MITRO210 : Automates et données structurées

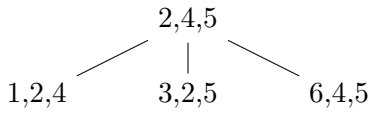
Feuille d'exercices 7 – Corrigé

Antoine Amarilli

1 Largeur arborescente

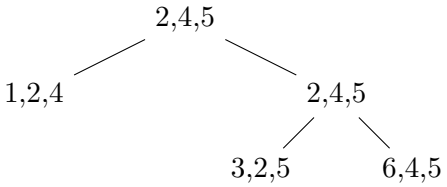
Question 0. Seules les forêts sont de largeur arborescente ≤ 1, donc ces graphes sont de largeur arborescente au moins 2.

Question 1. Pour G_2 on prend la décomposition arborescente suivante :



Pour G_3 on prend :

Question 2. On prend simplement :



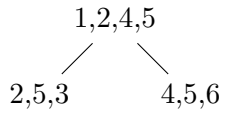
Question 3. Il suffit de remarquer qu'une décomposition arborescente de G est en particulier une décomposition arborescente de G' : le fait que G' ait moins d'arêtes implique qu'on impose moins de conditions sur une décomposition arborescente de G'.

Ainsi, si G est de largeur arborescente k, une décomposition arborescente de G est également une décomposition arborescente de G' qui est toujours de largeur arborescente k, donc qui témoigne du fait que G' est de largeur arborescente au plus k.

Question 4. Le graphe ainsi obtenu admet une 4-clique K_4 comme sous-graphe, en effet si l'on considère les sommets 1, 2, 4 et 5 on voit qu'il existe dans G'_1 une arête entre n'importe quel choix de deux sommets distincts dans cet ensemble.

On sait que K_4 est de largeur arborescente 3 d'après un résultat vu en cours, donc par la question précédente G'_1 l'est aussi.

Question 5. On prend simplement :



2 Propriétés élémentaires de la largeur arborescente

Question 0. La largeur arborescente de H est $\max(k, k')$. En effet, étant donné des décompositions arborescentes T et T' respectivement de G et G' et de largeur arborescente k et k' on obtient une décomposition arborescente de H simplement en reliant T et T' de façon quelconque : il est clair que ceci satisfait les conditions. La décomposition arborescente obtenue a largeur $\max(k, k')$.

Question 1. Prenons une décomposition arborescente de largeur k de G. On en construit une de G' en ajoutant le nouveau sommet x à l'image par β de tous les nœuds. On vérifie que les conditions sont respectées :

- Occurrence des sommets : c'est clair par occurrence des sommets de T plus le fait que x apparaît partout
- Occurrence des arêtes : c'est clair par occurrence des arêtes de T pour les anciennes arêtes, et occurrence des sommets de T plus occurrence de x partout pour les nouvelles arêtes
- Connexité : c'est clair pour les anciens sommets par connexité dans T, et c'est clair pour le nouveau sommet vu qu'il apparaît partout donc son sous-arbre d'occurrence est l'arbre entier.

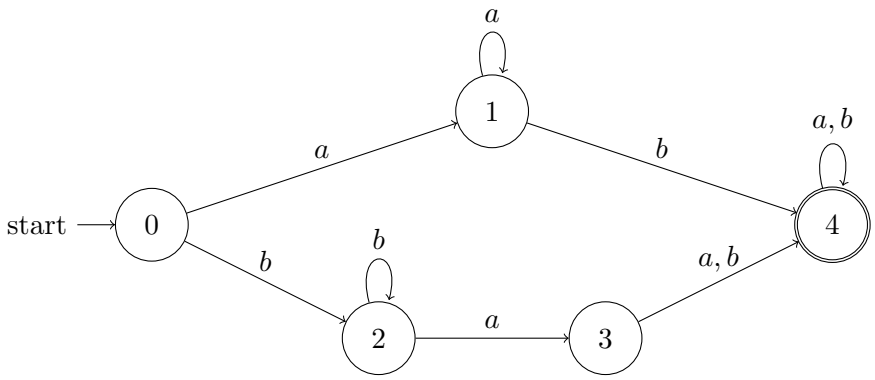
Du reste la nouvelle décomposition est manifestement de largeur $k + 1$ comme demandé.

3 Quelques révisions

Question 0. On obtient :

a	t	a	m	a	t	a	t	a	m	a	m	a
0	0	1	0	1	2	3	2	3	4	5	0	1

Question 1. On construit l'automate minimal pour ce langage :



On construit les éléments du monoïde de transition. Les mots ϵ et a et b donnent de nouveaux éléments, on obtient :

	0	1	2	3	4
ϵ	0	1	2	3	4
a	1	1	3	4	4
b	2	4	2	4	4

On considère les mots de longueur 2. Pour aa, puis pour ab, puis pour ba, on obtient de nouveaux éléments. La table contient alors :

	0	1	2	3	4
ϵ	0	1	2	3	4
a	1	1	3	4	4
b	2	4	2	4	4
aa	1	1	4	4	4
ab	4	4	4	4	4
ba	3	4	3	4	4

(Noter que, sans surprise, ab est un zéro.)

Pour bb, on remarque qu'on tombe sur le même élément que b. Ainsi on a la règle : $bb \to b$.

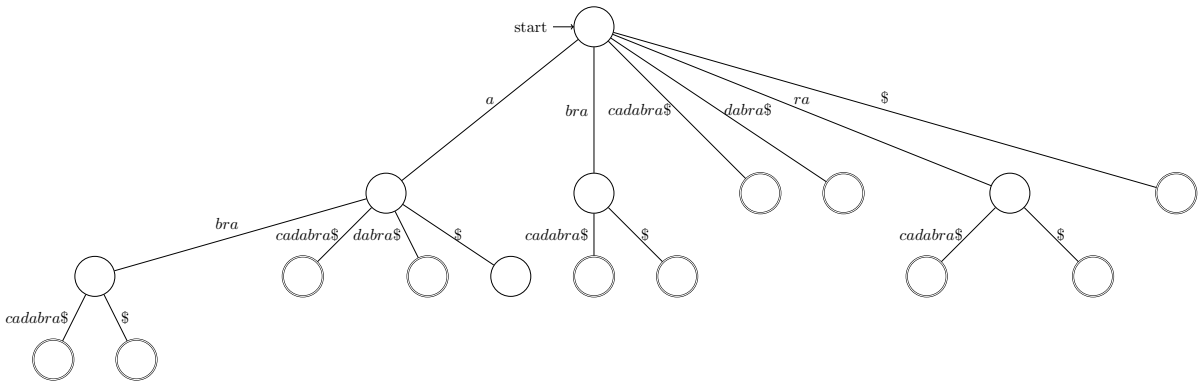
Passons aux mots de longueur 3 :

— Pour aaa, on obtient le même élément que aa, ainsi $aaa \to aa$.
— Pour aab, on obtient le même élément que ab (zéro), ainsi $aab \to ab$.
— Pour aba, là encore on obtient $aba \to ab$.
— Pour abb on l'ignore car bb est membre droit d'une règle.
— Pour baa on obtient le même élément que ab (c'est-à-dire qu'il est également envoyé sur le zéro), donc $baa \to ab$
— Pour bab on obtient $bab \to ab$
— Pour bba on l'ignore car bb est membre droit d'une règle.
— De même on ignore bbb.

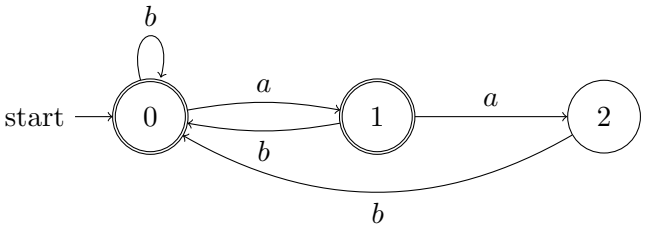
On n'a rien ajouté dans la table pour la longueur 3, donc on s'arrête.

Le monoïde est apériodique puisque le langage est sans étoile, en effet $\bar{\emptyset}(baa + ab)\bar{\emptyset}$ est une expression sans étoile pour le langage.

Question 2. On obtient :



Question 3. On pose l'automate minimal qui accepte les mots de cette forme :



On obtient le nombre de mots acceptés de longueur i en stockant, par programmation dynamique, le nombre de mots de longueur i menant de q_0 à q pour chaque q. On considère donc des vecteurs à 3 composantes.

Le cas de base pour $i = 0$ est le vecteur $\vec{v_0} = (1, 0, 0)$.

Pour la longueur $i + 1$, le nombre de mots de longueur i menant de q_0 à chaque état s'exprime comme un vecteur $\vec{v_{i+1}}$. Pour avoir ce nombre, on regarde le vecteur précédent $\vec{v_i}$.

Pour chaque lettre $x \in \{a, b\}$, pour chaque état q on ajoute $\vec{v_i}[q]$ à $\vec{v_{i+1}}[\delta(q, a)]$.

On a donc :

$$\begin{bmatrix} \vec{v_{i+1}}[0] \\ \vec{v_{i+1}}[1] \\ \vec{v_{i+1}}[2] \end{bmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{bmatrix} \vec{v_i}[0] \\ \vec{v_i}[1] \\ \vec{v_i}[2] \end{bmatrix}$$

Donc en posant M la matrice, on a que la réponse est la somme des composantes 1 et 2 de $M^{42}\vec{v_0}$. Autrement dit, la réponse est $\vec{v}M^{42}\vec{u}$ où $\vec{v} = (1, 1, 0)$ et $\vec{u} = (1, 0, 0)$.

4 3-coloriage de coût minimal

Question 0. Il y a des graphes qui ne sont pas 3-coloriables, par exemple K_4. Ainsi le coût minimal d'un 3-coloriage légal de K_4 n'est pas défini, quels que soient les poids. On convient alors que le coût vaut ∞.

Question 1. On énumère simplement tous les coloriages, pour chaque coloriage vérifie s'il est légal, si oui on calcule son coût, et on prend le minimum (initialisé à ∞). La complexité est $O((n+m) \times 3^n)$ où n est le nombre de sommets du graphe et m est le nombre d'arêtes.

Question 2. Pour représenter les contraintes de la forme $\lambda(u) = x$ pour un sommet u et une couleur x, il suffit de définir $\kappa(u, y) = 1$ pour tous les $y \neq x$, et de mettre $\kappa(v, z) = 0$ partout ailleurs. Le problème du coloriage de coût minimum nous renvoie le coût minimum d'un coloriage. Il est clair qu'il y a un solution à 3CC sur le graphe étiqueté d'entrée si et seulement s'il y a une solution au problème de 3-coloriage de coût minimal qui ait coût 0 (et qui respecte alors toutes les contraintes de λ). Ainsi, il suffit de vérifier si la réponse au problème du coloriage de coût minimal est 0 ou > 0. Au demeurant la réduction est en temps linéaire.

Question 3. Comme on choisit la solution sur chaque composante connexe indépendamment, le coloriage valide de coût minimal sur un graphe non connexe est clairement donné par des coloriages valides de coût minimal de chaque composante connexe, et le coût est alors la somme du coût des solutions sur chaque composante. Ainsi, il suffit de savoir résoudre le problème sur des graphes connexes.

Question 4. On procède par programmation dynamique. Pour chaque préfixe de longueur $1 \leq i \leq n$ et pour chaque couleur $x \in \Sigma$, on souhaite déterminer le coût minimal $C[i][x]$ d'un coloriage légal dont la dernière lettre soit x.

Le cas de base pour $i = 1$ est que $C[i][x]$ est donné par $\kappa(u_1, x)$.

Le cas d'induction est comme suit. Pour une position $i+1$, pour chaque couleur $x \in \Sigma$, pour avoir un coloriage légal où le $(i+1)$-ème caractère est x, on doit payer $\kappa(i+1, x)$, et on doit ensuite payer le coût minimal d'un coloriage des i premières positions dont la dernière lettre soit $y \neq x$. Ainsi on définit :

$$C[i+1][x] := \kappa(i+1, x) + \min_{y \neq x} C[i][y]$$

À la fin, comme la dernière couleur importe peu, on prend $\min_{x \in \Sigma} C[n][x]$.

Question 5. On enracine l'arbre en une feuille arbitraire : on obtient ainsi un arbre enraciné où chaque nœud interne a au plus deux enfants. Pour simplifier les notations, on peut ajouter aux nœuds internes n'ayant qu'un enfant des nœuds frais qui peuvent être gratuitement coloriés en n'importe quelle couleur. Ainsi, on peut supposer qu'on a affaire à un arbre binaire enraciné où chaque nœud a au plus deux enfants.

On calcule récursivement, pour chaque nœud n de l'arbre, pour chaque couleur $x \in \Sigma$, le coût minimal d'un coloriage légal du sous-arbre enraciné en n qui donne la couleur x à n.

Pour les feuilles n, on pose $C[n][x] := \kappa(n, x)$ comme pour les mots.

Pour les nœuds internes n ayant deux enfants n_1 et n_2, pour chaque couleur $x \in \Sigma$, on paie $\kappa(n, x)$ pour la couleur de ce nœud, et on a le choix de mettre des couleurs $y_1, y_2 \in \Sigma$ à n_1

et n_2 à condition que $x \neq y_1$ et $x \neq y_2$. On paie alors la somme du coût du coloriage pour le sous-arbre enraciné en n_1 avec une couleur y_1 à la racine dont on connaît inductivement le minimum $T[n_1][y_1]$, et de même pour n_2 et y_2. Ainsi :

$$C[n][x] = \kappa(n, x) + \min_{\substack{y_1, y_2 \in \Sigma \\ x \neq y_1, x \neq y_2}} (C[n_1][y_1] + C[n_2][y_2])$$

Remarquer qu'on peut aussi calculer le min sur y_1 et sur y_2 indépendamment.

À la fin, pour r la racine, on retourne $\min_{x \in \Sigma} C[r][x]$.

Question 6. Comme d'habitude, après avoir enraciné l'arbre, on rajoute des enfants virtuels si nécessaire pour que chaque nœud interne ait au moins 2 enfants. Pour les nœuds n ayant 3 enfants ou plus, on remplace n par une chaîne droite de nœuds dont on note qu'ils doivent tous être égaux (plus exactement qu'ils doivent tous porter la couleur du nœud le plus bas, et qu'ils prennent cette couleur gratuitement), en mettant les enfants de n comme enfants gauches de cette chaîne et enfants gauche et droite du dernier nœud.

Le cas de base pour $C[n][x]$ est inchangé, ainsi que le cas d'induction dans le cas où le nœud n'est pas un nœud qui doit porter la même couleur que son enfant droit.

Pour le cas d'induction d'un nœud n ayant pour enfants n_1 et n_2 et où n et n_2 doivent porter la même couleur parce qu'ils font partie d'une même chaîne, pour chaque $x \in \Sigma$ on sait que pour mettre la couleur x à ce nœud il faut une solution à droite avec cette même couleur, et on ne paie rien, et il faut une solution à droite ou la racine est d'une autre couleur. Ainsi :

$$C[n][x] := C[n_2][x] + \min_{y \neq x} C[n_1][y]$$

La fin de la preuve est la même.

Question 7. On va calculer récursivement sur la décomposition arborescente T, pour chaque nœud n, pour chaque choix de couleur pour les nœuds de $\beta(n)$ c'est-à-dire fonction c de $\beta(n)$ dans Σ, le coût minimal d'un coloriage de l'ensemble V_n des nœuds apparaissant dans le sous-arbre T_n de T enraciné en n qui soit légal pour les arêtes apparaissant dans T_n (c'est-à-dire les arêtes e pour lesquelles il existe un descendant n' de n telles que $e \subseteq \beta(n')$).

Le cas de base est celui d'une feuille n ; dans ce cas, pour chaque coloriage c de $\beta(n)$ qui assure que deux sommets adjacents de $\beta(n)$ ne reçoivent pas la même couleur, on définit :

$$T[n][c] := \sum_{v \in \beta(n)} \kappa(n, c(n))$$

On convient que $T[n][c] := \infty$ si c n'est pas un coloriage légal de $\beta(n)$.

Le cas d'induction est celui d'un nœud interne n ayant pour enfants n_1 et n_2. Dans ce cas, pour chaque coloriage c de $\beta(n)$ qui assure que deux sommets adjacents de $\beta(n)$ ne reçoivent pas la même couleur, il faut choisir un coloriage des nœuds de $V_{n_1} \setminus V_n$ et des nœuds de $V_{n_2} \setminus V_n$. On le fait en choisissant l'état des nœuds de $\beta(n_1) \setminus \beta(n)$, et de $\beta(n_2) \setminus \beta(n)$ (en prenant le minimum sur ces options), puis en utilisant les tableaux $T[n_1]$ et $T[n_2]$ pour connaître le coût minimaux de tels coloriages. Ceci dit, il y a une subtilité : T_{n_1} et T_{n_2} ne sont pas forcément disjoints, et ainsi les coloriages de $T[n_1]$ et $T[n_2]$ peuvent compter en double le coût certains nœuds. Mais on

sait par la propriété de connexité que $V_{n_1} \cap V_{n_2} \subseteq \beta(n_1) \cap \beta(n_2) \subseteq \beta(n)$. Ainsi, on peut ajuster le coût dans $T[n]$ en déduisant le coût des sommets qui ont déjà été comptés en double.

Formellement, définissons la notation $\kappa(S, c')$ pour S un ensemble de sommets et $c' : S \to \Sigma$ pour signifier $\sum_{v \in S} \kappa(v, c'(v))$. Notons $B_1 := \beta(n_1) \cap \beta(n)$ et $B_2 := \beta(n_2) \cap \beta(n)$. On pose alors :

$$T[n][c] := \kappa(\beta(n), c) + \min_{\substack{c_1 : \beta(n_1) \to \Sigma \\ \forall x \in B_1, c_1(x) = c(x)}} (T[n_1][c_1] - \kappa(B_1, c_1)) + \min_{\substack{c_2 : \beta(n_2) \to \Sigma \\ \forall x \in B_2, c_2(x) = c(x)}} (T[n_2][c_2] - \kappa(B_2, c_2))$$

On convient encore que $T[n][c] := \infty$ si le coloriage c de $\beta(n)$ n'est pas légal. À noter que l'expression ci-dessus peut également valoir ∞ même si c est un coloriage légal de $\beta(n)$, par exemple s'il n'existe pas de c_1 satisfaisant la condition tel que $T[n_1][c_1]$ ne soit pas ∞, et de même pour c_2 et n_2.

Et à la fin on renvoie $\min_c T[n][c]$ pour r la racine de T. Cette valeur peut valoir ∞ dans le cas où le graphe n'admet aucun 3-coloriage.

La complexité est de l'ordre de $O(|T| \times 3^{3(k+1)})$. À k fixé, si l'on mesure la complexité en fonction du graphe d'entrée G, comme on peut calculer T en temps linéaire en G, on obtient bien une complexité en $O(|G|)$.