Banques MP et MPI inter-ENS – Session 2025
Rapport relatif à l'épreuve orale d'informatique

— Épreuve commune à toutes les ENS
— Coefficients (en pourcentage du total des points de chaque concours) :

École	MP - Info	MPI - Math	MPI - Info	Info
Paris	15%	16,7%	22,2%	N.A.
Lyon	10,8%	10,4%	14,1%	14,1%
Paris-Saclay	23,1%	12,8%	12,8%	N.A.
Rennes	23,1%	N.A.	13,9%	N.A.

— Membres du jury :
  — Alexandre Debant
  — Romain Demangeon
  — Jérémy Dubut
  — Brice Minaud
  — Hugo Paquet
  — André Schrottenloher
  — Pierre Senellart

L'épreuve orale d'informatique fondamentale décrite dans ce rapport est commune à toutes les Écoles normales supérieures.

Cette année, le jury a interrogé :

— 104 candidat·e·s pour la filière MP. Les notes données s'échelonnent entre 4.8 et 20, avec une médiane à 12.6, une moyenne à 12.65 et un écart-type de 2.84. La figure 1 présente l'histogramme complet des notes.

— 149 candidat·e·s pour la filière MPI. Les notes données s'échelonnent entre 5 et 20, avec une médiane à 14, une moyenne à 13.37 et un écart-type de 3.15. (La différence de moyenne avec la filière MP est dûe davantage à une harmonisation avec des options différentes, qu'à une différence de niveau.) La figure 2 présente l'histogramme complet des notes.

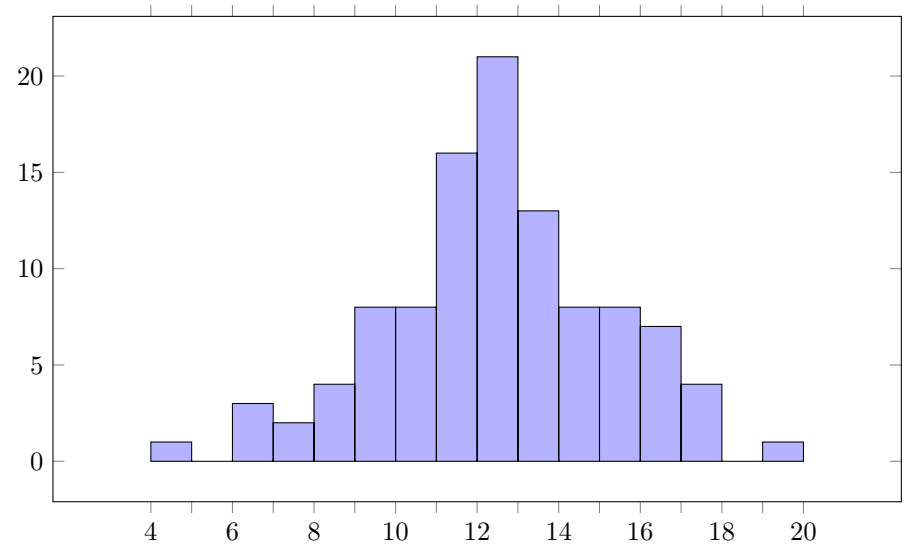


FIGURE 1 – Histogramme des notes de l'épreuve (filière MP). La colonne positionnée entre x et $x+1$ comptabilise les notes comprises entre x exclus et $x+1$ inclus.

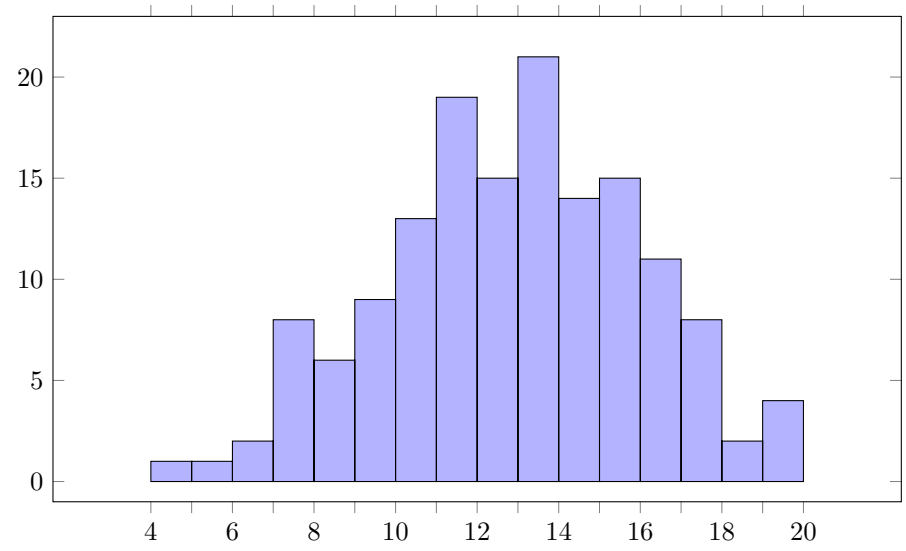


FIGURE 2 – Histogramme des notes de l'épreuve (filière MPI). La colonne positionnée entre x et $x+1$ comptabilise les notes comprises entre x exclu et $x+1$ inclus.

Après avoir reçu un sujet, les candidat·e·s disposent de 30 minutes de préparation, suivies de 28 minutes d'interrogation devant un des examinateurs. En effet, deux minutes sont utilisées par l'examinateur pour aller chercher la/le candidat·e en salle de préparation. Nous rappelons également que jusqu'à quatre candidat·e·s peuvent passer l'épreuve à la suite sur le même sujet, auquel cas la première/le premier d'entre eux est invité·e à patienter 30 minutes dans la salle de préparation à l'issue de son oral, afin de garantir la confidentialité du sujet.

Le jury a proposé 20 sujets originaux (10 pour le concours MP et 10 pour le concours MPI), dont la liste est donnée en annexe de ce document. Entre 12 et 16 candidat·e·s en moyenne ont été interrogé·e·s sur chaque sujet.

Comme il a été rappelé en préambule des sujets distribués :

> Le but de cette épreuve est d'évaluer la progression des candidates et candidats dans les questions, mais aussi la qualité de leur exposé des solutions, ainsi que l'autonomie dont elles ou ils font preuve pendant l'oral.

Chaque sujet débute par un énoncé qui présente un problème d'informatique et introduit ses notations, puis comporte des questions de difficulté globalement croissante. Ainsi, les premières questions visent à aider à la compréhension du sujet, et permettent aux candidat·e·s de s'approprier les définitions qui ont été données. Les dernières questions d'un sujet sont souvent des questions d'ouverture plus difficiles. Pour la plupart des sujets donnés, il n'est pas attendu que les candidat·e·s traitent l'intégralité des questions.

Les sujets proposés portent sur des thèmes variés de science informatique : langages, graphes, logique, *etc.*, en lien avec les programmes respectifs des filières MP et MPI. Ils nécessitent de s'approprier des concepts nouveaux, démontrer des résultats théoriques et construire des solutions techniques telles que des algorithmes. Bien que l'épreuve mette l'accent sur les concepts théoriques de l'informatique, on veille à garder à l'esprit le sens des objets que l'on étudie, et un certain sens pratique (par exemple dans la conception d'algorithmes ou l'estimation asymptotique de leur complexité) est apprécié.

Le jury tient à féliciter les candidates et candidats qui ont pour la plupart démontré des acquis très solides, et fait preuve d'idées excellentes malgré les contraintes de temps inhérentes à l'épreuve. Même celles et ceux ayant obtenu de moins bonnes notes ont montré des qualités certaines. En particulier, la plupart des candidat·e·s affichent une bonne maîtrise des concepts mathématiques essentiels (induction, disjonction de cas) ainsi qu'une bonne initiative (par exemple le fait de tester un algorithme sur un petit exemple avant d'en déduire le cas général).

Il est usuel que le jury engage une discussion sur certaines questions afin de guider les candidat·e·s dans leur résolution du problème. Une attitude constructive et positive est appréciée lors de ces échanges.

Le jury adresse les conseils suivants aux futur·e·s candidat·e·s.

Concours et autres voies d'accès. Les Écoles normales supérieures sont destinées notamment aux personnes intéressées par la recherche ou l'enseignement. Il peut être utile pour les candidat·e·s de connaître les différentes voies d'accès à ces écoles, présentées sur la page suivante :

`https://diplome.di.ens.fr/informatique-ens/`

Gestion du temps et de l'espace. Répéter ou recopier l'énoncé lors de l'oral est une perte de temps, l'examinateur ayant lui aussi le sujet sous les yeux. Le jury déconseille aux candidat·e·s l'effacement du tableau à la main, qui le rend rapidement illisible pour l'examinateur.

Raisonnements. Afin de démontrer leur maîtrise du programme d'informatique de leur filière, mais aussi leur compréhension des nouveaux concepts introduits par le sujet traité, le jury invite les candidat·e·s à s'assurer de la cohérence de ce qu'ils décrivent.

Par exemple, si le jury demande un identifiant de variable OCaml, celui-ci ne peut pas, par définition, être un mot clé du langage, ou une application de fonction. Des erreurs d'attention ou de compréhension peuvent arriver, mais la répétition de ces erreurs de syntaxe ou de typage donne l'impression très négative que la/le candidat·e n'a pas compris le sujet.

De manière générale, le jury conseille aux candidat·e·s d'utiliser les raisonnements par l'absurde uniquement lorsque ceux-ci sont nécessaires, et de préférer les inductions structurelles (par exemple sur un arbre) aux récurrences sur un critère numérique (hauteur de l'arbre). Celles-ci sont usuellement plus efficaces et / ou plus adaptées aux objets étudiés.

Pédagogie, intuition, formalisme. Si la situation le permet, il est souhaitable de décrire (en général oralement ou/et par un dessin) une preuve dans les grandes lignes avant de se lancer dans la preuve formelle. Cela permet à l'examinateur de s'assurer que la/le candidat·e a compris le principe de la preuve. De même, il est souhaitable de décrire un algorithme dans les grandes lignes avant de se lancer dans l'écriture du pseudo-code.

Si une question attend une réponse oui/non, il est conseillé d'annoncer cette réponse avant de se lancer dans la preuve. De même, si une question contient plusieurs résultats à prouver, annoncer lequel sera prouvé en premier, *etc.*

Certains sujets demandent d'absorber plusieurs notions nouvelles, parfois au moyen d'un formalisme assez lourd. On attend alors des candidat·e·s une capacité à s'affranchir du formalisme pour se concentrer sur la signification des objets. Cependant, si l'examinateur demande des précisions, la/le candidat·e doit être en mesure de justifier formellement son raisonnement.

Langage OCaml, programmation fonctionnelle. Lorsqu'une fonction OCaml doit être décrite, le jury suggère fortement aux candidat·e·s de réfléchir en premier abord à la signature de type de la fonction, qui leur permettra de clarifier leurs idées avant de proposer une implémentation.

Pour la filière MPI, le jury a remarqué que les candidat·e·s voyaient difficilement le parallèle entre variables libres en logique et dans des expressions OCaml.

Autres remarques.

- Les sujets comportent différents types de difficulté. Certains sujets, comme par exemple la plupart de ceux qui portent sur les graphes, présentent des variantes d'objets familiers, mais peuvent poser sur ces objets des questions difficiles. D'autres sujets introduisent au contraire des objets nouveaux, parfois abstraits, que les candidat·e·s doivent d'abord assimiler, et sur lesquels ils doivent se former rapidement une intuition ; quitte à ce que le sujet commence parfois par des questions plus simples, pour aider à cette assimilation. Le jury a eu l'impression que les candidat·e·s sont généralement plus performants sur le premier type de difficulté que sur le second.
- Le jury a été surpris qu'un certain nombre de candidat·e·s, tout en sachant bien répondre sur d'autres questions, ne semblait pas maîtriser l'algorithme du pivot de Gauss, ni arriver à justifier sa complexité.
- De même, la notion de graphe de flot de contrôle, mentionnée dans le programme de l'option informatique, ne semblait pas connue des candidat·e·s.
- Certain·e·s candidat·e·s ont eu des difficultés à écrire précisément des arguments simples comme des récurrences ou raisonnements par l'absurde, en utilisant correctement les quantificateurs, et en distinguant les implications des équivalences.

Annexe : sujets proposés pour la filière MP

Certains sujets sont accompagnés d'*ébauches* de solution.

Tri externe polyphasé

Nous nous intéressons au problème suivant. Imaginons que nous voulions trier le contenu d'un fichier dont la taille dépasse très largement la taille de la mémoire vive (où peuvent s'effectuer rapidement les opérations de tri), mais que nous ayons accès à de la mémoire externe (ici des bandes magnétiques) pour stocker les résultats des opérations de tri (partiels et finaux). Dans ce contexte, les coûts des opérations sur la mémoire vive sont négligeables par rapport à celles de lecture, d'écriture et de déplacements de la tête de lecture sur la mémoire externe.

Nous formalisons le problème de la manière suivante. Supposons que nous voulons trier N éléments. Supposons que nous avons accès à T mémoires externes supposées infinies. Chaque mémoire externe a une tête de lecture qui pointe sur une case donnée de la mémoire. Trois opérations peuvent être effectuées sur cette mémoire :

1. écrire un élément de la mémoire vive sur la case pointée par la tête de lecture,
2. lire l'élément pointé par la tête de lecture pour le mettre sur la mémoire vive et l'enlever de la mémoire externe,
3. déplacer la tête de lecture d'une case vers la gauche ou vers la droite.

Lorsque nous disons qu'une mémoire est vide, nous entendons que chacune de ses cases sont vides. Nous représenterons une case vide par le symbole $\varnothing$.
Sur la mémoire vive, nous ne pouvons avoir qu'au plus K éléments simultanément. Sur la liste des éléments sur la mémoire vive, il est possible d'appliquer tout algorithme par exemple pour la trier, y trouver l'élément maximal, comparer des éléments, etc... En plus des éléments de la liste à trier, la mémoire vive peut contenir des variables qui ne comptent pas dans le compte des K éléments, et qui peuvent être utilisés dans les algorithmes à écrire. Toutes les opérations sur la mémoire vive n'entreront pas dans le calcul de la complexité des algorithmes à écrire.
Le but est d'écrire la liste triée des N éléments sur l'une des mémoires externes.

Question 1. Supposons $T = 3$, $N = 6$ et $K = 3$. Montrez comment passer de la configuration :

$$1 : \text{vide}$$
$$2 : \text{vide}$$
$$3 : 6; 2; 4; 5; 3; 1$$

à la configuration :

$$1 : \text{vide}$$
$$2 : \text{vide}$$
$$3 : 1; 2; 3; 4; 5; 6$$

où dans les deux cas, les têtes de lecture sont toutes sur les premières cases des mémoires externes. Nous ne chercherons pas une solution optimale, mais nous attendrons un calcul précis du nombre d'opérations sur les mémoires externes.

Question 2. Supposons que $T \geq 3$ et que N est un multiple de K, disons $N = Kp$. Fixons $q \leq p$. Supposons qu'initialement nous sommes dans la configuration suivante :

$$1 : \text{vide}$$
$$2 : \text{vide}$$
$$3 : L_0$$

où L_0 est la liste non triée des N éléments et où les têtes de lecture sont toutes sur la première case de chaque mémoire. Décrire un algorithme en $6N$ opérations (écriture, lecture et déplacement) pour obtenir la configuration suivante :

$$1 : B_1; B_2; \ldots; B_q$$
$$2 : B_{q+1}; B_{q+2}; \ldots; B_{N/K}$$
$$3 : \text{vide}$$

où B_1, …, $B_{N/K}$ sont des blocs triés de K éléments et où les têtes de lecture sont sur les premières cases de chaque mémoire.

Solution : Nous commençons par créer une variable X initialisée à 1 sur la mémoire vive. Puis, nous répétons les étapes suivantes tant que la mémoire 3 n'est pas vide :

1. Nous effectuons K fois les opérations suivantes sur la mémoire 3 :
   a) lire l'élément pointée par la tête de lecture,
   b) déplacer la tête de lecture vers la droite.
2. Nous trions les K éléments présents sur la mémoire vive.
3. Pour chacun des K éléments dans l'ordre de la liste triée, nous effectuons les opérations suivantes, soit sur la mémoire 1 si $X \leq q$, soit sur la mémoire 2 sinon.
   a) écrire l'élément sur la mémoire 1 ou 2,
   b) déplacer la tête de lecture vers la droite.
4. ajouter 1 à X.
5. effacer tous les éléments sur la mémoire vive

Nous finissons par déplacer les trois têtes de lecture vers la gauche jusqu'à arriver à la première case de chaque mémoire externe.

Comptons le nombre d'opérations sur la mémoire externe :

1.a) N lectures
1.b) N déplacements
3.a) N écritures
3.b) N déplacements
+ $2N$ déplacements pour remettre les têtes de lecture.

Soit un total de $6N$ opérations.

Question 3. Supposons que $T = 4$ et que N/K soit une puissance de 2, disons $N = K2^n$ avec $n > 0$. En partant de la configuration finale de la question précédente avec $q = N/2K = 2^{n-1}$, écrivez un algorithme en $6nN$ opérations afin d'obtenir la configuration suivante (ou toute autre configuration à permutation des mémoires externes près) :

$$1 : L$$
$$2 : \text{vide}$$
$$3 : \text{vide}$$
$$4 : \text{vide}$$

où L est la liste triée de tous les éléments et toutes les têtes de lecture pointent vers la première case.

Solution : Nous allons effectuer $6N$ opérations pour aller d'une configuration :

$$\begin{aligned}
&1 : B_1^i; B_2^i; \ldots; B_{2^{n-i}}^i \\
&2 : B_{2^{n-i}+1}^i; B_{2^{n-i}+2}^i; \ldots; B_{2^{n+1-i}}^i \\
&3 : \text{vide} \\
&4 : \text{vide}
\end{aligned}$$

vers une configuration :

$$\begin{aligned}
&1 : \text{vide} \\
&2 : \text{vide} \\
&3 : B_1^{i+1}; B_2^{i+1}; \ldots; B_{2^{n-i-1}}^{i+1} \\
&4 : B_{2^{n-i-1}+1}^{i+1}; B_{2^{n-i-1}+2}^{i+1}; \ldots; B_{2^{n-i}}^{i+1}
\end{aligned}$$

où les B_j^k sont des blocs triés de $K2^{k-1}$ éléments et les têtes de lectures sont sur les premières cases. Observer par exemple que la configuration initiale de cette question correspond à la première configuration avec $i = 1$, et la configuration finale correspond à la seconde configuration avec $i = n + 1$ (à permutation près). Pour résoudre la question, il sera donc suffisant de répéter n fois ces opérations, soit un total de $6nN$ opérations comme attendu.

Nous allons décrire ces opérations par blocs, c'est-à-dire, que nous allons décrire une suite de $4K \cdot 2^i$ opérations pour aller de la configuration

$$\begin{aligned}
&1 : \varnothing^{ijK}; B_{j+1}^i; \ldots; B_{2^{n-i}}^i \\
&2 : \varnothing^{ijK}; B_{2^{n-i}+j+1}^i; \ldots; B_{2^{n+1-i}}^i \\
&3 : L \\
&4 : L'
\end{aligned}$$

où L et L' sont des suites quelconques d'éléments, et les têtes de lectures pointent sur les premières cases non vides des mémoires 1 et 2 et sur les premières cases vides des mémoires 3 et 4, vers une configuration :

$$\begin{aligned}
&1 : \varnothing^{i(j+1)K}; B_{j+2}^i; \ldots; B_{2^{n-i}}^i \\
&2 : \varnothing^{i(j+1)K}; B_{2^{n-i}+j+2}^i; \ldots; B_{2^{n+1-i}}^i \\
&3 : L; B_{j+1}^{i+1} \\
&4 : L'
\end{aligned}$$

où B_{j+1}^{i+1} est la liste triée obtenue en triant l'union des listes B_{j+1}^i et $B_{2^{n-i}+j+1}^i$ et où les têtes des lectures sont placées de la même façon sur les premières case non vides (resp. vides) des mémoires 1 et 2 (resp. 3 et 4). Pour résoudre le problème précédent, il suffit de répéter ces opérations 2^{n-i} fois (à permutation près, soit $4N$ opérations au total), puis de faire revenir toutes les têtes de lecture vers la première case ($2N$ déplacements au total). Soit un total de $6N$ opérations comme attendu.

Ces opérations consistent à effectuer une fusion comme dans le tri fusion classique. Précisément, on commence par créer deux variables entières X_1 et X_2 initialisées à 1. Nous lisons les mémoires 1 et 2 et déplaçons leurs tête de lecture vers la droite. Nous répétons les opérations suivantes tant que $X_1 < K2^{i-1}$ ou $X_2 < K2^{i-1}$:

1. Deux cas possibles :
   (a) il n'y a qu'un élément sur la mémoire vive, alors on écrit cet élement sur la mémoire 3
   (b) il y a deux élements sur la mémoire vive, alors on écrit le plus grand élément sur la mémoire 3
2. On incrémente de 1 la variable X_l où l est la mémoire de laquelle provient l'élément que l'on vient d'écrire.
3. On déplace la tête de lecture de la mémoire 3 vers la droite.
4. Si $X_l < K2^{i-1}$, on lit la mémoire l et bouge sa tête de lecture vers la droite.

Chaque itération demande :

1. 1 écriture
2. 0 opération
3. 1 déplacement
4. 1 écriture et 1 déplacement, sauf sur 2 itération (lorsque les deux listes deviennent vides), mais qui sont compensées par les opérations effectuées initialement.

Nous répétons ces itérations $K2^i$ fois, soit un total de $4K \cdot 2^i$ opérations.

Nous avons donc trouvé un premier algorithme de tri externe en $O(N \log_2(N/K))$ opérations sur les mémoires externes dans le cas où $T = 4$. Cet algorithme peut être adapté pour $T = 2p$ en $O(N \log_p(N/K))$. Nous nous intéressons maintenant au cas $T = 3$.

Question 4. En partant de la même configuration initiale que la question 3, adaptez le cas $T = 4$ de l'algorithme au cas $T = 3$ en effectuant $9nN$ opérations.

Solution : Nous pouvons adapter l'algorithme pour passer de la configuration :

$$\begin{aligned} &1 : B_1^i; B_2^i; \ldots; B_{2^{n-i}}^i \\ &2 : B_{2^{n-i}+1}^i; B_{2^{n-i}+2}^i; \ldots; B_{2^{n+1-i}}^i \\ &3 : \text{vide} \end{aligned}$$

vers une configuration :

$$\begin{aligned} &1 : \text{vide} \\ &2 : \text{vide} \\ &3 : B_1^{i+1}; B_2^{i+1}; \ldots; B_{2^{n-i}}^{i+1} \end{aligned}$$

avec les changements suivants :

— au lieu d'écrire les blocs triés alternativement sur les mémoires 3 et 4, nous les écrivons tous sur la mémoire 3

— au lieu de faire revenir la tête de lecture de la mémoire 3 vers la première case, nous la faisons revenir au milieu de la mémoire, c'est-à-dire, sur la première case de $B_{2^{n-i-1}+1}^{i+1}$

Tout ceci se fait en $5N + N/2$ opérations. Il faut alors effectuer $3N + N/2$ opérations pour obtenir la configuration :

$$\begin{aligned} &1 : \text{vide} \\ &2 : B_{2^{n-i-1}+1}^{i+1}; \ldots; B_{2^{n-i}}^{i+1} \\ &3 : B_1^{i+1}; \ldots; B_{2^{n-i-1}}^{i+1} \end{aligned}$$

qui consiste à répéter $N/2$ fois les opérations suivantes :

1. lire la mémoire 3
2. écrire cet élément sur la mémoire 2
3. déplacer les têtes de lecture des mémoires 2 et 3 vers la droite.

soit un total de $4 \cdot N/2 = 2N$ opérations, puis de faire revenir les têtes de lecture des mémoires 2 et 3 vers la première case, soit $N + N/2$ déplacements. Au total, cela requiert $9N$ opérations. En répétant, ces opérations n fois comme dans la question précédente (à permutation près), nous obtenons un algorithme en $9nN$ opérations.

Le problème du cas $T = 3$ avec une configuration initiale équilibrée est qu'il est nécessaire de copier une partie du contenu d'une mémoire sur une autre mémoire sans réellement progresser dans la construction de la liste triée.

Question 5. En partant d'une autre configuration initiale obtenue en question 2 bien choisie, adaptez l'algorithme de la question 3 de façon à ne faire *aucune* de ces copies de mémoire. Vous pourrez supposer que N est de la forme $N = KF_n$ pour une suite F_n bien choisie, dans quel cas vous pourriez montrer un algorithme en

$$3N + 6K \sum_{i=1}^{n-1} F_{n-i}F_i$$

opérations.

Solution : Le problème de la question 3 est que lorsque les mémoires 1 et 2 sont fusionnées, elles sont complètement vidées au même moment. L'idée est de faire en sorte que après la fusion des mémoires 1 et 2, uniquement la mémoire 1 est vide et la configuration de la mémoire 2 et 3 soit similaire à la configuration des mémoires 1 et 2 avant la fusion. L'idée est donc de définir une suite croissante $(F_n)_{n\in\mathbb{N}}$ telle que si $(F_{n-1}, F_n, 0)$ est la configuration avant fusion (c'est-à-dire, avec F_{n-1} blocks triés sur la mémoire 1, F_n sur la mémoire 2, et 0 sur la blocs 3), la configuration après fusion soit $(F_{n-2}, F_{n-1}, 0)$ à permutation près, ce qui permettrait d'itérer le processus de fusion. Si on suppose la suite croissante, après la fusion, la configuration est de la forme $(0, F_n - F_{n-1}, F_{n-1})$. Donc ce principe fonctionnerait si $F_n = F_{n-1} + F_{n-2}$. De plus, à la fin de l'algorithme, nous souhaiterions une configuration $(1, 0, 0)$ à permutation près, donc que $F_0 = 0$ et $F_1 = 1$. Au total, la bonne suite à choisir est la suite de Fibonacci !

L'algorithme fonctionnerait de la façon suivante en supposant $N = KF_n$ avec $n > 1$. On applique la question 1 avec $q = F_{n-2}$ pour obtenir la configuration :

$$\begin{aligned} &1 : B_1; B_2; \ldots; B_{F_{n-2}} \\ &2 : B'_1; B'_2; \ldots; B'_{F_{n-1}} \\ &3 : \text{vide} \end{aligned}$$

où tous les blocks sont de taille K et les têtes de lecture sont sur la première case. Après une phase de fusion des mémoires 1 et 2 et écriture sur la mémoire 3, similaire à la question 2, nous nous retrouvons sur une configuration de la forme :

$$\begin{aligned} &1 : \text{vide} \\ &2 : \varnothing^{KF_{n-2}}; B_1; B_2; \ldots; B_{F_{n-3}} \\ &3 : B'_1; B'_2; \ldots; B'_{F_{n-2}} \end{aligned}$$

où les blocks de la mémoire 2 sont de taille K et ceux de la mémoire 3 sont de taille $2K$. On fera en sorte que les têtes de lecture des mémoire 1 et 3 soient sur la première case alors que celle de la mémoire 2 est sur la première case de B_1. En analysant plus précisément le nombre d'opérations, il est nécessaire de faire :

- $2F_{n-2}$ lectures sur les mémoires 1 et 2
- $2F_{n-2}$ écritures sur la mémoire 3
- $4F_{n-2}$ déplacements vers la droite sur toutes les mémoires
- $3F_{n-2}$ déplacements vers la gauche sur les mémoires 1 et 3.

Soit un total de $11F_{n-2}$ opérations. Définissons C_i pour $1 \leq i < n$ comme (à permutation près) :

$$\begin{aligned} &1 : B_1; B_2; \ldots; B_{F_{n-i}} \\ &2 : \varnothing^{KF_{i-1}F_{n-i}}; B'_1; B'_2; \ldots; B'_{F_{n-(i+1)}} \\ &3 : \text{vide} \end{aligned}$$

où les blocks B_i sont de tailles KF_i et les blocks B'_i sont de taille KF_{i-1} et où les têtes de lecture sont sur les premières case des mémoires 1 et 3 et sur la première case du block B'_1 de la mémoire 2. Observez en particulier que la seconde configuration ci-dessus correspond à C_2.

En fusionnant les mémoires 1 et 2, et écrivant sur la mémoire 3, nous pouvons aller de la configuration C_i à la configuration C_{i+1}. Cela demande :

- $KF_{n-(i+1)}F_i$ lectures sur la mémoire 1
- $KF_{n-(i+1)}F_i$ déplacements vers la droite sur la mémoire 1
- $KF_{n-(i+1)}F_{i-1}$ lectures sur la mémoire 2
- $KF_{n-(i+1)}F_{i-1}$ déplacements vers la droite sur la mémoire 2
- $KF_{n-(i-1)}F_{i-1}$ déplacements vers la gauche sur la mémoire 2
- $KF_{n-(i+1)}F_{i+1}$ écritures sur la mémoire 3
- $KF_{n-(i+1)}F_{i+1}$ déplacements vers la droite sur la mémoire 3
- $KF_{n-(i+1)}F_{i+1}$ déplacements vers la gauche sur la mémoire 3

Observez en particulier que fusionner un block de la mémoire 1 (de taille KF_i) avec un block de la mémoire 2 (de taille KF_{i-1}) produit bien un block de taille KF_{i+1} comme souhaité.

Finalement, la configuration C_{n-1} est presque la configuration finale, sauf que la tête de lecture de la mémoire 2 n'est pas sur la première case. Il est alors nécessaire de faire KF_{n-2} déplacements vers la gauche.

Au lieu de faire la somme de toutes ces opérations, nous pouvons observer que ces opérations consistent à opérer sur des mémoires de la forme :

$$B_1; B_2; \ldots; B_{F_{n-i}}$$

où les blocks B_i sont de taille KF_i. Sur ces mémoires, nous effectuons :

- au moment de l'écriture de cette mémoire :
  - $KF_{n-i}F_i$ écritures
  - $KF_{n-i}F_i$ déplacements vers la droite
  - $KF_{n-i}F_i$ déplacements vers la gauche
- au moment de la fusion avec d'autres mémoires, qui se fait sur deux phases :
  - $KF_{n-i}F_i$ lectures
  - $KF_{n-i}F_i$ déplacements vers la droite
  - $KF_{n-i}F_i$ déplacements vers la gauche (qui se fait uniquement lors de la deuxième phase).

Soit un total de $6KF_{n-i}F_i$ opérations. A cela s'ajoute la lecture de la mémoire 3 lors de la phase d'initialisation pour un total de $3N$ opérations. Comme pour un i tel que $1 \leq i < n$, il y a une unique telle mémoire au cours de l'algorithme, cela fait un total de :

$$3N + 6K \sum_{i=1}^{n-1} F_{n-i}F_i.$$

Types à Gabarit

On se place dans un langage de programmation minimal, fonctionnel pur, **typé** polymorphe qui contient

- des noms, des définitions de fonctions (anonymes et nommées),
- un opérateur de définition récursive d'expressions,
- des définitions récursives de types inductifs :
  $\texttt{data D}\ t_1\ \ldots\ t_p = \texttt{C}_1\ T_1\ \mid\ \ldots\ \mid\ \texttt{C}_\texttt{k}\ T_k$
  où D est le constructeur de types que l'on définit, t_i des variables de types, T_i des types (contenant potentiellement le constructeur D et les variables t_j), et $\texttt{C}_i$ des constructeurs de termes,
- un opérateur reconnaissance de motif permettant de décomposer les éléments d'un type inductif.

On définit les types inductifs Nat et Liste ainsi :
$\texttt{data Nat} = \texttt{Z} \mid \texttt{S}\ (\texttt{Nat})$.
$\texttt{data Liste}\ A\ = \texttt{Nil}\ \mid\ \texttt{Cons}\ A\ (\texttt{Liste}\ A)$
ainsi on peut définir $\texttt{longueur} : \texttt{Liste}\ A \to \texttt{Nat}$ avec
$\texttt{letrec longueur}\ l\ = \texttt{decomposer}\ l\ :$
  $\texttt{cas Nil} = \texttt{Z}$
  $\texttt{cas}\ (\texttt{Cons}\ t\ q)\ = \texttt{S}\ (\texttt{longueur}\ q)$
On note $A_1 \to A_2 \to \cdots \to A_n \to B$ le type d'une fonction n-aire qui prend, dans l'ordre, une entrée de type A_1, …, une entrée de type A_n et qui produit un résultat de type B.
On identifiera un type avec l'ensemble de ses habitants (des entités qui ont ce type).

On note $E[X/Y]$ l'expression syntaxique E dans lequel toutes les occurrences de la sous expression Y sont remplacées par l'expression X.
Si le type inductif D est défini par $\texttt{data D}\ t_1\ \ldots\ t_p = \texttt{C}_1\ T_1\ \mid\ \ldots\ \mid\ \texttt{C}_\texttt{k}\ T_k$
alors, pour $n \in \mathbb{N}$, $\texttt{D}_n$ est la *n-approximation de* D, définie par :
$\texttt{D}_0\ t_1\ \ldots\ t_p = \emptyset$
$\texttt{D}_{n+1}\ t_1\ \ldots\ t_p = \{\texttt{C}_1\ x \mid x \in T'_1\} \cup\ \ldots\ \cup\ \{\texttt{C}_\texttt{k}\ x \mid x \in T'_k\}$
  avec $T'_j = T_j[\texttt{D}_n/\texttt{D}]$ pour tout j

Question 1. Décrire $\texttt{Nat}_n$ et $\texttt{Liste}_n\ A$.

Soit $\mathcal{S}$ un ensemble de *variables de taille*, distinctes des noms du langage.
Un *gabarit* S est donné par la grammaire :
$S ::= s \mid n \mid S + S \mid n \times S$ avec $n \in \mathbb{N}$ et $s \in \mathcal{S}$.
On ajoute au langage des types les annotations de gabarits pour les types inductifs, ainsi, si $\texttt{D}\ t_1\ \ldots\ t_p$ est un type, alors $\texttt{D}_S\ T_1\ \ldots\ t_p$ est un type, défini comme la S-approximation de D.

Question 2. Donner des définition et des types avec gabarits pour des fonctions calculant la somme de deux entiers, leur produit et le quotient de la division euclidienne du premier par le deuxième.

Question 3. On suppose qu'on a une fonction $\texttt{moyenne} : \texttt{Nat}_{2i} \to \texttt{Nat}_{2j} \to \texttt{Nat}_{i+j}$

Donner la définition et le type avec gabarits d'une fonction lisse qui prend en entrée une liste d'entiers et renvoie la liste des moyennes des éléments consécutifs de cette liste.

Question 4. Donner une condition utilisant les types à gabarits pour qu'une fonction définie récursivement soit terminante.

> On ajoute au langage une définition de types *coinductifs* :
> `codata` $D\ t_1\ \ldots\ t_p = \mathtt{C_1}\ T_1\ \mid\ \ldots\ \mid\ \mathtt{C_k}\ T_k$.
> dans ce cas, $D\ t_1\ \ldots\ t_p$ est le plus grand type (pour l'inclusion d'ensembles) qui vérifie l'équation
> $T = \{\mathtt{C_1}\ x \mid x \in T'_1\} \cup \ldots \cup \{\mathtt{C_k}\ x \mid x \in T'_k\}$ avec $T'_j = T_j[T/(D\ t_1\ \ldots\ t_p)]$ pour tout j

Question 5. Décrire les types suivants :

1. `codata Flot` $A =$ `FCons` A (`Flot` A).
2. `codata INat` $=$ `Z` $\mid$ `S INat`.
3. `codata IListe` $A\ =$ `Nil` $\mid$ `Cons` A (`IListe` A)
4. `data FFlot` $A =$ `FCons` A (`FFlot` A).

Question 6. Si $\mathcal{C}$ est l'ensemble des constructeurs du langage, $\mathcal{X}$ l'ensemble des noms du langage, on définit l'univers $\mathbb{U}$ comme unique solution de l'équation

$U = \mathcal{C} \cup \mathcal{X} \cup (U \to U) \cup (U \times U)$

Décrire $\mathbb{U}$.

Question 7. Soit D défini par `codata` $D\ t_1\ \ldots\ t_p = \mathtt{C_1}\ T_1\ \mid\ \ldots\ \mid\ \mathtt{C_k}\ T_k$.

Définir D^n sa n-approximation. Puis décrire les n-approximations des types de la Question 5.

Question 8. Définir `tete` et `queue`, deux fonctions permettant de récupérer la tête et la queue d'un flot, et leur donner des types avec gabarits.

Question 9. On définit ces objets du langage :

`letrec zeros` $=$ `FCons Z zeros`

`letrec zeros'` $=$ `FCons Z` (`queue zeros'`)

`letrec filtre` (`FCons` n_1 (`FCons` n_2 q)) $=$ `FCons` (`moyenne` n_1 n_2) (`filtre` q)

`letrec fadd` (`FCons` n_1 q_1) (`FCons` n_2 q_2) $=$ `FCons` (`somme` n_1 n_2) (`fadd` q_1 q_2)

$fs =$ `fadd` (`filtre` s) (`filtre` (`filtre` s))

Sont-ils utilisables ?

En induire une méthode pour définir proprement des objets manipulant des types co-inductifs.

Question 10. Trouver une condition sur les définitions de types pour que $lim_{n\to\infty} D_n = D$
et $lim_{n\to\infty} D^n = D$.

Déclasser dans les Douze Chemins

On étudie, compte et range des classes d'équivalence de fonctions d'un ensemble fini N à n éléments dans un ensemble fini X à k éléments.
On s'intéresse à quatre relations d'équivalence pour les fonctions :

- l'égalité, notée DD,
- l'égalité à permutation des éléments de N près, notée UD,
- l'égalité à permutation des éléments de X près, notée DU,
- l'égalité à permutation des éléments de N et de X près, notée UU.

Lorsqu'on a une égalité à permutation près sur un ensemble, on peut considérer que les éléments de cet ensemble sont "indistinguables" (**U***ndistinguished*) pour la comparaison des fonctions, ce qui explique les notations U et D.
En outre, on s'intéresse à trois caractéristiques des fonctions qu'on étudie :

- les fonctions *arbitraires* (notées A),
- les fonctions *injectives* (notées I),
- les fonctions *surjectives* (notées S).

Cela définit 12 (3×4) problèmes de dénombrement, appelés *douze chemins* (*twelvefold way*), ou *douze problèmes de Gian-Carlo Rota*. Par exemple, le problème noté IDU consiste à dénombrer et décrire les classes d'équivalence de fonctions injectives à permutation de X près.
Quand $j \in \mathbb{N}$, on pourra utiliser $[\![1; j]\!]$ pour désigner $\{i \in \mathbb{N} \mid 1 \leq i \leq j\}$.

Question 1. Dénombrer les classes des problèmes suivants :

1. ADD.
2. IDD
3. IUU
4. SUD

Question 2. Pourquoi la caractéristique *bijective* n'est-elle pas étudiée ?

Question 3. Les problèmes des douze chemins admettent chacun au moins une représentation naturelle de leurs classes d'équivalence et un *ordre lexicographique* sur ces représentations.

Donner de telles représentations et ordres pour les problèmes de la question 1.

Un *algorithme de déclassement* pour un problème des douze chemins est un algorithme qui prend en entrée un entier i et renvoie en sortie une représentation de la classe d'équivalence située à la $i+1$-ème place selon l'ordre lexicographique associée à ce problème.

Question 4. Donner des algorithmes de déclassement pour les problèmes de la question 1.

Dans la suite, on s'intéresse uniquement au problème SDU.

Question 5. Donner une relation de récurrence décrivant le dénombrement des fonctions de SDU.

Soit A et B deux ensembles d'entiers naturels, on définit $A \prec B = (\texttt{min}\ A \leq \texttt{min}\ B)$.
Soit A et B deux ensembles, on note $A - B$ pour $\{x \in A \mid x \notin B\}$.
Soit A et B deux ensembles d'entiers naturels, on définit :

$$A \lhd B = \begin{cases} & A = B \\ \text{ou} & A \subset B \text{ et } \texttt{max}\ A < \texttt{min}\ (B - A) \\ \text{ou} & B \subset A \text{ et } \texttt{min}\ (A - B) < \texttt{max}\ B \\ \text{ou} & \texttt{min}\ (A - B) < \texttt{min}\ (B - A) \end{cases}.$$

Question 6. Donner une représentation des classes de SDU, puis un ordre total sur ces classes.

Question 7. Donner les 10 premières fonctions, pour cet ordre, de SDU avec $n = 5$ et $k = 3$.

Question 8. Donner un algorithme de déclassement pour SDU.

Partitions et listes

Ce sujet concerne une représentation des partitions de l'ensemble $[\![1, K]\!] := \{1, \ldots, K\}$, où K est un entier supérieur ou égal à 1. On rappelle qu'une partition de $[\![1, K]\!]$ est un ensemble $P = \{B_1, \ldots, B_n\}$ où les B_i sont des sous-ensembles non-vides de $[\![1, K]\!]$ appelés *blocs*, disjoints deux à deux et tels que $\bigcup_{i=1}^{n} B_i = [\![1, K]\!]$.
Si $i \in [\![1, K]\!]$, on appelle *parent de i dans P* l'élément minimal du bloc qui contient i, et cet élément est dénoté $\mathrm{par}_P(i)$.
On note $[\![1, K]\!]^K$ l'ensemble des listes comprenant K éléments de $[\![1, K]\!]$. Pour toute liste $s \in [\![1, K]\!]^K$, on écrit $s = (s_1, \ldots, s_K)$, où s_i désigne le i-ème élément de la liste.
On appelle *liste K-incrémentale* une liste $s \in [\![1, K]\!]^K$ qui satisfait les propriétés suivantes pour tout $i \in \{1, \ldots, K\}$:

— $s_i \leq i$

— $s_i = s_{s_i}$.

On suppose que l'entier K est fixé pour tout le sujet.

Question 1. Donner un exemple d'une liste $s \in [\![1, K]\!]^K$ qui n'est pas K-incrémentale, et donner un exemple de liste 4-incrémentale.

Solution : Pour une liste non-incrémentale, il faut au minimum $K = 2$. Les listes $(2, 1)$ et $(2, 2)$ ne sont pas 2-incrémentales.

Les listes 4-incrémentales sont : $(1, 1, 1, 1)$, $(1, 1, 1, 4)$, $(1, 1, 3, 1)$, $(1, 1, 3, 3)$, $(1, 1, 3, 4)$, $(1, 2, 1, 1)$, $(1, 2, 1, 2)$, $(1, 2, 1, 4)$, $(1, 2, 2, 1)$, $(1, 2, 2, 2)$, $(1, 2, 2, 4)$, $(1, 2, 3, 1)$, $(1, 2, 3, 2)$, $(1, 2, 3, 3)$, $(1, 2, 3, 4)$.

Question 2. Soit $P = \{B_1, \ldots, B_n\}$ une partition de $[\![1, K]\!]$. Montrer que la liste $(\mathrm{par}_P(1), \ldots, \mathrm{par}_P(K))$ est K-incrémentale. Définir une bijection entre l'ensemble des listes K-incrémentales et l'ensemble des partitions de $[\![1, K]\!]$.

Solution : On vérifie les deux propriétés pour tout $i \in [\![1, K]\!]$. Par définition $s_i = \mathrm{par}_P(i)$ est l'élément minimal du bloc de P qui contient i. Donc $\mathrm{par}_P(i) \leq i$.

Par définition, $\mathrm{par}_P(i)$ est l'élément minimal du bloc qui contient i, et $\mathrm{par}_P(\mathrm{par}_P(i))$ est l'élément minimal du bloc qui contient $\mathrm{par}_P(i)$. Puisque $\mathrm{par}_P(i)$ et i sont dans le même bloc, ces deux éléments sont égaux.

Donc, pour toute partition P, on obtient une liste incrémentale qu'on note $\mathrm{liste}(P)$. Réciproquement, étant donné une liste incrémentale s, on peut définir une partition $\mathrm{part}(s)$: pour cela on prend la partition qui correspond à la relation d'équivalence $i \sim j \iff s_i = s_j$. Plus explicitement, $\mathrm{part}(s) = \{\{k \mid s_k = i\} \mid i \in [\![1, K]\!]\} \setminus \{\emptyset\}$.

Il faut montrer que les fonctions $\mathrm{liste}(-)$ et $\mathrm{part}(-)$ sont inverses.

— Pour toute liste s qui est K-incrémentale, on veut montrer que $\mathrm{liste}(\mathrm{part}(s)) = s$. On montre que pour tout indice i, $(\mathrm{liste}(\mathrm{part}(s)))_i = s_i$.

L'élément $(\mathrm{liste}(\mathrm{part}(s)))_i$ est par définition $\mathrm{par}_{\mathrm{part}(s)}(i)$, c'est-à-dire l'élément minimal du bloc qui contient i dans $\mathrm{part}(s)$. Par définition, ce bloc est l'ensemble des indices où apparait la valeur s_i, c'est-à-dire l'ensemble $\{j \mid s_j = s_i\}$. On sait que $s_{s_i} = s_i$ donc s_i est un élément du bloc, et c'est forcément le minimum, puisque pour tout j du bloc on a $s_i = s_j \leq j$.

— Pour toute partition P, on veut montrer que $\mathrm{part}(\mathrm{liste}(P)) = P$. Deux éléments i, j sont dans le même bloc de $\mathrm{part}(\mathrm{liste}(P))$ ssi $(\mathrm{liste}(P))_i = (\mathrm{liste}(P))_j$, c'est-à-dire $\mathrm{par}_i P = \mathrm{par}_j P$; c'est vrai si et seulement si i et j sont dans le même bloc de P.

Question 3. Montrer que si s est une liste K-incrémentale et i apparait dans s, alors $s_i = i$.

Solution : Preuve qui utilise la bijection : puisque si i apparait dans s alors i est un parent dans la partition P qui correspond à s. Cela signifie que i est l'élément minimal du bloc dans lequel il apparait. En particulier $\text{par}_P(i) = i$.
Preuve sans la bijection : si i apparait dans s alors on a $s_n = i$ pour un certain n. Donc $s_i = s_{s_n} = s_n = i$.

Un *croisement* dans une partition $P = \{B_1, \ldots, B_n\}$ de $[\![1, K]\!]$ est un quadruplet d'entiers $a, b, c, d \in [\![1, K]\!]$ tels que $a < b < c < d$ avec $a, c \in B_i$ et $b, d \in B_j$ pour $i \neq j$. Par exemple, si $K = 4$, la partition $\{\{1, 3\}, \{2, 4\}\}$ contient un croisement alors que $\{\{1, 4\}, \{2, 3\}\}$ n'en contient pas.
De la même manière, un *croisement* dans une liste $s \in [\![1, K]\!]^K$ est un quadruplet d'entiers $a, b, c, d \in [\![1, K]\!]$, tels que $a < b < c < d$, $s_a = s_c$ et $s_b = s_d$.
Une partition (ou liste) *non croisée* est une partition (ou liste) sans croisement.

Question 4. Montrer que la bijection de la Question 1 se restreint à une bijection entre les listes K-incrémentales non croisées et les partitions de $[\![1, K]\!]$ non croisées.

Solution : Soit s une liste incrémentale et P la partition correspondante. Si on a un croisement (a, b, c, d) dans s, alors on a un croisement dans P.

On cherche à éliminer les croisements d'une partition tout en préservant la cardinalité et l'élément minimal de chaque bloc. Pour cela, on définit une relation binaire "$\to$" sur l'ensemble $[\![1, K]\!]^K$ de la manière suivante : pour toute paire de listes (s, t), on dit que $s \to t$ s'il existe un croisement (a, b, c, d) dans s tel que $s_a < s_b$ et, pour tout $i \in [\![1, K]\!]$:

$$t_i = \begin{cases} s_c & \text{si } i = d \\ s_d & \text{si } i = c \\ s_i & \text{sinon.} \end{cases}$$

Par exemple, $(1, 2, 1, 3, 2) \to (1, 2, 2, 3, 1)$.

Question 5. Montrer que, si $s \to t$, alors t est strictement supérieur à s selon l'ordre lexicographique sur les listes d'entiers.

Solution : Soit (a, b, c, d) le croisement dans s qui donne lieu à la réduction $s \to t$. On remarque que s et t coincident pour tous les indices $< c$, donc il suffit de montrer que $s_c < t_c$. Mais en combinant les hypothèses on a $s_c = s_a < s_b = s_d = t_c$.

Question 6. Montrer que si s est K-incrémentale et comprend un croisement, alors il existe une liste t telle que $s \to t$.

Solution : Il faut montrer que s'il existe un croisement (a, b, c, d) dans s alors il existe forcément un croisement (a', b', c', d') tel que $s_{a'} < s_{b'}$. La solution est de prendre (s_b, s_a, b, c). Les conditions pour un croisement sont satisfaites : $s_b < s_a$, $s_a < b$ (puisque $s_a \leq a < b$), $b < c$, et $s_b = s_{s_b}$ et $s_c = s_a = s_{s_a}$.

Question 7. Montrer que si s est K-incrémentale et $s \to t$, alors t est K-incrémentale.

Solution : Si la réduction concerne un croisement (a, b, c, d) avec $s_a < s_b$, alors t est incrémentale : pour $i \neq c, d$, $t_i \leq i$ parce que $s_i = t_i$ et s est incrémentale ; pour $i = c$, $t_i = s_d = s_b < b < c$; et pour $i = d$, $t_i = s_c = s_a < a < d$.

Ensuite on doit montrer que $t_{t_i} = t_i$. On remarque que c et d n'apparaissent pas dans s, car sinon on aurait $c = s_c = s_a \leq a$ ou $d = s_d = s_b \leq b$. Donc, pour tout i, $t_{s_i} = s_{s_i}$. Si $i \neq c, d$, alors $t_{t_i} = t_{s_i} = s_{s_i} = s_i$, et $t_{t_c} = t_{s_d} = s_{s_d} = s_d = t_c$ et symétriquement $t_{t_d} = t_d$.

Question 8. Montrer qu'une liste K-incrémentale se réduit en une liste K-incrémentale sans croisement en un nombre fini d'étapes.

Solution : Il suffit de combiner les trois questions précédentes. L'ordre lexicographique sur un ensemble fini ne peut pas avoir de chaine infinie et strictement croissante.

Pomsets probabilistes

On fixe un ensemble L, dont les éléments sont vus comme des actions possibles d'un processus concurrent. On modélise un tel processus à l'aide d'ordres partiels étiquetés, comme suit.
Un *pomset* est un triplet $(P, \leq_P, \ell_P)$ où P est un ensemble fini, $\leq_P$ est un ordre partiel sur P, et $\ell_P : P \to L$ est une fonction.
Une *trace* d'un pomset $(P, \leq_P, \ell_P)$ est une paire (x, σ) où :

— $x \subseteq P$ est un ensemble clos par le bas, c'est-à-dire que si $p \in x$ et $q \in P$ avec $q \leq_P p$, alors $q \in x$. La cardinalité de x est dénotée $|x|$.

— $\sigma : x \to \{1, \ldots, |x|\}$ est une bijection telle que, pour tout $p, q \in x$, si $p \leq_P q$ alors $\sigma(p) \leq \sigma(q)$.

On désigne souvent le pomset $(P, \leq_P, \ell_P)$ simplement par P, et on appelle Traces(P) l'ensemble de ses traces.
On dit que deux traces (x, σ) et (x', σ') sont *équivalentes* s'il existe une bijection $\varphi : x \to x'$ telle que, pour tout $p \in x$, $\ell_P(p) = \ell_P(\varphi(p))$ et $\sigma(p) = \sigma'(\varphi(p))$.

Question 1. Soient a, b deux éléments distincts de L. Lister les traces du pomset à trois éléments $P = \{p_1, p_2, p_3\}$, où on a seulement $p_i \leq_P p_j$ si $i = 1$ et $j = 2$, ou si $i = j$, et $\ell_P(p_1) = \ell_P(p_3) = a$ et $\ell_P(p_2) = b$. Indiquer les paires de traces qui sont équivalentes.

Question 2. Pour un pomset $(P, \leq_P, \ell_P)$ arbitraire, définir une fonction mot : Traces$(P) \to L^*$, où L^* est l'ensemble des mots définis à partir de l'alphabet L. La fonction doit satisfaire la condition suivante : deux traces (x, σ) et (x', σ') de P sont équivalentes si et seulement si $mot((x, \sigma)) = mot((x', \sigma'))$.

On fixe un pomset P pour le reste du sujet, et on note $D(P)$ l'ensemble des sous-ensembles de P qui sont clos par le bas. Une *valuation probabiliste* sur P est une fonction $v : D(P) \to [0, 1]$ qui satisfait les conditions suivantes :

— $v(\emptyset) = 1$;

— Pour tout entier $n \geq 1$, si $x, y_1, \ldots, y_n \in D(P)$ avec $x \subseteq y_i$ pour tout $1 \leq i \leq n$, alors

$$\sum_{\substack{S \subseteq \{1,\ldots,n\} \\ S \neq \emptyset}} (-1)^{|S|+1} \, v\left(\bigcup_{i \in S} y_i\right) \leq v(x).$$

Question 3. Montrer que, pour toute valuation probabiliste v sur P, si $x, y \in D(P)$ et $x \subseteq y$, alors $v(x) \geq v(y)$. Donner un exemple de valuation probabiliste w pour le pomset P de la Question 1, telle que $w(P) < 1$.

Question 4. Soit $f : P \to [0, 1]$ une fonction arbitraire. Montrer que la fonction $v_f : D(P) \to [0, 1]$ définie par $v_f(x) = \prod_{p \in x} f(p)$ est une valuation probabiliste.

Si $(x, \sigma), (y, \tau) \in$ Traces(P), on écrit $(x, \sigma) \preceq (y, \tau)$ lorsque $x \subseteq y$ et $\sigma^{-1}(i) = \tau^{-1}(i)$ pour tout $i \in \{1, \ldots, |x|\}$. On note $\varepsilon \in$ Traces(P) la seule trace de la forme $(\emptyset, \sigma)$.
Une *valuation-traces* sur P est une fonction vt : Traces$(P) \to [0, 1]$ telle que $vt(\varepsilon) = 1$ et $vt((x, \sigma)) \geq vt((y, \tau))$ si $(x, \sigma) \preceq (y, \tau)$.

Question 5. Montrer que la relation $\preceq$ sur Traces(P) est un ordre partiel.

Question 6. Montrer que, si $v : D(P) \to [0,1]$ est une valuation probabiliste, la fonction

$$\begin{aligned} vt : \mathrm{Traces}(P) &\to [0,1] \\ (x,\sigma)) &\mapsto \frac{v(x)}{\#\left\{\sigma \mid \sigma : x \to \{1,\ldots,|x|\} \text{ est une bijection et } (x,\sigma) \in \mathrm{Traces}(p)\right\}}. \end{aligned}$$

est une valuation-traces.

Question 7. Peut-on définir une valuation probabiliste à partir d'une valuation-traces ?

Domination romaine des graphes

Soit $G = (V, E)$ un graphe non orienté, où V est un ensemble de sommets et E un ensemble d'arêtes, qui sont des couples de sommets distincts. Le *degré* de G est le nombre maximal de voisins d'un sommet de G, noté $\Delta(G)$. Pour tout $W \subseteq V$, on note $G[W]$ le sous-graphe de G induit par W, dans lequel on ne conserve que les sommets dans W et les arêtes entre eux.

Une fonction $f : V \to \{0, 1, 2\}$ est appelée une *fonction de domination romaine* (FDR) sur G si tout sommet x pour lequel $f(x) = 0$ possède un voisin y pour lequel $f(y) = 2$:

$$\forall x \in V, f(x) = 0 \implies \exists y \in V, \{x, y\} \in E \wedge f(y) = 2 \ .$$

On dit alors que y *protège* x.

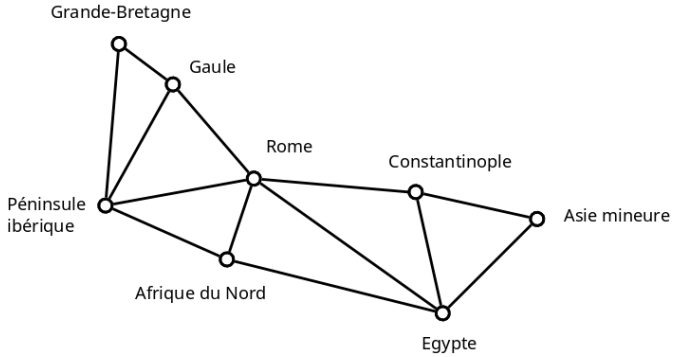
Le *poids* d'une FDR f est donné par : $\gamma(f) := \sum_{v \in V} f(v)$. On note $\mathcal{F}(G)$ l'ensemble des FDRs sur G. On note $\gamma_R(G)$ la valeur minimale du poids des FDRs sur G :

$$\gamma_R(G) = \min_{f \in \mathcal{F}(G)} \gamma(f) \ .$$

On dit que f est *optimale* ssi $\gamma(f) = \gamma_R(G)$.

On remarque qu'une FDR est définie par une partition de V. On utilisera l'abus de notation $f = (V_0, V_1, V_2)$ où V_i est l'ensemble des sommets x tels que $f(x) = i$. De plus on note n_0, n_1, n_2 les cardinaux respectifs de V_0, V_1, V_2, et n le cardinal de V.

Question 1. L'empire Romain sous Constantin (IIIe siècle après J-C.) est modélisé par le graphe ci-dessous. Chaque sommet représente une province, sur laquelle peuvent être stationnées zéro, une ou deux, armées, de sorte que chaque province sans armée peut être défendue en déplaçant une armée d'une province voisine qui en comporte deux. Combien d'armées suffisent-elles pour défendre l'empire romain ?



Solution : Quatre armées suffisent et de nombreuses méthodes sont possibles. L'une d'entre elles est de placer deux armées à Rome, une en Grande-Bretagne et une en Asie mineure.

Question 2. Soit G un graphe et $f = (V_0, V_1, V_2)$ une FDR optimale sur G. Montrer que :

1. $G[V_1]$ a degré au plus 1.
2. Il n'existe pas d'arête entre V_1 et V_2 dans G.
3. Tout sommet de V_0 est adjacent à au plus deux sommets de V_1.

Solution : 1. Supposons qu'il existe un sommet de degré 2 dans $G[V_1]$. Alors on a trois sommets dans V_1, que l'on va noter (x, y, z), tels que $f(x) = f(y) = f(z) = 1$ et $(x, y), (x, z) \in E$. La fonction f' telle que $f'(x) = 2$ et $f'(y) = f'(z) = 0$ est toujours une FDR : les sommets y, z sont maintenant protégés par x, et ils ne protégeaient aucun autre sommet. Mais f' est de poids strictement inférieur à f, c'est une contradiction.

2. Supposons qu'il existe $x \in V_1$ et $y \in V_2$ tels que $(x, y) \in E$. Soit f' telle que $f'(x) = 0$ et $f'(y) = 2$; alors f' est toujours une FDR : x est protégé par y, et x ne protégeait aucun autre sommet. Mais f' est de poids strictement inférieur à f, contradiction.

3. Supposons qu'il existe $x \in V_0$ adjacent à trois sommets y, z, t de V_1. Alors on peut modifier la fonction en : $f'(x) = 2$ et $f'(y) = f'(z) = f'(t) = 1$. Le poids est strictement inférieur et cette fonction est bien une FDR (y, z, t sont maintenant protégés par x) ; contradiction.

> Le *voisinage* d'un sommet $v \in V$, noté $N(v)$ est l'ensemble des voisins $\{u \in V, \{u, v\} \in E\}$. Le *voisinage clos*, noté $C(v)$, est défini par $C(v) := N(v) \cup \{v\}$. Cette définition s'étend naturellement à un ensemble de sommets.
>
> Soit $S \subseteq V$ et $v \in S$. Le sommet u est appelé un *voisin privé de v relativement à S* ssi $C(u) \cap S = \{v\}$. Il est dit *externe* s'il appartient à $V \backslash S$.

Question 3. Soit G un graphe et f une FDR optimale sur G. Soit $H = G[V_0 \cup V_2]$. Montrer que :

1. Tout sommet $v \in V_2$ a au moins deux voisins privés relativement à V_2 dans H.
2. Si $v \in V_2$ a exactement un voisin privé *externe* relativement à V_2 dans H, que l'on note $w \in V_0$, alors w n'a aucun voisin dans V_1.

Supposons de plus que f minimise n_1. Montrer que :

3. Tout sommet de V_1 n'admet aucun voisin dans V_1.
4. Tout sommet de V_0 est adjacent à au plus un sommet de V_1.
5. Soit $v \in V_2$ avec exactement deux voisins privés externes relativement à V_2 dans H, notés $w_1, w_2 \in V_0$. Alors il n'existe pas de sommets $y_1, y_2 \in V_1$ tels que (y_1, w_1, v, w_2, y_2) forme un chemin dans G.

Solution : Comme indiqué dans la question, les voisins privés de v relativement à S sont les sommets du graphe *dont le seul voisin dans S est v*. Noter que cela peut inclure v si v n'a pas d'autre voisin dans S. Les voisins privés externes sont, de plus, extérieurs à S (et donc n'incluent jamais v). De nouveau on procède par l'absurde.

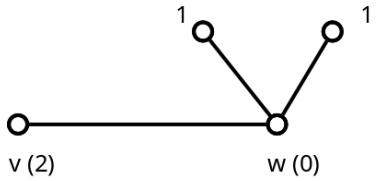
1. On se place dans le sous-graphe H. Les voisins privés de v relativement à V_2 sont les sommets de H dont le seul voisin dans V_2 est v, ce qui inclut v s'il n'a pas de voisin dans V_2. Supposons que v n'a qu'un seul voisin privé relativement à V_2. Il y a deux cas possibles :

— Soit ce voisin est v lui-même, ce qui signifie que v n'est pas protégé par un autre sommet de V_2, mais tout voisin de v dans V_0 est déjà protégé par un autre sommet de V_2. On peut donc changer la valeur en v à 1, contradiction.

— Soit ce voisin n'est pas v. Cela signifie que v est lui-même protégé par un autre sommet de V_2, et qu'il n'est « protecteur exclusif » que d'un seul sommet de V_0. On peut donc changer la valeur de ce sommet à 1, et la valeur de v à 0.

2. On sait d'après le point précédent que v a au moins deux voisins privés relativement à V_2 dans H. On doit donc être dans un cas où l'un de ces voisins est v, ce qui signifie que v n'a pas de voisin dans V_2.

De plus v n'est protecteur exclusif que du seul sommet $w \in V_0$ par hypothèse. On forme donc une nouvelle FDR en changeant la valeur de v à 0, et la valeur de tous les voisins $y \in N(w) \cap V_1$ à 0. On change la valeur de $f(w)$ à 2. Si w avait au moins un voisin dans V_1, cette FDR est de poids strictement inférieur, contradiction (voir le schéma ci-dessous).



3. C'est trivial : on pourrait réduire le nombre de 1 en associant les poids 2-0 à ces deux sommets.

4. Si un sommet de V_0 est adjacent à deux sommets de V_1, on peut remplacer ce sommet par 2 pour qu'il protège les deux sommets. Le poids est inchangé, mais n_1 diminue. Contradiction.

5. On suppose donc que v a exactement deux voisins $w_1, w_2 \in V_0$ dont le seul voisin dans V_2 est v lui-même. Si un chemin (y_1, w_1, v, w_2, y_2) existe, alors les valeurs de f sur ce chemin sont $(1, 0, 2, 0, 1)$. On change ces valeurs en $(0, 2, 0, 2, 0)$; il suffit de vérifier que la fonction modifiée est toujours une FDR. Cette nouvelle fonction a le même poids et moins de 1, contradiction.

Question 4. Soit G un graphe *sans sommet isolé*, n son nombre de sommets, et f une FDR optimale qui minimise n_1. Soit $c = |\{v \in V_0, |N(v) \cap V_2| \geq 2\}|$. Soit $a_i, i = 1, 2, \ldots, \Delta(G)$ le nombre de sommets de V_2 qui ont exactement i voisins privés externes relativement à V_2 dans H. Montrer que :

$$n_2 = \sum_{j=1}^{\Delta(G)} a_j \tag{1}$$

$$n_0 = \left(\sum_{j=1}^{\Delta(G)} j a_j \right) + c \tag{2}$$

$$n_1 \leq \left(a_2 + \sum_{j=3}^{\Delta(G)} j a_j \right) + c \tag{3}$$

Solution : La première égalité (sur n_2) est triviale : c'est par définition de n_2 (et la somme est jusqu'à $\Delta(G)$ par définition du degré du graphe).

Pour l'égalité sur n_0, on sépare d'abord les sommets de V_0 en :

— Ceux qui n'ont qu'un seul voisin dans V_2, notés V_0'

— Ceux qui ont deux ou plus voisins dans V_2, au nombre de c (par définition), notés V_0''

Chaque sommet de V_0' est voisin privé externe d'un sommet de V_2. Chaque sommet de V_2 a au moins un voisin privé externe dans V_0. En comptant le nombre de voisins privés externes des sommets de V_2, on énumère donc tous les sommets de V_0'. Par définition de a_j :

$$n_0 = \left(\sum_{j=1}^{\Delta(G)} j a_j \right) + c$$

Ce qui donne l'égalité sur n_0.

On prouve maintenant l'inégalité sur n_1. Par hypothèse, le graphe ne contient pas de sommets isolés. Tout sommet de V_1 admet donc au moins un voisin dans V_0 (il n'en a pas dans V_1 ni dans V_2 d'après les questions précédentes). Tout sommet de V_0 est voisin d'au plus un sommet de V_1 (cf. 3.3). On sépare les sommets de V_1 en :

— Ceux qui ont (au moins) un voisin dans V_0', notés V_1'

— Ceux qui n'en ont pas, notés V_1'' (et qui ont donc un voisin dans V_0'')

On a donc :

$$n_1 = |V_1'| + |V_1''| \leq |V_1'| + c \ .$$

Comme tout sommet de V_1' admet un voisin dans V_0', et tout sommet de V_0' est voisin privé externe d'un sommet de V_2, on se ramène à un comptage sur V_2. Pour chaque sommet de V_2 :

— S'il a trois ou plus voisins privés externes, alors on obtient le même nombre (potentiellement) d'éléments distincts de V_1'
— S'il a deux voisins privés externes, alors d'après 3.5, l'un de ces sommets n'a pas de voisin dans V_1, donc on obtient au plus un seul élément de V_1'
— S'il a un seul voisin privé externe, alors d'après 3.2 ce voisin n'a aucun voisin dans V_1, et donc on obtient zéro élément de V_1'.

On peut donc écrire :

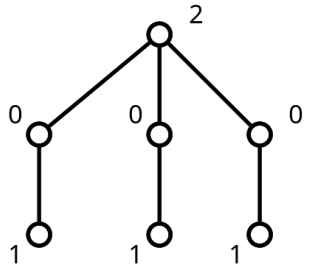
$$n_1 \leq a_2 + \sum_{j=3}^{\Delta(G)} j a_j + c \ .$$

Question 5. En reprenant les hypothèses de la Question 4, en déduire que $n_0 \geq 3n/7$. Donner un exemple de graphe pour lequel cette inégalité est une égalité.

Solution : On va montrer que $n_1 + n_2 \leq \frac{4}{3} n_0$ (ce qui donne immédiatement le résultat).

$$n_1 + n_2 \leq c + a_1 + 2a_2 + \sum_{j=3}^{\Delta(G)} (j+1) a_j \leq c + a_1 + 2a_2 + \frac{4}{3} \sum_{j=3}^{\Delta(G)} j a_j \leq \frac{4}{3} n_0 \ .$$

Pour atteindre la borne, prendre l'arbre suivant :



Il est facile de se convaincre que la FDR représentée ci-dessus est optimale et minimise le nombre de 1.

Question 6. Soit $\delta(G)$ le degré minimal de G, c'est-à-dire le nombre minimal de voisins des sommets de G. En particulier, si le graphe n'est pas connexe, on pose $\delta(G) = 0$.

Montrer que :

$$\gamma_R(G) \leq n \frac{2 + \ln((1+\delta(G))/2)}{1+\delta(G)} \ .$$

Indication : on s'intéressera au poids *moyen* d'une famille de FDRs bien choisie.

Solution : Étant donné un graphe G, on sélectionne un ensemble de sommets A en prenant des sommets au hasard avec probabilité p. La taille moyenne de A est np. Notons $\delta := \delta(G)$. Nous allons utiliser A comme ensemble V_2 pour une FDR, et utiliser la borne sur $\delta(G)$ pour borner le poids de cette fonction.

Soit $B = V - N[A]$ l'ensemble des sommets non protégés par A. La fonction $f = (V - (A \cup B), B, A)$ est une FDR pour G.

On calcule la taille moyenne de B. La probabilité que v soit dans B est égale à la probabilité que v ne soit pas dans A et qu'aucun sommet de A ne soit voisin de v. C'est-à-dire : $(1-p)^{1+deg(v)}$. Comme $deg(v) \geq \delta$ et $e^{-x} \geq 1 - x$,

on a : $\Pr(v \in B) \leq e^{-p(1+\delta)}$. Donc la taille moyenne de B est au plus $ne^{-p(1+\delta)}$, et le poids moyen de f est au plus $2np + ne^{-p(1+\delta)}$. On a donc :

$$\mathbb{E}(f(V)) \leq 2np + ne^{-p(1+\delta)}$$

où l'espérance est sur le choix de A. On choisit la valeur de p qui minimise cette fonction :

$$p = \frac{1}{1+\delta} \ln\left(\frac{1+\delta}{2}\right) \ .$$

Ce qui donne :

$$\mathbb{E}(f(V)) \leq n\frac{2 + \ln((1+\delta(G))/2)}{1+\delta(G)} \ .$$

Comme c'est le poids moyen de f, il existe au moins une FDR avec ce poids.

Question 7. Soit G un graphe sans sommet isolé. Montrer que : $\gamma_R(G) \geq \frac{2n}{\Delta(G)+1}$.

Solution : Soit $f = (V_0, V_1, V_2)$ une FDR. Comme tout $v \in V_0$ doit être adjacent à un sommet de V_2, on a : $n_0 \leq \Delta n_2$. De plus :

$$n = n_0 + n_1 + n_2 \leq n_1 + (\Delta + 1)n_2$$

Par ailleurs :

$$\gamma_R(G) = n_1 + 2n_2$$

On a donc :

$$\begin{aligned}\gamma_R(G)(\Delta+1) = n_1(\Delta+1) + 2n_2(\Delta+1) &\geq 2n_1 + 2(\Delta+1)n_2 \\ &\geq 2n \ .\end{aligned}$$

Notons qu'on a besoin de $\Delta(G) \geq 1$; ce n'est pas vrai si G est le graphe singleton de degré 0. (Mais tout graphe avec au moins une arête fonctionne).

Protocoles de calcul à base de cartes

On considère des cartes dont la face recto porte un symbole noté $\boxed{\heartsuit}$ (blanc) ou $\boxed{\spadesuit}$ (noir). Une carte face verso affiche le symbole $\boxed{?}$. Une *séquence* de cartes est un vecteur $(\alpha_0, \ldots, \alpha_{m-1})$ où α_i est (formellement) une paire indiquant le symbole de la carte (blanc ou noir) et son état (recto ou verso). Dans la suite de l'énoncé on écrira simplement des séquences de cartes comme $\boxed{?}\boxed{?}\boxed{\spadesuit}\boxed{\heartsuit}\boxed{\spadesuit}\boxed{\heartsuit}$ ...

On définit l'*encodage* d'un Booléen de la manière suivante : 0 est encodé par $\boxed{\spadesuit}\boxed{\heartsuit}$ et 1 par $\boxed{\heartsuit}\boxed{\spadesuit}$. On généralise l'encodage à des n-uplets de Booléens en plaçant les encodages côte à côte. On écrit par exemple $\underbrace{\boxed{?}\boxed{?}}_{b}\boxed{\spadesuit}\boxed{\heartsuit}\boxed{\spadesuit}\boxed{\heartsuit}$ pour une séquence de 6 cartes encodant $(b, 0, 0)$.

Soit $f : \{0,1\}^n \to \{0,1\}$ une fonction Booléenne. Un *protocole de calcul à cartes* pour la fonction f est un algorithme qui :
1. Prend en entrée une séquence de $m = 2n + m_b + m_n$ cartes de la forme :

$$\underbrace{\boxed{?}\boxed{?}}_{b_0}\underbrace{\boxed{?}\boxed{?}}_{b_1}\cdots\underbrace{\boxed{?}\boxed{?}}_{b_{n-1}}\underbrace{\boxed{\heartsuit}\boxed{\heartsuit}\cdots\boxed{\heartsuit}}_{m_b \text{ cartes}}\underbrace{\boxed{\spadesuit}\boxed{\spadesuit}\cdots\boxed{\spadesuit}}_{m_n \text{ cartes}},$$

encodant un vecteur Booléen $(b_0, \ldots, b_{n-1})$ *secret* ainsi qu'un nombre fixé de cartes additionnelles face visible.
2. Peut appliquer les opérations suivantes à la séquence de cartes :
   — $\mathsf{Permute}(\pi)$: on permute les cartes selon une permutation fixe π de $\{0, \ldots, m-1\}$:

$$(\alpha_0, \ldots, \alpha_{m-1}) \to (\alpha_{\pi(0)}, \ldots, \alpha_{\pi(m-1)})$$

   — $\mathsf{Coupe}(i, 2j)$: dans la sous-séquence des cartes $\alpha_i, \ldots, \alpha_{i+2j-1}$, on applique une *coupe* aléatoire :

$$(\alpha_i, \ldots, \alpha_{i+j-1}, \alpha_{i+j}, \ldots, \alpha_{i+2j-1}) \to \begin{cases} (\alpha_i, \ldots, \alpha_{i+j-1}, \alpha_{i+j}, \ldots, \alpha_{i+2j-1}) \\ (\alpha_{i+j}, \ldots, \alpha_{i+2j-1}, \alpha_i, \ldots, \alpha_{i+j-1}) \end{cases}$$

   où chaque cas arrive avec probabilité 1/2. En particulier, en appliquant une coupe sur des cartes face cachée, il est impossible de dire quel cas s'est produit.
   — $\mathsf{Retourne}(i)$: on retourne la carte à la position i, qui passe donc de recto à verso ou inversement.
3. Est tel qu'à la fin du protocole, les deux premières cartes de la séquence, *face verso*, encodent $f(b_0, \ldots, b_{n-1})$. Les autres cartes peuvent être dans un état quelconque.

Les actions effectuées peuvent dépendre des symboles observés sur les cartes retournées. De plus, on exige que le protocole ne révèle aucune information sur les entrées $b_0, \ldots, b_{n-1}$. On pourra justifier cette propriété de manière informelle.

Question 1. Donner un protocole calculant la fonction NOT sans révéler la valeur de son entrée : $\underbrace{\boxed{?}\boxed{?}}_{b} \to \underbrace{\boxed{?}\boxed{?}}_{\overline{b}}$

Solution : Le protocole consiste en un échange des deux cartes. Par définition, le résultat est un encodage de la négation du Booléen b. On remarque qu'aucune carte n'a été retournée au cours du protocole, donc aucune information sur la valeur de b n'a pu être obtenue.

Question 2. Sur l'entrée : $\underbrace{\boxed{?}\boxed{?}}_{a}\underbrace{\boxed{?}\boxed{?}}_{b}\underbrace{\boxed{\spadesuit}\boxed{\heartsuit}}_{0}$ on effectue les opérations suivantes :

— Retourner les deux cartes visibles face verso

— Appliquer la permutation suivante : $\underbrace{\boxed{?}\boxed{?}}_{a}\underbrace{\boxed{?}\boxed{?}}_{b}\underbrace{\boxed{?}\boxed{?}}_{0} \to \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}$ (positions $1\,2\,3\,4\,5\,6 \to 1\,3\,5\,2\,4\,6$)

— Appliquer une coupe (avec probabilité 1/2) : $\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?} \to \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}$ (positions $1\,2\,3\,4\,5\,6 \to 4\,5\,6\,1\,2\,3$)

— Appliquer la permutation suivante : $\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?} \to \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}$ (positions $1\,2\,3\,4\,5\,6 \to 1\,4\,2\,5\,3\,6$)

Montrer que la séquence de cartes obtenue est un encodage de : $(a \oplus r, b \oplus r, r)$, où $\oplus$ est le OU exclusif, et r vaut 0 ou 1 avec probabilité 1/2.

Solution : Calculons déjà le résultat de la permutation, en notant $\boxed{a_0}\boxed{a_1}$ et $\boxed{b_0}\boxed{b_1}$ les encodages de a et b.

Avec probabilité 1/2, la coupe laisse la séquence inchangée. La composition des deux permutations donne :

$$\boxed{a_0}\boxed{a_1}\boxed{b_0}\boxed{b_1}\boxed{\spadesuit}\boxed{\heartsuit} \to \boxed{a_0}\boxed{b_0}\boxed{\spadesuit}\boxed{a_1}\boxed{b_1}\boxed{\heartsuit} \to \boxed{a_0}\boxed{a_1}\boxed{b_0}\boxed{b_1}\boxed{\spadesuit}\boxed{\heartsuit}$$

c'est-à-dire qu'en fait, on n'a rien fait. On est dans le cas $r = 0$.

Avec probabilité 1/2, la coupe échange les deux moitiés de la séquence. Alors :

$$\boxed{a_0}\boxed{a_1}\boxed{b_0}\boxed{b_1}\boxed{\spadesuit}\boxed{\heartsuit} \to \boxed{a_0}\boxed{b_0}\boxed{\spadesuit}\boxed{a_1}\boxed{b_1}\boxed{\heartsuit} \to \boxed{a_1}\boxed{b_1}\boxed{\heartsuit}\boxed{a_0}\boxed{b_0}\boxed{\spadesuit} \to \boxed{a_1}\boxed{a_0}\boxed{b_1}\boxed{b_0}\boxed{\heartsuit}\boxed{\spadesuit}$$

On a donc interverti les cartes de a, b et du bit auxiliaire. On est dans le cas $r = 1$.

Dans les deux cas on a bien obtenu $(a \oplus r, b \oplus r, r)$.

Question 3. En déduire :

1. Un protocole pour l'opération COPY, qui sur l'entrée : $\underbrace{\boxed{?}\boxed{?}}_{a}\underbrace{\boxed{\spadesuit}\boxed{\heartsuit}}_{0}\underbrace{\boxed{\spadesuit}\boxed{\heartsuit}}_{0}$ renvoie $\underbrace{\boxed{?}\boxed{?}}_{a}\underbrace{\boxed{?}\boxed{?}}_{a}\underbrace{\boxed{\spadesuit}\boxed{\heartsuit}}_{0}$
2. Un protocole pour l'opération XOR, qui sur l'entrée : $\underbrace{\boxed{?}\boxed{?}}_{a}\underbrace{\boxed{?}\boxed{?}}_{b}\underbrace{\boxed{\spadesuit}\boxed{\heartsuit}}_{0}$, renvoie $\underbrace{\boxed{?}\boxed{?}}_{a}\underbrace{\boxed{?}\boxed{?}}_{a\oplus b}\underbrace{\boxed{\spadesuit}\boxed{\heartsuit}}_{0}$

Justifier que ces protocoles ne révèlent aucune information sur a et b.

Indication : si A et R sont des variables aléatoires indépendantes à valeurs dans $\{0, 1\}$, et $\Pr(R = 0) = \frac{1}{2}$, montrer que $\Pr(A \oplus R = 0) = \frac{1}{2}$.

Solution : Le protocole COPY est un cas particulier du protocole XOR.

Une fois appliqué le protocole de la question 2, il faut révéler certaines cartes. Révéler r ne nous avance en rien. On va plutôt révéler les deux premières cartes : on obtient la valeur de $(a \oplus r)$.

Si $a \oplus r = 0$ alors on sait que les cartes face cachée encodent $b \oplus a, a$. Il suffit de les permuter pour obtenir les encodages de $(a, a \oplus b)$ aux positions voulues.

Si $a \oplus r = 1$ alors les cartes face cachée encodent $\overline{b \oplus a}, \overline{a}$. On applique deux fois un NOT (échanger les cartes), et on effectue la même permutation.

La seule valeur observée au cours du protocole est $a \oplus r$. Or on a :

$$\begin{aligned}\Pr(A \oplus R = 0) &= \Pr(R = 0 \wedge A = 0) + \Pr(R = 1 \wedge A = 1) \\ &= \frac{1}{2}\Pr(A = 0) + \frac{1}{2}\Pr(A = 1) = \frac{1}{2}\ .\end{aligned}$$

Par indépendance de R. On peut même en déduire que $\Pr(A = 0 | A \oplus R = 0) = \Pr(A = 0 | A \oplus R = 1) = \Pr(A = 0)$. Ce qui formalise le fait que « observer $a \oplus r$ ne donne aucune information sur la valeur de a » (ici a est une variable aléatoire Booléenne, et r une variable aléatoire Booléenne uniforme indépendante de a).

Enfin, il en est de même pour b, puisque le bit observé est indépendant de b.

Question 4. Sur une entrée $\underbrace{\boxed{?}\boxed{?}}_{a}\boxed{\spadesuit}\boxed{\heartsuit}\underbrace{\boxed{?}\boxed{?}}_{b}$ encodant deux bits (a, b) et deux cartes additionnelles $\boxed{\spadesuit}\boxed{\heartsuit}$, on effectue les opérations suivantes :

— Retourner les cartes visibles : on obtient alors $\underbrace{\boxed{?}\boxed{?}}_{a}\underbrace{\boxed{?}\boxed{?}}_{0}\underbrace{\boxed{?}\boxed{?}}_{b}$

— Effectuer la permutation suivante : $\overset{1\ 2\ 3\ 4\ 5\ 6}{\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}} \to \overset{1\ 3\ 4\ 2\ 5\ 6}{\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}}$

— Effectuer une coupe (avec probabilité 1/2) : $\overset{1\ 2\ 3\ 4\ 5\ 6}{\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}} \to \overset{4\ 5\ 6\ 1\ 2\ 3}{\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}}$

— Effectuer la permutation suivante : $\overset{1\ 2\ 3\ 4\ 5\ 6}{\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}} \to \overset{1\ 4\ 2\ 3\ 5\ 6}{\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}}$

Compléter ce protocole et montrer qu'il permet de renvoyer $a \wedge b$.

Solution : On procède comme à la question précédente.

Dans le cas où la coupe a lieu :

$$\boxed{a_0}\boxed{a_1}\boxed{\spadesuit}\boxed{\heartsuit}\boxed{b_0}\boxed{b_1} \to \boxed{a_0}\boxed{\spadesuit}\boxed{\heartsuit}\boxed{a_1}\boxed{b_0}\boxed{b_1} \to \boxed{a_1}\boxed{b_0}\boxed{b_1}\boxed{a_0}\boxed{\spadesuit}\boxed{\heartsuit} \to \underbrace{\boxed{a_1}\boxed{a_0}}_{a\oplus 1}\underbrace{\boxed{b_0}\boxed{b_1}}_{b}\underbrace{\boxed{\spadesuit}\boxed{\heartsuit}}_{0}$$

Dans le cas où elle n'a pas lieu :

$$\boxed{a_0}\boxed{a_1}\boxed{\spadesuit}\boxed{\heartsuit}\boxed{b_0}\boxed{b_1} \to \boxed{a_0}\boxed{\spadesuit}\boxed{\heartsuit}\boxed{a_1}\boxed{b_0}\boxed{b_1} \to \underbrace{\boxed{a_0}\boxed{a_1}}_{a}\underbrace{\boxed{\spadesuit}\boxed{\heartsuit}}_{0}\underbrace{\boxed{b_0}\boxed{b_1}}_{b}$$

On remarque que les quatre cartes de droite encodent $b \wedge r, b \wedge \overline{r}$ où r est un bit aléatoire uniforme, qui vaut 1 dans le premier cas et 0 dans le deuxième. Dans le cas général, on a donc produit un encodage de : $a \oplus r, r \wedge b, \overline{r} \wedge b$.

Pour la suite du protocole, on retourne les deux premières cartes. Si on lit 0, alors on a obtenu $0|a \wedge b|\overline{a} \wedge b$, et $a \wedge b$ se trouve dans les cartes 3 et 4. Si on lit 1, alors $a \wedge b$ se trouve dans les cartes 5 et 6. On permute donc en fonction pour que le résultat soit toujours au même endroit. Comme précédemment, on n'observe que $a \oplus r$, qui est indépendant de a, et on ne révèle donc aucune information sur a ni b.

Question 5. Compléter le protocole précédent pour qu'il renvoie : $\boxed{\spadesuit}\boxed{\heartsuit}\underbrace{\boxed{?}\boxed{?}}_{b}\underbrace{\boxed{?}\boxed{?}}_{a\wedge b}$

Solution : À la fin du protocole AND précédent on a : $\underbrace{\boxed{\spadesuit}\boxed{\heartsuit}}_{0}\underbrace{\boxed{?}\boxed{?}}_{a\wedge b}\underbrace{\boxed{?}\boxed{?}}_{\overline{a}\wedge b}$. Maintenant le problème est de retransformer $\overline{a} \wedge b$ en b.

Pour ce faire, on applique un XOR, qui va XORer la valeur $a \wedge b$ dans la valeur $\overline{a} \wedge b$. On a bien : $(a \wedge b) \oplus (\overline{a} \wedge b) = (a \oplus \overline{a}) \wedge b = 1 \wedge b = b$.

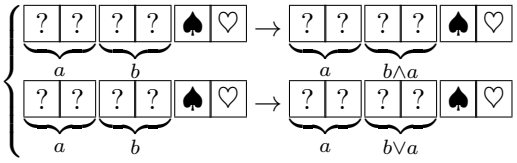
Question 6. Montrer que pour toute fonction booléenne $f : \{0,1\}^n \to \{0,1\}$ il existe un protocole permettant de la calculer. De combien de cartes auxiliaires a-t-on besoin ?

Solution : Toute fonction Booléenne peut s'écrire sous la forme d'une conjonction de clauses. Notons donc :

$$f(b_0, \ldots, b_{n-1}) = c_1(b_0, \ldots, b_{n-1}) \wedge \ldots \wedge c_{2^n}(b_0, \ldots, b_{n-1})$$

où chaque c_i est une disjonction contenant les b_i ou leur négation.

Pour commencer, notons qu'on peut construire un protocole pour OR en adaptant notre protocole pour AND (il suffit de composer avec des NOT en entrée et en sortie). On peut donc calculer :



Un début de solution est de remarquer qu'il existe bien un tel protocole obtenu en copiant autant de fois que nécessaire les entrées, et en appliquant une série de AND et de OR. Il faut maintenant minimiser le nombre de cartes auxiliaires.

On commence par montrer comment calculer une clause $c_i(b_0, \dots, b_{n-1})$ en utilisant quatre cartes auxiliaires (dont deux pour la sortie, et deux qu'on récupère ensuite) : on commence par faire une copie de b_0 dans la sortie, et ensuite une séquence de OR avec les valeurs de b_i ou $\overline{b_i}$ dont on a besoin.

On ajoute deux cartes auxiliaires supplémentaires pour stocker la sortie de f. Pour chaque clause, on calcule séparément sa valeur, ensuite on applique un protocole AND avec cette valeur et la sortie de f. Enfin, étape très importante, il faut effacer la valeur de la clause que l'on vient de calculer (afin de pouvoir réutiliser ces cartes). Pour ce faire, différentes méthodes sont possibles. On peut par exemple faire un OR avec la négation d'un des littéraux de la clause. Ou bien, approche assez élégante, appliquer une coupe aléatoire qui va échanger les deux cartes, avant de retourner celles-ci face visible.

À la fin du protocole, non seulement on récupère la sortie de f, mais aussi les $b_0, \dots, b_{n-1}$ inchangés, et les 4 cartes utilisées durant le calcul. Le protocole utilise $2n + 6$ cartes au total. Il ne révèle aucune information sur les bits d'entrée, car c'est le cas de tous les protocoles intermédiaires pour AND / OR / NOT / effacement.

Question 7. Donner un protocole de « random flip » qui sur une entrée $(b_0, \dots, b_{n-1})$ (face verso), renvoie $(b_0 \oplus r, \dots, b_{n-1} \oplus r)$ où r est une valeur aléatoire uniforme (face verso). En déduire un protocole à $2n$ cartes calculant la fonction « égalité » :

$$f(b_0, \dots, b_{n-1}) = \begin{cases} 1 \text{ si } b_0 = b_1 \dots = b_{n-1} \\ 0 \text{ sinon} \end{cases} \tag{1}$$

Solution : Le protocole de random flip est simplement une permutation qui regroupe les indices impairs (resp. pairs), suivie d'une coupe en deux paquets de n cartes, suivie de la même permutation. Dans un cas toutes les paires de cartes sont permutées localement (d'où le flip), dans l'autre cas toutes les paires sont laissées inchangées.

Sur l'entrée $(b_0, \dots, b_{n-1})$ on applique le random flip. On retourne les deux premières cartes, et on observe ainsi $b_0 \oplus r$.

Si $b_0 \oplus r = 0$ alors $f(b_0, \dots, b_{n-1}) = 1 \iff \forall i \geq 1, b_i \oplus r = 0 \iff \bigwedge \overline{b_i \oplus r} = 1$. Il suffit donc d'appliquer un NOT aux bits restants et de calculer leur AND (à l'aide des deux cartes révélées comme auxiliaires).

Si $b_0 \oplus r = 1$ alors $f(b_0, \dots, b_{n-1}) = 1 \iff \forall i \geq 1, b_i \oplus r = 1 \iff \bigwedge b_i \oplus r = 1$. On utilise le même principe, mais sans les NOTs.

> Une fonction Booléenne est dite *symétrique* si $f(\pi(b_0, \dots, b_{n-1})) = f(b_0, \dots, b_{n-1})$ pour toute permutation π de $\{0, \dots, n-1\}$.

Question 8. Donner un protocole calculant sur l'entrée $(b_0, \dots, b_{n-1})$, un encodage de $\sum b_i$.

En déduire un protocole pour les fonctions symétriques avec seulement 2 cartes additionnelles pour $n \geq 4$.

Solution : On va utiliser le fait qu'une fonction symétrique est de la forme : $f(b_0, \ldots, b_{n-1}) = g(\sum b_i)$ où $\sum b_i$ est un entier naturel, que l'on peut interpréter comme un vecteur Booléen de taille $\lceil \log_2 n \rceil$, et g est une autre fonction Booléenne. En effet, si $(b_0, \ldots, b_{n-1})$ et $(b'_0, \ldots, b'_{n-1})$ sont deux vecteurs ayant le même nombre de "1" (donc la même somme), il existe une permutation π qui transforme l'un en l'autre, donc $f(b_0, \ldots, b_{n-1}) = f(b'_0, \ldots, b'_{n-1})$.

On réduit donc le problème à l'implémentation d'un protocole de somme (qui produit l'encodage de $\sum b_i$), suivi du protocole pour g. D'après la question 6, le protocole pour g n'a besoin que de 6 cartes additionnelles. Comme $n \geq 4$, on a $n - \lceil \log_2 n \rceil \geq 2$, et par conséquent après avoir calculé la somme, on pourra récupérer au moins 4 cartes supplémentaires, ce qui garantit qu'on peut calculer g.

Pour le protocole de somme : il faut implémenter un additionneur. Il suffit pour cela de faire un « half adder » qui de deux bits a, b, calcule $(a \wedge b, a \oplus b)$. En effet, supposons qu'on ait un registre qui encode un entier i sur m bits, suivi d'un Booléen a. Sur cette entrée $i_{m-1} \ldots i_0 a$ où i_0 est le bit de poids faible de i, on exécute :

$$i_{m-1} \ldots i_0 a \to i_{m-1} \ldots i_1 (i_0 \oplus a)(a \wedge i_0)$$

et ensuite on additionne $a \wedge i_0$ (la retenue) à $i_{m-1} \ldots i_1$, de manière récursive, pour obtenir les bits suivants. Ceci nous donne l'addition d'un bit à un entier. On peut donc ensuite ajouter tous les bits un par un à notre registre de résultat. Ce n'est pas la méthode la plus efficace en nombre d'étapes, mais elle suffit largement dans la mesure où on s'intéresse surtout au nombre de cartes.

Le « half adder » peut être implémenté en utilisant deux cartes auxiliaires, et en composant des protocoles XOR et AND. On peut remarquer que ce n'est pas difficile avec quatre cartes auxiliaires. Nous montrons ci-dessous comment se restreindre à deux cartes.

Dans une première étape on calcule $(\overline{a} \wedge \overline{b}, a \wedge b, 0)$ (sur les 6 cartes d'entrée). Pour ce faire :

$$(a, b, 0) \xmapsto{\text{XOR}} (a, a \oplus b, 0) \mapsto (a, \overline{a \oplus b}, 0)$$

Puis :

$$(a, \overline{a \oplus b}, 0) \xmapsto{\text{AND (partiel)}} (a \wedge \overline{a \oplus b}, \overline{a} \wedge \overline{a \oplus b}, 0)$$

On montre ensuite que $a \wedge \overline{a \oplus b} = a \wedge b$ et $\overline{a} \wedge \overline{a \oplus b} = \overline{a} \wedge \overline{b}$.

Enfin :

$$(\overline{a} \wedge \overline{b}, a \wedge b, 0) \xmapsto{\text{XOR}} ((\overline{a} \wedge \overline{b}) \oplus (a \wedge b), (a \wedge b), 0) = (\overline{a \oplus b}, a \wedge b, 0) \xmapsto{\text{NOT}} (a \oplus b, a \wedge b, 0)$$

Ce qui termine le protocole.

Graphes décomposables en cycles

On note $[\![a, b]\!] = \{a, a+1, \ldots, b\}$ un intervalle entier, et $[a, b]$ un intervalle réel (fermé).
On note $|S|$ la cardinalité d'un ensemble S.
Graphe. Dans tout l'énoncé, on considère des graphes orientés. Formellement, un graphe est une paire (V, E) où V est un ensemble non vide de sommets, et $E \subseteq V \times V$ est un ensemble d'arêtes orientées. Un sommet v peut avoir une arête (v, v) de lui-même vers lui-même, qu'on appelle une boucle.
Sous-graphe. $G = (V, E)$ est un *sous-graphe* de $G' = (V', E')$ si et seulement si $V = V'$ et $E \subseteq E'$.
Voisinage. Soit $G = (V, E)$ un graphe, et $U \subseteq V$. On appelle *voisinage* de U l'ensemble $\mathsf{v}_G(U)$ des sommets atteignables depuis U en suivant une arête de E : $\mathsf{v}_G(U) = \{v \in V : \exists u \in U, (u, v) \in E\}$.
Graphe ouvert. Un graphe $G = (V, E)$ est dit *ouvert* si $\forall U \subseteq V, |\mathsf{v}_G(U)| \geq |U|$.
Cycle. Un *cycle* d'un graphe (V, E) est une suite de sommets deux à deux distincts $(v_1, \ldots, v_k)$ telle que $\forall i \in [\![1, k-1]\!], (v_i, v_{i+1}) \in E$, et $(v_k, v_1) \in E$. On note $v_1 \to v_2 \to \cdots \to v_k \to v_1$. Le cas $k = 1$ correspond à une boucle, qui est considérée comme un cycle.
Graphe cyclique. Un graphe G est *cyclique* s'il consiste en un unique cycle qui passe par tous les sommets : G est de la forme $(\{v_1, \ldots, v_n\}, \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n), (v_n, v_1)\})$.
Décomposition en cycles. Soit $G = (V, E)$ un graphe. Une *décomposition en cycles* de G est un sous-graphe G' de G tel que toutes les composantes connexes de G' sont cycliques.

Question 1. Montrer que tout graphe ouvert contient au moins un cycle.

Solution : Comme le graphe est ouvert, en particulier tout sommet v a un degré sortant au moins 1 (car $|\mathsf{v}_G(\{v\})| \geq |\{v\}| = 1$). Il suffit de partir d'un sommet quelconque, et de suivre une arête sortante jusqu'au sommet suivant, puis réitérer. Lorsqu'on finit par repasser par un sommet déjà visité, on a parcouru un cycle.

Question 2. Montrer que tout graphe qui admet une décomposition en cycles est ouvert.

Solution : Soit G un graphe et G' sa décomposition en cycles. Sur $G' = (V, E')$, on peut observer que E' définit une relation bijective $V \to V$, puisque chaque sommet a un prédécesseur et un successeur. En particulier, dans G', $|\mathsf{v}_{G'}(U)| = |U|$ pour tout $U \subseteq V$. Or le voisinage dans G contient le voisinage dans G'. Donc dans G, $|\mathsf{v}_G(U)| \geq |U|$ pour tout $U \subseteq V$.

Question 3.

1. Soit $G = (V, E)$ un graphe. Montrer que la fonction $U \mapsto |\mathsf{v}_G(U)|$ est *sous-modulaire*, c'est-à-dire :
$$\forall X \subseteq V, \forall Y \subseteq V, \quad |\mathsf{v}_G(X)| + |\mathsf{v}_G(Y)| \geq |\mathsf{v}_G(X \cup Y)| + |\mathsf{v}_G(X \cap Y)|.$$
2. On dit que $X \subseteq V$ est *étroit* si $|\mathsf{v}_G(X)| = |X|$. Supposons que G est ouvert. Déduire de l'inégalité précédente que si X et Y sont étroits, alors $X \cup Y$ et $X \cap Y$ le sont aussi.

Solution :

1. Le résultat se montre bien en considérant chaque cas pour $v \in V$:
   — Si v est voisin de X (i.e. $v \in \mathsf{v}_G(X)$) mais pas voisin de Y, alors il est compté une fois à gauche et une fois à droite dans l'inégalité de la question.
   — De même si v est voisin de Y mais pas de X.
   — Si v est voisin de $X \cap Y$, il est compté deux fois de chaque côté.

— Si v est voisin de X et de Y mais pas de $X \cap Y$, il est compté deux fois à gauche mais une seule fois à droite.
— Si v n'est voisin ni de X ni de Y il est compté zéro fois de chaque côté.

Alternativement, une manière plus abstraite d'écrire la preuve est d'observer :

$$\mathsf{v}_G(X \cup Y) = \mathsf{v}_G(X) \cup \mathsf{v}_G(Y)$$
$$\mathsf{v}_G(X \cap Y) \subseteq \mathsf{v}_G(X) \cap \mathsf{v}_G(Y)$$

L'inégalité en bas vient de ce qu'il peut y avoir des voisins communs à X et à Y sans qu'ils soient voisins de $X \cap Y$, comme précédemment. De là, on déduit :

$$\begin{aligned}|\mathsf{v}_G(X)| + |\mathsf{v}_G(Y)| &= |\mathsf{v}_G(X) \cup \mathsf{v}_G(Y)| + |\mathsf{v}_G(X) \cap \mathsf{v}_G(Y)| \\ &\geq |\mathsf{v}_G(X \cup Y)| + |\mathsf{v}_G(X \cap Y)|.\end{aligned}$$

2. En utilisant successivement le caractère étroit de X et Y, puis la sous-modularité de $|\mathsf{v}_G(\cdot)|$, puis le caractère ouvert de G, on a :

$$|X| + |Y| = |\mathsf{v}_G(X)| + |\mathsf{v}_G(Y)| \geq |\mathsf{v}_G(X \cup Y)| + |\mathsf{v}_G(X \cap Y)| \geq |X \cup Y| + |X \cap Y| = |X| + |Y|.$$

Toutes les inégalités sont donc des égalités, ce qui force $|\mathsf{v}_G(X \cup Y)| = |X \cup Y|$ et $|\mathsf{v}_G(X \cap Y)| = |X \cap Y|$.

Question 4. Soit $G = (V, E)$ un graphe ouvert avec un sommet $v \in V$ qui a un degré sortant 2 : c'est-à-dire qu'il existe $x \neq y \in V$ tels que $(v, x) \in E$ et $(v, y) \in E$. Montrer qu'au moins un des graphes $G_x = (V, E \setminus \{(v, x)\})$ ou $G_y = (V, E \setminus \{(v, y)\})$ est ouvert.

Solution : Il peut être difficile de trouver le « bon » ensemble sur lequel raisonner. Supposons par l'absurde que ni G_x ni G_y n'est ouvert. Alors il existe $X \subseteq V$ tel que $|\mathsf{v}_{G_x}(X)| < |X|$.

— Première observation : $v \in X$, sinon X aurait le même voisinage dans G que dans G_x, et serait donc un contre-exemple au caractère ouvert de G.
— Deuxième observation : v est le seul prédécesseur de x dans X (i.e. le seul sommet w de X tel que $(w, x) \in E$). Sinon, à nouveau, X aurait le même voisinage dans G que dans G_x.
— Troisième observation : X est étroit (dans G) puisqu'il a au plus un voisin de plus dans G que dans G_x.

Symétriquement, il existe $Y \subseteq V$ avec les trois mêmes propriétés (en remplaçant x par y et X par Y ci-dessus).

Par la question 3, $X \cap Y$ est étroit. Si on enlève v de $X \cap Y$, sa cardinalité diminue de 1 ; mais, par les observations précédentes, ni x ni y ne peuvent être dans $\mathsf{v}_G(X \cap Y \setminus \{v\})$, donc :

$$|\mathsf{v}_G(X \cap Y \setminus \{v\})| \leq |\mathsf{v}_G(X \cap Y)| - 2 = |X \cap Y| - 2 < |X \cap Y| - 1 = |(X \cap Y) \setminus \{v\}|.$$

L'ensemble $X \cap Y \setminus \{v\}$ témoigne du fait que G n'est pas ouvert, une contradiction.

Question 5. En déduire la réciproque de la question 2 : tout graphe ouvert admet une décomposition en cycles.

Solution : Soit $G = (V, E)$ un graphe ouvert. Par la question précédente, tant que G possède un sommet de degré sortant au moins 2, on peut supprimer une arête en restant ouvert. On continue jusqu'à ce qu'il n'y ait plus de sommet de degré sortant 2. On arrive finalement à un sous-graphe ouvert $G' = (V, E')$ dont tous les sommets sont de degré sortant 1. (Variante équivalente : on prend pour G' un élément minimal pour l'ordre partiel « A est sous-graphe de B », parmi les sous-graphes ouverts de G.)

Soit $v \in V$, et $A = \{a \in V : (a, v) \in E'\}$ ses prédécesseurs dans G'. Puisque les degrés sortants sont tous 1, $\mathsf{v}_{G'}(A) = \{v\}$, donc comme G' est ouvert, $|A| \leq 1$. Donc tous les degrés entrants dans G' sont au plus 1. Comme

les sommes des degrés entrants et des degrés sortants sur V entier doivent être égales, on déduit que tous les degrés entrants sont en fait exactement 1.

Il reste à observer qu'un graphe dont tous les degrés entrants et sortants sont égaux à 1 est en fait une union disjointe de cycles (autrement dit, ses composantes connexes sont cycliques). En effet, en partant d'un sommet arbitraire et en suivant successivement les arêtes sortantes, la seule possibilité est qu'on finit par revenir au sommet de départ. On conclut que G' est une décomposition en cycles de G.

Matrice de diffusion. Une matrice $M = (m_{i,j})_{i,j\in[\![1,n]\!]} \in [0,1]^{n\times n}$ est dite *matrice de diffusion* si la somme de ses entrées sur chaque ligne et chaque colonne est égale à 1 :

$$\forall i \in [\![1,n]\!],\ \sum_{j\in[\![1,n]\!]} m_{i,j} = \sum_{j\in[\![1,n]\!]} m_{j,i} = 1.$$

Question 6. Soit M une matrice de diffusion. Montrer qu'il existe une permutation π de $[\![1,n]\!]$ telle que $\forall i \in [\![1,n]\!], m_{i,\pi(i)} > 0$.

Solution : Soit $V = [\![1,n]\!]$, et $E = \{(i,j) : m_{i,j} > 0\}$. On considère le graphe $G = (V,E)$.

Pour utiliser la question 5, montrons que G est ouvert. Soit $U \leq V$. Soit $U' = \mathsf{v}_G(U)$. Par construction, pour $i \in U$ et $j \in V \setminus U'$, $m_{i,j} = 0$. L'idée est que si on regarde la sous-matrice avec indices dans $U \times U'$, en sommant toutes ses entrées horizontalement, c'est-à-dire ligne par ligne, on trouve $|U|$ (parce que les entrées hors des colonnes U' sur les lignes U sont nulles) ; et en sommant verticalement on trouve au plus $|U'|$ (parce qu'on somme une matrice de diffusion sur quelque chose d'inclus dans $|U'|$ colonnes). Comme les deux sommes doivent être égales, $|U| \leq |U'|$.

Version formelle de la même chose :

$$\sum_{i\in U}\sum_{j\in U'} m_{i,j} = \sum_{i\in U}\sum_{j\in[\![1,n]\!]} m_{i,j} = |U|\cdot 1 = |U|$$
$$\sum_{j\in U'}\sum_{i\in U} m_{i,j} \leq \sum_{j\in U'}\sum_{i\in[\![1,n]\!]} m_{i,j} = |U'|\cdot 1 = |U'|.$$

L'égalité entre les premiers termes de chaque ligne donne $|U| \leq |U'|$.

Puisque G est ouvert, par la question 5, il contient un sous-graphe (V,E') dont toutes les composantes connexes sont cycliques. Comme observé dans la question 2, E' peut être vu comme le graphe fonctionnel d'une bijection $\pi : V \to V$, en posant $E' = \{(v,\pi(v)) : v \in V\}$. (Les cycles de E' correspondent d'ailleurs à la décomposition en cycles de π.) Par définition de E, $m_{i,j} > 0$ pour tout $(i,j) \in E \supseteq E'$, en particulier $m_{i,\pi(i)} > 0$ pour tout i.

Question 7. Soit M une matrice de diffusion. Montrer qu'il existe un entier $k \in \mathbb{N}^*$, un vecteur $(\lambda_1,\dots,\lambda_k) \in [0,1]^k$ avec $\sum_{i\in[\![1,k]\!]} \lambda_i = 1$, et des matrices de permutation $P_1,\dots,P_k$, tels que $M = \sum_{i\in[\![1,k]\!]} \lambda_i P_i$.

Solution : Posons $M_1 = M$. Par la question précédente, il existe une permutation π telle que $m_{i,\pi(i)} > 0$ pour tout i. Soit $\lambda_1 = \min_{i\in[\![1,n]\!]} m_{i,\pi(i)}$. Soit P_1 la matrice de permutation associée à π.

Si $\lambda_1 = 1$, nécessairement $M_1 = P_1$, et on a fini. Sinon, on pose $M_2 = M_1 - \lambda_1 P_1$. Remarquons que M_2 est à valeur dans $\mathbb{R}_+$, et ses lignes et colonnes se somment toutes à $1-\lambda_1$. À une constante $1-\lambda_1$ près, c'est donc une matrice de diffusion, et on peut de nouveau appliquer la question 6. (Alternativement, on peut poser $M_2' = 1/(1-\lambda_1)M_2$ pour se ramener vraiment à une matrice de diffusion, mais ce n'est pas nécessaire, et implique de garder trace des constantes multiplicatives pour les compenser à la fin.) On répète le même processus en partant de M_2, pour construire λ_2, P_2 etc.

Cet algorithme termine en $O(n^2)$ étapes, parce que M_2 contient au moins une entrée nulle de plus que M_1 : celle de coordonnée $(j, \pi(j))$ pour $j = \text{argmin}_{i \in [\![1,n]\!]} m_{i,\pi(i)}$. Lorsque l'algorithme termine, on a obtenu les λ_i et P_i souhaités. Le fait que $\sum \lambda_i = 1$ peut se montrer via un invariant adéquat le long de la récurrence, mais c'est aussi impliqué de toute façon par le fait que la somme le long d'une ligne ou colonne de $\sum \lambda_i P_i$ est égale à $\sum \lambda_i$.

Dans toute la suite de l'énoncé, on fixe $0 < m \leq n$ deux entiers, et :

$$S = \{(s_1, \ldots, s_n) \in [0,1]^n : \sum_{i \in [\![1,n]\!]} s_i = m\} \qquad \hat{s} = (\underbrace{1, \ldots, 1}_{m}, \underbrace{0, \ldots, 0}_{n-m}) \in S.$$

Question 8. Montrer que pour tout $s \in S$, il existe une matrice de diffusion M telle que $s = M\hat{s}$.

Solution : On peut se contenter d'au plus deux valeurs distinctes sur chaque ligne de M.

Pour la i-ième ligne de M, on prend la ligne dont les m premières entrées sont s_i/m, et (si $n > m$) les $n - m$ dernières entrées sont $(1 - s_i)/(n - m)$. On voit que $M\hat{s} = s$, et que chaque ligne de M se somme à 1.

Il reste à montrer que les colonnes se somment à 1. Pour $j \leq m$, la j-ième colonne se somme à $\sum s_i/m = 1$. Si $n > m$, pour $n - m < j \leq n$, la j-ième colonne se somme à $\sum(1 - s_i)/(n - m) = (n - m)/(n - m) = 1$.

Convexité. Une fonction $f : \mathbb{R}^n \to \mathbb{R}$ est *convexe* si pour tout k-uplet $w_1, \ldots, w_k$ de vecteurs dans $\mathbb{R}^n$, et pour tout $(\lambda_1, \ldots, \lambda_k) \in [0,1]^k$ tel que $\sum_{i \in [\![1,k]\!]} \lambda_i = 1$:

$$f\Big(\sum_{i \in [\![1,k]\!]} \lambda_i w_i\Big) \leq \sum_{i \in [\![1,k]\!]} \lambda_i f(w_i).$$

Symétrie. Une fonction $f : \mathbb{R}^n \to \mathbb{R}$ est *symétrique* si pour tout vecteur $(v_1, \ldots, v_n) \in \mathbb{R}^n$, et pour toute permutation π de $[\![1, n]\!]$:

$$f(v_{\pi(1)}, \ldots, v_{\pi(n)}) = f(v_1, \ldots, v_n).$$

Question 9. Déduire des questions précédentes que pour toute fonction $f : \mathbb{R}^n \to \mathbb{R}$ symétrique et convexe :

$$\max_{s \in S} f(s) = f(\hat{s}).$$

Solution : Soit $s \in S$ et M tel que $s = M\hat{s}$. Par la question 6, il existe une combinaison convexe de matrices de permutations telle que $M = \sum \lambda_i P_i$. En utilisant la convexité puis la symétrie de f, on a :

$$f(s) = f(M\hat{s}) = f\Big(\sum_{i \in [\![1,n]\!]} \lambda_i P_i \hat{s}\Big) \leq \sum_{i \in [\![1,n]\!]} \lambda_i f(P_i \hat{s}) = \sum_{i \in [\![1,n]\!]} \lambda_i f(\hat{s}) = f(\hat{s}).$$

Question 10. Si on assigne m objets dans m zones mémoires, en tirant la zone choisie uniformément aléatoirement et indépendamment pour chaque objet, un résultat classique (utilisé dans des constructions de tables de hachage), est que l'espérance du nombre d'objets contenus dans la zone mémoire qui reçoit le plus d'objets est $O(\log m)$.

Supposons maintenant qu'on assigne $n \geq m$ objets ayant chacun une taille $t_i \in [0,1]$ pour $i \in [\![1, n]\!]$, toujours en envoyant chaque objet dans une zone mémoire parmi m, choisie de manière uniforme et indépendante pour chaque objet. On suppose que la taille totale assignée est $\sum_{i \in [\![1,n]\!]} t_i = m$. À chaque zone mémoire i, on associe sa charge X_i, qui est la somme des tailles des objets assignés à la zone i.

Montrer (sans faire de calcul) que l'espérance de la charge maximale $X = \max_{i \in [\![1,m]\!]} X_i$ est $O(\log m)$.

Indication : on admettra que $f : (v_1, \ldots, v_n) \mapsto \mathbb{E}(X)$ est convexe.

Solution : Les vecteurs $(t_1, \ldots, t_n)$ sont dans $[0,1]^n$ et doivent satisfaire $\sum t_i = m$, donc ils vivent dans S. D'autre part, on admet que f est convexe, et elle est clairement symétrique, donc on peut appliquer la question précédente pour déduire que f atteint son maximum en $\hat{s}$. Or le cas $(t_1, \ldots, t_n) = \hat{s}$ revient à assigner m objets de même taille 1 dans m zones mémoire : ce qui est exactement le cas classique non pondéré, pour lequel l'énoncé dit que l'espérance de la charge maximale est $O(\log m)$. C'est tout : on a conclu.

Montrer que $f : (v_1, \ldots, v_n) \mapsto \mathbb{E}(X)$ est convexe est hors-sujet. Cela peut se montrer par des règles de préservation de fonctions convexes (par somme, et composition par une fonction convexe croissante), en observant qu'une espérance sur un ensemble fini n'est pas autre chose qu'une somme, à multiplication par un scalaire près.

Fibrations de graphes

Dans ce sujet, on appelle *graphe* un tuple $(S, A, \text{cible}, \text{source})$ où S et A sont des ensembles finis et $\text{cible} : A \to S$ et $\text{source} : A \to S$ sont des applications. Les éléments de S sont les *sommets* du graphe, et les éléments de A sont les *arcs*. On note qu'il peut y avoir plusieurs arcs entre deux sommets.
Si $G = (S, A, \text{cible}, \text{source})$ et $G' = (S', A', \text{cible}', \text{source}')$ sont des graphes, un *homomorphisme* $f : G \to G'$ est une paire de fonctions $f_0 : S \to S'$ et $f_1 : A \to A'$ satisfaisant les deux propriétés suivantes pour tout arc $a \in A$:

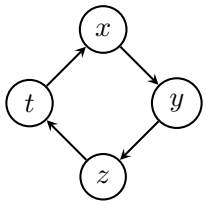
$$\text{cible}'(f_1(a)) = f_0(\text{cible}(a)) \qquad \text{et} \qquad \text{source}'(f_1(a)) = f_0(\text{source}(a)).$$

Un *arc entrant* d'un sommet $x \in S$ est un arc $a \in A$ telle que $\text{cible}(a) = x$. On dit qu'un homomorphisme $f : G \to G'$ est une *fibration* si, pour tout $x \in S$ et pour tout arc entrant a' de $f_0(x)$, il existe un unique arc entrant a de x telle que $f_1(a) = a'$.

Question 1. Donner un exemple d'homomorphisme de graphes qui n'est pas une fibration.

Solution : L'exemple le plus simple : l'inclusion du graphe $\{b\}$ (un seul sommet, pas d'arête) dans le graphe $\{a \to b\}$ (deux sommets, une arête).

Question 2. Donner un exemple de fibration $f : G \to G'$ où G est le graphe ci-dessous et G' n'est pas isomorphe à G.



Solution : Deux possibilités (parmi d'autres) :

- on peut faire l'union disjointe de G avec n'importe quel autre graphe H et regarder l'inclusion $G \to G + H$.
- on peut regarder une fonction qui replie G vers un graphe plus petit, attention que la propriété de fibration soit bien respectée (chaque flèche entrante vers l'image d'un sommet a un unique relèvement).

Question 3. Soit $f : G \to G'$ une fibration, où $G = (S, A, \text{cible}, \text{source})$. Soient $x, y \in S$ des sommets de G qui ont la même image dans G' : $f_0(x) = f_0(y)$.

Montrer qu'il existe une bijection φ de l'ensemble $A_{\to x}$ des arcs entrants de x vers l'ensemble $A_{\to y}$ des arcs entrants de y, telle que, pour tout $a \in A_{\to x}$, $f_0(\text{source}(a)) = f_0(\text{source}(\varphi(a)))$.

Solution : Il y a une fonction $A_{\to x} \to A_{\to f(x)}$ donnée par l'action de l'homomorphisme f. La propriété de fibration dit exactement que cette fonction a une inverse.

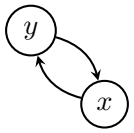
Puisque $f(x) = f(y)$, on a juste à composer les deux bijections, pour obtenir $A_{\to x} \cong A_{\to f(x)} = A_{\to f(y)} \cong A_{\to y}$.

On modélise un réseau distribué à l'aide d'un graphe $(S, A, \text{cible}, \text{source})$. Les sommets représentent les agents du réseau. On fixe un ensemble Q et un état initial $q_0(x)$ pour chaque sommet $x \in S$. Chaque arc $x \xrightarrow{a} y$ représente un canal par lequel un agent x communique son état à l'agent y.
Tous les agents sont régis par la même fonction d'évolution $\delta : Q \times \mathcal{M}(Q) \to Q$, qui met à jour l'état d'un agent en fonction du multi-ensemble d'états qu'il reçoit. On a noté ici $\mathcal{M}(Q)$ l'ensemble des multi-ensembles finis d'éléments de Q.
(Pour rappel, un multi-ensemble est une collection non-ordonnée où un même élément peut apparaitre plusieurs fois. On utilise la notation $\{\!|\cdot|\!\}$ pour les multi-ensembles : par exemple, $\{\!|1, 1, 2|\!\}$ est le multi-ensemble qui contient deux copies de 1 et une copie de 2.)
Étant donné un état initial $q_0(x) \in Q$ pour chaque sommet $x \in S$, on définit l'état $q_n(x)$ de x à l'étape $n \geq 1$ comme suit :

$$q_n(x) = \delta(q_{n-1}(x), \{\!|q_{n-1}(\text{source}(a)) \mid a \in A_{\to x}|\!\}).$$

Dans le reste du sujet, on fixe $Q = \{0, 1\}$.

Question 4. Soit G le graphe représenté ci-dessous :



Pour $q_0(x) = q_0(y) = 0$ et $\delta : Q \times \mathcal{M}(Q) \to Q$ la fonction définie par

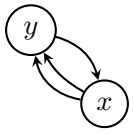
$$\delta(q, m) = \begin{cases} 0 & \text{si } q = 1, m = \{\!|1|\!\} \\ 1 & \text{si } q = 0, m = \{\!|0|\!\} \\ 0 & \text{sinon,} \end{cases}$$

donner les trois premières étapes d'évolution du réseau.

Solution : On commence avec $q_0(x) = q_0(y) = 0$. Ensuite on a $q_1(x) = \delta(0, \{\!|0|\!\}) = 1$ et de la même manière $q_1(y) = 1$. Ensuite $q_2(x) = q_2(y) = 0$, et on voit que ça boucle : $q_3(x) = q_3(y) = 1$, etc.

Pour un graphe G, on appelle *fonction d'élection* une fonction $\delta : Q \times \mathcal{M}(Q) \to Q$ avec la propriété suivante : pour tout choix $q_0 : S \to Q$ d'états initiaux, il existe $n \in \mathbb{N}$ et $x \in S$ tel que $q_n(x) = 1$ et $q_n(y) = 0$ pour tout $y \neq x$.

Question 5. Donnez une fonction d'élection pour le graphe ci-dessous.



Solution : La fonction

$$\delta(q, m) = \begin{cases} 1 & \text{si } m \text{ est un multi-ensemble à deux éléments} \\ 0 & \text{sinon} \end{cases}$$

donne toujours y élu en une étape.

Question 6. Soient $G = (S, A, \text{cible}, \text{source})$ et $G' = (S', A', \text{cible}', \text{source}')$ des graphes, et soit $f : G \to G'$ une fibration *propre*, c'est-à-dire que pour tout $x \in S'$, la fibre $f_0^{-1}\{x\}$ n'est pas un singleton. Montrer qu'il ne peut pas exister de fonction d'élection pour G.

Solution : On regarde le cas où $q_0(x) = 0$ pour tout $x \in S$ et on montre que, quelle que soit la fonction de transition δ, alors pour tout $n \in \mathbb{N}$, si deux sommets x, y de G ont la même image dans G', alors $q_n(x) = q_n(y)$.

Preuve par induction sur n : le cas de base est donné en hypothèse.

Cas inductif pour $n \geq 1$. On a $q_{n-1}(x) = q_{n-1}(x')$ par l'hypothèse d'induction, et on montre que $\{\!|q_{n-1}(\text{source}(a)) \mid a \in A_{\to x}|\!\} = \{\!|q_{n-1}(\text{source}(a)) \mid a \in A_{\to x'}|\!\}$. On rappelle qu'il existe une bijection $\varphi : A_{\to x} \to A_{\to x'}$ telle que $f_0(\text{source}(\varphi(a)) = f_0(\text{source}(a))$. On applique l'hypothèse d'induction à source(a) et source$(\varphi(a))$, pour en déduire que $q_{n-1}(\text{source}(a)) = q_{n-1}(\text{source}(\varphi(a)))$.

Maintenant, s'il y avait une fonction d'élection pour G, alors pour un certain n on aurait un seul x élu avec $q_n(x) = 1$ et $q_n(y) = 0$ pour $y \neq x$. C'est une contradiction parce que x doit avoir le même état que tous les éléments de sa fibre, et par hypothèse les fibres ne sont pas des singletons.

On considère un modèle d'exécution alternatif dans lequel les agents sont mis à jour un par un, et dans un ordre choisi de manière non-déterministe..
On fixe un état initial $q_0(x)$ pour tout $x \in S$ et, pour tout $n \geq 1$, un ordre total $<_n$ sur l'ensemble S des sommets. Pour une fonction de transition $\delta : Q \times \mathcal{M}(Q) \to Q$, l'exécution est donnée par $q_n(x) = \delta(q_{n-1}(x), m)$, où

$$m = \{\!|q_n(\text{source}(a)) \mid a \in A_{\to x} \text{ et source}(a) <_n x|\!\} \cup \{\!|q_{n-1}(\text{source}(a)) \mid a \in A_{\to x} \text{ et source}(a) \geq_n x|\!\}$$

Dans ce nouveau modèle, on dit que δ est une *fonction d'élection* si l'exécution atteint toujours un état d'élection (où il existe x avec $q_n(x) = 1$ et $q_n(y) = 0$ pour tout $y \neq x$) en temps n fini, quels que soient les états initiaux et les ordres $<_n$.

Question 7. Montrer que, si $f : G \to G'$ est une fibration propre, et si pour tout $x \in S'$ le graphe G restreint à la fibre $f_0^{-1}\{x\}$ est acyclique, alors il ne peut pas exister de fonction d'élection pour G.

Solution : On commence par choisir n'importe quel ordre $<'$ sur les sommets de G'. Ensuite, on choisit un ordre $<$ sur les sommets de G tel que :

- si $f(x) <' f(y)$ alors $x < y$.
- si $f(x) = f(y)$ (i.e. x et y sont dans la même fibre) et il y a une arête $x \to y$, alors $x > y$.

On pose $<_n := <$ pour tout $n \geq 1$.

Ensuite, on fait exactement la même induction que la question précédente, pour montrer que deux sommets dans la même fibre ont forcément toujours le même état à chaque étape.

Multiplication scalaire - forme non-adjacente

Soit $(G, +)$ un groupe abélien. Dans ce sujet, nous nous intéressons à calculer efficacement $k \cdot P$ où $k \in \mathbb{Z}$ et $P \in G$. Nous évaluerons la complexité de nos algorithmes selon deux paramètres : le nombre d'additions dans G et le nombre de doublements, i.e., $2 \cdot P$. Le calcul d'un inverse est supposé négligeable.
Un algorithme nécessitant m additions et n doublements aura une complexité notée $m\mathsf{A} + n\mathsf{D}$. Nous supposerons que A est significativement plus coûteux que D.

Pour ce faire, nous ne représenterons pas les entiers k sous leur forme binaire, mais sous leur forme NAF (non-adjacent form).
Définition. Une *forme non-adjacente* (NAF) de longueur ℓ d'un entier positif k est une expression $k = \Sigma_{i=0}^{\ell-1} k_i 2^i$ où :

- (1) $k_i \in \{-1, 0, 1\}$,
- (2) $k_{\ell-1} \neq 0$,
- (3) pour tout $i \in \{0, \ldots, \ell - 1\}$, $k_i \neq 0 \Rightarrow k_{i+1} = 0$, i.e., il n'y a pas deux chiffres consécutifs non-nuls.

Lorsque cela n'ajoute pas d'ambiguité, nous pourrons écrire : $k = \langle k_{\ell-1}, \ldots, k_0 \rangle$.

Question 1. Donner la forme NAF des entiers 3, 9, 23.

Question 2. Soit k un entier positif. Montrer que :

a. k a une unique forme NAF, notée $NAF(k)$.
b. la longueur de $NAF(k)$ est égale à la longueur de la représentation de k en binaire plus-ou-moins 1.

On admettra :

c. Soit k tiré selon une distribution de probabilité uniforme dans $[0, 2^n]$. Notons $NAF(k) = \langle k_{\ell-1}, \ldots, k_0 \rangle$. Nous avons $\Sigma_{i=0}^{\ell-1} |k_i|$ est en moyenne (sur tous les tirages possibles) égale à $\ell/3$.

Question 3. Soit k un entier positif.
Donner un algorithme permettant de calculer sa forme NAF et donner sa complexité.

Question 4. Soit $k \in \mathbb{Z}/2^m\mathbb{Z}$ et $P \in G$. Donner un algorithme exploitant la forme NAF et permettant de calculer $k \cdot P$. On évaluera également sa complexité en moyenne.

Nous souhaitons maintenant généraliser la forme NAF en considérant des "chiffres" k_i ne vivant plus nécessairement dans $\{-1, 0, 1\}$, mais dans un ensemble plus grand.
Définition. Une *w-forme non-adjacente* (w-NAF) de longueur ℓ d'un entier positif k, notée $NAF_w(k)$, est une expression $k = \Sigma_{i=0}^{\ell-1} k_i 2^i$ où :

- (1) $|k_i| < 2^{w-1}$ et, si non-nul, k_i est impair,
- (2) $k_{\ell-1} \neq 0$,
- (3) pour toute suite de w coefficients consécutifs, i.e., $k_i, \ldots, k_{i+w-1}$, au plus l'un est non-nul.

On notera que $NAF(k) = NAF_2(k)$.

Nous supposerons que les propriétés énoncées en Question 2. sont toujours vraies pour $NAF_w(k)$. De plus, la propriété c. peut être reformulée en : le nombre moyen de "chiffres" non-nuls est égal à $l/(w+1)$.

Question 5. Soit $k \in \mathbb{Z}/2^m\mathbb{Z}$ et $P \in G$. Donner un algorithme exploitant la forme w-NAF et permettant de calculer $k \cdot P$. On évaluera également sa complexité.

Question 6. Soit $t = (t_{\ell-1}, \ldots, t_0)$. Nous notons $w(t)$ le poids de Hamming de t, i.e., le nombre d'éléments non-nuls de t.

Soit k un entier positif.
Soit $(k'_{\ell-1}, \ldots, k'_0)$ un encodage arbitraire de k tel que $k = \Sigma_{i=0}^{\ell-1} k'_i 2^i$, et $k'_i < 2^{w-1}$, et, si non-nul, est impair.
Montrer que $w((k'_{\ell-1}, \ldots, k'_0)) \geq w(NAF_w(k))$.

Question 7. Jusqu'à maintenant, nous nous sommes intéressés à optimiser le calcul de $k \cdot P$. Dans de nombreuses applications, ce sont des calculs de $i \cdot P + j \cdot Q$ pour différentes valeurs de i et j (à P et Q fixés) qui sont nécessaires. En nous autorisant à stocker au plus 2^{2w} points lors d'un pré-calcul, donner un algorithme qui permet de calculer $i \cdot P + j \cdot Q$ en environ $\lceil m/w \rceil \mathsf{A} + \lceil m/w \rceil w\mathsf{D}$ (hors pré-calcul).

a. donner l'algorithme et justifier sa complexité
b. donner un algorithme permettant d'effectuer de manière efficace le pré-calcul

Note : nous pourrons considérer que i et j sont tous deux représentés par m bits. Le plus petit des deux pourra donc avoir les premiers bits de poids fort à 0.

Annexe : sujets proposés pour la filière MPI

Certains sujets sont accompagnés d'*ébauches* de solution.

Systèmes bien structurés

Un ordre partiel est un ensemble S équipé d'une relation $\leq$ qui est réflexive, transitive et anti-symmétrique. On dit qu'un ordre partiel est beau si pour toute suite infinie

$$s_0, \ldots, s_n, \ldots,$$

de S, il existe une paire croissante : $i < j$ et $s_i \leq s_j$. Dans un ordre partiel $(S, \leq)$, un élément s est dit minimal dans $S' \subseteq S$ si pour tout $s' \in S' \setminus \{s\}$, $s' \not\leq s$. On note $\min(S')$, l'ensemble des éléments minimaux de S'. Un sous-ensemble S' de S est dit clos par le haut si pour tout $s_1 \in S'$ et pour tout $s_2 \in S$ tel que $s_1 \leq s_2$, $s_2 \in S'$. La clôture par le haut d'un sous-ensemble S' de S, dénoté par $\uparrow S'$ est le plus petit sous-ensemble clos par le haut contenant S'. Il correspondant à :

$$\uparrow S' = \left\{ s_2 \mid \exists\, s_1 \in S'. s_1 \leq s_2 \right\}.$$

Dans la suite, on suppose que $(S, \leq)$ est un bel ordre partiel.

Question 1. Montrez les propriétés suivantes sur les beaux ordres partiels :

1. Un sous-ensemble $S' \subseteq S$ a un nombre fini d'éléments minimaux.
2. Un ensemble S' clos par le haut est égal à la clôture par le haut de l'ensemble de ses éléments minimaux, c'est-à-dire
$$S' = \uparrow \min(S').$$
3. Toute suite $S_0 \subseteq S_1 \subseteq \ldots \subseteq S_n \subseteq \ldots$ de sous-ensembles de S clos par le haut est stationaire, c'est-à-dire qu'il exists N tel que pour tout $n \geq N$, $S_N = S_n$.

Solution :

1. Supposons que nous avons un sous-ensemble S' avec un nombre infini d'éléments minimaux. On a donc au moins une suite infinie
$$s_0, \ldots, s_n, \ldots$$
d'éléments minimaux de S' deux à deux distincts. Comme $\leq$ est beau, il existe une paire croissante $i < j$ et $s_i \leq s_j$. Comme s_j est minimal, on obtient une contradiction.
2. Soient $\min(S') = \{s_1, \ldots, s_n\}$ Par définition, s_1, ..., s_n appartiennent à S'. Comme S' est clos par le haut et $\uparrow \{s_1, \ldots, s_n\}$ est le plus petit clos par le haut contenant s_1, ..., s_n, alors $\uparrow \{s_1, \ldots, s_n\} \subseteq S'$. Dans l'autre sens, il suffit de montrer que pour tout $s \in S'$, il existe un élément minimal s' de S' tel que $s' \leq s$. Supposons que cela ne soit pas le cas et que l'on a un élément $s \in S$ qui ne soit pas au-dessus d'un élément minimal. En particulier, s n'est pas minimal. On contruit alors une suite strictement décroissante d'éléments non-minimaux dans S' par récurrence. On commence par définir $s_0 = s$. Si on suppose $s_0, ..., s_n$ construite, comme s_n n'est pas minimal, il existe $s_{n+1} \in S' \setminus \{s_n\}$ tel que $s_{n+1} \leq s_n$. Si s_{n+1} était minimal, alors s serait au-dessus d'un élément minimal, ce qui est impossible par hypothèse. Maintenant, comme $\leq$ est beau, il existe une paire croissante ce qui contredit la décroissance stricte.
3. Supposons que nous ayons une suite non-stationnaire. Nous ponvons en extraire une suite strictement croissante d'ensembles clos par le haut $S_0 \subset S_1 \subset \ldots \subset S_n \subset \ldots$.

   Pour tout n, il existe donc $s_n \in S_{n+1} \setminus S_n$. Comme $\leq$ est beau, il existe toutefois une paire croissante, $i < j$ et $s_i \leq s_j$. Comme $s_i \in S_{i+1}$ et S_{i+1} est clos par le haut, $s_j \in S_{i+1}$. Comme $s_j \notin S_j$ et $S_{i+1} \subseteq S_j$ (car $i < j$), $s_j \notin S_{i+1}$. Contradiction.

Un système de transition est un couple $(S, \to)$ où S est un ensemble (d'états) et $\to \subseteq S \times S$ est une relation (de transition). Nous noterons $s_1 \to s_2$ pour $(s_1, s_2) \in \to$. Nous noterons $\to^*$ pour la clôture reflexive et transitive de $\to$, c'est-à-dire, $s \to^* s'$ si et seulement s'il existe une suite finie $s = s_0 \to s_1 \to \ldots \to s_n = s'$ pour un $n \geq 0$. Un système de transition bien structuré est un triplet $(S, \to, \leq)$ où $(S, \to)$ est un sytème de transition et $(S, \leq)$ est un bel ordre partiel, satisfaisant la condition de compatibilité : si $s_1 \to s_2$ et $s_1 \leq s_3$, alors il existe s_4 tel que $s_2 \leq s_4$ et $s_3 \to s_4$.

Question 2. Nous appelons système d'addition de vecteurs avec états un quadruplet (d, V, Q, E) avec $d \in \mathbb{N}^*$, $V \subseteq \mathbb{Z}^d$ et (Q, E) un graphe dirigé fini avec nœuds Q et arêtes E. Étant donné un tel quadruplet (d, V, Q, E), montrez que la donnée de $(S, \to, \leq)$ avec :

— $S = Q \times \mathbb{N}^d$,
— $(q, u) \to (p, v)$ si $(q, p) \in E$ et $v - u \in V$,
— $(q, u) \leq (p, v)$ si $q = p$ et $u \leq v$ pour l'ordre produit, c'est-à-dire si $u = (u_1, \ldots, u_d)$ et $v = (v_1, \ldots, v_d)$, $u \leq v$ si pour tout i, $u_i \leq v_i$,

est un système de transition bien structuré.

Solution : On vérifie toutes les propriétés de la définition :

— $\leq$ est un ordre partiel : parce que l'égalité sur les états et l'ordre produit sur les vecteurs sont des ordres partiels.
— $\leq$ est un bel ordre partiel : suppose une suite infinie

$$(q_0, u_0), \ldots, (q_n, u_n), \ldots$$

Par le principe des tiroirs, comme Q est fini, il existe q et une sous-suite infinie de la forme

$$(q, v_0), \ldots, (q, v_n), \ldots$$

Donc il suffit de montrer que l'ordre produit sur les vecteurs est un beau quasi-ordre, par induction sur $d > 0$.

— $d = 1$: si on supppose une suite infinie d'entiers naturels $i_0, \ldots, i_n, \ldots$. On peut en réalité montrer qu'il y a une sous-suite infinie croissante. Soit la suite est bornée et par le principe des tiroirs il y a une sous-suite constante. Soit la suite est non-bornée et on peut construire une sous-suite inifinie strictement croissante par induction. En effet, définissons une fonction $\varphi : \mathbb{N} \to \mathbb{N}$ strictement croissante telle que pour tout $j < k$, $i_{\varphi(j)} < i_{\varphi(k)}$, par récurrence forte : $\varphi(0) = 0$; supposons que φ est définie sur $\{0, \ldots, j\}$ comme la suite des i_n est non-bornée, il existe un élément de la liste strictement plus grand que $i_{\varphi(j)}$. On définit $\varphi(j+1)$ comme l'indice de cet élément.
— $d + 1$: On suppose une suite infinie de $\mathbb{N}^{d+1}$, vue comme une suite de $\mathbb{N}^d \times \mathbb{N}$:

$$(u_0, i_0), \ldots, (u_n, i_n), \ldots$$

Par le cas précédent, on a une sous-suite infinie

$$(u_{\varphi(0)}, i_{\varphi(0)}), \ldots, (u_{\varphi(n)}, i_{\varphi(n)}), \ldots$$

telle que

$$i_{\varphi(0)}, \ldots, i_{\varphi(n)}, \ldots$$

la deuxième composante est strictement croissante. En appliquant l'hypothèse de récurrence sur

$$u_{\varphi(0)}, \ldots, u_{\varphi(n)}, \ldots$$

on obtient une paire croissante $j < k$ et $u_{\varphi(j)} \leq u_{\varphi(k)}$ et donc $(u_{\varphi(j)}, i_{\varphi(j)}) \leq (u_{\varphi(k)}, i_{\varphi(k)})$.

— si $(q,u) \to (p,v)$ et $(q,u) \leq (r,w)$, alors $q = r$ et $v - u \in V$. Montrons que $(p, w + (v-u))$ convient :
  — $(r,w) \to (p, w+(v-u))$: comme $q = r$ et $(q,p) \in E$, $(r,p) \in E$. De plus, $w + (v-u) - w = v - u \in V$.
  — $(p,v) \leq (p, w+(v-u))$: On a
$$v = u + (v-u) \leq w + (v-u)$$
parce que $u \leq w$. Au passage, cela prouve que $w + (v-u) \in \mathbb{N}^d$ et donc est un état valide.

Étant donné un sous-ensemble S' de l'ensemble des états S d'un système de transition bien structuré, on définit
$$\mathrm{Pred}\left(S'\right) = \left\{ s \in S \mid \exists\, t \in S'.\, s \to t \right\}$$

Question 3. Montrez que pour tout $S' \subseteq S$, $\uparrow \mathrm{Pred}\,(S') \subseteq \mathrm{Pred}\,(\uparrow S')$. Donnez un contre-exemple à l'inclusion inverse.

Solution : Soit $s' \in \uparrow \mathrm{Pred}\,(S')$. Cela signifie qu'il existe $s \in \mathrm{Pred}\,(S')$ avec $s \leq s'$, et donc qu'il existe $t \in S'$ avec $s \to t$. Comme le système est bien structuré, il existe t' tel que $t \leq t'$ et $s' \to t'$. C'est-à-dire $t' \in \uparrow S'$ et $s' \in \mathrm{Pred}\,(\uparrow S')$.

Pour le contre-exemple, construisons un système d'addition de vecteurs avec un état (donc on va l'ignorer). Choisissons $d = 1$, $V = \{-1\}$ et $S' = \{0\}$. Dans ce cas, $\uparrow S' = \mathrm{Pred}\,(\uparrow S') = \mathbb{N}$ et $\mathrm{Pred}\,(S') = \uparrow \mathrm{Pred}\,(S') = \varnothing$.

Question 4. En utilisant justicieusement les questions 1 et 3, donner un algorithme pour résoudre le problème de décision suivant (problème de couvrement).

Entrées : $(S, \to, \leq)$ un système bien structuré, s, t deux états

Question : existe-t-il t' tel que $s \to^* t'$ et $t' \geq t$?

Nous ne supposerons pas que S est fini, et vous devrez donc faire particulièrement attention à la représentation des sous-ensembles à manipuler. Vous pourrez néanmoins supposer par exemple que la relation $\leq$ est calculable, c'est-à-dire qu'elle est donnée par un algorithme de la forme :

Entrées : s, t deux états

Question : $s \leq t$?

Suivant cette idée, quelles hypothèses devez-vous ajouter à $(S, \to, \leq)$ pour faire fonctionner l'algorithme ?

Solution : Par les questions 1.1 et 1.2, on observe que l'on peut représenter de manière canonique un ensemble clos par le haut par un sous-ensemble fini de S. En effet, la question 1.2. montre que l'ensemble de ses éléments minimaux (qui est fini par la question 1.) caractérise de manière unique un ensemble clos par le haut.

L'idée de l'algorithme est de construire la suite $S_k = \left\{ s' \mid \exists t' \geq t.\, s' \to^k t' \right\}$. Pour cela, on montre que S_k est clos par le haut par récurrence sur k.

— $k = 0$: S_0 est précisément la clôture par le haut de t.
— Supposons S_k est clos par le haut. Par définition, $S_{k+1} = \mathrm{Pred}\,(S_k)$ qui est clos par le haut par la question 5.

Maintenant, les ensembles $T_k = \bigcup_{i=0}^{k} S_i$ forment une suite croissante de clos par le haut, et par la question 1.3., cette suite est stationnaire. De plus, le problème de couvrement se traduit comme $s \in \bigcup_{k=0}^{+\infty} S_k = \bigcup_{k=0}^{N} S_k$ pour un certain N.

Cela amène à l'Algorithme 1.

Maintenant, pour rendre cet algorithme faisable, il est nécessaire de représenter de manière finie les ensembles S_k et T_k à l'aide des éléments minimaux. Initialement, $\min(T_0) = \min(S_0) = \{t\}$.

Algorithme 1 Squelette de l'algorithme

Argument : $(S, \to, \leq)$ un système bien structuré, s, t deux états
Ensure: existe-t-il t' tel que $s \to^* t'$ et $t' \geq t$?
1: $T_0 \leftarrow \uparrow\{t\}$
2: $S_1 \leftarrow \mathrm{Pred}\,(\uparrow\{t\})$
3: **tant que** $S_{k+1} \not\subseteq T_k$ **faire**
4: $\quad T_{k+1} \leftarrow T_k \cup S_{k+1}$
5: $\quad S_{k+2} \leftarrow \mathrm{Pred}\,(S_{k+1})$
6: **fin tant que**
7: **renvoyer** $s \in T_\infty$? ▷ T_∞ est le dernier T_k après la boucle while.

Supposons que nous puissions calculer $\min(T_k)$ et $\min(S_k)$.

Supposons tout d'abord que nous savons calculer $\min(S_{k+1})$. Comme $T_{k+1} = T_k \cup S_{k+1}$, $\min(T_{k+1}) = \min(\min(T_k) \cup \min(S_{k+1}))$. Comme nous savons calculer $\min(T_k)$ et $\min(S_{k+1})$ et que ceux-ci sont des ensembles finis, il suffit de savoir calculer les éléments minimaux d'ensembles finis. Ceci demande qu'un nombre fini de comparaisons, et cela est possible dès que $\leq$ est décidable, c'est-à-dire que nous avons algorithme qui décide le problème :

Entrées : deux états s et t
Question : $s \leq t$?

C'est la première hypothèse que nous ajoutons.

Maintenant, comme $S_{k+1} = \mathrm{Pred}\,(S_k) = \mathrm{Pred}\,(\uparrow\min(S_k)) = \mathrm{Pred}\left(\bigcup_{s' \in \min(S_k)} \uparrow\{s'\}\right) = \bigcup_{s' \in \min(S_k)} \mathrm{Pred}\,(\uparrow\{s'\})$, et que $\min(S_k)$ est fini, il suffit de pouvoir calculer $\min(\mathrm{Pred}\,(\uparrow\{s'\}))$ pour chaque s', c'est-à-dire d'avoir un algorithme PredClot(s') résolvant le problème suivant :

Entrées : un état s'
Sortie : $\min(\mathrm{Pred}\,(\uparrow\{s'\}))$.

Ceci est la deuxième hypothèse à ajouter.

Enfin, il faut vérifier que les tests des lignes 3 et 7 soient décidables avec ces représentation. Tout d'abord, le test de la ligne 3, demande de tester l'inclusion d'un clos par le haut dans un autre. Cela peut se faire avec un nombre fini de comparaisons : il suffit de vérifier si tous les éléments minimaux de S_{k+1} sont plus grands qu'au moins un élément minimal de T_k. Ceci est décidable par la première hypothèse. Le test de la ligne 7 demande à décider si $s \in T_N$. Pour cela, il suffit de vérifier qu'il existe $s' \in \min(T_N)$ tel que $s' \leq s$, ce qui demande qu'un nombre fini de comparaisons ce qui est décidable par la première hypothèse. Ceci donne donc l'Algorithme 2.

Question 5. Vérifiez que les systèmes d'addition de vecteurs avec états tels que V est fini satisfont les conditions supplémentaires de la question précédente.

Solution : La première hypothèse est évidente. Pour la deuxième hypothèse, fixons $(q, u) \in Q \times \mathbb{N}^d$. Démontrons que

$$\mathrm{Pred}\,(\uparrow\{(q,u)\}) = \left\{(p,v) \in Q \times \mathbb{N}^d \mid (p,q) \in E \wedge \exists w \in V.\, u - w \leq v\right\}.$$

En effet, $(p,v) \in \mathrm{Pred}\,(\uparrow\{(q,u)\})$ si et seulement si $(p,q) \in E$ et il existe u' tel que $u' - v \in V$ et $u \leq u'$. En prenant $w = u' - v$, ceci est vrai si et seulement si il existe $w \in V$ et que $u \leq w + v$, c'est-à-dire $u - w \leq v$.

Pour calculer les éléments minimaux de cet ensemble, il faut toutefois être attentif : étant donné $w \in V$, $u - w$ n'appartient pas forcément à $\mathbb{N}^d$. Définissons $u \ominus w \in \mathbb{N}^d$ tel que si $u = (u_1, \ldots, u_d)$ et $w = (w_1, \ldots, w_d)$, $u \ominus w = (\max(u_1 - w_1, 0), \ldots, \max(u_d - w_d, 0))$. Par l'égalité précédente, on a alors :

$$\mathrm{Pred}\,(\uparrow\{(q,u)\}) = \uparrow\{(p, u \ominus w) \mid (p,q) \in E \wedge w \in V\}$$

et donc

$$\min(\mathrm{Pred}\,(\uparrow\{(q,u)\})) = \min(\{(p, u \ominus w) \mid (p,q) \in E \wedge w \in V\}).$$

Algorithme 2 Algorithme plus précis

Argument : $(S, \to, \leq)$ un système bien structuré, s, t deux états
Ensure: existe-t-il t' tel que $s \to^* t'$ et $t' \geq t$?

```
fonction NonInclusion(A, B)                    ▷ Prend A et B ensembles finis et retourne ↑A ⊈ ↑B ?
    pour x ∈ A faire
        pour y ∈ B faire
            si y ⋠ x alors
                renvoyer Vrai
            fin si
        fin pour
    fin pour
    renvoyer Faux
fin fonction
fonction Appartient(A)                         ▷ Prend A ensemble fini et retourne s ∈ ↑A ?
    pour x ∈ A faire
        si x ≤ s alors
            renvoyer Vrai
        fin si
    fin pour
    renvoyer Faux
fin fonction
fonction Union(L)                              ▷ Prend L liste d'ensembles finis et retourne min ⋃_{A∈L} A
    si L = [] alors
        renvoyer {}
    fin si
    A ← hd(L)
    M ← Union(tl(L))
    pour x ∈ A faire
        nonDansClotM ← Vrai
        pour y ∈ M faire
            si y ≤ x alors
                nonDansClotM ← Faux
                Break
            fin si
            si x ≤ y alors
                Retirer y de M
            fin si
        fin pour
        si nonDansClotM alors
            Ajouter x à M
        fin si
    fin pour
    renvoyer M
fin fonction
fonction Converture
    T ← {t}
    S ← PredClot(t)
    tant que NonInclusion(S, T) faire
        T ← Union([T, S])
        S ← Union([PredClot(s') pour s' ∈ S])
    fin tant que
    renvoyer Appartient(T) ?
fin fonction
```

Comme l'ensemble $\{(p, u \ominus w) \mid (p, q) \in E \wedge w \in V\}$ est fini, on peut calculer son ensemble des éléments minimaux par un nombre fini de comparaisons.

Fixons un état $s \in S$. L'arbre fini d'accessibilité depuis s est l'arbre T défini par les propriétés suivantes :

- Chaque nœud de cet arbre est étiqueté par un état $s' \in S$.
- La racine est étiquetée par s.
- Si un nœud étiqueté par s' est subsumé, c'est-à-dire, s'il existe un ancêtre s'' dans cet arbre tel que $s'' \leq s'$, alors ce nœud est une feuille.
- Sinon, un nœud étiqueté par s' a un enfant étiqueté par s'' pour chaque $s' \to s''$.

Jusqu'à la fin de ce sujet, nous supposerons que l'ensemble

$$\mathrm{Post}(s) = \{s' \mid s \to s'\}$$

est fini pour tout $s \in S$.
Finalement, nous pourrons utiliser sans justification le théorème de König : un arbre a un nombre fini de nœuds si et seulement si chacun de ses nœuds a un nombre fini d'enfants et si toutes ses branches sont de longueur finie.

Question 6. Montrez que cet arbre est fini.

Solution : On utilise le théorème de König. Le branchement fini est par hypothèse. Montrons que toute branche est finie. En effet, si cet arbre a une branche inifinie, alors nous avons une suite infinie :

$$s = s_0 \to s_1 \to \ldots \to s_n \to \ldots$$

avec pour tout $i < j$, $s_i \not\leq s_j$ (sinon s_j est subsumé et n'aurait pas d'enfant). Cela contredit le fait que $(S, \leq)$ est un beau ordre partiel.

Nous supposerons que le système bien structuré est strictement compatible, c'est-à-dire, que si $s_1 \to s_2$ et $s_1 < s_3$ alors il existe s_4 tel que $s_2 < s_4$ et $s_3 \to s_4$.

Question 7. Dans le même esprit que la question 4, donnez un algorithme pour résoudre le problème de décision suivant (problème de bornitude) :

Entrées : $(S, \to, \leq)$ un système bien structuré avec compatibilité stricte, s un état
Question : L'ensemble $\mathrm{Post}^*(s) = \{s' \mid s \to^* s'\}$ est-t-il fini ?

Solution : Il faut d'abord décrire un algorithme pour calculer l'arbre fini d'accessibilité, voir 3. Cet algorithme termine pour les raisons suivantes :

- La boucle "pour" (lignes 14-18) termine par branchement fini.
- La boucle "tant que" interne (lignes 8-10) termine parce que tout noœud n'a qu'un nombre fini d'ancêtres.
- La boucle "tant que" externe (lignes 3-19) termine parce que chaque itération soit ajoute un nœud dans T, soit n'ajoute pas de nœud dans T et enlève un élément de L et comme l'arbre fini d'accessibilité est fini par la question précédente.

Algorithme 3 Algorithme pour l'arbre d'accessibilité

Argument : $(S, \to, \leq)$ un système bien structuré, s un état
Ensure: T arbre fini d'accessibilité depuis s

T = arbre avec un noeud étiqueté par s et sans enfant
$L = [T]$ liste contenant que la racine
tant que L n'est pas vide **faire**
    $T' = \mathrm{hd}(L)$
    s' = étiquette de T'
    $L = \mathrm{tl}(L)$
    T'' = parent de T'
    tant que T'' n'est pas la racine et T'' ne subsume pas T' **faire**
        T'' = parent de T''
    fin tant que
    si T'' n'est pas la racine **alors**
        Continue ▷ T' est subsumé et donc n'a pas d'enfant
    fin si
    pour s'' tel que $s' \to s''$ **faire**
        T'' = arbre avec un nœud étiqueté par s'' et sans enfant
        Ajoute T'' à L
        Ajoute T'' aux enfants de T'
    fin pour
fin tant que
renvoyer T

De plus, pour écrire cet algorithme, il est nécessaire que 1) il existe un algorithme pour décider $\leq$, 2) un algorithme qui, étant donné un état s', liste les éléments de $\mathrm{Post}(s')$.

Maintenant, la bornitude peut se voir sur l'arbre d'accessibilité de la manière suivante. L'ensemble $\mathrm{Post}^*(s) = \{s' \mid s \to^* s'\}$ est infini si et seulement si il existe une feuille étiquetée par s' qui soit subsumée par un nœud étiqueté par s'' avec $s'' < s'$.

En effet, supposons qu'il existe une telle feuille s' et un nœud s'' qui le subsume strictement. On définit une suite infinie d'états de la façon suivante. Soit $s'' = s_0 \to s_1 \to \ldots \to s_n = s'$ le chemin de s'' à s' dans l'arbre d'accessibilité. Comme $s'' < s'$, $n > 0$. Nous étendons cela en une suite $(s_i)_{i \in \mathbb{N}}$ telle que :

— pour tout i, $s_i \to s_{i+1}$,
— pour tout i, $s_i < s_{i+n}$

par récurrence forte. Tout d'abord, nous observons que ces deux propriétés sont vraies pour $0 \leq i < n$. Supposons maitenant que s_i est défini pour $0 \leq i \leq p$ pour $p \geq n$. Par hypothèse de récurrence, $s_{p-n} \to s_{p-n+1}$ et $s_{p-n} < s_p$. Par compatibilité stricte, il existe s_{p+1} tel que $s_{p-n+1} < s_{p+1}$ et $s_p \to s_{p+1}$ comme voulu. Par conséquent, l'ensemble $\{s_{in} \mid i \in \mathbb{N}\}$ est un sous-ensemble infini de $\mathrm{Post}^*(s)$.

Dans l'autre sens, supposons que $\mathrm{Post}^*(s)$ est infini. Définissons l'arbre suivant :

— ses nœuds sont les séquences finies $s = s_0 \to s_1 \to \ldots s_n$ telles que $s_i \neq s_j$ pour $i \neq j$,
— la racine est s,
— le parent de $s = s_0 \to s_1 \to \ldots s_n$ pour $n > 0$ est $s_0 \to s_1 \ldots \to s_{n-1}$.

Comme $\mathrm{Post}^*(s)$ est infini, cet arbre a un nombre infini de nœuds. Par le théorème de König, comme cet arbre est à branchement fini, il a une branche infinie. Cette branche infinie correspond à une suite infinie :

$$s = s_0 \to s_1 \to \ldots s_n \to \ldots$$

où tous les s_n sont deux-à-deux différents. Par définition de l'arbre d'accessibilité, il y a un préfixe maximal de cette suite

$$s = s_0 \to s_1 \to \ldots s_i$$

qui est un chemin dans l'arbre d'accessibilité. Comme ce chemin est maximal, cela signifie que s_i est une feuille de l'arbre d'accessibilité. Comme $s_i \to s_{i+1}$, par définition de l'arbre d'accessibilité, cela signifie qu'il existe $j < i$ tel que $s_j \leq s_i$. Comme $s_j \neq s_i$, $s_j < s_i$ comme annoncé.

Finalement, cette caractérisation de la bornitude en terme de feuilles de l'arbre d'accessibilité se teste par un nombre fini de comparaisons.

Réseaux booléens et leurs points fixes

Un *réseau booléen* à n composantes est une fonction $f : \{0,1\}^n \to \{0,1\}^n$. Une *configuration* de f est un n-uplet $x \in \{0,1\}^n$. La i-ème composante de x est notée x_i. Pour $1 \leq i \leq n$, la i-ème composante de $f(x)$ est notée $f_i(x)$, où f_i est une fonction de $\{0,1\}^n$ dans $\{0,1\}$. On note également e_j le n-uplet ne comportant qu'un seul "1" en position j.
L'ensemble des configurations est muni de la relation d'ordre partielle : $x \leq y \iff \forall i, x_i \leq y_i$. Si x et y sont deux configurations, on note $x + y$ l'addition modulo 2 composante à composante.

Un *graphe signé* est un couple $G = (V, E)$ où $E \subseteq V \times V \times \{-1, 1\}$. V est l'ensemble des sommets, E l'ensemble des arcs, qui peuvent être positifs (signe 1) ou négatifs (signe -1). Il est possible d'avoir à la fois un arc positif et un arc négatif d'un sommet vers un autre. On note $|G|$ le graphe orienté obtenu en ignorant les signes. On étend la notion de signe à tout chemin de G en faisant le produit des signes des arcs.

Le *graphe d'interaction* d'un réseau booléen f est le graphe signé $G(f)$ défini comme suit. L'ensemble des sommets est $V = \{1, \ldots, n\}$. Pour tout $i, j \in V$, il existe un arc positif (resp. négatif) $j \to i$ si et seulement si il existe une configuration x telle que $x_j = 0$ et $f_i(x + e_j) - f_i(x)$ est strictement positif (resp. négatif). En d'autres termes, un arc $j \to i$ signifie que f_i dépend de la j-ème variable. Si l'arc est positif, on dit que j est un *activateur* de i. Si l'arc est négatif, on dit que j est un *inhibiteur* de i.

Question 1. Soit le réseau booléen à 3 composantes suivant :

$$f_1(x) := x_3 \wedge (\neg x_1 \vee \neg x_2)$$
$$f_2(x) := x_3 \wedge x_1$$
$$f_3(x) := x_2$$

Calculer la sortie de f sur chaque entrée possible et donner le graphe $G(f)$.

On note : $x \leq_i^G y$ si et seulement si $x_j \leq y_j$ pour tous les activateurs j de i, et $x_j \geq y_j$ pour tous les inhibiteurs j de i.
Pour deux configurations x, y, on note $\Delta(x, y) \subseteq \{1, \ldots, n\}$ l'ensemble des positions où elles diffèrent, et la *distance de Hamming* entre x et y est $|\Delta(x, y)|$.

Question 2. Soit f un réseau Booléen et G son graphe d'interactions. Soit $x \neq y$ tels que $x \leq_i^G y$.

1. Supposons qu'il existe j tel que $x_j < y_j$. Montrer que $x + e_j \leq_i^G y$ et $f_i(x + e_j) \geq f_i(x)$.
2. Supposons qu'il existe j tel que $y_j < x_j$. Montrer que $x \leq_i^G y + e_j$ et $f_i(y + e_j) \leq f_i(y)$.

Dans les questions suivantes on considère un graphe signé à n sommets $\{1, \ldots, n\}$ noté G, et on note $F(G)$ l'ensemble des réseaux booléens qui ont G pour graphe d'interactions.

Question 3. Montrer que si $f \in F(G)$, alors pour tout $1 \leq i \leq n$ et toutes configurations x, y :

$$x \leq_i^G y \implies f_i(x) \leq f_i(y)$$

On procédera par induction sur la distance de Hamming entre x et y.

Pour un réseau booléen f, un *point fixe* de f est une configuration x telle que $f(x) = x$.
De plus, pour un graphe G on note $G[U]$ le sous-graphe induit par le sous-ensemble de sommets U, c'est-à-dire en retirant les sommets hors de U et les arcs attachés.

Question 4. Soit $f \in F(G)$ avec deux points fixes distincts x et y. Montrer qu'il existe deux sommets i, j (pas nécessairement distincts) tels que G a un arc $j \to i$ de signe $(y_j - x_j)(y_i - x_i)$.

Question 5. Montrer que si $f \in F(G)$ a deux points fixes distincts x et y, alors $G[\Delta(x, y)]$ possède un cycle positif.

On admet le théorème suivant.

> Si G est fortement connexe et sans cycle négatif, alors il existe une partition de ses sommets en deux ensembles A et B telle qu'un arc (u, v) de G est positif si et seulement si u et v sont dans la même partie.

Un réseau est dit *monotone* si pour toutes configurations x, y :

$$x \leq y \implies f(x) \leq f(y)$$

Question 6. Montrer que si G est fortement connexe et sans cycle négatif, alors il existe une configuration z telle que, pour tout $f \in F(G)$, le réseau $h(x) := f(x + z) + z$ est monotone.

Montrer que le graphe d'interactions de h s'obtient à partir de celui de G en rendant positifs tous les arcs.

Indice : Commencer avec une partition des sommets A, B donnée par le théorème. Définir z par $z_i = 1$ si $i \in A$ et $z_i = 0$ sinon.

Question 7. Montrer que si G est fortement connexe et sans cycle négatif, alors tout $f \in F(G)$ a au moins deux points fixes.

Question 8. Montrer que si G n'a pas de cycle négatif, alors tout $f \in F(G)$ a au moins un point fixe.

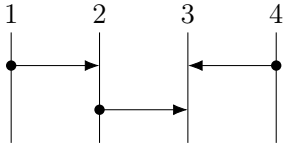
Tri parallélisable

Dans tout l'énoncé, on note $[\![a, b]\!]$ l'intervalle entier $\{a, a+1, \ldots, b\}$.

Soit $n \geq 2$ un entier.
Comparateur. Soit $i, j \in [\![1, n]\!]$ deux entiers distincts. Le *comparateur* de i vers j, noté $i{\bullet\!\to}j$, est défini sur les suites d'entiers $(u_1, \ldots, u_n)$ par $i{\bullet\!\to}j(u_1, \ldots, u_n) = (v_1, \ldots, v_n)$ avec :

$$v_i = \min(u_i, u_j), \quad v_j = \max(u_i, u_j), \quad \text{et } \forall k \notin \{i, j\}, v_k = u_k.$$

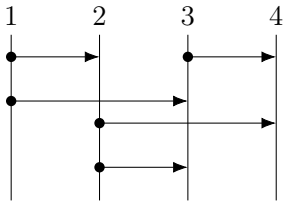
Exemple. $2{\bullet\!\to}3(10, 7, 5, 20) = (10, 5, 7, 20)$.
Multicomparateur. Un *multicomparateur* est un ensemble de comparateurs $\{a_1{\bullet\!\to}a_2, \ldots, a_{2\ell-1}{\bullet\!\to}a_{2\ell}\}$ tel que les a_k sont deux à deux distincts. En tant que fonction, un multicomparateur est interprété comme la composition des comparateurs qu'il contient (l'ordre de composition n'a pas d'importance).
Exemple. $\{1{\bullet\!\to}2, 3{\bullet\!\to}4\}$ est un multicomparateur, mais pas $\{1{\bullet\!\to}2, 2{\bullet\!\to}3\}$.
Réseau. Un *réseau* est une suite de multicomparateurs. En tant que fonction, un réseau est interprété comme la composition des multicomparateurs qu'il contient, dans l'ordre de la suite (premier multicomparateur en premier). La *profondeur* du réseau est la longueur de la suite qui le définit.
Exemple. $(\{1{\bullet\!\to}2, 4{\bullet\!\to}3\}, \{2{\bullet\!\to}3\})$ est un réseau de profondeur 2, qu'on peut représenter comme suit.



Réseau de tri. Un *réseau de tri* est un réseau R avec la propriété que pour toute suite d'entiers $(u_1, \ldots, u_n)$ en entrée, $R(u_1, \ldots, u_n)$ renvoie la suite triée par ordre croissant.
Exemple. $(\{1{\bullet\!\to}2\}, \{1{\bullet\!\to}3\}, \{2{\bullet\!\to}3\})$ est un réseau de tri de profondeur 3 sur $n = 3$ éléments.

Question 1. Construire un réseau de tri de profondeur 3 pour $n = 4$.

Solution : Une solution est $(\{1{\bullet\!\to}2, 3{\bullet\!\to}4\}, \{1{\bullet\!\to}3, 2{\bullet\!\to}4\}, \{2{\bullet\!\to}3\})$.



Une manière de le justifier : $(\{1{\bullet\!\to}2, 3{\bullet\!\to}4\}, \{2{\bullet\!\to}4\})$ garantit que l'élément maximal arrive en position 4. Symétriquement $(\{1{\bullet\!\to}2, 3{\bullet\!\to}4\}, \{1{\bullet\!\to}3\})$ garantit que l'élément minimal arrive en position 1. Donc, après $(\{1{\bullet\!\to}2, 3{\bullet\!\to}4\}, \{1{\bullet\!\to}3, 2{\bullet\!\to}4\})$, seuls les éléments en position 2 et 3 peuvent être potentiellement en mauvaise place. Le dernier comparateur $2{\bullet\!\to}3$ s'assure que leur position est correcte.

Question 2. Montrer qu'un réseau de tri sur n éléments doit être de profondeur au moins $\log_2(n)$.

Remarque : on préférera si possible un raisonnement sans calculs lourds, et qui donne la borne $\log_2(n)$ exactement, plutôt que de l'obtenir à une constante près.

Solution : Indication : partir de l'élément en position 1 (par exemple), et regarder dans combien de positions différentes l'élément peut se retrouver après les k premiers multicomparateurs du réseau.

On voit facilement que c'est borné par 2^k, parce que dans un multicomparateur, chaque position ne peut être envoyée que sur deux positions possibles. Or si le réseau trie correctement toutes les suites d'entiers, l'élément en position 1 doit nécessairement pouvoir être envoyé à terme sur toutes les n positions possibles. Il s'ensuit que la profondeur p du réseau doit satisfaire $2^p \geq n$, ce qui conclut la question.

Remarque. Cette approche est déconseillée, mais un certain nombre de candidats connaissent probablement le résultat qu'un algorithme de tri reposant sur des comparaisons doit effectuer $\Omega(n \log n)$ comparaisons. En effet un tel algorithme induit une injection des permutations à n éléments vers la suite binaire des résultats des comparaisons. Le nombre c de comparaisons satisfait donc $2^c \geq n!$. Il pourrait être tentant de s'y ramener en remarquant qu'un multicomparateur contient au plus $n/2$ comparateurs, puisque tous les entiers intervenant dans les comparateurs sont deux à deux distincts. Il s'ensuit immédiatement qu'il doit y avoir $\Omega(\log n)$ multicomparateurs dans un réseau de tri. Mais cela ne donne pas immédiatement la forme simple $\log_2(n)$ demandée. Obtenir cette forme revient à montrer $\log_2(n!) > n/2 \cdot \log_2(n)$, i.e. $n! > n^{n/2}$. Pour n pair, cela peut se voir par l'observation $i \cdot (n+1-i) \geq n$ pour $1 \leq i \leq n/2$, qui découpe le produit $n!$ en $n/2$ sous-produits de deux termes, chaque sous-produit étant $\geq n$. Le cas n impair se déduit aisément. (Pour montrer $\log_2(n!) > n/2 \cdot \log_2(n)$, l'utilisation de Stirling ou $n! > \left(\frac{n}{e}\right)^n$ est aussi possible, mais découragée.)

Question 3. Pour $n \geq 2$ quelconque, construire un réseau de tri de profondeur $O(n)$.

Solution : Un point de départ est de remarquer que le tri bulle se traduit naturellement en un réseau :

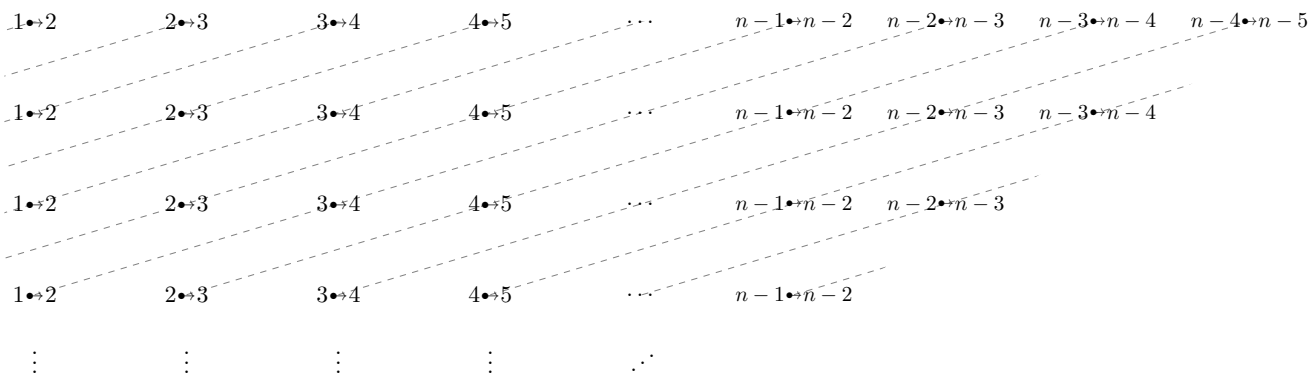
$$\begin{aligned}(&\{1\bullet\!\!\to 2\}, \{2\bullet\!\!\to 3\}, \ldots, \{n-1\bullet\!\!\to n\},\\ &\{1\bullet\!\!\to 2\}, \{2\bullet\!\!\to 3\}, \ldots, \{n-2\bullet\!\!\to n-1\},\\ &\ldots,\\ &\{1\bullet\!\!\to 2\}).\end{aligned}$$

Ce réseau est de profondeur $O(n^2)$.

Pour obtenir un circuit de profondeur $O(n)$, on peut grouper judicieusement les comparateurs en multicomparateurs. Par exemple : on prend chaque comparateur $i\bullet\!\!\to j$, dans l'ordre du réseau, et on le fait « remonter » le plus haut possible tant qu'il ne croise pas un comparateur où i ou j apparaît. On obtient :

$$\begin{aligned}(&\{1\bullet\!\!\to 2\},\\ &\{2\bullet\!\!\to 3\},\\ &\{1\bullet\!\!\to 2, 3\bullet\!\!\to 4\},\\ &\{2\bullet\!\!\to 3, 4\bullet\!\!\to 5\},\\ &\{1\bullet\!\!\to 2, 3\bullet\!\!\to 4, 5\bullet\!\!\to 6\},\\ &\{2\bullet\!\!\to 3, 4\bullet\!\!\to 5, 6\bullet\!\!\to 7\},\\ &\ldots).\end{aligned}$$

Visuellement, si on écrit chaque remontée de bulle sur une ligne, cela revient à grouper les comparateurs suivant les droites de pente 1/2. Sur le dessin suivant, les comparateurs traversés par la même ligne sont groupés en un multicomparateur ; les multicomparateurs du réseau correspondent aux lignes qu'on traverse en partant en haut à gauche et en descendant.



Autre solution : si n est pair (le cas impair est similaire), on peut aussi répéter n fois la suite de deux multicomparateurs

$$(\{1\bullet\!\!\to 2, 3\bullet\!\!\to 4, \ldots, n-1\bullet\!\!\to n\}, \{2\bullet\!\!\to 3, 4\bullet\!\!\to 5, \ldots, n-2\bullet\!\!\to n-1\})$$

Ceci donne essentiellement un « sur-réseau » du réseau de la solution précédente, dans le sens où le k-ième multicomparateur de la première solution est inclus dans le k-ième multicomparateur de la seconde solution.

Rotation. Une suite $(u_1, \ldots, u_n)$ est une *rotation* de $(v_1, \ldots, v_n)$ s'il existe $k \in [\![1, n]\!]$ tel que

$$(u_1, \ldots, u_n) = (v_k, v_{k+1}, \ldots, v_n, v_1, v_2, \ldots, v_{k-1}).$$

Suite vallonnée. Une suite $(u_1, \ldots, u_n)$ est *croissante* (resp. décroissante) si pour tout $i \in [\![1, n-1]\!]$, $u_i \leq u_{i+1}$ (resp. $u_i \geq u_{i+1}$). Une suite $(u_1, \ldots, u_n)$ est *croissante-décroissante* s'il existe $k \in [\![1, n]\!]$ tel que $(u_1, \ldots, u_k)$ est croissante et $(u_k, \ldots, u_n)$ est décroissante. Une suite $(u_1, \ldots, u_n)$ est *vallonnée* si elle est une rotation d'une suite croissante-décroissante.

Question 4. Soit $n = 2m$ avec $m \geq 1$. Soit M le multicomparateur $\{1\bullet\!\!\to m+1, 2\bullet\!\!\to m+2, \ldots, m\bullet\!\!\to 2m\}$. Soit $(u_1, \ldots, u_n)$ une suite vallonnée, et soit $(v_1, \ldots, v_n) = M(u_1, \ldots, u_n)$.

a. Montrer que $(v_1, \ldots, v_m)$ et $(v_{m+1}, \ldots, v_{2m})$ sont vallonnées.

b. Montrer $\forall x \in \{v_1, \ldots, v_m\}, \forall y \in \{v_{m+1}, \ldots, v_{2m}\}, x \leq y$.

Indication : pour ces deux questions, il peut être utile de voir $(u_1, \ldots, u_n)$ non pas comme une suite quelconque indexée par $[\![1, n]\!]$, mais comme une suite périodique de période n indexée par $\mathbb{Z}$. De même, on peut voir $(v_1, \ldots, v_m)$ et $(v_{m+1}, \ldots, v_{2m})$ comme deux suites périodiques de période $m = n/2$. Essayer de construire ces deux dernières à partir de la première.

Solution :

Indication 2 : en plus de la suite périodique correspondant à $(u_1, \ldots, u_n)$, considérer aussi la même suite translatée d'une demi-période ($n/2$ positions).

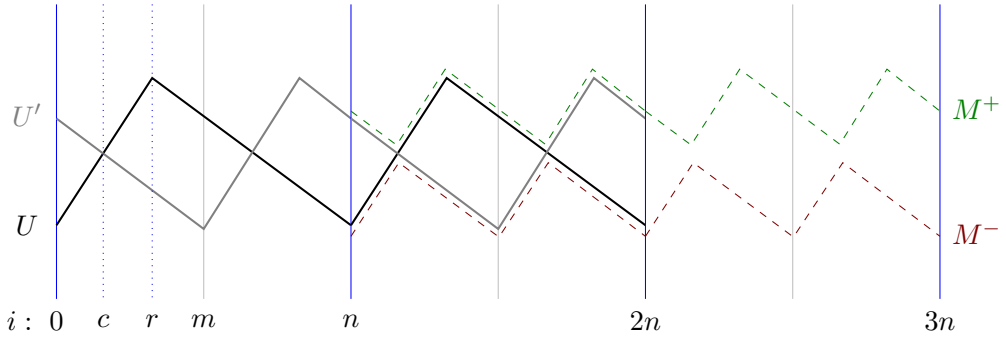
Soit $(U_i)_{i\in\mathbb{Z}}$ la suite périodique de période n définie par $i = j \bmod n \implies U_i = u_j$, pour $i \in \mathbb{Z}$ et $j \in [\![1, n]\!]$. Soit $(U'_i)_{i\in\mathbb{Z}}$ la suite U_i décalée de $n/2$: c'est-à-dire $U'_i = U_{i+n/2}$. On peut définir $(M_i^+)_{i\in\mathbb{Z}}$ et $(M_i^-)_{i\in\mathbb{Z}}$ par $M_i^+ = \max(U_i, U'_i)$ et $M_i^- = \min(U_i, U'_i)$ pour $i \in \mathbb{Z}$.

Une première observation est que M^+ et M^- sont de période $m = n/2$. Une deuxième observation est que M^- est constituée d'une répétition de $(v_1, \ldots, v_m)$; de même M^+ est une répétition de $(v_{m+1}, \ldots, v_{2m})$. Les deux questions de l'énoncé se traduisent donc en deux questions sur M^- et M^+, à savoir :

a. Il existe k tel que $(M_k^-, \ldots, M_{k+m-1}^-)$ est croissante-décroissante ; de même pour M^+.

b. Tous les éléments de M^- sont inférieurs à tous les éléments de M^+.

Les deux propriétés qu'on souhaite montrer sont clairement invariantes par translation de U, donc sans perte de généralité on peut supposer que U_0 est le minimum de U. Soit $r \in [\![1, n-1]\!]$ minimal tel que que U_r est le maximum de U. Sans perte de généralité, $r < m$ (quitte à remplacer U par V avec $V_i = U_{-i}$, qui préserve toutes le propriétés étudiées). Les deux propriétés demandées sont immédiates sur un dessin.



Les propriétés « se voient sur le dessin », mais pour faire plus proprement, on peut raisonner comme suit. Sur l'intervalle $[\![0, r]\!]$, U est croissante, tandis que U' est décroissante, nécessairement à valeurs dans $[\![U_0, U_r]\!]$ (puisque U_0 et U_r sont les minimum et maximum de U). Il s'ensuit que les deux suites se « croisent », au sens suivant : il existe $c \in [\![1, r-1]\!]$ tel que $U_i \leq U'_i$ pour $i \in [\![0, c]\!]$, et $U'_i \leq U_i$ pour $i \in [\![c+1, r]\!]$.

— Sur l'intervalle $[\![0, c]\!]$, $M^-_i = U_i$, en particulier M^- est croissante sur cet intervalle.
— Sur l'intervalle $[\![c+1, r]\!]$, $M^-_i = U'_i$, en particulier M^- est décroissante sur cet intervalle.
— Sur l'intervalle $[\![r, m-1]\!]$, M^- est le minimum de deux suites décroissantes, donc décroissante.

On conclut que M^- est croissante-décroissante sur $[\![0, m-1]\!]$; noter que les deux intervalles où la suite décroît ont un point en commun, de sorte que tout se recolle bien. On procède de même pour M^+. (Alternativement, on peut ramener le cas de M^+ au cas M^- déjà traité en remplaçant U par $-U$.) Le raisonnement précédent montre aussi que M^- est inférieur ou égal à U_c partout ; symétriquement M^+ est supérieur ou égal à U_c partout ; ce qui donne la seconde propriété demandée.

Question 5.

a. Soit $n = 2^k$ une puissance de 2. Construire un réseau R de profondeur $O(\log n)$ tel que pour toute suite vallonnée $(u_1, \ldots, u_n)$, $R(u_1, \ldots, u_n)$ renvoie la suite triée par ordre croissant.
b. Soit $n = 2^k$ une puissance de 2. Construire un réseau de tri sur n éléments de profondeur $O(\log^2 n)$.
c. Même question pour $n \geq 2$ quelconque.

Indication. Le lemme suivant est-il vrai ? Sinon, essayer de trouver une condition sur R qui le rende vrai.

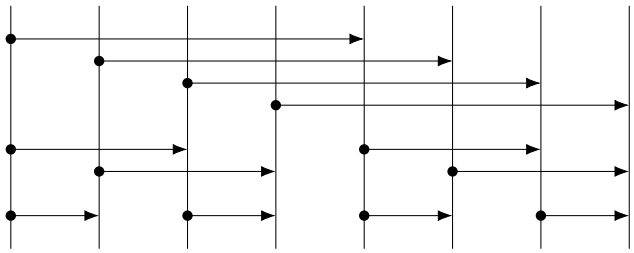
Lemme. Étant donné $m < n$, si on part d'un réseau de tri R sur n éléments et qu'on supprime les comparateurs qui font intervenir des positions $> m$, on obtient un réseau de tri sur m éléments.

Solution :

a. La question précédente fait tout le travail. En reprenant les notations de cette question, on voit que le multicomparateur M envoie une suite vallonnée quelconque $(u_1, \ldots, u_n)$ sur une suite $(v_1, \ldots, v_n)$ telle que tous les éléments de $(v_{m+1}, \ldots, v_{2m})$ sont plus grands que tous les éléments de $(v_1, \ldots, v_m)$, par la propriété (a). Il s'ensuit que pour trier $(v_1, \ldots, v_n)$ en ordre croissant, il suffit de trier séparément la moitié gauche et droite. Or par la propriété (b), ces sous-suites gauches et droites sont elles-mêmes vallonnées, donc on s'est ramené du problème de trier une suite vallonnée de longueur n, au problème de trier deux suites vallonnées de longueur $n/2$. Pour initier la récurrence, on remarque qu'il est trivial de trier une suite de longueur 2, avec un comparateur.

Dans cette construction inductive, lorsqu'on applique un réseau distinct aux moitiés gauche et droite des positions, ils agissent sur des positions deux à deux distinctes. La profondeur de deux réseaux appliqué en parallèle est le maximum de la profondeur des deux réseaux. Il s'ensuit dans notre cas que la profondeur du

réseau est exactement k, puisque le « réseau » qui sert dans l'induction est juste un multicomparateur, et qu'il y a k niveaux d'induction.



b. Supposons qu'on dispose d'une réseau de tri sur $n/2$ éléments. Pour trier n éléments, on applique notre réseau sur les $n/2$ premiers éléments, et le même réseau dans le sens inverse sur les $n/2$ derniers éléments. En sortie, on obtient nécessairement une suite croissante sur les $n/2$ premières positions, puis décroissante sur les suivantes ; en particulier on obtient une suite vallonnée. Il reste à appliquer le réseau qui trie les suites vallonnées, qu'on a construit précédemment, pour obtenir un réseau de tri sur n éléments. (A posteriori, le réseau qu'on a construit dans la question précédente permet de réaliser une fusion au sens du tri fusion, quitte à trier la seconde suite dans le sens décroissant.)

   On obtient ainsi une construction inductive. Chaque niveau d'induction fait intervenir la construction de tri pour les suites vallonnées, qui a une profondeur $O(\log n)$. Il y a $\log_2 n$ niveaux de récursion : on obtient finalement une construction de profondeur $O(\log^2 n)$.

c. Question difficile. Le lemme de l'indication est faux tel quel. Un contre-exemple est le réseau de tri :

$$(\{1\bullet\!\!\to 3, 3\bullet\!\!\to 2, 1\bullet\!\!\to 3, 2\bullet\!\!\to 3\}).$$

   Pour voir que c'est un réseau de tri, on peut remarquer que les trois premiers comparateurs placent l'élément minimal en position 1 (en fait, c'est un tri bulle, mais qui intervertit les deux dernières positions) ; ensuite le comparateur $2\bullet\!\!\to 3$ ordonne les deux éléments restants. Maintenant, si on supprime les comparateurs qui font intervenir la position 3 en espérant obtenir un réseau de tri sur 2 éléments, la déception est inévitable, parce qu'on obtient en fait le réseau vide.

   La condition suivante suffit à rendre le lemme vrai : tous les comparateurs qui interviennent dans le réseau sont orientés « vers la droite », c'est-à-dire qu'ils sont de la forme $i\bullet\!\!\to j$ avec $i < j$. En effet, dans ce cas, le réseau « tronqué » à m positions agit sur une suite en entrée $(u_1, \ldots, u_m)$ exactement comme le réseau de départ agit sur la suite $(u_1, \ldots, u_m, +\infty, \ldots, +\infty)$ de longueur n. En effet c'est vrai pour chaque comparateur qui compose le réseau. On conclut qu'on obtient en sortie la suite triée.

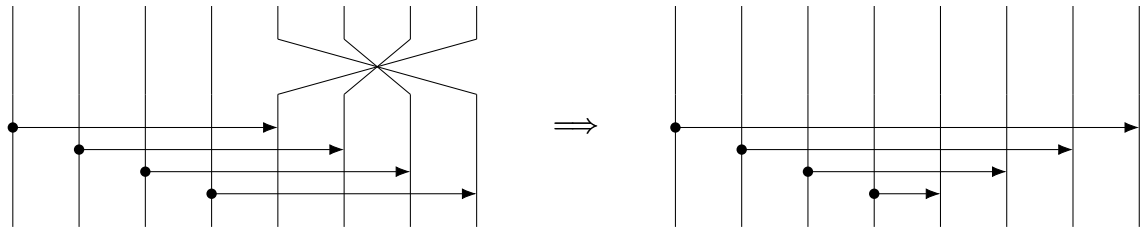
   Avec cette condition, on s'est ramené à la question suivante : trouver une variante de la construction qu'on a faite jusqu'ici, où tous les comparateurs vont « vers la droite ».

   Dans la construction de la question précédente, on a des comparateurs vers la gauche à cause du fait qu'on applique un réseau de tri à l'envers sur la moitié droite de l'état, lors de la construction inductive. Cela venait du fait que le multicomparateur M attendait en entrée une suite croissante-décroissante (ou plus généralement, vallonnée). On serait sauvé si le multicomparateur M avait le même comportement, avec en entrée une suite qu'on peut appeler *demi-croissante*, qui est croissante sur chaque moitié des positions ($[\![1, n/2]\!]$ et $[\![n/2+1, n]\!]$). Avec ça, on pourrait faire la même récurrence qu'avant sans avoir besoin d'inverser le tri dans l'induction, et donc sans jamais générer de comparateurs vers la gauche.

   On s'est ainsi ramené à devoir créer un réseau qui trie correctement les suites demi-croissantes. Or si on applique une permutation « miroir » sur la moitié droite de l'état, une suite demi-croissante devient une suite croissante-décroissante, donc vallonnée. Si on prend le multicomparateur M, et qu'on le compose en entrée avec une permutation miroir sur sa moitié droite, alors on obtient un multicomparateur qui prend en entrée une suite demi-croissante, et donne une sortie qui a exactement les même propriétés que M sur une suite vallonnée. Pour justifier ce point plus en détail : si on applique une permutation de $[\![1, n]\!]$ à toutes les positions intervenant dans un multicomparateur (ou dans un réseau), et qu'on applique la même permutation sur la suite en entrée, on obtient la même sortie que dans le multicomparateur (ou réseau) d'origine avec la

suite d'origine, composé en sortie avec la même permutation. C'est immédiat par composition en vérifiant sur un comparateur. De plus, les deux propriétés (a) et (b) de la question 4 sont invariantes par la permutation miroir sur la moitié droite de l'état, qu'on applique ici. Donc en composant M avec cette permutation, on a bien ce qu'on veut.

Le comparateur obtenu s'écrit explicitement : $M' = \{1 \bullet\!\!\to 2m, 2 \bullet\!\!\to 2m-1, \ldots, m \bullet\!\!\to m+1\}$. On voit qu'il n'utilise que des comparateurs vers la droite.



En conclusion, si on prend le réseau qui trie les suites vallonnées, et qu'on remplace le tout premier multi-comparateur M par M', on obtient un réseau qui trie les suites demi-croissantes. En utilisant ce réseau dans la construction inductive, on n'a plus besoin d'inverser le tri sur la moitié droite, et on obtient un réseau qu'on peut tronquer à volonté comme dans le lemme.

Réseaux équivalents. On dit que deux réseaux sont *équivalents* s'il est possible de passer de l'un à l'autre en effectuant uniquement des opérations du type : insérer ou supprimer un multicomparateur vide ; ou bien, choisir un comparateur dans un des multicomparateurs constituant le réseau, et le déplacer dans le multicomparateur immédiatement avant ou après. (Cette opération n'est autorisée que si le multicomparateur d'arrivée reste valide au regard de la définition de multicomparateur.)
Chemins. Soit R un réseau sur n éléments, de profondeur k. Un *chemin* de R est une suite $(c_1, \ldots, c_{k+1})$ de positions (éléments de $[\![1, n]\!]$) telle que pour tout $i \in [\![1, k]\!]$, soit $c_{i+1} = c_i$, soit $c_i \bullet\!\!\to c_{i+1}$ ou $c_{i+1} \bullet\!\!\to c_i$ apparaît dans le i-ième multicomparateur de R (les deux cas ne sont pas exclusifs). La *distance* du chemin est le nombre d'indices $i \in [\![1, k]\!]$ tels que le second cas est vrai ; autrement dit, c'est le nombre de comparateurs par lesquels passe le chemin.

Question 6. Soit R un réseau quelconque. On se pose la question de trouver un réseau de profondeur p minimale parmi les réseaux équivalents à R.

a. Montrer que p est égale à la distance maximale parmi les chemins de R.
b. Proposer un algorithme en temps polynomial qui prend en entrée un réseau, et renvoie un réseau équivalent de profondeur minimale.

Solution : La question 6 est juste là pour occuper des candidats exceptionnels au cas où.

a. La question peut se traduire dans le langage des ordres partiels. Étant donné un réseau R fixé, on prend l'ensemble E de ses comparateurs comme ensemble de base. On définit l'ordre partiel $\leq$ sur E par $a < b$ s'il existe un chemin de R qui passe par a puis b, dans cet ordre. L'observation clé est qu'un chemin est une chaîne maximale pour cet ordre partiel, et réciproquement. De plus, découper le réseau en multicomparateurs revient exactement à partitionner E en antichaînes. La question qui est posée est donc un corollaire du lemme de Mirsky :

Lemme de Mirsky. Soit $(E, \leq)$ un ordre partiel. La cardinalité minimale d'une partition de E en antichaînes est égale à la longueur maximale d'une chaîne de E.

Preuve. Si on a une partition en antichaînes, chaque élément d'une chaîne donnée doit appartenir à une et une seule antichaîne de la partition. On déduit que la cardinalité de la partition doit être supérieure ou égale

à la longueur de n'importe quelle chaîne. Réciproquement, soit ℓ la longueur de la plus longue chaîne, et soit $1 \leq k \leq \ell$. On peut grouper ensemble les éléments de E tels que la plus longue chaîne qui part de l'élément est de longueur k. On obtient une partition de E en ℓ classes. Or ces classes sont des antichaînes. En effet, si $a, b \in E$ sont tels que la plus longue chaîne qui part de a (resp. b) est de longueur k, $a \leq b$ serait une contradiction parce que a serait le début d'une chaîne de longueur $k+1$.

b. Vu la réponse à la question précédente, cela revient à calculer, étant donné un ordre partiel, la longueur $f(c)$ de la plus longue chaîne qui part d'un élément donné c. On peut supposer que l'ordre est donné sous la forme de l'ensemble R des paires (a, b) de E^2 telles que $a < b$. En effet, on peut facilement calculer cette information en temps polynomial à partir du réseau.

   Il suffit ensuite de remonter l'ordre par la fin. On prend les éléments a tels que $(a, *)$ n'apparaît pas dans R (qui correspondent aux extrémités de chaînes maximales au sens de l'inclusion). Pour ces éléments, $f = 0$. On retire ensuite toutes les paires $(*, a)$ de R, pour les éléments a obtenus précédemment. On peut calculer f sur l'ensemble restant par récurrence ; et on lui ajoute 1 partout (parce qu'on a retiré un élément de toutes les chaînes maximales de l'ensemble d'origine).

   (On peut trouver ce processus sans jamais parler d'ordre partiel, mais c'est un langage commode pour le présenter.)

Raffinement d'évènements

On se place dans un langage récursif, purement fonctionnel, avec des définitions de types inductifs et un système de types polymorphes similaires à ceux d'OCaml. On pourra utiliser indifféremment du pseudocode fonctionnel ou de la syntaxe OCaml.
On suppose l'existence :

- de types de bases usuels, comme `Bool` le type des booléens, `Int` le type des entiers, `String` le type des chaînes de caractères, et `Unit` le type de l'unique élément ().
- de définitions de types paramétrés avec des constructeurs, comme :
  `type ArbreB A = Feuille | Noeud A (ArbreB A) (ArbreB A)`
- de types enregistrements $\{\mathtt{a_1} : \mathtt{T_1}, \ldots, \mathtt{a_n} : \mathtt{T_n}\}$ qui permettent de définir, par exemple, les points du plan avec `type Point = P {abscisse : Int, ordonnee : Int}` qu'on peut utiliser avec, par exemple, `origine = P {abscisse = 0, ordonnee = 0}`.
- de types listes natifs [`A`] avec les fonctions de manipulation usuelles (construction, longueur, tête, queue, ...).

Un *type de machine* est un type abstrait, qui représente l'état d'un système. On utilisera `M` pour représenter un tel type et `m` pour représenter des éléments de ce type, appelés *machines*. L'*invariant* d'une machine est un prédicat sur un type de machine, de type $\mathtt{M} \to \mathtt{Bool}$ qui décrit les propriétés qu'on attend d'un élément de `M`; on supposera qu'il est unique (pour chaque type `M`, on définit un unique invariant `inv_M`).
Un *évènement* d'un type de machine `M` d'entrée `A` et de sortie `B` est la donnée d'un prédicat sur les machines et les entrées, appelé *garde*, et d'une fonction des entrées et des machines dans les sorties et les machines :

```
type Evenement M A B  =  E {
                         garde : A → M → Bool
                         action : A → M → (B, M)
                         }
```

L'idée est qu'étant donnée une machine, un évènement et une entrée, si la garde de l'évènement est vraie pour la machine et l'entrée, on peut appliquer l'action de l'évènement à la machine et l'entrée et récupérer une nouvelle machine (la machine dans un nouvel état) et une sortie.
Un évènement est *sûr* quand l'implication suivante est vraie : "Si l'invariant de la machine est vrai, et la garde est vraie pour la machine et l'entrée, alors l'invariant de la nouvelle machine obtenue après l'action est vrai".

Question 1. On considère comme premier type de machine, les *compteurs bornés*. Un compteur borné est composé de deux entiers : une *valeur maximale*, fixe et positive, et une *valeur actuelle*, comprise entre 0 et la valeur maximale.

Donner une définition des compteurs bornés, de leur invariant, et d'évènements d'incrémentation et de décrémentation. Prouver leur sûreté.

Un *raffinement* $\mathcal{R}$ entre les types de machine $\mathtt{M_1}$ et $\mathtt{M_2}$ est une relation entre les éléments de $\mathtt{M_1}$ et $\mathtt{M_2}$ telle que si $(\mathtt{m_1}, \mathtt{m_2}) \in \mathcal{R}$ et l'invariant de $\mathtt{m_2}$ est vrai, alors l'invariant de $\mathtt{m_1}$ est vrai.

Question 2. On dispose d'une fonction `lift` : $\texttt{M}_2 \to \texttt{M}_1$. Quelle condition sur `lift` permet de construire un raffinement (non-trivial) entre $\texttt{M}_1$ et $\texttt{M}_2$?

L'inverse est-il vrai ?

Question 3. Donner un type de machine représentant les piles de taille bornée, puis exhiber un raffinement entre les compteurs bornés et ces piles bornées.

Soit un raffinement $\mathcal{R}$ entre $\texttt{M}_1$ et $\texttt{M}_2$.
Soit deux évènements $\texttt{e}_1$: `Evenement` $\texttt{M}_1$ $\texttt{A}_1$ $\texttt{B}_1$ et $\texttt{e}_2$: `Evenement` $\texttt{M}_2$ $\texttt{A}_2$ $\texttt{B}_2$.
On dit que $\texttt{e}_2$ *raffine* $\texttt{e}_1$ *vis-à-vis* de $\mathcal{R}$ (ou simplement e_2 *raffine* e_1 quand il est clair de quel raffinement $\mathcal{R}$ on parle) quand il existe deux fonctions
$\texttt{lift\_in} : \texttt{A}_2 \to \texttt{A}_1$
$\texttt{lift\_out} : \texttt{B}_2 \to \texttt{B}_1$
qui vérifient :

- *Renforcement* : Pour tout $\texttt{x}, \texttt{m}_1, \texttt{m}_2$ tels que $\texttt{m}_1 \ \mathcal{R} \ \texttt{m}_2$, si $\texttt{e}_2$`.garde x` $\texttt{m}_2$, alors $\texttt{e}_1$`.garde (lift_in x)` $\texttt{m}_1$.
- *Simulation* : Pour tout $\texttt{x}, \texttt{m}_1, \texttt{m}_2$ tels que $\texttt{m}_1 \ \mathcal{R} \ \texttt{m}_2$ et $\texttt{e}_2$`.garde x` $\texttt{m}_2$,
  si $(\texttt{b}_2, \texttt{m}'_2) = \texttt{e}_2$`.action x` $\texttt{m}_2$
  et $(\texttt{b}_1, \texttt{m}'_1) = \texttt{e}_1$`.action x` $\texttt{m}_1$,
  alors $\texttt{m}'_1 \ \mathcal{R} \ \texttt{m}'_2$ et $\texttt{b}_1 =$ `lift_out` $\texttt{b}_2$.

Question 4. Montrer que si $\texttt{e}_2$ raffine $\texttt{e}_1$ et $\texttt{e}_1$ est sûr, alors $\texttt{e}_2$ est sûr.

Question 5. Donner deux évènements pour les piles bornées qui raffinent les évènements des compteurs bornés vis-à-vis du raffinement de la Question 3.

On définit le type `Mealy A B` $= \texttt{M}\ \{\texttt{run} : \texttt{A} \to (\texttt{B}, \texttt{Mealy A B})\}$.

Question 6. Le type `Mealy A B` est-il un type inductif ?

Question 7. Exprimer la définition de l'empilage dans une pile bornée en utilisant le type `Mealy A B` (pour un `A` et un `B` bien choisis).

Question 8. Tenter d'en induire une fonction `evt_vers_Mealy` qui prend un évènement et le transforme en *machine de Mealy* (en une entité qui utilise un type `Mealy A B`).

Résolution efficace de systèmes linéaires creux

Notation. Soit $\mathbb{K} = \mathbb{Z}/2\mathbb{Z}$ le corps à deux éléments. On note $\deg(P)$ le degré d'un polynôme $P \in \mathbb{K}[X]$. Étant donné $P, Q \in \mathbb{K}[X]$, on note P rmod Q le reste dans la division euclidienne de P par Q. Un vecteur $x \in \mathbb{K}^n$ est implicitement vu commes un vecteur colonne ; sa transposée x^t est un vecteur ligne. La cardinalité d'un ensemble E est notée $|E|$.
Poids de Hamming. Le *poids de Hamming* d'un vecteur $v \in \mathbb{K}^n$, noté $\mathsf{H}(v)$, est le nombre de coordonnées non-nulles du vecteur. De même, le poids de Hamming d'une matrice est le nombre d'entrées non-nulles de la matrice.
Problème de résolution linéaire. Le *problème de résolution linéaire* prend en entrée une matrice $M \in \mathbb{K}^{n\times n}$, et un vecteur $b \in \mathbb{K}^n$. Il demande de trouver un vecteur $x \in \mathbb{K}^n$ tel que $Mx = b$, si un tel x existe, ou de renvoyer $\perp$ sinon.
Représentation par indice. Soit $M = (m_{i,j}) \in \mathbb{K}^{m\times n}$ une matrice. La *représentation par indice* de M est l'ensemble $\{(i, j) : m_{i,j} = 1\}$. Noter que pour m et n fixés, cette représentation définit la matrice de manière unique, en utilisant $O(\mathsf{H}(M))$ entiers.
Dans tout l'énoncé, on supposera que la matrice M en entrée du problème de résolution linéaire est donnée sous forme de sa représentation par indice.

Question 1. Esquisser sans détailler un algorithme en temps $O(n^3)$ pour résoudre le problème de résolution linéaire.

Question 2. Soit $G = (V, E)$ un graphe formé d'un ensemble de sommets V, et un ensemble d'arêtes E. Les arêtes de E sont des paires non-ordonnées $\{v, w\}$ d'éléments de V, avec $v \neq w$. (On suppose que E est implémenté sous forme d'une liste de paires.) Proposer un algorithme en temps $O(|V| + |E|)$ qui calcule la partition de V en composantes connexes.

Indication : on pourra d'abord créer un dictionnaire qui à chaque sommet associe la liste de ses voisins.

Question 3.

a. Soit $M \in \mathbb{K}^{n\times n}$ une matrice telle que chaque *ligne* de M a un poids de Hamming au plus 2. Proposer un algorithme qui résout le problème de résolution linéaire pour une telle matrice M, en temps $O(n)$.

b. Même question si chaque *colonne* de M, et non plus chaque ligne, a un poids de Hamming au plus 2.

Indication : Dans les deux cas, il peut être utile de définir un graphe bien choisi (pas nécessairement le même), et de chercher d'abord un algorithme en temps $O(n^2)$, avant de raffiner en temps linéaire.

Polynôme minimal d'une matrice. Étant donné une matrice $M \in \mathbb{K}^{n\times n}$, son *polynôme minimal* est le polynôme $P = \sum_{i=0}^{d} \lambda_i X^i \in \mathbb{K}[X]$ de degré $d \leq n$ minimal tel que $P(M) = \sum_{i=0}^{d} \lambda_i M^i = 0$.

Question 4. Proposer un algorithme qui résout le problème de résolution linéaire en temps $O(n^2 + n\mathsf{H}(M))$, lorsque la matrice M en entrée du problème est inversible, et que son polynôme minimal est connu.

Indication : trouver une expression de M^{-1} sous forme d'un polynôme en M.

Matrice de Hankel. Soit $(a_0, \ldots, a_{2n-1})$ une suite d'éléments de $\mathbb{K}$. La *matrice de Hankel* générée par cette suite est la matrice $M = (m_{i,j}) \in \mathbb{K}^{n\times(n+1)}$, avec n lignes et $n+1$ colonnes, telle que $\forall i, j, m_{i,j} = a_{i+j}$ (on numérote en partant de 0).

$$M = \begin{pmatrix} a_0 & a_1 & a_2 & \ldots & a_n \\ a_1 & a_2 & a_3 & \ldots & a_{n+1} \\ a_2 & a_3 & a_4 & \ldots & a_{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & a_n & a_{n+1} & \ldots & a_{2n-1} \end{pmatrix}$$

Problème de Hankel. Le *problème de Hankel* prend en entrée une suite $(a_0, \ldots, a_{2n-1})$ d'éléments de $\mathbb{K}$. Il demande de trouver un vecteur $v \in \mathbb{K}^{n+1}$ non nul tel que $Mv = 0$, où M est la matrice de Hankel générée par la suite $(a_0, \ldots, a_{2n-1})$.
Problème de réduction polynomiale. Le *problème de réduction polynomiale* prend en entrée une suite $(a_0, \ldots, a_{2n-1})$ d'éléments de $\mathbb{K}$. Il demande de trouver un polynôme $V \in \mathbb{K}[X]$ non nul de degré au plus n tel que $\deg(VA \bmod X^{2n}) < n$, où $A = \sum_{i=0}^{2n-1} a_i X^i \in \mathbb{K}[X]$.

Question 5. Montrer que si on considère le problème de Hankel et le problème de réduction polynomiale tous les deux avec en entrée la même suite $(a_0, \ldots, a_{2n-1})$, alors $v = (v_0, \ldots, v_n)$ est solution du problème de Hankel si et seulement si $V = \sum_{i=0}^{n} v_i X^{n-i}$ est solution du problème de réduction polynomiale.

Polynôme minimal d'une suite. Étant donné une suite $(v_0, \ldots, v_m)$ d'éléments de $\mathbb{K}$, le *polynôme minimal* de cette suite est le polynôme $P = \sum_{i=0}^{d} \lambda_i X^i \in \mathbb{K}[X]$ de degré $d \leq m$ minimal tel que $\forall k \leq m - d, \sum_{i=0}^{d} \lambda_i u_{k+i} = 0$.

Question 6. Supposons qu'on sait résoudre le problème de réduction polynomiale en temps $O(n^2)$. On souhaite maintenant résoudre efficacement le problème de résolution linéaire pour les matrices de faible poids de Hamming. Pour cela, on introduit deux hypothèses simplificatrices.

— M est inversible, et son polynôme minimal est de degré n.
— Si u, v sont deux vecteurs tirés uniformément et indépendamment dans $\mathbb{K}^n$, alors avec probabilité au moins $1/4$, le polynôme minimal de la suite $(u^t M^i v)_{i=0}^{2n-1}$ est de degré n.

a. Montrer qu'avec probabilité au moins $1/4$, le polynôme minimal de la suite $(u^t M^i v)_{i=0}^{2n-1} \in \mathbb{K}^{2n}$ est égal au polynôme minimal de M.
b. En déduire un algorithme qui résout le problème de résolution linéaire pour de telles matrices M en temps $O(n^2 + n\mathsf{H}(M))$ en espérance.

Question 7. Donner un algorithme qui résout le problème de réduction polynomiale en temps $O(n^2)$.

Complexité arithmétique de la factorisation

Dans cet exercice, on s'intéresse à la *complexité arithmétique* des algorithmes. On utilise les opérations de base suivantes sur des entiers relatifs : addition, soustraction, multiplication, division euclidienne. Le coût de chacune de ces opérations est 1, *quelle que soit la taille des entiers en entrée et en sortie.* Pour écrire les algorithmes, on se contentera d'un pseudocode simple permettant de compter facilement ces opérations.

On s'intéresse au problème de la factorisation d'un entier :

Entrée : $N = P \times Q$ de taille n bits (c'est-à-dire $N < 2^n$) où P et Q sont deux nombres premiers

Sortie : P, Q

Notre objectif est de démontrer le résultat suivant :

Il existe un algorithme de factorisation utilisant $\mathcal{O}(n)$ opérations arithmétiques.

Question 1. Donner un algorithme pour calculer l'exponentielle en $\mathcal{O}(n)$ opérations arithmétiques :

Entrée : N, K de taille n bits

Sortie : N^K

Solution : Il s'agit de l'algorithme d'exponentiation rapide.

Question 2. Donner un algorithme pour calculer le PGCD en $\mathcal{O}(n)$ opérations arithmétiques :

Entrée : N, M de taille n bits

Sortie : $PGCD(N, M)$

Solution : Il s'agit de l'algorithme d'Euclide. Nous démontrons qu'il termine en $\mathcal{O}(n)$ opérations.

Pour se faire il suffit de noter qu'après deux itérations, les nombres sont réduits de moitié au moins. En effet, si on suppose que $N \leq M$: $PGCD(N, M) = PGCD(M \bmod N, N)$ où $M \bmod N \leq N$, et $\min(N, M \bmod N) \leq M/2$ tandis que $\max(N, M \bmod N) \leq N$. La première inégalité n'est pas triviale, elle se déduit de $N + (M \bmod N) \leq M$ (car $N \leq M$) et donc que le plus petit des deux est inférieur à $M/2$.

Par conséquent la somme des tailles des deux nombres décroît d'au moins 1 par itération de l'algorithme, et on termine (i.e., atteindre 0) en $\mathcal{O}(n)$ étapes. Chaque étape coûte $\mathcal{O}(1)$ opérations arithmétiques.

Question 3. En supposant qu'il existe un algorithme de calcul de la factorielle :

Entrée : N de taille n bits

Sortie : $N!$

en $F(n)$ opérations arithmétiques, montrer qu'il existe un algorithme de factorisation de complexité $\mathcal{O}(n \times F(n))$ (une preuve de sa correction n'est pas exigée, on se concentrera sur sa description).

Solution : On recherche le plus petit i tel que $i! \mod N = 0$. On aura alors $i = \max(P, Q)$.

Pour calculer si $i! \mod N = 0$, il suffit de calculer un reste dans la division euclidienne par N, qui ne coûte qu'une opération arithmétique par hypothèse de l'énoncé.

Soit $i_0 = \max(P, Q)$ la valeur que l'on recherche. On effectue une recherche dichotomique de i_0 sur l'intervalle $[1; N]$. À chaque étape, on regarde si $i! \bmod N = 0$ pour le milieu de l'intervalle, si oui on réduit la borne supérieure, sinon on augmente la borne inférieure.

Comme l'intervalle de départ est de taille N, on a $\mathcal{O}(\log N) = \mathcal{O}(n)$ étapes de calcul. Chacune requiert un calcul de factorielle et une opération arithmétique.

Notons que l'algorithme ainsi obtenu ne suffit pas à résoudre le problème de l'énoncé, car même avec un calcul de la factorielle en $\mathcal{O}(n)$ opérations, on obtiendrait $\mathcal{O}\big(n^2\big)$ opérations.

Question 4. Dans cette question, on s'intéresse au calcul d'un coefficient binomial.

Entrée : N, M de taille n bits, $M \leq N$

Sortie : $\binom{N}{M}$

Montrer que ce calcul peut être fait en $\mathcal{O}(n)$ opérations arithmétiques. On pourra utiliser l'identité suivante :

$$\forall k, \ell \in \mathbb{N}, \quad (2^\ell + 1)^k = \sum_{j=0}^{k} \binom{k}{j} 2^{\ell j}$$

Solution : On va commencer par implémenter l'opération de *sélection* :

Entrée : N de taille n bits, deux indices $m_0 \leq m_1 \leq n$

Sortie : l'entier obtenu en écrivant les bits de N situés entre m_0 et m_1

L'opération en elle-même est : $(N \mod 2^{m_1} - N \mod 2^{m_0})/2^{m_0}$. Pour simplifier l'exercice, il est raisonnable d'admettre dans un premier temps que la sélection est toujours en temps $\mathcal{O}(1)$. Cependant, on a besoin de connaître 2^{m_0} et 2^{m_1}. Il faudra donc vérifier par la suite que tous les indices nécessaires aux sélections peuvent être précalculés efficacement.

Pour résoudre le problème il suffit maintenant de :

1. Choisir une valeur de ℓ suffisamment grande : il suffit d'avoir $\ell \geq k$ dans l'identité, donc on peut prendre $\ell = k = N$. On a donc :
$$(2^N + 1)^N = \sum_{j=0}^{N} \binom{N}{j} 2^{Nj}$$
On veut donc sélectionner les bits des positions NM à $NM + N$.
2. Calculer $2^N + 1$ en $\mathcal{O}(n)$ opérations arithmétiques par exponentiation rapide.
3. Calculer $(2^N + 1)^N$ en $\mathcal{O}(n)$ opérations arithmétiques par exponentiation rapide.
4. Calculer 2^{NM} et 2^{NM+N} à l'aide de plusieurs exponentiations rapides (d'abord 2^N, puis $(2^N)^M$).
5. Effectuer l'opération de sélection.

Question 5. En utilisant la question précédente, montrer qu'on peut calculer la factorielle en $\mathcal{O}(n)$ opérations arithmétiques.

Solution : On utilise le fait que pour tout entier pair N :

$$N! = \binom{N}{N/2} ((N/2)!)^2 \ .$$

Pour commencer on peut décrire un algorithme récursif trivial pour calculer $K!$:

— Si K est impair, calculer $K! = K \times (K-1)!$

— Si K est pair, calculer $\binom{K}{K/2}$ avec l'identité de la question 4, et renvoyer $K! = \binom{K}{K/2}((K/2)!)^2$.

Comme le calcul du coefficient binomial prend $\mathcal{O}(n)$ opérations, l'algorithme complet prend $\mathcal{O}\big(n^2\big)$ opérations. Cependant, dans le calcul du coefficient binomial, la complexité est dominée par l'exponentiation (ainsi que les puissances de 2 pour la sélection). On va donc modifier notre algorithme récursif pour qu'il calcule *en même temps* l'exponentiation et la factorielle.

Construisons donc un algorithme récursif qui calcule à la fois $K!$, $(2^N+1)^K$ et 2^{KN}. Notons ici que N est devenu un paramètre fixe, et l'algorithme fonctionne pour $K \leq N$. On l'appellera ensuite avec $K = N$.

En précalcul, on commence par calculer 2^N et $2^N + 1$ en $\mathcal{O}(n)$ opérations, dont on aura besoin pour la suite. Ensuite :

1. Si K est impair, on a $K! = K \times (K-1)!$ et $(2^N+1)^K = (2^N+1)^{K-1}(2^N+1)$, et $2^{KN} = 2^{(K-1)N}2^N$, et on utilise un appel récursif.
2. Si K est pair, on utilise :

$$K! = \binom{K}{K/2}((K/2)!)^2 \text{ et } (2^N+1)^K = ((2^N+1)^{K/2})^2 \text{ et } 2^{NK} = (2^{N(K/2)})^2$$

   — On utilise un appel récursif pour calculer $(2^N+1)^{K/2}$ et $(K/2)!$ et $2^{N(K/2)}$
   — On utilise deux opérations arithmétiques pour calculer $(2^N+1)^K$ et $((K/2)!)^2$ et 2^{NK}
   — On calcule $\binom{K}{K/2}$ à l'aide d'une sélection dans les bits de $(2^N+1)^K$ (on sélectionne les bits des positions $(K/2)N$ à $(K/2)N+N$)
   — On en déduit $K!$ par une multiplication

Question 6. Est-ce optimal ?

Solution : Il faut remarquer que si les nombres donnés en entrée sont plus petits que 2^n, alors tout algorithme effectuant c opérations arithmétiques ne peut calculer que des nombres plus petits que $(2^n)^{2^c}$. En effet, chaque opération ne peut au maximum que doubler la taille des nombres actuellement en mémoire.

Prenons $2^n > N \geq 2^{n-1}$. On a asymptotiquement $N! = \Theta(\sqrt{2^{n-1}}2^{(n-1)2^{n-1}}e^{-2^{n-1}})$ d'après la formule de Stirling. Tout algorithme de calcul de la factorielle nécessite donc au moins $\mathcal{O}(n)$ opérations arithmétiques. À constante près, notre algorithme est donc optimal.

Question 7. En adaptant l'algorithme de la question 5, montrer qu'on peut calculer le plus petit entier i tel que $(2^i)! \neq 0 \mod N$ et $(2^{i+1})! = 0 \mod N$ (sur une entrée N de taille n bits) en $\mathcal{O}(n)$ opérations arithmétiques.

Solution : Soit i le plus petit entier tel que $2^i > N$.

On effectue un appel à la fonction récursive de la question 5 pour calculer $(2^i)!$, avec la modification suivante : à chaque appel récursif, on mémoise la valeur de sortie. On va donc calculer toutes les factorielles $(2^j)!$ pour $1 \leq j \leq i$.

Pour chacune d'entre elles, on calcule une division euclidienne par N et on trouve le résultat.

Question 8. Donner un algorithme de factorisation utilisant $\mathcal{O}(n)$ opérations arithmétiques.

Solution : On commence par utiliser la question précédente. Soit i_0 tel que $(2^{i_0})! \neq 0 \mod N$ et $(2^{i_0+1})! = 0 \mod N$. Nous voulons utiliser la même méthode que la question 3 : trouver le plus petit i tel que $i! \mod N = 0$. Pour rendre le problème un peu plus facile, nous allons trouver le plus petit entier *pair* satisfaisant cette relation ; il suffira ensuite de tester i et $i-1$.

Nous avons donc un intervalle $[2^{i_0}, 2^{i_0+1}]$ à chercher dichotomiquement. On commence par éliminer le cas où $PGCD((2^{i_0})!, N) > 1$. En effet, si c'est le cas, comme N ne divise pas $(2^{i_0})!$, la valeur $PGCD((2^{i_0})!, N)$ nous donnerait un facteur non trivial de N, ce qui termine l'algorithme.

Dans la suite on considère donc : $PGCD((2^{i_0})!, N) = 1$.

Soit I pair dans l'intervalle $[2^{i_0}; 2^{i_0+1}]$. On a :

$$I! = \binom{I}{I/2}((I/2)!)^2$$

On sait que $PGCD(N, (I/2)!) = 1$, en effet $I/2 < 2^{i_0}$ et $PGCD((2^{i_0})!, N) = 1$ par hypothèse. Donc N divise $I!$ si et seulement si N divise $\binom{I}{I/2}$.

Au cours de notre recherche dichotomique dans l'intervalle $[2^{i_0}; 2^{i_0+1}]$, nous allons donc calculer un certain nombre de coefficients $\binom{I}{I/2}$. Il faut montrer que tous ces coefficients peuvent être calculés en $\mathcal{O}(n)$ opérations. Comme précédemment, on va utiliser la sélection dans le nombre $(2^N+1)^I$, et comme précédemment, il faut principalement vérifier qu'on peut amortir le coût de l'exponentiation (cela vaut à la fois pour $(2^N+1)^I$ et pour 2^{NI}, nous écrivons seulement le premier dans la suite).

La recherche dichotomique va produire $\mathcal{O}(n)$ valeurs de I, chacune différant de la précédente par une puissance de 2 qui va en se réduisant. En ayant précalculé les puissances $(2^N+1)^{2^j}$ pour tous j, on peut donc obtenir la nouvelle valeur $(2^N+1)^I$ en une seule opération arithmétique, et calculer ensuite le coefficient binomial suivant par sélection.

Voici l'algorithme complet :

Entrée : i_0
Sortie : $I \in [2^{i_0}; 2^{i_0+1}]$ tel que $I \bmod N = 0$ et $I - 2 \bmod N \neq 0$.

1: Calculer $2^N + 1$
2: Pour $j = 1$ à i_0, calculer $X_{2^j} = (2^N+1)^{2^j}$
3: $L, R \leftarrow 2^{i_0}, 2^{i_0+1}$
4: $I \leftarrow (L+R)/2$ ▷ Invariant : I reste pair
5: Calculer $X := (2^N+1)^I$
6: **tant que** $R - L \geq 4$ **faire** ▷ Invariant : $R - L$ est une puissance de 2
7: Calculer $\binom{I}{I/2}$ par sélection sur X
8: **si** $\binom{I}{I/2} \mod N = 0$ **alors**
9: $R' \leftarrow I$ ▷ On déplace R vers I
10: $I' \leftarrow I + (R-L)/4$
11: $X' \leftarrow X \times X_{(R-L)/4}$
12: $R, I, X \leftarrow R', I', X'$
13: **sinon**
14: $L' \leftarrow I$ ▷ On déplace L vers I
15: $I' \leftarrow I - (R-L)/4$
16: $X' \leftarrow X / X_{(R-L)/4}$
17: $L, I, X \leftarrow L', I', X'$
18: **fin si**
19: **fin tant que**

Graphes d'influence

On note $[\![a, b]\!]$ l'intervalle entier $\{a, a+1, \ldots, b\}$. On note $|S|$ la cardinalité d'un ensemble S.
Graphe simple. Dans tout l'énoncé, on considère des graphes simples : c'est-à-dire qu'ils sont définis par un ensemble de sommets V, et un ensemble d'arêtes E, et chaque arête est une paire non ordonnée $\{i, j\}$ d'éléments distincts de V.
Isomorphisme de graphe. Deux graphes $G = (V, E)$ et $G' = (V', E')$ sont dits isomorphes s'il existe une bijection $\varphi : V \to V'$ telle que $E' = \{\{\varphi(i), \varphi(j)\} : \{i, j\} \in E\}$.
Graphe d'influence. Soit $\vec{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$. Au vecteur $\vec{v}$, on associe le graphe $\mathsf{Infl}(\vec{v}) = (V, E)$ avec pour sommets $V = [\![1, n]\!]$, et pour arêtes $E = \{\{i, j\} : |v_i - v_j| < 1\}$. Un graphe $G = (V, E)$ est un *graphe d'influence* s'il existe $\vec{v} \in \mathbb{R}^{|V|}$ tel que G est isomorphe à $\mathsf{Infl}(v)$.

Informellement, on peut penser aux sommets i d'un graphe d'influence comme étant des *agents* ; v_i est l'*opinion* du i-ième agent ; et un agent *influence* un autre agent si leurs opinions sont suffisamment proches.

Question 1. Donner un exemple simple de famille infinie de graphes d'influence (deux à deux non isomorphes). Donner une exemple de graphe qui n'est pas un graphe d'influence.

Question 2.

1. Montrer que tout graphe d'influence $G = (V, E)$ est un *graphe d'incomparabilité* : c'est-à-dire qu'il existe un ordre partiel $\preceq$ sur V tel que $\{i, j\} \in E$ si et seulement si $i \not\preceq j$ et $j \not\preceq i$.
2. Montrer que tout graphe d'influence $G = (V, E)$ est *cordal* : c'est-à-dire que pour toute suite de sommets $k_1, \ldots, k_m$ formant un cycle de longueur $m > 3$, il existe $i, j \in [\![1, m]\!]$ tels que $|j - i| \neq 1 \mod m$ et $\{k_i, k_j\} \in E$.

Pour $G = (V, E)$ un graphe et $i \in V$, on note $G[i] = \{j \in V : \{i, j\} \in E\} \cup \{i\}$ le *voisinage* de i.
Soit :

$$f : \mathbb{R}^n \to \mathbb{R}^n$$
$$(v_i)_{i \in [\![1,n]\!]} \mapsto \left(\frac{\sum_{j \in \mathsf{Infl}(\vec{v})[i]} v_j}{|\mathsf{Infl}(\vec{v})[i]|} \right)_{i \in [\![1,n]\!]}.$$

Autrement dit, f remplace l'opinion de chaque agent par la moyenne des opinions des agents qui l'influencent (lui-même compris).
Chaîne d'influence. Une *chaîne d'influence* est une suite de vecteurs $\vec{v}^k \in \mathbb{R}^n$ telle que pour tout $i \in \mathbb{N}$, $\vec{v}^{k+1} = f(\vec{v}^k)$.
Dans toute la suite, on supposera que le vecteur initial $\vec{v}^0$ d'une chaîne d'influence est ordonné par ordre croissant : $v_1^0 \leq v_2^0 \leq \cdots \leq v_n^0$.

Question 3. Soit $(\vec{v}^k)_{k \in \mathbb{N}}$ une chaîne d'influence.

1. Montrer que pour tout $k \in \mathbb{N}$, $v_1^k \leq v_2^k \leq \cdots \leq v_n^k$.
2. Montrer que $(v_1^k)_{k \in \mathbb{N}}$ est croissante, et $(v_n^k)_{k \in \mathbb{N}}$ est décroissante.

Question 4. Soit $(\vec{v}^k)_{k \in \mathbb{N}}$ une chaîne d'influence. Pour $i \in [\![1, n]\!]$ fixé, on définit la suite u_k par $u_k = 1$ si l'arête $\{i, i+1\}$ existe dans $\mathsf{Infl}(\vec{v}^k)$, et $u_k = 0$ sinon. Montrer que $(u_k)_{k \in \mathbb{N}}$ est stationnaire.

Rappel. Une suite $(a_k)_{k \in \mathbb{N}}$ est dite *stationnaire* s'il existe $b \in \mathbb{N}$ tel que $\forall k \geq b, a_k = a_b$.

Question 5. Montrer que pour toute chaîne d'influence, après un nombre fini d'itérations de f, plus aucun agent ne change d'opinion. (Autrement dit, toute chaîne d'influence est stationnaire.)

Question 6. Donner un algorithme qui, étant donné en entrée une suite stationnaire de graphes $(G_i)_{i\in\mathbb{N}}$ (représentée par un segment initial au-delà duquel la suite reste constante) :

— termine et renvoie `VRAI` dans le cas où il existe $\vec{v}^0 \in \mathbb{R}^n$ tel que $(G_i)_{i\in\mathbb{N}}$ est la suite des graphes d'influence de la chaîne d'influence initiée par $\vec{v}^0$;

— renvoie `FAUX` ou ne termine pas dans le cas contraire.

Question 7. Montrer que pour tout $n \in \mathbb{N}$, il existe un nombre fini de suites de graphes qui sont les graphes d'influence successifs d'une chaîne d'influence dans $\mathbb{R}^n$.

Calcul de l'arbre de dominance

Dans ce sujet, nous nous intéressons à des graphes orientés enracinés, i.e., $G = (V, E, r)$ où V est l'ensemble fini des sommets, E l'ensemble des arêtes orientées, r le sommet racine. Nous supposerons que pour tout sommet $v \in V$ il existe un chemin de r à v, noté $r \to^* v$.
Définition. Étant donné un graphe $G = (V, E, r)$, on dit que $x \in V$ *domine* $y \in V$, noté $x > y$, si, $x \neq y$ et pour tout chemin $r = s_0 \to s_1 \to \ldots \to s_n = y$ dans G, il existe i tel que $s_i = x$.
On dit que x est un *dominant immédiat* de y, noté $idom(y)$, si $x > y$ et pour tout $z \in V$ tel que $z > y$ et $z \neq x$, on a $z > x$.
Définition. Étant donné un graphe $G = (V, E, r)$. On note $A(G) = (V, E_d, r)$ où $E_d = \{x \to y \mid x, y \in V \text{ et } x = idom(y)\}$.
$A(G)$ est appelé l'*arbre de dominance* de G.
L'objectif de ce sujet est de calculer efficacement $A(G)$.

Question 1. Donner l'arbre de dominance du graphe donné en Figure 1.

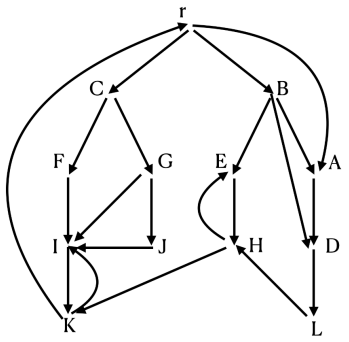


FIGURE 1 – Graphe de la question 1

Question 2. Montrer que $A(G)$ est bien défini, i.e., $idom(x)$ est bien défini pour tout $x \neq r$ et $A(G)$ forme un arbre enraciné en r.

Question 3. a. Donner un algorithme basé uniquement sur des parcours, permettant de calculer $A(G)$.
b. Quelle est sa complexité ?

Soit $G = (V, E, r)$ un graphe. Pour chaque sommet $x \in V$, nous notons $i(x)$ l'indice de visite lors d'un parcours en profondeur issu de r. Nous notons $T(G)$ l'arbre couvrant obtenu à la fin du parcours, i.e., le graphe (V, E_T, r) où E_T est composé des arcs de liaisons durant le parcours.

Définition. Soit $w \neq r$. On note $sdom(w) = argmin_v\{i(v) \mid \text{il existe un chemin } v = v_0 \to \ldots \to v_k = w \text{ dans } G \text{ tel que } i(v_j) > i(w) \text{ pour } 1 \leq j \leq k-1\}$.

Question 4.
a. Donner l'arbre couvrant ainsi que les indices de visite obtenus via un parcours en profondeur depuis r dans le graphe donné en Figure 1.
b. Préciser pour chaque sommet w, le $sdom(w)$ correspondant.

Question 5. Soient $x, y \in V$ tels que $i(x) \leq i(y)$. Montrer que tout chemin $x \to^* y$ dans G contient un sommet a qui est un ancêtre commun à x et y dans $T(G)$.

Question 6. Montrer que pour tout $w \neq r$, $sdom(w)$ est un ancêtre de w dans $T(G)$.

Question 7. Montrer que :

$$sdom(w) = argmin(\quad \{i(v) \mid (v,w) \in E \text{ and } i(v) < i(w)\} \\ \cup \{i(sdom(u)) \mid i(u) > i(w) \text{ and } \exists (v \to w) \in E \text{ tel que } u \to^* v \text{ dans } T(G)\} \ )$$

Axiome. soit $w \neq r$ et $u = argmin_u\{i(sdom(u)) \mid sdom(w) \to^+ u \to^* w$ dans T(G)$\}$. On a

$$idom(w) = \begin{cases} sdom(w) & \text{si } sdom(w) = sdom(u) \\ idom(u) & \text{sinon} \end{cases}$$

Question 8. a. Donner un algorithme qui permet de calculer $idom(w)$ pour tout $w \neq r$.
Conseil : nous pourrons envisager de re-construire $T(G)$ en partant d'une forêt de sommets et en les fusionnant lors d'un parcours des sommets dans un ordre décroissant de leur index.

b. Étudier la complexité de l'algorithme. Nous tiendrons compte de la structure de données utilisée pour manipuler la forêt.

Forme SSA – placement optimal des fonctions de jointure

Nous considérons le petit langage de programmation composé des commandes suivantes :

Commandes :

$$
\begin{array}{rcl}
P, Q & := & \textbf{return } expr_{int} \\
& | & x := expr_{int} \text{ où } x \in \mathcal{X} \\
& | & P; Q \\
& | & \textbf{if } expr_{bool} \textbf{ then } P \textbf{ else } Q \textbf{ endif} \\
& | & \textbf{repeat } P \textbf{ until } expr_{bool}
\end{array}
$$

Expressions :

$$expr_{bool} := expr_{int} \circ expr_{int} \text{ avec } \circ \in \{==, <, \leq\}$$

Expressions d'entiers :

$$
\begin{array}{rcl}
expr_{int} & := & n \in \mathbb{Z} \\
& | & x \in \mathcal{X} \\
& | & expr_{int} + expr_{int} \\
& | & expr_{int} * expr_{int} \\
& | & expr_{int} \,/\, expr_{int} \\
& | & expr_{int} \,\%\, expr_{int}
\end{array}
$$

Définition. Étant donné un programme, son graphe de flot de contrôle (CFG – control flow graph) est le graphe orienté $G = (V, E)$ tel que V est l'ensemble des points de contrôle du programme (c.-à-d., les commandes) et $s_1 \to s_2 \in E$ si s_2 peut suivre s_1 lors de l'exécution du programme.

Définition. Un graphe de flot de contrôle $G = (V, E)$ est sous forme SSA (single-assignment form) si, pour toute variable $x \in \mathcal{X}$, il existe au plus une *affectation de* x, c.-à-d., au plus un sommet $s \in V$ de la forme $x := expr$.
Nous supposerons l'existence d'un sommet *Entry* relié à l'ensemble des points d'entrées du programme.

Une façon classique de mettre un CFG quelconque sous forme SSA est de dupliquer les variables utilisées plusieurs fois. Une difficulté peut alors apparaître après les boucles **repeat ... until** ou les conditionnelles **if-then-else** ; comment savoir quelle variable utiliser ?
Pour résoudre ce problème, il est habituel d'ajouter au CFG une *fonction de jointure*, qui prend la forme d'une affectation $x_i := \varphi(x_{i_1}, \ldots, x_{i_n})$ où $\varphi(\cdot)$ représente une fonction abstraite (c.-à-d., sans sémantique fixée) lorsque nécessaire. L'arité de la fonction $\varphi(\cdot)$ dans un sommet s du CFG dépend du nombre d'affectations de x dont dépend s, c.-à-d., du nombre de sommets s' contenant une affectation de x et tel que s est accessible depuis s' dans le CFG.

L'objectif de ce sujet est de trouver un placement optimal des fonctions de jointure afin d'en réduire le nombre.

Question 1. a. Donner le CFG correspondant au programme décrit en Figure 1.
b. Mettre ce CFG sous forme SSA.

```
x := 42;
i := 1;
repeat
  if x%2 == 0 then
    y := x/2
  else y := 3 * x + 1
  endif;
  x := y;
  i := i + 1;
until i < 10;
return x
```

FIGURE 1 – Programme calculant le 10ième terme de la suite de Syracuse commençant à 42.

Étant donné un CFG $G = (V, E)$, on dit que $X \in V$ *domine* $Y \in V$, noté $X \geq Y$, si, pour tout chemin $Entry = s_0 \to s_1 \to \ldots \to s_n = Y$ dans G, il existe i tel que $s_i = X$.
On dit que X *domine strictement* Y, noté $X > Y$, si X domine Y et $X \neq Y$.
On note $X \not> Y$ si X ne domine pas strictement Y.
On note $idom(Y)$ le plus proche dominant strict de Y.

Question 2. a. Montrer que la relation de *domination* est une relation réflexive transitive.

b. Montrer que si $X \to^* Y$ est un chemin simple (c.-à-d., sans cycle) et $X > Y$ alors $X > Z$ pour tout Z appartenant au chemin (et différent de X).

c. En déduire que $idom(X)$ est bien définie pour tout $X \neq Entry$.

Soit $G = (V, E)$ un CFG et $G_d = (V, E_d)$ son arbre de dominance associé, c.-à-d., $E_d = \{X \to Y \mid X, Y \in V \text{ et } X = idom(Y)\}$.
On note :

— $DF(X) = \{Y \mid \exists P \in V.\ (P \to Y) \in E \wedge X \geq P \wedge X \not> Y\}$

— $DF_{local}(X) = \{Y \mid (X \to Y) \in E \wedge X \not> Y\}$

— $DF_{up}(Z) = \{Y \in DF(Z) \mid idom(Z) \not> Y\}$

Question 3. Montrer que $DF(X) = DF_{local}(X) \cup \bigcup_{(X \to Z) \in E_d} DF_{up}(Z)$.

Question 4. En déduire un algorithme permettant de calculer $DF(X)$ ainsi que sa complexité. Nous supposerons que nous avons déjà pré-calculé l'arbre de dominance.

Soient $p : X_0 \to^+ X_{n_p}$ et $q : Y_0 \to^+ Y_{n_q}$ deux chemins non-nuls dans un CFG. On dit que p *et* q *convergent vers* Z si : (1) $X_0 \neq Y_0$, (2) $X_{n_p} = Y_{n_q} = Z$, et (3) $(X_i = Y_j) \Rightarrow (i = n_p \text{ ou } j = n_q)$.

Nous notons $J(S) = \{Z \mid \exists p : X_0 \to^+ X_{n_p},\ q : Y_0 \to^+ Y_{n_q}$ tel que p et q convergent vers Z et $X_0, Y_0 \in S\}$.

Question 5. Définir, à partir de la fonction $J(\cdot)$, l'ensemble des sommets où il est nécessaire de placer une fonction $\varphi(\cdot)$. Nous supposerons que toutes les variables sont initialisées (avec donc une affectation à ces variables) dans $Entry$.

Question 6. Nous notons $DF(S) = \bigcup_{X \in S} DF(X)$ et définissons $DF^+(S) = \lim_{i \to +\infty} DF_i(S)$ où $DF_1(S) = DF(S)$ et $DF_{i+1}(S) = DF(S \cup DF_i(S))$.
Montrer que pour toute variable $x \in \mathcal{X}$, $DF^+(A(x))$ est égal à l'ensemble des positions où nous souhaitons positionner des fonctions $\varphi(\cdot)$, avec $A(x) = \{Z \mid$ il y a une affectation de x dans $Z\}$.

Question 7. En déduire un algorithme qui permet de définir les positions où ajouter les fonctions φ. Donner sa complexité.

Théorème de Hennessy-Milner

Un système de transition étiqueté dans l'alphabet A est un triplet (S, i, Δ) avec S un ensemble (d'états), $i \in S$ (état initial) et Δ un sous-ensemble de $S \times A \times S$ (transitions). On notera $s \xrightarrow{a} s'$ pour $(s, a, s') \in \Delta$.
Nous nous intéresserons à une logique pour ces systèmes appelée la logique d'Hennessy-Milner. Ses formules sont générées par la grammaire :

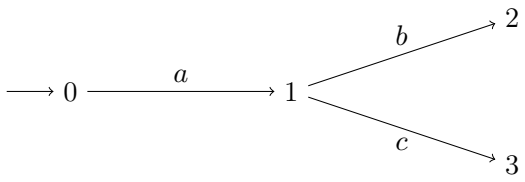
$$\varphi, \psi ::= \top \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle a \rangle \varphi$$

avec $a \in A$. Nous disons qu'un état s de (S, i, Δ) satisfait une formule φ et nous notons $s \vDash \varphi$, par récurrence sur φ :

- $s \vDash \top$, c'est-à-dire, tout état satifait la formule vraie,
- $s \vDash \neg\varphi$ si et seulement si $s \nvDash \varphi$,
- $s \vDash \varphi \wedge \psi$ si et seulement si $s \vDash \varphi$ et $s \vDash \psi$,
- $s \vDash \langle a \rangle \varphi$ si et seulement s'il existe s' tel que $s \xrightarrow{a} s'$ et $s' \vDash \varphi$.

Nous disons qu'un système (S, i, Δ) satisfait une formule φ si $i \vDash \varphi$.

Question 1. Donnez des exemples de formules satisfaites et non satisfaites par le système suivant :



La flèche entrante en 0 indique qu'il s'agit de l'état initial.

Solution : Quelques exemples de formules satisfaites :

- $\top$
- $\langle a \rangle \top$
- $\langle a \rangle \langle b \rangle \top$
- $\langle a \rangle \langle c \rangle \top$
- $\langle a \rangle (\langle b \rangle \top \wedge \langle c \rangle \top)$
- $\langle a \rangle \langle b \rangle \top \wedge \langle a \rangle \langle c \rangle \top$
- $\langle a \rangle \neg\langle a \rangle \top$

Quelques exemples de formules non satisfaites :

- $\neg\top$
- $\neg\langle a \rangle \top$
- $\langle b \rangle \top$
- $\langle a \rangle \langle a \rangle \top$
- $\langle a \rangle \neg\langle c \rangle \top$

Question 2. Soient $\varphi = \langle a \rangle (\langle b \rangle \top \wedge \langle c \rangle \top)$ et $\psi = \langle a \rangle \langle b \rangle \top \wedge \langle a \rangle \langle c \rangle \top$. Montrez que φ implique ψ mais pas inversement, c'est-à-dire, montrez 1) que tout système qui satisfait φ satisfait ψ, mais 2) qu'il existe un système qui satisfait ψ mais pas φ.

Solution : Commençons par 1). En déroulant la définition, il y a équivalence entre tous les énoncés suivants :

— un système (S, i, Δ) satisfait φ,

— $i \vDash \varphi$,

— il existe s_1 tel que $i \xrightarrow{a} s_1$ et $s \vDash \langle b \rangle \top \wedge \langle c \rangle \top$,

— il existe s_1, s_2 et s_3 tels que $i \xrightarrow{a} s_1$, $s_1 \xrightarrow{b} s_2$ et $s_1 \xrightarrow{c} s_3$.

Similairement, il y a équivalence entre :

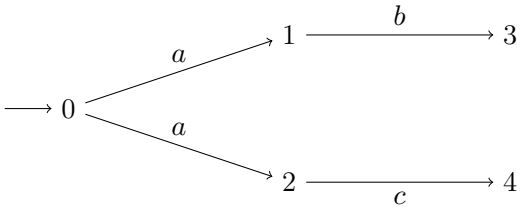
— un système (S, i, Δ) satisfait ψ,

— $i \vDash \psi$,

— $i \vDash \langle a \rangle \langle b \rangle \top$ et $i \vDash \langle a \rangle \langle c \rangle \top$,

— il existe t_1, t_2, t_3 et t_4 tels que $i \xrightarrow{a} t_1$, $i \xrightarrow{a} t_2$, $t_1 \xrightarrow{b} t_3$ et $t_2 \xrightarrow{c} t_4$.

En prenant $t_1 = t_2 = s_1$, $t_3 = s_2$ et $t_4 = s_3$, nous remarquons que si un système satisfait φ, il satisfait aussi ψ. Par contre, le système suivant :



satisfait ψ (prendre $t_i = i$), mais ne satisfait pas φ. En effet, nous avons deux choix : soit $s_1 = 1$, soit $s_1 = 2$. Dans le premier cas, il n'y a pas de c-transition et s_3 ne peut pas être défini. Dans le deuxième, il n'y a pas de b-transition et s_2 ne peut pas être défini.

Une bisimulation entre deux systèmes (S, i, Δ) et (S', i', Δ') est une relation $R \subseteq S \times S'$ entre leurs états qui satifait les propriétés suivantes :

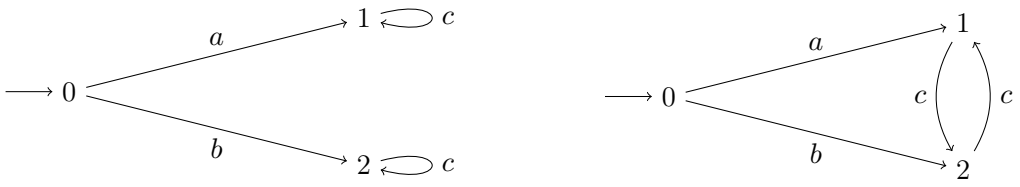
— $(i, i') \in R$,

— si $(s, s') \in R$ et si $s \xrightarrow{a} t$, alors il existe t' tel que $(t, t') \in R$ et $s' \xrightarrow{a} t'$,

— si $(s, s') \in R$ et si $s' \xrightarrow{a} t'$, alors il existe t tel que $(t, t') \in R$ et $s \xrightarrow{a} t$.

Quand une telle bisimulation existe, nous disons que les systèmes sont bisimilaires.

Question 3. Montrez que les deux systèmes suivants sont bisimilaires :



Solution : Une bisimulation est donnée par $R = \{(0,0), (1,1), (1,2), (2,1), (2,2)\}$.

Question 4. Montrez que la bisimilarité (le fait pour deux systèmes d'être bisimilaires) est une relation d'équivalence. Vous montrerez en détail le cas de la transitivité.

Solution : — reflexivité : la diagonale $\{(s,s) \mid s \in S\}$ est une bisimulation entre (S,i,Δ) et lui-même.

— symmétrie : si R est une bisimulation entre (S,i,Δ) et (S',i',Δ'), alors la relation converse $R^\dagger = \{(s',s) \mid (s,s') \in R\}$ est une bisimulation entre (S',i',Δ') et (S,i,Δ).

— transitivité : si R est une bisimulation entre (S,i,Δ) et (S',i',Δ'), et si Q est une bisimulation entre (S',i',Δ') et (S'',i'',Δ''), alors la relation composite

$$R;Q = \{(s,s'') \mid \exists\, s'.\, (s,s') \in R \wedge (s',s'') \in Q\}$$

est une bisimulation entre (S,i,Δ) et (S'',i'',Δ''). En effet :

— comme $(i,i') \in R$ et $(i',i'') \in Q$, alors $(i,i'') \in R;Q$.

— Suppose $(s,s'') \in R;Q$ et $s \xrightarrow{a} t$. Par définition de $R;Q$, il existe s' tel que $(s,s') \in R$ et $(s',s'') \in Q$. Comme R est une bisimulation, il existe t' tel que $s' \xrightarrow{a} t'$ et $(t,t') \in R$. Comme Q est une bisimulation, il existe t'' tel que $s'' \xrightarrow{a} t''$ et $(t',t'') \in Q$. Par définition de $R;Q$, $(t,t'') \in R;Q$.

— La dernière condition est similaire.

Question 5. Montrez que si deux systèmes sont bisimilaires, alors ils satisfont les mêmes formules.

Solution : Supposons donnée une bisimulation R entre (S,i,Δ) et (S',i',Δ'). Il faut ici faire un peu attention à la formulation de la récurrence à prouver. Nous allons prouver que pour toute formule φ, pour toute paire d'états $(s,s') \in R$, $s \vDash \varphi$ si et seulement si $s' \vDash \varphi$, par récurrence sur la formule φ.

— $\varphi \equiv \top$: évident.

— $\varphi \equiv \neg\psi$: on a les équivalences suivantes

  — $s \vDash \varphi$

  — $s \nvDash \psi$ (définition de $\vDash$)

  — $s' \nvDash \psi$ (hypothèse d'induction)

  — $s' \vDash \varphi$ (définition de $\vDash$)

— $\varphi \equiv \varphi_1 \wedge \varphi_2$: similaire.

— $\varphi \equiv \langle a \rangle \psi$: on a les *implications* suivantes

  — $s \vDash \varphi$

  — il existe t tel que $s \xrightarrow{a} t$ et $t \vDash \psi$ (definition de $\vDash$)

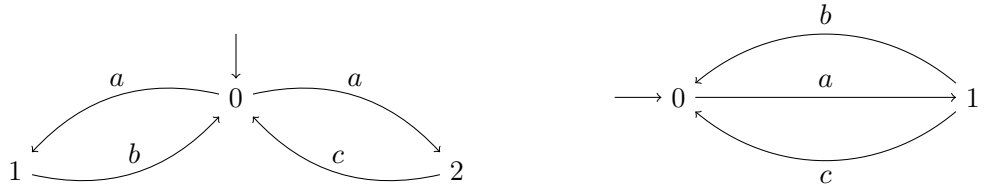
  — il existe t' tel que $s' \xrightarrow{a} t'$, $(t,t') \in R$ et $t \vDash \psi$ (R est une bisimulation)

  — il existe t' tel que $s' \xrightarrow{a} t'$ et $t' \vDash \psi$ (hypothèse de récurrence)

  — $s \vDash \varphi$ (définition de $\vDash$)

  L'implication dans l'autre sens est similaire. En utilisant cette propriété sur toute formule φ et sur (i,i') (comme $(i,i') \in R$ par définition d'une bisimulation), pour toute formule φ, $i \vDash \varphi$ si et seulement si $i' \vDash \varphi$. Donc (S,i,Δ) et (S',i',Δ') satisfont les mêmes formules.

Question 6. Utilisez ce résultat pour montrer que les deux systèmes suivants ne sont pas bisimilaires :



Solution : Par contraposé, pour montrer que ces systèmes ne sont pas bisimilaires, il suffit d'exhiber une formule satisfaite par l'un, mais pas par l'autre. Par ce que nous avons vu à la question 2., nous pouvons choisir $\langle a \rangle (\langle b \rangle \top \wedge \langle c \rangle \top)$ qui est satisfaite par le système de droite mais pas celui de gauche.

Question 7. Montrer que ces deux systèmes ne sont pas bisimilaires, mais satisfont les mêmes formules :

- $(S, 0, \Delta)$ avec :
  - $S = \{0\} \cup \{(i, n) \mid 1 \leq i \leq n\}$,
  - $0 \xrightarrow{a} (1, n)$ pour tout $n \geq 1$,
  - $(i, n) \xrightarrow{a} (i + 1, n)$ pour tout $1 \leq i < n$.
- $(S', 0, \Delta')$ avec :
  - $S' = \{0\} \cup \{(i, n) \mid 1 \leq i \leq n\} \cup \{(i, \infty) \mid 1 \leq i\}$,
  - $0 \xrightarrow{a} (1, n)$ pour tout $n \geq 1$,
  - $0 \xrightarrow{a} (1, \infty)$,
  - $(i, n) \xrightarrow{a} (i + 1, n)$ pour tout $1 \leq i < n$,
  - $(i, \infty) \xrightarrow{a} (i + 1, \infty)$ pour tout $1 \leq i$.

Solution : Supposons qu'ils soient bisimilaires, et que donc nous avons une bisimulation R entre eux. En particulier, $(0, 0) \in R$. Comme $0 \xrightarrow{a} (1, \infty)$ dans le second système, alors il doit exister n tel que $((1, n), (1, \infty)) \in R$. Par récurrence, nous pouvons alors déduire que $((n, n), (n, \infty)) \in R$. Comme $(n, \infty) \xrightarrow{a} (n+1, \infty)$ dans le second système mais qu'il n'y a pas de transition sortante de (n, n) dans le premier, nous arrivons à la conclusion que R ne peut pas être une bisimulation.

Pour la seconde partie, commençons par montrer par récurrence sur φ que pour tout i, si l'ensemble $\{n \mid (i, n) \vDash \varphi\}$ est infini alors $(i, \infty) \vDash \varphi$ et si l'ensemble $\{n \mid (i, n) \nvDash \varphi\}$ est infini alors $(i, \infty) \nvDash \varphi$

- $\varphi \equiv \top$: évident.
- $\varphi \equiv \neg\psi$: Si $\{n \mid (i, n) \vDash \varphi\}$ est infini alors par définition de $\vDash$, $\{n \mid (i, n) \nvDash \psi\}$ est infini. Par hypothèse de récurrence, cela implique que $(i, \infty) \nvDash \psi$ et que donc $(i, \infty) \vDash \varphi$. L'autre partie de l'énoncé est similaire.
- $\varphi \equiv \varphi_1 \wedge \varphi_2$: Si $\{n \mid (i, n) \vDash \varphi\}$ est infini, alors $\{n \mid (i, n) \vDash \varphi_j\}$ est infini pour $j \in \{1, 2\}$. Par hypothèse de récurrence, cela implique que $(i, \infty) \vDash \varphi_j$ pour $j \in \{1, 2\}$ et que $(i, \infty) \vDash \varphi_1 \wedge \varphi_2$. Si $\{n \mid (i, n) \nvDash \varphi\}$ est infini, alors par le principe des tiroirs, au moins l'un des $\{n \mid (i, n) \nvDash \varphi_j\}$ est infini. Par l'hypothèse de récurrence, cela implique que $(1, \infty)$ ne satisfait pas au moins l'un des φ_j. Cela signifie que $(i, \infty) \nvDash \varphi$.
- $\varphi \equiv \langle b \rangle \psi$ avec $b \neq a$: évident car aucun des (i, n) et des (i, ∞) ne peut satisfaire cette formule.
- $\varphi \equiv \langle a \rangle \psi$: Si $\{n \mid (i, n) \vDash \varphi\}$ est infini, alors $\{n \mid (i + 1, n) \vDash \psi\}$ est infini. Par hypothèse de récurrence, cela implique que $(i + 1, \infty) \vDash \psi$ et que donc $(i, \infty) \vDash \varphi$. L'autre partie de l'énoncé est similaire.

Nous utilisons ce résultat pour montrer que par récurrence sur la formule φ que $0 \vDash \varphi$ dans le premier système si et seulement si $0 \vDash \varphi$ dans le second. Le seul cas intéressant est quand $\varphi \equiv \langle a \rangle \psi$. Deux cas possibles :

- soit $(1, n) \vDash \psi$ pour un certain n, alors $0 \vDash \varphi$ dans les deux systèmes.
- soit $(1, n) \nvDash \psi$ pour tout n. Dans ce cas, par ce qui précède, $(1, \infty) \nvDash \psi$. Alors dans ce cas, $0 \nvDash \varphi$ dans les deux systèmes.

> Nous dirons qu'un système (S, i, Δ) est à image finie si pour tout état s et lettre a, l'ensemble $\{s' \mid s \xrightarrow{a} s'\}$ est fini.

Question 8. Montrez que si deux systèmes à image finie satisfont les mêmes formules, alors ils sont bisimilaires.

Solution : Nous montrons que la relation suivante :

$$R = \{(s, s') \mid \forall \varphi.\, s \vDash \varphi \Leftrightarrow s' \vDash \varphi\}$$

est une bisimulation.

— $(i, i') \in R$ par hypothèse que les systèmes satisfont les mêmes formules.
— Supposons $(s, s') \in R$ et $s \xrightarrow{a} t$. Cela signifie que $s \vDash \langle a \rangle \top$. Donc par hypothèse, $s' \vDash \langle a \rangle \top$, donc il existe au moins un t' tel que $s' \xrightarrow{a} t'$. Supposons par l'absurde qu'aucun des tels t' soit tel que $(t, t') \in R$. Cela signifie que pour chaque t', il existe une formule $\varphi_{t'}$ telle que $t \vDash \varphi_{t'}$ et $t' \nvDash \varphi_{t'}$. Nous considérons la formule suivante :

$$\varphi = \langle a \rangle \bigwedge_{t' | s' \xrightarrow{a} t'} \varphi_{t'}$$

qui est bien définie car les systèmes sont à image finie. Maintenant, nous observons que $s \vDash \varphi$ car t est tel que $s \xrightarrow{a} t$ et $t \vDash \bigwedge_{t' | s' \xrightarrow{a} t'} \varphi_{t'}$. Par contre, $s' \nvDash \varphi$ car pour tout t' tel que $s' \xrightarrow{a} t'$, $t' \nvDash \bigwedge_{t' | s' \xrightarrow{a} t'} \varphi_{t'}$, ce qui contredit l'hypothèse que $(s, s') \in R$. Nous concluons donc qu'il existe t' tel que $s' \xrightarrow{a} t'$ et $(t, t') \in R$.
— Le cas $(s, s') \in R$ et $s' \xrightarrow{a} t'$ est similaire.