Banque MP inter-ENS – Session 2022

Rapport relatif à l'épreuve orale d'informatique fondamentale spécifique Ulm

— **Écoles partageant cette épreuve :** ENS Ulm

— **Coefficients** (en pourcentage du total des points de chaque concours) :
  — Concours MP option MPI : 23,1%
  — Concours INFO : 13,3%

— **Membres du jury :** Laurent Bienvenu, Brice Minaud, Charles Paperman, Pierre Senellart

Modalités de l'épreuve. L'épreuve orale d'informatique fondamentale décrite dans ce rapport est spécifique au concours d'entrée de l'École normale supérieure de Paris, et est entièrement indépendante de l'épreuve analogue qui figure au concours des autres Écoles normales supérieures. L'épreuve dure une heure, sans préparation, et vise à interroger les candidates et candidats sur des questions d'informatique fondamentale au tableau. Elle couvre des notions d'informatique principalement théoriques, mais diffère d'une épreuve de mathématiques en cela que les sujets conduisent en l'étude de notions propres à l'informatique, telles que des algorithmes ou programmes, des langages formels, des graphes, etc. Cette épreuve ne mesure pas la compétence des candidates et candidats en informatique pratique, même s'il leur est parfois demandé de présenter certains points en pseudocode.

Les sujets sont présentés aux candidates et candidats sous forme imprimée, et quelques minutes leur sont laissées pour prendre connaissance des définitions préliminaires et des premières questions. Les examinateurs ont généralement laissé les candidates et candidats traiter le début des sujets en autonomie, en progressant naturellement vers un dialogue interactif pour des questions plus délicates, ou quand il s'avérait nécessaire de préciser certains points des réponses proposées. Les sujets étaient toujours composés d'un unique problème formé de plusieurs questions successives ; pour celles et ceux qui parvenaient à la fin des questions imprimées, les questions suivantes étaient posées directement à l'oral au tableau.

Résultats. Cette année, le jury a examiné 79 candidates et candidats admissibles aux concours MP et INFO. Le jury n'a pas connaissance de quel concours est présenté par les candidates et candidats qu'il auditionne, et les évalue donc de la même manière. Conformément aux instructions fournies pour l'harmonisation, les notes se sont réparties de 5 à 19,4 avec une moyenne de $12,02$ et un écart type de 3,46. Malgré ces écarts inhérents à une épreuve de concours, le jury tient à féliciter l'ensemble des candidats dont le niveau global était élevé.

Programme. L'épreuve porte sur le programme de l'option informatique des *deux* années de classes préparatoires (MPSI et MP), ainsi que sur le programme informatique commun, et peut également faire appel à des compétences mathématiques exigibles suivant les programmes de cette discipline. Il est fortement recommandé aux candidates et candidats de *prendre connaissance de ces programmes* et de s'assurer qu'ils et elles maîtrisent effectivement les points qui y figurent. Les examinateurs ont parfois, au fil de l'oral, demandé à la candidate ou au candidat de préciser un point sous la forme d'une question de cours. Une réponse correcte et immédiate est alors attendue, le cas contraire étant fortement sanctionné.

Bien entendu, les sujets proposés aux candidates et aux candidats demandaient d'explorer des notions nouvelles allant au-delà du programme, et qui étaient donc définies rigoureusement au préalable. Dans l'ensemble, les candidates et candidats se sont bien appropriés ces notions.

Sujets Comme les années précédentes, par souci de transparence, et pour permettre à toutes les candidates et à tous les candidats de préparer cette épreuve de manière équitable, ce rapport de concours inclut en annexe l'intégralité des sujets posés cette année. [1] Soulignons que, présentés tels quels, les sujets sont de difficulté inégale. Pour certains on attendait une résolution complète ou presque quand pour d'autres arriver à la moitié constituait une bonne performance. Par ailleurs, certaines questions s'accompagnaient d'indications orales sans lesquelles la résolution aurait été particulièrement ardue dans le temps imparti.

Critères d'évaluation et recommendations aux candidats Le but des épreuves orales est d'évaluer un ensemble de compétences que l'on espère révélateur d'une aptitude future à faire de la recherche : s'approprier des notions nouvelles, réfléchir en autonomie, expliquer ses intuitions, identifier les difficultés et proposer une ou des approches pertinentes à un problème non trivial, exposer sa solution de façon claire et synthétique, etc.

A ce titre, nous encourageons les candidates et candidats à expliquer à voix haute les approches qu'ils ont tentées lorsqu'ils butent sur une question, afin d'engager un dialogue avec l'examinateur. Une fois la solution à une question trouvée, on attend de la candidate ou candidat qu'il ou elle sache l'exposer clairement, de préférence oralement mais en s'aidant du tableau si nécessaire (par exemple pour écrire un morceau de pseudocode). Certains candidates ou candidats présentent leur solution de façon trop vague ou incomplète quand d'autres tombent dans l'excès inverse en écrivant l'intégralité de la solution au tableau comme s'il s'agissait d'une épreuve écrite (ce qui n'est pas sanctionné en tant que tel mais fait perdre beaucoup de temps). En revanche, si l'examinateur demande une clarification écrite sur un point précis, la candidate ou candidat ne doit pas y rechigner. Un point important de l'évaluation porte sur la capacité de la candidate ou du candidat à réagir aux indications de l'examinateur pour continuer à avancer. Rester bloqué dans une voie sans issue en ignorant les indications est fortement pénalisant.

Un autre point délicat est le niveau de formalisme attendu. Parfois, la définition de l'objet d'étude nécessitait un niveau élevé de formalisme initial pour plus de précision, mais on attendait de la candidate ou candidat qu'il ou elle sache s'affranchir de celui-ci une fois la définition comprise. Dans d'autres cas, le sujet revêtait un caractère intrinsèquement formel, et l'on attendait à l'inverse que la candidate ou candidat sache jongler avec ce formalisme durant l'oral, ce qui n'a hélas pas toujours été le cas. Dans un cas comme dans l'autre, une réponse informelle n'est bien sûr pas pénalisée dès l'instant où le candidat sait préciser sa réponse sur demande de l'examinateur.

Le jury a pu constater que le programme est dans l'ensemble bien maîtrisé. Des lacunes ont cependant été constatées sur le thème des langages formels et automates finis. Alors que la clôture par intersection des langages réguliers est au programme, certains candidats ne connaissent pas la construction de l'automate produit ni ne savent la retrouver avec indications. Par ailleurs, le jury estime que les candidates et candidats doivent être capables d'exhiber un langage non-régulier et de prouver sa non-régularité.

Annexe. La suite de ce document présente l'intégralité des sujets qui ont été posés, sous la forme des feuilles distribuées aux candidates et candidats.

1. Des exemples de corrigés de ces mêmes exercices, donnés à titre indicatif, sont disponibles depuis `https://diplome.di.ens.fr/informatique-ens/`.

B1 – Graphe coucou

Soit X un ensemble de cardinalité n. On appelle « objets » les éléments de X. Soit V un ensemble de cardinalité $m \geq n$. On appelle « cases » les éléments de V. On suppose qu'on dispose de deux fonctions $H_1 : X \to V$ et $H_2 : X \to V$.

Une *assignation* des objets X dans les cases V est une fonction $f : X \to V$. On dit alors que x est assigné dans la case $f(x)$. L'assignation f est *valide* si elle respecte les contraintes suivantes :

1. l'objet $x \in X$ ne peut être assigné que dans la case $H_1(x)$ ou dans la case $H_2(x)$;
2. chaque case reçoit au plus un objet.

On dit qu'une configuration (X, V, H_1, H_2) est *acceptable* s'il existe une assignation valide.

Question 0 (Préliminaires).

a. Donner un exemple de configuration acceptable, et de configuration non acceptable.

b. Proposer un algorithme naïf pour tester si une configuration est acceptable. Quelle est sa complexité ?

Question 1. Le *graphe coucou* (où « coucou » fait référence à l'oiseau) $G(X, V, H_1, H_2)$ est le graphe non-orienté dont l'ensemble des sommets est V, et dont les arêtes sont $\{H_1(x), H_2(x)\}$ pour $x \in X$. On note que ce graphe peut contenir plusieurs arêtes entre deux sommets fixés (lorsqu'il existe $x \neq x'$ tels que $\{H_1(x), H_2(x)\} = \{H_1(x'), H_2(x')\}$), et qu'il peut contenir des boucles (lorsqu'il existe x tel que $H_1(x) = H_2(x)$). On continuera néanmoins d'utiliser le terme « graphe » pour ces objets, et c'est systématiquement dans ce sens que le terme graphe est utilisé dans la suite.

Montrer que l'énoncé « (X, V, H_1, H_2) est acceptable » peut être reformulé de manière équivalente sous la forme : « il existe une manière d'orienter les arêtes de $G(X, V, H_1, H_2)$ de telle sorte que le graphe orienté qu'on obtient satisfait une condition simple ».

Question 2. On dit qu'un graphe connexe $G = (V, E)$ est *complexe* si $|E| > |V|$ (où $|E|$ désigne les arêtes comptées avec leur multiplicité).

a. Montrer qu'un graphe connexe $G = (V, E)$ est un arbre (c'est-à-dire : connexe et acyclique) si et seulement si $|E| = |V| - 1$.

b. Montrer qu'un graphe connexe est non complexe si et seulement si il contient au plus un cycle.

c. Montrer que si $G(X, V, H_1, H_2)$ ne contient pas de composante connexe complexe, la configuration associée est acceptable.

d. Montrer la réciproque : (X, V, H_1, H_2) est acceptable si et seulement si $G(X, V, H_1, H_2)$ ne contient pas de composante connexe complexe.

Question 3. Proposer un algorithme en temps linéaire qui prend en entrée un graphe coucou G, et détermine si G est acceptable.

Question 4. Si une configuration est acceptable, quel est le nombre d'assignations valides ?

Question 5. Supposons qu'on connaît une assignation valide f pour la configuration (X, V, H_1, H_2). On souhaite ajouter un nouvel élément $x \notin X$ dans X ; on suppose que $H_1(x)$ et $H_2(x)$ sont déjà définis. Pour cela, on utilise l'algorithme suivant. Soit $x_0 = x$. On insère x_0 dans la case $H_1(x_0)$. Si la case est libre, on a fini. Sinon, soit x_1 l'élément tel que $f(x_1) = H_1(x_0)$. Supposons $f(x_1) = H_1(x_1)$ (le cas $f(x_1) = H_2(x_1)$ est similaire). On insère toujours x_0 dans $H_1(x_0)$, mais on déplace l'élément x_1 de la case $H_1(x_1)$ vers la case $H_2(x_1)$. Si cette case est libre, on a fini. Sinon, on continue d'itérer le même processus. (Cette manière de « voler » les cases déjà occupées est ce qui donne le nom « coucou » à l'algorithme.)

Montrer que si $(X \cup \{x\}, V, H_1, H_2)$ est acceptable, l'algorithme précédent termine. Borner sa complexité.

Question 6. On s'intéresse maintenant au cas où chaque case peut recevoir jusqu'à k objets. On dit qu'une configuration est k-acceptable s'il existe une assignation telle que chaque case reçoit au plus k objets.

a. Traduire la propriété d'être k-acceptable en termes d'orientabilité des arêtes du graphe coucou associé, dans l'esprit de la question 1.

b. Pour $G = (V, E)$ un graphe et $V' \subseteq V$, le sous-graphe *induit* par V' est le graphe $(V', \{\{x, y\} : \{x, y\} \in E \text{ et } x, y \in V'\})$. Montrer que G est k-acceptable si et seulement si tout sous-graphe induit (V', E') de G vérifie $|E'| \leq k|V'|$.
Indication pour le sens difficile : Pour une assignation donnée, on peut définir le *surplus* d'une case qui reçoit t objets comme étant $\max(0, t - k)$. Le *surplus* de l'assignation est la somme des surplus de toutes les cases. Si la configuration n'est pas k-acceptable, on peut considérer une assignation dont le surplus est minimal (nécessairement ≥ 1). Soit Δ l'ensemble des sommets de surplus strictement positif pour cette assignation. Raisonner sur $\Delta' \supseteq \Delta$ l'ensemble des sommets depuis lesquels il existe un chemin orienté vers un sommet de Δ (ou dans l'autre sens, suivant le sens dans lequel le candidat a choisi d'orienter les arêtes à la question 1).

c. Montrer que dans le cas $k = 1$, cette condition est bien équivalente à celle de la question 2c.

Question 7. On revient au cas $k = 1$. Cette fois, on autorise à stocker un objet x en partie dans $H_1(x)$, et en partie dans $H_2(x)$. Plus précisément, une $\mathbb{Q}$-*assignation* des objets X dans les cases V est une fonction $f : X \to ([0, 1] \cap \mathbb{Q})^m$. On note $f(x)[i]$ la i-ième composante de $f(x)$. La $\mathbb{Q}$-assignation est *valide* si elle respecte les contraintes suivantes :

1. l'objet $x \in X$ est assigné entièrement dans $H_1(x)$ et $H_2(x)$, formellement : $\sum_{v \in V} f(x)[v] = 1$, mais $\forall v \notin \{H_1(x), H_2(x)\}, f(x)[v] = 0$;
2. chaque case contient au plus l'équivalent d'un objet, formellement : $\forall v, \sum_{x \in X} f(x)[v] \leq 1$.

Une configuration est dite $\mathbb{Q}$-acceptable s'il existe une $\mathbb{Q}$-assignation valide.

a. Soit une configuraiton (X, V, H_1, H_2) et le graphe coucou associé G. Soit G^k le graphe obtenu en multipliant la multiplicité de toutes les arêtes de G par k. Montrer que G est $\mathbb{Q}$-acceptable si et seulement si il existe un certain k tel que G^k est k-acceptable.

b. Montrer qu'une configuration (X, V, H_1, H_2) est $\mathbb{Q}$-acceptable si et seulement si elle est acceptable.

Corrigé

Question 0a. La question 2c donne la recette pour construire des exemples. Pour le cas acceptable, on peut prendre $|X| = |V| = 1$ et il n'y a pas le choix. Pour le cas non acceptable, on peut prendre $|X| = |V| = 2$, et $H_1 = H_2$ des fonctions constantes.

Question 0b. L'algorithme le plus naïf est de tester tous les choix possibles pour chaque $f(x)$, entre $H_1(x)$ et $H_2(x)$. La complexité de cet algorithme est exponentielle.

Question 1. Pour une assignation f, on peut orienter l'arête $\{H_1(x), H_2(x)\}$ vers $f(x)$ (si c'est une boucle, il n'y a qu'un orientation possible). L'assignation est valide ssi le graphe orienté obtenu vérifie que tout sommet a un degré entrant au plus 1.

Question 2a. Il est clair que dans un arbre, $|E| = |V| - 1$. On peut le montrer par récurrence. On peut aussi le voir en choisissant une racine, et en associant à chaque arête le sommet adjacent le plus éloigné de la racine : cette association définit une injection $E \to V$, où seule la racine n'a pas d'arête associée. Réciproquement, si $|E| = |V| - 1$ dans un graphe connexe, si on prend un arbre couvrant, il doit vérifier la même égalité, or il a le même nombre de sommets, donc il a le même nombre d'arêtes et coïncide avec le graphe.

Question 2b. Pour un arbre, $|V| = |E| + 1$. Pour un graphe connexe avec un seul cycle, en retirant un sommet du cycle on obtient un arbre, donc $|V| = |E|$. Un graphe connexe qui contient au plus un cycle est donc non complexe. Réciproquement, si un graphe connexe contient au moins deux cycles, il faut supprimer au moins deux arêtes distinctes pour avoir un arbre, donc $|E| > |V|$.

Question 2b. On peut oublier la configuration et raisonner sur le graphe. Le graphe est acceptable ssi chacune de ses composantes connexes est acceptable. D'après la question précédente, une telle composante est soit un arbre, dans ce cas on fixe une racine, et on oriente tout vers les feuilles. Sinon, c'est un cycle + des arbres sortant du cycle. Pour le cycle, on oriente tout dans le même sens, et pour les arbres sortant du cycle, on oriente tout vers les feuilles.

Question 2d. Pour la réciproque, si un graphe connexe est complexe, $|E| > |V|$, donc par le principe des tiroirs au moins une case reçoit au moins deux objets. On peut raisonner de manière équivalente sur les degrés en utilisant que la somme des degrés entrants de chaque sommet $= |E|$. Il y a aussi une manière plus longue, qui consiste à observer que si on a deux cycles dans la même composante, soit ils s'intersectent, soit ils sont liés par un chemin, et dans les deux cas il n'y a pas moyen d'assigner de manière valide.

Question 3. Il suffit de vérifier $|V| \geq |E|$ pour chaque composante connexe (V, E). Donc il suffit de trouver les composantes connexes, ce qui se fait en temps linéaire. De manière équivalente, on peut générer successivement des arbres couvrants maximaux, et vérifier à chaque fois qu'il y a au plus une arête hors de l'arbre liant deux sommets de l'arbre.

Question 4. Pour un arbre, on peut orienter en fixant une racine et en orientant vers les feuilles, donc il y a au moins $|V|$ choix. Réciproquement, pour montrer qu'il n'y a pas d'autre choix, on peut utiliser $|V| > |E|$ pour argumenter qu'au moins un sommet est de degré entrant 0, à partir de là toute l'orientation est déterminée (ce sommet joue le rôle de la racine). Il y a donc $|V|$ choix exactement.

Pour une composante connexe avec un seul cycle, il y a deux orientations pour le cycle, et aucun choix pour le reste.

Si le graphe a a composantes qui ont un cycle, et b composantes qui sont des arbres de tailles $t_1, \ldots, t_b$, le nombre d'assignations possibles est $2^a \cdot t_1 \cdots t_b$.

Question 5. Ca se voit mieux sur un dessin. Supposons $H_1(x) \neq H_2(x)$ (l'autre cas est similaire). En suivant l'algorithme, soit on atteint finalement une case libre, et on a gagné. Soit on finit par revenir sur un sommet v qu'on a déjà visité. Dans ce cas, si on a suivi un chemin $v_1, \ldots, v_k$ pour arriver de $v_1 = H_1(x)$ jusqu'au sommet $v_k = v$ la première fois, alors quand on revient sur v la deuxième fois, on repart dans l'autre sens en visitant $v_k \ldots v_1$. Rendu à $v_1 = H_1(x)$, on va ensuite sur $H_2(x)$, et on recommence la même exploration « de l'autre côté ». Comme avant, soit on arrive finalement à une case libre, soit on rencontre un cycle. Mais le second cas est impossible, parce que le graphe est acceptable, donc on termine. Au final, chaque sommet de la composante est visité au plus 2 fois, donc on peut borner la complexité par la taille de la composante contenant l'arête $\{H_1(x), H_2(x)\}$.

Question 6a. Le graphe est k-acceptable ssi il est possible d'orienter les arêtes de sorte que le degré entrant de chaque arête soit au plus k.

Question 6b. S'il existe $V' \subseteq V$ tel que le graphe induit (V', E') vérifie $|E'| > k|V'|$, par le principe des tiroirs on ne peut pas placer tous les $|E'|$ objets dans les $|V'|$ cases, donc le graphe n'est pas acceptable.

La réciproque est le sens difficile. *Claim :* Pour $V' = \Delta'$ comme donné dans l'indication, le graphe (V', E') induit par V' satisfait $|E'| > k|V'|$. Pour la suite, il est plus commode d'inverser les orientations, donc le nombre d'objets dans une case correspond au degré *sortant* du sommet. On argumente que tous les sommets de Δ' ont un degré sortant au moins k, sinon on prend un chemin orienté d'un sommet de Δ (de degré sortant $> k$) vers le sommet de $V' = \Delta'$ de degré sortant $< k$, chemin qui existe par définition de Δ'. En inversant le sens des arêtes le long du chemin, on diminue strictement le surplus, ce qui contredit la minimalité. Donc tous les sommets de V' ont un degré sortant au moins k. Par ailleurs, V' est clos en suivant des chemins orientés dans G, donc le degré *sortant* de tous ces sommets est le même dans G et dans le graphe induit par V'. Comme certains de ces sommets (ceux de Δ) ont un degré sortant $> k$, et tous un degré sortant au moins k, en utilisant que $|E'|$ est égal à la somme des degrés sortants des sommets de V', on conclut $|E'| > k|V|$.

Question 6c. Il suffit d'argumenter que la condition « 1-acceptable » est vraie pour tout sous-graphe induit ssi elle est vraie pour les graphes induits correspondants aux composantes connexes. Dans le sens direct c'est immédiat. Pour la réciproque, si on prend un sous-graphe induit quelconque, alors ses composantes connexes contiennent au plus un cycle, parce que la propriété « toutes les composantes contiennent au plus un cycle » est clairement préservée par passage à un sous-graphe, donc toutes les composantes du sous-graphe vérifient $|E| \leq |V|$, donc le sous-graphe aussi.

Question 7a. Supposons que la configuration est $\mathbb{Q}$-acceptable, et prenons une assignation f qui en témoigne. On prend pour k le PPCM des dénominateurs des $f(x)[v]$, et on multiplie f par k, de sorte que l'image de f vive dans les entiers$\{0, \ldots, k\}$, au lieu de vivre dans $[0, 1] \cap \mathbb{Q}$. On réinterprète alors $f(x)[H_1(x)]$ comme le fait qu'on oriente $f(x)[H_1(x)]$ parmi les k arêtes de G^k entre $H_1(x)$ et $H_2(x)$ provenant de l'objet x vers $H_1(x)$, et $f(x)[H_2(x)]$ parmi ces arêtes vont dans l'autre sens. f témoigne alors du fait que G^k est k-acceptable. La réciproque est similaire.

Question 7b. Il est clair que si une configuration est acceptable, elle est $\mathbb{Q}$-acceptable. Dans l'autre sens, si la configuration est $\mathbb{Q}$-acceptable, alors par Q7a, G^k est k-acceptable pour un certain k, donc par Q6b tout sous-graphe induit de G^k vérifie $|E| \leq k|V|$, donc tout sous-graphe induit de G vérifie $|E| \leq |V|$, donc par Q2, G est acceptable.

B2 – Théorème de Gale-Ryser

On note $[\![a, b]\!]$ l'ensemble d'entiers $\{a, a+1, \ldots, b\}$. Soit $S_n = [\![0, n]\!]^n$ les séquences de longueur n dans $[\![0, n]\!]$. Pour $a \in S_n$, on note $a = (a_1, \ldots, a_n)$ les coordonnées de a.

Étant donnés $a, b \in S_n$ tels que $\sum a_i = \sum b_i$, on dit que b *majorise* a, noté $a \preceq b$, si et seulement si :

$$\forall k \in [\![1, n]\!], \quad \sum_{i=1}^{k} a_i \leq \sum_{i=1}^{k} b_i.$$

On rappelle qu'un ordre est une relation réflexive, transitive et anti-symétrique.

On note $\mathcal{M}_n = \{0, 1\}^{n \times n}$ l'ensemble des matrices $n \times n$ à valeur dans $\{0, 1\}$.

Question 0.

a. Montrer que $\preceq$ est un ordre sur S_n.

b. Donner un exemple de deux séquences $a, b \in S_n$ avec $\sum a_i = \sum b_i$, telles que a et b sont incomparables pour $\preceq$.

Question 1. Soit $a \in S_n$. On appelle *conjugué* de a la séquence $a^* \in S_n$ définie par $a^*_i = |\{k \in [\![1, n]\!] : a_k \geq i\}|$.

a. Montrer que a^* est triée par ordre décroissant, et que $\sum_{i \in [\![1,n]\!]} a_i = \sum_{i \in [\![1,n]\!]} a^*_i$.

b. On suppose a triée par ordre décroissant. On définit la matrice $M(a) = (m(a)_{i,j})_{i,j \in [\![1,n]\!]} \in \mathcal{M}_n$ associée à a de la manière suivante : la i-ième ligne de $M(a)$ contient des 1 sur les a_i premières colonnes, et des 0 ensuite. Quelle relation existe entre la matrice associée à a, et la matrice associée à a^* ?

c. Montrer que pour tout $a, b \in S_n$, $a^* = b^*$ si et seulement si il existe une permutation π de $[\![1, n]\!]$ telle que $a_i = b_{\pi(i)}$.

d. Montrer que a est trié par ordre décroissant si et seulement si $a^{**} = a$.

Question 2. Pour $M = (m_{i,j})_{i,j \in [\![1,n]\!]} \in \mathcal{M}_n$, on note $r(M) \in S_n$ la séquence des sommes des lignes de M, *i.e.* $r(M)_i = \sum_{j \in [\![1,n]\!]} m_{i,j}$. On note de même $c(M) \in S_n$ la séquences des sommes des colonnes de M.

Soient $r, c \in S_n$ avec $\sum r_i = \sum c_i$. On dit que (r, c) est *compatible* si et seulement si il existe une matrice $M \in \mathcal{M}_n$ telle que $r = r(M)$ et $c = c(M)$.

a. Donner un exemple de deux séquences $r, c \in S_n$ avec $\sum r_i = \sum c_i$, telles que (r, c) est incompatible.

b. Soient r', c' les séquences définies par $r'_i = r_{\pi(i)}$ et $c'_i = c_{\rho(i)}$, pour des permutations π, ρ quelconques. Montrer que (r, c) est compatible si et seulement si (r', c') est compatible.

c. Suite à la question précédente, on peut donc supposer sans perte de généralité que r et c sont triées par ordre décroissant. Montrer que si (r, c) sont compatibles, alors $r \preceq c^*$.

Question 3. On souhaite montrer la réciproque. Soit $r, c \in S_n$ avec $\sum r_i = \sum c_i$ et $r \preceq c^*$. On considère l'algorithme suivant, qui tente de créer une matrices d'adjacence $n \times n$ dont les sommes des lignes et sommes des colonnes correspondent à (r, c). L'algorithme part de la matrice nulle, et parcourt les lignes dans l'ordre $1, \ldots, n$, et dans chaque ligne, parcourt les entrées dans l'ordre $1, \ldots, n$. À chaque entrée aux coordonnées (i, j), l'algorithme remplace 0 par 1 si et seulement si la somme de la ligne i reste inférieure à r_i, et la somme de la colonne j reste inférieure à c_j.

Montrer que la matrice produite par cet algorithme témoigne du fait que (r, c) est compatible.

Indication : on peut faire une récurrence sur le nombre de coordonnées non nulles dans r.

Question 4. Soient $a, b \in S_n$ avec $\sum a_i = \sum b_i$. Montrer $a^* \preceq b^* \Leftrightarrow b^{**} \preceq a^{**}$.
Indication : il est possible de le déduire des questions précédentes.

Question 5. Soit G un graphe (non-orienté) biparti : les sommets de G sont partitionnés en deux ensembles V, W, et toutes les arêtes sont de la forme $\{v, w\}$ avec $(v, w) \in V \times W$. On associe à G la séquence $d(v)_{v \in V}$ des degrés des sommets de V, et la séquence $d'(w)_{w \in W}$ des degrés des sommets de W.

Étant donné deux séquences finies d'entiers d, d' de longueur respectives n et m, proposer un algorithme qui détermine en temps polynomial s'il existe un graphe biparti avec $|V| = n$ et $|W| = m$, tel que d et d' sont les séquences des degrés de V et W.

Question 6. Soit Ω un ensemble, et $\mathcal{A}$ une partition de Ω. On dit que $S \subseteq \Omega$ est *transverse* si pour tout $A \in \mathcal{A}$, $S \cap A$ contient au plus un élément. On pose $\mathcal{A} = \{A_1, \ldots, A_n\}$ avec $a_i = |A_i|$ tel que $a_1 \geq \cdots \geq a_n$. Soit $r_1 \geq \cdots \geq r_m$ une séquence d'entiers dans $[\![0, n]\!]$.

Montrer qu'il existe une famille de transverses deux à deux disjoints de cardinalités $r_1, \ldots, r_m$ si et seulement si $(r_n^*, \ldots r_1^*) \preceq (a_n, \ldots, a_1)$. (La condition ne requiert pas $\sum_{i \in [\![1,n]\!]} a_i = \sum_{i \in [\![1,n]\!]} r_i^*$.)

Corrigé

Question 0a. On peut vérifier à la main que c'est réflexif, transitif, antisymétrique. Alternativement, on peut argumenter que pour k fixé, la condition $\sum_{i\leq k} a_i \leq \sum_{i\leq k} b_i$ définit un préordre parce que c'est de la forme $f(a) \leq f(b)$ pour f fixée et $\leq$ est un ordre, et $\preceq$ est l'intersection de ces préordres donc aussi un préordre, donc il y a seulement à montrer la réflexivité.

Question 0b. On peut prendre (3,1,1,1) et (2,2,2,0).

Question 1a. On peut le faire directement, ou le déduire de la question suivante.

Question 1b. Les matrices $M(a)$ et $M(a^*)$ sont transposées l'une de l'autre. Une manière de le faire apparaître est de définir $M(a)$ de manière équivalente par : $m(a)_{i,j} = 1$ si et seulement si $a_j \geq i$. Si on écrit la somme de la i-ème ligne, on retombe sur la définition de a^*_i.

Question 1c. Immédiat.

Question 1d. a^{**} est en ordre décroissant suite à 1a. Dans l'autre sens, si on suppose a en ordre décroissant, on remarque que $a \mapsto M(a)$ est inversible sur les suites a en ordre décroissant, donc $a^{**} = a$ s'ensuit du fait que la transposition de matrice est involutive.

Question 2a. On peut prendre (2,0) et (2,0) : on ne peut pas avoir les deux « 1 » à la fois sur la même colonne et sur la même ligne.

Question 2b. Si (r, c) est compatible avec comme témoin la matrice M, alors (r', c') est compatible avec comme témoin la matrice M' obtenue en permutant les lignes et colonnes de M selon π et ρ.

Question 2c. Soit M une matrice qui témoigne du fait que (r, c) est compatible. Une manière de faire est de montrer que $M = M(r)$ est un « pire cas », comme suit. Si toutes les colonnes de M sont de la forme 1^k0^{n-k} alors $M = M(r)$ et on a gagné : en effet dans ce cas, par Q1, $c = r^*$, donc l'inégalité à vérifier devient $r \preceq r^{**}$. Or par Q1 à nouveau, $r^{**} = r$. Sinon, on pose $M_1 = M$, et on prend une colonne de M_1 qui n'est pas de la forme 1^k0^{n-k}. On la modifie pour qu'elle soit de cette forme, en préservant le nombre de 1. La nouvelle matrice M_2 ainsi obtenue vérifie $c(M_1) = c(M_2)$, et $r(M_1) \preceq r(M_2)$. En itérant ce processus, on arrive à la matrice $M_\ell = M(r)$. Comme $r(M_i)$ ne fait qu'augmenter pour $\preceq$, on a fini.

Question 3. On peut faire une récurrence sur le nombre de coordonnées non nulles dans r. Sur la première ligne, l'algorithme place toujours r_1 « 1 » suivis de zéros. L'idée est de « supprimer » cette ligne : plus exactement, on remplace r_1 par 0 (quitte à permuter r pour maintenir l'ordre décroissant), et on diminue de 1 les r_1 premières coordonnées de c (quitte à permuter, à nouveau). On remarque que l'algorithme décrit est équivalent à faire un appel récursif au même algorithme avec (r, c) modifié comme ci-dessus (à une permutation des lignes et des colonnes près). Pour faire une induction, il reste à montrer que la transformation préserve la contrainte $r \preceq c^*$, ce qui ne pose pas de difficulté.

Question 4. En utilisant Q1, de manière équivalente, il suffit de prouver $a \preceq b \Leftrightarrow b^* \preceq a^*$ lorsque a, b sont triés par ordre décroissant. Ou encore de manière équivalente, $a \preceq b^* \Leftrightarrow b \preceq a^*$ (substitution de variable $b \mapsto b^*$). Or la condition à gauche est équivalente au fait que (a, b) est compatible via les questions précédentes, et la condition à droite de même pour (b, a). Mais (a, b) est clairement compatible ssi (b, a) est compatible.

Question 5. C'est immédiat en passant par la matrice d'adjacence : il suffit de vérifier $d \preceq d'^*$ (quitte à ajouter de sommets de degré 0 pour avoir la même longueur).

Question 6. Pour montrer que la condition $(s_1, \ldots, s_n) \preceq (r_1^*, \ldots r_n^*)$ doit être vraie si les transverses existent, on peut considérer la matrice booléenne dont les lignes sont indexées par les transverses, et les colonnes par les ensembles de $\mathcal{A}$. Une coordonnée vaut 1 si et seulement si les deux ensembles correspondant à la ligne et la colonne ont une intersection non nulle. Pour cette matrice, la séquence des sommes des lignes est r, et la somme c_i de la i-ième colonne est au plus a_i. Soit π une permutation telle que $c \circ \pi$ est triée par ordre décroissant, on a donc $(c_{\pi(1)}, \ldots, c_{\pi(n)}) \preceq (r_1^*, \ldots, r_n^*)$, qui implique $(r_n^*, \ldots, r_1^*) \preceq (c_{\pi(n)}, \ldots, c_{\pi(1)}) \preceq (a_n \ldots, a_1)$.

Dans l'autre sens, soit $\delta = \sum_{i \in [\![1,n]\!]} a_i - \sum_{i \in [\![1,n]\!]} r_i$. On augmente les séquences s et r en créant les éléments $s_0 = \max(a_1, r_1 - \delta, m)$ et $r_0 = s_0 + \delta$. Ces éléments sont choisis de sorte que $\sum_{i \in [\![0,n]\!]} a_i = \sum_{i \in [\![0,n]\!]} r_i$, que les séquences restent décroissantes, et que $(s_0, \ldots, s_n) \preceq (r_0^*, \ldots, r_n^*)$. On peut alors appliquer Q3 pour construire la matrice d'adjacence qui correspond. Les transverses sont construites en suivant la procédure inverse de ce qu'on avait fait dans le sens direct (on donne un élément de A_i à la j-ième transverse ssi il y a un 1 dans l'entrée (i, j) de la matrice, et en ignorant la ligne et la colonne d'indice 0).

B3 – Tri aveugle

Un serveur dispose d'un nombre non borné de blocs mémoire, indexés par $\mathbb{N}^*$. Le serveur n'effectue aucun calcul. D'un autre côté, un client effectue des calculs arbitraires, mais dispose seulement de deux blocs mémoire, plus des registres qui permettent de stocker des variables auxiliaires : compteurs, etc.

Les seules interactions possibles entre le client et le serveur sont de la forme suivante : le client envoie au serveur une requête $\mathsf{query}(t, a, b)$. Ensuite :

— si $t = 0$, le serveur renvoie le contenu du a-ième bloc (« accès en lecture »).

— si $t = 1$, le serveur remplace le contenu du a-ième bloc par b (« accès en écriture »).

Tri aveugle. Initialement, la mémoire du serveur contient une liste d'objets $X = (x_1, x_2, \dots, x_n)$. L'objet x_i est stocké dans le bloc i. Le client souhaite trier ces objets suivant un ordre $<$.

Pour cela, le client exécute un algorithme $\mathcal{A}$, qui inclut des instructions query. Soit (t_i, a_i, b_i) les arguments successivement pris par query au cours d'une exécution donnée de l'algorithme. On appelle $(a_1, a_2, \dots)$ la *séquence d'accès* de cette exécution.

Pour X et $<$ fixés, si $\mathcal{A}$ est probabiliste, il induit une distribution $\mathcal{D}_{\mathcal{A}}(X, <)$ sur les séquences d'accès : la probabilité de chaque séquence dépend des choix aléatoires effectués par l'algorithme. Si $\mathcal{A}$ n'est pas probabiliste, $\mathcal{D}_{\mathcal{A}}(X, <)$ est concentré en un point.

Un algorithme de tri est dit « aveugle » si $\mathcal{D}_{\mathcal{A}}(X, <) = \mathcal{D}'_{\mathcal{A}}(n)$ dépend uniquement de n. Informellement : un tri est aveugle si les accès mémoire de l'algorithme ne contiennent aucune information sur l'ordre dans lequel les éléments sont triés.

Question 1.

a. Nommer un algorithme de tri en $O(n \log n)$. Cet algorithme est-il aveugle ?

b. Proposer un algorithme de tri aveugle pour trier deux objets. Proposer un algorithme de tri aveugle pour n objets en temps $O(n^2)$.

Question 2. Un algorithme de *tri aléatoire* est un algorithme qui choisit un ordre $<_\$$ uniformément aléatoirement, et trie suivant cet ordre. Un algorithme de tri aléatoire aveugle est un algorithme de tri aléatoire qui est aveugle au même sens que plus haut.

a. Soit G un groupe, et $g \in G$ fixé. Soit u un élément tiré uniformémement dans G. Montrer que la distribution de gu est uniforme.

b. Étant donné un algorithme de tri aléatoire aveugle en temps T, proposer un algorithme de tri aveugle en temps $T + O(n \log n)$.

Question 3. Dans toute la suite, on suppose que n est une puissance de 2. Soit $z = O(\log n)$ un entier qui divise n. Dans tout l'exercice, on suppose qu'on sait tirer une fonction h uniformément aléatoirement parmi les fonctions $X \to \{0,1\}^z$. Tirer un tel h et l'évaluer se fait en temps constant.

Pour réaliser un tri aléatoire, on souhaite trier X suivant l'ordre induit par $h : x <_h y \Leftrightarrow h(x) \prec h(y)$, où $\prec$ est l'ordre lexicographique. Pour cela, on crée sur le disque n/z « paquets », pour l'instant de n blocs chacun. Initialement, le i-ième paquet contient les objets $x_{z(i-1)+1}, x_{z(i-1)+2}, \dots, x_{z(i-1)+z}$. On suppose qu'on dispose d'un algorithme Compaction qui prend en entrée deux paquets et une fonction $f : X \to \{0,1\}$, et place les objets dont l'image par f est 0 dans le premier paquet, et les objets dont l'image par f est 1 dans le second paquet.

En faisant $O(n \log n)$ appels à Compaction, proposer un algorithme qui permet d'obtenir des paquets triés suivant $<_h$, c'est-à-dire : les éléments du paquet i sont tous inférieurs pour $<_h$ aux éléments du paquet i+1. On ne se préoccupe pas d'être aveugle.

Question 4. Montrer que pour tout paquet construit par COMPACTION au cours de l'algorithme précédent, il existe un entier i tel que la probabilité que le paquet reçoive plus de $6z$ objets est :

$$g(z,t) = \sum_{k=6z+1}^{zt} C_{zt}^{k} t^{-k} (1-t^{-1})^{zt-k} \qquad \text{où } t = 2^i.$$

Question 5. Le but de cette question est de montrer que g décroit exponentiellement avec z.

a. Montrer :
$$C_n^k \leq \frac{n^k}{k!} \leq \left(\frac{ne}{k}\right)^k.$$

b. En déduire $g(z,t) \leq 2^{-6z}$.
Indication : montrer $g(z,t) \leq \sum_{k=6z+1}^{\infty} 2^{-k}$.

Question 6. Suite à la question précédente, on suppose dorénavant que tous le paquets traités par COMPACTION contiennent au plus $6z$ objets. On fixe la taille des paquets à $6z$ exactement, quitte à remplir avec des objets factices.

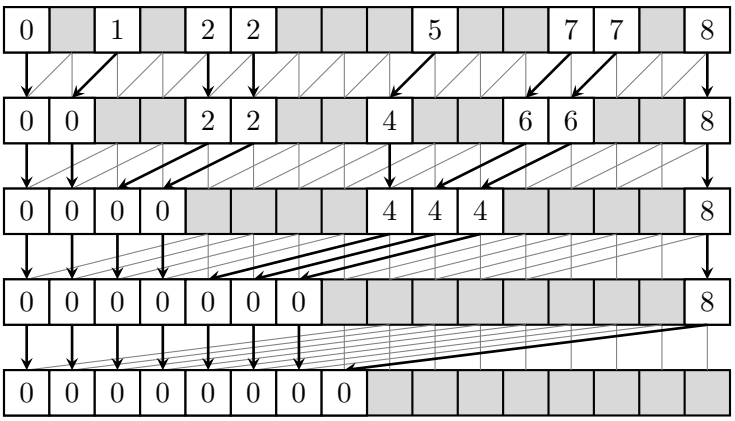
a. Au terme de l'algorithme de la question 3, on obtient n/z paquets triés entre eux contenant des objets réels (ceux de X) et des objets factices. On souhaite créer un algorithme qui extrait la liste triée X de ces paquets, pour réaliser un tri aléatoire aveugle. Cet algorithme d'extraction peut-il se permettre de révéler (via ses accès mémoire) combien d'objets réels sont contenus dans chaque paquet ?
Indication : montrer que la distribution des nombres d'objets par paquet ne dépend pas de l'ordre $<_h$.

b. Proposer un algorithme d'extraction au sens de la question précédente.

Question 7. Proposer un algorithme de tri aveugle en temps $O(n \log^c n)$, pour c une constante.

Question 8 (bonus). On considère des algorithmes de compaction en un sens plus général. Un algorithme de compaction en ce nouveau sens prend en entrée une séquence d'objet X et $f : X \to \{0,1\}$. Il renvoie la liste triée suivant f (noter qu'il n'y a plus de notion de paquets).

Proposer un algorithme de compaction aveugle en temps $O(n \log n)$, en s'inspirant du dessin ci-dessous.



Corrigé

Question 1a. Être aveugle dépend de l'implémentation en mémoire. Néanmoins, pour tout choix de tri standard en $O(n \log n)$, l'algorithme n'est pas aveugle. Pire : pour les implémentations les plus « normales » du tri rapide et du tri fusion (qui sont les réponses attendues), on peut déduire intégralement l'ordre initial des éléments en observant les accès mémoire.

Question 1b. Pour trier deux éléments, on peut les charger dans la mémoire privée, trier localement, et repousser le résultat vers le disque. Pour n objets, on peut faire un tri bulle, en observant que le tri bulle revient à faire $O(n^2)$ tris sur deux éléments (qu'on sait faire de manière aveugle), et de plus les paires d'éléments entre lesquels on fait le tri sont fixées à l'avance.

Question 2a. Il suffit de l'écrire : $\Pr[gu = x] = \Pr[u = g^{-1}x] = 1/|G|$. Le point clef est que $x \mapsto gx$ est une permutation.

Question 2b. Pour réaliser un tri aveugle suivant $<$, on réalise d'abord un tri aveugle suivant un ordre uniforme $<_h$, puis on effectue un algorithme de tri quelconque non aveugle suivant l'ordre $<$, par exemple le tri fusion. Le point clef est que pour un tri standard raisonnable (comme le tri fusion), les accès mémoires dépendent uniquement de la permutation π induite par $<$ sur X, i.e. la permutation entre la liste d'origine et la liste triée. Le tri aléatoire sur $<_h$ revient à effectuer une permutation aléatoire π_h sur X. Ensuite, le tri pas aveugle a des accès mémoires qui sont déterminés entièrement par $\pi \circ \pi_h^{-1}$. Mais par Q5a, $\pi \circ \pi_h^{-1}$ est uniforme. En résumé, les accès mémoire de l'algorithme global contiennent : (1) les accès de l'algorithme aveugle ; puis (2) les accès d'un tri standard pour un ordre uniforme. On voit que dans les deux cas, ça ne dépend pas de $<$; c'est entièrement déterminé par n.

Question 3. Le plus simple est de prendre chaque paire de deux paquets consécutifs, et faire tourner Compaction sur chaque paire, en triant suivant le premier bit de h (i.e. on prend $f(x) = h(x)_1$). De cette manière, tous les éléments dont le premier bit (par h) est 0 sont dans les paquets d'indice pair (si numérote en commençant par 0), et ceux dont le premier bit est à 1 dans les paquets d'indice impair. On peut ensuite répéter le processus récursivement, séparément sur les paquets d'indice pair, et ceux d'indice impair, en compactant cette fois suivant le second bit, etc. Au bout de $\log(n/z)$ niveaux de récursion, les paquets sont triés au sens de l'énoncé.

Les candidats peuvent proposer d'autres solutions, mais toute solution raisonnable va avoir le même type de propriété. Les solutions « déraisonnables » incluent celles où par exemple on fusionne tous les paquets en un seul (techniquement possible dans la mesure où on autorise pour l'instant un paquet à contenir n éléments).

Question 4. Si on voit l'algorithme comme dans la réponse à la question précédente, au i-ième niveau de récurrence, un paquet peut recevoir les objets qui étaient à l'origine dans 2^i paquets, donc $z2^i$ objets possibles, et il contient exactement ceux parmi ces objets dont les i premiers bits par h ont une valeur fixe. La formule binomiale donne le résultat. (Il peut être utile de représenter la situation par un dessin en k couches successives, la première couche représentant les paquets d'origine, etc.)

Question 5a. La première inégalité est claire, la seconde s'ensuit en remarquant $e^k \geq k^k/k!$ via l'expression en série entière de e^k.

Question 5b. Le calcul n'est pas pénible :

$$\begin{aligned} g(z,t) &= \sum_{k=6z+1}^{zt} C_{zt}^{k} t^{-k}(1-t^{-1})^{zt-k} \\ &\leq \sum_{k=6z+1}^{\infty} \left(\frac{zte}{k}\right)^{k} t^{-k} \\ &\leq \sum_{k=6z+1}^{\infty} \left(\frac{ze}{6z}\right)^{k} \\ &\leq \sum_{k=6z+1}^{\infty} 2^{-k} \\ &\leq 2^{-6z}. \end{aligned}$$

Question 6a. Comme le dit l'indication, le nombre d'éléments réels par paquet ne révèle pas d'information. Ici, il est important de bien comprendre que le but de l'algorithme est d'effectuer un tri aléatoire. La fonction h est une variable auxiliaire de l'algorithme. Révéler des informations sur h n'a pas d'importance, tant que ça ne révèle pas d'information sur le tri effecuté, c'est-à-dire sur $<_h$.

Une manière de le voir est un argument de symétrie. Notons h' la fonction h tronquée aux $\log(n/z)$ premiers bits. Révéler le nombre d'éléments par paquet revient exactement à révéler le multi-ensemble (non ordonné) $E = \{h'(x) : x \in X\}$. De plus, soit $\Pi(X)$ le groupe des permutations sur X, alors $<_{h\circ\pi}$ pour $\pi \in \Pi(X)$ parcourt tous les ordres sur X exactement une fois, en laissant E invariant. Apprendre E ne révèle donc strictement rien sur $<_h$.

Question 6b. Avec Q6a, l'extraction est facile : on peut d'abord trier chaque paquet séparément (par exemple avec l'algorithme de Q1b). Ensuite, on place un curseur qui lit les éléments de chaque paquet l'un après l'autre, et les copie dans une nouvelle liste quand ils sont réels. Si les éléments factices ont été gérés de manière raisonnable (par exemple en les plaçant toujours après les éléments réels dans un paquet donné), la position des éléments réels à l'intérieur du paquet ne contient pas d'information.

Question 7. On applique la construction vue jusqu'ici. Il reste juste à expliquer comment on effectue Compaction de manière aveugle. Pour ça, on peut utiliser le tri de Q1b (tri bulle). Comme c'est sur $z = O(\log n)$ éléments, ça ne coûte qu'un facteur polylog. Avant d'appliquer Compaction, on peut supposer qu'on parcourt les deux paquets en annotant les éléments factices, de sorte qu'il y ait exactement z éléments associés à 0 au total, et z associés à 1, pour la compaction. Au final, n/z paquets coûtant chacun $O(z^2)$ pour tri bulle, fois $\log(n/z)$ niveaux de récursion dans Q3, donc $O(n \log^2 n)$.

Question 8. Les éléments blancs de la première ligne sont les objets x tels que $f(x) = 0$. Le numéro sur la case de l'objet x est le nombre de cases d_x dont l'objet doit être déplacé vers la gauche, de manière à ce que les objets avec $f(x) = 0$ occupent les premières cases, dans le même ordre que celui d'origine. Cette quantité peut être calculée à la volée, ou on peut parcourir d'abord la liste et annoter les objets (comme dans le dessin). Ensuite, la première « couche » de l'algorithme consiste à déplacer d'un cran vers la gauche les objets tels que $d_x \bmod 2 = 1$. Ensuite, une manière d'interpréter l'algorithme est qu'on appelle récursivement le même algorithme sur les cases d'indice pair, et celles d'indice impair (qui n'interagissent plus dans le reste de l'algorithme).

Pour voir que ça marche, il y a un certain nombre de propriétés à justifier.

— Il n'y a pas de collision au cours de l'algorithme : deux objets avec $f(x) = 0$ ne sont jamais assigné dans la même case. Avec la vision par récurrence de l'algorithme, il suffit de vérifier que c'est vrai sur la première couche, ce qui est facile.

— Au terme de l'algorithme, les objets de $f^{-1}(0)$ occupent bien les premières cases. Pour ça, il suffit de montrer que l'objet x est bien déplacé de d_x cases au total. On remarque que la i-ième couche revient à déplacer l'objet de 2^i cases ssi le i-ième bit b_i de d_x est 1 (en partant de 0 pour le plus petit), et on a bien $d_x = \sum b_i 2^i$.

— On sait effecuter chaque couche de manière aveugle. Pour ça, il suffit d'observer qu'à chaque couche, un objet ne peut être déplacé que vers deux destinations possibles. On peut donc charger l'objet en mémoire privée, lire les deux destinations dans un ordre fixe, en écrivant l'objet dans celle qui convient.

— L'algorithme s'occupe seulement des objets tels que $f(x) = 0$. Les autres objets peuvent être écrasés. Pour avoir un algorithme de compaction complet, on peut commencer par dupliquer la liste d'objets X, puis appeler l'algorithme sur la première copie, et l'algorithme « dual » qui place les objets tels que $f(x) = 1$ le plus à droite possible sur la deuxième copie. Ensuite, on peut facilement fusionner les deux listes de manière aveugle.

B4 – Circuit universel

Pour un graphe orienté G, on note $\Delta^+(G)$ le maximum des degrés sortants des sommets de G, $\Delta^-(G)$ le maximum des degrés entrants, et $\Delta^*(G) = \max(\Delta^+(G), \Delta^-(G))$. On note $\mathsf{GOA}_d(n)$ l'ensemble des graphes orientés acycliques (GOA) à n sommets tels que $\Delta^* \leq d$. (Pour rappel, un graphe acyclique est un graphe tel qu'il n'existe pas de chemin non trivial d'un sommet à lui-même.)

Ordre topologique. On dit qu'une séquence ordonnée $(v_1, \ldots, v_n)$ des sommets d'un GOA est un *ordre topologique* si pour tous v_i, v_j tels qu'il existe un chemin orienté $v_i \to v_j$, on a $i \leq j$.

Plongement. Un *plongement* d'un GOA $G = (V, E)$ dans un GOA $G' = (V', E')$ est une paire (f, g), où $f : V \to V'$ est une injection, et g envoie $(u, v) \in E$ sur un chemin orienté de $f(u)$ vers $f(v)$, de telle sorte que pour $e \neq e'$, les chemins $g(e)$ et $g(e')$ sont disjoints (pas d'arête commune).

Graphe universel. Soit $G_{\mathrm{u}} = (V_{\mathrm{u}}, E_{\mathrm{u}})$ un GOA muni d'une séquence de sommets distingués $P = (p_1, \ldots, p_n) \in V_{\mathrm{u}}^n$ deux à deux distincts, appelés « pôles ». On dit que G_{u} est *universel* pour $\mathsf{GOA}_d(n)$ si pour tout $G \in \mathsf{GOA}_d(n)$ muni d'un ordre topologique $(v_1, v_2, \ldots, v_n)$, il existe un plongement de G dans G_{u} qui envoie v_i sur p_i.

Question 1.

a. Soit $k < n$. Donner un GOA universel pour $\mathsf{GOA}_k(n)$.

b. Donner un GOA G_{u} universel pour $\mathsf{GOA}_1(4)$ tel que $\Delta^*(G_{\mathrm{u}}) = 2$.

Question 2.

a. Soit G un graphe non-orienté dont les sommets sont de degré au plus 2. On suppose que G est *biparti*, c'est-à-dire qu'il existe une partition des arêtes en V_1, V_2 telle que toutes les arêtes sont de la forme $\{v_1, v_2\}$ pour $v_i \in V_i$. Montrer qu'on peut colorier les arêtes de G en deux couleurs, de sorte que toute paire d'arêtes avec un sommet commun a des couleurs distinctes.

b. Soit $G = (V, E) \in \mathsf{GOA}_2(n)$. Montrer qu'il existe une partition des arêtes $E = E_1 \cup E_2$ telle que $(V, E_i) \in \mathsf{GOA}_1(n)$ pour $i \in \{1, 2\}$.

Question 3. Dans cette question, on suppose qu'on connaît un graphe G universel pour $\mathsf{GOA}_1(n-1)$ avec $\Delta^*(G) \leq 2$, tel que les pôles sont de degré entrant et sortant au plus 1.

a. Donner une construction d'un graphe universel G_{u} pour $\mathsf{GOA}_2(n)$ avec $\Delta^*(G_{\mathrm{u}}) \leq 2$, qui contient deux copies de G comme sous-graphes.

b. En déduire une construction d'un graphe universel G'_{u} pour $\mathsf{GOA}_1(2n)$ avec $\Delta^*(G'_{\mathrm{u}}) \leq 2$, qui contient encore deux copies de G comme sous-graphes.

Question 4. Montrer qu'il existe un graphe universel G_{u} pour $\mathsf{GOA}_2(n)$ avec $\Delta^*(G_{\mathrm{u}}) \leq 2$. Borner son nombre de sommets.

Circuit. On appelle *circuit* à n entrées et m portes un GOA $G = (V, E)$ dont les sommets sont partitionnés comme suit :

- n sommets « entrée » $(s_1, \ldots, s_n)$, qui ont degré entrant 0 et degré sortant au plus 1 ;
- un sommet « sortie » t, de degré entrant 1 et degré sortant 0 ;
- m autres sommets appelés « portes », qui sont de deux types : NAND, de degré entrant 2 et sortant 1, et COPIE, de degré entrant 1 et sortant 2.

Soit $\mathcal{C}_{n,m}$ l'ensemble des circuits avec n entrées et m portes. Étant donné $C \in \mathcal{C}_{n,m}$, et $x = (x_1, \ldots, x_n) \in \{0,1\}^n$, on associe par induction une valeur booléenne à chaque arête de C : les arêtes provenant des entrées s_i prennent la valeur x_i, les arêtes sortant des portes COPIE prennent la valeur de l'arête en entrée, et les arêtes en sortie des portes NAND prennent la valeur $\neg(a \wedge b)$, où a, b sont les valeurs des arêtes en entrée. On définit $C(x)$ comme la valeur de l'arête menant au sommet de sortie t.

Question 5. Montrer que pour tout n, il existe m tel que pour toute fonction booléenne $f : \{0,1\}^n \to \{0,1\}$, il existe un circuit $C \in \mathcal{C}_{n,m}$ satisfaisant $C(x) = f(x)$ pour tout x.

Circuit universel. On dit qu'un circuit $C_{\mathrm{u}} \in \mathcal{C}_{n_{\mathrm{u}},m_{\mathrm{u}}}$ est *universel* pour $\mathcal{C}_{n,m}$ avec $n \leq n_{\mathrm{u}}$ si pour tout $C \in \mathcal{C}_{n,m}$, il existe $c \in \{0,1\}^{n_{\mathrm{u}}-n}$ tel que pour tout x, $C(x) = C_{\mathrm{u}}(x, c)$.

Question 6. Soit $m' = m + n$. Montrer qu'il existe un circuit universel pour $\mathcal{C}_{n,m}$ avec un nombre de portes $O(m' \log m')$.

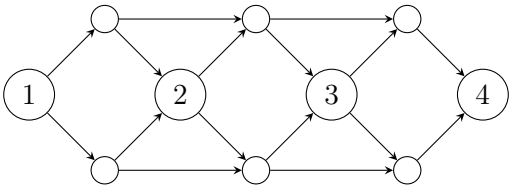
Question 7. On souhaite généraliser le résultat de la question 2b. Soit $G = (V, E) \in \mathsf{GOA}_d(n)$. Montrer qu'il existe une partition des arêtes $E = E_1 \cup \cdots \cup E_d$ telle que $(V, E_i) \in \mathsf{GOA}_1(n)$ pour tout i.

Pour cela, le théorème de Kőnig peut être utile. Soit $G = (V, E)$ un graphe non-orienté biparti. Une *couverture* de G est un sous-ensemble R de sommets tel que toute arête est adjacente à au moins un sommet de R. Un *couplage* de G est un sous-ensemble C d'arêtes tel qu'aucune paire d'arêtes de C n'a de sommet en commun. Le théorème de Kőnig dit que la taille minimale d'une couverture, et la taille maximale d'un couplage, sont égales.

Corrigé

Question 1a. On peut prendre $V = \{1, \ldots, n\}$, et $E = \{(i, j) : i < j\}$. N'importe quel GOA sur V en ordre topologique se plonge dedans en prenant l'identité comme plongement.

Question 1b. On peut prendre :

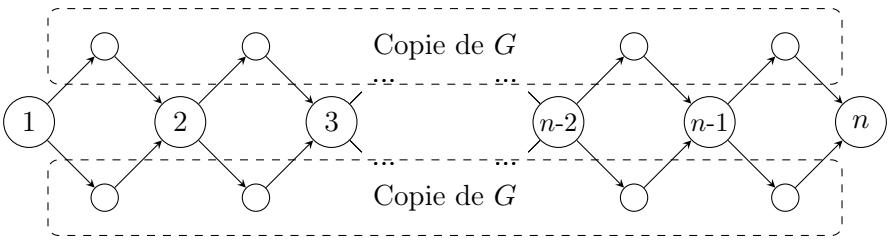


Pour vérifier, on peut faire des cas, en remarquant qu'on peut se limiter aux graphes de $\mathsf{GOA}_1(4)$ qui sont maximaux pour l'inclusion des arêtes, et il n'y en a pas beaucoup : $\{(1,2),(2,3),(3,4)\}$, $\{(1,2),(2,4)\}$, $\{(1,3),(2,4)\}$, $\{(1,3),(3,4)\}$, $\{(1,4),(2,3)\}$.

Question 2a. Comme le degré maximum est au plus 2, les composantes connexes du graphe sont des chemins ou des cycles de longueur paire. Dans les deux cas, le 2-coloriage des arêtes est facile.

Question 2b. La graphe biparti $(V \times \{0,1\}, \{\{(u,0),(v,1)\} : (u,v) \in E\})$ vérifie l'hypothèse de Q2a, donc on peut colorier les arêtes avec deux couleurs. La contrainte du 2-coloriage dit exactement que l'ensemble des arêtes d'une même couleur est dans $\mathsf{GOA}_1(n)$ (on identifie ici les arêtes du graphe biparti avec celles du graphe d'origine).

Question 3a. On utilise le schéma ci-dessous, où les $n-1$ sommets intermédiaires dans chaque « copie de G » sont pris comme pôles à l'intérieur de G.



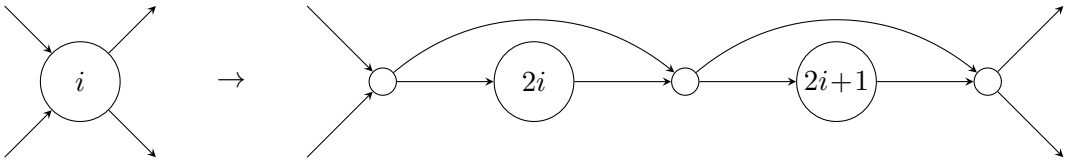
Pour voir que ça marche, on utilise la partition des arêtes en deux couleurs $E_1 \cup E_2$ de Q2b, telle que chaque couleur donne un graphe (V, E_i) dans $\mathsf{GOA}_1(n)$. L'idée est de faire passer les arêtes de E_1 dans la copie supérieure de G, et celles de E_2 dans la copie inférieure.

La manière de faire passer les arêtes de E_1 dans la première copie de G est claire à partir du schéma. Pour préciser : à partir de (V, E_1), on définit un graphe de $\mathsf{GOA}_1(n-1)$, sur les sommets $\{1, \ldots, n-1\}$, tel que chaque arête (i, j) pour $j > i+1$ de E_1 donne une arête $(i, j-1)$ dans le nouveau graphe. On plonge ce nouveau graphe dans G. En raccordant avec les arêtes sur le schéma, on déduit un plongement des arêtes de E_1 telles que $j > i+1$ dans la partie supérieure du schéma. Pour les arêtes de E_1 de la forme $(i, i+1)$, on utilise le chemin de longueur 2 qui est sur le schéma.

Il faut montrer que cette construction donne un plongement, c'est-à-dire que les chemins utilisés sont disjoints. Il ne peut pas y avoir de collision dans les arêtes internes de G. Restent les $2(n-1)$ arêtes dans la partie supérieure du schéma, mais chacune de ces arêtes ne peut apparaître que dans un chemin généré par une arête entrante ou sortante d'un sommet de V dans E_1, donc elle ne peut apparaître que dans un seul chemin.

Il faut aussi montrer que le graphe obtenu vérifie $\Delta^* \leq 2$. Pour les sommets communs de G (ceux qui ne sont pas des pôles), c'est vrai par hypothèse. Pour les pôles, par hypothèse ils sont de degré entrant et sortant 1 dans G, donc 2 dans le schéma.

Question 3b. On part de la construction de Q3a, mais on la transforme pour diviser chacun des n pôles en deux pôles, comme suit :



On montre que cette construction donne le graphe universel recherché. Soit G' un graphe de $\mathsf{GOA}_1(2n)$. On fusionne deux sommets consécutifs de ce graphe ; le graphe obtenu est dans $\mathsf{GOA}_2(n)$. Il se plonge donc dans la construction de Q3a. Pour le graphe de départ, on traite toutes les arêtes entre deux « superpôles » (groupe de deux sommets consécutifs $2i$, $2i+1$) de la même manière que le plongement dont on vient de discuter. Il reste à expliquer ce qui se passe à l'intérieur d'un superpôle. Pour chaque superpôle, dans la transformation ci-dessus, le graphe à droite (division du superpôle) reçoit au plus un chemin entrant et sortant, et il n'est pas difficile de voir qu'on peut les connecter aux pôles $2i$ et $2i+1$ de manière quelconque.

Question 4. Q3b donne une construction récursive d'un graphe universel pour $\mathsf{GOA}_1(2n)$ à partir d'un graphe universel pour $\mathsf{GOA}_1(n-1)$. Si $A(n)$ est le nombre d'arêtes du graphe universel construit de cette manière, on a $A(2n) = 2A(n-1) + O(n)$, dont on peut déduire un borne $O(n \log n)$ sur le nombre d'arêtes de la construction finale. Une manière de le faire : d'abord pour ne pas s'embêter, on peut utiliser des graphes universels pour $\mathsf{GOA}_1(n)$ et non $\mathsf{GOA}_1(n-1)$ dans la construction récursive, quitte à ignorer un sommet, et partir sur $A(2n) \leq 2A(n) + Cn$. En prenant $C' = \max(C, A(1))$ on peut vérifier par récurrence une borne $C'n(1 + \log n)$ pour les graphes de taille une puissance de 2. Pour les graphes d'une autre taille, on peut arrondir à la puissance de 2 supérieure, puisqu'un graphe universel pour $\mathsf{GOA}_1(n)$ est aussi universel pour $\mathsf{GOA}_1(k)$, $k < n$ (quitte à ignorer des pôles), et l'arrondi à la puissance supérieure induit au plus un facteur 2 sur le nombre de sommets, ce qui ne change pas la forme de la borne.

Au terme de cette construction, on obtient un graphe universel pour $\mathsf{GOA}_1(n)$ de taille $O(n \log n)$. Par Q3a, on déduit une construction d'un graphe universel pour $\mathsf{GOA}_2(n)$ avec une borne de la même forme.

Question 5. Immédiat en observant que NAND est une porte universelle (on code facilement $\neg$, puis $\wedge$, puis $\vee$). Il suffit alors d'avoir une formule logique, par exemple en CNF, pour f, utiliser la porte COPIE autant de fois que nécessaire pour recopier les entrées utilisées plusieurs fois, puis coder la formule dans un circuit.

Question 6. On prend le circuit universel de Q4 pour $m' + 1$ sommets. Les n premiers pôles vont servir à encoder les entrées, et les m suivants les portes, avec le dernier pôle correspondant à la sortie. Plus précisément, à partir du graphe universel, on supprime toutes les arêtes qui arrivent sur un pôle encodant une entrée du circuit. Les pôles entrée sont ensuite transformés en fils entrants du circuit. Les pôles qui encodent des portes sont remplacés par une porte universelle de deux bits vers un (plus un dédoublement du fil de sortie via une porte COPIE), dont le comportement est déterminé par un nombre constant de bits de contrôle, ajoutés aux fils d'entrée du circuit. Les sommets du graphe qui ne sont pas des pôles ont tous degré entrant et sortant 2. On les transforme en un circuit qui échange ou non les deux fils entre entrée et sortie, en fonction d'un nouveau bit de contrôle.

Pour un circuit quelconque de $\mathcal{C}_{n,m}$, on considère le GOA associé. On ordonne les sommets en ordre topologique, de manière à ce que les fils d'entrée arrivent en premier, et la porte de sortie arrive en dernier. On plonge ce graphe dans le graphe universel qui sous-tend le circuit universel précédent. Ensuite, on utilise les bits de contrôle des sommets communs (non pôles) pour reproduire le plongement,

et les bits de contrôles des pôles correspondant aux portes pour reproduire le comportement de la porte voulue.

Question 7. On construit le graphe non orienté biparti $(V \times \{0,1\}, \{\{(u,0),(v,1)\} : (u,v) \in E\})$ comme dans Q2b, de degré au plus d. Quitte à ajouter des arêtes, on peut supposer sans perte de généralité que les sommets du graphe ont tous degré exactement d (en observant que s'il y a un sommet de degré $< d$ d'un côté du graphe biparti, il y a en nécessairement un de l'autre côté). En effet, l'ajout d'arête ne peut que rendre le problème demandé plus difficile.

La couverture minimale d'un tel graphe est de cardinalité n, en effet il y a dn arêtes et chaque sommet en couvre au plus d, donc on ne peut pas faire moins. Par le théorème de Kőnig, il s'ensuit qu'il existe un couplage de cardinalité n. On met les arêtes de ce couplage dans E_1, puis on retire les arêtes du graphe biparti. On obtient un graphe biparti régulier de degré $d-1$, et on recommence par récurrence pour construire E_2, etc. On vérifie facilement que chaque (V, E_i) est dans $\mathsf{GOA}_1(n)$.

B5 – Graphes triangulés

Pour S un ensemble, on note $\mathcal{P}_2(S)$ l'ensemble des sous-ensembles de cardinalité 2 de S. Un *graphe* G est une paire (V, E) avec $E \subseteq \mathcal{P}_2(V)$. Étant donné $S \subseteq V$, on note $G[S] = (S, E \cap \mathcal{P}_2(S))$ le *sous-graphe de G induit par S*. Un graphe est *cyclique* de taille $n \geq 3$ s'il est isomorphe à $(\mathbb{Z}/n\mathbb{Z}, \{\{i, i+1\} : i \in \mathbb{Z}/n\mathbb{Z}\})$. Un graphe est *triangulé* si tous ses sous-graphes induits cycliques (s'il en existe) sont de taille 3.

Question 0. Donner un example de graphe triangulé, et de graphe non triangulé.

Question 1. Montrer qu'un graphe est triangulé si et seulement si pour toute suite de sommets formant un cycle de longueur au moins 4, il existe une *corde*, c'est-à-dire une arête qui lie deux sommets non consécutifs du cycle.

Question 2. Étant donné deux sommets a, b non adjacents dans un graphe $G = (V, E)$, un *séparateur* de a et b est un ensemble de sommets $S \subseteq V \setminus \{a, b\}$ tel que lorsqu'on enlève les sommets S de G, a et b se retrouvent dans des composantes connexes distinctes (formellement, « enlever » les sommets S signifie qu'on se place dans le graphe $G[V \setminus S]$). Un séparateur est dit *minimal* s'il est minimal pour l'ordre d'inclusion.

a. Soit G un graphe tel que tout séparateur minimal induit un graphe complet, c'est-à-dire : $G[S] = (S, \mathcal{P}_2(S))$. Montrer que G est triangulé.

b. Montrer la réciproque : si G est triangulé, alors tout séparateur minimal induit un graphe complet.

Question 3. Soit v un sommet d'un graphe $G = (V, E)$. On note $\mathsf{adj}_G(v)$ l'ensemble des sommets adjacents à v dans G. On dit que v est *simpliciel* si $\mathsf{adj}_G(v)$ induit un graphe complet.

Montrer que tout graphe triangulé qui n'est pas un graphe complet contient au moins deux sommets simpliciels non adjacents.

Indication : considérer un séparateur S pour deux sommets a, b non adjacents, et chercher un sommet simpliciel dans A (resp. B), la composante connexe de a (resp. b) dans $G[V \setminus S]$.

Question 4. Soit $G = (V, E)$ un graphe à n sommets. Soit $\sigma = (v_1, \ldots, v_n)$ un ordre sur les sommets. On dit que σ est un *schéma d'élimination* si pour tout i, v_i est simpliciel dans $G[\{v_i, \ldots, v_n\}]$.

a. Montrer qu'un graphe est triangulé si et seulement si il admet un schéma d'élimination.

b. Proposer un algorithme en temps polynomial pour tester si un graphe est triangulé.

Question 5. Le *nombre chromatique* $\chi(G)$ d'un graphe $G = (V, E)$ est le plus petit entier k tel que G admet un k-coloriage, c'est-à-dire un coloriage des sommets en k couleurs tel que toute paire de sommets adjacents a des couleurs distinctes. Le *nombre de clique* $\omega(G)$ de G est le nombre de sommets de la plus grande clique de R, c'est-à-dire le plus grand sous-graphe induit complet.

a. Montrer $\chi(G) \geq \omega(G)$.

b. Soit G un graphe triangulé, et S un séparateur. Soient $A_1, \ldots A_k$ les composantes connexes de $G[V \setminus S]$. Montrer :

$$\chi(G) = \max_i \chi(G[S \cup A_i]).$$

c. Montrer que pour un graphe triangulé, $\chi(G) = \omega(G)$.

Question 6. Soit $C = (V, E)$ un graphe cyclique de taille n. Une *triangulation* de C est un graphe triangulé (V, E') avec $E' \supseteq E$.

a. Montrer qu'une triangulation de C est minimale, au sens de l'ordre induit par l'inclusion des arêtes, si et seulement si elle est minimale pour l'ordre induit par le nombre d'arêtes.

b. Donner une formule explicite pour le nombre de triangulations minimales de C.

Corrigé

Question 1. Immédiat.

Question 2a. Soit un cycle $v_1, \ldots, v_k$ de longeur au moins 4. Soit S un séparateur de v_1, v_3. Alors S doit contenir v_2 et v_i pour un certain $i > 3$. Comme S induit une clique, il existe une corde entre v_2 et v_i.

Question 2b. Soit S un séparateur minimal de a, b. Soit A (resp. B) les composantes connexes de a (resp. b) dans $G[V \setminus S]$. Soit $x, y \in S$. Par minimalité de S, il existe un chemin de x vers a passant uniquement par des sommets de A et de x vers b passant uniquement par des sommets de B, de même pour y. Il existe donc un cycle $x, a_1, \ldots, a_k, y, b_1, \ldots, b_\ell$, avec $a_i \in A$, $b_j \in B$, qu'on choisit de longueur minimale : il n'existe donc aucune arête entre un a_i et un $a_{i'}$ ou entre un b_j et un $b_{j'}$ (sinon le cycle ne serait pas minimal). En fait, la seule arête possible dans ce cycle est entre x et y et une telle arête existe car le graphe est triangulé.

Question 3. En suivant l'indication, supposons d'abord que $G[A \cup S]$ n'est pas un graphe complet, alors par récurrence il contient deux sommets simpliciels non adjacents. Comme S induit un graphe complet, au moins l'un des sommets est dans A. Si A est complet, on conclut également que A contient un sommet simpliciel v. Ce sommet reste simpliciel dans G parce que $\mathsf{adj}_G(v) \subseteq A \cup S$. Par symétrie, il y a également un sommet simpliciel dans B.

Question 4a. Un sous-graphe d'un graphe triangulé reste triangulé, donc par récurrence, Q3 implique qu'un graphe triangulé admet un ordre d'élimination. Réciproquement, soit $v_1, \ldots, v_k$ un cycle de longueur au moins 4, et soit v_i le sommet du cycle qui apparaît le premier dans un ordre d'élimination. Lorsque v_i est supprimé, il est simpliciel, ce qui implique qu'il existe une corde entre ses deux voisins (qui n'ont pas encore été supprimés).

Question 4b. On remarque qu'on peut choisir arbitrairement le premier sommet simpliciel à éliminer pour construire un ordre d'élimination. Pour tester si un graphe est triangulé, on cherche un premier sommet simpliciel, en parcourant les sommets, puis on l'enlève et on recommence. Le graphe est triangulé ssi ce processus élimine tous les sommets du graphe. Il est clair que la complexité de chaque étape est polynomiale.

Question 5a. Immédiat.

Question 5b. L'inégalité $\geq$ est claire, donc on se concentre sur $\leq$. Puisque S est complet, $G[S]$ admet un unique coloriage avec $|S|$ couleurs, à permutation des couleurs près. En fixant un tel coloriage, on peut l'étendre à un coloriage de $A_i \cup S$ avec $\chi(A_i \cup S)$ couleurs. On obtient un coloriage de G avec $\max_i \chi(G[S \cup A_i])$ couleurs.

Question 5c. Il suffit de montrer que ω vérifie une égalité du même type que χ dans Q5b : $\omega(G) = \max_i \omega(G[S \cup A_i])$. On a ensuite le résultat par induction. Pour montrer l'égalité, on observe que comme il n'y a pas d'arête entre un sommet de A_i et A_j pour $i \neq j$, toute clique maximale de G est incluse dans $A_i \cup S$ pour un certain i.

Question 6a. On montre par récurrence qu'un triangulation minimale possède $n-3$ cordes (on appelle « cordes » les éléments de $E' \setminus E$). Une manière de faire est de prendre une corde quelconque. Elle divise C en deux cycles de longueur i et $n+2-i$, et E' induit une triangulation minimale sur chaque cycle. Réciproquement, l'union de triangulations sur chacun des deux cycles donne une triangulation de C, donc par minimalité il n'y a pas d'arête hors de l'union. Il y a donc $(i-3)+(n+2-i-3)+1 = n-3$ cordes.

L'induction précédente montre aussi qu'une triangulation minimale d'un graphe cyclique est planaire, donc Q6 se dessine bien.

Question 6b. On peut tenter une récurrence en prenant une corde quelconque qui divise le cycle en deux comme en Q6a, mais il est difficile d'obtenir une formule explicite de cette manière. Un meilleure manière est d'observer qu'en prenant une arête de C et en la comprimant en un point, on envoie la triangulation E' vers une triangulation d'un cycle de longueur $n-1$. Notons $C_n = (\mathbb{Z}_n, \{\{i, i+1\} : i \in \mathbb{Z}_n\})$. La transformation précédente n'est pas une bijection entre les triangulations de C_n et celles de C_{n-1}, mais elle peut le devenir en rajoutant des « annotations ». Dans C_n, on distingue une arête du cycle E autre que $\{n-1, 0\}$ (« première configuration »). Dans C_{n-1}, on distingue une arête quelconque de E', et on l'oriente (« deuxième configuration »). La transformation de la première configuration vers la seconde consiste à « refermer » l'arête distinguée (fusionnant ses deux sommets en un seul), et distinguer à la place l'arête qui amène sur le sommet qu'on vient de fusionner, en orientant vers ce sommet. Dans l'autre sens, on « dédouble » le sommet vers lequel pointe l'arête distinguée pour former un triangle, et distingue le nouveau côté de ce triangle. Dans cette seconde transformation, on choisit de renuméroter les sommets de sorte que si l'arête distinguée pointait vers i (dans $\mathbb{Z}_{n-1}$), le nouveau côté est entre i et $i+1$ (dans $\mathbb{Z}_n$). On remarque qu'on ne peut jamais créer de cette manière une nouvelle arête distinguée $\{n-1, 0\}$, qui correspond bien au fait qu'on a interdit de distinguer cette arête dans la première configuration. On obtient une bijection entre les deux configurations.

Si on note t_n le nombre de triangulations de C_n, la bijection donne : $(n-1)t_n = 2(2n-5)t_{n-1}$. On pose $c_{n-2} = t_n$ (pour avoir la même indexation que les nombres de Catalan c_n), de sorte que la relation de récurrence devient $(n+2)c_{n+1} = 2(2n+1)c_n$, d'où l'on déduit par des calculs élémentaires :

$$c_n = \frac{(2n)!}{(n+1)!n!} = \frac{1}{n+1}\binom{2n}{n}.$$

C1 – Automates et monoïdes

Un *semigroupe* est un couple $(E, \cdot)$ où E est un ensemble et $\cdot$ une *loi de multiplication interne associative*. On rappelle :

— Une (loi de) multiplication (interne) est dite *associative* si $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.

— Un élément neutre est un élément e tel que pour tout $x \in E$, on a $x \cdot e = e \cdot x = x$.

Un *monoïde* est un semigroupe qui dispose d'un élément neutre. Dans la suite, on identifiera le semigroupe (monoïde) et son ensemble. Un semigroupe (monoïde) est dit fini s'il dispose d'un ensemble fini d'éléments.

Question 0. Préliminaires

a Montrez qu'un monoïde possède un unique élément neutre.

b Pour A un alphabet fini et $\cdot$ la concaténation, montrez que $(A^*, \cdot)$ est un monoïde.

Question 1. Monoïdes des transformations Soit Q un ensemble fini. Montrez que l'ensemble des fonctions de $Q \to Q$ est un monoïde. On appelle ce monoïde le *monoïde des transformations* de Q.

Un monoïde N est un *sous-monoïde* d'un monoïde M s'il existe un morphisme injectif de N vers M. Montrez que tout monoïde fini $M = (E, \cdot)$ est un sous-monoïde d'un monoïde des transformations.

Question 2. Monoïdes des transitions Soit $\mathcal{A} = (Q, A, \delta, i, F)$ un automate déterministe fini avec Q les états, A l'alphabet, $\delta \colon Q \times A \to Q$ les transitions, $i \in Q$ l'état initial et $F \subseteq Q$ les états finaux. Dans la suite on note $q \cdot a = \delta(q, a)$ et pour $u_1 \cdots u_k \in A^*$ on note $q \cdot u := (\cdots (q \cdot u_1) \cdot u_2 \cdots) \cdot u_k$.

On appelle *fonctions de transitions* de l'automate $\mathcal{A}$, les fonctions $f \colon Q \to Q$ tel qu'il existe un mot $u \in A^*$ avec $f(q) = q \cdot u$ pour tout $q \in Q$.

a. Montrez que l'ensemble des fonctions de transition de $\mathcal{A}$ forme un monoïde fini. On l'appelle le *monoïde des transitions* de $\mathcal{A}$.

b. Quelle est la complexité de vérifier qu'une fonction $f \colon Q \to Q$ est dans le monoïde des transitions de $\mathcal{A}$.

c. En déduire un algorithme qui énumère les fonctions de transition de l'automate $\mathcal{A}$. Donnez sa complexité en fonction de $|Q|$, le nombre d'états de $\mathcal{A}$.

d. Proposez un algorithme polynomial en $|Q|$ et N où N est le nombre de fonctions de transition de $\mathcal{A}$.

e. Un langage $L \subseteq A^*$ est *reconnu* par un monoïde M s'il existe un morphisme $\varphi \colon A^* \to M$ tel que $L = \varphi^{-1}(\varphi(L))$. Montrez que le monoïde des transitions reconnait le langage calculé par l'automate $\mathcal{A}$.

Question 3. Reconnaissabilité et régularité Un langage est *reconnaissable* s'il est reconnu par un monoïde fini. Montrez qu'un langage est reconnaissable si et seulement s'il est calculable par un automate fini.

Question 4. Monoïdes des relations Soit Q un ensemble fini. On rappelle qu'une *relation* sur Q est un sous-ensemble de $Q \times Q$. Pour R_1 et R_2 deux relations sur Q, on note $R_1 \circ R_2$ leur composition, c'est-à-dire l'ensemble $\{(x, y) \in Q \times Q \mid \exists z, (x, z) \in R_1 \text{ et } (z, y) \in R_2\}$.

a. Montrez que l'ensemble des relations sur un ensemble fini, muni de la composition, est un monoïde.

b. Donnez la taille du monoïde en fonction de la taille de l'ensemble Q.

Question 5. Automates non déterministe Soit $\mathcal{A} = (Q, A, \delta, I, F)$ un automate non déterministe fini avec Q les états, A l'alphabet, $\delta \colon Q \times A \times Q$ les transitions, $I \in Q$ les états initiaux et $F \subseteq Q$ les états finaux. Dans la suite on note $q \cdot a = \delta(q, a) \subseteq Q$ et pour $u_1 \cdots u_k \in A^*$ on note $q \cdot u := (\cdots (q \cdot u_1) \cdot u_2 \cdots) \cdot u_k$.

On appelle *relations de transition* de l'automate $\mathcal{A}$, les relations R de Q tel qu'il existe un mot $u \in A^*$ avec $(q, q \cdot u) \in R$ pour tout $q \in Q$.

a. Montrez que l'ensemble des relations d'un automate forme un monoïde fini. On l'appelle le *monoïde des relations* de l'automate.

b. Montrez que le monoïde des relations reconnait le langage calculé par l'automate $\mathcal{A}$.

c. Donnez rapidement un algorithme pour calculer le monoïde des relations d'un automate et indiquez sa complexité.

Question 6. Monoïde syntaxique et borne inférieur de complexité Le *monoïde syntaxique* est le plus petit monoïde qui reconnaît le langage régulier L. Soit L un langage régulier dont l'automate et dont le monoïde syntaxique est de taille n. Montrer que tout automate non déterministe calculant L est de taille au moins $\sqrt{\log n}$.

Corrigé

Question 0.

a *Montrez qu'un monoïde possède un unique élément neutre* : question de cours sur la théorie des groupes dans un contexte un peu différent. Si e et f sont neutres alors $ef = f = e$.

b *Pour A un alphabet fini et $\cdot$ la concaténation, montrez que $(A^*, \cdot)$ est un monoïde :* L'élément neutre est le mot vide et la concaténation est associative. Il y a pleins de preuve de l'associativité de la concaténation sur les mots. L'étudiant doit être capable d'en formaliser une.

Question 1. *Montrez que tout monoïde fini $M = (E, \cdot)$ est un sous-monoïde d'un monoïde des transformations.* L'étudiant doit penser à prendre le monoïde de transformation de E et d'associer à chaque élément $m \in M$ la fonction $f_m \colon x \in M \mapsto m \cdot x$ et montrer que c'est un morphisme injectif.

Question 2.

a *montrez que l'ensemble des fonctions de transitions de $\mathcal{A}$ forme un monoïde fini. on l'appelle le* monoïdes des transitions *de $\mathcal{A}$.* Il suffit d'illustrer que le mot vide est un élément neutre (encore) et que la composition de chemins dans un automate est associatives.

b *donnez un algorithme pour calculer le monoïde des transitions d'un automate en entrée et indiquez sa complexité.* Question plus difficile. Dans l'absolu il faut calculer par saturation l'ensemble des fonctions de transitions en partant des générateurs. Il y a plusieurs points à préciser : la représentation interne d'une *fonctions de transitions* et la complexité d'une compositions. Il faut aussi identifier une borne supérieur. Pour ça, on peut utiliser la question d'avant qui donne une borne supérieur en n^n.

c *un langage $L \subseteq A^*$ est* reconnu *par un monoïde m s'il existe un morphisme $\varphi \colon A^* \to M$ tel que $L = \varphi^{-1}(\varphi(L))$. montrez que le monoïde des transitions reconnait le langage calculé par l'automate $\mathcal{A}$.* Question facile : il faut identifier qu'un mot est accepté ssi il va de l'état initial a un état final dans $\mathcal{A}$ et donc si ça transition respecte cette contrainte.

Question 3. *Montrez qu'un langage est reconnaissable si et seulement s'il est calculable par un automate fini.*

Le sens retour est une application de la question 2. Le sens direct nécessite de produire un automate à partir du morphisme $\varphi \colon A^* \to M$. Il suffit de prendre le graphe de Cayley à droite : $Q = M$ avec $\delta(m, a) := m \cdot \varphi(a)$ et l'élément neutre comme état initial.

Question 4.

a. *Montrez que l'ensemble des relations sur un ensemble fini, muni de la composition, est un monoïde.* Pas de difficulté particulière sur cette question.

b. *Donnez la taille du monoïde en fonction de la taille de l'ensemble Q.* Il faut penser ici à l'isomorphimse entre les relations sur un ensemble fini et les matrices booléenne de taille $|Q|$. Ce qui donne une borne sup (atteinte) en 2^{n^2}.

Question 5.

a. *Montrez que l'ensemble des relations d'un automate forme un monoïde fini. On l'appelle le* monoïde des relations *de l'automate.* La question est similaire à la question 3 sans difficulté particulière. On peut laisser l'étudiant aller vite dessus.

b. *Montrez que le monoïde des relations reconnait le langage calculé par l'automate $\mathcal{A}$.* Il faut avoir bien compris la sémantique des automates non déterministe et des relations. Donc montrer que le calcul par relation propage l'invariant : *il existe un chemin entre l'état initial et l'état courant*

c. *Donnez un algorithme pour calculer le monoïde des relations d'un automate et indiquez sa complexité.* Cette question nécessite de trouver une bonne représentation pour les relations. Avec les matrices booléennes c'est le même algorithme par saturation que l'algorithme de la question 3. La complexité provient de la borne supérieur de la question précédente.

Question 6. Monoïde syntaxique et borne inférieur de complexité *Soit L un langage régulier dont l'automate est de taille n et dont le monoïde syntaxique est de taille au moins $f(n)$. Montrer que tout automate non déterministe calculant L est de taille au moins* $\log(\sqrt{f(n)})$. Cette question est en faite une conclusion simple qui donne une jolie borne inférieur sur le NFA. Si le NFA est de taille N, alors son monoïde de relation est de taille au plus $2^{N^2} \geq f(n)$.

C2 – Automates et ordres partiels

Soit $\mathcal{A} = (Q, A, \delta, I, F)$ un automate fini (non nécessairement déterministe). Pour $q \in Q$ et $u \in A^*$ on note $q \cdot u$ l'ensemble des états accessibles depuis q en lisant u. Si $\mathcal{A}$ est déterministe, alors on note également $q \cdot u$ (l'unique) état accessible depuis q en lisant u.

L'automate $\mathcal{A}$ est *partiellement ordonné* si tous ses cycles sont de taille au plus un. Formellement, si pour tout $q, q' \in Q$ et $u, v \in A^*$, tel que $q' \in q \cdot u$ et $q \in q' \cdot v$, alors $q = q'$.

Question 0.

a. Justifiez le nom *partiellement ordonné* pour ces automates.

b. Montrez que le langage $\{abaa, baaa, abb\}$ est calculable par un automate déterministe partiellement ordonné.

c. Proposez un exemple de langage non fini calculable par un automate déterministe partiellement ordonné.

d. Montrez que le langage sur l'alphabet $A = \{a, b, c\}$, $K = A^*ac^*aA^*$ est calculable par un automate non déterministe partiellement ordonné.

Question 1. Clôture par les opérations booléennes. Montrez l'intersection et l'union de deux langages calculables par des automates partiellement ordonnés sont également calculables par des automates partiellement ordonnés.

Est-ce que les opérations d'union et d'intersection sont toujours vérifiées quand on considère des automates partiellement ordonnés déterministes ?

Question 2. Les langages finis. Un automate $\mathcal{A} = (Q, A, \delta, I, F)$ est *strictement partiellement ordonné* s'il est partiellement ordonné et si pour tout etat q et lettre $a \in A$, on a $q \notin \delta(q, a)$.

a. Montrer qu'un langage est calculable par un automate strictement partiellement ordonné si et seulement s'il est fini.

b. Illustrer par un exemple que la taille de l'automate d'un langage fini peut être beaucoup plus petit que la taille du langage.

Question 3. Langages clos par sur-mots. Soit $u = a_1 \cdots a_n \in A^*$.
On note $\uparrow u$ le langage $A^*a_1A^*a_2A^* \cdots a_nA^*$. Un langage $L \subseteq A^*$ est dit *clos par sur-mots* si pour tout mot $u \in L$ on a $\uparrow u \subseteq L$. On souhaite montrer qu'un langage clos par sur-mots est régulier et calculable par un automate déterministe partiellement ordonné. On admettra dans la suite le résultat suivant.

Lemme (Higman).
Soit $(w_i)_{i\in\mathbb{N}}$ une suite infinie de mots sur l'alphabet A. Il existe deux entiers i et j tels que w_i est un sur-mot de w_j.

a. Montrez que l'intersection de deux langages clos par sur-mots est clos par sur-mots.

b. Soit $F \subset A^*$ un ensemble fini de mots. Montrer que $\bigcup_{u\in F} \uparrow u$ est calculable par un automate déterministe partiellement ordonné.

c. Montrez qu'un langage L est clos par sur-mots si et seulement s'il existe un ensemble fini F tel que $L = \bigcup_{u\in F} \uparrow u$ et conclure.

Question 4. Monômes et automates non-déterministe partiellement ordonnés. On appelle *monôme* un langage régulier de la forme $B_0^* a_0 B_1^* a_1 \cdots B_n^*$ avec $B_i \subseteq A$ et $a_i \in A$ pour tout i. On utilise la convention $\varnothing^* = \{\epsilon\}$ avec ϵ le mot vide.

a. Montrez qu'un langage régulier est calculable par un automate non-déterministe partiellement ordonné si et seulement s'il est égal à une union finie de monômes.

b. Montrez que l'intersection de deux monômes est une union finie de monômes.

c. Montrez que le langage $T = (ab)^*$ n'est pas calculable par un automate non-déterministe partiellement ordonné.

d. Conclure qu'il existe un langage L tel que L est calculable par un automate non-déterministe partiellement ordonné mais pas L^c.

Corrigé

Question 0. Des dessins d'automates suffisent ici.

Question 1. Clôture par les opérations booléennes. Il suffit de faire un produit cartésien d'automate. Le déterminisme n'entraine pas de difficulté particulière.

Question 2. Les langages finis. Un automate $\mathcal{A} = (Q, A, \delta, I, F)$ est *strictement partiellement ordonné* s'il est partiellement ordonné et si pour tout etat q et lettre $a \in A$, on a $q \notin \delta(q, a)$.

a. *Montrer qu'un langage est calculable par un automate strictement partiellement ordonné si et seulement s'il est fini.* On définit la profondeur de l'automate par induction depuis les états puits. Un état puit est de profondeur 0. Pour un état q, sa profondeur est le max de la profondeur des états tel que $q' \in q \cdot a$. On prouve que si on choisit un état q de profondeur d comme unique état initial dans l'automate, alors l'automate ne calcul que des mots de taille au plus d. Dans le sens retour, on considère l'automate $Q = \{u\}^n \cup \{\bot\}$ avec $u \cdot a = ua$ comme transition si $|u| < n$.

b. *Illustrer par un exemple que la taille de l'automate d'un langage fini peut être beaucoup plus petit que la taille du langage.* Par exemple $(a + b)^n$ est de taille 2^n mais l'automate de taille n.

Question 3. Langages clos par sur-mots.

a. *Montrez que l'intersection de deux langages clos par sur-mots est clos par sur-mots.* Si $u \in L \cap K$ et $v \in \uparrow(u)$ alors $v \in L$ et $v \in K$ donc $v \in L \cap K$.

b. *Montrez que pour tout mot $u \in A^+$, $\uparrow u$ est calculable par un automate déterministe partiellement ordonné.* Montrer que c'est par un automate non déterministe partiellement ordonné est trivial mais c'est pas la question ! Deux stratégie : une preuve par induction sur n en prenant comme hypothèse d'induction que l'automate a une unique état puit acceptant. Sinon, on reprend la construction powerset et on remarque que le powerset de l'automate naif est partiellement ordonné car on atteint que des segments $0, \cdots, i$.

c. *Montrez en appliquant le lemme d'Higman qu'un langage L est clos par sur-mots si et seulement s'il existe un ensemble fini F tel que $L = \bigcup_{u \in F} \uparrow u$.* On raisonne par l'absurde, on en déduit une famille de mots infinie où aucun mot est un sous-mot d'un autre et on conclut par le lemme d'Higman.

d. *Conclure.* On applique les questions précédentes et la question 1 pour la cloture par union.

Question 4. Monômes et automates non-déterministe partiellement ordonnés.

a. *Montrez qu'un langage réguliers est calculable par un automate non-déterministe partiellement ordonnés si et seulement s'il est égal à une union de monôme* Pour le sens direct : on montre dans un premier temps le résultat pour les automate filiforme (i.e. qui sont sans branchements). Puis on peut montrer qu'un automate non-déterministe partiellement ordonnés est une union d'automate filiforme et conclure.

b. *Montrez que l'intersection de deux monômes est une union de monômes.* Conséquence immédiate de la question 1 et de la question précédente.

c. *Montrez que le langage $T = (ab)^*$ n'est pas calculable par un automate non-déterministe partiellement ordonné.* Par l'absurde. T est égale à une union de monôme. On montre que tout monome qui calcul T accepte des mots de T^c. On pose un monome $K = B_0^* a_0 B_1^* \cdots B_n^*$ et $u = (ab)^p$ avec $p \geq 2n + 1$.

Si $u \in K$, alors il existe $v_0, \ldots v_n$ tel que $u = v_0 a_0 v_1 \cdots v_n$. Comme $p \geq 2n + 1$, nécessairement, il existe i tel que $|v_i| \geq 2$ et donc $\{a, b\} \subseteq B_i$. On a donc également que le mot $u' = v_0 a_0 \cdots a_{i-1} a a a_{i+1} \cdots v_n$ appartient dans K. Mais $u' \notin (ab)^*$.

d. *Conclure qu'il existe un langage L qui est une union de monômes mais pas son complément.* Montre que T^c est calculable par une union de monôme.

C3 – Langages réguliers de Delphes

Soit A et B deux alphabets et $u = u_1 \cdots u_n \in A^n$ et $v = v_1 \cdots v_n \in B^n$. On note $u * v$ le mot $(u_1, v_1) \cdots (u_n, v_n)$ de $(A \times B)^n$. Soit $\mathcal{A} = (Q, A \times B, \delta, I, F)$ un automate fini (non nécessairement déterministe) on note $L(\mathcal{A})$ le langage reconnu par $\mathcal{A}$.

Un *oracle* pour $\mathcal{A}$ est un ensemble $O \subseteq B^*$ tel $|O \cap B^n| = 1$ pour tout entier $n \in \mathbb{N}$. On note $\mathcal{A}(O)$ le langage des mots reconnus par $\mathcal{A}$ avec l'oracle O, c'est-à-dire,

$$\mathcal{A}(O) = \{u \in A^* | \exists v \in O, u * v \in L(\mathcal{A})\}$$

On appelle *langages réguliers de Delphes* les langages reconnus par des automates avec oracle.

Question 0. Montrez que le langage $\{a^n b^n \mid n \in \mathbb{N}\}$ est un langage régulier de Delphes.

Question 1. Montrez que l'intersection et l'union de deux langages réguliers de Delphes est un langage régulier de Delphes.

Question 2.

a. Montrez que la classe des langages réguliers est dénombrable.

b. Montrez que la classe des langages réguliers de Delphes n'est pas dénombrable.

Question 3. Montrez que le langage sur $\{a, b\}$ des mots avec autant de a que de b n'est pas un langage régulier de Delphes.

Question 4. Soit L un langage régulier de Delphes avec $L = \mathcal{A}(O)$. Soit $v \in B^*$, On note $E_v = \{u \mid u * v \in L(\mathcal{A})\}$.

Montrez que si L est un langage régulier alors le langage $K = \{v \mid E_v \subseteq L\}$ est un langage régulier.

Question 5. Dans la suite, on note $<_{\text{lex}}$ l'ordre lexicographique sur les mots pour un ordre abitraire sur l'alphabet A.

Soit L un langage régulier sur l'alphabet A, montrez que le langage $L_{\text{lex}} = \{u \mid u \in L \wedge \exists v \in L \cap A^{|u|}, v <_{\text{lex}} u\}$ est également un langage régulier.

Question 6. Soit L un langage régulier reconnu par un automate avec oracle $\mathcal{A}(O)$. Montrez qu'il existe un oracle O' régulier tel que $\mathcal{A}(O) = \mathcal{A}(O') = L$.

Question 7. Un oracle O est dit *uniforme* s'il existe un mot infini $w \in B^\omega$ tel que O est la suite des préfixes de w. Dans ce cas, on note $\mathcal{A}(w)$ pour $\mathcal{A}(O)$.

a. Montrez que $\{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas reconnaissable par un automate avec oracle uniforme.

b. Pour K un langage, on note $\textsc{Prefix}(K)$ l'ensemble de ces préfixes. C'est-à-dire, l'ensemble $\textsc{Prefix} = \{u | \exists v \in A^*, uv \in K\}$. Un langage K est clos par préfixe si $K = \textsc{Prefix}(K)$. Montrez que si K est langage régulier non fini clos par préfixe alors il existe u et v tel que $\textsc{Prefix}(uv^*) \subseteq K$.

c. Soit L un langage régulier reconnu par un automate avec oracle uniforme $\mathcal{A}(w)$. Montrez qu'il existe $u, v \in B^*$ tel que tel que $\mathcal{A}(O) = \mathcal{A}(uv^\omega)$.

Corrigé

Question 0. Il suffit de prendre pour l'oracle $0^{\frac{n}{2}}10^{\frac{n}{2}}$, qui marque la moitié du mot.

Question 1. Une construction d'automate produit sans difficulté particulière.

Question 2. Les automates sont dénombrables, on peut utiliser n'importe des suites finis d'entiers dans $\mathbb{N}$ pour coder ensemble, alphabet, transitions, ect et conclure.

Les automates de Delphes ne sont pas dénombrable : on peut prendre un langage qui accepte le développement décimale d'un réel en utilisant l'oracle pour ça.

Question 3. Pleins d'arguments sont possibles. Voici une proposition. On regarde les mots de la forme $u_i = a^i b^{n-i}$ sur un oracle de taille $2n$. Nécessairement il existe i, j tel que $q.u_i = q.u_j$ mais il existe v tel que $u_i v \in L$ et $u_j v \notin L$ ce qui donne une contradiction.

Question 4. On construit un automate pour K en prenant le produit cartésien pour $\mathcal{A}$ avec l'automate pour L en effaçant les transition de l'alphabet A.

Formellement $(q, q').b \to (p, p')$ s'il existe a tel que $q.(a, b) = p$ et $q'.a = p'$.

Ça donne un automate non déterministe mais il marche.

Question 5. On réalise une copie de l'automate qu'on encode par un produit cartésien et une autre copie qui stock la valeur d'un mot virtuelle. On va deviner non déterministiquement le mot v associer au mot u.

Dans la première copie, v et u on la même valeur, dans la seconde copie ils peuvent diverger arbitrairement.

pour chaque $(q, 0)$ on a les transitions $(q, 0).a = (q.a, 0)$ et $(q, 0).a = (q.a, 1, q.b)$ pour $b < a$. Et $(q, 1, q').a = (q.a, 1, q'.b)$ pour toute lettre b.

Question 6. On utilise encore une fois K mais il faut extraire de K un oracle régulier, i.e. un langage régulier tel que $A^n \cap T$ est de taille 1 pour tout n. Pour ça on peut prendre $K \setminus K^c_{\text{lex}}$ qui a la bonne forme puisque ça sélectionne le plus petit mot de K pour chaque taille par ordre lexicographique.

Question 7.

a. Encore un argument de pompage. On peut utiliser le fait qu'un configuration de l'algorithme c'est une paire (q, i) où i est la position sur l'oracle. Si on prends n suffisamment grand et on considère $c_i = (q_{\text{init}}.a^{2n+i}b^{n-i}n, 3n)$ pour $i \in \{0, \dots, |Q|\}$. Il y a nécessairement deux configurations égale, on peut conclure.

b. Pas de difficulté ici. Pour tout langage régulier non fini, on peut trouver u, v, w avec v non vide tel que $uv^*w \subseteq K$. Le reste suit.

c. Il faut prendre un langage un peu différent de la question 5 : $\{v \mid E_{v'} \subseteq L \text{ pour } v' \in \textsc{Prefix}(v)\}$. Montrer que c'est un langage régulier non vide, non fini, et conclure avec la question d'avant.

C4 – Circuits booléens linéaires

Soit $\mathcal{X}_n = \{x_1, \ldots, x_n\}$ un ensemble énuméré de variables.

Un circuit booléen sur $\mathcal{X}_n$ est un graphe orienté acyclique (V, E), où les sommets de degré entrant 0, les entrées du circuit, sont étiquetés par des variables de $\mathcal{X}_n$ et les autres noeuds, les portes internes du circuit, sont étiquetés par des fonctions $\{0,1\}^r \to \{0,1\}$ avec r le degré entrant du noeud. Dans la suite pour une porte p du circuit, on note e_p la fonction qui l'étiquète. On ne considère que des fonctions parmi $\wedge, \vee, \neg$ respectivement le ET, OU, NON logique des booléens en entrée. Une *sortie du circuit* est un noeud de degré sortant 0.

Une assignation booléenne de $\mathcal{X}_n$ est une fonction $g\colon \mathcal{X}_n \to \{0,1\}$. Pour une porte p du circuit on définit son évaluation g_p comme suit :

— Pour une entrée du circuit $p = x_i$ est $g_p = g(x_i)$

— Pour un noeud interne p avec $p_1, \ldots p_r$ ses antécédents dans le circuits, $g_p = e_p(g_{p_1}, \ldots, g_{p_k})$

Une fonction $f\colon \{0,1\}^n \to \{0,1\}^k$ est calculée par un circuit si pour toute évaluation g, nous avons $f(g(x_1), \ldots, g(x_n)) = (g_{o_1}, \ldots, g_{o_k})$ avec $o_1, \ldots, o_k$ une séquence de portes de sortie du circuit. Un ensemble de mots $E \subseteq \{0,1\}^n$ est calculé par un circuit si sa fonction caractéristique (c'est-à-dire, la fonction $\mathbf{1}_E$ tel que $\mathbf{1}_E(u) = 1$ ssi $u \in E$) est calculée par le circuit.

FIGURE 1 – Un circuit calculant l'ensemble $\{00, 01, 11\}$

Dans la suite, on appellera *câbles* les arêtes du graphes.

Question 0. Montrez que toute fonction $f\colon \{0,1\}^n \to \{0,1\}$ est calculable par un circuit booléen. Indiquez le nombre de portes nécessaires et le nombre de câbles nécessaires.

Quelle relation y a-t-il entre le nombre de portes et le nombre de câbles dans un circuit ?

Question 1. La profondeur du circuit est le plus long chemin d'une entrée à la sortie.

Soit $k \in \mathbb{N}$. On note T_k^n l'ensemble des mots de $\{0,1\}^n$ qui ont au plus k positions à 1.

1. Proposez un circuit avec un nombre de câbles linéaire en n (mais pas forcément linéaire en k, qui est vue comme une constante fixée) qui calcule l'ensemble T_k^n. Quelle est sa profondeur en fonction de n ?
2. Proposez un circuit de profondeur constante qui calcule l'ensemble T_k^n. Quelle est sa taille (câbles et portes) en fonction de n ?

Question 2. On va construire un circuit de profondeur constante avec un nombre de câbles linéaire en n qui calcule T_k^n. Pour ce faire, on passe par une fonction intermédiaire qu'on nomme PREFIX-$\vee_n\colon \{0,1\}^n \to \{0,1\}^n$ telle que PREFIX- $\vee_n\,(x_1, \ldots, x_n)_i = \bigvee_{j \le i} x_j$.

Montrez que si un circuit de profondeur constante avec un nombre de câbles linéaire en n pour PREFIX-$\vee_n$ existe alors il en existe également un pour T_k^n.

Question 3. Construire un circuit de profondeur constante avec un nombre de câbles linéaire en n pour PREFIX-$\vee_n$.

Indication. On pourra regrouper les entrées par paquets de taille $\sqrt{n}$ afin d'identifier à gros trait où peut être le premier 1 dans le mot et appliquer de manière parcimonieuse un circuit naïf des ensembles portes de taille $\sqrt{n}$.

Question 4. Soit $\text{BIN}_n \colon \{0, \dots, 2^n\} \to 2^n$, la fonction qui associe un nombre à son écriture en binaire.

On considère maintenant la fonction $\text{ADDITION}_n \colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{n+1}$ qui réalise l'addition d'entiers représenté en binaire. Formellement, pour deux entiers $i, j < 2^n$, alors $\text{ADDITION}_n(\text{BIN}_n(i), \text{BIN}_n(j)) = \text{BIN}_{n+1}(i+j)$.

1. Montrez que ADDITION_n est calculable par un circuit avec un nombre de câbles linéaire en n.
2. Montrez que ADDITION_n est calculable par un circuit avec un nombre de câbles polynomial en n. et de profondeur constante.

Question 5. Montrez en appliquant une méthode similaire à la question 3, que si on sait implémenter ADDITION_n à profondeur constante avec un circuit utilisant $n \leq f(n) \leq n^2$ câbles, alors il est possible de construire un circuit avec un nombre de câbles en $O(\sqrt{n} f(\sqrt{n}))$.

En déduire, que pour tout $\epsilon > 0$, il existe un circuit calculant ADDITION_n utilisant un nombre de câbles $O(n^{1+\epsilon})$.

Question 6. Soit $G = (V, E)$ un graphe acyclique avec n sources $s_1, \dots, s_n$ (noeuds de degré entrant 0) et n sorties $o_1, \dots o_n$ (noeuds de degré sortant 0).

On dit que G est un n-super concentrateur si pour un entier $k < n$ et pour toute séquence $i_1 < j_1 < i_2 < \cdots < i_k < j_k$ il existe des k-chemins disjoints qui relient i_1 à j_1, i_2 à j_2, etc.

On admet le théorème difficile suivant.

Théorème.
Pour tout entier d, il existe une fonction $f_d \colon \mathbb{N} \to \mathbb{N}$ croissante et non-bornée telle que tout n-super concentrateur de profondeur d a un nombre d'arêtes au moins égal à $\Omega(n f_d(n))$.

En déduire qu'il n'existe pas de circuit de profondeur constante et avec un nombre de câbles linéaire en n qui calcule ADDITION_n.

On pourra éventuellement s'aider du théorème de *Menger* : étant donné deux ensemble de sommets I et J, le nombre maximal de chemins disjoints reliant I et J est égal à la taille de la plus petite coupe entre I et J.

Corrigé

Question 0 Une mise en forme normale suffit. Formellement $\bigvee_{u \in f^{-1}(1)} C_u$ où C_u est la clause qui décrit uniquement le mot u.

Question 1 L'objectif est qu'il trouve la construction qui réalise $+1$ en binaire sur une séquence de $\log(k)$ bit a saturation (si on déborde on reste a $1111 \cdots 111$). Cela peut être fait avec un circuit de taille exponentiel en k via la question précédente. On peut réaliser une construction par induction qui calcule le nombre de 1 dans $x_0, \cdots x_i$ sur $\log(k)$ bit avec saturation. Cela donne immédiatement le résultat avec un circuit de taille linéaire en câble et de profondeur linéaire.

À profondeur constante : on devine k positions et on vérifie qu'elle sont à 1. On obtient alors un circuit avec une disjonction de n^k porte ET.

Question 2 La réduction se voit très bien avec T_2.

Sur $x_1, \ldots, x_n$ en entrée, on note $y_1, \ldots y_n = \text{PREFIX-}\wedge_n (x_1, \ldots, x_n)$ et $z_i = y_i \wedge y_{i-1} \wedge x_i$. On remarque que z_i est à 1 ssi $i > j$ avec $j = \min_j x_j = 1$. Finalement $T_2 = \bigvee z_i$.

On peut généraliser en faisant une induction immédiate sur k. On conclut grâce à une petit induction immédiate.

Question 3 L'indication est importante.

On fait des paquet de $\sqrt{n}$ et on fait le OU de chaque paquet. Ça donne $\sqrt{n}$ bit. On peut faire une fonction préfixe naïve sur ça. C'est quadratique mais sur un ensemble de porte petit. Ça nous donne permet de récupérer le premier bloc où il y a un 1. On peut ensuite appliquer la fonction préfixe naïf sur ce bloc là uniquement.

La construction a quelques subtilité. Si on note $y_1, \ldots y_{\sqrt{n}}$ l'application du préfixe sur les blocs, alors on va construire $z_1, \ldots, z_{\sqrt{n}}$ qui va contenir une copie du premier bloc où il y a un 1.

Pour ça, on remarque que le premier bloc où il y a un 1 est identifié via $f_i = \neg y_{i-1} \wedge y_i$. Et donc on peut poser $z_i = \bigvee_{j=0}^{\sqrt{n}} f_j \wedge x_{j\sqrt{n}+i}$.

Pour conclure, il suffit de poser pour $o_i = z_i \wedge f_{\lceil i/\sqrt{n} \rceil} \vee (y_{i-1} \wedge y_i)$: c'est à dire, soit une sortie est dans le premier bloc avec un 1 et associé a une entrée qui est à 1 soit la sortie est après (et donc $y_{i-1} \wedge y_i$ est vrai).

Question 4 Pour la première question, on peut appliquer la même méthode que pour la fonction préfixe. On va simplement calculer séquentiellement le bit de retenu et le propager. Ça donne les bornes recherchés.

Pour la seconde version, il faut faire une construction plus compliqué : on calcul pour chaque position s'il y a une retenu qui se propage jusqu'à la position.

Pour ce faire, on doit faire une disjonction de tous les cas possible, ce qui donne un circuit quadratique. Une fois la retenu calculer, la conclusion est triviale.

Question 5 L'idée ici est de refaire la même construction par bloc de taille $\sqrt{n}$ que dans la question 3. Ça permet d'amortir un coup quadratique pour l'addition en faisant des blocs de taille $\sqrt{n}$. On va calculer l'addition séparément dans chaque bloc et calculer une retenue inter bloc potentiel et appliquer son effet à posteriori. Outre l'addition dans chaque bloc, le reste des opération est faisable avec un nombre linéaire de câble. Donc ça donne une complexité en $O(\sqrt{n} f(\sqrt{(n)}))$. Le point délicat c'est la propagation de retenue d'un bloc aux suivant. Mais ça encore, ça peut être fait avec un circuit linéaire qui va déterminer pour chaque retenu inter bloc jusqu'où potentiellement elle peut se propager. Pour ça, il faut l'information pour chaque bloc de si une retenu y est intercepté où non. Autrement dit, si deux positions alignées sont à 0.

Cette information peut être calculé avec un nombre linéaire de câbles. On remarque que pour appliquer une retenue à postériori sur un bloc, il suffit de prendre la négations de tous les bits jusqu'à la première position où elle est intercepté, ce qui peut être fait grâce à une fonction préfixe.

On obtient bien une complexité en $O(n+\sqrt{n}f(\sqrt{n})) = O(\sqrt{n}f(\sqrt{n}))$. Si on part avec une complexité en n^2, on obtient ensuite $n\sqrt{n}$ puis $n\sqrt{\sqrt{n}}$, et ainsi de suite. On obtient donc $nn^{1/2^d}$. On prend d suffisamment grand pour que $1/2^d < \epsilon$, ce qui permet de conclure.

Question 6 Voir le théorème 3.6 du papier :

Lower bounds for constant depth circuits for prefix problems, Ashok K. Chandra, Steven Fortune And Richard Lipton. ICALP 1983

L1 – Chip firing game

Un CFG ('chip firing game') est un jeu à un joueur qui se joue sur un graphe non-orienté $G = (V, E)$, qu'on suppose de plus connexe. Sur chaque sommet du graphe se trouve un certain nombre de jetons, ce qu'on représente par une fonction $j : V \to \mathbb{N}$, qu'on appelle une *configuration*. Un *coup* du joueur consiste à choisir un sommet $v \in V$ sur lequel se trouvent au moins autant de jetons que v a de voisins (ou encore : $j(v) \geq deg(v)$) et déplacer simultanément un jeton de v vers chacun de ses voisins (on obtient donc une nouvelle configuration j' où $j'(v) = j(v) - deg(v)$, $j'(u) = j(u) + 1$ pour tout $(u, v) \in V$ et $j'(u) = j(u)$ pour les autres sommets de G).

Une partie finie à partir d'une configuration c_0 est une suite de coups

$$c_0 \to c_1 \to \ldots \to c_n$$

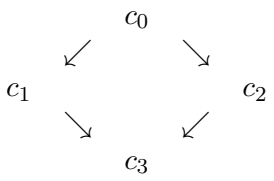
telle qu'aucun coup ne peut être joué à partir de la configuration c_n (on dit que la configuration c_n est *terminale*). De même, une partie infinie à partir d'une configuration c_0 est une suite infinie de coups

$$c_0 \to c_1 \to \ldots \to c_n \to \ldots$$

(dans laquelle aucune configuration c_i n'est donc terminale).

Question 1. Donner un exemple de partie finie sur un graphe de votre choix à partir d'une configuration de votre choix. Même question pour avec une partie infinie.

Question 2. Montrer que si c_0 est une configuration dans laquelle deux coups différents $c_0 \to c_1$ et $c_0 \to c_2$ sont possibles, alors il existe une configuration c_3 telle que $c_1 \to c_3$ et $c_2 \to c_3$ sont des coups. Graphiquement :



Question 3. Déduire de la question précédente qu'à partir d'une configuration c_0, s'il existe une partie finie alors toutes les parties sont finies, ont la même longueur et se terminent toutes sur la même configuration terminale.

Question 4. Montrer que s'il existe une partie infinie, alors chaque sommet est choisi par le joueur infiniment souvent dans la partie.

Question 5. Dans cette question on fixe un graphe $G = (V, E)$ et on s'intéresse maintenant au lien entre nombre total de jetons $\sum_{v \in V} j(v)$ sur le graphe et l'existence d'une partie finie. Montrer par un exemple que l'existence d'une partie finie ne dépend pas uniquement du nombre total de jetons.

Ce que l'on veut déterminer sont les deux valeurs seuil $T^*(G)$ et $T^\infty(G)$ telles que :

— Toute configuration avec un nombre total $t < T^*(G)$ de jetons n'admet que des parties finies.

— Toute configuration avec un nombre total $t > T^\infty(G)$ de jetons n'admet que des parties infinies.

— Pour $T^*(G) \leq t \leq T^\infty(G)$, certaines configurations à t jetons n'admettent que des parties finies, d'autres configurations à t jetons n'admettent que des parties infinies.

Question 6. Quelle est la valeur de $T^\infty(G)$?

Question 7. Montrer que $T^*(G) = |E|$. *Indication : On pourra s'aider de la notion suivante. Supposons que l'on mette un ordre total $\prec$ sur V. On note $deg^+(u, \prec)$ le nombre de voisins v de u tels que $v \prec u$. Etant donnés une configuration j et un ordre $\prec$, on dit que u est* deficient (pour l'ordre $\prec$) *si $j(u) < deg^+(u, \prec)$.*

Corrigé

Question 1. Pour les parties finies on a l'embarras du choix, on peut même prendre n'importe quel graphe avec aucun jeton dessus ! Pour les parties infinies, on prend par exemple un graphe à deux sommets avec un jeton sur l'un des deux qui fait 'ping-pong' indéfiniment.

Question 2. Raffinons la notation, en utilisant $c \xrightarrow{v} c'$ pour dire que c' a été obtenue à partir de c en jouant sur le sommet v.

Maintenant, si on a deux coups possibles $c_0 \xrightarrow{u} c_1$ et $c_0 \xrightarrow{v} c_2$ ($u \neq v$), alors le sommet v peut être joué dans la position c_1 : en effet, il peut être joué dans la position c_0 et comme $u \neq v$, le nombre de jetons du sommet v dans c_1 est au moins égal à celui de ce même sommet dans c_0. On a donc un coup valide $c_1 \xrightarrow{v} c_3$. Par le même raisonnement, on a un coup valide $c_2 \xrightarrow{u} c'_3$. Un coup pouvant être vu comme une addition de vecteur (en terme de nombre de jetons), jouer d'abord sur le sommet u puis sur le sommet v et jouer d'abord sur le sommet v puis le sommet u aboutit à la même configuration, d'où $c_3 = c'_3$.

Question 3. On appelle *hauteur* d'une configuration la plus courte partie finie que l'on peut obtenir à partir de celle-ci. Par convention cette hauteur est ∞ s'il n'existe pas de partie finie. On prouve le résultat voulu par récurrence sur la hauteur h d'une configuration.

Si une configuration c a hauteur 0, c'est qu'elle est terminale. Toutes les propriétés que l'on voulait sont vérifiées trivialement.

Supposons maintenant le résultat vrai jusqu'à hauteur h. Soit une configuration c_0 de hauteur $h+1$ et soit

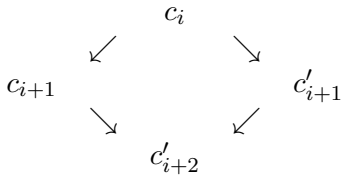
$$c_0 \to c_1 \to \ldots \to c_{h+1}$$

une plus courte partie finie, avec c_{h+1} terminale.

La configuration c_1 a hauteur $\leq h$ (puisqu'on a une partie de longueur h à partir de c_1), donc par récurrence, à partir de c_1, toutes les parties sont finies, ont même longueur (et cette longueur est donc h) et se terminent toutes sur même configuration, qui est donc c_{h+1}.

On distingue deux cas :

- Soit le coup $c_0 \to c_1$ était le seul possible à partir de c_0. Dans ce cas, d'après ce qu'on vient d'établir sur c_1, toutes les parties à partir de c_0 ont longueur $h+1$ et se terminent en c_{h+1}, ce que l'on voulait.
- Sinon, considérons n'importe quel un autre coup possible $c_0 \to c'_1$ avec $c'_1 \neq c_1$. Dans ce cas, en partant de $i = 0$, tant que $c_{i+1} \neq c_i$, on utilise la Question 2 pour obtenir un diamant



Comme c_{h+1} est terminale, ce processus va donc finir par s'arrêter, c'est-à-dire qu'on aura $c'_{i+1} = c'_{i+1}$ pour un certain i entre 0 et h. On a donc montré l'existence d'une partie de longueur h à partir de c'_1 menant à c_{h+1}. Par hypothèse de récurrence, toute partie à partir de c'_1 a longueur h et se termine par c_{h+1}. Comme on a montré cela pour tout premier coup possible à partir de c_0, on a bien le résultat voulu.

Question 4. Supposons qu'on ait une partie infinie. Par le principe des tiroirs au moins un sommet est choisi infiniment souvent au cours de cette partie. On va maintenant montrer que l'ensemble I des sommets choisis infiniment souvent est clos par voisinage. Comme le graphe est connexe, cela prouve le résultat.

Soit $u \in I$ et v adjacent à u. Pour tout N, il existe un moment de la partie où u a été choisi N fois. Pour chacun des coups où u a été choisi, v a reçu un jeton. Comme v ne peut pas avoir à ce moment-là plus de jetons que le nombre total t de jetons, il a dû en perdre dans le même temps au moins $N - t$. Or v ne peut perdre des jetons que quand il est choisi, et en perd $d = deg(v)$ à chaque fois. Donc si u a été choisi N fois, v a dans le même temps été choisi au moins $(N - t)/d$ fois. N est arbitrairement grand, t et d sont des constantes, on a donc ce que l'on voulait.

Question 5. Prenons le graphe triangle (3 sommets, trois arêtes), et 3 jetons. Si on répartit les jetons en $1 - 1 - 1$ on a une configuration terminale. Si on répartit en $2 - 1 - 0$ alors on peut jouer sur le sommet à 2 jetons et obtenir une nouvelle configuration $0 - 2 - 1$ qui n'est que le symétrisé de la position de départ, on n'a donc que des parties infinies.

Question 6. Regardons le nombre maximum de jetons d'une configuration terminale. Pour qu'on ne puisse pas jouer sur un sommet v, il faut qu'il y ait au plus $deg(v) - 1$ jetons dessus. Le nombre maximum de jetons d'une configuration terminale est donc $\sum_v (deg(v) - 1) = 2|E| - |V|$. Donc : si on a strictement plus que $2|E| - |V|$ jetons, on n'arrivera jamais à une configuration terminale, le nombre de jetons restant constant. A l'inverse, en mettant exactement $deg(v) - 1$ jetons sur chaque sommet v, on obtient une configuration terminale. Ces deux faits établissent respectivement $T^\infty \leq 2|E| - |V|$ et $T^\infty \geq 2|E| - |V|$, d'où $T^\infty = 2|E| - |V|$.

Question 7. Toute la solution à cette question repose sur le lemme suivant (qui peut être donné en indication supplémentaire) : Soit c_0 une configuration et $\prec_0$ un ordre sur les sommets ; si on joue un coup $c_0 \to c_1$, alors il existe un ordre $\prec_1$ tel que l'ensemble des sommets déficients de c_1 pour l'ordre $\prec_1$ est plus petit que, ou égal à, l'ensemble des sommets déficients de c_0 pour $\prec_0$. En effet, soit v le sommet sur lequel s'est joué le coup. Prenons pour $\prec_1$ l'ordre dont le plus petit élément est v et où les autres sommets sont ordonnés selon $\prec_0$. Clairement, le sommet v n'est pas déficient dans $(c_1, \prec_1)$ puisque, étant le plus petit élément pour $\prec_1$, on a $deg^+(v, \prec_1) = 0$. Pour les sommets u non-adjacents à v, la quantité $j(u) - deg^+(u, \prec)$ est inchangée par le coup joué puisque $j(u)$ et $deg^+(u, \prec)$ sont tous deux inchangés. Pour les sommets u adjacents à v, $j(u)$ augmente de 1 après le coup, tandis que $deg^+(u, \prec)$ augmente d'au plus 1 (dans le cas où $u \prec_0 v$ mais $v \prec_1 u$). Donc un sommet non-déficient reste non-déficient après le coup.

Considérons une partie infinie. Mettons un ordre initial $\prec_0$ arbitraire sur les sommets et mettons à jour l'ordre $\prec$ à chaque coup comme dans le lemme. A chaque coup, l'ensemble des sommets déficients diminue ou reste le même donc l'ensemble des sommets déficients finit par se stabiliser. Mais il ne peut se stabiliser que sur l'ensemble vide ! En effet, par la Question 4, tout sommet d'une partie infinie est joué infiniment souvent, et quand un sommet est choisi pour jouer un coup, ce sommet ne peut être déficient, quel que soit l'ordre. Donc, au bout d'un moment, aucun sommet n'est plus déficient. Au moment où cela arrive, on a pour tout sommet v l'inégalité $j(v) \geq deg^+(v, \prec)$ et donc le nombre total de jetons $\sum_v j(v)$ est supérieur ou égal à $\sum_v deg^+(v, \prec) = |E|$. Le nombre de jetons étant inchangé

au cours du temps, cela veut dire qu'il fallait au moins $|E|$ jetons au départ pour avoir une partie infinie.

Réciproquement, si on nous donne $|E|$ jetons, on peut obtenir une partie infinie ainsi. On prend un ordre initial $\prec_0$ quelconque sur les sommets puis on prend initialement $j(u) = deg^+(u, \prec_0)$ sur chaque sommet, de sorte qu'on a placé exactement $|E|$ jetons et qu'aucun sommet n'est déficient. Observons alors que si l'on est dans une configuration $(c, \prec)$ sans somment déficient, il est toujours possible de jouer. En effet, soit v le plus grand élément pour $\prec$; pour ce somment on a $deg(v) = deg^+(v, \prec)$ donc $j(v) \geq deg(v)$, ce qui veut dire qu'on peut jouer un coup en choisissant v. Par le lemme ci-dessus, on peut alors mettre un nouvel ordre sur V pour n'avoir toujours aucun sommet déficient. On peut alors jouer un nouveau coup, et ainsi de suite. Par récurrence, on obtient une partie infinie.

L2 – Autour de l'inégalité de Kraft

Dans ce qui suit, on se fixe un alphabet fini Σ de cardinalité $K \geq 2$. On rappelle qu'un mot $u \in \Sigma^*$ est *préfixe* d'un mot $w \in \Sigma^*$ s'il existe $u' \in \Sigma^*$ tel que $uu' = w$. On dit qu'un ensemble de mots $\Gamma \subseteq \Sigma^*$ est *sans préfixe* si aucun mot u de Γ n'est préfixe d'un autre mot v de Γ ($v \neq u$).

Question 1. Donner un exemple d'ensemble Γ sans préfixe sur l'alphabet $\Sigma = \{a, b, c\}$, contenant au moins 6 mots. Donner maintenant un exemple d'ensemble sans préfixe infini.

Question 2. Soit $\Gamma \subseteq \Sigma^*$ un ensemble de mots sans préfixe ne contenant pas le mot vide. Montrer que Γ possède la *propriété de décomposabilité unique* : si un mot w peut s'écrire à la fois $w = u_1 \dots u_n$ et $w = v_1 \dots v_m$ avec tous les u_i et v_j dans Γ, alors $n = m$ et $u_i = v_i$ pour tout i.

Question 3. Etant donné un ensemble fini de mots Γ, proposer un algorithme naïf pour tester si Γ est sans préfixe. Quelle est sa complexité ?

Question 4. Un arbre K-aire est un arbre dont chaque noeud a au plus K fils. Une feuille de l'arbre est un noeud qui n'a aucun fils. On appelle *hauteur* d'un noeud sa distance à la racine (cette dernière a donc hauteur 0). Dit autrement, la racine a hauteur 0 et si un noeud a hauteur h, ses fils ont hauteur $h + 1$.

Soit T un arbre K-aire fini. Montrer que

$$\sum_{\sigma \text{ feuille de } T} K^{-hauteur(\sigma)} \leq 1$$

Dans quel cas a-t-on égalité ?

Question 5. En utilisant la question précédente, montrer que si $\Gamma \subseteq \Sigma^*$ est un ensemble fini sans préfixe, alors

$$\sum_{w \in \Gamma} K^{-|w|} \leq 1$$

On appelle cette inégalité l'*inégalité de Kraft*. Dans quel cas a-t-on égalité ? Est-ce que l'inégalité de Kraft reste vraie pour les ensembles infinis (sans préfixe) ?

Question 6. Proposer maintenant un algorithme en temps linéaire pour tester si un ensemble de mots finis est sans préfixe.

Question 7. On va maintenant donner une preuve algorithmique de la réciproque de l'inégalité de Kraft : si $(l_i)_{i=1}^n$ est une famille d'entiers telle que

$$\sum_{i=1}^{n} K^{-l_i} \leq 1$$

alors il existe une famille de mots distincts $(u_i)_{i=1}^n$ de Σ^* formant un ensemble sans préfixe avec $|u_i| = l_i$ pour tout i.

Donner un algorithme prenant en entrée une telle suite l_i qui retourne une telle suite u_i. On demande de plus que l'algorithme soit 'en ligne', c'est-à-dire que l'algorithme lit les l_i un par un et produit u_i immédiatement après la lecture de l_i, sans attendre la valeurs des l_j pour $j > i$. *Indication : Utiliser un invariant faisant intervenir, à l'étape $t + 1$, l'écriture en base K de $1 - \sum_{i=1}^{t} K^{-l_i}$.*

Question 8. Montrer que l'inégalité de Kraft reste vraie pour tout ensemble de mots Γ (fini ou infini) vérifiant la propriété de décomposabilité unique de la Question 2. *Indication : On pourra considérer la série entière $\sum_n a_n x^n$ où a_n est le nombre de mots de Γ de longueur n – que sait-on de son rayon de convergence ? – et étudier $(\sum_n a_n x^n)^N$ pour un 'grand' entier N.*

Corrigé

Question 1. Trouver des ensembles finis sans préfixe est trivial. Pour un ensemble infini, on peut prendre par exemple

$$\Gamma = \{a^n b \mid n \in \mathbb{N}\}$$

Question 2. Par récurrence sur la longueur de w. Pour $|w| = 0$ aucune décomposition n'est possible car Γ ne contient pas le mot vide, la propriété est donc satisfaite de facto. Supposons maintenant qu'on a deux décompositions

$$w = u_1 \dots u_n = v_1 \dots v_m$$

(avec les u_i et v_j dans Γ, donc en particulier non-vides). Sans perte de généralité on a $0 < |u_1| \leq |v_1|$, et l'égalité ci-dessus implique que u_1 est donc un préfixe de v_1, donc $u_1 = v_1$ car Γ est sans préfixe. Il suit que $u_2 \dots u_n = v_2 \dots v_m$ et on conclut par récurrence, le mot $w' = u_2 \dots u_n$ étant de longueur strictement inférieure à celle de w.

Question 3. On teste pour toute paire (u, v) de mots distincts de Γ si u est un préfixe de v. Chaque comparaison prend un temps $O(|u| + |v|)$ donc on a une complexité totale en $O(n^2)$, n étant la taille mémoire de l'entrée Γ.

Question 4. Par récurrence sur la hauteur h de l'arbre, on montre en même temps l'inégalité et le fait qu'on a égalité si et seulement si l'arbre est localement complet (tout noeud est soit une feuille soit a exactement K fils). On notera $f(T) = \sum_{\sigma \text{ feuille de } T} K^{-hauteur(\sigma)}$.

Pour $h = 0$, on a une seule feuille de hauteur 0 donc $\sum_{w \in \Gamma} K^{-|w|} = 1$ et l'arbre est bien localement complet.

Pour $h + 1$, on se observe que

$$f(T) = \frac{1}{K} \sum_i f(T'_i)$$

la somme étant prise sur les sous-arbres non-vides de T. En effet la hauteur d'un noeud dans un T'_i est un de moins que sa hauteur dans T. Par récurrence, on a $f(T'_i) \leq 1$ pour tout i et on a au plus K sous-arbres, d'où l'inégalité $f(T) \leq 1$. Pour avoir l'égalité, il faut avoir exactement K sous-arbres et $f(T'_i) = 1$ pour chacun d'entre eux, ce qui oblige par récurrence a avoir chaque T'_i localement complet et donc T doit être lui-même localement complet.

Question 5. Il y a une interprétation assez évidente d'ensemble préfixe (non-vide) en terme d'arbre. Soit h la longueur du plus long mot de Γ. On peut identifier chaque mot de Γ comme un noeud de l'arbre K-aire parfait T de hauteur h, où la racine correspond au mot vide et inductivement si un noeud correspond au mot w, ses fils correspondent aux mots $\{wa \mid a \in \Sigma\}$. Le fait d'être sans préfixe est équivalent à ce que parmi les noeuds de Γ on n'en ait pas un qui soit préfixe d'un autre. On peut alors considérer l'arbre $T_\Gamma = \{u \mid u \text{ est préfixe d'un mot } w \in \Gamma\}$. Si Γ est sans préfixe, chaque $w \in \Gamma$ est une feuille de T_Γ, avec $hauteur(w) = |w|$. L'inégalité de Kraft suit.

De plus, on a égalité quand Γ est *maximal*, c'est-à-dire contenu dans aucun ensemble de mots sans préfixe strictement plus grand. Par la Question 4, il suffit de montrer que Γ est maximal si et seulement si son arbre T_Γ est localement parfait. En effet, si T_Γ n'est pas localement parfait, un noeud non

feuille w a l'un de ses fils wa manquants, on peut donc rajouter le mot wa à Γ en préservant la propriété d'être sans préfixe. A l'inverse, si Γ n'est pas maximal, considérons le mot le plus court parmi les mots qu'on peut ajouter à Γ. Ecrivons ce mot va avec $v \in \Sigma^*$ et $a \in \Sigma$ (pour être complètement précis il faut traiter le cas dégénéré où ce mot est le mot vide ; ceci arrive seulement quand $\Gamma = \emptyset$, et dans ce cas il n'y a rien à démontrer). Par hypothèse, v est préfixe d'un mot de Γ donc v est dans T_Γ mais pas va : T_Γ n'était donc pas parfait.

Montrons enfin que l'inégalité de Kraft reste vraie pour les ensembles sans préfixe infini. Soit Γ un tel ensemble, et notons Γ_n l'ensemble des n-premiers éléments de Γ. Comme on a à faire à une série de termes positifs uniquement, par le théorème de converge monotone,

$$\sum_{w\in\Gamma} 2^{-|w|} = \lim_n \sum_{w\in\Gamma_n} 2^{-|w|}$$

Or Γ_n est sans préfixe pour tout n (un sous-ensemble d'un ensemble sans préfixe est sans préfixe), donc $\sum_{w\in\Gamma_n} 2^{-|w|} \leq 1$ pour tout n et la limite est donc elle-même ≤ 1. [Par contre, pour les ensembles infinis on n'a plus égalité ssi maximalité].

Question 6. Il suffit de placer les mots un par un dans un arbre K-aire comme dans la question précédente. A chaque ajout on parcourt l'arbre et si on rencontre un noeud déjà pris, alors Γ n'est pas sans préfixe. Si aucun conflit n'a lieu durant la construction, alors Γ est sans préfixe. Clairement cet algorithme tourne en temps linéaire.

Question 7. Là encore on va placer les mots u_i dans un arbre K-aire dont ils seront des feuilles (ce qui garantira la propriété d'être sans préfixe). On commence avec l'arbre réduit à la racine, qui va grandir au fur et à mesure. A chaque étape on aura deux types de feuilles : les feuilles déjà prises par un mot de w et les feuilles disponibles. De plus, on va s'arranger pour qu'après l'étape t (c'est-à-dire après avoir construit et placé $(u_i)_{i=1}^t$) en écrivant le réel $1 - \sum_{i=1}^t K^{-l_i}$ en base K :

$$1 - \sum_{i=1}^{t} K^{-l_i} = \sum_{n=0}^{\infty} a_n K^{-n}$$

où a_n est nulle à partir d'un certain rang, on ait exactement a_n feuilles de disponibles à la hauteur n. Initialement c'est bien ce qu'on a : la racine a hauteur 0 et est disponible.

Supposons qu'on ait jusqu'à maintenant respecté cette propriété après t étapes. On lit maintenant la valeur l_{t+1} et o distingue deux cas :

— Soit il y a au moins une feuille de disponible à la hauteur l_{t+1}. Dans ce cas on prend une telle feuille et elle devient notre u_{t+1} (et donc devient non-disponible)

— Soit aucune feuille de hauteur l_{t+1} n'est disponible. Dans ce cas on cherche le plus grand $n < l_{t+1}$ tel qu'on a une feuille w de disponible à la hauteur n. On agrandit alors l'arbre sous cette feuille, en ajoutant les feuilles disponibles suivantes :

$$\{wa^m b \mid b \in \Sigma, b \neq a, m \in [0, l_{t+1} - n - 1]\}$$

où a est une lettre fixée de Σ, ainsi que les noeuds internes

$$\{wa^m \mid m \in [0, l_{t+1} - n - 1]\}$$

et en ajoutant également la feuille $wa^{l_{t+1}-n}$, qu'on attribue à u_{t+1}, et qui a bien longueur $t+1$. Dans les deux cas on a bien préservé l'écriture en base K de $1 - \sum_{i=1}^t K^{-l_i}$: le premier cas correspond à

$$\begin{array}{cccccc} & \ldots & a_{n-2} & a_{n-1} & a_n & \ldots \\ - & & 0 & 0 & 1 & \\ \hline & \ldots & a_{n-2} & a_{n-1} & (a_n-1) & \ldots \end{array}$$

(avec $a_n > 0$) tandis que le second correspond à la propagation de la retenue

$$\begin{array}{ccccccccc} & \ldots & a_{n-2} & a_{n-1} & a_n & 0 & \ldots & 0 \\ - & & 0 & 0 & 0 & 0 & \ldots & 1 \\ \hline & \ldots & a_{n-2} & a_{n-1} & (K-1) & (K-1) & \ldots & (K-1) \end{array}$$

Ceci prouve la correction de l'algorithme. [Il fonctionne, comme le test de la question précédente, en temps linéaire].

Question 8. Formons la série entière proposée $H(x) = \sum_n a_n x^n$. Comme a_n est le nombre de mots de Γ de longueur n, on a $a_n \leq K^n$. Donc si $x < 1/K$, on a convergence absolue de la série, le rayon de convergence est donc $\geq 1/K$. De plus, tous les a_n étant positifs, on a, encore une fois par le théorème de convergence monotone,

$$\sum_{w\in\Gamma} K^{-|w|} = \sum_n a_n K^{-n} = \lim_{x\to(1/K)^-} \sum_n a_n x^n$$

On va maintenant montrer que $\lim_{x\to(1/K)^-} \sum_n a_n x^n \leq 1$. Soit $x \in [0, 1/K[$ et N un entier quelconque. On a

$$\left(\sum_n a_n x^n\right)^N = \sum_n \left(\sum_{\substack{i_1,\ldots,i_N \\ i_1+\ldots+i_N=n}} a_{i_1} + \ldots + a_{i_N} \right) x^n$$

Mais le terme

$$\sum_{\substack{i_1,\ldots,i_N \\ i_1+\ldots+i_N=n}} a_{i_1} + \ldots + a_{i_N}$$

est exactement le nombre de façons de choisir un N-uplet $(u_1, \ldots, u_N)$ avec tous les u_i dans Γ et tels que $|u_1|+\ldots+|u_N| = n$. Mais par la propriété d'unique décomposabilité, à chaque tel choix correspond un unique $w = u_1 \ldots u_N$ de longueur n. On a donc

$$\sum_{\substack{i_1,\ldots,i_N \\ i_1+\ldots+i_N=n}} a_{i_1} + \ldots + a_{i_N} \leq K^n$$

pour tout n. Donc, pour tout $x < 1/K$ fixé, on a

$$H(x)^N = \left(\sum_n a_n x^n\right)^N = \sum_n \left(\sum_{\substack{i_1,\ldots,i_N \\ i_1+\ldots+i_N=n}} a_{i_1} + \ldots + a_{i_N} \right) x^n \leq \sum_n K^n x^n < \infty$$

En appelant $G(x) = \sum_n K^n x^n$, qui est aussi une série entière de rayon de convergence $\geq 1/K$, on a établi :

$$H(x)^N \leq G(x)$$

pour tout $|x| < 1/K$ et tout ăN. Cela implique nécessairement $H(x) \leq 1$ pour tout $|x| < 1/K$, car sinon l'inégalité ci-dessus serait violée pour N assez grand. On a donc bien

$$\lim_{x\to(1/K)^-} H(x) \leq 1$$

ce que l'on voulait.

L3 – Ordres et intervalles

On rappelle que sur un ensemble E donné, un *ordre strict* est une relation $\prec$ irréflexive et transitive (et donc aussi anti-symétrique), pas nécessairement totale (pour $x, y \in E$, on peut n'avoir ni $x \prec y$, ni $y \prec x$).

Question 1. Soit $\mathcal{I}$ l'ensemble des intervalles $[a, b]$ (avec $a \leq b$) de $\mathbb{R}$. Montrer que la relation $\prec_{\mathcal{I}}$ definie sur $\mathcal{I}$ par

$$[a, b] \prec_{\mathcal{I}} [c, d] \Leftrightarrow b < c$$

est bien un ordre strict.

Soit E un ensemble et $\prec$ un ordre strict sur cet ensemble. On dit que $\prec$ est un *ordre d'intervalles* si on peut associer à chaque $x \in E$ un intervalle $[a_x, b_x]$ de $\mathbb{R}$ de telle sorte que pour tout $x, y \in E$:

$$x \prec y \Leftrightarrow [a_x, b_x] \prec_{\mathcal{I}} [a_y, b_y]$$

Dans ce qui suit, E sera un ensemble fini à n éléments qu'on identifie à $\{1, \ldots, n\}$ et $\prec$ une relation d'ordre strict sur E représenté par une matrice M de taille $n \times n$ où $M[i, j] = 1$ si $i \prec j$, et $M[i, j] = 0$ sinon.

Question 2. Montrer que si $\prec$ est un ordre d'intervalles, alors $(E, \prec)$ a la propriété suivante, qu'on note $(\star)$: il n'existe pas $x, x', y, y' \in E$ tels que : $x \prec x'$, $y \prec y'$, $x \not\prec y'$ et $y \not\prec x'$.

On admet pour le moment que la réciproque de la question précédente est vraie (c'est le théorème de Fishburn, qu'on démontrera dans les questions suivantes) : si $(E, \prec)$ a la propriété $(\star)$, alors $\prec$ est un ordre d'intervalles. En déduire un algorithme polynomial pour tester si la matrice M représente un ordre d'intervalles. Quelle est en fonction de n la complexité de cet algorithme ?

Question 3. Pour tout $x \in E$, on pose

$$D(x) = \{y \in E \mid y \prec x\}$$

Montrer que si $(E, \prec)$ possède la propriété $(\star)$, alors il possède la propriété $(\star\star)$ suivante : pour tout $x, x' \in E$, on a soit $D(x) \subseteq D(x')$, soit $D(x') \subseteq D(x)$.

Question 4. Montrer maintenant que si $(E, \prec)$ possède la propriété $(\star\star)$, alors $\prec$ est un ordre d'intervalles (ce qui conclut la preuve du théorème de Fishburn). *Indication : en raisonnant pas récurrence, on pourra isoler un élément x qui soit le 'plus maximal' possible en utilisant la question précédente.*

Question 5. A partir des questions précédentes, donner maintenant un algorithme en $O(n^2)$ qui décide si M représente un ordre d'intervalles.

Corrigé

Question 1. La vérification est triviale.

Question 2. Par l'absurde, si on a cette configuration sur un ordre d'intervalle, alors les quatre $\prec$-inégalités nous donnent respectivement :

— $b_x < a_{x'}$

— $b_y < a_{y'}$

— $a_{y'} \leq b_x$

— $a_{x'} \leq b_y$

En mettant cela ensemble :

$$a_{x'} \leq b_y < a_{y'} \leq b_x < a_{x'}$$

contradiction.

L'algorithme qui découle de cela est très simple : on teste tous les quadruplets ! Coût (en temps) $O(n^4)$.

Question 3. Si on n'a ni $D(x) \subseteq D(x')$ ni $D(x') \subseteq D(x)$, alors il existe $y \in D(x) \setminus D(x')$ et $z \in D(x') \setminus D(x)$. Autrement dit : $y \prec x$, $y \not\prec x'$, $z \prec x'$ et $z \not\prec x'$. Ceci viole la condition $(\star)$.

Question 4. On procède par récurrence. Bien sûr pour un ensemble à 0 ou 1 élément il n'y a rien à prouver. Pour $n+1$ éléments, d'après la question précédente, il existe un x tel que $D(y) \subseteq D(x)$ pour tout $y \in E$. Cela veut dire en particulier que x est $\prec$-supérieur à tout élément non-maximal : si $y \prec y'$, alors $y \in D(y') \subseteq D(x)$, càd $y \prec x$, et incomparable avec les autres éléments maximaux (sinon ils ne seraient pas maximaux !).

Par récurrence, $\prec$ est un ordre d'intervalles sur $E \setminus \{x\}$. On peut donc considérer une famille d'intervalles $([a_y, b_y])_{y \in E \setminus \{x\}}$ qui témoigne de ce fait. Attribuons à x un intervalle $[a_x, b_x]$ où a_x est plus grand que tous les b_y, ce qui place alors $[a_x, b_x]$ $\prec_I$-au dessus de tous les autres, et $b_x > a_x$ est quelconque. Modifions alors les intervalles associés aux autres éléments maximaux : pour tout y maximal, remplaçons l'intervalle $[a_y, b_y]$ par $[a_y, b_x]$. Ce nouvel intervalle reste bien $\prec_I$-au dessus de ceux qui étaient déjà plus petits que lui, et reste incomparable à ceux auquel il était déjà incomparable (en effet, pour un intervalle $[a, b]$ $\prec_{\mathcal{I}}$-maximal, être incomparable avec $[c, d]$ veut dire que $a < d$, or la modification que l'on fait ci-dessus ne change pas les extrémités gauches des intervalles et ne fait qu'augmenter les extrémités droites). Enfin, $[a_y, b_x]$ est incomparables avec $[a_x, b_x]$ puisque $b_x > a_x > a_y$. On a donc bien un ordre d'intervalles, ce qui termine la récurrence.

Question 5. On a établi que d'être un ordre d'intervalle est équivalent à la propriété $(\star\star)$, c'est cette dernière que notre algorithme va tester.

On commence, pour tout $j \in \{1, ..., n\}$, par compter la cardinalité de $D(j)$ tel que défini dans la question précédente. C'est simplement la somme des éléments de la j-ème colonne de M, donc coût $O(n^2)$.

On trie alors les entiers $j \in \{1, ..., n\}$ selon la taille de $D(j)$:

$$|D(j_1)| \leq |D(j_2)| \leq \cdots |D(j_n)|$$

ce qui a un coût de $O(n \log n)$.

Pour que la propriété $(\star\star)$ soit vérifiée, il faut et il suffit que $D(j_k) \subseteq D(j_{k+1})$ pour tout $k \in \{1, \dots n-1\}$. Cela se vérifie en temps $O(n^2)$: on initialise un ensemble H à $\emptyset$ et pour tout $k \in \{1, \dots n-1\}$, on vérifie si $H \subseteq D(j_{k+1})$. Si oui $H := D(j_{k+1})$, sinon, on a montré que l'ordre n'est pas un ordre d'intervalle. Si on a bien $H \subseteq D(j_{k+1})$ pour tout $k \in \{1, \dots n-1\}$, alors l'ordre est bien un ordre d'intervalles. Cette dernière partie de l'algorithme a un coût $O(n^2)$. Coût total : $O(n^2)$.

L4 – Mariage en treillis

Soient A et B deux ensembles finis disjoints, qu'on suppose de même cardinalité n. On cherche à « marier » les éléments de A aux éléments de B, chaque élément devant être marié à exactement un élément de l'autre ensemble (un mariage $\mathcal{M}$ peut donc être représenté par une bijection de A dans B). De plus, chaque élément de A (respectivement de B) a un ordre de préférence strict et total entre les éléments de B (respectivement de A). On notera $<^x$ l'ordre associé à l'élément x (le x pouvant être omis quand le contexte est clair). On appellera *système de préférences* la liste des ordres $<^x$ pour $x \in A \cup B$.

Un mariage $\mathcal{M}$ est *instable* s'il existe un quadruplet $(a_1, a_2, b_1, b_2) \in A \times A \times B \times B$ tel que

— a_1 est marié à b_2 via $\mathcal{M}$;

— b_1 est marié à a_2 via $\mathcal{M}$;

— a_1 préfère b_1 à b_2 (autrement dit : $b_2 <^{a_1} b_1$) ;

— b_1 préfère a_1 à a_2 (autrement dit : $a_2 <^{b_1} a_1$).

Un mariage qui n'est pas instable est dit *stable*.

Question 1. Donner un exemple d'ensembles A, B à deux éléments chacun avec des ordres de préférence tel qu'il existe plus d'un mariage stable.

L'algorithme de Gale–Shapley permet de montrer qu'il existe toujours au moins un mariage stable et d'en trouver un de façon efficace. L'idée de l'algorithme est que les éléments de A font des propositions aux éléments de B. Quand un élément de A fait une proposition à un élément de B, celui-ci peut soit la refuser définitivement, soit l'accepter provisoirement (en attendant une éventuelle meilleure offre). Toute nouvelle acceptation vaut rejet définitif des propositions précédemment acceptées. Cela donne l'algorithme suivant.

```
Initialiser M = []
Tant qu'un élément x de A n'est pas marié
        y = élément de B que x préfère parmi ceux à qui x n'a pas encore proposé
        Si y n'est pas marié dans M
                Ajouter (x,y) à M
        Sinon si y est marié dans M à x' et y préfère x à x'
                Retirer (x',y) de M
                Ajouter (x,y) à M
        Sinon
                 Ne rien faire

Retourner M
```

Question 2. Montrer que l'algorithme de Gale–Shapley termine. Proposer des structures de données permettant d'implémenter l'algorithme le plus efficacement possible ; quelle est la complexité qui en résulte ?

Question 3. Montrer la correction de cet algorithme, à savoir qu'on obtient bien un mariage stable à la fin de l'exécution. *Indication : Fixer un $a \in A$ et regarder ce qui lui arrive au cours de l'algorithme. Même chose pour un $b \in B$.*

Question 4. Soit $\mathcal{E}$ un ensemble non vide et $\preceq$ une relation d'ordre (pas nécessairement totale) sur $\mathcal{E}$. On dit qu'un élément $e \in \mathcal{E}$ est *maximal* s'il n'existe pas $e' \neq e$ tel que $e \preceq e'$. On dit que e est un élément *maximum* (ou *le* maximum car il est alors unique) si pour tout e' on a $e' \preceq e$.

Un ensemble quelconque non vide $\mathcal{E}$ muni d'une relation d'ordre a-t-il toujours un élément maximal ? Un élément maximum ? Et si $\mathcal{E}$ est fini ?

Question 5. On suppose maintenant fixés les ensembles A, B et le système de préférences. On définit un ordre partiel $\preceq$ sur les mariages stables, où $\mathcal{M}_1 \preceq^A \mathcal{M}_2$ veut dire que les éléments de A sont tous au moins aussi satisfaits dans le mariage $\mathcal{M}_2$ que dans le mariage $\mathcal{M}_1$ (c'est-à-dire, pour tout $a \in A$, si a est marié avec b_1 dans $\mathcal{M}_1$ et b_2 dans $\mathcal{M}_2$, alors $b_1 \preceq^a b_2$). $\mathcal{M}_1 \preceq^B \mathcal{M}_2$ est défini de façon similaire, quand tous les éléments de B sont au moins satisfaits dans $\mathcal{M}_2$ que dans $\mathcal{M}_1$.

Montrer que si $\mathcal{M}_1$ et $\mathcal{M}_2$ sont deux mariages stables et que $\mathcal{M}_1 \preceq^A \mathcal{M}_2$, alors $\mathcal{M}_2 \preceq^B \mathcal{M}_1$.

Question 6. Toujours en supposant fixés les ensembles A, B et le système de préférences, on définit l'opération $\vee$ sur les mariages stables de la façon suivante : pour tout $a \in A$, si a est marié à b_1 dans $\mathcal{M}_1$ et à b_2 dans $\mathcal{M}_2$, on marie a à son élément préféré de $\{b_1, b_2\}$. Montrer qu'on obtient bien ainsi un mariage stable, qu'on note donc $\mathcal{M}_1 \vee \mathcal{M}_2$.

Question 7. Montrer que dans l'ensemble des mariages stables, il existe un élément maximum et un élément minimum pour l'ordre $\preceq^A$.

Question 8. Montrer que le mariage produit par l'algorithme de Gale–Shapley est en fait l'élément maximum pour $\preceq^A$. *Indication : pour $a \in A$, on dit que $b \in B$ est un* partenaire valide *de a s'il existe un mariage stable où a est marié à b. Dans l'exécution de Gale–Shapley, considérer le premier moment où un élément de A est rejeté par un de ses partenaires valides.*

Corrigé

Question 1. Par exemple, avec $A = \{a_1, a_2\}$ et $B = \{b_1, b_2\}$, on peut prendre $<^{a_1} = \{(b_1, b_2)\}$, $<^{a_2} = \{(b_2, b_1)\}$, $<^{b_1} = \{(a_2, a_1)\}$ et $<^{b_2} = \{(a_1, a_2)\}$. Alors les deux mariages suivants :

$$\mathcal{M}_B = \{(a_1, b_1), (a_2, b_2)\} \qquad \mathcal{M}_A = \{(a_1, b_2), (a_2, b_1)\}$$

sont stables car, dans $\mathcal{M}_B$, les éléments de B sont mariés à leur élément de A préféré ; et dans $\mathcal{M}_A$, les éléments de A sont mariés à leur élément de B préféré.

Question 2. À chaque passage dans la boucle, une paire (x, y) est retirée de la liste des propositions pas encore faites. Comme il y a n^2 propositions potentielles, l'algorithme termine au bout de $O(n^2)$ itérations.

On peut faire en sorte que la recherche d'un élément non encore marié soit en $O(1)$, et que le contenu de la boucle soit également en $O(1)$:

— on représente le mariage M par deux tableaux indiquant à quel élément de A un élément de B est marié, et à quel élément de B un élément de A est marié ;

— on maintient également une liste simplement chaînée contenant les éléments de A non mariés ;

— enfin, on maintient pour chaque élément de A un pointeur vers la position dans la liste de préférences de A du premier élément de B auquel A n'a pas encore proposé.

La complexité d'initialisation de ces deux structures de données est en $O(n)$. À chaque itération de la boucle, on enlève le premier élément de la liste chaînée pour obtenir un élément x, on récupère l'élément suivant de B dans sa liste de préférences et on incrémente le pointeur correspondant, puis on utilise les tableaux pour maintenir le mariage et on ajoute éventuellement un autre élément de A à la liste chaînée quand on change l'élément auquel un élément de B est marié. Cela donne bien une complexité de $O(1)$ pour chaque itération de la boucle.

Au final, on obtient une complexité de $O(n) + O(n^2) \times O(1) = O(n^2)$.

Question 3. Avant toute chose, regardons ce qui se passe pour un $b \in B$ fixé. Pendant un temps, b n'est pas marié. A la première proposition, b se marie. Par la suite, b peut se remarier plusieurs fois – et à chaque fois obtient un conjoint qu'il préfère au précédent – mais reste marié à tout instant.

Si on se fixe un $a \in A$, on voit que a va proposer selon son ordre de préférence, en descendant dans la liste à chaque rejet. Chaque a alterne donc des périodes de mariage et de célibat, ses conjoints diminuant en qualité (dans son ordre de préférence) au cours de l'algorithme.

Ces remarques étant faites, on doit d'abord montrer que l'algorithme retourne bien une bijection. Premièrement, un invariant de boucle est que M^{-1} est une injection partielle de B dans A à tout moment de l'algorithme (en effet quand une nouvelle paire (x', y) est ajoutée, on retire l'ancienne (x, y)). Comme on l'a vu dès qu'un y reçoit une proposition il se marie et reste marié, donc si le M final n'est pas une bijection, c'est qu'un certain y n'a jamais reçu de proposition. Mais alors par comptage, à la fin de l'algorithme un certain $x \in A$ est célibataire. Par la discussion ci-dessus, x a donc fait une proposition à tous les éléments de B, y compris y, contradiction.

Le M final est donc bien une bijection. Supposons que ce M ne soit pas stable, et qu'il existe donc (a_1, b_1, a_2, b_2) comme dans la définition. Si a_1 préfère b_1 à b_2, cela veut dire que a_1 a proposé à b_1 avant b_2 et a été rejeté (soit immédiatement, soit suite à une autre proposition que b_1 a reçue plus tard). Au moment de ce rejet, b_1 était marié à un certain a_3 qu'il préférait donc à a_1. Mais on a vu que les éléments de B amélioraient leurs mariages au cours de l'algorithme, donc b_1 préfère son conjoint final, a_2, à a_3. En d'autres termes, $a_2 \geq^{b_1} a_3 >^{b_1} a_1$. Cela contredit notre hypothèse que b_1 préfère a_1 à a_2.

Question 4. Évidemment certains ordres n'ont ni maximum ni même élément maximal, par exemple $(\mathbb{Z}, \leq)$. En revanche, si $\mathcal{E}$ est fini, on montre facilement par récurrence qu'il possède un élément maximal (mais pas forcément de maximum !).

Question 5. Supposons $\mathcal{M}_1 \preceq^A \mathcal{M}_2$. Soit b un élément quelconque de B. Supposons que b soit marié à a_1 dans $\mathcal{M}_1$ et à a_2 dans $\mathcal{M}_2$. Si ces éléments sont distincts, soit b' le conjoint de a_2 dans $\mathcal{M}_1$. On sait que a_2 préfère b à b' (car $\mathcal{M}_1 \preceq^A \mathcal{M}_2$). Par stabilité dans $\mathcal{M}_1$, il faut donc que b préfère a_1 à a_2. En d'autres termes, b préfère son conjoint dans $\mathcal{M}_1$ à son conjoint dans $\mathcal{M}_2$ (s'ils sont distincts). L'élément b étant quelconque, cela établit bien $\mathcal{M}_2 \preceq^B \mathcal{M}_1$.

Question 6. Il convient d'abord de montrer que $\mathcal{M}_1 \vee \mathcal{M}_2$ est bien un mariage, c'est-à-dire qu'on n'a pas une situation où deux éléments distincts a_1 et a_2 se retrouvent mariés à un même b dans $\mathcal{M}_1 \vee \mathcal{M}_2$. Supposons que cela arrive. Sans perte de généralité, on va supposer que a_1 était marié à b dans $\mathcal{M}_1$ et a_2 à b dans $\mathcal{M}_2$. Toujours sans perte de généralité, supposons que b préfère a_1 à a_2. Alors on a une instabilité dans le mariage $\mathcal{M}_2$: b préfère a_1 à son conjoint a_2, et a_1 préfère b à son conjoint puisque par définition de $\mathcal{M}_1 \vee \mathcal{M}_2$, b est le conjoint préféré de a entre les deux mariages.

Montrons maintenant que le mariage est stable. Si on constate une instabilité (a_1, b_1, a_2, b_2) dans $\mathcal{M}_1 \vee \mathcal{M}_2$, cette instabilité n'existant pas dans $\mathcal{M}_1$ ou $\mathcal{M}_2$ (ils sont supposés stables), cela veut dire que les paires (a_1, b_2) et (a_2, b_1) existent dans deux mariages différents, disons $\mathcal{M}_1$ et $\mathcal{M}_2$ respectivement. Dans $\mathcal{M}_2$, b_1 est marié à a_2 mais préfère a_1 (par l'instabilité ci-dessus). Par stabilité, a_1 préfère son conjoint dans $\mathcal{M}_2$, appelons-le b_3, à b_1. Mais par définition de $\mathcal{M}_1 \vee \mathcal{M}_2$, a_1 préfère b_2 à b_3 et donc à b_1. Cela contredit l'instabilité de l'hypothèse.

Question 7. Soit $\mathcal{E}$ l'ensemble des mariages stables. L'opération $\vee$ définie à la question précédente est associative et $\mathcal{M}_1 \vee \mathcal{M}_2$ est supérieur ou égal à $\mathcal{M}_1$ et $\mathcal{M}_2$ pour l'ordre $\preceq^A$. L'élément $\bigvee_{\mathcal{M} \in \mathcal{E}} \mathcal{M}$ est donc l'élément maximum de $\mathcal{E}$.

Pour le minimum, il suffit de remarquer que A et B jouent des rôles symétriques. On a donc un élément $\mathcal{M}$ qui est maximum pour l'ordre $\preceq^B$, qui est donc minimum pour l'ordre $\preceq^A$ (pour ceci on doit remarquer que la conclusion de la Question 5 est en fait un « si et seulement si », par symétrie).

Question 8. En utilisant l'indication : considérons l'instant t où pour la première fois, un élément $a \in A$ soit rejeté par un partenaire valide b, qui lui préfère alors a'. Par validité, considérons un mariage stable $\mathcal{M}$ où a est marié à b. Dans ce mariage, appelons b' le conjoint de a'. Comme b préfère a' à a, par stabilité de $\mathcal{M}$, a' préfère b' à b. Donc dans Gale–Shapley, a' a proposé à b' (qui l'a rejeté) avant de proposer à b. Mais par définition de l'instant t, le rejet de a' par b' a eu lieu après l'instant t, ce qui veut dire qu'à l'instant t, a' n'avait pas encore proposé à b. C'est une contradiction car à l'instant t, b se marie avec a'.

On a donc établi que tout a se retrouve marié au plus désirable des partenaires valides de sa liste, ce qui exprime bien que l'algorithme retourne le mariage stable $\preceq^A$-maximal.