Exercices de colles d'informatique

MPI/MPI*

Guillaume Chirache

Ceci est une sélection d'exercices que j'ai écrits pour des colles d'informatique en MPI/MPI* au lycée Paul-Valéry au cours de l'année 2025-2026.

N'hésitez pas à me contacter pour signaler des erreurs, demander des indications ou proposer des corrections.

Table des matières

Tournois

On appelle tournoi un graphe $T = (S, A)$ orienté simple tel que pour tout couple $(u, v) \in S^2$ avec $u \neq v$, un et un seul des arcs (u, v) et (v, u) appartient à A.

1) Combien un tournoi acyclique peut-il avoir de tris topologiques ?
2) Montrer qu'un tournoi possède un cycle si et seulement s'il possède un cycle de longueur 3.
3) Montrer que tout tournoi possède un chemin hamiltonien (un chemin passant par tous les sommets).

Dijkstra et arêtes négatives

On considère un graphe orienté $G = (S, A)$ et une fonction de poids $w : A \to \mathbb{Z}$. On fixe une source $s \in S$ et on suppose tous les sommets atteignables depuis s.

1) Donner une condition nécessaire et suffisante (le montrer) pour qu'il y ait un plus court chemin de s vers tout autre sommet t.
2) On suppose maintenant cette condition remplie. L'algorithme de Dijkstra calcule-t-il toujours les plus courts chemins ?
3) Est-il possible d'ajouter le minimum de w à toutes les arêtes pour se ramener au cas où les arêtes sont de poids positif ?
4) Proposer une variante de l'algorithme pour résoudre le problème et donner sa complexité en temps.

Théorème de Hall

On souhaite montrer le théorème suivant (Hall, 1935) : un graphe $G = (S, A)$ de bipartition X, Y admet un couplage M saturant X (i.e. $\forall x \in X, \exists y \in S, \{x, y\} \in M$) si et seulement si pour tout $U \subseteq X$, $|N(U)| \geqslant |U|$ (où $N(U)$ est l'union des voisinages des sommets de U).

1) L'une des implications est immédiate : la montrer.

2) Pour l'autre sens, supposons avoir un couplage maximum M qui ne recouvre pas $x \in X$. Soit U (resp. V) l'ensemble des sommets de X (resp. Y) reliés à x par un chemin alternant. On note de plus $M' \subseteq M$ l'ensemble des arêtes de M incidentes à un sommet de $U \cup V$.

   a) Montrer que M' est un couplage parfait entre $U \setminus \{x\}$ et V.

   b) Montrer que $N(U) \subseteq V$.

   c) Conclure que $|N(U)| < |U|$.

3) Application : montrer que tout graphe biparti k-régulier (tout sommet a k voisins) admet un couplage parfait.

Théorème de Myhill-Nerode

L'objectif de cet exercice est de caractériser précisément la minimalité d'un automate déterministe en démontrant le théorème de Myhill-Nerode (1958).

Soit Σ un alphabet. À un langage $L \subseteq \Sigma^*$, on associe la relation binaire $\equiv_L$ définie sur Σ^* par :

$$u \equiv_L v \text{ si et seulement si } \forall x \in \Sigma^*, ux \in L \iff vx \in L.$$

À titre d'exemples, soit $L_1 = \{w \in \{a, b\}^* \mid ab \text{ est un facteur de } w\}$ et $L_2 = \{a^n b^n \mid n \in \mathbb{N}\}$ sur l'alphabet $\Sigma = \{a, b\}$.

Dans cet exercice, tous les automates considérés sont des automates finis déterministes complets.

1) Montrer que $a \equiv_{L_1} aa$ mais que $a \not\equiv_{L_2} aa$.

2) Montrer que pour tout langage L, $\equiv_L$ est une relation d'équivalence.

3) Identifier les classes d'équivalence de $\equiv_{L_1}$ et $\equiv_{L_2}$.

4) Soit $n \in \mathbb{N}^*$ et $L \subseteq \Sigma^*$. Montrer qu'il existe un automate à n états reconnaissant L si et seulement si $\equiv_L$ possède au plus n classes d'équivalences.

5) Application : existe-t-il un automate qui reconnaît L_1 (resp. L_2) ? Si oui, le donner.

Un langage non régulier

Sur l'alphabet $\Sigma = \{0, 1\}$, on considère le langage

$$L = \{uu \mid u \in \Sigma^*\}.$$

Le but de l'exercice est de montrer par l'absurde que L n'est pas régulier. Supposons disposer d'un automate fini déterministe complet $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ tel que $\mathcal{L}(\mathcal{A}) = L$.

La fonction de transition δ est étendue en une fonction $\delta^* : Q \times \Sigma^* \to Q$ définie récursivement par $\delta^*(q, \varepsilon) = q$ pour tout $q \in Q$ et $\delta^*(q, w\sigma) = \delta(\delta^*(q, w), \sigma)$ pour $(q, w, \sigma) \in Q \times \Sigma^* \times \Sigma$. Ainsi, un mot $w \in \Sigma^*$ appartient à L si et seulement si $\delta^*(q_0, w) \in F$.

1) Justifier que $\delta^*(q_0, 0000) \neq \delta^*(q_0, 1100)$ et en déduire que $\delta^*(q_0, 00) \neq \delta^*(q_0, 11)$.

2) Montrer que pour tout $q \in Q$ et tous $x, y \in \Sigma^*$:

$$\delta^*(q, xy) = \delta^*(\delta^*(q, x), y).$$

Indication : on pourra fixer q et x et procéder par induction structurelle sur y.

3) Montrer que tous les états accessibles de $\mathcal{A}$ sont co-accessibles.

4) Soit x et y deux mots distincts et de même longueur sur l'alphabet Σ. Montrer que $\delta^*(q_0, x) \neq \delta^*(q_0, y)$.

5) En déduire une contradiction.

Des langages réguliers ?

Étant donné un mot $w = a_0 ... a_{n-1} \in \Sigma^*$, on appelle **miroir** de w le mot $\overline{w} = a_{n-1} ... a_0$.

Pour chacun des langages suivants, déterminer s'il est régulier ou non.

1) Le langage $\{w \in \{0,1\}^* \mid w = \overline{w}\}$ sur l'alphabet $\{0,1\}$ (les mots de ce langage sont appelés des **palindromes**).

2) Le langage $\{0^{3n} \mid n \in \mathbb{N}\}$ sur l'alphabet $\{0\}$.

3) Le langage $\{0^{2^n} \mid n \in \mathbb{N}\}$ sur l'alphabet $\{0\}$.

4) Le langage $\{0^p \mid p \text{ est un nombre premier}\}$ sur l'alphabet $\{0\}$.

5) Le langage $\{0^n \mid \text{l'écriture décimale de } \pi \text{ contient } n \text{ zéros consécutifs}\}$ sur l'alphabet $\{0\}$.

Complexité de la déterminisation d'un automate fini

Dans cet exercice, l'alphabet est $\Sigma = \{0,1\}$. Pour $n \in \mathbb{N}^*$, on définit le langage

$$L_n = \{a_0 ... a_{k-1} \in \Sigma^* \mid k \geqslant n \text{ et } a_{k-n} = 0\}.$$

1) Décrire par une phrase le langage L_n.

2) Pour $n \in \mathbb{N}^*$, donner un automate non déterministe $\mathcal{A}_n$ à $n+1$ états tel que $\mathcal{L}(\mathcal{A}_n) = L_n$.

3) Soit $n \in \mathbb{N}^*$ et $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ un automate déterministe tel que $\mathcal{L}(\mathcal{A}) = L_n$. On considère la fonction de transition étendue δ^* telle que définie en cours.

   a) Soit $w_1, w_2 \in \Sigma^n$ deux mots différents de longueur n. Montrer que $\delta^*(q_0, w_1) \neq \delta^*(q_0, w_2)$.

   Indication : on pourra chercher un suffixe $u \in \Sigma^*$ tel qu'exactement l'un des deux mots $w_1 u$ et $w_2 u$ soit dans L_n.

   b) En déduire que $|Q| \geqslant 2^n$.

4) En déduire que le problème de la déterminisation d'un automate fini n'admet pas d'algorithme polynomial.

Un langage sans boucles infinies

> Le but de cet exercice est de montrer qu'un langage dans lequel il n'est pas possible de faire des boucles infinies est nécessairement moins « expressif » que les langages de programmation usuels comme OCaml ou C, au sens où il ne peut pas résoudre certains problèmes de décision pourtant décidables.

On suppose disposer d'un langage de programmation, appelé Finito, qui ne permet que de résoudre des problèmes de décision et avec lequel il est impossible de faire des boucles infinies : les

programmes Finito prennent une chaîne de caractères en entrée et ils terminent toujours, en répondant soit $\top$ (« oui ») soit $\bot$ (« non »).

On suppose que le problème consistant à décider si une chaîne de caractères donnée est un programme Finito valide est décidable. Autrement dit, il existe une fonction OCaml `is_valid_finito_program`, de type `string -> bool`, qui renvoie `true` si la chaîne de caractères passée en argument est un programme Finito valide, et `false` sinon.

De plus, il est possible d'écrire en OCaml un interpréteur Finito, c'est-à-dire qu'il existe une fonction OCaml `eval_finito_program`, de type `string -> string -> bool`, telle que `eval_finito_program program input` vaut `true` si le programme Finito `program` répond $\top$ sur `input`, et `false` s'il répond $\bot$ (on n'impose rien dans le cas où `program` n'est pas un programme Finito valide : `eval_finito_program` peut par exemple échouer dans ce cas).

Montrer qu'il existe un problème de décision (dont les instances sont les chaînes de caractères) qui est décidable en OCaml (c'est-à-dire au sens du cours), mais qui n'est résolu par aucun programme Finito.

L'idée initiale vient d'une pâle (examen) du cours INF412 (fondements de l'informatique : logique, modèles, calculs) donné par Olivier Bournez à l'X. La question était formulée avec le formalisme des machines de Turing.

Casernes de pompiers

Vous êtes chargé(e) de revoir le plan des secours publics d'une région rurale en décidant dans quels villages une caserne de pompiers doit être installée. Votre budget est limité et vous pouvez installer au plus $n \geqslant 1$ casernes.

La région possède un ensemble fini V de villages, avec $|V| > n$. La distance entre les villages est modélisée par une application $d : V^2 \to \mathbb{N}$. Cette application est symétrique et, naturellement, $d(x, x) = 0$ pour tout $x \in V$. Comme le réseau routier de la région est à peu près cohérent, cette application vérifie de plus l'inégalité triangulaire :

$$\forall (x, y, z) \in V^3, d(x, z) \leqslant d(x, y) + d(y, z).$$

Étant donné un sous-ensemble $C \subseteq V$ de villages dotés d'une caserne, on note $r(C)$ la distance maximale d'un village à une caserne de pompiers :

$$r(C) = \max_{x \in V} \left(\min_{y \in C} d(x, y) \right).$$

Le « problème des casernes de pompiers » consiste à trouver un sous-ensemble $C \subseteq V$ de cardinal $|C| \leqslant n$ qui minimise $r(C)$.

1) Justifier qu'il existe une solution optimale avec $|C| = n$.

2) On propose de choisir l'ensemble C avec l'algorithme suivant :

> $C \leftarrow \{x\}$, où x est pris arbitrairement dans V
> **while** $|C| \leqslant n$ **do**
> $r \leftarrow r(C)$
> $C \leftarrow C \cup \{x\}$, où $x \in V \setminus C$ est tel que $r = \min_{y \in C} d(x, y)$
> **end**

   a) Explique le principe de l'algorithme et justifier qu'il est polynomial.

b) Comment évolue $r(C)$ au cours de l'algorithme ?

c) Soit C l'ensemble de villages trouvé en fin d'algorithme. Montrer qu'il existe $n+1$ villages à distance au moins $r(C)$ les uns des autres.

d) En déduire que pour tout ensemble $C' \subseteq V$ de n villages, $r(C) \leqslant 2 \times r(C')$.

Indication : on utilisera l'inégalité triangulaire.

e) Montrer alors que l'algorithme proposé est une approximation de facteur 2 du problème des casernes de pompiers.

La suite de l'exercice consiste à montrer qu'à moins que P = NP, il n'est pas possible de trouver une meilleure approximation. On introduit le problème de décision DominatingSet :

DominatingSet
Entrée : un graphe $G = (S, A)$ non orienté et un entier k
Question : existe-t-il un ensemble de sommets $D \subseteq S$ de taille $|D| \leqslant k$ tel que tout sommet du graphe est dans D ou voisin d'un sommet dans D ?

On appelle 3-SAT la restriction de SAT aux formules sous forme normale conjonctive où chaque clause possède au plus trois littéraux. On rappelle que 3-SAT est NP-complet.

3) a) Justifier que DominatingSet est dans NP.

b) Montrer que DominatingSet est NP-complet en construisant une réduction polynomiale de 3-SAT à DominatingSet.

Indication : on introduira trois sommets par variable (un pour chaque valeur de vérité et un de contrôle) et un sommet par clause.

4) Montrer que si P $\neq$ NP, alors pour tout $1 \leqslant \rho < 2$, le problème des casernes de pompiers n'admet aucune approximation polynomiale de facteur ρ .

5) On ne suppose plus que l'application d vérifie l'inégalité triangulaire. Montrer alors que si P $\neq$ NP, aucune approximation polynomiale de facteur constant du problème des casernes de pompiers n'est possible.

Somme d'un sous-ensemble

On considère le problème d'optimisation suivant, appelé SmallestSubsetSum : étant donné un ensemble fini non vide $E \subseteq \mathbb{Z}$, on cherche un sous-ensemble non vide $S \subseteq E$ tel que $|\sum_{n \in S} n|$ est minimal.

1) Résoudre le problème pour $E = \{7, 8, -6, 15, -1\}$ et $E = \{-10, 6, -4, 3\}$.

On introduit la variante décisionnelle ZeroSubsetSum qui consiste, étant donné $E \subseteq \mathbb{Z}$ fini, à déterminer s'il existe un sous-ensemble non vide $S \subseteq E$ de somme nulle.

2) Montrer que ZeroSubsetSum est dans la classe NP.

On souhaite maintenant montrer que ZeroSubsetSum est en fait NP-complet. Pour cela, on va prouver que 3-SAT $\leqslant_{\mathrm{P}}$ ZeroSubsetSum. Étant donné une instance φ de 3-SAT à n variables $x_1, ..., x_n$ et m clauses $C_1, ..., C_m$, on va construire un ensemble fini $E \subseteq \mathbb{N}^*$ d'entiers tel que

$$\varphi \text{ est satisfiable} \iff \exists S \subseteq E, \sum_{n \in S} n = \underbrace{11...11}_{n \text{ chiffres}} \underbrace{44...44}_{m \text{ chiffres}}.$$

3) a) On admet la faisabilité d'une telle construction en temps polynomial. Montrer alors que ZeroSubsetSum est NP-difficile.

   b) On souhaite maintenant réaliser la construction admise. Dans un premier temps, proposer une construction de E telle que φ est satisfiable si et seulement s'il existe un sous-ensemble S de E dont la somme est de la forme

$$\sum_{n\in S} n = \underbrace{11...11}_{n \text{ chiffres}} c_1...c_m,$$

   où c_j est un chiffre non nul pour tout $j \in [\![1, m]\!]$.

   Indication : l'ensemble E pourra être constitué de deux entiers a_i et b_i par variable x_i.

   c) Modifier la construction précédente en ajoutant des éléments à E afin que la condition souhaitée initialement soit vérifiée. Conclure.

4) En déduire que si $\mathsf{P} \neq \mathsf{NP}$, alors aucune approximation polynomiale à facteur constant de SmallestSubsetSum n'est possible.

5) Proposer néanmoins un algorithme par séparation et évaluation qui résout le problème.

Hypothèse du temps exponentiel

Pour $k \in \mathbb{N}^*$, on appelle k-SAT la restriction de SAT aux instances sous forme normale conjonctive dans lesquelles les clauses sont **distinctes** et contiennent k littéraux. Pour ces problèmes, lorsqu'on parle de complexité en $\mathcal{O}(f(n))$ pour $f : \mathbb{N} \to \mathbb{N}$, il est entendu que n est le nombre de variables de l'instance considérée. On rappelle que 3-SAT est NP-complet.

1) a) Montrer que pour tout $k \geqslant 3$, le problème k-SAT est NP-complet.

   b) Pour $k \geqslant 3$ fixé, proposer un algorithme naïf pour résoudre k-SAT et donner sa complexité en la taille de l'entrée sous la forme $\mathcal{O}(f(n))$.

2) Soit $k \geqslant 3$. Proposer un algorithme par retour sur trace résolvant k-SAT dont la complexité temporelle est en $\mathcal{O}(T(n))$, pour une fonction $T : \mathbb{N} \to \mathbb{N}$ vérifiant

$$T(n) \leqslant \mathcal{O}(1) + \sum_{i=1}^{k} T(n-i).$$

   Ce résultat permet en fait de montrer que $T(n) = \mathcal{O}\big(2^{\delta n}\big)$ pour un certain $\delta < 1$, ce que l'on admettra.

Pour $k \geqslant 3$, on pose :

$$s_k = \inf\big\{\delta \in \mathbb{R}_+ \;\big|\; k\text{-SAT admet une solution en } \mathcal{O}\big(2^{\delta n}\big)\big\}.$$

3) Montrer que la suite $(s_k)_{k\geqslant 3}$ converge.

Même si les exposants de la question 2 peuvent être améliorés, on pense qu'il n'est pas possible de trouver d'algorithme fondamentalement meilleur. La conjecture suivante a été formulée en 1999 :

Hypothèse du temps exponentiel (1999)
L'**hypothèse du temps exponentiel** (ETH) conjecture que $s_3 > 0$. On parle d'hypothèse **forte** du temps exponentiel (SETH) lorsqu'on conjecture de plus que $\lim_{k\to+\infty} s_k = 1$.

La suite de l'exercice consiste à étudier des conséquences de cette hypothèse.

4) Montrer que si la conjecture ETH est vraie, alors $\mathsf{P} \neq \mathsf{NP}$.

On considère maintenant le problème TwoSetsOrtho :

TwoSetsOrtho

- **Entrée** : $(d, N) \in \mathbb{N}^2$ et deux N-uplets
$$X = (x_1, ...x_N), Y = (y_1, ..., y_N) \in \left(\{-1, 0, 1\}^d\right)^N$$
de vecteurs de $\{-1, 0, 1\}^d$
- **Question** : existe-il un couple $(i, j) \in [\![1, N]\!]^2$ tel que x_i et y_j sont orthogonaux pour le produit scalaire canonique sur $\mathbb{R}^d$?

5) a) Proposer un algorithme naïf pour TwoSetsOrtho. Quelle est sa complexité ?

b) Montrer que si la conjecture SETH est vraie, alors pour tout $\varepsilon > 0$ et tout $c > 0$, il n'existe pas d'algorithme résolvant TwoSetsOrtho en $\mathcal{O}\left(N^{2-\varepsilon} d^c\right)$.

L'idée initiale vient d'un DM du cours INF550 (algorithmique avancée) donné par Gilles Schaeffer à l'X.