DS 2-A

CCINP

MPI/MPI*, lycée Faidherbe

Résumé

Le sujet comporte 1 exercice tiré des exercices d'oral proposés par le concours CCINP en sujets 0 de préparation en 2022, un exercice sur les automates tiré de l'épreuve d'option informatique MP du concours CCINP en 2022 et d'un problème sur les arbres couvrant tiré de l'épreuve d'option informatique MP du concours CCINP en 1999. En option informatique MP, un sujet est composé de deux exercices et un problème.

Il est demandé de rédiger les différentes parties sur des copies séparées.

I CCINP oraux 0, exercice 2 (type A)

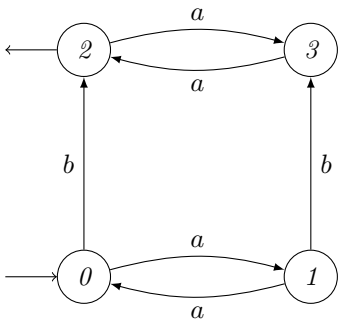
Question 1 Rappeler la définition d'un langage régulier.

C'est un langage pour lequel il existe une expression régulière le dénotant.

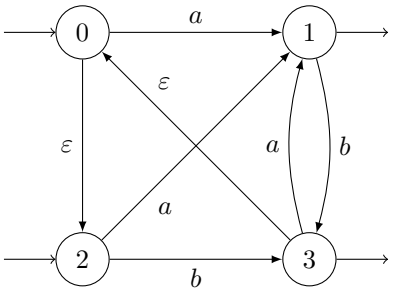
Question 2 Les langages suivants sont-ils réguliers ? Justifier.

1. $L_1 = \{a^n b a^m \ ; \ n, m \in \mathbb{N}\}$.
2. $L_2 = \{a^n b a^m \ ; \ n, m \in \mathbb{N}, n \leqslant m\}$.
3. $L_3 = \{a^n b a^m \ ; \ n, m \in \mathbb{N}, n > m\}$.
4. $L_4 = \{a^n b a^m \ ; \ n, m \in \mathbb{N}, n + m \equiv 0 \mod 2\}$.

1. *L_1 est dénoté par* `a*ba*` *donc régulier.*
2. *Si L_2 est régulier, soit N l'entier fourni par le lemme de l'étoile.*
   On choisit $u = a^N b a^n \in L_2$; on doit pouvoir décomposer $u = u_1 \cdot u_2 \cdot u_3$ avec $|u_1 \cdot u_2| \leqslant N$ et $u_2 \neq \varepsilon$. On a donc $u_1 = a^p$, $u_2 = a^q$ et $u_3 = a^{N-p-q} b a^N$ et on doit avoir $u_1 \cdot u_2^k \cdot u_3 \in L_2$.
   Pour $k = 2$, on aurait $a^{N+q} b a^N$, ce qui est faux car $N + q > N$.
   Par l'absurde L_2 n'est pas régulier.
3. *Si L_3 était régulier alors $L_2 = L_1 \cap \left(\{a, b\}^* \setminus L_3\right)$ le serait.*
   L_2 n'est pas régulier.
4. *L_4 est reconnu par l'automate suivant, il est donc régulier.*



On considère l'automate non déterministe suivant

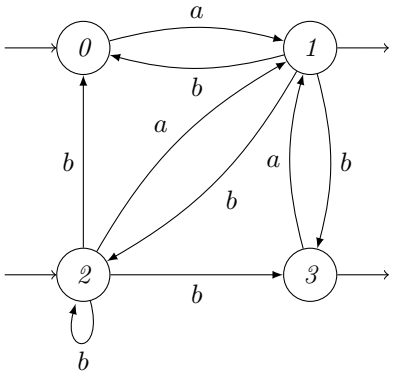


Question 3

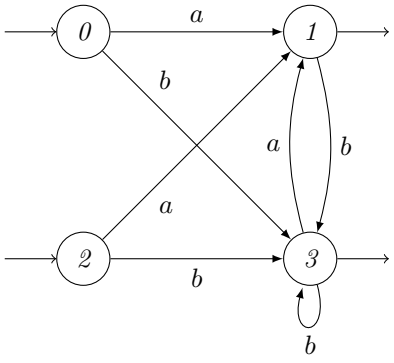
1. Déterminiser cet automate.
2. Construire une expression régulière dénotant le langage reconnu par cet automate.
3. Décrire simplement avec des mots, le langage reconnu par cet automate.

---

1. *On commence par enlever les transitions spontanées. Il y a la méthode du cours où on remplace chaque calcul* $s_0 \xrightarrow{x} s_1 \xrightarrow{\varepsilon} s_2 \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_n$ *par une transition* $s_0 \xrightarrow{x} s_n$.



On peut aussi compléter dans l'autre sens en remplaçant $s_0 \xrightarrow{\varepsilon} s_1 \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_{n-1} \xrightarrow{x} s_n$ *par* $s_0 \xrightarrow{x} s_n$.



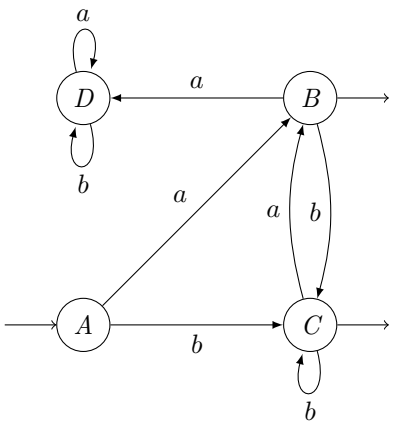
On peut alors déterminiser.
Dans le premier cas on obtient

	$\{0,2\}$	$\{1\}$	$\{0,2,3\}$	$\varnothing$
a	$\{1\}$	$\varnothing$	$\{1\}$	$\varnothing$
b	$\{0,2,3\}$	$\{0,2,3\}$	$\{0,2,3\}$	$\varnothing$
	A	B	C	D

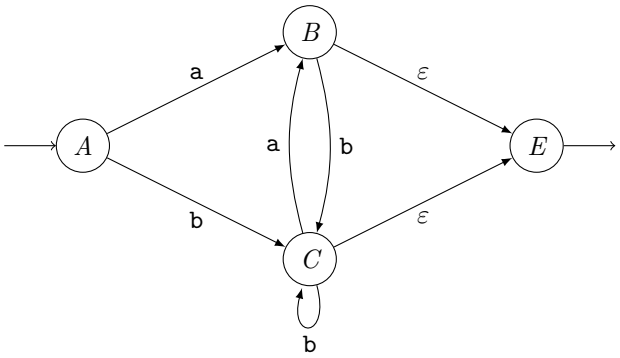
Dans le secon cas on obtient

	$\{0,2\}$	$\{1\}$	$\{3\}$	$\varnothing$
a	$\{1\}$	$\varnothing$	$\{1\}$	$\varnothing$
b	$\{3\}$	$\{3\}$	$\{3\}$	$\varnothing$
	A	B	C	D

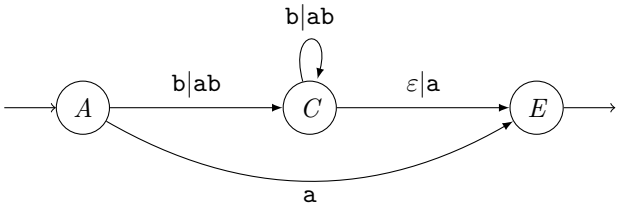
Les automates sont les mêmes, en renommant comme indiqué par les tableaux.



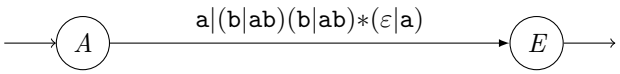
2. *On peut émonder l'automate pour simplifier*
   On normalise en ajoutant un état final, l'état initial est déjà normalisé.



On élimine l'état B.



On élimine l'état C.



3. *Le langage reconnu est celui des mots ne possédant pas deux a consécutifs.*

II CCINP MP 2022, exercice I

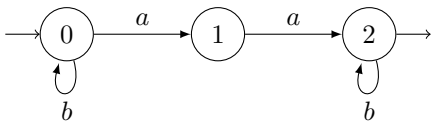
Un **automate augmenté** est un couple $(\mathcal{A}, O)$ où $\mathcal{A} = (\Sigma, Q, \Delta, I, F)$ un automate fini non déterministe et $O \subset Q$ un sous-ensemble d'états de Q. On notera par la suite $\mathcal{A}_O$ le couple $(\mathcal{A}, O)$.
La notion de calcul est la même entre $\mathcal{A}$ et $\mathcal{A}_O$: une suite $q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \cdots \xrightarrow{x_n} q_n$ avec $q_{i+1} \in \Delta(q_i, x_i)$ pour tout i. L'étiquette du calcul est le mot $x_1 x_2 \cdots x_n$.
L'ensemble O introduit la notion de **calcul réussi au seuil** s :
c'est un calcul $q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \cdots \xrightarrow{x_n} q_n$ tel que

- $q_0 \in I$,
- $q_n \in F$,
- pour tout $i \geqslant 0$ tel que $i + s \leqslant n$, $\{q_i \cdots q_{i+s}\} \cap O \neq \varnothing$.

Le **langage reconnu par $\mathcal{A}_O$ au seuil** s, noté $L_s(\mathcal{A}_O)$, est l'ensemble des étiquettes des calculs réussis au seuil s.

II.1 Exemple

Dans cette partie, on utilise l'automate $\mathcal{A}$ suivant



Question 4 Donner sans justification $L_2\left(\mathcal{A}_{\{0\}}\right)$.

Le langage est dénoté par `b*aa`.

Question 5 Donner, en justifiant votre réponse, $L_2\left(\mathcal{A}_{\{1\}}\right)$.

Un calcul ne peut passer qu'une fois par l'état 1 avec l'étiquette aa. Un calcul réussi au seuil 2 ne peut donc contenir que 1 ou 0 lettre avant le premier a et après le second a.
Le langage est donc $\{aa, baa, aab, baab\}$.

Question 6 Soit $s \in \mathbb{N}$.
Donner, en justifiant votre réponse, le cardinal de $L_s(\mathcal{A}_\varnothing)$ en fonction de s.

Les mots de $L_s(\mathcal{A}_\varnothing)$ *ne peuvent avoir plus de* $s-1$ *lettres.*
Comme tout mot reconnu par l'automate doit contenir aa, on a $L_s(\mathcal{A}_\varnothing) = \varnothing$ *pout* $s \in \{0, 1, 2\}$.
Pour $s > 2$, *on peut encadrer aa par* $s-3$ *lettres b donc* $L_s(\mathcal{A}_\varnothing) = \{b^i aab^j \ ; \ i + j \leqslant s - 3\}$.

II.2 Cas général

$\mathcal{A}$ est maintenant un automate non déterministe non précisé.

Question 7 Donner un majorant du nombre de calculs réussis au seuil s ne passant par aucun état de O.

Les calculs réussis au seuil s *ne passant pas par* O *ne peuvent contenir que* $s-1$ *lettres au plus.*
Le langage est donc inclus dans $\Sigma^0 \cup \Sigma^1 \cup \cdots \Sigma^{s-1}$. *Si* σ *contient* m *lettres, le nombre maximal de mots est donc* $\displaystyle\sum_{k=0}^{s-1} m^k = \frac{m^s - 1}{m - 1}$.

Question 8 Soit maintenant un calcul réussi au seuil s qui passe au moins par un état de O. On note $\tilde{q} = q_0 \cdots q_l$ la suite des états correspondants et $i_0 \cdots i_p \in [\![0; l]\!]$ la suite des indices dans $\tilde{q}$ correspondant aux états de O.
Donner, sans justification, en fonction de s, :

1. une majoration de i_0,
2. un encadrement de i_p,
3. une majoration de $i_{j+1} - i_j$ pour $0 \leqslant j < p-1$.

---

1. $i_0 \leqslant s$,
2. $l - s \leqslant i_p \leqslant l$,
3. $i_{j+1} - i_j \leqslant s+1$.

Question 9 Soit $\mathcal{A}_O$ un automate augmenté, avec $|Q| = p$, où Q est l'ensemble des états de l'automate.
Pour $s \in \mathbb{N}$, construire, en justifiant votre construction, un automate fini à $(s+1)\cdot p$ états reconnaissant $L_s(\mathcal{A}_O)$.

On définit un nouvel automate $\mathcal{A}' = (\Sigma, Q', \Delta', I', F')$.

- *On crée, comme suggéré par le cardinal, $s+1$ copies identique de l'automate : $Q' = Q \times \{0,1,2,\dots,s\}$. L'indice h dans (q,h) compte le nombre d'états n'appartenant pas à O jusqu'à q.*
- *Pour toute transition $q \xrightarrow{x} q'$ de $\mathcal{A}$ on associe des transitions dans $\mathcal{A}'$.*
  - *Si $q' \in O$, on définit les transitions $(q,k) \xrightarrow{x} (q',0)$ pour tout k : arriver dans O remet le compteur à 0.*
  - *Si $q' \notin O$, on définit les transitions $(q,k) \xrightarrow{x} (q',k+1)$ pour $0 \leqslant k < s$, (q,s) est bloquant si on n'arrive pas dans O.*
- *$I' = (I \cap O) \times \{0\} \cup (I' \setminus O) \times \{1\}$: on démarre avec un compteur à 0 ou 1.*
- *$T' = T \times \{0,1,\dots,s\}$.*

Prouvons qu'on a $L(\mathcal{A}') \subset L_s(\mathcal{A}_O)$.
Si $u \in L(\mathcal{A}')$, on considère un calcul réussi pour u :
$(q_0,h_0) \xrightarrow{x_1} (q_1,h_1) \xrightarrow{x_2} (q_2,h_2) \cdots \xrightarrow{x_n} (q_n,h_n)$ avec $(q_0,h_0) \in I'$ et $(q_n,h_n) \in T'$.
On a $h_0 = 0$ avec $q_0 \in I \cap O$ ou $h_0 = 1$ avec $q_0 \in I \setminus O$; dans les deux cas $q_0 \in I$.
De même $(q_n,h_n) \in T \times \{0,1,\dots,s\}$ donc $q_n \in T$.
Par définition, toute transition $(q,i) \xrightarrow{x} (q',i')$ de $\mathcal{A}'$ provient d'une transition $q \xrightarrow{x} q'$ de $\mathcal{A}$.
On a donc un calcul réussi dans $\mathcal{A}$, $q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \cdots \xrightarrow{x_n} q_n$ donc $u \in L(\mathcal{A})$.
De plus $h_{j+1} \in \{0, h_j+1\} \cap \{0,1,\dots,s\}$. Si on avait $h_j > 0$, $h_{j+1} > 0$, …, $h_{j+s} > 0$ alors on aurait $h_j \geqslant 1$ et $h_{j+s} = h_j + s \geqslant s+1$, ce qui est impossible. On en déduit qu'il existe p tel que $h_{j+p} = 0$.
Or, si $(q,0)$ est un état initial alors $q \in O$ et si $(q,0)$ est l'état d'arrivée d'une transition alors aussi $q \in O$. donc $q_{j+p} \in O$ pour au moins un indice entre i et $i+s$.
Cela est la définition de $u \in L_s(\mathcal{A}_O)$: on a prouvé l'inclusion.

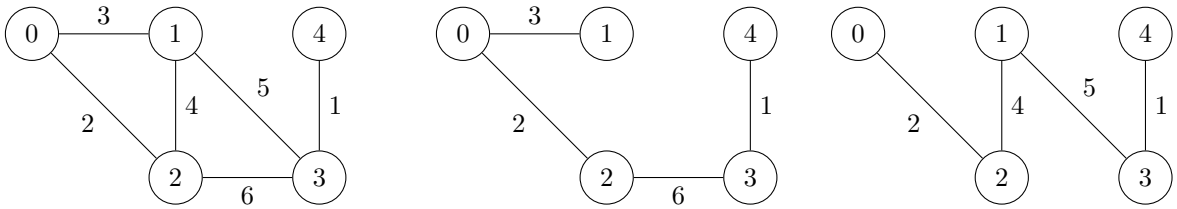
Prouvons qu'on a $L_s(\mathcal{A}_O) \subset L(\mathcal{A}')$.
Soit $u \in L_s(\mathcal{A}_O)$: $q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \cdots \xrightarrow{x_l} q_l$ avec $q_0 \in I$ et $q_l \in F$ est un calcul réussi pour u tel que, pour tout $j < l-s$, $\{q_j, q_{j+1}, \cdots q_{j+s}\} \cap O \neq \varnothing$.
On définit $h_0 = 0$ si $q_0 \in O$ et $h_0 = 1$ sinon puis, par récurrence, $h_{j+1} = h_j + 1$ si $q_{j+1} \notin O$ et $h_{j+1} = 0$ sinon. Avec les notations de l'exercice précédent, on a $h_j = 0$ si et seulement si j est l'un des indices de la forme i_k.

- *$i_0 \leqslant s$ implique que, de 0 à $i_0 - 1$, h_j ne peut croître que de $s-1$ au maximum. Comme la valeur initiale est au plus 1, on a $h_j \leqslant s$ pour tout $j < i_0$.*
- *$i_p \geqslant l - s$ implique que, de i_p à l, h_j ne peut croître que de s au maximum avec $h_{i_k} = 0$ donc $h_j \leqslant s$ pour tout $j \geqslant i_p$.*
- *$i_{j+1} - i_j \leqslant s+1$ et $h_{i_j+k} = h_{i_j} + k = k$ implique $h_{i_j+k} \leqslant h_{i_{j+1}-1} = i_{j+1} - 1 - i_j \leqslant s$. Ainsi $h_k \leqslant s$ pour tout k entre i_j et i_{j+1}.*

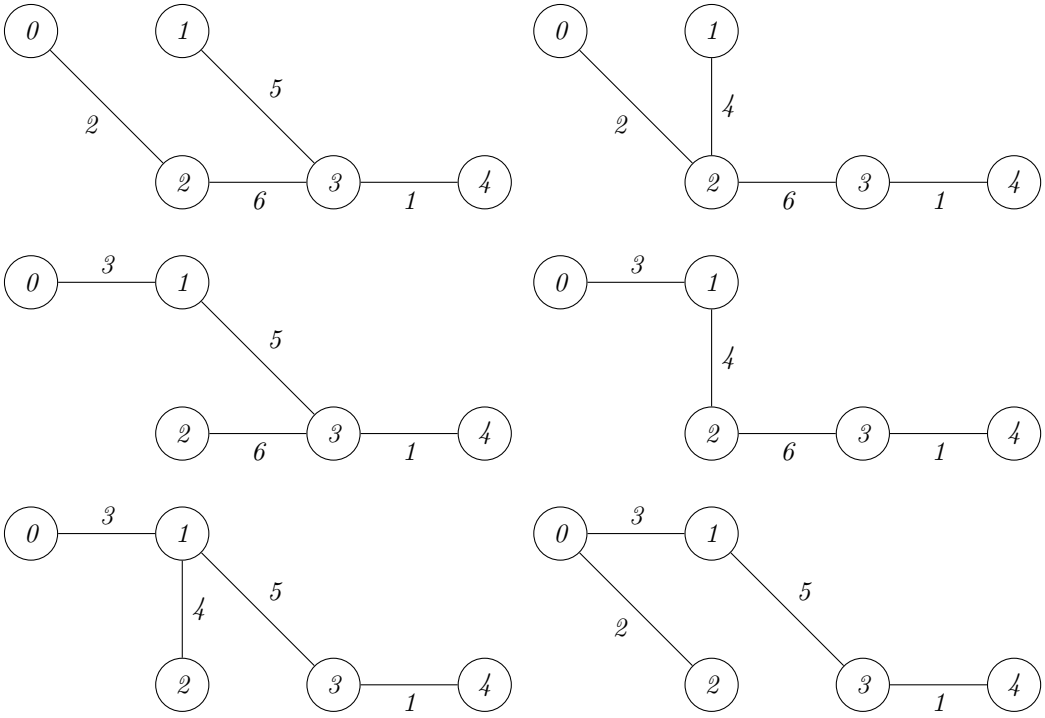
Dans tous les cas, on a $h_j \leqslant s$ pour tout j. Le calcul peut alors se traduire dans $\mathcal{A}'$ et $u \in L(\mathcal{A}')$.

III CCINP MP 1999, Problème

Le but de cette partie est l'étude de l'algorithme de Prim pour la construction d'un arbre couvrant minimal (ACM) dans un graphe non orienté connexe valué (chaque arête a un poids).
Cette étude sera réalisée en deux étapes : l'étude d'un algorithme de construction d'un arbre couvrant, puis l'étude de l'algorithme de Prim qui produit un un arbre couvrant minimal.
Exemple : un graphe G_0 et 2 de ses arbres couvrants, il y en a 8 en tout.



Question 10 Donner une représentation graphique de tous les autres arbres couvrants.



Question 11 Quel est le poids minimal d'un arbre couvrant ?

Le poids minimum est 11, obtenu pour le dernier arbre couvrant ci-dessus.

On suppose dans le problème que les sommets sont des entiers.
Un graphe est défini par la liste de ses sommets et la liste de ses arêtes : une arête est définie par un triplet (s, t, p) où s et t sont les sommets joints par l'arête et p est le poids de l'arête.

```
type arete = int * int *int
type graphe = {sommets : int list; aretes : arete list}
```

L'exemple ci-dessus est défini par

```
let g0 = {sommets = [0; 1; 2; 3; 4];
          aretes = [(0, 1, 3); (0, 2, 2); (1, 2, 4); (1, 3, 5);
                    (2, 3, 6); (3, 4, 1)]}
```

Remarque On tiendra compte du fait qu'une arête admet deux représentations : (s,t,w) et (t,s,w) : une seule appartient à la liste dans le type défini.

On pourra utiliser les fonctions du module `List` en particulier

- `List.length : a' list -> int` qui renvoie le nombre d'éléments d'une liste
- `List.mem : 'a -> a' list -> bool` telle que l'appel `List.mem e l` renvoie la valeur `true` si e est un des éléments de la liste l et la valeur `false` sinon.

III.1 Construction d'un arbre couvrant

On se donne un graphe connexe $G = (S, A)$.
On va construire les arbres couvrants de G en définissant des **sous-arbres** de G. Ce sont des graphes $G' = (S', A')$ inclus dans G ($S' \subset S$ et $A' \subset A$) qui sont des arbres (connexes et acycliques).
L'algorithme construit une suite de sous-arbres $T_i = (S_i, A_i)$ en partant d'un sous-arbre $(\{s_0\}, \varnothing)$ comportant un seul sommet puis en ajoutant un sommet et une arête à chaque étape, jusqu'à atteindre une solution (S, A_p). L'arête ajoutée à chaque étape relie un sommet du sous-arbre à un sommet externe au sous-arbre. L'algorithme s'arrête lorsque tous les sommets de S ont été ajoutés.

III.1.a Correction de l'algorithme

Question 12 Montrer que si $T_i = (S_i, A_i)$ est un sous-arbre et si $\alpha = (s,t,p)$ est une arête de G n'appartenant pas à A_i avec $s \in S_i$ et $t \in S_i$ alors le graphe $\big(S_i, A_i \cup \{\alpha\}\big)$ contient au moins un cycle.

Comme s et t sont des sommets de l'arbre T_i il existe un chemin de s à t dont les arêtes appartiennent à A_i,. L'adjonction de (s,t,p) ferme le chemin en un cycle de $\big(S_i, A_i \cup \{\alpha\}\big)$.

Question 13 Montrer que si $T_i = (S_i, A_i)$ est un sous-arbre et si $\alpha = (s,t,p)$ est une arête de G n'appartenant pas à A_i avec $s \notin S_i$ et $t \notin S_i$ alors le graphe $\big(S_i \cup \{s,t\}, A_i \cup \{\alpha\}\big)$ n'est pas connexe.

Les arêtes de A_i ne peuvent joindre que des sommets de S_i donc il n'existe pas de chemin vers un sommet de S_i vers s ou t dans A_i. L'arête α n'a pas d'extrémité dans S_i donc $\big(S_i \cup \{s,t\}, A_i \cup \{\alpha\}\big)$ n'est pas connexe.

Question 14 Montrer que si $T_i = (S_i, A_i)$ est un sous-arbre et si $\alpha = (s,t,p)$ est une arête de G n'appartenant pas à A_i avec $s \in S_i$ et $t \notin S_i$ alors le graphe $\big(S_i \cup \{t\}, A_i \cup \{\alpha\}\big)$ est un sous-arbre.

On note $T_{i+1} = \big(S_i \cup \{t\}, A_i \cup \{\alpha\}\big)$.
Entre deux points de S_i il existe un chemin dans T_i donc un chemin dans T_{i+1}.
Si x est un sommet de A_i il existe un chemin de x à s dans A_i donc, en ajoutant l'arête α de s à t, il existe un chemin de x à t dans T_{i+1}.
Ainsi il existe un chemin entre deux sommets de T_{i+1} : il est connexe.
Si T_{i+1} contenait un cycle celui-ci ne pourrait pas contenir t car t est de degré 1 dans T_{i+1} ; ce serait donc un cycle dans T_i, ce que l'acyclicité de T_i rend impossible : T_{i+1} est acyclique donc est un arbre.

Question 15 Montrer que l'algorithme termine et fournit un arbre couvrant de G après $k-1$ étapes où k est le cardinal de S.

Si on a $\omega(S') = \varnothing$ alors, en posant $S'' = S \setminus S'$, il n'existe pas d'arêtes entre un sommet de S' et un sommet de S'' donc il n'existe pas de chemin entre un sommet de S' et un sommet de S''. Comme S est connexe, cela impose $S' = \varnothing$ ou $S'' = \varnothing$ donc $S' = S$.
Par contraposée, si $S' \neq \varnothing$ et $S' \neq S$ alors $\omega(S') \neq \varnothing$.

À l'initialisation, on a $T_0 = (\{p_0\}, \varnothing)$, sous-arbre de G.
À chaque étape, on ajoute un sommet et une arête donc, si T_i est défini, $T_i = (S_i, A_i)$ avec $|S_i| = i+1$ et $|A_i| = i$.
On suppose $T_i = (S_i, A_i)$ défini et sous-arbre de G avec $0 \leqslant i < |S| - 1 = k - 1$.
On a alors $|S_i| = i + 1 < |S|$ donc $S_i \neq \varnothing$ et $S_i \neq S$. D'après la remarque précédente $\omega(S_i)$ est non vide donc on peut trouver une arête joignant un sommet de S_i et un sommet de $S \setminus S_i$, ce qui permet de construire T_{i+1}.
D'après la question précédente T_{i+1} est toujours un sous-arbre de G.
À partir de T_0 on peut donc construire T_1, T_2, …, T_{k-1} sous-graphe de G avec T_i contenant $i+1$ sommets. En particulier T_{k-1} contient k sommets donc il contient tous les sommets de G.
C'est alors un arbre couvrant de G

III.1.b Cocycles

Si $S' \subset S$ est un ensemble de sommets, le cocycle de S', noté $\omega(S')$, est le sous-ensemble des arêtes de G reliant un sommet de S' à un sommet de $S \setminus S'$.
Dans l'algorithme, l'arête ajoutée à (S_i, A_i) sera donc prise dans $\omega(S_i)$

Question 16 Écrire une fonction `cocycle : int list -> arete list -> arete list` telle que l'appel `cocycle q ar` calcule le cocycle $\omega(Q)$ du sous-ensemble de sommets Q représenté par `q` lorsque `ar` est l'ensemble des arêtes de G.

On commence par reconnaître les arêtes sortantes

```
let unSeulDans (s, t, p) sommets =
      (List.mem s sommets) && not (List.mem t sommets)
   || (List.mem t sommets) && not (List.mem s sommets)
```

```
let rec cocycle sommets aretes =
  match aretes with
  |[] -> []
  |a :: reste -> if unSeulDans a sommets
      then a :: (cocycle sommets reste)
      else cocycle sommets reste
```

Question 17 Estimer la complexité de `cocycle` en fonction des tailles des ensembles considérés.

`List.mem` est de complexité linéaire en la taille de la liste et on nl'applique 4 fois pour chaque arête : la complexité est un $\mathcal{O}(|\texttt{sommets}| \cdot |\texttt{aretes}|)$.

III.1.c Construction d'un arbre couvrant

Question 18 Écrire une fonction `construire : graphe -> graphe` appliquant l'algorithme précédent, telle que l'appel `construire g` construit un arbre couvrant de G, représenté par `g`.

```
let construire g =
   let rec suivant a =
      match cocycle a.sommets g.aretes with
      |[] -> a
      |(s, t, p) :: _ ->
         if List.mem s a.sommets
         then suivant {sommets = t :: (a.sommets);
                       aretes = (s, t, p) :: a.aretes}
         else suivant {sommets = s :: (a.sommets);
                       aretes = (s, t, p) :: a.aretes}
  in suivant {sommets = [List.hd g.sommets] ; aretes = []}
```

Question 19 Donner une estimation de la complexité de la fonction `construire`.

$G = (S, A)$ avec $|S| = n$ et $|A| = p$.
À chaque appel de la fonction auxiliaire, on calcule le cocycle avec au plus n sommets et une liste de p arêtes. Il y a $n - 1$ appels récursifs donc la complexité totale est un $\mathcal{O}(n^2 \cdot p)$.

III.2 Algorithme de Prim

Pour calculer un arbre couvrant minimal, il est nécessaire d'adapter l'algorithme précédent. Pour cela, l'arête ajoutée au sous-arbre à chaque étape de l'algorithme ne sera plus choisie au hasard dans le cocycle mais sera l'arête de poids minimal parmi les arêtes possibles du cocycle. L'algorithme obtenu est appelé algorithme de Prim.

III.2.a Correction

Soient (S', A') un sous-arbre de G, $T = (S, A_{min})$ un arbre couvrant minimal de G avec $A' \subsetneq A_{min}$ et $\alpha = (s, t, p) \in \omega(A')$ de poids minimal.

Question 20 Montrer que, si $\alpha \notin A_{min}$ alors $\big(S, A_{min} \cup \{\alpha\}\big)$ contient un cycle et que ce cycle contient une arête $\alpha' = (s, t', p')$ différente de α et appartenant à $\omega(S')$.

Si $\alpha \notin A_{min}$ alors c'est une arête joignant deux sommets d'un arbre sans être une arêt de l'arbre : elle crée dont un cycle dans $\big(S, A_{min} \cup \{\alpha\}\big)$ dont α est une des arêtes.
Le complément de α dans le cycle est le chemin de s vers t dans l'arbre et il joint un sommet de S' et un sommet de $S \setminus S'$: il contient donc (au moins) une arête α' joignant un sommet de S' et un sommet de $S \setminus S'$: $\alpha' \in \omega(S')$.

Question 21 Prouver que $T' = \big(S, A_{min} \cup \{\alpha\} \setminus \{\alpha'\}\big)$ est un arbre couvrant minimal de G.

On note $\alpha' = (s', t', p')$. Comme α est minimale das $\omega(S')$ et $\alpha' \in \omega(S')$, on doit avoir $p \leqslant p'$.
De plus T' est toujours connexe car on peut remplacer l'arête α' dans un chemin par le complémentaire de α' dans le cycle obtenu en ajoutant α. Comme T' a autant d'arêtes que T, c'est à dire $|S| - 1$ c'est toujours un arbre couvrant.
Si on note W le poids de T et W' celui de T', on a $W' = W + p - p'$.
Comme T est minimal, on doit avoir $W' \geqslant W$ donc $p \geqslant p'$.
Les deux inégalités impliquent $p = p'$ donc $W = W'$ et T' est aussi de poids minimal, c'est bien un ACM de G.

Question 22 Montrer que 1'application de 1'algorithme de Prim produit un ACM de G.

On montre que, si on choisit une arête de poids minimal dans $\omega(S_i)$ pour tout $i \in \{0, 1, \dots, |S|-1\}$ alors le sous-arbre $T_i = (S_i, A_i)$ que l'on construit vérifie qu'il existe un ACM $T = (S, A_{min})$ tel que $A_i \subset A_{min}$.
C'est vrai pour $i = 0$ car alors $A_0 = \varnothing$ est inclus dans tout ensemble.
On suppose que, pour $i \leqslant |S| - 2$, on a $T_i = (S_i, A_i)$ avec $A_i \subset A_{min}$ et $T = (S, A_{min})$ ACM. Soient α de poids minimum dans $\omega(S_i)$, $S_{i+1} = S_i \cup \{s\}$ avec s extréùité de α n'appartenant pas à S_i et $A_{i+1} = A_i \cup \{\alpha\}$.

- *Si $\alpha \in A_{min}$ alors $A_{i+1} = A_i \cup \{\alpha\} \subset A_{min}$ et $T_{i+1} = (S_{i+1}, A_{i+1})$ vérifie bien la propriété.*
- *Si $\alpha \notin A_{min}$, la question précédente montre qu'il existe $\alpha' \in \omega(S_i \cap A_{min}$ donc $\alpha' \notin A_i$, telle que $T' = \big(S, A_{min} \cup \{\alpha\} \setminus \{\alpha'\}\big)$ est un ACM. On a alors $A_{i+1} = A_i \cup \{\alpha\} \subset A_{min} \cup \{\alpha\} \setminus \{\alpha'\}$ et $T_{i+1} = (S_{i+1}, A_{i+1})$ vérifie bien la propriété.*

Le résultat est donc valide pour tout $i \in \{0, 1, \dots, |S|-1\}$.
En particulier, pour $i = |S| - 1$, on a un arbre couvrant de G dont l'ensemble des arêtes est inclus dans l'ensemble des arêtes d'un ACM. Comme le cardinal de l'ensemble des arête d'un arbre couvrant est $|S| - 1$, les deux ensembles d'arêtes sont égaux et l'arbre couvrant produit est un ACM.

III.2.b Construction

Question 23 Modifier la fonction `cocycle` de la question 16 pour construire une liste ordonnée selon les poids croissants.

On commence par une fonction d'insertion dans une liste ordonnée selon les poids.

```
let rec inserer (s, t, p) liste =
   match liste with
   |[] -> [(s, t, p)]
   |(s', t', p') :: reste ->
      if p <= p'
      then (s, t, p) :: liste
      else (s', t', p') :: (inserer (s, t, p) reste)
```

```
let rec cocycle1 sommets aretes =
  match aretes with
  |[] -> []
  |a :: reste -> if unSeulDans a sommets
      then inserer a(cocycle sommets reste)
      else cocycle sommets reste
```

Question 24 Donner une estimation de la complexité de la fonction `cocycle` modifiée.

Dans la boucle de `cocycle` on ajoute la complexité de `inserer` qui est linéaire en la taille de la liste, ici majorée par le nombre d'arêtes.
On aboutit à une complexité en $\mathcal{O}\big((|\texttt{sommets}| + |\texttt{aretes}|) \cdot |\texttt{aretes}|\big)$

Question 25 Donner une estimation de la complexité de la fonction `construire` utilisant la nouvelle fonction `cocycle`.

Avec les notation de la question 19, on a une complexité en $\mathcal{O}\big(n \cdot p \cdot (n+p)\big)$.
Pour un graphe connexe, on a $p \geqslant n - 1$ donc la complexité est en $\mathcal{O}\big(n \cdot p^2\big)$.