Feuille d'exercices n°17 - Révisions

Notions abordées

- recherche de motif : algorithme de Boyer-Moore, automate des motifs
- compression : algorithme LZW
- programmation dynamique (plusieurs exemples)
- classes de complexité P et NP, réduction
- diviser pour régnér et calcul de complexité

Exercice 1 : Algorithme de Boyer-Moore (simplifié)

Q. 1 Quel problème résout l'algorithme de Boyer-Moore ?

Q. 2 Donner le pseudo-code d'un algorithme naïf qui résout ce problème. Préciser sa complexité.

Q. 3 Rappeler l'idée de l'algorithme de Boyer-Moore.

Q. 4 On suppose que la fenêtre de texte $t[\![i, i+|x|[\![$ ne coïncide pas avec le motif x du fait d'un mauvais caractère à l'indice j, *i.e.* parce que $t_{i+j} \neq x_j$. En se concentrant sur ce caractère, quel est l'indice i' qui définit la prochaine fenêtre du texte qui est susceptible de coïncider avec le motif ?

Q. 5 Appliquer l'algorithme découlant de la remarque précédente sur le texte `tatatututoto` et pour le motif `tuto` ?

Q. 6 Identifier quelles données peuvent être précalculées à partir du motif x pour rendre plus efficace le calcul du décalage proposé à la **Q.4**. Donner alors le pseudo-code de ce pré-calcul et préciser sa complexité.

Q. 7 On suppose que la fenêtre de texte $t[\![i, i+|x|[\![$ ne coïncide pas avec le motif x du fait d'un mauvais caractère à l'indice j, *i.e.* parce que $t_{i+j} \neq x_j$. On suppose que ce caractère est le premier en partant de la droite où la fenêtre du texte et le motif ne coïncident plus, ainsi $t[\![j+1, i+|x|[\![= x[\![j+1, |x|[\![$. En se concentrant sur ce suffixe, quel est l'indice i' qui définit la prochaine fenêtre du texte qui est susceptible de coïncider avec le motif ?

Q. 8 Appliquer l'algorithme découlant de la remarque précédente ♣ sur le texte `babbababbbaab` et pour le motif `babaab` ?

Q. 9 Identifier quelles données peuvent être précalculées à partir du motif x pour rendre plus efficace le calcul du décalage proposé à la **Q.7**. Donner alors le pseudo-code de ce pré-calcul et préciser sa complexité.

Q. 10 Appliquer l'algorithme de Boyer-Moore combinant les deux remarques précédentes sur le texte `taratatauturlututu` et pour le motif `turlututu` ?

Q. 11 Proposer le pseudo-code complet de l'algorithme de Boyer-Moore. Quelle est la complexité de cet algorithme en fonction de la taille du motif et de celle du texte ?

♣. et de la remarque précédente uniquement

Exercice 2 : Recherche de motif avec un automate

On fixe Σ un alphabet, et un mot $x = x_1x_2\dots x_m \in \Sigma^+$. Pour un mot $t \in \Sigma^\star$ de longeur $n \in \mathbb{N}$ et $j \in [\![1, n]\!]$, on dira qu'une **occurence** de x se termine à l'indice j dans t si et seulement si $x_1x_2\dots x_m = t_{j-m+1}\dots t_{j-1}t_j$. Dans ce cadre là, les mots x et t ont des rôles très différents, c'est pourquoi on appellera x le **motif**, c'est-à-dire le mot dont on cherche des occurences, et on appellera **texte** un mot $y \in \Sigma^\star$ dans lequel on cherche des occurences du motif. Le but de cet exercice est de construire, à partir du motif seulement, un automate déterministe qui permette de calculer efficacement toutes les occurences du motif dans un texte.

Pour tout mot $w \in \Sigma^\star$, on note $\mathscr{P}(w)$, resp. $\mathcal{S}(w)$, l'ensemble des préfixes, resp. suffixes, de w.

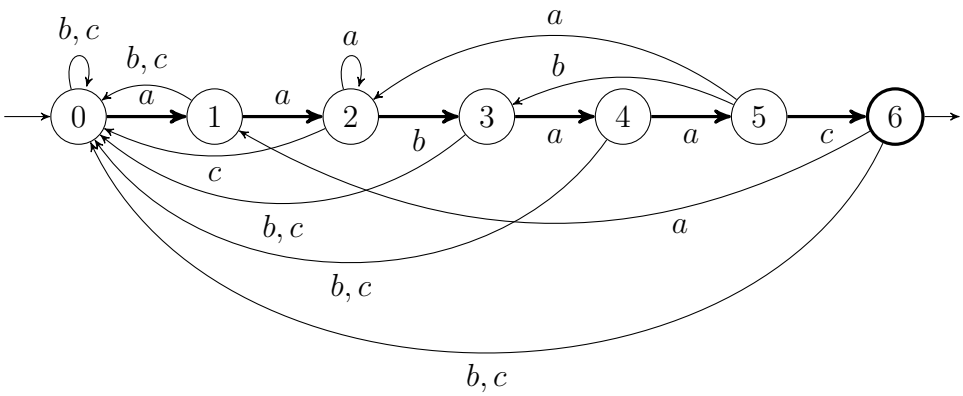
$$\forall w \in \Sigma^\star,\ \mathscr{P}(w) = \{p \in \Sigma^\star \mid \exists \overline{w} \in \Sigma^\star, w = p\cdot\overline{w}\} \quad \text{et} \quad \mathcal{S}(w) = \{s \in \Sigma^\star \mid \exists \underline{w} \in \Sigma^\star, w = \underline{w}\cdot s\}$$

Q. 1 Soit $u \in \Sigma^\star$. Pour $s \in \mathcal{S}(u)$ que vaut $\mathcal{S}(s)$ en fonction de $\mathcal{S}(u)$? *Une brève justification suffit.*

Q. 2 Soit $u \in \Sigma^\star$. $\mathscr{P}(u) \cap \mathcal{S}(u)$ peut-il être vide ?

Q. 3 Soit $u \in \Sigma^\star$. Donner un majorant de $\{|w| \mid w \in \mathscr{P}(u)\}$ et montrer qu'il est atteint (c'est donc un maximum).

L'automate du motif x est un automate déterministe à $m+1$ états qui servent à représenter la progression dans le motif. Par exemple l'automate du motif $aabaac$ est le suivant.



Afin de définir formellement un tel automate pour un motif x quelconque, on introduit la fonction f qui à une longueur $i \in [\![0, m]\!]$ et une lettre $a \in \Sigma$ associe la longueur du plus long suffixe de $x_1\dots x_i\cdot a$ qui soit aussi un préfixe de x.

$$f = \begin{pmatrix} [\![0,m]\!]\times\Sigma & \to & [\![0,m]\!] \\ (i,a) & \mapsto & \max\limits_{w \in \mathscr{P}(x)\cap\mathcal{S}(x_1\dots x_i a)} |w| \end{pmatrix}$$

Q. 4 Pour $i \in [\![0, m-1]\!]$, que vaut $f(i, x_{i+1})$?

On définit alors $\mathcal{A}_x = (\Sigma, Q = [\![0,m]\!], I = \{0\}, F = \{m\}, \delta)$ où $\delta = \{(i, a, f(i,a)) \mid i \in [\![0,m]\!], a \in \Sigma\}$

Q. 5 Pour le motif $x = babba$ sur l'alphabet $\Sigma = \{a, b\}$ (on a donc $m = 5$), dresser la table des $f(i,z)$ pour $(i,z) \in [\![0,m]\!]\times\Sigma$ puis dessiner $\mathcal{A}_x$.

Q. 6 Donner le pseudo-code d'un algorithme naïf construisant l'automate $\mathcal{A}_x$ à partir du motif x. On attend un algorithme naïf en $\mathcal{O}(|\Sigma|m^3)$.

Puisque l'automate $\mathcal{A}_x$ est complet et déterministe, pout tout état $q \in Q$ et toute lettre $a \in \Sigma$, il existe un unique état $q' \in Q$ tel que $(q, a, q') \in \delta$. On note cet état $\delta^1(q, a)$, ce qui revient à noter δ^1 la fonction de transition de $\mathcal{A}_x$. De plus on note $\delta^\star$ la fonction de transition étendue de $\mathcal{A}_x$.

$$\delta^\star = \left(\begin{array}{lcl} Q \times \Sigma^\star & \longrightarrow & Q \\ (q, \varepsilon) & \mapsto & q \\ (q, z \cdot u) & \mapsto & \delta^\star\Big(\delta^1(q, z), u\Big) \\ \text{où } z \in \Sigma & & \end{array} \right) = \left(\begin{array}{lcl} Q \times \Sigma^\star & \longrightarrow & Q \\ (q, \varepsilon) & \mapsto & q \\ (q, u \cdot z) & \mapsto & \delta^1\Big(\delta^\star(q, u), z\Big) \\ \text{où } z \in \Sigma & & \end{array} \right)$$

On a affirmé en introduction que les états de l'automate $\mathcal{A}_x$ servent à représenter la progression dans le motif x. Cette affirmation vague peut maintenant être reformulée de manière plus précise. Pour tout mot $u \in \Sigma^\star$, sur l'automate $\mathcal{A}_x$, la lecture de u mène à un état qui indique la longueur du plus long préfixe de x qui est un suffixe de u. Avec les notations introduites plus haut, cet invariant peut être reformulé comme suit.

$$\forall u \in \Sigma^\star,\ \delta^\star(0, u) = \max_{w \in \mathcal{P}(x) \cap \mathcal{S}(u)} |w|$$

Q. 7 Démontrer la propriété précédente par récurrence sur la taille de u.

Q. 8 Expliquer comment utiliser l'automate $\mathcal{A}_x$ pour repérer toutes les occurences de x dans un texte t de taille n ? Quelle serait la complexité temporelle de cette procédure (sans compter le pré-traitement consistant à créer $\mathcal{A}_x$) ? Quelle serait la complexité spatiale d'une telle méthode ?

Exercice 3 : Algorithme de Lempel-Ziv-Welsh (LZW)

Q. 1 Rappeler quel est le principe de la compression selon LZW. Que peut-on dire des longueurs des motifs ajoutés au dictionnaire au cours de la compression ? Que peut-on dire des motifs ajoutés au dictionnaire lors de la décompression par rapport à ceux ajoutés lors de la compression ?

Q. 2 Exécuter à la main la compression de LZW sur le mot `gogogo`. *On pourra prendre comme dictionnaire de base celui qui numérote les lettres de l'alphabet minsucule de 0 à 25.*

Q. 3 Exécuter à la main la décompression de LZW sur la chaîne obtenue à la question précédente, sans utiliser le dictionnaire construit précédemment.

On cherche maintenant à implémenter la compression et la décompression selon LZW en OCaml. On utilise pour implémenter les dictionnaires (et les ensembles si besoin) le module `Hashtbl`. On fournit dans le fichier `compagnon_lzw.ml` plusieurs fonctions utiles, notamment de manipulation élémentaires des dictionnaires mis en jeu lors de la compression ou la décompression.

Q. 4 Coder en Ocaml la compression et la décompression selon LZW. Tester ces procédures sur des textes, et calculer les taux de compression observés.

Q. 5 Citer un autre algorithme de compression (au programmme). En quoi ces deux méthodes de compression diffèrent-elles ?

Exercice 4 : Plus long sous-mot/facteur commun

Q. 1 Rappeler les définitions de **facteur** d'un mot et **sous-mot** d'un mot. Donner un exemple illustrant la différence entre les deux.

Q. 2 Définir le problème du plus long sous-mot commun.

Q. 3 Définir le problème du plus long facteur commun.

Q. 4 Expliquer comment résoudre le problème du plus long sous-mot commun par programmation dynamique.

Q. 5 Expliquer comment résoudre le problème du plus long facteur commun par programmation dynamique.

Éléments pour établir un algorithme de programmation dynamique

→ identifier une famille de "sous-problèmes"[a] qu'il est intéressant de résoudre ;
→ identifier quels paramètres permettent de caractériser ces sous-problèmes relativement à l'instance de départ ;
→ définir une quantité paramétrée qui représente la valeur optimale du sous-problème défini par ces paramètres ;
→ justifier que cette famille de quantités permet de résoudre le problème initial ;
→ dire comment calculer les cas de base, *i.e.* les quantités pour les paramètres minimaux ;
→ dire comment calculer l'une de ces quantités à partir des valeurs pour des paramètres plus petits ;
→ si on opte pour de l'impératif, fournir le pseudo-code qui explique quelle taille de tableau alouer pour stocker ces quantités, dans quel ordre on va remplir le tableau, comment on va ensuite récupérer la valeur optimale ;
→ dans le cas où seule la valeur optimale nous intéresse, il est souvent possible de modifier l'algorithme précédent en vue de réduire sa consommation mémoire ;
→ dans le cas où une solution optimale doit aussi être fournie, on complète l'algorithme précédent en ajoutant le plus souvent l'entegistrement d'information pertinente lors du remplissage du tableau, et en ajoutant une phase de reconstruction de la solution qui remonte dans le tableau à partir d'une case de valeur optimale.

a. On appelle ici, comme c'est l'usage, famille de sous-problèmes une famille d'instances, instances du problème initial ou parfois d'une variante de celui-ci.

Exercice 5 : L'étoile d'un langage dans P est aussi dans P

Le but de cet exercice est de montrer que pour L un langage de P, L^* est aussi dans P.

Q. 1 Rappeler ce que signifie $L \in \mathsf{P}$ et ce que signifie $L^* \in \mathsf{P}$.

Q. 2 Combien y a-t-il de découpages en facteurs non-vides d'un mot de taille $n > 0$? En déduire la complexité de l'algorithme naïf qui teste l'appartenance d'un mot à L^* à partir d'un test d'appartenance à L en envisageant un à un tous les découpages possibles ?

Q. 3 Combien y a-t-il de facteurs non-vides d'un mot de taille n ? En déduire quel schéma algorithmique il pourrait être judicieux de mettre en place pour tester l'appartenance à L^* ?

Q. 4 À quelle condidtion un mot de taille 0 est-il dans L^* ?
À quelle condidtion un mot de taille 1 est-il dans L^* ?

Q. 5 Soit $w \in \Sigma^\star$ de taille $n \in \mathbb{N}^*$. Pour $\{(i,j) \in [\![1,n]\!]^2 \mid i \leqslant j\}$, on définit $A_{i,j} = \begin{cases} \mathsf{V} & \text{si } w_i \dots w_j \in L^* \\ \mathsf{F} & \text{sinon} \end{cases}$

Donner une caractérisation récursive de A.

Q. 6 Déduire des questions précédentes un algorithme qui teste en temps polynomial l'appartenance à L^*, en supposant que le test d'appartenance à L est en temps polynomial.

Q. 7 Conclure.

Exercice 6 : Énumération des mots reconnus par un automate

Dans cet exercice on se propose d'énumérer les mots reconnus par un automate (non nécessairement déterministe). Un tel ensemble pouvant bien sûr être infini, on cherche à fournir un algorithme qui, étant donné un automate $\mathcal{A}$ sur un alphabet Σ et un entier $n \in \mathbb{N}$ produit l'ensemble des mots de Σ^n reconnus par $\mathcal{A}$, ensemble qu'on notera $\mathcal{L}_n(A)$ dans la suite de cet exercice :

$$\mathcal{L}_n(\mathcal{A}) \overset{\text{déf}}{=} \mathcal{L}(\mathcal{A}) \cap \Sigma^n$$

Étant donné un automate $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ et un état $q \in Q$, on note $\mathcal{A}_q$ l'automate $(\Sigma, Q, \{q\}, F, \delta)$. La seule différence entre $\mathcal{A}_q$ et $\mathcal{A}$ se situe au niveau des états initiaux : ceux de $\mathcal{A}$ sont remplacés par le seul état q dans $\mathcal{A}_q$.

Q. 1 Étant donné un automate $\mathcal{A} = (\Sigma, Q, I, F, \delta)$, exprimer $\mathcal{L}_0(\mathcal{A}_q)$ en fonction de $q \in Q$?

Q. 2 Étant donné un automate $\mathcal{A} = (\Sigma, Q, I, F, \delta)$, donner et prouver, pour tout $(n, q) \in \mathbb{N}^\star \times Q$, une relation entre $\mathcal{L}_n(\mathcal{A}_q)$ et les $\mathcal{L}_p(\mathcal{A}_{q'})$ pour $p < n$ et $q' \in Q$.

Q. 3 Donner, pour tout $n \in \mathbb{N}$, une relation entre $\mathcal{L}_n(\mathcal{A})$ et les $\mathcal{L}_n(\mathcal{A}_q)$ pour $q \in Q$.

Les questions précédentes devraient donner une idée d'algorithme récursif permettant d'énumérer les mots d'une longueur donnée reconnus par un automate donné. Toutefois, avant de se jeter dans l'implémentation, on s'interroge sur la complexité de l'algorithme envisagé.

Q. 4 Donner, pour chaque $n \in \mathbb{N}$ un automate $\mathcal{A}_n$ à n états et $\Theta(n)$ transitions♣ tel que l'arbre des appels récursifs pour le calcul de $\mathcal{L}_n(\mathcal{A}_n)$ est de taille exponentielle en n, tandis que le cardinal de $\mathcal{L}_n(\mathcal{A}_n)$ est en $\mathcal{O}(n)$.

Q. 5 Donner, en fonction du nombre d'états M de l'automate $\mathcal{A}$ et de l'entier n une majoration du nombre d'ensembles $\mathcal{L}_p(\mathcal{A}_q)$ qui entrent en jeu dans le calcul récursif de $\mathcal{L}_n(\mathcal{A})$.

Q. 6 Proposer un schéma algorithmique permettant la réconciliation des observations des deux dernières questions.

Q. 7 Proposer une implémentation en OCaml de cet algorithme. On pourra supposer que l'ensemble Q des états de l'automate est de la forme $[\![0, n-1]\!]$, avec $n \in \mathbb{N}$. On accordera une attention particulière au fait que :

- il n'est pas nécessaire d'avoir un espace mémoire en $\Theta(Mn)$, un $\Theta(n)$ suffit.
- si l'on itère sur chaque état, et que pendant chacune de ces itérations, on itère sur ses prédécesseurs, il est peut être envisageable de ne parcourir qu'une fois la liste des transitions.

♣. on entend par là que pour tout $n \in \mathbb{N}$, $\mathcal{A}_n$ a exactement t_n transitions, et que $(t_n)_{n\in\mathbb{N}} \in \Theta(n)$.

Exercice 7 : Suites récurrentes de complexité

Dans cet exercice il vous est demandé de donner, pour chacune des suites suivantes :
- un Θ décrivant son comportement asymptotique ;
- un programme OCaml **dont cette suite est la complexité**, pour des constantes a et b au choix[♡] ;
- un schéma de l'arbre des appels récursifs d'un appel au programme proposé.

a) $\begin{cases} u_0 &= 1 \\ u_n &= u_{n-1} + a \quad a \in \mathbb{R}^{+\star} \end{cases}$

b) $\begin{cases} u_0 &= 1 \\ u_n &= u_{n-1} + an \quad a \in \mathbb{R}^{+\star} \end{cases}$

c) $\begin{cases} u_0 &= 1 \\ u_n &= au_{n-1} + b \quad a \in [2, +\infty[, b \in \mathbb{R}^{+\star} \end{cases}$

d) $\begin{cases} u_0 &= 1 \\ u_n &= u_{n/2} + b \quad b \in \mathbb{R}^{+\star} \end{cases}$

e) $\begin{cases} u_0 &= 1 \\ u_n &= u_{n/2} + bn \quad b \in \mathbb{R}^{+\star} \end{cases}$

f) $\begin{cases} u_0 &= 1 \\ u_1 &= 1 \\ u_n &= u_{\lceil \frac{n}{2} \rceil} + u_{\lfloor \frac{n}{2} \rfloor} + b \quad b \in \mathbb{R}^{+\star} \end{cases}$

g) $\begin{cases} u_0 &= 1 \\ u_n &= au_{n/2} + b \quad a \in [2, +\infty[, b \in \mathbb{R}^{+\star} \end{cases}$

Exercice 8 : Quelques équivalents classiques

Q. 1 Donner le comportement asymptotique des suites suivantes[♠] au moyen de la notation Θ.

a) $\lfloor \log_2(n) \rfloor$

b) $2^{\lfloor \log_2(n) \rfloor}$

Q. 2 Donner le comportement asymptotique des sommes suivantes[♢] au moyen de la notation Θ.

a) $\sum_{k=1}^{n} k$
b) $\sum_{k=1}^{n} k^2$
c) $\sum_{k=0}^{n} 2^k$
d) $\sum_{k=0}^{n} \frac{1}{2^k}$
e) $\sum_{k=0}^{n} 2^k$
f) $\sum_{k=1}^{n} \frac{1}{k}$
g) $\sum_{k=1}^{n} \log k$
h) $\sum_{k=0}^{n} k \log k$

Exercice 9 : Éléments majoritaires

On dit qu'un élément x est majoritaire dans un multi-ensemble E de cardinal n lorsque le nombre d'occurrences de x dans E est strictement supérieur à $\frac{n}{2}$. On suppose donc un multi-ensemble E représenté par un tableau de ses éléments et on cherche si oui ou non E admet un élément majoritaire (s'il existe il est unique). Dans le cas d'une réponse affirmative on renverra l'élément en question. On s'intéresse dans cet exercice au nombre de tests d'égalité (ou inégalité évidemment) effectués entre des éléments du multi-ensemble E. On suppose définie une fonction nbOccur prenant en arguments un tableau T, un élément x, et deux indices i et j du tableau et retournant le nombre d'occurrences de x dans le tableau T entre les indices i et j (au sens large). On suppose de plus que l'appel nbOccur(T, x, i, j) fait $j - i + 1$ comparaisons.[♣]

♡. Insistons : il s'agit écrire un programme OCaml ayant cette suite comme complexité et non un programme OCaml calculant cette suite.

♠. *i.e.* implicitement indexées par n

♢. *i.e.* des suites, implicitement indexées par n, des sommes jusqu'au rang n

♣. Les suppositions des deux phrases précédentes sont là pour vous faire gagner du temps, il **faut** que vous sachiez implémenter un tel algorithme nbOccur.

Q. 1 Rappeler la définition d'un algorithme naïf majoritaire(T) retournant s'il existe un élément majoritaire du multi-ensemble E représenté par le tableau T. On précisera la complexité pire cas de l'algorithme.

Q. 2 Supposant connu l'élément majoritaire du sous-tableau $T[0..n/2-1]$ ♣ et l'élément majoritaire du sous-tableau $T[n/2..n-1]$ où T est un tableau de taille n, donner l'élément majoritaire de T en faisant de l'ordre de n comparaisons.

Q. 3 Donner alors un algorithme du paradigme diviser pour régner permettant la résolution du problème de la recherche d'un élément majoritaire.

Q. 4 Donner la complexité pire cas dans le cas où la taille du tableau donné en entrée est une puissance de 2.

Exercice 10 : Sous-tableau de somme maximale

Étant donné un tableau d'entiers, le problème du sous-tableau maximum consiste à trouver un ensemble d'indices consécutifs (*i.e.* un intervalle d'indices) non vide qui maximise la somme des éléments du tableau. Par exemple, le sous-tableau maximum du tableau `[-3, 4, 5, -1, 2, 3, -6, 4]` est le tableau`[4, 5, -1, 2, 3]` de valeur 13. Étant donné un tableau T on note donc stm(T) la valeur du sous-tableau maximum.

Q. 1 Formaliser ce problème comme un problème d'optimisation.

Q. 2 Dans un premier temps, on considère l'algorithme de résolution ci-dessous. Quelle est la complexité de cet algorithme ?

Algorithme 1 : Sous tableau maximum

Entrée : A un tableau de taille n
Sortie : stm(A)

```
1 A_max ← max A[1], ..., A[n] ;
2 pour i = 1 à n faire
3     pour j = i à n faire
4         sum ← 0 ;
5         pour k = i à j faire
6             sum ← sum + A[j] ;
7             si sum > A_max alors
8                 A_max ← sum
9 retourner A_max
```

Q. 3 Proposer une variante en $\Theta(n^2)$ de cet algorithme.

Dans la suite de l'exercice, on s'intéresse à des algorithmes de type diviser pour régner. Pour un tableau A indicé par $[1..n]$ avec $n > 1$, on note $A_1 = A[1..m]$ et $A_2 = A[m+1..n]$ les deux sous-tableaux définis par $m = \left\lfloor \frac{n+1}{2} \right\rfloor$. On note alors stm$_1(A)$ = stm(A_1) et stm$_2(A)$ = stm(A_2). On introduit aussi stm$_3(A)$ la valeur maximum d'un sous-tableau de A qui couvre à la fois les indices m et $m+1$.

Q. 4 Comment calculer stm$_3(A)$ efficacement ? Quelle est la complexité en fonction de n la taille de A ?

♣. comprendre : on sait s'il existe un élément majoritaire et s'il existe, on sait qui il est

Q. 5 Connaissant $\mathsf{stm}_1(A)$, $\mathsf{stm}_2(A)$ et $\mathsf{stm}_3(A)$, quelle est la valeur de $\mathsf{stm}(A)$? Dans le cadre d'une méthode diviser pour régner qui calcule $\mathsf{stm}(A)$ en calculant récursivement et $\mathsf{stm}_1(A)$ et $\mathsf{stm}_2(A)$ en quoi consisterait l'opération de fusion ? Quelle serait la complexité de cette opération ?

Q. 6 Soit T_n le nombre d'opérations élémentaires (additions et comparaisons d'éléments du tableau) que réalise cette méthode pour un tableau de taille n. Donner l'équation de récurrence satisfaite par T_n. En déduire la complexité de la méthode.

On s'intéresse maintenant à une autre approche diviser pour régner afin de réduire encore la complexité. Pour ce faire, on définit, pour un tableau A indicé par $[1..n]$ les valeurs suivantes :

- $\mathsf{pref}(A) = \max\{\sum_{k=1}^{i} A[k] | i \in [1..n]\}$
- $\mathsf{suff}(A) = \max\{\sum_{k=i}^{n} A[k] | i \in [1..n]\}$
- $\mathsf{tota}(A) = \sum_{k=1}^{n} A[k]$

Autrement dit, $\mathsf{pref}(A)$ (resp. $\mathsf{suff}(A)$) est la valeur maximum d'un sous-tableau préfixe (resp. suffixe) de A, et $\mathsf{tota}(A)$ est la somme des valeurs de A.

Q. 7 Expliquer comment calculer ces valeurs pour un tableau A ?

Q. 8 Quelle est la complexité de l'algorithme diviser pour régner associé ? Justifier.

Exercice 11 : Multiplication d'entiers selon Karatsuba

Dans cet exercice on s'intéresse au problème de la multiplication de grands entiers machines. On suppose les entiers représentés par la séquence indicée des bits de leur décomposition en base 2. Par exemple, l'entier 14 est représenté par la séquence $(1, 1, 1, 0)$. On appellera alors taille d'un entier la longueur d'une telle séquence. On suppose de telles séquences stockées dans des tableaux permettant l'indexation en temps constant. On remarque que les multiplications et divisions entières par des puissances de 2 correspondent à des décalages de bits. En effet si $(u_{n-1}, u_{n-2}, \ldots, u_0)$ est la représentation d'un entier u et $p \in \mathbb{N}$ alors $u/2^p$ est représenté par $(u_{n-p-1}, u_{n-p-2}, \ldots, u_p)$ et $u \times 2^p$ par $(u_{n-1}, u_{n-2}, \ldots, u_0, \underbrace{0, 0, \ldots 0}_{p})$. Dans tout l'exercice on s'intéresse uniquement au problème de la multiplication de deux entiers de même taille et on mesure la complexité au regard du nombre de multiplications de nombres à 1 bit. Ainsi l'opération 1×0 est considérée comme une opération atomique de coût 1 alors que le coût de calcul de 1110110×101110 dépend de l'algorithme choisi pour faire le calcul de $\times$.

Q. 1 Expliquer en quoi l'hypothèse d'une taille commune des deux opérandes n'est pas trop simplificatrice.

Q. 2 Rappeler l'algorithme de multiplications naïf d'entiers (celui appris en primaire). Quelle est sa complexité ?

On remarque que si $u = u_a + 2^p u_b$ avec $u_a < 2^p$ et $v = v_a + 2^p v_b$ avec $v_a < 2^p$ alors $u \times v = u_a \times v_a + 2^p(u_b \times v_a + v_b \times u_a) + 2^{2p} u_b \times v_b$.

Q. 3 Proposer un algorithme de multiplication d'entiers utilisant le paradigme diviser-pour-régner et utilisant la remarque précédente. Faire une rapide étude de complexité dans le cas où les entrées sont de taille 2^p. Cet algorithme diviser-pour-régner présente-t-il un avantage par rapport à l'algorithme naïf.

Q. 4 En s'intéressant à la valeur de $u_a v_a + u_b v_b - (u_a - u_b)(v_a - v_b)$ proposer un algorithme faisant 3 appels récursifs sur des entrées dont la taille est divisée par 2.

Q. 5 Faire une étude rapide de complexité dans le cas où les entiers en arguments sont de taille $n = 2^p$.

Q. 6 Donner la complexité pire cas de l'algorithme de la question 4 au moyen d'un Θ. Au vu de la question précédente, on pourra intuiter puis démontrer par récurrence un encadrement sur le nombre de multiplications élémentaires effectuées par l'algorithme de la question 4 sur deux entrées de taille respectivement n et m.

Q. 7 Dans les questions précédentes nous avons ignorer le coût des additions et soustractions sur les grands entiers. On ne suppose plus que c'est le cas : une addition, soustraction sur deux entiers de tailles n et m coûte dorénavant $\max(n, m)$. Faire une rapide étude de complexité de l'algorithme de la question 4 (on se contentera des entiers de la forme 2^p). Conclure.

Exercice 12 : Coloration et couverture de graphe

Dans cet exercice on considère deux problèmes de décision associés à des problèmes de minimisation sur les graphes non orientés. Le problème de coloration consiste à colorier les sommets d'un graphe avec le moins de couleur possible de sorte que deux sommets reliés par une arête ne soient pas de la même couleur. Le problème de couverture par des cliques consiste à répartir les sommets d'un graphe en le moins de groupe possible de sorte que chacun de ces groupes forme une clique♣.

Q. 1 Formaliser les problèmes de décision ColorationMin et Couv.Clique respectivement associés au problème de coloration et au problème de couverture par des cliques.

Q. 2 Montrer que Couv.Clique $\preccurlyeq_P$ ColorationMin et ColorationMin $\preccurlyeq_P$ Couv.Clique.

Exercice 13 : Partition Bis

On rappelle que le problème SubsetSum ci dessous est NP complet.

$$\begin{cases} \textbf{Entrée :} \text{ Un entier } n \in \mathbb{N}, \text{ une suite finie } (w_i)_{i \in [\![1,n]\!]} \in \mathbb{N}^n, \text{ un entier } W \\ \textbf{Sortie } \textbf{:} \text{ Existe-t-il } I \subseteq [\![1,n]\!] \text{ tel que } \sum_{i \in I} w_i = W \text{ ?} \end{cases}$$

On définit le problème PartitionBis comme prenant en entrées : un entier n, une famille de n nombres entiers, deux entiers W_1 et W_2 et demandant s'il est possible de partitionner cette famille d'entiers de sorte que la somme des éléments de la première partition soit inférieur à W_1, et la somme des éléments de la seconde partition soit inférieur à W_2.

Q. 1 Formaliser le problème PartitionBis.

Q. 2 Montrer que PartitionBis $\in$ NP.

Q. 3 Montrer que PartitionBis est NP difficile.

♣. On rappelle qu'une **clique** est un ensemble de sommets qui induit un sous-graphe complet.

Exercice 14 : Problèmes NP-difficiles de dominants

Dans cet exercice on s'intéresse à deux problèmes de décision sur des graphes non orientés. Afin de définir ces problèmes, on donne d'abord quelques définitions.

Définition 0.1

Soit $G = (S, A)$ un graphe non orienté.

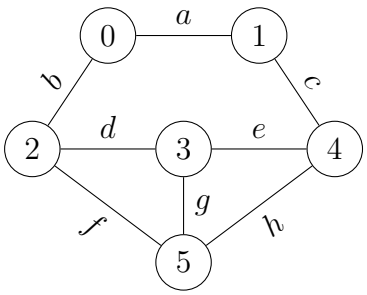
- *Deux sommets u et v sont **voisins** dans G ssi $\{u, v\} \in A$.*
- *Deux arêtes e et f sont **adjacentes** dans G ssi $e \cap f \neq \emptyset$.*
- *Soit $R \subseteq S$ un sous-ensemble de sommets.*
  *R est un **ensemble de sommets dominant** de G ssi $\forall u \in S \setminus R, \exists v \in R, \{u, v\} \in A$, $e \cap R \neq \emptyset$, autrement dit ssi chaque sommet de G qui n'est pas dans R est voisin d'un sommet de R.*
- *Soit $D \subseteq A$ un sous-ensemble d'arêtes.*
  *D est un **ensemble d'arêtes dominant** de G ssi $\forall e \in A \setminus D$, $\exists d \in D, e \cap d \neq \emptyset$, autrement dit ssi chaque arête de G qui n'est pas dans D est adjacente à une arête de D.*

Définition 0.2

*Soit $G = (S, A)$ un graphe non orienté. Le **graphe adjoint** de G, noté $L(G)$ [♡], est le graphe non orienté qui représente la relation d'adjacence des arêtes de G.*

$$L(G) \stackrel{\text{déf}}{=} (A, B) \text{ où } B = \{\{e, f\} \mid e \in A, f \in A, e \cap f \neq \emptyset\}$$

Q. 1 Représenter $L(G)$ pour le graphe G ci-dessous.



On remarque que, peu importe le graphe non orienté $G = (S, A)$, S est toujours un ensemble de sommets dominant, et A est toujours un ensemble d'arêtes dominant. On considère alors les deux problèmes de décision suivants.

DOM.SOMMETS
- **Entrée :** Un graphe non orienté $G = (S, A)$, un seuil $K \in \mathbb{N}$.
- **Sortie :** G admet-il un ensemble de sommets dominant R tel que $|R| \leqslant K$?

DOM.ARÊTES
- **Entrée :** Un graphe non orienté $G = (S, A)$, un seuil $K \in \mathbb{N}$.
- **Sortie :** G admet-il un ensemble d'arêtes dominant D tel que $|D| \leqslant K$?

Q. 2 Justifier que le problème DOM.SOMMETS est dans la classe NP.

Q. 3 Proposer une relation de réduction polynomiale entre ces deux problèmes et la démontrer.

Q. 4 Que peut-on en déduire quant à la difficulté de ces problèmes de décision ?

♡. Ce graphe est appelé **line graph** de G en anglais.

Exercice 15 : Problèmes Dom.Sommets et Couv.Ens

1. Introduction des problèmes

Étant donné une famille $\mathcal{F}$ de sous-ensembles d'un ensemble X, on appelle **couverture** de X une sous-famille de $\mathcal{F}$ dont l'union est X. Le problème de la couverture par ensembles minimum consiste à trouver une couverture de cardinal minimum. Le problème de décision associé est le suivant.

$$\text{Couv.Ens} \begin{cases} \textbf{Entrée :} \text{ Un ensemble fini } X \neq \emptyset, n \in \mathbb{N}^*, (X_i)_{i\in[\![1,n]\!]} \in \mathcal{P}(X)^n, \text{ un seuil } K \in \mathbb{N}. \\ \textbf{Sortie :} \text{ Existe-t-il } J \subseteq [\![1,n]\!] \text{ tel que } X = \bigcup_{j\in J} X_j \text{ et } |J| \leqslant K \text{ ?} \end{cases}$$

Q. 1 Soit $(X, n, (X_i)_{i\in[\![1,n]\!]}, K)$ une instance de Couv.Ens. Donner une condition suffisante pour que X admette une couverture (de cardinal quelconque) et justifier que l'on peut tester cette condition en temps polynomial.

Étant donné un graphe non orienté $G = (S, A)$, on appelle **ensemble de sommets dominant** de G un sous-ensemble $R \subseteq S$ tel que chaque sommet de G qui n'est pas dans R est voisin d'un sommet de R, *i.e.* tel que $\forall u \in S \setminus R, \exists v \in R, \{u, v\} \in A,\ e \cap R \neq \emptyset$. On remarque que S est toujours un ensemble de sommets dominant. Le problème du dominant minimum consiste alors à trouver l'ensemble de sommets de plus petit cardinal qui est un dominant. Le problème de décision associé est le suivant.

$$\text{Dom.Sommets} \begin{cases} \textbf{Entrée :} \text{ Un graphe non orienté } G = (S, A) \text{ avec } S \neq \emptyset, \text{ un seuil } K \in \mathbb{N}. \\ \textbf{Sortie :} \text{ } G \text{ admet-il un ensemble de sommets dominant } R \text{ tel que } |R| \leqslant K \text{ ?} \end{cases}$$

Q. 2 Donner une instance négative et une instance positive de Dom.Sommets.

2. Réduction de Dom.Sommets à Couv.Ens

Q. 3 Montrer que Dom.Sommets se réduit en temps polynomial à Couv.Ens.

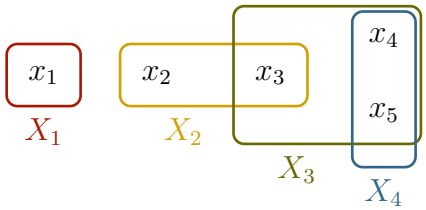
3. Réduction de Couv.Ens à Dom.Sommets

Soit $(X, n, (X_i)_{i\in[\![1,n]\!]}, K)$ une instance de Couv.Ens. On suppose que X admet au moins une couverture. De plus, quitte à renommer les éléments de X, on suppose que $X \cap [\![1, n]\!] = \emptyset$. On pose alors $I = [\![1, n]\!]$, $S = X \sqcup I$, $A = A_1 \sqcup A_2$ où

$$A_1 = \{\{i, j\} \mid (i, j) \in I^2, i \neq j\} \text{ et } A_2 = \{\{i, x\} \mid i \in I, x \in X, x \in X_i\}.$$

On considère enfin le graphe $G = (S, A)$, ainsi (G, K) est une instance de Dom.Sommets.

Q. 4 Dessiner le graphe G pour l'instance représentée ci-dessous où $X = \{x_1, x_2, x_3, x_4, x_5\}$, $n = 4$, $X_1 = \{x_1\}$, $X_2 = \{x_2, x_3\}$, $X_3 = \{x_3, x_4, x_5\}$ et $X_4 = \{x_4, x_5\}$.



Q. 5 Supposons qu'il existe $J \subseteq [\![1, n]\!]$ une couverture de X. Montrer qu'il existe un dominant de G de même cardinal.

Q. 6 Supposons que $R \subseteq S$ est un dominant de cardinal minimum pour G. Posons $J = R \cap I$.

$a)$ Montrer que J n'est pas nécessairement une couverture pour X.
On pourra s'appuyer sur l'exemple ci-dessus.

$b)$ Montrer que si $R \subseteq I$, alors J (et donc R) est une couverture pour X.

$c)$ Montrer que si $|R \cap X| > 0$, alors il existe $R' \subseteq S$ un autre dominant minimum pour G tel que $|R' \cap X| < |R \cap X|$.

Q. 7 Montrer que si R est un dominant de G cardinal minimum, alors il existe une couverture de X de cardinal $|R|$.

Q. 8 Montrer finalement que Couv.Ens se réduit en temps polynomial à Dom.Sommets.