Chapitre 5

Trois exemples d'algorithmes de graphes

Hugo Salou MPI*

Dernière mise à jour le 20 mars 2023

Table des matières

1 Composantes fortement connexes (CFC) 2

1.1 Rappels 4

1.2 Rangement particulier de graphe 5

1.3 Graphe transposé et CFC 7

1.4 Calcul de tri préfixe 8

1.5 Algorithme de Kosaraju 9

1.6 Applications 9

2 Arbres couvrants de poids minimum 11

3 Couplage dans un graphe biparti 15

Annexe A. Remarques supplémentaires 18

Dans ce chapitre, on s'intéresse aux graphes. Nous rappellerons les notions et algorithmes de graphes vus l'année dernière. On considère 3 exemples d'algorithmes de graphes.

Par exemple, l'année dernière, nous avons vu comment décomposer un graphe non-orienté en composantes connexes : on choisit un sommet au hasard, puis on parcours les voisins de ce sommets, et on répète. Mais, dans un graphe orienté, la notion de « composante connexe » n'est plus la même dans un graphe orienté. C'est l'algorithme décrit dans la section 1.

1 Composantes fortement connexes (CFC)

Dans la suite de cette section, $G = (S, A)$ est un graphe orienté.

Définition : On dit que $v \in S$ est *accessible* depuis $u \in S$ s'il existe un chemin de u à v, que l'on note $u \xrightarrow{\star} v$. De même, on dit que $v \in S$ est *co-accessible* depuis $u \in S$ s'il existe un chemin de v à u, que l'on note $v \xrightarrow{\star} u$.

Définition ($u \sim_G v$) : On note $u \sim_G v$ si $u \xrightarrow{\star} v$ et $v \xrightarrow{\star} u$.

Remarque :
La relation $\sim_G$ est une relation d'équivalence.

Digression L'année dernière, dans un graphe non orienté, on peut noter $\sim$ la relation d'équivalence induite par, si $\{u, v\} \in A$, alors $u \sim v$. Dans ce cas, le même chemin permet d'aller de u à v, puis de v à u, et ce chemin est le même. Mais, dans un graphe orienté, si $u \sim_G v$, les chemins de u à v puis de v à u ne sont pas forcément les mêmes.

Définition (CFC) : On appelle *composantes fortement connexes* (CFC) d'un graphe G, les classes d'équivalences de la relation $\sim_G$.

Définition : On dit d'un graphe ayant une unique composante fortement connexe qu'il est *fortement connexe*.

Exemple :
Le graphe ci-dessous a deux composantes fortement connexes, il n'est donc pas fortement connexe.

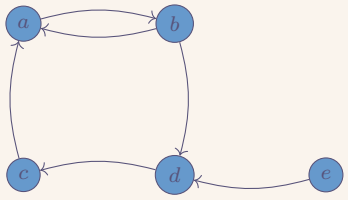

Figure 1 – Graphe non fortement connexe

Définition : On appelle *ensemble fortement connexe* un ensemble $V \subseteq S$ tel que G_V est fortement connexe où G_V est le *graphe induit* par V : $G_V = (V, A \cap V^2)$.

Exemple :
Avec le graphe précédent, l'ensemble $\{a, b\}$ est un ensemble fortement connexe.

Lemme : Si W est une composante fortement connexe de G, alors W est un ensemble fortement connexe.

Preuve :
Soit W une composante fortement connexe. On doit montrer que W est un ensemble fortement connexe, *i.e.* le graphe G_W est fortement connexe, *i.e.* pour tout couple de sommets $(u, v) \in W^2$, $u \sim_{G_W} v$. Étant donné que W est une composante connexe, $u \sim_G v$, donc il existe $(n, m) \in \mathbb{N}^2$ et deux chemins

$$u \to v_1 \to v_2 \to \cdots \to v_n = v \quad \text{et} \quad v \to u_1 \to u_2 \to \cdots \to u_m = u.$$

Soit alors $i \in [\![1, n]\!]$. On a donc deux chemins

$$v \to u_1 \to u_2 \to \cdots \to u_m = u \to v_1 \to \cdots \to v_i$$

et

$$v_i \to v_{i+1} \to \cdots \to v_n = v.$$

Ainsi, $v \sim_G v_i$ et $v_i \in W$. De même, pour tout $j \in [\![1, m]\!]$, $u_j \in W$. Ainsi, $u \sim_{G_w} v$ (en utilisant les mêmes chemins). Donc, G_W est fortement connexe puisque toute paire de sommets est équivalent) □

Propriété : Les composantes fortement connexes sont les ensembles fortement connexes qui sont maximaux pour l'inclusion.

Preuve : " $\Longleftarrow$ " Soit V un ensemble fortement connexe maximal pour l'inclusion.
Remarque : on a $V \neq \varnothing$. En effet, un singleton est un ensemble fortement connexe, et donc V ne sera pas maximal pour l'inclusion.
Soit $(u, v) \in V^2$. Par définition de fortement connexe, $u \sim_{G_V} v$ donc $u \sim_G v$. Soit alors W la classe des éléments de $V \subseteq W$. L'ensemble W est fortement connexe (d'après le lemme précédent). Ainsi, $W = V$, W est donc une composante fortement connexe.

" $\Longrightarrow$ " Soit V une composante fortement connexe. L'ensemble V est fortement connexe, d'après le lemme précédent. Soit $W \supseteq V$ tel que W est fortement connexe. Montrons que $V = W$.

- Si $W \setminus V = \varnothing$, alors OK.
- Si $W \setminus V \neq \varnothing$, alors soit $z \in W \setminus V$. L'ensemble W est fortement connexe. Soit $u \in V$. On a $u \sim_{G_W} z$, donc $u \sim_G z$, donc $z \in V$ ce qui est absurde.

□

Définition : On appelle *graphe réduit* de G le graphe orienté $\hat{G} = (\hat{S}, \hat{A})$ où

$$\hat{S} = \{\bar{x} \mid x \in S\} \quad \text{et} \quad \hat{A} = \big\{(\bar{x}, \bar{y}) \mid (x, y) \in A \text{ et } \bar{x} \neq \bar{y}\big\}.$$

Exemple :
À faire : Figure

Remarque : — $\hat{G}$ est acyclique.

— Pour tout couple $(\bar{x}, \bar{y}) \in \hat{S}^2$, si $\bar{x} \xrightarrow{\star}_{\hat{G}} \bar{y}$, alors $\forall u \in \bar{x}, \forall v \in \bar{y}, x \xrightarrow{\star}_{G} v$.

1.1 Rappels

Définition : La *bordure* d'un ensemble de sommets $V \subseteq S$, noté $\mathscr{B}(V)$, est l'ensemble des successeurs de V non dans V :

$$\mathscr{B}(V) = \{s \in S \setminus V \mid \exists u \in V,\ (u, s) \in A\}.$$

Définition (parcours) **:** Un *parcours* est une permutation des sommets $(L_1, L_2, \dots, L_n)$ telle que, pour $i \in [\![1, n-1]\!]$,

$$L_i \in \mathscr{B}(\{L_1, \dots, L_{i-1}\}) \quad \text{ou} \quad \mathscr{B}(\{L_1, \dots, L_{i-1}\}) = \varnothing.$$

On dit d'un L_i avec $i \in [\![1, n]\!]$ tel que $\mathscr{B}(\{L_1, \dots, L_{i-1}\}) = \varnothing$, que c'est un *point de régénération* du parcours.

Lemme : Si $V \subseteq S$ est tel que $\mathscr{B}(V) = \varnothing$, il n'existe aucun chemin d'un sommet de V à un chemin de $S \setminus V$.

Preuve (par l'absurde) *:*
Soit un chemin $V \ni u_0 \to u_1 \to \cdots \to u_k \in S \setminus V$ avec $k \in \mathbb{N}$. Soit $I = \{i \in [\![0, k]\!] \mid u_i \in S \setminus V\}$. L'ensemble I est non vide, car $u_k \in S \setminus V$, et $I \subseteq N$. Il admet donc un plus petit élément ; nommons le $i_0 \in [\![0, k]\!]$. On a $i_0 \neq 0$ car $u_0 \in V$. Ainsi, $u_{i_0 - 1} \in V$, $u_{i_0} \in S \setminus V$, et $(u_{i_0 - 1}, u_{i_0}) \in A$. Donc, $u_{i_0} \in \mathscr{B}(V)$, ce qui est absurde. □

Définition : Soit $(L_1, L_2, \dots, L_n)$ un parcours de G. On note K le nombre de ses points de régénération. On note $(r_k)_{k \in [\![1,k]\!]}$ l'extractrice des points de régénération. En notant de plus $r_{K+1} = n + 1$. Le *partitionnement associé au parcours* est alors

$$\left\{ \{L_{r_i}, L_{r_i+1}, \dots, L_{r_{i+1}-1}\} \,\middle|\, i \in [\![1, K]\!] \right\}..$$

Exemple :
Si $n = 10$ et les points de régénération sont d'indices 1, 4, 7 et 8, alors le partitionnement associé est donc

$$\Big\{ \{1, 2, 3\}, \{4, 5, 6\}, \{7\}, \{8, 9, 10\} \Big\}.$$

Définition : Un partitionnement P_1 est un *raffinement* d'un partitionnement P_2 dès lors que

$$\forall C_1 \in P_1,\ \exists C_2 \in P_2,\ C_1 \subseteq C_2.$$

Propriété : Les composantes fortement connexes sont un raffinement des partitionnement des parcours.

Preuve :
Soient u et v deux sommets de la même composante fortement connexes. Supposons que u et v ne sont pas dans la même partie du partitionnement pour un parcours $(L_1, \dots, L_n)$ du graphe. Soit i_u et i_v tels que $u = L_{i_u}$ et $v = L_{i_v}$. Sans perdre en généralité, on peut supposer $i_u \leqslant i_v$. Il existe alors $i_0 \in]\!]i_u, i_v]\!]$ tels que L_{i_0} est un point de régénération. Alors,

$$\mathscr{B}(\{L_1, L_2, \dots, L_{i_0-1}\}) = \varnothing.$$

D'après le lemme précédent, il n'existe pas de chemin de u à v, ce qui est absurde car u et v sont dans la même composante fortement connexe. □

1.2 Rangement particulier de graphe

Définition (Sommet ouvert) **:** Soit $(L_1, \dots, L_n)$ un parcours de G. Pour $k \in [\![1, n]\!]$ et $i \in [\![1, k[\![$, on dit que L_i est *ouvert* à l'étape k si

$$\mathrm{Succ}(L_i) \not\subseteq \{L_j \mid j \in [\![1, k[\![\}$$

où $\mathrm{Succ}(L_i)$ est l'ensemble des successeurs de L_i.

Cette définition nous permet de définir les parcours en largeur et en profondeur.

Définition (parcours en largeur) **:** Soit $(L_1, \dots, L_n)$ un parcours. Il est dit *en largeur* si chaque sommet du parcours qui n'est pas un point de régénération est un successeur du premier sommet ouvert à cette étape :

$$\forall k \in [\![2, n]\!], \qquad \mathscr{B}(\{L_j \mid j \in [\![1, k[\![\}) = \varnothing \quad \text{ou} \quad L_k \in \mathrm{Succ}(L_{i_0})$$

avec $i_0 = \min\{i \in [\![1, k[\![\mid L_i \text{ ouvert à l'étape } k\}$.

Définition (parcours en profondeur) **:** Soit $(L_1, \dots, L_n)$ un parcours. Il est dit *en largeur* si chaque sommet du parcours qui n'est pas un point de régénération est un successeur du dernier sommet ouvert à cette étape :

$$\forall k \in [\![2, n]\!], \qquad \mathscr{B}(\{L_j \mid j \in [\![1, k[\![\}) = \varnothing \quad \text{ou} \quad L_k \in \mathrm{Succ}(L_{i_0})$$

avec $i_0 = \max\{i \in [\![1, k[\![\mid L_i \text{ ouvert à l'étape } k\}$.

Exemple :
Dans le graphe ci-dessous, un parcours en largeur est

$$\underline{a} \to c \to b \to d \to f \rightsquigarrow \underline{e}.$$

Les sommets soulignées sont les points de régénération. On peut remarquer qu'il n'y a pas unicité du parcours en largeur, on aurait pu commencer le parcours par $\underline{a} \to b \to c \to \cdots$, et ce parcours est aussi un parcours en largeur.

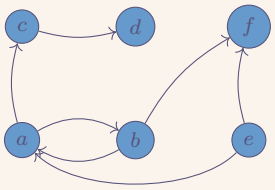

FIGURE 2 – Exemple de graphe orienté – parcours en largeur

Sur le même graphe, un parcours en profondeur est

$$\underline{a} \to b \to f \to c \to d \rightsquigarrow \underline{e}.$$

De même, il n'y a pas unicité du parcours en profondeur.

Définition (tri topologique) **:** Soit $(T_i)_{i\in[\![1,n]\!]}$ une permutation des sommets. On dit que T est un *tri topologique* si

$$\forall (i,j) \in [\![1,n]\!]^2, \qquad \text{si } (T_i, T_j) \in A \text{ alors } i \leqslant j.$$

EXEMPLE :
Un tri topologique du graphe ci-dessous est la permutation indiquée dans le graphe.

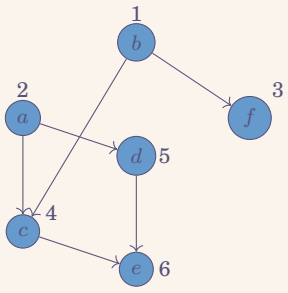

FIGURE 3 – Tri topologique d'un graphe

Définition : Soit une permutation de sommets $(T_i)_{i\in[\![1,n]\!]}$. On appelle *rang* de $u \in S$ dans T le plus petit indice dans T d'un élément accessible ($u \xrightarrow{\star} v$) et co-accessible ($v \xrightarrow{\star} u$) depuis u. On définit

$$\mathrm{rang}_T(u) = \min\{i \in [\![1,n]\!] \mid T_i \sim_G u\}.$$

Définition : Étant donné une permutation $(T_i)_{i\in[\![1,n]\!]}$ des sommets de G, on définit la relation

$$\preccurlyeq_T = \{(u,v) \in S^2 \mid \mathrm{rang}(u) \leqslant \mathrm{rang}(v)\}.$$

On dit alors que T est un *tri préfixe* dès lors que, pour tout $(u,v) \in S^2$, si $u \xrightarrow{\star} v$ alors $u \preccurlyeq_T v$.

Remarque :
Étant donné une permutation T, pour tout couple de sommets (u, v),

$$u \sim_G v \iff (u \preccurlyeq_T v \text{ et } v \preccurlyeq_T u).$$

1.3 Graphe transposé et cfc

Définition : Étant donné un graphe $G = (S, A)$, on appelle *graphe transposé* de G, que l'on note $G^\top$, le graphe

$$G^\top = \Big(S, \big\{(y, x) \in S^2 \mid (x, y) \in A\big\}\Big).$$

Propriété : Soit T un tri préfixe de G. Soit L un parcours de $G^\top$ utilisant l'ordre des points de régénération induit par T. Alors, la partition associée à L est la décomposition en composantes forment connexes.

Preuve :
Soit T un tri préfixe du graphe G. Soit L un parcours de $G^\top$. Montrons que le partitionnement associé à L est la décomposition en composantes fortement connexes. Il suffit de montrer que, si u et v sont dans la même partition du parcours L de $G^\top$: $u \sim_G v$.

Remarque : les composantes fortement connexes de G et $G^\top$ sont les mêmes.

Soient u et v deux sommets dans la même partition de L. C'est donc qu'il existe un point de régénération L_{r_k} tel que $L_{r_k} \xrightarrow{\star}_{G^\top} u$ et $L_{r_k} \xrightarrow{\star}_{G^\top} v$. Ainsi, $L_{r_k} \xleftarrow{\star}_{G} u$ et $L_{r_k} \xleftarrow{\star}_{G} v$. Alors, $\mathrm{rang}_T(u) \leqslant \mathrm{rang}_T(L_{r_k})$.

Supposons que $L_{r_k} \not\xrightarrow{\star}_{G} u$. Alors, $\mathrm{rang}_T(u) \neq \mathrm{rang}_T(L_{r_k})$. D'où, il existe $w \sim_G u$ apparaissant avant (strictement) L_{r_k} dans T. Ce qui est absurde car on aurait dû visiter w avant.

On en déduit que $L_{r_k} \xrightarrow{\star}_{G} u$, et $v \xrightarrow{\star}_{G} u$, De même pour v, on a $L_{r_k} \xrightarrow{\star}_{G} v$ et $u \xrightarrow{\star}_{G} v$. Finalement, on a

$$u \sim_G v.$$

□

Exemple :

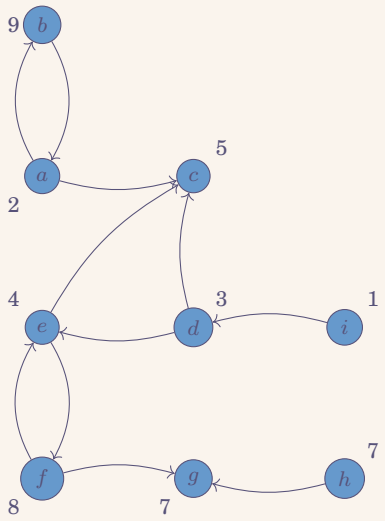

FIGURE 4 – Exemple de tri préfixe

Un tri préfixe dans le graphe ci-dessous est $g \leftarrow f \leftarrow c \leftarrow e \leftarrow d \rightsquigarrow i \rightsquigarrow b \leftarrow a \rightsquigarrow h$. Il s'agit d'un parcours en profondeur.

1.4 Calcul de tri préfixe

On peut donc donner un algorithme calculant un tri préfixe. On donne cet algorithme en impératif, même si la version récursive est plus simple.

Algorithme 1 Calcul d'un tri préfixe

Entrée Un graphe $G = (S, A)$.
Sortie Un tri préfixe des sommets de G.

```
1:  Procédure ExploreDescendants(s, Visités, Res)
2:      Entrée Un graphe G = (S, A), Res, Visités, s ∈ S.
3:      Sortie Modifie Res et Visités de sorte que Res soit un parcours préfixe de Visités et des
        sommets accessibles depuis s.
4:      todo ← pileVide
5:      empiler((s, Succ(s)), todo)
6:      Visités ← {s} ∪ Visités
7:      tant que todo ≠ pileVide faire
8:          (x, ℓ) ← depiler(todo)
9:          si ℓ = ( ) alors
10:             Res ← x · Res
11:         sinon
12:             t · ℓ' ← ℓ        ▷ on sépare la tête t du reste ℓ' de la pile ℓ.
13:             empiler((x, ℓ'), todo)
14:             si t ∉ Visités alors
15:                 Visités ← {t} ∪ Visités
16:                 empiler((t, Succ(t)), todo)

17: Visités ← ∅
18: Res ← ( )
19: tant que S \ Visités ≠ ∅ faire
20:     s ← un sommet de S \ Visités
21:     ExploreDescendants(s, Visités, Res)
22: retourner Res
```

On montre la correction de cet algorithme. On cherche des invariants *intéressants*, que l'on ne prouvera pas. Pour la boucle "tant que," dans la procédure ExploreDescendants, on choisit les invariants

1. pour tout couple de sommets $(u, v) \in S^2$, si $\mathscr{C}\ell(u) \subseteq \mathrm{Res}$ et que $u \xrightarrow{\star} v$, alors $\mathscr{C}\ell(v) \subseteq \mathrm{Res}$ et $\mathrm{rand}_{\mathrm{Res}}(u) \leqslant \mathrm{rang}_{\mathrm{Res}}(v)$,
2. $K(\mathrm{todo}) \cup \mathrm{Res} = \mathrm{Visités}$, où où $K(\mathrm{todo})$ est l'ensemble des premières composantes des couples de todo,
3. les clés de todo, du fond de la pile au sommet forment un chemin,
4. si u est un élément de $\mathrm{Visités}$, et v est un descendant de u,
   - ou bien $v \in \mathrm{Res}$,
   - ou bien $v \in K(\mathrm{todo})$
   - ou bien v est un descendant d'un élément d'une liste adjointe à un élément $w \in K(\mathrm{todo})$ tel que $u \xrightarrow{\star} w$.

On admet que ces 4 propriétés sont invariantes. À la fin, $\forall x \in \mathrm{Res}$, $\mathscr{C}\ell(x) \subseteq \mathrm{Res}$, et dnc l'invariant 1 assure alors que nous avons un tri préfixe.

1.5 Algorithme de Kosaraju

Algorithme 2 Algorithme de Kosaraju

Entrée Un graphe $G = (S, A)$
Sortie Les composantes fortement connexes de G
1: On calcule un tri préfixe de G.
2: On parcours $G^\top$ en utilisant l'ordre T comme points de régénération.
3: On retourne le plus petit partitionnement associé au parcours.

1.6 Applications

Théorème : 2-cnf-sat $\in \mathbf{P}$.

Exemple :
On considère la formule $H = (x \vee \neg y) \wedge (\neg y \vee z) \wedge (y \vee \neg z) \wedge (y \vee z)$. Elle est équivalente à

$$\begin{aligned} H' = \quad & (\neg x \to \neg y) \\ & \wedge (y \to x) \\ & \wedge (y \to z) \\ & \wedge (\neg z \to \neg y) \\ & \wedge (z \to y) \\ & \wedge (\neg y \to \neg z) \\ & \wedge (\neg y \to z) \\ & \wedge (\neg z \to y) \end{aligned}$$

En replaçant les $\to$ par des arrêtes dans un un graphe, on obtient celui représenté ci-dessous.

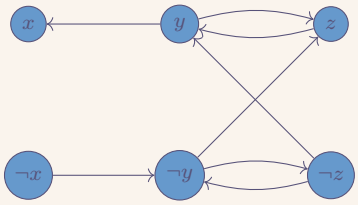

FIGURE 5 – Représentation d'une formule 2-CNF-SAT par un graphe

Preuve :
Soit $H \in$ 2CNF. On pose

$$H = (\ell_{1,1} \vee \ell_{1,2}) \wedge \ldots \wedge (\ell_{n,1} \vee \ell_{n,2}).$$

Dans la suite, on note $\ell^c_{i,j}$ le littéral opposé à $\ell_{i,j}$. À la formule H, nous associons le graphe G_H défini comme suit :

$$S_H = \{(\ell_{i,j}) \mid i \in [\![1,n]\!], j \in \{1,2\}\} \cup \{(\ell^c_{i,j}) \mid i \in [\![1,n]\!], j \in \{1,2\}\},$$
$$A_H = \{(\ell^c_{i,1}, \ell_{i,2} \mid i \in [\![1,n]\!]\} \cup \{(\ell^c_{i,2}, \ell_{i,1}) \mid i \in [\![1,n]\!]\}.$$

Lemme : Si ρ est un modèle de H et $u \xrightarrow{\star} v$ tel que $[\![u]\!]^\rho = \boldsymbol{V}$, alors $[\![v]\!]^\rho = \boldsymbol{V}$.

Preuve :
Soit ρ un modèle de H. Montrons par récurrence P_n : "si $u \xrightarrow{\star} v$ par un chemin de longueur n et $[\![u]\!]^\rho = \boldsymbol{V}$, alors $[\![v]\!]^\rho = \boldsymbol{V}$."

— P_0 : $u = v$, donc OK.

— P_{n+1} : supposons P_n vraie pour $n \in \mathbb{N}$. Soient u et v tels que

$$u \to u_1 \to u_2 \to \cdots \to u_n \to u_{n+1} = v$$

et $[\![u_n]\!]^\rho = \boldsymbol{V}$. D'après P_n, $[\![u_n]\!]^\rho = \boldsymbol{V}$. Or, $(u_n, u_{n+1}) \in A_H$. C'est donc que $u_n^c \vee u_{n+1} \in H$. Or, $[\![u_n]\!]^\rho = \boldsymbol{V}$, donc $[\![u_n^c]\!]^\rho = \boldsymbol{F}$. Or, $[\![u_n \vee v]\!]^\rho = \boldsymbol{V}$ et donc $[\![v]\!]^\rho = \boldsymbol{V}$.

On conclut par récurrence. □

Propriété : H est satisfiable si, et seulement si aucune variable et sa négation ne se trouvent dans la même CFC de G_H.

Preuve " $\Longrightarrow$ " Par contraposée, soit x et $\neg x$ se trouvant dans la même CFC de G_H. On procède par l'absurde. Soit ρ un environnement propositionnel tel que $[\![H]\!]^\rho = \boldsymbol{V}$.

— Si $\rho(x) = \boldsymbol{V}$, alors $[\![x]\!]^\rho = \boldsymbol{V}$. Or, $x \xrightarrow{\star} \neg x$. Alors, d'après le lemme, $[\![\neg x]\!]^\rho = \boldsymbol{V}$ et donc $\rho(x) = \boldsymbol{F}$, absurde.

— Si $\rho(x) = \boldsymbol{F}$, alors $[\![\neg x]\!]^\rho$. Or, $\neg x \xrightarrow{\star} x$ et donc, d'après le lemme, $[\![x]\!]^\rho = \boldsymbol{V}$ d'où $\rho(x) = \boldsymbol{V}$, absurde.

Il n'existe donc pas un tel ρ.

" $\Longleftarrow$ " Si G_H est telle qu'aucune variable et sa négation soient dans la même CFC. Soit $x \in \mathbb{Q}$ une variable propositionnelle. Soit $C_1, \ldots, C_p$ les composantes fortement connexes du graphe, triées par ordre topologique *i.e.* pour $(i,j) \in [\![1,p]\!]^2$, si $j > i$, alors il n'y a pas de chemin d'un élément de C_j vers un élément de C_i. Regardons alors où sont rangés x et $\neg x$. Si $x \in C_i$ et $\neg x \in C_j$ avec $i < j$, on définit alors $\rho(x) = \boldsymbol{F}$. Sinon, on définit $\rho(x) = \boldsymbol{V}$. Montrons alors que ρ est un modèle de H : $[\![H]\!]^\rho = \boldsymbol{V}$. Par l'absurde, soit $\ell_{i,1} \vee \ell_{i,2}$ une 2-clause de H telle que $[\![\ell_{i,1} \vee \ell_{i,2}]\!]^\rho = \boldsymbol{F}$, donc $[\![\ell_{i,1}]\!]^\rho = \boldsymbol{F}$ et $[\![\ell_{i,2}]\!]^\rho = \boldsymbol{F}$. Alors, $(\ell^c_{i,2}, \ell_{i,1})$ est une arrête de G_H et, $(\ell^c_{i,1}, \ell_{i,2})$ est une arrête de G_H. Ainsi, en notant a l'indice de la composante $\ell^c_{i,2}$, b l'indice de $\ell_{i,1}$, c l'indice de $\ell^c_{i,1}$ et d l'indice de $\ell_{i,2}$, on a $a \leqslant b$, $b < c$ par définition de ρ, $c \leqslant d$ et $d < a$ par définition de ρ, d'où

$$a \leqslant b < c \leqslant d < a,$$

ce qui est absurde. On a donc pour toute 2-clause $\ell_{i,1} \vee \ell_{i,2}$ de H, $[\![\ell_{i,1} \vee \ell_{i,2}]\!]^\rho = \boldsymbol{V}$ et donc $[\![H]\!]^\rho = \boldsymbol{V}$.

□

Algorithme 3 Solution au problème 2CNFSAT

Entrée H une 2-CNF
Sortie ρ un modèle de H ou None si H n'est pas satisfiable

1: On construit G_H
2: On construit les CFC $C_1, \dots, C_p$ de G_H (dans un ordre topologique)
3: **si** il existe x et $i \in [\![1, p]\!]$ tel que $x \in C_i$ et $\neg x \in C_i$ **alors**
4: | **retourner** None
5: **sinon**
6: | **retourner** ρ défini comme

$$\rho : \mathbb{Q} \longrightarrow \mathbb{B}$$
$$x \longmapsto \begin{cases} \boldsymbol{F} & \text{si } i < j \\ \boldsymbol{V} & \text{sinon} \end{cases}$$

| où $x \in C_i$ et $\neg x \in C_j$.

□

2 Arbres couvrants de poids minimum

EXEMPLE :
On considère le graphe ci-dessous.

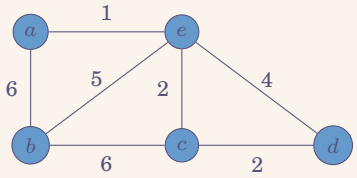

FIGURE 6 – Arbre pondéré

On cherche à « supprimer » des arrêtes de ce graphe afin d'avoir un poids total minimum, tout en conservant la connexité du graphe. Une structure assurant cette condition est un arbre.

Pour résoudre ce problème, on part du graphe vide, et on ajoute les arrêtes les moins coûteuses en premier.

Définition (Arbre) : Soit $G = (S, A)$ un graphe non-orienté. On dit que G est un *arbre* si G est connexe et acyclique.

Définition (Arbre couvrant) : Étant donné un graphe non orienté pondéré par poids positifs $G = (S, A, c)$,[1] on dit de $G' = (S', A')$ que c'est un *arbre couvrant* de G si $S' = S$ et $A' \subseteq A$, et G' est un arbre.

Définition (Arbre couvrant de poids minimum) : Étant donné un graphe non orienté pondéré $G = (S, A, c)$ et un arbre couvrant $T = (S', A')$, on appelle *poids* de l'arbre T la

1. on dit que c est la fonction de pondération de ce graphe

valeur $\sum_{a \in A'} c(a)$.

Si G est connexe, il admet au moins un arbre couvrant, on peut définir l'*arbre couvrant de poids minimum* (ACPM).

On définir alors le problème

$$\text{ACPM}^2 \begin{cases} \textbf{Entrée} & : G = (S, A, c) \text{ connexe} \\ \textbf{Sortie} & : \text{le poids de l'arbre couvrant de poids minimum.} \end{cases}$$

Algorithme 4 Algorithme de KRUSKAL

Entrée $G = (S, A, c)$ un graphe connexe
Sortie Un arbre couvrant de poids minimum

$B \leftarrow \varnothing$
$U \leftarrow \varnothing$
tant que il existe u et v tels que $u \not\sim_B v$ **faire**
  Soit $\{x, y\} \in A \setminus U$ de poids minimal
  si $x \sim_B y$ **alors**
    $U \leftarrow \{\{x, y\}\} \cup U$
  sinon
    $U \leftarrow \{\{x, y\}\} \cup U$
    $B \leftarrow \{\{x, y\}\} \cup B$
retourner $T = (S, B)$

Propriété : L'algorithme de KRUSKAL est correct.

Preuve :
1. Il existe un arbre couvrant de poids minimum utilisant les arrêtes de B ;
2. $B \subseteq U \subseteq A$;
3. $\forall \{u, v\} \in U,\ u \sim_B v$.

Ces trois propriétés sont invariantes.

Initialement $B = \varnothing = U$, donc OK.

Propagation Soient $\underline{B}$ et $\underline{U}$ (resp. $\bar{B}, \bar{U}$) les valeurs de B et U avant (resp. après) une itération de boucle. Supposons que $\underline{B}$ et $\underline{U}$ satisfont les propriétés 1, 2 et 3. Montrons que $\bar{B}$ et $\bar{U}$ les satisfont aussi.

2. On a $\{x, y\} \in A$ et $\underline{B} \subseteq \underline{U} \subseteq A$, donc
$$\bar{B} \subseteq \underline{B} \cup \{\{x, y\}\} \subseteq \underline{U} \cup \{\{x, y\}\} \subseteq A.$$
3. Soit $\{u, v\} \in \bar{U}$.
   — Si $\{u, v\} \in \underline{U}$, alors de 3, $u \sim_{\underline{B}} v$. Or, $\underline{B} \subseteq \bar{B}$ et donc $u \sim_{\bar{B}} v$.
   — Sinon, $\{u, v\} = \{x, y\}$, alors $x = u$ et $v = y$.
      — Sous-cas 1 : $\bar{B} = \underline{B} \cup \{\{x, y\}\}$, alors $x \sim_{\bar{B}} y$.
      — Sous-cas 2 : $\bar{B} = \underline{B}$, alors par condition du **si**, $x \sim_{\underline{B}} y$ et donc $x \sim_{\bar{B}} y$.
1. Soit $\mathcal{T}$ un ACPM contenant $\underline{B}$.
   — Cas 1 : $\bar{B} = \underline{B}$, OK
   — Cas 2 : $\bar{B} = \underline{B} \cup \{\{x, y\}\}$.
      — Sous-cas 1 : $\{x, y\} \in \mathcal{T}$, alors $\mathcal{T}$ est un ACPM qui contient $\bar{B}$.
      — Sous-cas 2 : $\{x, y\} \notin \mathcal{T}$, $\mathcal{T}$ est un arbre couvrant, donc il contient une chaîne de x à y :
$$\overset{x}{\overset{\|}{x_0}}, \{x_1, x_2\}, \ldots, \{x_{n-1}, \overset{y}{\overset{\|}{x_n}}\}.$$
Or, $\forall i \in [\![1, n-1]\!],\ x_i \sim_{\underline{B}} x_{i+1}$. Par transitivité, on a donc $x = x_0 \sim_{\underline{B}} x_n = y$, ce qui n'est pas le cas. Il existe donc $i_0 \in [\![0, n-1]\!]$, tel que $x_{i_0} \not\sim_{\underline{B}} x_{i_0+1}$ et donc $\{x_{i_0}, x_{i_0+1}\} \notin \underline{U}$. D'où, d'après 3, on a $\{x_{i_0}, x_{i_0+1}\} \notin \underline{B}$ Considérons

2. Arbre Couvrant de Poids Minimum

alors $\mathcal{T}' = (\mathcal{T} \setminus \{\{x_{i_0}, x_{i_0+1}\}\}) \cup \{\{x, y\}\}$. Montrons que $\mathcal{T}'$ est un ACPM contenant B, en commençant par montrer que c'est un arbre couvrant. L'arbre $\mathcal{T}'$ a $n-1$ arrêtes (autant que $\mathcal{T}$). Montrons que $\mathcal{T}'$ est connexe. Soit $(a, b) \in S^2$. $\mathcal{T}$ est connexe, soit donc une chaîne

$$C : \quad a = u_0, u_1, \dots, u_n = b$$

de $\mathcal{T}$. Si la chaîne C n'utilise pas l'arrête $\{x_{i_0}, x_{i_0+1}\}$, alors C est une chaîne de $\mathcal{T}'$. Sinon, on pose μ et τ tels que

$$\underbrace{a, \dots, x_{i_0}}_{\mu}, \underbrace{x_{i_0+1}, \dots, b}_{\tau}.$$

Soit alors la chaîne

$$\overbrace{a, \dots, x_{i_0}}^{\mu}, x_{i_0-1}, x_{i_0-2}, \dots, x_0 = x,$$
$$\underbrace{b, \dots, x_{i_0+1}}_{\tau}, x_{i_0+2}, \dots, x_{n-1}, x_n = y$$

qui est dans $\mathcal{T}'$. Montrons que le poids est minimum. Notons $P(\mathcal{T})$ le poids de l'arbre. On a donc

$$P(\mathcal{T}') = P(\mathcal{T}) + c(\{x, y\}) - c(\{x_{i_0}, x_{i_0+1}\}).$$

Par choix glouton, $(\{x_{i_0}, x_{i_0+1} \notin U\}), c(\{x, y\}) \leqslant c(\{x_{i_0}, x_{i_0+1}\})$ donc $P(\mathcal{T}') \leqslant P(\mathcal{T})$, et $\mathcal{T}$ étant de poids min, $P(\mathcal{T}') = P(\mathcal{T})$ et $\mathcal{T}'$ est un ACPM contenant B.

Les invariants le sont. □

À la fin, B induit un graphe connexe et B est contenu dans un ACPM, c'en est donc un.

Une structure pour la gestion des partitions : UnionFind.

Définition (Type de données abstrait UnionFind) **:** On définit le type de données abstrait UnionFind comme contenant

- un type `t` de partitions ;
- un type `elem` des éléments manipulés par les partitions ;
- `initialise_partition` : `elem list` $\to$ `t` retournant le partitionnement dans lequel chaque élément est seul dans sa classe ;
- `find` : (`t * elem`) $\to$ `elem` retournant un représentant de la classe de l'élément. Si deux éléments x et y sont dans la même classe, dans le partitionnement p, alors `find`$(p, x) =$ `find`(p, y) ;
- `union` : (`t*elem*elem`) $\to$ `t` retourne le partitionnement dans lequel on a fusionné les classes des arguments.

EXEMPLE :
On réalise le *pseudo-code* ci-dessous.

- $p \leftarrow$ `initialise_partition`$([1, 2, 3, 4, 5]) \rightsquigarrow \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$
- `find`$(p, 1) = 1$
- `union`$(p, 1, 3) \rightsquigarrow \{\{1, 3\}, \{2\}, \{4\}, \{5\}\}$
- `find`$(p, 1) =$ `find`$(p, 3)$

On implémente ce type abstrait en OCAML.

REMARQUE (Niveau zéro – listes de liste) :

```
1 type 'a t = 'a list list
```

```
2
3  let initialise_partition (l: 'a list): 'a t =
4    List.map (fun x -> [ x ] ) l
5
6  let rec find (p: 'a t) (x: 'a): 'a =
7    match p with
8    | classe :: classes ->
9        if List.mem x classe then List.hd classe
10       else find classes x
11   | [] -> raise Not_Found
12
13 let est_equiv (p: 'a t) (x: 'a) (y: 'a): bool =
14   (find p x) = (find p y)
15
16 let rec extrait_liste (x: 'a) (p: 'a t): 'a list * 'a p =
17   match p with
18   | classe :: classes ->
19       if List.mem x classe then (classe, classes)
20       else
21         let cl, cls' = extrait_liste x classes in
22         (cl, classe :: cls')
23   | [] -> raise Not_Found
24
25 let union (p: 'a t) (x: 'a) (y: 'a): 'a t =
26   if est_equiv p x y then p
27   else
28     let cx, p' = extrait_liste x p in
29     let cy, p'' = extrait_liste y p' in
30     (cx @ cy) :: p''
```

Code 1 – Implémentation du type UnionFind en OCaml

Remarque (Niveau un – tableau de classes) :
Dans la case du tableau, on inscrit le numéro de sa classe. Pour `find`, on prend le premier ayant la même classe. Pour `union`, on re-numérote vers un numéro commun. Par exemple,

0	1	0	0	1	2
0	1	2	3	4	5

$\longleftrightarrow \quad \{\{0,2,3\},\{1,4\},\{5\}\}.$

Remarque (Niveau deux – tableau de représentants) :
Dans les cases du tableau, on écrit le représentant de la classe de i. Pour `find`, on lit la case. Pour `union`, on re-numérote vers un numéro commun. Par exemple,

2	4	2	2	4	5
0	1	2	3	4	5

$\longleftrightarrow \quad \{\{0,2,3\},\{1,4\},\{5\}\}.$

Remarque (Niveau trois – arbres) :
Pour `union`$(0,1)$, on cherche le représentant de 0 (2) puis celui de 1 (4). On fait pointer 4 vers 2. Pour la suite de l'implémentation, *c.f.* DM3.

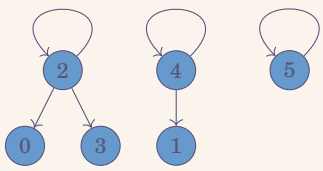

Figure 7 – Représentation par des arbres

Avec cette nouvelle structure, on peut maintenant revenir sur l'algorithme de Kruskal.

Algorithme 5 Algorithme de Kruskal – version 2

Entrée Un graphe $G = (S, A, c)$ un graphe non orienté, pondéré
Sortie Un acpm

```
Soit (e_i)_{i∈⟦1,m⟧} un tri des arrêtes par coût croissant
f ← 0          ▷ Nombre d'union effectuées
p ← initialise_partition(S)
I ← 0
B ← ∅
tant que f < n − 1 faire
    {x, y} ← e_I
    si find(p, x) ≠ find(p, y) alors
        p ← union(p, x, y)
        B ← B ∪ {{x, y}}
        f ← f + 1
    I ← I + 1
retourner (S, B)
```

Étude de complexité. Notons $C^n_{\texttt{find}}$ un majorant du coût de `find` sur une structure contenant n éléments, notons $C^n_{\texttt{union}}$ un majorant du coût de `union` sur une structure contenant n éléments, et notons $C^n_{\texttt{init}}$ un majorant du coût de `init` sur une structure contenant n éléments. La complexité de cet algorithme est de

$$\mathcal{O}\big(C^n_{\texttt{init}} + 2m\, C^n_{\texttt{find}} + n\, C^n_{\texttt{union}} + m \log_2 m\big).$$

3 Couplage dans un graphe biparti

Définition (Couplage) : On appelle *couplage* d'un graphe non orienté $G = (S, A)$, la donnée d'un sous-ensemble $C \subseteq A$ tel que

$$\forall \{x, y\}, \{x', y'\} \in C, \qquad \{x, y\} \cap \{x', y'\} \neq \varnothing \implies \{x, y\} = \{x', y'\}.$$

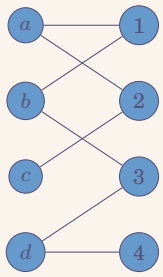

Figure 8 – Exemple de couplage

Exemple :
On réutilise l'exemple ci-dessous dans toute la section. L'ensemble $C = \{\{a, 2\}, \{b, 3\}\}$ est un couplage. Mais, l'ensemble $C' = \{\{a, 1\}, \{a, 2\}\}$ n'en est pas un.

Définition : Un couplage est dit *maximal* s'il est maximal pour l'inclusion ($\subseteq$). Un couplage est dit *maximum* si son cardinal est maximal.

Exemple :
Dans l'exemple précédent,

— le couplage $C = \{\{a, 2\}, \{b, 3\}\}$ n'est ni maximal, ni maximum ;
— le couplage $C' = \{\{a, 2\}, \{b, 3\}, \{d, 4\}\}$ est maximal mais pas maximum ;
— le couplage $C'' = \{\{a, 1\}, \{b, 3\}, \{c, 2\}, \{d, 4\}\}$ est maximum.

Remarque :
Dans toute la suite, on ne considère que des graphes bipartis.

Définition : Étant donné un graphe biparti $G = (S, A)$ et un couplage C, un sommet x est dit *libre* dès lors que

$$\forall \{y, z\} \in C,\ x \notin \{y, z\}.$$

Une chaîne élémentaire[3] $(c_0, c_1, \ldots, c_{2p+1})$ est dit *augmentante* si

— c_0 et c_{2n+1} sont libres ;
— $\forall i \in [\![0, p]\!]$, $\{c_{2i}, c_{2i+1}\} \in A \setminus C$;
— $\forall i \in [\![0, p-1]\!]$, $\{c_{2i+1}, c_{2i+2}\} \in C$.

Exemple :

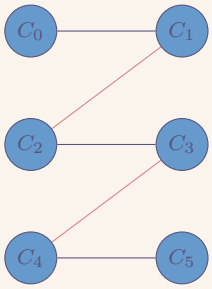

Figure 9 – Chaîne augmentante

Exemple :
Dans l'exemple de cette section, $(d, 4)$ et $(c, 2, a, 1)$ sont deux chaînes augmentantes.

Propriété : Étant donné un graphe biparti $G = (S, A)$ avec $S = S_1 \cup S_2$ (partitionnement du graphe biparti), un couplage C est maximum si, et seulement s'il n'admet pas de chaînes augmentantes.

Preuve : " $\Longrightarrow$ " Soit C un couplage admettant une chaîne augmentante. Montrons que C n'est pas maximum. Soit la chaîne augmentante[4]

$$c_0 \to c_1 \Rightarrow c_2 \to c_3 \Rightarrow c_4 \to \cdots \to c_{2p-1} \Rightarrow c_{2p} \to c_{2p+1}.$$

3. *i.e.* une chaîne sans boucles.

On considère alors le couplage

$$C' = \Big(C \setminus \{\{c_{2i+1}, c_{2i+2}\} \mid i \in [\![0, p-1]\!]\}\Big) \cup \{\{c_{2i}, c_{2i+1}\} \mid i \in [\![0, p]\!]\}.$$

On transforme donc la chaîne en

$$c_0 \Rightarrow c_1 \to c_2 \Rightarrow c_3 \to \cdots \to c_{2p-1} \to c_{2p} \Rightarrow c_{2p+1}.$$

C'est bien un couplage, et $\mathrm{Card}(C') = \mathrm{Card}\, C + 1$. C n'est donc pas un couplage maximum.

" $\Longleftarrow$ " Soit C un couplage non maximum. Montrons que C admet une chaîne augmentante. Soit M un couplage maximum, et $D = C \triangle M = (C \setminus M) \cup (M \setminus C)$. On a $\mathrm{Card}\, C < \mathrm{Card}\, M$ et $\mathrm{Card}(C \setminus M) < \mathrm{Card}(M \setminus C)$. On remarque que, si $c_0 \to c_1 \to c_2 \to \cdots \to c_{p-1} \to c_p$ est une chaîne de D, (si $c_0 \to c_1 \in C \setminus M$ et $c_1 \to c_2 \in C \setminus M$ donc c_1 est dans deux arrêtes distinctes d'un couplage C, ce qui est absurde ; de même pour les autres arrêtes). Ainsi, 2 arrêtes consécutives ne sont pas dans la même composante de l'union $(C \setminus M) \cup (M \setminus C)$. Considérons la relation d'équivalence $\sim$ sur D définie par $\{x, y\} \sim \{z, t\} \overset{\text{def.}}{\Longleftrightarrow}$ il existe une chaîne de D utilisant l'arrête $\{x, y\}$ et l'arrête $\{z, t\}$. Soit le partitionnement $D_1, \ldots, D_q$ de D par $\sim$. Par inégalité de cardinal, il existe un D_i tel que

$$\mathrm{Card}\{e \in D_i \mid e \in C\} < \mathrm{Card}\{e \in D_i \mid e \in M\}.$$

L'ensemble D_i contient alors une chaîne augmentante.

□

Algorithme 6 ChaîneAugmentante : Trouver une chaîne augmentante dans un graphe biparti $G = (S, A)$ muni d'un couplage C partant d'un sommet $s \in S$

```
1:  Procédure Augmente(x, chaîne)
2:      pour y ∈ Succ(x) \ chaîne faire
3:          si y est libre dans C alors
4:              retourner Some(chaîne ⊎ (y))
5:          sinon
6:              Soit z tel que {y, z} ∈ C.
7:              r ← Augmente(z, chaîne ⊎ (y, z))
8:              si r ≠ None alors
9:                  retourner r
10:         retourner None
11: si s est libre dans C alors
12:     retourner Augmente(s, (s))
13: sinon
14:     retourner None
```

Remarque :
Si un sommet n'est pas libre dans le couplage C, il n'est pas libre dans les couplage obtenus par inversion de chaîne depuis C.

Algorithme 7 Calcul d'un couplage maximum

Entrée $G = (S, A)$ un graphe biparti, avec $S = S_1 \cup S_2$

```
1:  C ← ∅
2:  Done ← ∅
3:  tant que ∃x ∈ S1 \ Done faire
4:      Soit un tel x.
5:      r ← ChaîneAugmentante(G, C, x)
6:      si r ≠ None alors
7:          Some(a) ← r
8:          On inverse la chaîne a dans C.
9:      Done ← {x} ∪ Done
10: retourner C
```

4. On représente $\Rightarrow$ pour les arrêtes dans le couplage C.

Annexe A. Remarques supplémentaires

Le parcours d'un graphe $G = (S, A)$ a une complexité en $\mathcal{O}(|S| + |A|)$.