Problème 1 - Mots transposés, automates et palindromes

Dans tout ce problème, les mots sont construits sur un alphabet à deux lettres $\Sigma = \{a, b\}$. Le mot vide est noté ε.

Si $u = a_1 \dots a_n$ est un mot formé des lettres $a_1, \dots, a_n$, alors on note ${}^t u = a_n \dots a_1$ son transposé. Si L est un langage sur Σ, on note ${}^t L$ le langage formé des transposés des mots de L.

Pour $n \in \mathbb{N}^*$, si u est un mot de longueur au moins n, on note $u|_n$ son préfixe de longueur n. On note L_n l'ensemble des mots u de longueur au moins $2n$ tels que ${}^t(u|_n)$ soit le suffixe de longueur n de u.

Question 1)

a) Montrez que L_1 est un langage régulier en donnant une expression régulière qui le définit.

b) Montrez la même chose en donnant cette fois-ci un automate non déterministe qui reconnaît le langage L_1 : on impose la contrainte que cet automate ne doit avoir qu'un seul état initial et un seul état final.

Question 2) Montrez que L_2 est de même un langage régulier en donnant un automate non déterministe qui reconnaît le langage L_2 : on impose encore la contrainte que cet automate ne doit avoir qu'un seul état initial et un seul état final.

Question 3) Déterminisez l'automate précédent.

Question 4) Montrez plus généralement que pour tout $n \in \mathbb{N}^*$, L_n est un langage reconnaissable (indication : écrivez L_n comme une union finie de langages reconnaissables).

Question 5) Montrez que si L est un langage reconnaissable, alors ${}^t L$ l'est aussi.

Question 6) Un mot u est un palindrome quand ${}^t u = u$. On note P le langage formé des palindromes et on suppose que P est un langage reconnaissable.

a) Justifiez qu'il existe un automate déterministe complet qui reconnaît P.

b) Soit δ la fonction de transition de cet automate et δ^* la fonction de transition étendue. On note q_0 l'état initial de cet automate.
Montrez qu'il existe $(i, j) \in \mathbb{N}^2$ tel que $i < j$ et $\delta^*(q_0, (ab)^i) = \delta^*(q_0, (ab)^j)$.

c) Montrez alors que le mot $(ab)^i(ba)^j$ est reconnu par l'automate.

d) Concluez.

Problème 2 - Arbres binaires de recherche

On considère des arbres binaires dont les étiquette sont des éléments d'un ensemble ordonné.

Un arbre binaire est soit vide, soit un nœud formé d'une étiquette et de deux arbres binaires :

```
type 'a arbre = Vide | Noeud of ('a arbre * 'a * 'a arbre);;
```

Avec ce type, Noeud(fg, x, fd) désigne l'arbre étiqueté par x, de fils gauche fg et de fils droit fd.

Dans la suite, la lettre t désigne un arbre binaire (« tree » en anglais).

Question 1) Écrivez deux fonctions etiquette_droite t et etiquette_gauche t qui calculent l'étiquette la plus à droite (resp. à gauche) d'un arbre t non vide. Si l'arbre est vide, on interrompt le calcul avec failwith "un␣message". Quelle est la complexité de ces fonctions en fonction de h, hauteur de l'arbre ? en fonction de n, nombre d'éléments de l'arbre ?

Question 2) Écrivez une fonction etiquettes t qui calcule la liste de toutes les étiquettes des nœuds d'un arbre en effectuant un parcours gauche-droite de l'arbre : les étiquettes doivent se lire de gauche à droite, c'est-à-dire celles d'un fils gauche avant celle de la racine, suivie de celles du fils droit. Quelle est sa complexité en fonction de n, nombre d'éléments de l'arbre ?

Un arbre binaire est un arbre binaire de recherche (ABR) quand :

— il est Vide ou
— il est de la forme Noeud(fg, x, fd) où fg et fd sont deux ABR et toutes les étiquettes y des nœuds du fils gauche, toutes les étiquettes z des nœuds du fils droit vérifient l'inégalité $y < x < z$ (remarque : les étiquettes sont donc toutes distinctes).

Question 3) Écrivez une fonction abr t qui vérifie si un arbre est un ABR.

Question 4) Montrez qu'un arbre est un ABR si et seulement si la liste de ses étiquettes calculées par le parcours gauche-droite précédent est une liste triée par ordre strictement croissant.

Question 5) Écrivez une fonction construire_abr l qui utilise la méthode « diviser pour régner » pour construire à partir d'une liste triée par ordre strictement croissant un ABR. Quelle est sa complexité en fonction de n, longueur de la liste ? Quelle est la hauteur d'un tel ABR en fonction de n ?

Problème 3 - Graphes triangulés

Soit $G = (S, A)$ un graphe non orienté : S est un ensemble de sommets, A un ensemble d'arêtes $\{a, b\}$ où a, b sont deux sommets distincts. Deux sommets sont dits adjacents ou voisins quand ils sont les sommets d'une arête. On note $V(a)$ l'ensemble des voisins d'un sommet a. Le degré d'un sommet a est le nombre de ses voisins, noté $\deg(a)$: $\deg(a) = \operatorname{card} V(a)$.

Si S' est un sous-ensemble de S, on note $G[S']$ le graphe (S', A') où A' est le sous-ensemble de A formé des arêtes qui relient deux sommets de S'.

Un chemin dans G est une suite de sommets $(s_0, \ldots, s_p)$ telle que pour tout $i \in [\![1, p-1]\!]$, $\{s_i, s_{i+1}\}$ soit une arête de G (un chemin peut être réduit à un seul sommet) : s_0 et s_p sont les extrémités du chemin, p est la longueur du chemin (*i.e.* le nombre d'arêtes). Deux sommets sont connectés quand il existe un chemin dont ils sont les extrémités. Le graphe est dit connexe quand tous les sommets sont connectés deux à deux. Si a est un sommet de G, la composante connexe de a, notée $C(a)$, est l'ensemble des sommets de G connectés à a.

Partie 1 - Séparateurs

Deux sommets a, b sont séparables quand il existe une partie T de S, qui ne contient ni a, ni b, telle que a et b ne soient pas connectés dans $G[S - T]$. Une telle partie T est alors appelé un (a, b)-séparateur (ou séparateur de a et b) et on dit qu'il est minimal quand aucune de ses parties strictes n'est un (a, b)-séparateur.

Question 1) Un exemple : dans cette question $S = \{1, 2, 3, 5, 6, 15\}$ et deux éléments distinct de S sont reliés par une arête quand l'un des deux divise l'autre. Représentez graphiquement le graphe $G = (S, A)$. Donnez tous les $(2, 5)$-séparateurs et précisez lesquels sont minimaux.

Question 2) Montrez que deux sommets a et b sont séparables si et seulement si ils ne sont pas voisins. En particulier, justifiez qu'on peut alors trouver un (a, b)-séparateur qui contient moins de $\min(\deg a, \deg b)$ sommets.

Un graphe est dit séparable quand il existe deux sommets séparables. Un sous-ensemble T de S est appelé une clique quand le graphe $G[T]$ n'est pas séparable.

Question 3) Donnez un exemple de graphe non séparable à 4 éléments. Si G est un graphe non séparable à n sommets, combien a-t-il d'arêtes ?

Question 4) Soit a, b deux sommets non voisins.

a) À quelle condition l'ensemble vide est-il un (a, b)-séparateur ?

b) On suppose que a et b sont connectés. Montrez qu'il existe un singleton séparateur de a et b si et seulement si tous les chemins reliant a et b ont un sommet commun.

Question 5) Soit u, v deux sommets séparables de G et T un (u, v)-séparateur. On note U, V les composantes connexes de u et v respectivement dans le graphe $G[S - T]$. Montrez que les voisins de u dans G sont dans $T \cup U$.

Question 6) Soit a, b deux sommets séparables. Montrez que $V(a) \cap C(b)$ est un (a, b)-séparateur minimal.

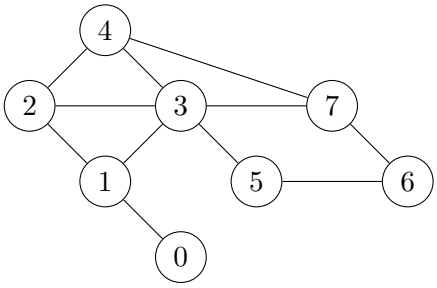
Question 7) Donnez le principe d'un algorithme qui calcule un (a, b)-séparateur minimal. Quelle est sa complexité en fonction de n, nombre de sommets du graphe et m, nombre d'arêtes du graphe ?

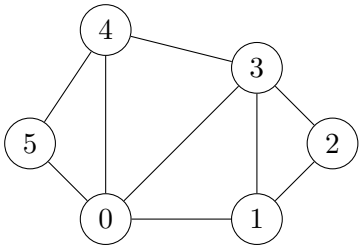
Partie 2 - Graphes triangulés

Un cycle du graphe G est un chemin dont les extrémités sont égales. Un cycle est élémentaire quand il ne passe pas deux fois par le même sommet, sauf les extrémités. Enfin, une corde d'un cycle élémentaire est une arête qui relie deux sommets non consécutifs.

On dit que le graphe G est triangulé quand tout cycle de longueur au moins égale à 4 possède une corde.

Question 1) Exemples : parmi les deux graphes ci-dessous, lequel est triangulé, lequel ne l'est pas ?





Question 2) Montrez que si un graphe est triangulé séparable et si T est un séparateur minimal, alors T est une clique.

Un sommet d'un graphe est dit simplicial lorsque ses voisins sont deux à deux adjacents. On suppose dans toute la suite que les graphes sont triangulés.

Question 3) Dans le cas où G est un graphe non séparable, justifiez qu'il possède un sommet simplicial.

Question 4) Dans le cas où G est un graphe séparable, on choisit u, v deux sommets séparables de G et T un (u, v)-séparateur minimal. On note U, V les composantes connexes de u et v respectivement dans le graphe $G[S-T]$. On suppose que $G[T \cup U]$ est un graphe séparable qui possède deux sommets simpliciaux dans $G[T \cup U]$ et non adjacents .

a) Montrez que l'un des deux est dans U.

b) Montrez qu'il est alors un sommet simplicial dans G.

Question 5) Montrez par récurrence sur le nombre de sommets de G que si G est un graphe triangulé, alors G possède un sommet simplicial et que si de plus G est séparable, alors il en possède au moins deux non adjacents.

Question 6) Donnez le principe d'un algorithme qui permet de savoir si un graphe est triangulé ou pas.

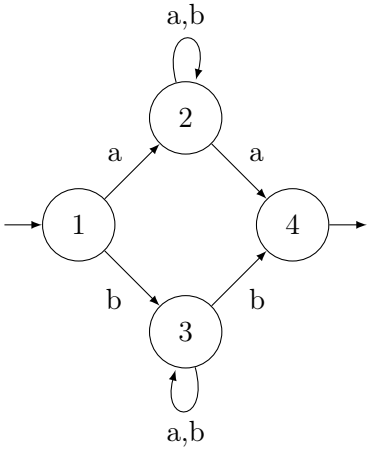
Devoir surveillé 3 - Corrigé

Problème 1

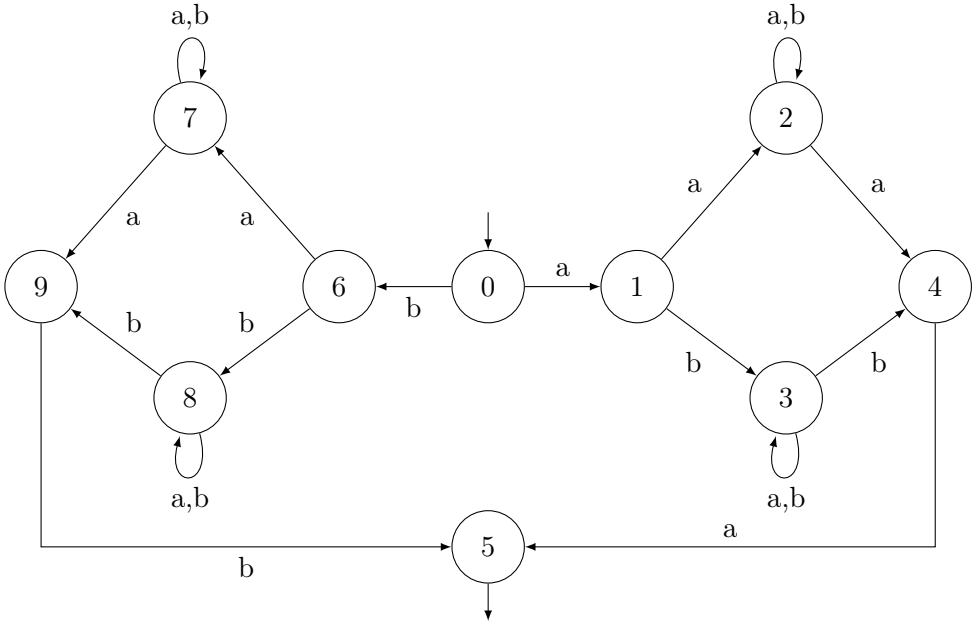
Question 1)

a) L_1 est le langage régulier défini par l'expression régulière $a(a|b)^*a \,|\, b(a|b)^*b$

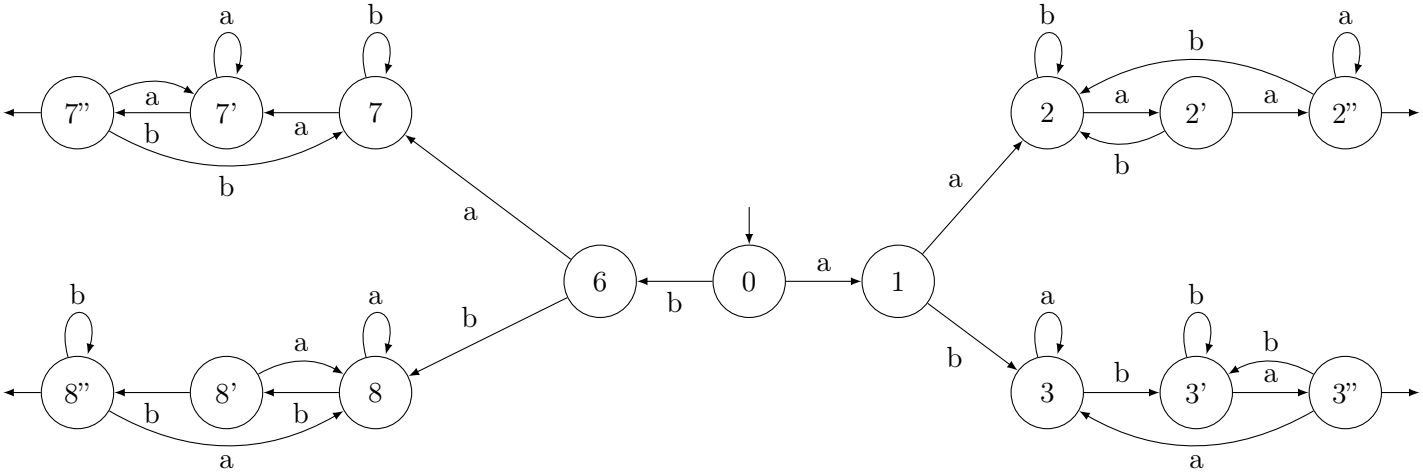
b)



Question 2)



Question 3) On applique l'algorithme des parties vu en cours qui permet de déterminiser un automate non déterministe. Sans détail, on obtient ici par exemple l'automate suivant :



Question 4) $L_n = \bigcup_{u \in \Sigma^n} u\Sigma^* {}^t u$ est une union finie de langages reconnaissables, donc il est aussi reconnaissable.

Question 5) Si L est un langage reconnaissable, alors il existe un automate qui reconnaît le langage L. Il suffit de considérer l'automate obtenu en échangeant les états initiaux en états finaux et vice-versa et de retourner le sens de toutes les transitions : ce nouvel automate reconnaît ${}^t L$.

Question 6)

a) Tout langage reconnaissable est reconnu par un automate déterministe, qui peut ensuite être complété en ajoutant un état-puits, en transformant les blocages par des transitions vers cet état-puits et en ajoutant toutes les transitions possibles de cet état-puits vers lui-même. L'automate ainsi construit reconnaît le même langage.

b) Le nombre d'états de l'automate est fini, or l'ensemble des mots $(ab)^i$ est infini, donc l'application (bien définie car l'automate est complet) $i \mapsto \delta^*(q_0, (ab)^i)$ n'est pas injective, donc par définition de l'injectivité, il existe $(i, j) \in \mathbb{N}^2$ tel que $i < j$ et $\delta^*(q_0, (ab)^i) = \delta^*(q_0, (ab)^j)$.

c) Le mot $(ab)^j(ba)^j$ est reconnu par l'automate donc $\delta^*(q_0, (ab)^j(ba)^j) = t$ est un état terminal.
Or $t = \delta^*(q_0, (ab)^j(ba)^j) = \delta^*(\delta^*(q_0, (ab)^j), (ba)^j)$ donc $t = \delta^*(\delta^*(q_0, (ab)^i), (ba)^j) = \delta^*(q_0, (ab)^i(ba)^j)$.
Donc le mot $(ab)^i(ba)^j$ est reconnu par l'automate : contradiction, car ce n'est pas un palindrome.

d) On a ainsi montré que le langage des palindromes n'est pas un langage reconnaissable (donc pas régulier).

Problème 2

Question 1)

```
let rec etiquette_droite t =
  match t with
  | Vide -> failwith "vide"
  | Noeud(fg, x, fd) -> if fd = Vide then x else etiquette_droite fd;;

let rec etiquette_gauche t =
  match t with
  | Vide -> failwith "vide"
  | Noeud(fg, x, fd) -> if fg = Vide then x else etiquette_gauche fg;;
```

Pour récupérer les étiquettes droite et gauche, on parcourt linéairement une seule branche de l'arbre, donc la complexité est en $O(h)$. Comme $h \leqslant n$, on a aussi une complexité en $O(n)$.

Question 2)

```
let rec etiquettes t =
  match t with
  | Vide -> []
  | Noeud(fg, x, fd) -> etiquettes fg @ (x :: etiquettes fd);;
```

Cette fois-ci il faut parcourir tout l'arbre et on passe une et une seule fois par chaque nœud, donc le parcours se fait en $O(n)$ appels récursifs. Comme chaque étiquette est concaténée une seule fois et que la concaténation des listes est de complexité linéaire par rapport à la première, on en déduit que la complexité totale est en $O(n^2)$.

Question 3)

```
let rec abr t =
  match t with
  | Vide -> true
  | Noeud(Vide,x,fd) -> abr fd && x < etiquette_gauche fd
  | Noeud(fg,x,Vide) -> abr fg && etiquette_droite fg < x
  | Noeud(fg,x,fd) -> abr fd && x < etiquette_gauche fd
                      && abr fg && etiquette_droite fg < x;;
```

Avant de comparer les étiquettes du fils gauche avec celle de la racine, on teste d'abord si le fils gauche est un ABR. Si oui, son plus grand élément est son étiquette droite, donc pour que les étiquettes du fils gauche soient toutes inférieures à celle de la racine, il faut et il suffit que sa plus grande étiquette le soit. Et on fait pareil pour l'autre fils en échangeant droite et gauche.

Question 4) Par récurrence sur le nombre d'éléments de l'arbre.

Pour un arbre vide ou à un seul élément, l'équivalence est vraie.

Soit un arbre à n éléments. Ses fils gauche et droit ont strictement moins de n éléments, on suppose que l'équivalence est vraie pour ces deux arbres.

Si l'arbre est un ABR, alors ses fils le sont aussi donc par hypothèse de récurrence, leurs étiquettes sont rangées par ordre croissant. L'arbre est un ABR donc l'étiquette de la racine est plus grande que celles du fils gauche et plus petite que celles du fisl droit. Comme on l'intercale entre les deux listes, on obtient une liste triée par ordre croissant.

Réciproquement, si les étiquettes sont rangées par ordre croissant, alors comme celles du fils gauche sont à gauche de la liste, elles le sotn aussi, donc par hypothèse de récurrence, le fils gauche est un ABR. Et de même pour le fils droit. Comme l'étiquette de la racine est entre les deux listes, elle est donc plus grande que toutes les étiquettes du fils gauche et plus petite que celles du fils droit, donc l'arbre est un ABR.

D'après le principe de récurrence forte, l'équivalence est vraie pour tous les arbres.

Question 5) Il faut d'abord une fonction qui scinde la liste en deux moitiés presque égales. Celle-ci est de complexité $O(n)$.

Puis la méthode diviser pour régner donne la relation $C(n) = 2C(n/2) + O(n)$ donc $C(n) = O(n \log n)$.

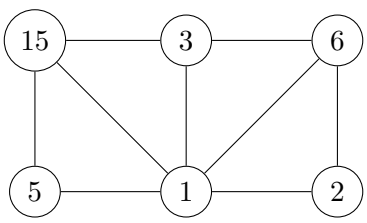
```
let scinder l =
  let n = list_length l in
  let rec scindaux k l =
    if k = 0 then ([],l)
    else match l with
         | [] -> failwith "liste vide"
         | x :: q -> let (l1,l2) = scindaux (k-1) q in
                     (x :: l1, l2)
  in
  scindaux (n/2) l;;

let rec construire_abr l =
  match l with
  | [] -> Vide
  | _ -> let (l1, x :: l2) = scinder l in
         Noeud( construire_abr l1, x, construire_abr l2);;
```

Problème 3

Partie 1

Question 1)



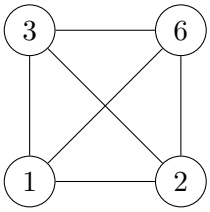
Les $(2,5)$-séparateurs sont les parties $\{1,6\}$, $\{1,3\}$, $\{1,15\}$, $\{1,3,6\}$, $\{1,3,15\}$, $\{1,6,15\}$, $\{1,3,6,15\}$.

Les minimaux sont donc les parties $\{1,6\}$, $\{1,3\}$, $\{1,15\}$.

Question 2) Si deux sommets sont voisins, alors pour supprimer l'arête entre les deux, il faut supprimer l'un des sommets, ce qui est contraire à la définition de séparateur, donc ils sont inséparables.

Si deux sommets ne sont pas voisins, alors l'ensemble des voisins d'un des deux sommets est un séparateur, car tout chemin reliant les deux sommets doit passer par cet ensemble. En prenant le plus petit des deux ensembles de voisins, on peut donc trouver un (a, b)-séparateur qui contient moins de $\min(\deg a, \deg b)$ sommets.

Question 3) Un graphe non séparable est un graphe complet, c'est-à-dire dont tous les sommets sont 2 à 2 reliés :



Plus généralement, un graphe non séparable à n sommets contient donc $\frac{n(n-1)}{2}$ arêtes, c'est-à-dire le nombre de parties à deux éléments dans un ensemble à n éléments.

Question 4)

a) L'ensemble vide est-il un (a, b)-séparateur si et seulement si a et b sont déjà séparés, donc s'ils ne sont pas connectés.

b) Si tous les chemins reliant a et b ont un sommet commun, alors en supprimant ce sommet, on sépare les sommets a et b, donc le singleton sommet commun est un séparateur.

Réciproquement, si en supprimant un point, alors on supprime tous les chemins reliant a et b, alors cela signifie que ces chemins passent tous par ce point.

Question 5) S'il existe un voisin de u dans V, alors on peut en une arête passer de u à la composante connexe de v, donc u et v sont reliés par un chemin : impossible car on les a séparés par le séparateur.

Question 6) On sait déjà que $V(a)$ est un (a, b)-séparateur, donc on peut en extraire un minimal.

Un chemin reliant a et b passe nécessairement par un voisin de a (car a et b séparables), qui est donc relié à b, donc si on retire les points de $V(a) \cap C(b)$, on supprime toute possibilité de relier a et b, donc $V(a) \cap C(b)$ est un (a, b)-séparateur.

Il est minimal, car s'il en existe un plus petit (au sens de l'inclusion), alors il existe un point de $V(a) \cap C(b)$ qu'on peut ne pas enlever : ce point est à la fois relié à b et à a, donc a et b sont reliés, ce qui contredit le fait qu'on ait un séparateur.

Question 7) D'après le cours, on sait qu'un parcours en profondeur ou en largeur d'un graphe permet de calculer la composante connexe d'un sommet en complexité $O(m + n)$ en choisissant une représentation des sous-ensembles de S sous forme de tableaux de booléens.

Puis intersecter cet ensemble avec les voisins de a (qui sont au pire au nombre de $n - 1$) se fait en complexité linéaire $O(n)$ grâce à ce tableau (vérifier qu'un élément est dans la composante connexe se fait en complexité constante dans ce cas). Donc on peut obtenir un séparateur minimal en $O(n + m)$.

Partie 2

Question 1) Le premier n'est pas triangulé, car le cycle $(3, 5, 6, 7)$ est de longueur 4 et n'a pas de corde. Le second est triangulé, car tout sommet appartient à un triangle d'arêtes.

Question 2) Une clique est un ensemble de points deux à deux voisins.

Soit T un (a, b)-séparateur minimal de G. Tout sommet de T est sur un chemin qui connecte a à b, sinon on pourrait le retirer de T et obtenir un séparateur plus petit.

On suppose par l'absurde que T possède deux sommets x et y non voisins. On prend le plus petit cycle qui passe par a, x, b, y. Ce cycle est au moins de longueur 4. Comme le graphe est triangulé, et que les sommets de part et d'autre du séparateur ne sont pas adjacents (par choix d'un cycle minimal), le cycle admet nécessairement la corde $\{x, y\}$: contradiction.

Question 3) Un graphe non séparable est complet : si on supprime un sommet, il reste complet, donc tout point est simplicial.

Question 4)

a) Si les deux sont dans T, alors comme T est un séparateur minimal dans un graphe triangulé, c'est une clique, donc les deux sommets simpliciaux sont voisins, ce qui contredit l'hypothèse.

b) Le sommet qui est dans U a tous ses voisins dans $T \cup U$, donc comme il est simplicial dans $G[T \cup U]$, il l'est encore dans G.

Question 5) Par récurrence sur la taille du graphe.

Si G est d'ordre 1 ou 2, il est triangulé et possède un sommet simplicial au moins. Si G est d'ordre 3, triangulé et séparable, alors il possède deux points séparables, qui sont simpliciaux (réfléchir à la forme du graphe).

Si la proposition est vraie pour les graphes ayant moins de n sommets, alors soit G un graphe triangulé à n sommets.

Si G est séparable, tous les sommets sont simpliciaux.

Sinon soient deux sommets u et u non adjacents, et soit T un (u, u)-séparateur minimal. On note U, V les composantes connexes de u et v respectivement dans le graphe $G[S - T]$.

Deux cas se présentent.

— Soit $G[T \cup U]$ est non séparable, alors dans ce cas, tout point est simplicial : on en choisit un dans U, il est simplicial dans G.

— Sinon par hypothèse de récurrence $G[T \cup U]$ possède deux sommets simpliciaux non adjacents donc d'après la question précédente, celui qui est dans U est simplicial dans G.

De même, il existe un sommet simplicial dans V.

Ceci prouve donc la proposition de récurrence au rang n.

D'après le principe de récurrence forte, la proposition est vraie.

Question 6) En appliquant récursivement la proposition précédente, on montre que si G est triangulé, on trouve un sommet simplicial. Après suppression, le graphe reste triangulé, et on recommence jusqu'à aboutir au graphe vide.

Réciproquement, si un graphe n'est pas triangulé, c'est-à-dire s'il existe un cycle sans corde de plus de trois sommets, aucun de ces sommets ne pourra être éliminé.

L'algorithme consiste donc à trouver un sommet simplicial : s'il n'y en a pas, le graphe n'est pas triangulé. S'il y en a un, on le supprime et le graphe initial est triangulé si et seulement il le reste après suppression.