Chapitre 6 : Trois algorithmes sur les graphes

1 Algorithme de Kosaraju

L'algorithme de Kosaraju est un algorithme permettant le calcul des composantes fortement connexes (notées CFC) d'un graphe orienté. Dans toute cette section, on travaille sur un graphe orienté $G = (S, A)$, on notera $n = \text{card}(S)$ et $m = \text{card}(A)$.

Notation 1.1

Pour $(u, v) \in S^2$, on note $u \xrightarrow{*} v$ (resp. $u \xrightarrow{k} v$) s'il existe un chemin (resp. un chemin de longueur k) menant de u à v dans G.

1.1 Tri préfixe (rappels)

Rappel 1.2

Soit $T = (T_i)_{i \in [\![1,n]\!]}$ une permutation des sommets de G.
*On définit le **rang** d'un sommet u dans la permutation T, noté $rg_T(u)$, comme étant le plus petit indice d'un élément dans la même CFC que u.*

$$rg_T(u) \overset{\text{déf}}{=} \min\{i \in [\![1, n]\!] \mid T_i \sim_G u\}$$

*On dit que T est un **tri préfixe** de G dès lors que $\forall (u, v) \in S^2, (u, v) \in A \Rightarrow rg_T(u) \leqslant rg_T(v)$.*

Exercice de cours 1.3

Rappeler comment on calcule un tri préfixe d'un graphe, et avec quelle complexité.

Définition 1.4

Pour $(u, v) \in S^2$ deux sommets du graphe :
- *u est **descendant** (resp. **ascendant**) de v si et seulement si $v \xrightarrow{*} u$ (resp. $u \xrightarrow{*} v$) ;*
- *u est **descendant propre** (resp. **ascendant propre**) de v si et seulement si $v \xrightarrow{*} u$ et $u \not\xrightarrow{*} v$ (resp. $u \xrightarrow{*} v$ et $v \not\xrightarrow{*} u$).*

Exercice de cours 1.5

Soit $(u, v) \in S^2$. Soit T un tri préfixe de G.
- Si $u \sim v$, que peut-on dire de $\text{rg}_T(u)$ et $\text{rg}_T(v)$? Peut-on savoir qui de u et v apparaît en premier dans T ?
- Si u est descendant propre de v que peut-on dire de $\text{rg}_T(u)$ et $\text{rg}_T(v)$? Peut-on savoir qui de u et v apparaît en premier dans T ?

1.2 Graphe transposé et CFC

Définition 1.6

*On appelle **graphe transposé** de G le graphe $G^t = (S, B)$ où $B = \{(v, u) \mid (u, v) \in A\}$.*
Autrement dit c'est le graphe obtenu en retournant tous les arcs.

Remarque 1.7

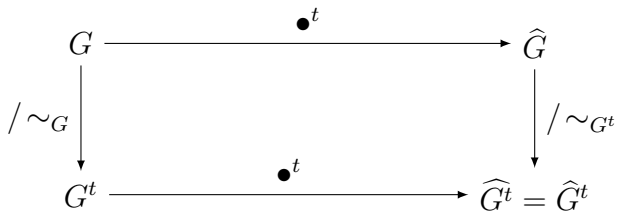
Le nom vient de la représentation matricielle : la matrice d'adjacence du graphe transposé est la transposée de la matrice d'adjacence du graphe initial.

Exercice de cours 1.8

Donner le pseudo-code d'un algorithme de calcul du graphe transposé en complexité linéaire, pour une représentation par matrice d'adjacence, puis pour une représentation par table de listes d'adjacence.

Proposition 1.9

- *La relation $\sim$ définie par la mutuelle accessibilité dans G est la même que celle définie de même dans G^t. Par conséquent les CFC de G sont les mêmes que celles de G^t.*
- *Le graphe réduit du transposé est le transposé du graphe réduit, i.e. $\widehat{G^t} = \widehat{G}^t$.*
  Autrement dit le passage au quotient selon $\sim$ et le retournement des arcs commutent.



Exercice de cours 1.10

Démontrer le second point de la proposition 1.9.

Remarque 1.11

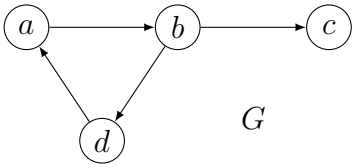
Soit $(u, v) \in S^2$.
u est ascendant (propre) de v dans G si et seulement si u est descendant (propre) de v dans G^t.
u est descendant (propre) de v dans G si et seulement si u est ascendant (propre) de v dans G^t.

Vocabulaire 1.12

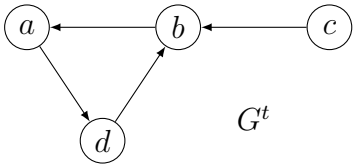
Soit T une permutation des sommets. Soit L un parcours de G^t.
*On dit que les points de régénération sont **choisis prioritairement selon** T si les points de régénération de L apparaissent dans le même ordre dans T et dans L.*
De manière équivalente, pour tout $r \in [\![1, n]\!]$ tel que L_r est point de régénération de L, $L_r = T_{i_0}$ avec $i_0 = \min \{i \in [\![1, n]\!] \mid T_i \notin \{L_j \mid j \in [\![1, r[\![\}\}$.

Lemme 1.13

Soit T une permutation des sommets de G. Soit L un parcours de G. Si les points de régénération de L sont choisis prioritairement selon T, alors pour tout L_r point de régénération de L, pour tout $i \geqslant r$, $rg_T(L_r) \leqslant rg_T(L_i)$.



G



G^t

Démonstration : Par l'absurde, si $\mathsf{rg}_T(L_i) < \mathsf{rg}_T(L_r)$, alors soit v le premier sommet dans T tel que $v \sim L_i$, v apparaît à l'indice $\mathsf{rg}_T(L_i)$ dans T donc avant L_r. Étant donné que L_i n'est pas visité au moment où L_r est choisi comme point de régénération, et que $v \xrightarrow{*} L_i$, v n'a pas non plus été visité à ce moment là♣ et donc v aurait dû être choisi. □

Proposition 1.14

Soit T un tri préfixe de G. Soit L un parcours de G^t.
Si les points de régénération de L sont choisis prioritairement selon T, alors la partition associée à L est la décomposition en CFC de G^t et G.

Démonstration : Puisque G et G^t ont les mêmes CFC, on ne précisera donc pas toujours de quelles CFC on parle. On sait déjà que deux sommets de la même CFC sont nécessairement dans la même partie selon L (voir chapitre sur les parcours). Il reste à montrer que deux sommets dans la même partie selon L sont nécessairement de la même CFC, autrement dit qu'ils sont mutuellement accessibles l'un depuis l'autre. On reprend les notations de la définition du partitionnement associé à un parcours :

- K le nombre de points de régénération de L ;
- $(r_k)_{k\in[\![1,K]\!]} \in [\![1,n]\!]^K$ les indices de ces points de régénération, en ordre strictement croissant ;
- $r_{K+1} = n+1$;
- $\forall k \in [\![1,K]\!]$, $C_k = \{L_j \mid j \in [\![r_k, r_{k+1}[\![\}$.

Par l'absurde supposons qu'il existe $(u,v) \in S^2$ appartenant à la même partie C_k et tels que $u \not\sim v$. Puisque L est un parcours de G^t, on sait que u et v sont tous les deux accessibles depuis L_{r_k} dans G^t (par des arguments de bordure non vide déjà développés dans le chapitre sur les parcours).

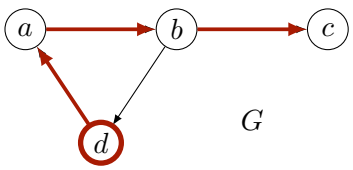
On en déduit que, dans G^t, L_{r_k} n'est pas accessible depuis u ou pas accessible depuis v (sinon en concaténant les chemins on aurait $u \sim v$). Quitte à échanger u et v, on suppose que L_{r_k} n'est pas accessible depuis u dans G^t. Ainsi u est un descendant propre de L_{r_k} dans G^t, autrement dit u est un ascendant propre de L_{r_k} dans G. Puisque T est un tri préfixe de G on en déduit $\mathsf{rg}_T(u) < \mathsf{rg}_T(L_{r_k})$.

Par ailleurs, en appliquant 1.13 à T et L (respectivement permutation des sommets et parcours du graphe G^t) : $\mathsf{rg}_T(L_{r_k}) \leqslant \mathsf{rg}_T(u)$.
Absurde. On en déduit que deux sommets appartenant à une même partie selon L sont nécessairement dans la même CFC. □

Remarque 1.15

Attention il faut bien passer au graphe transposé et le parcourir en choisissant les points de régénération selon un tri préfixe, rester sur le même graphe avec le miroir d'un tri préfixe ne suffit pas.

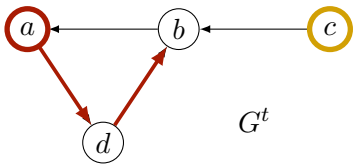
Démonstration : On considère le graphe G ci-dessous à gauche (on donne ci-dessous à droite G^t son graphe transposé). Notons $T = [a,b,c,d]$, ainsi T est un tri préfixe de G, et $\overleftarrow{T} = [d,c,b,a]$ son miroir.
Si on fait un parcours du graphe G en choisissant les points de régénération selon $\overleftarrow{T} = [d,c,b,a]$, on obtient le parcours $[\underline{d},a,b,c]$, et donc la partition $\{\{a,b,c,d\}\}$. Or ce n'est pas la décomposition en CFC puisque a et c par exemple sont dans la même partie, alors que $c \not\xrightarrow{*} a$. En effet, dans G on peut aller de b à c mais pas de c à b, mais on ne s'en rend pas compte ici car on visite b avant c ….



G

♣. par des arguments de bordure non vide déjà développés dans le chapitre sur les parcours

Si on fait un parcours du graphe G^t en choisissant les points de régénération selon le tri préfixe $T = [a, b, c, d]$, on obtient $\{\{a, d, b\}, \{c\}\}$. Cette décomposition est bien la décomposition en CFC. En effet ici, le fait qu'on ne puisse pas aller de c à b dans G, qui se traduit par l'impossibilité d'aller de b à c dans G^t, a forcé le parcours à prendre un nouveau point de régénération entre b et c.



□

1.3 Algorithme de Kosaraju

D'après la propriété précédente on peut décomposer un graphe orienté en CFC avec n'importe quel parcours pourvu qu'on choisisse les points de régénération (y compris le premier point du parcours) selon un tri préfixe. De plus on a vu précédemment qu'on peut construire un tri préfixe par un parcours en profondeur. L'algorithme de Kosaraju propose donc d'appliquer deux fois la même routine de parcours en profondeur : la première fois sur le graphe G avec des points de régénérations arbitraires afin d'obtenir T un tri préfixe de G, la seconde fois sur le graphe G^t en choisissant les points de régénération selon l'ordre établi par T. La partition associée à ce second parcours fournit alors la décomposition en CFC.

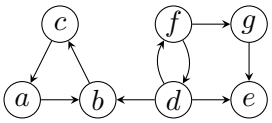
Algorithme 0 : Algorithme de Kosaraju

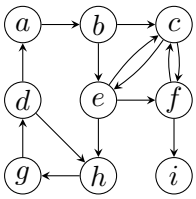
Entrée : Un graphe orienté $G = (S, A)$
Sortie : La décomposition de G en CFC

1 On calcule L un tri préfixe de G ;
2 On parcourt G^t, suivant l'ordre induit par L ;
3 On retourne le partitionnement associé à ce parcours ;

Exercice de cours 1.16

Appliquer l'algorithme de Kosaraju aux graphes ci-dessous.





Proposition 1.17

La complexité♣ *de l'algorithme de Kosaraju est en* $O(n + m)$.

Démonstration : On représente les graphes par tables de listes d'adjacence, ainsi les deux parcours sont en $O(n + m)$. Le calcul de G^t se fait aussi en $O(n + m)$ (*Cf.* exercice de cours 1.8). □

Exercice de cours 1.18

On décide d'améliorer l'algorithme de Kosaraju en profitant du premier parcours pour tester si le graphe est, ou non, sans circuit. Expliquer en quoi cette information est pertinente.

1.4 Application à 2-Sat

Cette section établit un corollaire de l'existence de l'algorithme de Kosaraju, et plus généralement de l'existence d'un algorithme de complexité polynomiale permettant le calcul des composantes

♣. Sous réserve que l'ensemble des sommets du graphe considéré soit de la forme $[\![1, n]\!]$

fortement connexes d'un graphe orienté. On rappelle que le problème de décision 2-SAT est défini comme suit.

2-SAT : $\begin{cases}\textbf{Entrée :} \text{ Une formule } H \in \mathbb{F}_p(\mathfrak{Q}) \text{ donnée comme conjonction de 2-clauses}\\ \textbf{Sortie :} \ H \text{ est-elle satisfiable ?}\end{cases}$

Corollaire 1.19

2-SAT $\in$ P.

Démonstration : Soit $\mathfrak{Q}$ un ensemble de variables propositionnelles. Soit $H = (l_{1,1} \vee l_{1,2}) \wedge \ldots \wedge (l_{t,1} \vee l_{t,2})$ une instance de 2-SAT. Dans la suite de cette preuve, lorsque l, est un littéral, on note l^c le littéral opposé.

On remarque que lorsqu'un littéral $l_{i,1}$ est interprété à F, $l_{i,2}$ doit nécessairement être interprété à V pour que H soit satisfaite (car la clause $(l_{i,1} \vee l_{i,2})$ doit en particulier l'être).

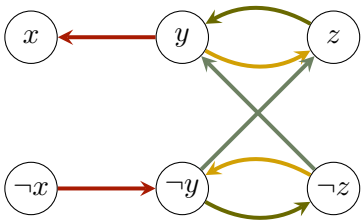
Ainsi si $l_{i,1}^c$ est interprété à V, $l_{i,2}$ doit aussi être interprété à V. De même $l_{i,2}^c$ est interprété à V, $l_{i,1}$ doit aussi être interprété à V.

On traduit cette dépendance au moyen d'un graphe orienté sur les littéraux. Plus précisément, on construit le graphe G_H de la manière suivante :

- l'ensemble de ses sommets est l'ensemble des littéraux sur $\mathfrak{Q}$, *i.e.* $S = \mathfrak{Q} \cup \{\neg p \mid p \in \mathfrak{Q}\}$;
- l'ensemble de ses arcs est $A = \{(l_{i,1}^c, l_{i,2}) \mid i \in [\![1,t]\!]\} \cup \{(l_{i,2}^c, l_{i,1}) \mid i \in [\![1,m]\!]\}$.

Exemple 1.20

Par exemple la formule $(x \vee \neg y) \wedge (\neg y \vee z) \wedge (y \vee \neg z) \wedge (y \vee z)$, le graphe associé est le suivant.



Le graphe G_H est construit de manière à assurer le lemme suivant.

Lemme 1.21

Soit $(u,v) \in S^2$. Si $u \overset{*}{\to} v$ dans G_H et si ρ est un modèle de H ♣ tel que $[\![u]\!]^\rho = V$ alors $[\![v]\!]^\rho = V$.

Démonstration : On le montre par récurrence sur la longueur du chemin menant d'un sommet à l'autre.

- Soit $(u,v) \in S^2$ tels que $u \overset{0}{\to} v$. Dans ce cas $u = v$, donc $[\![u]\!]^\rho = [\![v]\!]^\rho = \mathsf{V}$.
  Ainsi la propriété est vraie au rang 0.
- Soit $k \in \mathbb{N}$. Supposons la propriété est vraie au rang k. Soit $(u,v) \in S^2$ tels que $u \overset{k+1}{\to} v$. Soit ρ soit un modèle de H tel que $[\![u]\!]^\rho = \mathsf{V}$. Puisque $u \overset{k+1}{\to} v$, il existe $w \in S$ tel que $u \overset{k}{\to} w$ et $w \overset{1}{\to} v$. Comme $u \overset{k}{\to} w$ et $[\![u]\!]^\rho = \mathsf{V}$, on a aussi $[\![w]\!]^\rho = \mathsf{V}$ par hypothèse de récurrence. De plus comme $(w,v) \in A$, $w^c \vee v$ ou $v \vee w^c$ est une clause de H, et donc satisfaite par ρ. Ainsi $[\![w^c]\!]^\rho + [\![v]\!]^\rho = \mathsf{V}$, or $[\![w^c]\!]^\rho = \overline{[\![w]\!]^\rho} = \overline{\mathsf{V}} = \mathsf{F}$, on en déduit que $[\![v]\!]^\rho = \mathsf{V}$. La propriété est donc vraie au rang $k+1$.

□

Proposition 1.22

H est satisfiable si et seulement si aucune CFC de G_H ne contient à la fois une variable et sa négation.

Démonstration :

⇒ Si H est satisfiable, on dispose alors d'un environnement propositionnel $\rho \in \mathbb{B}^{\mathfrak{Q}}$ tel que $[\![H]\!]^\rho = \mathsf{V}$.
Supposons par l'absurde qu'il existe une CFC du graphe G_H contenant les littéraux x et $\neg x$.
Par définition d'une CFC, $x \overset{*}{\to} \neg x$ et $\neg x \overset{*}{\to} x$.

♣. Cela signifie que ρ est un environnement propositionnel tel que $[\![H]\!]^\rho = \mathsf{V}$

- Si $\rho(x) = \mathsf{V}$ alors $[\![x]\!]^\rho = \mathsf{V}$. Puisque $x \overset{*}{\to} \neg x$, on déduit du lemme 1.21 que $[\![\neg x]\!]^\rho = \mathsf{V}$ soit $\rho(x) = \mathsf{F}$.
- Si $\rho(x) = \mathsf{F}$ alors $[\![\neg x]\!]^\rho = \mathsf{V}$. Puisque $\neg x \overset{*}{\to} x$, on déduit du lemme 1.21 que $[\![x]\!]^\rho = \mathsf{V}$ soit $\rho(x) = \mathsf{V}$.

Les deux cas étant absurdes, aucune CFC de G_H ne contient une variable et sa négation.

$\Leftarrow$ Réciproquement supposons qu'aucune CFC de G_H ne contient une variable et sa négation.
Montrons que H est alors satisfiable en exhibant un modèle.
Notons $(C_1, C_2, \dots, C_s)$ un tri topologique du graphe réduit de G_H. On peut définir l'environnement $\rho : \mathcal{Q} \to \mathbb{B}$ par $\rho(x) = \mathsf{V}$ si et seulement si $i < j$ où $(i, j) \in [\![1, s]\!]^2$ sont tels que $\neg x \in C_i$ et $x \in C_j$.
Ainsi pour toute variable $x \in \mathcal{Q}$, si $[\![x]\!]^\rho = \mathsf{V}$ alors x est dans une CFC d'indice strictement supérieur à la CFC contenant $\neg x$. Étant donné qu'aucune variable n'apparaît dans la même CFC que sa négation, si $[\![x]\!]^\rho = \mathsf{F}$ alors x est dans une CFC d'indice strictement inférieur à la CFC contenant $\neg x$. Autrement dit, si $[\![\neg x]\!]^\rho = \mathsf{V}$ alors $\neg x$ est dans une CFC d'indice strictement supérieur à la CFC contenant x.
Ainsi pour tout littéral l, si $[\![l]\!]^\rho = \mathsf{V}$ alors l^c est dans une CFC d'indice strictement inférieur à la CFC contenant l.
Supposons alors par l'absurde qu'il existe une clause $l_{i,1} \vee l_{i,2}$ de H telle que $[\![l_{i,1} \vee l_{i,2}]\!]^\rho = \mathsf{F}$. Notons alors :

- k_1 l'indice de la CFC de $l_{i,1}$;
- k_1' l'indice de la CFC de $l_{i,1}^c$;
- k_2 l'indice de la CFC de $l_{i,2}$;
- k_2' l'indice de la CFC de $l_{i,2}^c$.

Puisque $[\![l_{i,1} \vee l_{i,2}]\!]^\rho = \mathsf{F}$, $[\![l_{i,1}]\!]^\rho = \mathsf{F}$ et $[\![l_{i,2}]\!]^\rho = \mathsf{F}$, soit $[\![l_{i,1}^c]\!]^\rho = \mathsf{V}$ et $[\![l_{i,2}^c]\!]^\rho = \mathsf{V}$, donc par construction de ρ :
- $l_{i,1} = (l_{i,1}^c)^c$ est dans une CFC d'indice strictement inférieur à la CFC contenant $l_{i,1}^c$, soit $k_1 < k_1'$;
- $l_{i,2} = (l_{i,2}^c)^c$ est dans une CFC d'indice strictement inférieur à la CFC contenant $l_{i,2}^c$, soit $k_2 < k_2'$.

Pourtant, puisque $l_{i,1} \vee l_{i,2}$ est une clause de H, le graphe G_H contient l'arc $(l_{i,1}^c, l_{i,2})$ et l'arc $(l_{i,2}^c, l_{i,1})$, qui impliquent respectivement que $k_1' \leqslant k_2$ et $k_2' \leqslant k_1$. Mises bout à bout ces quatre inégalités donnent : $k_1 < k_1' \leqslant k_2 < k_2' \leqslant a$. Absurde Finalement toute clause $l_{i,1} \vee l_{i,2}$ de H est satisfaite par ρ, donc ρ est bien modèle de H et H satisfiable.

□

L'algorithme suivant résout donc 2-Sat et il est de complexité polynomiale.

Algorithme 1 : Satisfiabilité d'une 2-CNF

Entrée : Une 2-CNF H
Sortie : H est-elle satisfiable ?

1 Construire le graphe G_H associé à la formule H ;
2 Calculer les CFC de G_H ;
3 Tester si aucune variable et sa négation ne sont dans la même CFC ;

□

Exercice de cours 1.23

Justifier plus précisément pourquoi cet algorithme est de complexité polynomiale.

Remarque 1.24

On remarque que la preuve précédente fournit en fait un algorithme permettant non seulement de tester si une 2-CNF est satisfiable, mais aussi d'en construire un modèle.

Exercice de cours 1.25

Construire un modèle de la formule de l'exemple 1.4 en suivant la construction de la preuve précédente (*i.e.* donner d'abord les CFC du graphe, puis donner un tri topologique du graphe réduit avant de donner fixer la valeur de vérité pour chaque variable).

Exercice de cours 1.26

Donner un modèle de $(\neg y \vee x) \wedge (y \vee \neg z) \wedge (y \vee \neg z) \wedge (y \vee z) \wedge (\neg y \vee z)$.

Remarque 1.27

On remarque que la résolution d'une instance H de 2-SAT dépend seulement de l'ensemble des paires de littéraux apparaissant dans la formule. En effet si on change l'ordre des clauses, leur multiplicité ou l'ordre de deux littéraux au sein d'une clause, on e change pas le graphe G_H associé à la formule H, ainsi on ne change pas la manière de décider si H est satisfiable (ni celle, le cas échéant, de trouver un modèle de H).

Exercice de cours 1.28

Appliquer l'algorithme précédent aux deux instances 2-SAT suivantes (représentées ici par l'ensemble de leurs clauses) $\Gamma_1 = \{\neg p \vee q, \neg q \vee r, \neg q \vee s, \neg s \vee \neg r\}$ et $\Gamma_2 = \Gamma_1 \cup \{p \vee q\}$.

2 Arbres couvrants de poids minimum

Dans toute cette section, on travaille sur un graphe non orienté pondéré $G = (S, A, c)$ où $c \in \mathcal{F}(A, \mathbb{N})$. On notera $n = \text{card}(S)$ et $m = \text{card}(A)$.

2.1 Arbres couvrants

Définition 2.1

*On rappelle que G est **acyclique** s'il n'existe aucun cycle élémentaire de longueur supérieure ou égale à 3. On rappelle que G est **connexe** si pour tout couple de sommets il existe une chaîne ayant ces deux sommets comme extrémités. G est un **arbre** si et seulement si G est connexe et acyclique.*

Proposition 2.2

Les 5 propositions ci-dessous sont équivalentes.
- *G est connexe et acyclique*
- *G est connexe et $|A| = |S| - 1$*
- *G est acyclique et $|A| = |S| - 1$*
- *G est minimal parmi les sous-graphes connexes de K_n* ♣
- *G est maximal parmi les sous-graphes acycliques de K_n*

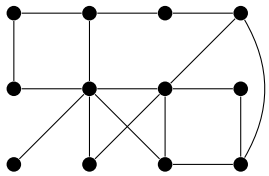
Démonstration : La preuve est un exercice de TD. □

Définition 2.3

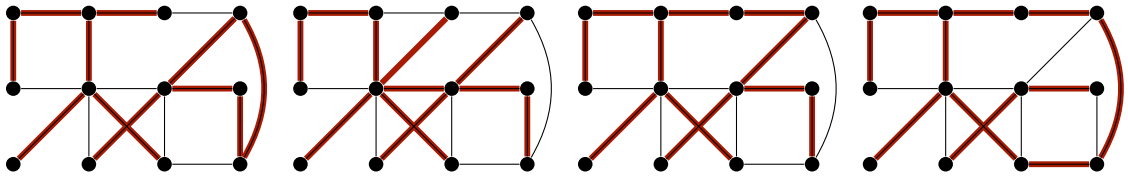
*On dit d'un graphe $G' = (S', A')$ que c'est un **arbre couvrant** de G dès lors que : $S' = S$, $A' \subseteq A$ et G' est un arbre.*

Exercice de cours 2.4

On considère le graphe G ci-dessous.

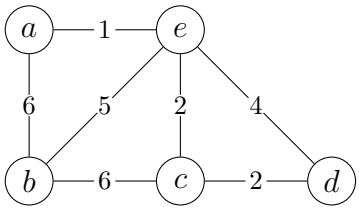


Parmi les graphes ci-dessous, désignés par les arêtes —, lesquels sont des arbres couvrants du graphe G, justifier.

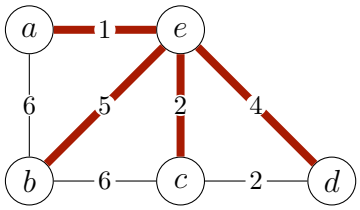


Définition 2.5

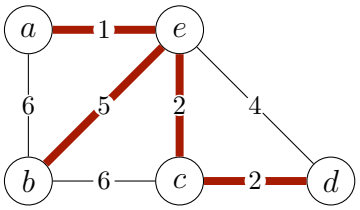
*Si $A' \subseteq A$, le **poids** de A' est $\sum_{\{x,y\}\in A'} c(x,y)$, et parfois noté $c(A')$. Si $T = (S', A')$ est un arbre, son **poids**, parfois noté $c(T)$ est $c(A')$. Autrement dit le poids d'un arbre est la somme des poids de ses arêtes.*



Exemple



Exemple avec arbre couvrant



Exemple avec arbre couvrant de poids minimum

Exercice de cours 2.6

Démontrer qu'un graphe admet un arbre couvrant si et seulement il est connexe.

Exercice de cours 2.7

Soit un graphe connexe G à n sommets et m arêtes. Justifier que le nombre d'arbres couvrants de G est fini en donnant, en fonction de n et m, une borne sur le nombre d'arbres couvrants.

Définition 2.8

Étant donné qu'il y a un nombre fini non nul d'arbres couvrants d'un graphe connexe, on peut alors considérer le problème d'optimisation suivant.

$$\textit{ACPM} : \begin{cases} \textbf{\textit{Entrée :}}\ \textit{Un graphe connexe non orienté pondéré } G = (S, A, c) \\ \textbf{\textit{Sortie :}}\ \textit{Un arbre couvrant de poids minimum} \end{cases}.$$

Définition 2.9

*On dit d'un graphe $G' = (S', A')$ que c'est une **forêt couvrante** de G dès lors que : $S' = S$, $A' \subseteq A$ et G' est acyclique et $\sim_G = \sim_{G'}$. Autrement dit, une forêt couvrante d'un graphe G est un sous-graphe de G acyclique qui a exactement les mêmes composantes connexes que G.*