) *vérifie la formule* φ si pour toute exécution E, $0 \vDash_E \varphi$ est équivalent à "E est acceptée par A".

Pour chacune des formules suivantes, donner un automate qui vérifie cette formule :

$$\neg a \qquad \Box a \qquad a \,\mathcal{U}\, (b \wedge c) \qquad \Box \diamond a$$

k-sélection

Pour $(m, n) \in \mathbb{N}^2$, on note $[\![m; n]\!] = \{i \in \mathbb{N} \mid m \leq i \leq n\}$.
On considère un ensemble $\mathcal{E}$ totalement ordonné avec $\leq$.
Soit $E = (e_i)_{1 \leq i \leq n} \in \mathcal{E}^n$ une séquence finie de n élements de $\mathcal{E}$.
Un *k-ième élément* de E est un élément $\mathbf{e}$ tel qu'il existe G, P deux ensembles d'entiers et $j \in \mathbb{N}$:

- $\mathbf{e} = e_j$ ($\mathbf{e}$ est dans E)
- $(G, P, \{j\})$ est une partition de $[\![1; n]\!]$
- $\forall p \in P$, $e_p \leq \mathbf{e}$ (chaque élément de E dont l'indice est dans P est plus petit que $\mathbf{e}$)
- $\forall g \in G$, $\mathbf{e} \leq e_g$ (chaque élément de E dont l'indice est dans G est plus grand que $\mathbf{e}$)
- $|P| = k - 1$ (le cardinal de P est $k - 1$)

Question 1. On considère, pour cette question, $\mathcal{E} = \mathbb{N}$ avec l'ordre standard.

Trouver, quand c'est possible, un k-ième élément des séquences suivantes pour chaque valeur de k dans $\mathbb{N}$:

1. $(5, 10, 2, 1)$
2. $(i)_{1 \leq i \leq n}$
3. $(0, \ldots, 0, 1, \ldots, 1)$ avec m fois l'élement 0 et $n - m$ fois l'élément 1

Question 2. Etudier l'existence et l'unicité d'un k-ième élément de E dans le cas général.

Question 3. Le problème *k-sélection* consiste à trouver un k-ième élément pour une séquence E de taille n.

Encadrer la complexité $\mathcal{C}(n)$ *en nombre de comparaisons* de ce problème, c'est-à-dire trouver deux fonctions f et g "suffisamment proches" telles que $\mathcal{C}(n) = O(g(n))$ et $f(n) = O(\mathcal{C}(n))$.

Question 4. Soit $E = (e_i)_{1 \leq n}$ et $(G, P, \{j\})$ une partition de $[\![1; n]\!]$ telle que :

- $\forall p \in P$, $e_p \leq e_j$
- $\forall g \in G$, $e_j \leq e_g$
- $|P| = m$

Que peut-on dire d'un k-ième élément de E ? Justifier formellement.

Question 5. En déduire un algorithme de *k-sélection*.

Etudier sa terminaison, sa correction, sa complexité en espace et sa complexité en temps (en nombre de comparaisons) dans le pire des cas (qu'on explicitera) et en moyenne, en supposant que les éléments de E sont énumérés dans un ordre uniformément aléatoire.

Une médiane de E est un $\lfloor \frac{n}{2} \rfloor$-ième élément de E.

Question 6. Comment calculer la médiane d'une séquence ayant moins de 5 éléments ?

Question 7. Supposons qu'on partitionne E en sous-séquences $\{S_l\}_{1 \leq l \leq \lceil \frac{n}{5} \rceil}$ de 5 éléments (éventuellement, la dernière sous-séquence a moins de 5 éléments) et que pour chacune de ces sous-séquences on récupère la médiane m_l.

On considère m la médiane de $(m_l)_{1 \leq l \leq \lceil \frac{n}{5} \rceil}$.

Combien d'éléments de E sont plus petits que m, combien sont plus grands que m ?

Question 8. En déduire un algorithme de k-sélection, linéaire dans le pire des cas.

Jeu de devinettes

Alice et Bob jouent au jeu suivant. Alice lance un dé à 6 faces en cachette et note la valeur obtenue : appelons-la x. Maintenant, Bob essaye de deviner x en posant des questions de la forme "est-ce que x vaut 4 ?" et Alice répond par oui ou non. Le but de Bob est de deviner x le plus rapidement possible en posant uniquement des questions de la forme "$\mathcal{Q}(n)$: est-ce que x vaut n ?" où $n \in \mathbb{N}$. Notons $G(x)$ le nombre de questions qu'il doit poser avant de trouver x. Plus précisément, si Bob pose les questions $\mathcal{Q}(n_1), \mathcal{Q}(n_2), \ldots$ alors $G(x) = \min\{i : n_i = x\}$. Alice et Bob répètent maintenant ce jeu un grand nombre de fois. On s'intéresse au nombre moyen de questions que Bob doit poser pour trouver la bonne valeur.

Question 1. *Donner une stratégie pour Bob lui permettant de poser, en moyenne,* $21/6$ *questions. On suppose que le dé est équilibré. Que se passe-t-il si le dé a* n *faces ?*

Bob souhaite améliorer ses chances. Avant le jeu, et sans qu'Alice ne le sache, il modifie le dé afin que les faces 2, 4 et 6 soient deux fois plus probables que les faces 1, 3 et 5.

Question 2. *Que devient la stratégie précédente ? Montrer que Bob dispose maintenant d'une stratégie plus efficace.*

On généralise la situation : Alice choisie une valeur $X \in \{1, \ldots, n\}$ selon une distribution de probabilité connue de Bob. On note p_i la probabilité que Alice choisisse la valeur i. On suppose pour simplifier les notations que $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n$. On note encore $G(X)$ le nombre de questions que Bob doit poser pour deviner x. Le nombre moyen de questions que Bob pose est donc l'espérance de $G(X)$: $E_X[G(X)]$.

Question 3. *Montrer que la stratégie optimale pour Bob vérifie* $E_X[G(X)] = \sum_{i=1}^{n} p_i \cdot i$. *On supposera que Bob a une stratégie déterministe, c'est à dire qu'il pose toujours les questions dans le même ordre.*

Question 4. *Proposer une modélisation lorsque Bob peut avoir une stratégie non-déterministe, et montrer que le résultat précédent est toujours vrai.*

Pour une variable aléatoire X, on note $C(X) := E_X[G(X)]$ le nombre moyen de questions pour une stratégie optimale G pour X, étudiée ci-dessus.

Question 5. *Donner une distribution* X *à valeur dans* $\{1, \ldots, n\}$ *qui maximise* $C(X)$.

On reprend l'exemple de Alice et Bob du départ mais cette fois-ci, Alice lance n pièces équilibrées et Bob doit deviner le nombre de pièces qui sont tombées sur le côté "pile".

Question 6. *Analyser cet exemple.*

On s'intéresse maintenant au cas où X est à valeurs dans un ensemble infini dénombrable, qu'on supposera être $\mathbb{N}$ sans perte de généralité.

Question 7. *Donner un exemple où* $C(X)$ *est fini et un autre où* $C(X)$ *est infini.*

Le problème des millionnaires socialistes

Alice dispose d'une valeur x et Bob d'une valeur y. Ils veulent savoir s'ils ont la même valeur (c'est à dire si $x = y$), sans révéler leur propre valeur dans le cas où elles seraient différentes. On peut imaginer, par exemple, qu'il s'agit de la réponse à un calcul difficile, de vérifier un mot de passe, le montant de leur fortune, ou encore une combinaison du tiercé. Le but de cet exercice est d'explorer des solutions possibles à ce problème. On suppose dans tout ce problème que $x, y \in E$, où E est un ensemble connu de taille n.

Question 1. *Donner une solution à ce problème en supposant l'existence d'une tierce partie neutre, Isaac. Quel est le défaut de cette solution ? Que peut-il se passer si Isaac n'est pas honnête ?*

Isaac leur propose une solution ne nécessitant pas son aide. Il suggère à Alice et Bob de se mettre d'accord sur une fonction de hachage $h : \mathbb{N} \to \{0, \ldots, 2^{64} - 1\}$. Intuitivement, cette fonction doit être très facile à calculer, mais étant donné t, il est très difficile de calculer z tel que $h(z) = t$. Autrement dit, h est très dur à inverser. Le protocole proposé par Issac est le suivant : Alice calcule $h(x)$ et Bob calcule $h(y)$. On compare ensuite $h(x)$ avec $h(y)$ et on déduit si $x = y$ ou non. On supposera que h est injective sur les n valeurs possibles qui intéressent Alice et Bob.

Question 2. *Expliquer pourquoi cette solution ne marche pas du tout lorsque n est petit, et révèle quand même beaucoup d'information sur x et y même lorsque n est grand.*

On propose maintenant une solution sans tierce partie. Alice et Bob achètent n urnes de vote. Chaque urne a une unique clef permettant de la fermer, et un trou permettant d'insérer un bout de papier à l'intérieur. Le protocole est le suivant :

1. Alice marque de façon clairement visible les n valeurs sur les urnes, une par urne.
2. Alice garde dans sa poche la clé de l'urne marquée par x.
3. Alice détruit les $n - 1$ autres clés devant Bob.
4. Bob insère dans l'urne y un papier avec "oui", et dans toutes les autres urnes un papier avec "non".
5. Sans la présence de Bob, Alice ouvre l'urne x avec sa clé et récupère le bout de papier.
6. Alice donne ensuite le bout de papier à Bob.

Question 3. *Expliquer en quoi le protocole ci-dessus résout le problème si on suppose que Alice et Bob sont honnêtes. Comment Alice ou Bob pourraient essayer de tricher ? Discuter les solutions possibles. En supposant les participants honnêtes, quel est selon vous la principale limitation de ce protocole ?*

On cherche maintenant une solution informatique à ce problème. Pour ce faire, on s'intéresse à une primitive appelée le transfert inconscient. On suppose que Bob dispose de n messages $m_1, \ldots, m_n$, encodés par des entiers. De son côté, Alice dispose d'un indice $i \in \{1, \ldots, n\}$. Le but du protocole est de faire en sorte que Alice apprenne m_i, en gardant m_j secret pour tous les $j \neq i$ et que Bob n'apprenne pas la valeur de i.

Question 4. *Montrer que si on sait réaliser un tel transfert inconscient, alors on peut résoudre le problème de départ. Quel est le lien entre cette solution et celle avec les urnes ?*

Question 5. *Sans rentrer dans les détails, les implémentations d'un tel transfert inconscient nécessitent que Alice et Bob s'envoient $O(n)$ messages. Quels sont les avantages de ce protocole par rapport à celui des urnes ? Quelles limitations existent encore ?*

Question 6. *Expliquer comment modifier le protocole pour que Alice et Bob puissent savoir si $x \geqslant y$ au lieu de $x = y$. Justifier pourquoi il ne révèle pas d'information sur la valeur de x et y, autre que le fait que $x \geqslant y$ ou $x < y$.*

Equations de mots

Soit Σ un alphabet fini et x une variable. On note ε le mot vide. Pour tout mot $w \in \Sigma^*$, on note $|w|$ sa longueur. Etant donné $w \in \Sigma^*$ et $1 \leqslant i \leqslant j \leqslant |w|$, on pose $w[i:j] = w_i w_{i+1} \cdots w_j$ le facteur de w entre les indices i et j (inclus). On pourra aussi utiliser la notation $w[i:]$ pour signifier $w[i:|w|]$ et $w[:j]$ pour $w[1:j]$.
Une équation de mots (à une variable) est une équation de la forme

$$A_0 x A_1 x \cdots x A_r = B_0 x B_1 x \cdots x B_s \tag{1}$$

où $A_0, \ldots, A_r, B_0, \ldots, B_s \in \Sigma^*$ sont des mots. Une solution est un mot $x \in \Sigma^*$ qui satisfait (1). Par exemple, sur l'alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, l'équation

$$xx\mathtt{baababa} = \mathtt{ababa}x\mathtt{ab}x.$$

admet $x = \mathtt{ababaababa}$ comme solution mais pas $x = \mathtt{a}$.

Question 1. *Donner toutes les solutions de* $\mathtt{ab}x = x\mathtt{ba}$ *sur l'alphabet* $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

Question 2. *Soit* $U, V \in \Sigma^*$ *non vides. Montrer que l'équation* $xx = UxV$ *admet au plus une solution que l'on identifiera, ainsi qu'une condition nécessaire et suffisante d'existence.*

Question 3. *Montrer que le cas général* (1) *peut toujours se ramener au cas où* $A_0 \neq \varepsilon$ *et* $B_0 = \varepsilon$.

On suppose maintenant que l'équation est de la forme obtenue ci-dessus, c'est à dire $A_0 \neq \varepsilon$ et

$$A_0 x A_1 x \cdots x A_r = x B_1 x \cdots x B_s. \tag{2}$$

Question 4. *Montrer que si* x *est solution de* (2) *alors il existe* $k \in \mathbb{N}$ *et* $0 \leqslant \ell < |A_0|$ *tel que* $x = A_0^k A_0[1:\ell]$.

On dit que $A, B \in \Sigma^*$ sont conjugués s'il existe $u, v \in \Sigma^*$ tels que $A = uv$ et $B = vu$. On dit que $P \in \Sigma^*$ est primitif si $P \neq \varepsilon$ et P n'apparaît que deux fois comme facteur de PP. Formellement, P est primitif si et seulement si pour tout mots $u, v \in \Sigma^*$, $P^2 = uPv$ implique que $u = \varepsilon$ ou $v = \varepsilon$. Par exemple, $\mathtt{ab}$ est primitif mais $P = \mathtt{abab} = (\mathtt{ab})^2$ ne l'est pas car il apparaît 3 fois dans $PP = (\mathtt{ab})^4$.

Question 5. *Soit* $R \in \Sigma^*$ *primitif et* $k \geqslant 1$. *Montrer que l'ensemble des solutions de* $R^k x = xR^k$ *est* R^*.

Question 6. *En déduire que pour tout* $P \in \Sigma^*$, *il existe* $R \in \Sigma^*$ *primitif et* $k \in \mathbb{N}$ *tel que* $P = R^k$.

Question 7. *Soit* P *primitif et* $k \geqslant 1$. *Donner l'ensemble des* $x, y \in \Sigma^*$ *tels que* $xPy = P^k$.

Question 8. *Que peut-on dire de l'ensemble des solutions de* (2) *dans le cas où* $r \neq s$ *?*

Question 9. *Résoudre* (2) *dans le cas où* $r = s = 1$ *et* $A_1 = \varepsilon$. *On identifiera une condition nécessaire et suffisante d'existence.*

Question 10. *Résoudre* (2) *dans le cas où* $r = s = 2$ *et* $A_2 = \varepsilon$.

Racines de langages réguliers

Soit Σ un alphabet fini et $L \subseteq \Sigma^*$ un langage. On définit la racine n^{ieme} de L par

$$\sqrt[n]{L} = \{w \in \Sigma^* : w^n \in L\}$$

et on posera $\sqrt{L} = \sqrt[2]{L}$. On définit par ailleurs le radical de L par

$$\sqrt[\infty]{L} = \{w \in \Sigma^* : \exists n \geqslant 1\ w^n \in L\}.$$

Le but de ce problème est d'étudier le radical de L. On notera ε le mot vide.

Question 1. *Soit $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ et $m \geqslant 1$ un entier quelconque. Donner $\sqrt{L_1}$, $\sqrt[3]{L_1}$ et $\sqrt[\infty]{L_1}$ où $L_1 = \{\mathtt{a}^m\}$, et $\sqrt{L_2}$ où $L_2(m) = (\mathtt{a}\Sigma^{2m})^*$.*

Dans la suite de ce problème, on suppose que L est régulier et reconnu par un automate fini déterministe (AFD) complet $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ où Q est l'ensemble des états, $q_0 \in Q$ est l'état initial, $\delta : Q \times \Sigma \to Q$ la fonction de transition et $F \subseteq Q$ l'ensemble des états finaux. On étendra δ aux mots par induction : $\delta(q, \varepsilon) = q$ et $\delta(q, wa) = \delta(\delta(q, w), a)$ pour tous $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.

Question 2. *Montrer qu'il existe une constante n_0 (qui dépend de $\mathcal{A}$) telle que pour tout $u \in \Sigma^*$ et $n \geqslant n_0$, si $u^n \in L$ alors il existe $n' < n_0$ tel que $u^{n'} \in L$.*

Question 3. *Que peut-on en déduire sur la relation entre $\sqrt[\infty]{L}$ et les $\sqrt[n]{L}$?*

On considère l'automate $\mathcal{B} = (\Sigma, Q^Q, \delta', q'_0, F')$ où Q^Q est l'ensemble des fonctions $f : Q \to Q$, $q'_0 = (q \mapsto q)$ et pour tout $f \in Q^Q$ et $\alpha \in \Sigma$,

$$\delta'(f, \alpha) = (q \mapsto \delta(f(q), \alpha)).$$

On ne précise pas F' pour l'instant.

Question 4. *Montrer que pour tout $w \in \Sigma^*$, $\delta'(q'_0, w) = (q \mapsto \delta(q, w))$ en étendant δ' aux mots comme pour δ.*

Question 5. *Soit $n \geqslant 1$. Expliquer comment choisir F' pour que $\mathcal{B}$ reconnaisse $\sqrt[n]{L}$.*

Question 6. *Montrer qu'il existe un choix de F' tel que $\mathcal{B}$ reconnaisse $\sqrt[\infty]{L}$.*

Question 7. *Proposer un choix de F' tel que $\mathcal{B}$ reconnaisse $\sqrt[\infty]{L}$, et qui n'utilise pas le résultat de la question 3. Donner un algorithme qui calcule ce F'.*

Soit $k \in \mathbb{N}$ et $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. On pose

$$L(k) = \{w \in \Sigma^* : \forall 1 \leqslant i \leqslant |w|\ i \equiv 0 \pmod{k} \Rightarrow w_i = \mathtt{a}\}.$$

Autrement dit, $L(k)$ contient les mots dont toutes les lettres aux positions qui sont des multiples de k sont $\mathtt{a}$.

Question 8. *Calculer $\sqrt{L(k)}$. Montrer que $L(k)$ est reconnu par un AFD à k états. Argumenter que tout AFD qui reconnaît $\sqrt{L(k)}$ a au moins $\Omega(2^k)$ états. Comparer à la construction de la question 5.*

Nombre de mots dans un langage régulier

Soit Σ un alphabet fini et $L \subseteq \Sigma^*$ un langage régulier. On notera ε le mot vide. On suppose que L est régulier et reconnu par un automate fini déterministe (AFD) complet $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ où Q est l'ensemble des états, $q_0 \in Q$ est l'état initial, $\delta : Q \times \Sigma \to Q \cup \{\perp\}$ la fonction de transition et $F \subseteq Q$ l'ensemble des états finaux. On autorisera l'automate à ne pas être complet, auquel cas δ renvoie $\perp$ si une transition n'existe pas. On étendra δ aux mots par induction : $\delta(q, \varepsilon) = q$, $\delta(q, wa) = \delta(\delta(q, w), a)$ et $\delta(\perp, w) = \perp$ pour tous $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.
Pour tout $n \in \mathbb{N}$, on note $L(n) = |L \cap \Sigma^n|$ le nombre de mots de longueur n dans L. Le but de ce problème est d'étudier la suite $(L(n))_n$.

Question 1. *Donner un algorithme qui calcule $L(n)$ pour un entier n donné. Quelle est sa complexité ?*

Question 2. *Donner un automate $\mathcal{A}_1$ à deux états pour $L_1 = \Sigma^* \setminus (\Sigma^* \mathtt{a}\mathtt{a} \Sigma^*)$ où $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.*

Question 3. *Montrer que $(L_1(n))_n$ satisfait une récurrence linéaire d'ordre 2 que l'on explicitera. Reconnaissez-vous cette suite ?*

Sans perte de généralité, on suppose maintenant que $Q = \{1, \dots, d\}$. On considère la matrice $A \in \mathbb{N}^{d \times d}$ définie pour tout $i, j \in Q$ par

$$A_{i,j} = |\{\alpha \in \Sigma : \delta(j, \alpha) = i\}|.$$

On pose $x \in \mathbb{N}^d$ le vecteur colonne défini par $x_{q_0} = 1$ et $x_j = 0$ pour tout $j \neq q_0$.

Question 4. *Montrer que pour tout $n \in \mathbb{N}$ et $i \in Q$, $(A^n x)_i = |\{w \in \Sigma^n : \delta(q_0, w) = i\}|$.*

Question 5. *En déduire un vecteur $y \in \mathbb{N}^n$ tel que $L(n) = y^T A^n x$.*

Question 6. *Donner un algorithme beaucoup plus efficace qui calcule $L(n)$ pour un entier n donné. On pourra exprimer sa complexité en comptant le nombre d'additions et multiplications d'entiers qu'il effectue.*

Soit p_A le polynôme caractéristique de A. On rappelle que $p_A(A) = 0$ d'après le théorème de Cayley-Hamilton. On dit que qu'une suite $(u_n)_n$ satisfait une suite récurrente linéaire d'ordre p s'il existe $a_0, \dots, a_{p-1} \in \mathbb{R}$ tels que $u_{n+p} = \sum_{i=0}^{p-1} a_i u_{n+i}$ pour tout $n \in \mathbb{N}$. Par exemple, la suite de Fibonacci satisfait la récurrence linéaire $F_{n+2} = F_{n+1} + F_n$ d'ordre 2.

Question 7. *Montrer que $(L(n))_n$ est une suite récurrente linéaire d'ordre d.*

Question 8. *Montrer que pour tout suite récurrente linéaire $(u_n)_{n \in \mathbb{N}}$ d'ordre f, il existe des vecteurs $X, Y \in \mathbb{R}^f$ une et matrice $A \in \mathbb{R}^{f \times f}$ tels que $u_n = Y^T A^n X$.*

Question 9. *Soit L' un langage régulier. Montrer que $(L(n) - L'(n))_n$ est une suite récurrente linéaire. Quel est son ordre ?*

Question 10. *Donner un algorithme qui, étant donné deux langages réguliers L et L', décide si $L(n) = L'(n)$ pour tout $n \in \mathbb{N}$. Quelle est sa complexité ?*

Division de polynômes

Dans ce problème, lorsque l'on demande une complexité, on comptera le nombre d'opérations arithmétiques (additions, multiplications, divisions) dans $\mathbb{Q}$. On se contentera de donner les complexités asymptotiques avec la notation grand O. On rappelle qu'un polynôme non-nul $p \in Q[X]$ est unitaire si son coefficient dominant est 1. On notera $\deg(P)$ le degré du polynôme P et $\mathrm{dom}(P)$ son coefficient dominant. Dans ce problème, on représentera un polynôme $P = \sum_{i=0}^{d} a_i X^i$ de degré d par le tableau a de ses coefficients (qui est donc de taille $d+1$).

Question 1. *Rappeler brièvement l'algorithme naïf pour multiplier deux polynômes de degrés d et d' respectivement. Donner sa complexité.*

On pose dans la suite $M(d)$ la complexité de multiplier deux polynômes de degré d. On suppose que cette fonction est croissante et satisfait $M(d) \geqslant d$ et $M(2d) \geqslant 2M(d)$ pour tout d.

Question 2. *Rappeler l'algorithme naïf pour faire la division Euclidienne d'un polynôme A de degré d_A par un polynôme de B de degré d_B. Donner sa complexité.*

Question 3. *Justifier qu'on peut toujours se ramener, dans le cas de la division Euclidienne, au cas où B est unitaire. Quelle est la complexité de cette opération ?*

Soit $P = \sum_{i=0}^{d} p_i X^i$ un polynôme tel que $p_0 \neq 0$. On considère la suite de polynômes définie par

$$f_0 = p_0^{-1}, \qquad f_{i+1} = 2f_i - Pf_i^2.$$

Question 4. *Montrer que $Pf_i \equiv 1 \bmod X^{2^i}$ pour tout $i \in \mathbb{N}$.*

Question 5. *Étant donné un entier k, donner un algorithme pour calculer un polynôme F tel que $BF \equiv 1 \bmod X^k$. Analyser sa complexité en fonction de k, d et la fonction $M(\cdot)$.*

Question 6. *Améliorer votre algorithme pour obtenir une complexité en $O(M(2k))$.*

Soit $P \in \mathbb{Q}[X]$ de degré d et $k \geqslant \deg(P)$. On pose $\mathrm{rev}_k(P) = X^k P(1/X)$ le polynôme renversé de P de degré k.

Question 7. *A quoi correspond cette opération au niveau des coefficients de P ?*

Question 8. *En utilisant l'algorithme de la question 6, donner un algorithme pour calculer la division euclidienne de deux polynômes de degrés d en $O(M(2d))$.*

Pour la culture : il est possible de multiplier deux polynômes en $M(d) = O(d \log d)$ grâce à la transformation de Fourier discrète.

Pré-traitement de tableaux

Un tableau `t` de taille n est une structure de données permettant de stocker n entiers. Étant donné un index $i \in \mathbb{N}$, $1 \leq 0 \leq n-1$, la case numéro i du tableau `t`, `t[i]`, peut être lue ou modifiée en temps constant $O(1)$.
On considère un processus en trois étapes sur un tableau :

1. l'initialisation du tableau à la fin de laquelle chaque case contient un élément donné.
2. une phase de pré-traitement, où l'on dispose d'un temps et d'un espace supplémentaire donné pour stocker des informations.
3. une phase de requêtes, où l'on reçoit successivement une suite de requêtes à effectuer. Seule la prochaine requête à satisfaire est connue, et elle doit être effectuée en un temps donné.

On considère la phase d'initialisation terminée sur un tableau `t` de taille n, et que les requêtes sont toutes du type `MIN(i,j)` avec $0 \leq i \leq j \leq n-1$, et où la réponse à renvoyer est $\min_{i \leq k \leq j} t[k]$.

Question 1. Donner un processus qui répond aux requêtes en temps constant et utilise un pré-traitement en temps et en espace $O(n^3)$.

Question 2. Donner un processus qui répond aux requêtes en temps constant et utilise un pré-traitement en temps et en espace $O(n^2)$.

Question 3. Donner un processus qui répond aux requêtes en temps $O(\log n)$ et utilise un pré-traitement en temps et en espace $O(n)$.

Question 4. Donner un processus qui répond aux requêtes en temps constant et utilise un pré-traitement en temps et en espace $O(n \log n)$.

On considère toujours la phase d'initialisation terminée sur un tableau `t` de taille n, mais les requêtes sont désormais toutes du type `SUM(i,j)` avec $0 \leq i \leq j \leq n-1$, et où la réponse à renvoyer est $\sum_{i \leq k \leq j} t[k]$.

Question 5. Donner un processus qui répond aux requêtes en temps constant et utilise un pré-traitement en temps et en espace $O(n)$.

On considère maintenant un processus sur un très grand tableau `t` de taille n qu'il est trop coûteux d'initialiser. On suppose qu'il est possible de définir un tel tableau en temps constant, mais sans pouvoir modifier les données qu'il contient. Aucune hypothèse ne peut donc être faite sur les valeurs initiales des cases `t[i]`, $0 \leq i \leq n-1$. Les requêtes sont soit du type `ECRIRE(i,x)`, soit du type `LIRE(i)`, avec $0 \leq i \leq n-1$ et $x \in \mathbb{N}$. La requête `LIRE(i)` doit renvoyer le dernier `x` pour lequel une requête `ECRIRE(i,x)` a été soumise, ou `null` si aucune requête `ECRIRE(i,x)` n'a été soumise. Il y aura m requêtes `ECRIRE` au total. Il est autorisé d'utiliser un espace supplémentaire de taille $O(\min(m,n))$, par exemple pour créer un tableau non initialisé.

Question 6. Donner un processus qui répond aux requêtes en temps constant et utilise un pré-traitement en temps constant.

Ordonnancement non-clairvoyant

On considère n tâches à exécuter sur une unique machine, une tâche à la fois. La tâche numéro i, pour $1 \leq i \leq n$, doit être exécutée pendant une durée totale égale à $x_i \in \mathbb{Q} \cap [1; +\infty[$ pour être terminée.
On autorise la possibilité d'interrompre l'exécution d'une tâche pour pouvoir avancer sur l'exécution d'une autre tâche. Un *ordonnancement* $\mathcal{S}$ définit quelle tâche est exécutée à quel moment. On peut le noter comme une suite de paires $\{i_j, d_j\}_{j \in \mathbb{N}^*}$, où la tâche i_1 est exécutée pendant une durée d_1, puis la tâche i_2 pendant une durée d_2, ... Un ordonnancement est valide si la somme des durées d'exécution de chaque tâche i est au moins égale à x_i. La tâche i est déclarée terminée à l'instant $T_i^{\mathcal{S}}$ à partir duquel elle a été exécutée pendant une durée x_i au total.
L'objectif de ce sujet est d'étudier le problème qui consiste à calculer un ordonnancement $\mathcal{S}$ minimisant $\sum_{i=1}^{n} T_i^{\mathcal{S}}$.
La particularité est que l'algorithme qui prend les décisions d'ordonnancement ne connaît pas les valeurs x_i à l'avance, toutes les tâches sont donc équivalentes au départ. Seule la terminaison d'une tâche est connue de l'algorithme, qui peut donc utiliser cette information pour prendre les décisions futures.

Question 1. Proposer un algorithme qui aboutit toujours à un ordonnancement $\mathcal{S}$ minimisant $\max_{1 \leq i \leq n} T_i^{\mathcal{S}}$ au lieu de minimiser $\sum_{i=1}^{n} T_i^{\mathcal{S}}$.

Question 2. En supposant que les valeurs x_i sont connues de l'algorithme, proposer un algorithme qui aboutit toujours à un ordonnancement $\mathcal{S}$ minimisant $\sum_{i=1}^{n} T_i^{\mathcal{S}}$.

Afin d'évaluer la performance d'un algorithme A qui aboutit à un ordonnancement $\mathcal{S}_{\{x_i\}}$ dépendant de la valeur, initialement inconnue, des x_i, on utilise la quantité suivante :

$$c(A) = \sup_{(x_1, x_2 \ldots x_n) \in (\mathbb{Q} \cap [1; +\infty[)^n} \frac{\sum_{j=1}^{n} T_j^{\mathcal{S}_{\{x_i\}}}}{OPT_{\{x_i\}}}$$

où $OPT_{\{x_i\}}$ représente le coût optimal d'un ordonnancement connaissant les valeurs des x_i à l'avance (i.e., $OPT_{\{x_i\}} = \min_{\mathcal{S}} \sum_{i=1}^{n} T_i^{\mathcal{S}}$).
L'objectif est de trouver un algorithme A menant à une valeur $c(A)$ aussi faible que possible, en particulier qui est bornée par une constante ne dépendant pas de n.

On s'intéresse à l'algorithme Round-Robin, qui exécute chaque tâche non terminée à tour de rôle, pendant une durée ε négligeable.
Soit $d^{\mathcal{S}}(i, j)$ qui représente la durée pendant laquelle la tâche i a été exécutée avant $T_j^{\mathcal{S}}$.
On suppose pour simplifier les notations que $x_1 \leq x_2 \leq \cdots \leq x_n$, cet ordre étant évidemment inconnu de l'algorithme.

Question 3. Exprimer le coût $OPT_{\{x_i\}}$ et le coût de l'algorithme Round Robin $RR_{\{x_i\}}$. La quantité $d^{\mathcal{S}}(i, j)$ peut être utile comme étape intermédiaire.

Question 4. Calculer $c(RR)$.

Question 5. On suppose pour cette question uniquement que les x_i peuvent avoir comme valeur tout nombre rationnel strictement positif. Justifier que l'algorithme Round-Robin n'est pas bien défini.

Tourner la page SVP

On souhaite maintenant prouver que pour tout algorithme A, $c(A) \geq 2-2/(n+1)$. Soit A un algorithme résolvant ce problème, et X un entier arbirairement grand. On souhaite donc construire une instance pour laquelle l'ordonnancement renvoyé par A a un coût éloigné de $OPT_{\{x_i\}}$, c'est-à-dire que l'on laisse tourner l'algorithme A et l'on peut choisir les valeurs x_i en fonction des décisions prises par A. Tout d'abord, on exécute A pendant une durée X sans terminer une seule tâche.

Question 6. Montrer le résultat souhaité.

Recherche linéaire

On considère une clôture rectiligne infinie, trouée à un unique endroit, ainsi qu'une vache positionnée d'un côté de la clôture. L'objectif de la vache est de se déplacer le long de la clôture jusqu'à se trouver en face du trou. La position du trou reste totalement inconnue jusqu'à ce que la vache soit positionnée en face.
On identifie la clôture avec la ligne des réels et on suppose que la vache est située à l'origine, donc à la position 0. Soit $X \in \mathbb{R}^+ \cap [1; +\infty[$ la distance entre la position initiale de la vache et le trou dans la clôture et $\gamma \in \{-1, 1\}$ la direction du trou, qui est donc à la position γX.
On dit qu'une vache suit une *stratégie* A définie par une suite de valeurs $\{a_i\}_{i\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$ si la vache se déplace successivement aux positions a_i, pour $i \in \mathbb{N}$, en suivant le plus court chemin entre a_i et a_{i+1}.
Le coût $C_A^{\gamma,X}$ d'une stratégie A sur une instance donnée équivaut à la distance parcourue par la vache avant de trouver le trou, exprimée en fonction de X. La solution optimale, connaissant la position exacte, a donc un coût de X correspondant à la distance à parcourir pour la vache.
On définit la qualité d'une stratégie A comme le ratio maximum entre le coût de la solution trouvée par A et la solution optimale :

$$c(A) = \max_{\gamma\in\{-1,1\}} \sup_{X\in\mathbb{R}^+\cap[1;+\infty[} \frac{C_A^{\gamma,X}}{X}.$$

Si la direction du trou est connue, il suffit à la vache de se déplacer dans cette direction jusqu'à se retrouver en face du trou. Cette solution a donc toujours un coût égal à X et une qualité optimale égale à 1.

Question 1. Proposer un algorithme minimisant $c(A)$ en supposant la valeur X connue mais pas γ.

Question 2. Quelle serait une conséquence importante de l'abandon d'une borne inférieure sur la valeur de X, ce qui reviendrait donc à supposer seulement $X \in \mathbb{R}^+$?

Question 3. On considère l'algorithme D^+ allant successivement aux positions $x_i = (-2)^i$ pour $i = 0, 1, 2, \ldots$. Déterminer la valeur de $c(D^+)$.

Question 4. On considère l'algorithme D^- allant successivement aux positions $y_i = -(-2)^i$ pour $i = 0, 1, 2, \ldots$. Déterminer la valeur maximale du ratio $\min\left(C_{D^+}^{\gamma,X},\ C_{D^-}^{\gamma,X}\right)/X$.

Question 5. Déduire de la question précédente un algorithme randomisé (utilisant de l'aléatoire) dont le ratio maximal de l'espérance du coût de la solution divisé par X est plus petit que $c(D^+)$.

Question 6. Soit une stratégie S allant successivement aux positions $z_0, z_1, z_2 \ldots$ tels que :

- $|z_i| \leq |z_{i+2}|$ pour tout i
- le signe des z_i est alterné
- $1 \leq |z_0| \leq 4$
- $|z_n| \leq 3|z_{n-1}| - \sum_{i=0}^{n-2} |z_i|$

Montrer que $c(S) = c(D^+)$.

Question 7. On considère maintenant le problème à deux dimensions : la vache se déplace dans un plan infini et doit trouver n'importe quel point d'une droite inconnue. La droite est trouvée seulement lorsque la vache se trouve sur un point de la droite.
On suppose que le point le plus proche de la droite est à distance 1 de la position initiale de la vache. Proposer une stratégie de recherche en essayant de parcourir le moins de distance possible dans le pire cas.

Question 8. Sans faire d'analyse, proposer une stratégie de recherche pour le même problème à deux dimensions lorsque la distance entre la droite et la position initiale est inconnue.

Problèmes de cache

Soit $P = \{p_1, p_2, \ldots, p_n\}$ un ensemble d'éléments appelés *pages*. On considère un cache de capacité k initialement vide, avec $k < n$ et une séquence de requêtes $\sigma = (r_1, r_2, \ldots, r_{|\sigma|})$ où chaque $r_i \in P$ arrive successivement dans cet ordre.
Pour satisfaire une requête, la page associée doit être présente dans le cache. Si elle n'est pas présente, il faut l'ajouter au cache, ce qui a un coût de 1. Si le cache contient déjà k pages différentes, il faut d'abord enlever une page du cache avant d'en rajouter une. Enlever une page du cache n'a pas de coût supplémentaire.
L'objectif est de minimiser le coût total pour satisfaire les requêtes σ. Noter que les seules décisions importantes consistent à choisir quelle page enlever du cache lorsque l'on doit rajouter une nouvelle page à un cache plein.

Question 1. Montrer que le coût minimum pour $k = 4$, $P = \{a, b, c, d, e, f\}$, et $\sigma = (a, b, c, d, e, f, a, b, e, d)$ est égal à 7.

Question 2. Si l'ensemble des requêtes σ est connu, donnez un algorithme simple aboutissant à un coût optimal. La preuve d'optimalité est demandée.

Indication : pour prouver l'optimalité de la solution, on pourra considérer une solution optimale qui coincide avec la solution proposée pendant un temps maximal.

On suppose maintenant que l'on ne connaît pas l'ensemble des requêtes σ dès le départ, mais seulement celle que l'on doit satisfaire actuellement. La prochaine requête est révélée une fois la requête actuelle satisfaite.
Afin d'évaluer la performance d'un algorithme A qui aboutit à un coût C_σ^A dépendant de σ, on utilise la quantité suivante :

$$c(A) = \sup_{n \in \mathbb{N},\ \sigma \in \mathbb{P}^n} \frac{C_\sigma^A}{OPT_\sigma}$$

où OPT_σ représente le coût optimal d'un algorithme connaissant σ à l'avance.

Question 3. Soit l'algorithme LFU (Less Frequently Used) qui élimine toujours la page présente dans le cache qui a été la moins demandée jusqu'à présent. Montrer que $c(LFU)$ est infini.

Question 4. Soit A' un algorithme qui a connaissance de la requête à satisfaire, ainsi que de la suivante. Montrer qu'il existe un algorithme A ayant seulement connaissance de la requête à satisfaire tel que $c(A) \leq c(A')$.

Question 5. Montrer que, pour une valeur de n bien choisie, pout toute séquence σ, il existe une solution de coût au plus $\frac{|\sigma|}{k} + k$.

En déduire que tout algorithme déterministe A (i.e., n'ayant pas recours à l'aléatoire) respecte $c(A) \geq k$.

Question 6. Soit l'algorithme LRU (Least Recently Used) qui élimine toujours la page présente dans le cache qui a été demandée le moins récemment. Montrer que $c(LRU) = k$.

Question 7. Montrer que $\sup_{n \in \mathbb{N},\ \sigma \in \mathbb{P}^n} \frac{LRU_\sigma^{2k}}{OPT_\sigma^k} = 2$, où LRU_σ^{2k} est le coût de l'algorithme LRU sur l'instance σ avec un cache de taille $2k$ et OPT_σ^k est le coût d'une solution optimale sur l'instance σ avec un cache de taille k.

Couvertures par sommets

Un graphe non-orienté G est défini par un ensemble fini de sommets V et d'arêtes E qui est un ensemble de paires (deux éléments non-ordonnés) de V. On suppose que $\forall v \in V, \{v, v\} \notin E$. Un ensemble $S \subseteq V$ est une couverture de G si $\forall \{u, v\} \in E,\ (u \in S \ \vee \ v \in S)$. On considère une fonction de poids $w : V \to \mathbb{Q}^+$. Par extension, on note $w(S) = \sum_{v \in S} w(v)$ pour tout ensemble $S \subseteq V$.

L'objectif est de trouver une couverture S de G de poids $w(S)$ faible. Soit C^* une couverture de G de poids minimal et $OPT = w(C^*)$.

On dit que w est une fonction par degrés s'il existe $c \in \mathbb{Q}^+$ tel que $\forall v \in V, w(v) = c \cdot deg(v)$, où le degré $deg(v)$ est le nombre d'arêtes incidentes à v dans G, i.e., $deg(v) = cardinal(\{u \in V \mid \{u, v\} \in E\})$.

On s'intéresse tout d'abord à trouver une couverture de faible poids pour des cas particuliers de fonction w.

Question 1. Si w est une fonction par degrés, montrer que $w(V) \leq 2 \cdot OPT$.

Question 2. Comment calculer efficacement la plus grande fonction par degrés p inférieure à une fonction w donnée ?

Le terme $G \setminus S$ pour $S \subseteq V$ est défini comme le graphe G auquel les sommets de S et les arêtes qui ont au moins une extrémité dans S sont supprimés.

Question 3. Si w est une fonction constante, montrer que l'algorithme CONSTANT ci-dessous renvoie une couverture de poids au plus $2 \cdot OPT$.

Algorithme 1 CONSTANT$(G = (V, E), w)$

$S \leftarrow \emptyset$
tant que $E \neq \emptyset$ **faire**
  Choisir une arête $\{u, v\} \in E$ arbitrairement
  $S \leftarrow S \cup \{u, v\}$
  $G \leftarrow G \setminus \{u, v\}$ ▷ cf encadré
fin tant que
retourner S

On s'intéresse maintenant à une fonction $w : V \to \mathbb{Q}^+$ quelconque.

Algorithme 2 QUELCONQUE$(G = (V, E), w)$

$G_0 \leftarrow G$
$t \leftarrow 0$
$w_0 \leftarrow w$
répéter
  $D_t \leftarrow \{u \in V_t : deg_t(u) = 0\}$ ▷ deg_t est le degré dans G_t
  $p_t \leftarrow$ plus grande fonction par degrés p inférieure à w_t dans G_t
  $S_t \leftarrow \{u \in V_t : p_t(u) = w_t(u)\}$
  $G_{t+1} \leftarrow G_t \setminus (D_t \cup S_t)$ ▷ cf encadré
  $w_{t+1} \leftarrow w_t - p_t$
  $t \leftarrow t + 1$
jusqu'à ce que G_t n'a pas d'arête
retourner $C = \bigcup_{k=0}^{t-1} S_k$

Question 4. Montrer que le résultat renvoyé par QUELCONQUE est une couverture, et que l'algorithme termine en temps polynomial en $|V|$ et $|E|$.

Question 5. Montrer que le poids de la couverture renvoyée par QUELCONQUE est au plus $2 \cdot OPT$.

Question 6. Montrer qu'il existe une instance sur laquelle le poids de la couverture renvoyée par QUELCONQUE est égal à $2 \cdot OPT$

Dimension des ordres partiels finis

Rappels : un ordre partiel sur un ensemble E est une relation $\preceq$ binaire réflexive, transitive et anti-symétrique. On dit alors que $(E, \preceq)$ et un ensemble partiellement ordonné. On note $\preceq^{-1}$ la relation inverse de $\preceq$, où pour tout $x, y \in E$ on pose $x \preceq^{-1} y$ ssi $y \preceq x$. On abrège $\neg(x \preceq y) \wedge \neg(y \preceq x)$ par $x \| y$. Un ordre total est un ordre partiel $\preceq$ qui est total, i.e. $\forall x, y \in E(x \preceq y \vee y \preceq x)$.

Nouveautés : Soit E un ensemble *fini non-vide* et $\preceq$ un ordre partiel sur E. S'il existe $d \in \mathbb{N}\backslash\{0\}$ vérifiant les deux propriétés suivantes, on appelle d la dimension de $(E, \preceq)$, qu'on note aussi $d = \dim(E, \preceq)$. (Notons que la deuxième propriété implique l'unicité de d.)

- Il existe un ensemble I à d éléments et une famille $(\trianglelefteq_i)_{i \in I}$ d'ordre totaux sur E dont l'intersection est $\preceq$, autrement dit $\preceq$ égal $\cap_{i \in I} \trianglelefteq_i$. Dit encore autrement, $\forall x, y \in E(x \preceq y \Leftrightarrow \forall i \in I, x \trianglelefteq_i y)$.
- Il n'existe pas d'ensemble I non-vide à moins de d éléments vérifiant cette propriété d'intersection.

Question 1. 1. Un ordre total est-il un ordre partiel ? Si oui, a-t-il une dimension ? Si oui laquelle ?

2. Mêmes trois questions pour la relation égalité sur E (i.e. x et y sont en relation ssi $x = y$).

Question 2. Soient $(E, \preceq)$ un ensemble partiellement ordonné de dimension d.

1. Soient $F \subseteq E$ et $\preceq_F$ la restriction de $\preceq$ à F. Cette restriction est-elle un ordre partiel ? Si oui, $(F, \preceq_F)$ a-t-il une dimension ? Si oui peut-on la comparer à d ?
2. Mêmes trois questions pour $\preceq^{-1}$.

Question 3. Soit E et F deux ensembles finis non vides et disjoints. Soit $\preceq_E$ et $\preceq_F$ deux ordres partiels sur E et F de dimension d_E et d_F.

1. Soit $\preceq = pile(\preceq_E, \preceq_F)$ la relation binaire sur $E \cup F$ définie par $x \preceq y$ si $x \in E$ et $y \in F$, ou $x \preceq_E y$, ou $x \preceq_F y$. Est-elle un ordre partiel ? A-t-elle une dimension ? Laquelle ?
2. (Sans justifier) Soit $\preceq$ la relation binaire sur $E \cup F$ définie par $\preceq := \preceq_E \cup \preceq_F$. Est-elle un ordre partiel ? A-t-elle une dimension ? Laquelle ?

Question 4. Soient $(E, \preceq)$ un ensemble fini non-vide partiellement ordonné et $x, y \in E$ incomparables pour $\preceq$.

1. Montrer qu'il existe un ordre partiel $\preceq_{xy}$ contenant $\preceq$ et tel que $x \preceq_{xy} y$.
2. Montrer qu'il existe un ordre total $\trianglelefteq$ contenant $\preceq$ et telle que $x \trianglelefteq y$. (On dit alors que $\trianglelefteq$ est une extension linéaire de $\preceq$.)

Question 5. Soit $(E, \preceq)$ un ensemble fini non-vide partiellement ordonné. Montrer qu'il existe un ensemble fini non-vide I et une famille $(\trianglelefteq_i)_{i \in I}$ d'ordre totaux sur E dont l'intersection est $\preceq$, c'est-à-dire $\preceq = \cap_{i \in I} \trianglelefteq_i$.

Question 6. Soit $(E_i, \leq_i)_{i \in I}$ une famille non vide d'ensembles finis non-vides et totalement ordonnés. Soit $E := \prod_{i \in I} E_i$ le produit Cartésien des E_i et soit $\preceq$ l'ordre produit des $\leq_i$. Plus précisément, $x \preceq y$ si $\forall i \in I, x_i \leq_i y_i$. Comparer $|I|$ et la dimension d de $(E, \preceq)$.

Coupures dans les ordres partiels

On fixe un ensemble E et un ordre partiel $\preceq$ sur E, i.e. $\preceq$ est une relation binaire réflexive, transitive et antisymmétrique. On note $\preceq^{-1}$ la relation inverse de $\preceq$, où pour tout $x, y \in E$ on pose $x \preceq^{-1} y$ ssi $y \preceq x$. On abrège $\neg(x \preceq y) \wedge \neg(y \preceq x)$ par $x \| y$. Un ordre total est un ordre partiel $\preceq$ qui est total, i.e. $\forall x, y \in E(x \preceq y \vee y \preceq x)$.

On appelle segment initial de $(E, \preceq)$ tout ensemble $T \subseteq E$ tel que pour tout $x, y \in E$, si $y \in T$ et $x \preceq y$, alors $x \in T$. De même, un segment final de $(E, \preceq)$ est un ensemble $T \subseteq E$ tel que pour tout $x, y \in E$, si $x \in T$ et $x \preceq y$, alors $y \in T$.

Question 1.
1. Nommer deux segments initiaux de $(E, \preceq)$.
2. Que peut-on dire du complémentaire $E \setminus T$ d'un segment initial T ?

Question 2. Soient $(T_i)_i \in I$ une famille non-vide de segments initiaux.
1. Que peut-on dire de leur intersection $A := \cap_{i \in I} T_i$?
2. Que peut-on dire de leur union $B := \cup_{i \in I} T_i$?

Pour tout $T \subseteq E$, soit $IS(T)$ l'intersection de tous les segments initiaux incluant T, soit $L(T)$ l'ensemble des minorants de T, i.e. $L(T) := \{x \in E \mid \forall y \in T, x \preceq y\}$, soit $U(T)$ l'ensemble des majorants de T.

Question 3.
1. Les fonctions IS, L, U sont-elles croissantes pour l'inclusion ? Décroissantes ? Ni l'une ni l'autre ?
2. Soit $T \subseteq E$. Montrer que $IS(T)$ est le plus petit segment initial incluant T. Quelle propriété a $L(T)$?

Question 4.
1. Soit $T \subseteq E$. Comparer $IS(T)$ et $L(T)$ pour l'inclusion.
2. Caractériser les T tels que $IS(T) = L(T)$.

Question 5.
1. Comparer $IS(T)$ et $L(U(T))$ pour l'inclusion. Pour la suite, on définit $C(T) := L(U(T))$. (On appelle C l'opérateur de coupure.)
2. L'inclusion de la question 5.1 est-elle parfois/toujours/jamais stricte ? Calculer $C(\{x\})$.

Question 6.
1. Pour l'inclusion, C est-elle croissante, décroissante, ni l'une ni l'autre ?
2. Soit $T \subseteq E$. Comparer $C(T)$ et $C \circ C(T)$ pour l'inclusion.
3. Soit $T \subseteq E$. Caractériser $C(T)$ parmi les $C(X)$, $X \subseteq E$.
4. Caractériser les $T \subseteq E$ tels que $T = C(T)$.

Ensemble des parties et ordre inclusion

Soit E un ensemble fini non-vide. Un ordre partiel sur E est une relation binaire $\preceq$ sur E qui est réflexive, transitive et antisymmétrique. On abrège $\neg(x \preceq y) \wedge \neg(y \preceq x)$ par $x \| y$. Un ordre total est un ordre partiel $\preceq$ qui est total, i.e. $\forall x, y \in E, x \preceq y \vee y \preceq x$.

Pour tout ordre partiel $\preceq$ sur E, une chaîne de $(E, \preceq)$ est une partie $C \subseteq E$ telle que la restriction de $\preceq$ à C, notée $\preceq_C$, est un ordre total sur C. Une antichaîne de $(E, \preceq)$ est une partie $A \subseteq E$ dont les éléments sont deux à deux incomparables pour $\preceq$, i.e. $\forall x, y \in A, x \| y$.

Une chaîne C est dite maximale s'il n'existe pas de chaîne C' telle que $C \subsetneq C'$. Une chaîne C est dite de cardinalité maximale s'il n'existe pas de chaîne C' telle que $|C| < |C'|$. On définit de même les antichaînes maximales et les antichaînes de cardinalité maximale. On appelle largeur de $(E, \preceq)$ la cardinalité maximale de ses antichaînes.

Question 1.
1. Comparer les concepts de chaîne maximale et chaîne de cardinalité maximale : l'un implique-t-il l'autre ? L'autre implique-t-il l'un ?
2. Même question pour les antichaînes.

Question 2. Reprendre la question 1 dans le cas $(\mathcal{P}(E), \subseteq)$, où on admet que l'inclusion induit un ordre partiel.

Pour tout $n \in \mathbb{N}$ on note $[n] := \{1, 2, \ldots, n\}$ (en particulier $[0] = \emptyset$), et $P_n := (\mathcal{P}([n]), \subseteq)$.

Question 3. Soit $n \in \mathbb{N}$ arbitraire.
1. Donner un exemple de chaîne de cardinalité maximale dans P_n.
2. Combien y a-t-il de chaîne de cardinalité maximale dans P_n ?
3. Soit $E \subseteq [n]$. Combien de chaînes maximales de P_n contiennent E ?

Question 4. Soient $n \in \mathbb{N}$ et $\mathcal{A}$ une antichaîne de P_n. Pour tout $X \in \mathcal{A}$, soit $\mathcal{C}_X$ l'ensemble des chaînes maximales de P_n qui contiennent X.
1. Comparer $n!$ et $\left| \cup_{X \in \mathcal{A}} \mathcal{C}_X \right|$
2. Comparer $\left| \cup_{X \in \mathcal{A}} \mathcal{C}_X \right|$ et $\sum_{X \in \mathcal{A}} |X|! \cdot (n - |X|)!$

Question 5. En déduire la largeur de P_n.

Question 6. Pour tout $n \in \mathbb{N}$ et $k \in \mathbb{N}$ inférieur ou égal à la largeur de P_n, existe-t-il dans P_n une antichaîne maximale de cardinalité k ?

Cycles eulériens

On considère les graphes finis et orientés $G = (V, E)$. Ainsi, V est fini non vide et E est une relation binaire arbitraire sur V, c'est-à-dire $E \subseteq V \times V$. Toute arête $(u, v) \in E$ est dite sortante de u et entrante dans v. Si $u = v$, on dit aussi que c'est une boucle. Un chemin de G est une suite $v_0, v_1, \ldots, v_n \in V$ où $n \in \mathbb{N}$ et telle que pour tout $i < n$, on a $(v_i, v_{i+1}) \in E$. Le graphe G est dit fortement connexe si pour tout $u, v \in V$ il existe un chemin $v_0, v_1, \ldots, v_n \in V$ tel que $v_0 = u$ et $v_n = v$. Un chemin est dit eulérien s'il emprunte chaque arête exactement une fois. Un cycle dans un graphe est un chemin dont le premier sommet coïncide avec le dernier. Un cycle eulérien est un cycle qui est un chemin eulérien. Le degré sortant $outdeg(v)$ d'un sommet $v \in V$ est le nombre d'arêtes en sortant, et son degré entrant $indeg(v)$ est le nombre d'arêtes y entrant. Un sommet v est dit équilibré si $indeg(v) = outdeg(v)$. Le graphe G est dit équilibré si tous ses sommets sont équilibrés. Il est dit (u, v)-quasi-équilibré si $u, v \in V$ et tout sommet dans $V \setminus \{u, v\}$ est équilibré, et si $indeg(u) = outdeg(u)+1$ et $indeg(v) = outdeg(v)-1$. Il est dit quasi-équilibré s'il existe $u, v \in V$ tel qu'il soit (u, v)-quasi-équilibré.

Question 1. Soit (V, E) et (V, F) deux graphes équilibrés où $F \subseteq E$. Que peut-on dire du graphe $(V, E \setminus F)$?

Question 2.
1. Dans un graphe (u, v)-quasi-équilibré, u et v sont-ils égaux parfois, toujours, jamais ?
2. Comparer les concepts de graphe équilibré et graphe fortement connexe. L'un implique-t-il l'autre ?
3. Comparer les concepts de graphe quasi-équilibré et graphe fortement connexe. L'un implique-t-il l'autre ?

Question 3. Soit $G = (V, E)$ un graphe équilibré.
1. Quelles sont les arêtes $e \in E$ telles que $G' := (V, E \setminus \{e\})$ est équilibré ?
2. Quelles sont les arêtes $e \in E$ telles que $G' := (V, E \setminus \{e\})$ est quasi-équilibré ?

Question 4. Soit $G = (V, E)$ un graphe (u, v)-quasi-équilibré.
1. Quelles sont les arêtes $e \in E$ telles que $G' := (V, E \setminus \{e\})$ est équilibré ?
2. Quelles sont les arêtes $e \in E$ sortant de v telles que $G' := (V, E \setminus \{e\})$ est quasi-équilibré ?

Question 5. Soit un graphe (V, E) avec un cycle eulérien.
1. Montrer que l'ensemble des sommets W de deux degrés non nuls induit un graphe (W, E) fortement connexe.
2. Quelle propriété supplémentaire a le graphe original (V, E) ?

Question 6. Soient $G = (V, E)$ un graphe équilibré et $(v_0, v_1) \in E$. Existe-t-il toujours un cycle qui emprunte (v_0, v_1) et qui soit sans répétition d'arête ?

Un graphe (V, E) est dit faiblement connexe si pour tout $u, v \in V$ il existe $v_0, v_1, \ldots, v_n \in V$ tels que $v_0 = u$ et $v_n = v$ et pour tout $i < n$, on a $(v_i, v_{i+1}) \in E$ ou $(v_{i+1}, v_i) \in E$.

Question 7.
1. Comparer les concepts de forte connexité et faible connexité. L'un implique-t-il l'autre ? L'autre implique-t-il l'un ?
2. Qu'en est-il pour les graphes équilibrés ?

Question 8. Caractériser les graphes (V, E) qui ont un cycle eulérien.

Arbres couvrants

On définit deux types de graphes *sans boucle* et avec possibles *arêtes multiples* entre deux sommets. Soit $V = \{v_1, \dots, v_p\}$ des sommets. Un graphe est défini par une liste d'arêtes $E = (e_1, \dots, e_q)$, avec répétitions possibles. Pour les graphes non-orientés (resp. orientés) chaque arête est une paire $\{v_i, v_j\}$ de sommets (resp. un couple (v_i, v_j) avec $i \neq j$). Pour tout graphe orienté $G = (V, E)$, on définit sa version non-orientée $\overline{G}$ naturellement par $(V, \overline{E})$ où $\overline{E} = (\overline{e_1}, \dots, \overline{e_q})$ et $\overline{(v_i, v_j)} = \{v_i, v_j\}$. On insiste que dans ce problème E n'est pas un ensemble, mais une liste avec répétitions possibles.

Question 1. Soit $g = (V, E)$ un graphe non-orienté. Combien de graphes orientés G sont tels que $\overline{G} = g$?

Soit G un graphe orienté. On définit sa matrice d'incidence $M(G)$ et sa matrice laplacienne $L(G)$.

— pour $i \in [p] := \{1, \dots, p\}$ et $j \in [q]$, on pose $M(G)_{i,j} = \begin{cases} 1 \text{ si } e_j \text{ finit en } v_i \\ -1 \text{ si } e_j \text{ commence en } v_i \\ 0 \text{ sinon} \end{cases}$

— pour $i, j \in [p]$, on pose $L(G)_{i,j} = \begin{cases} -m_{ij} \text{ si } i \neq j \text{ et il y a } m_{ij} \text{ arêtes entre } v_i \text{ et } v_j \text{ dans } \overline{G} \\ deg(v_i) \text{ si } i = j \text{ où } deg(v_i) \text{ est degré de } v_i \text{ dans } \overline{G} \end{cases}$

Question 2.
1. Quelle particularité simple a la matrice laplacienne d'un graphe ?
2. Pour deux graphes orientés G et H tels que $\overline{G} = \overline{H}$, comparer $L(G)$ et $L(H)$. Puis, comparer $M(G)$ et $M(H)$ en terme de leurs colonnes et de leurs rangs.

Question 3. Soit $g = (V, E)$ un graphe non-orienté avec $p \geq 2$ sommets et p arêtes. Montrer que g a un cycle. (On précise que deux arêtes distinctes entre deux mêmes sommets constituent un cycle de longueur 2)

Question 4. Soit G un graphe orienté à p sommets et q arêtes.
1. Montrer que $rang(M(G)) < p$
2. Montrer que si $\overline{G}$ contient un cycle, alors $rang(M(G)) < q$.

Question 5. Trouver un relation entre L et MM^t, où M^t est la transposée de M, et L et M sont des abréviations de $L(G)$, $M(G)$.

On dit d'un graphe non-orienté qu'il est un arbre s'il est connexe et sans cycle. Soit $g = (V, E)$ un graphe non-orienté à p sommets. On admet l'équivalence entre les assertions suivantes.

1. g est un arbre.
2. g est connexe et a $p - 1$ arêtes.
3. g a $p - 1$ arêtes et pas de cycle.

Un arbre couvrant d'un graphe non-orienté $g = (V, E)$ est un sous-ensemble $E' \subseteq E$ tel que (V, E') est un arbre.

Question 6. Soit $G = (V, E)$ un graphe orienté avec $p \geq 2$ sommets et $p - 1$ arêtes. Soit M_0 la matrice obtenue en enlevant la dernière ligne de $M(G)$. Montrer que si $\overline{G}$ est un arbre, alors $|\det M_0| = 1$, et que sinon $\det M_0 = 0$.

Tourner la page SVP

Soient deux entiers $m \leq n$ et $R \subseteq [n]$ avec $|R| = m$. Soit A une matrice avec m lignes et n colonnes. On dénote par $A[R]$ la matrice $m \times m$ obtenue à partir de A en ne gardant que les *colonnes* d'indice dans R. De même si B a n lignes et m colonnes, on dénote par $B[R]$ la matrice $m \times m$ obtenue à partir de B en ne gardant que les *lignes* d'indice dans R. On admet alors que $\det(AB) = \sum_{R \subseteq [n], |R|=m} \det(A[R]) \det(B[R]))$.

Question 7. Soit $G = (V, E)$ un graphe orienté. Soit L_0 la matrice $L(G)$ privée de sa ligne p et colonne p. Montrer que $\det L_0 = \kappa(\overline{G})$, où $\kappa(\overline{G})$ est le nombre d'arbres couvrants de $\overline{G}$.

Facteurs de mots infinis

Soit Σ un ensemble *fini non-vide* (appelé alphabet, d'éléments appelés lettres). L'ensemble des mots finis construits à partir de Σ est noté Σ^*. Le mot vide est noté ε. L'ensemble des mots infinis construits à partir de Σ est noté $\Sigma^{\mathbb{N}}$. Un mot infini α est de la forme $a_0a_1a_3\dots$, où $a_i \in \Sigma$ pour tout $i \in \mathbb{N}$, et correspond formellement à une application $\alpha : \mathbb{N} \to \Sigma$. Pour tout $u, v, w \in \Sigma^*$ et $\alpha \in \Sigma^{\mathbb{N}}$, on note uv la concaténation de u et v et $u\alpha$ la concaténation de u et α, on dit que u est préfixe de uv et de $u\alpha$, que v est suffixe de uv, que α est suffixe de $u\alpha$, et que le mot v est facteur de uvw et de $uv\alpha$. Un mot infini $\alpha \in \Sigma^{\mathbb{N}}$ est dit univers (pour Σ) si chaque mot fini de Σ^* est facteur de α.

Question 1. Montrer qu'il existe un mot univers pour Σ.

Question 2.
1. Soient un mot univers $\alpha \in \Sigma^{\mathbb{N}}$ et $u \in \Sigma^*$. Considérons la situation où u apparait infiniment souvent (i.e. à un nombre infini de positions) en tant que facteur de α. Cette situation arrive-t-elle toujours/parfois/jamais ?
2. Un suffixe d'un mot univers est-il toujours/parfois/jamais univers ?

Soit $w \in \Sigma^*$. Un mot infini $\alpha \in \Sigma^{\mathbb{N}}$ est appelé w-mot si les deux conditions suivantes sont vérifiées :
- w n'est pas facteur de α.
- Tout mot dont w n'est pas facteur est facteur de α.

Question 3.
1. Existe-t-il un ε-mot ?
2. Soit w un mot commençant et terminant par la même lettre $a \in \Sigma$. Existe-t-il toujours/parfois/jamais un w-mot ?

Question 4. On suppose ici que $|\Sigma| \geq 3$. Pour quels $w \in \Sigma^* \setminus \{\varepsilon\}$ existe-t-il un w-mot ?

Question 5. On suppose ici que $\Sigma = \{0, 1\}$. Pour quels $w \in \Sigma^* \setminus \{\varepsilon\}$ existe-t-il un w-mot ?

On étudie maintenant les mots en dimension 2. Pour tout $l, h \in \mathbb{N} \setminus \{0\}$, soit $R_{l,h} := \{0, \dots, l-1\} \times \{0, \dots, h-1\}$ un domaine rectangulaire. Pour tout $x, y \in \mathbb{N}$, soit $(x, y) + R_{l,h} := \{x, \dots, x+l-1\} \times \{y, \dots, y+h-1\}$ la translation de $R_{l,h}$ par le vecteur (x, y). On appelle "mot rectangulaire" toute application $s : \{0, \dots, l\} \times \{0, \dots, h\} \to \Sigma$ avec $l, h \in \mathbb{N}$. On appelle "mot infini 2D" toute application $\beta : \mathbb{N}^2 \to \Sigma$. La notion de facteur se généralise naturellement aux mots rectangulaires et infinis 2D. Un mot infini 2D est univers si tous les mots rectangulaires en sont facteurs. Pour tout mot rectangulaire s, un s-mot est un mot infini 2D β tel que :
- s n'est pas facteur de β.
- Tout mot rectangulaire dont s n'est pas facteur est facteur de β.

Question 6.
1. Existe-t-il un mot infini 2D univers ?
2. Pour tout mot rectangulaire s, existe-t-il un s-mot ?
3. Reprendre toutes les questions avec Σ infini.