Parcours de graphes orientés

Jean Goubault-Larrecq

1er décembre 2020

Résumé

Nous traitons des parcours en profondeur de graphes orientés, consolidant les démonstrations du BBC, pages 107–115, et en apportant des variantes.

Ce texte se fonde sur, et adapte, une partie du livre *Éléments d'algorithmique*, par Danièle Beauquier, Jean Berstel, et Philippe Chrétienne, que l'on dénotera familièrement par BBC. Les résultats de ce livre seront annoncés sous la forme **Proposition 4.5 (BBC, page 128)** ou **Lemme 4.6 (BBC, page 129)** par exemple. Le résultats propres à ce texte-ci suivront une numérotation simple, comme **Lemme 6** ou **Proposition 9**.

Étant donné un graphe orienté fixé $G \stackrel{\text{def}}{=} (S, A)$, je noterai $u \to v$ pour dire qu'il y a un arc de u à v, autrement dit que $(u, v) \in A$.

Une *source* d'un graphe orienté G est un sommet à partir duquel tout sommet est accessible. Un graphe orienté n'a pas nécessairement d'source. Étant donné un graphe orienté $G \stackrel{\text{def}}{=} (S, A)$, on peut former un nouveau graphe G_* en lui adjoignant un nouveau sommet $*$, et de nouveaux arcs $* \to u$, $u \in S$. Le sommet $*$ est alors une source de G_*. Dans la suite, et sauf exception, on supposera que $G \stackrel{\text{def}}{=} (S, A)$ est un graphe orienté d'unique source $*$. (Ceci simplifie le cas général traité dans le BBC.)

source

1 Arbres orientés

Définition 1 *Un* arbre *(orienté) est un triplet $(S, *, p)$ où $* \in S$, p est une application de $S \setminus \{*\}$ dans S, et pour tout $u \in S$, il existe un $n \in \mathbb{N}$ tel que $p^n(u) = * : *$ est la* racine *de l'arbre, $p(u)$ est le* prédécesseur *de u, et seule la racine n'a pas de prédécesseur.*

arbre racine prédécesseur

Ceci n'est pas (du tout) la définition d'un arbre du BBC, même s'il y a des liens.

Un arbre peut être vu comme un graphe orienté (S, A) avec source $*$, où $A \stackrel{\text{def}}{=} \{(p(u), u) \mid u \in S \setminus \{*\}\}$. Il y a alors un *unique* chemin $* = p^n(u) \to p^{n-1}(u) \to \cdots \to p(u) \to u$ de la racine $*$ vers n'importe quel sommet u.

Lemme 2 *Dans un arbre $T \stackrel{def}{=} S, *, p)$:*

1. *La relation $\to_T^*$ est une relation d'ordre.*

2. *Si $u \to_T^* v$ alors il existe un unique chemin de u à v dans l'arbre.*
3. *Deux sommets quelconques u et v de S ont une borne inférieure $u \wedge v$ pour $\to_T^*$, appelée leur* ancêtre commun le plus proche. ancêtre commun le plus proche
4. *Si $u \to_T^* w$ et $v \to_T^* w$, alors u et v sont* comparables*, c'est-à-dire $u \to_T^* v$ ou $v \to_T^* u$.* comparables
5. *Pour tous sommets u, v, s'il existe un chemin de u à v et $u \neq v$, alors il existe un chemin de u à $p(v)$ (et un arc de $p(v)$ à v).*

Démonstration. On démontre d'abord que : $(*)$ si $p^m(u)$ est défini et est égal à u, alors $m = 0$. Sinon, $p^{km}(u)$ serait lui aussi défini et égal à u pour tout $k \in \mathbb{N}$. Ceci impliquerait que $p^\ell(u)$ serait défini pour une infinité de valeurs de ℓ. Or il existe un entier n tel que $p^n(u) = *$, et donc tel que $p^\ell(u)$ ne soit défini pour aucun $\ell \leq n+1$, ce qui est impossible. Donc $m = 0$.

1. La réflexivité et la transitivité sont évidentes. Supposons $u \to_T^* v$ et $v \to_T^* u$. Donc $u = p^m(v)$ et $v = p^{m'}(u)$ pour deux entiers m et m'. On en déduit $p^{m+m'}(u) = u$, et donc, par $(*)$, que $m + m' = 0$, en particulier $u = v$.

2. Si $u \to_T^* v$, alors $u = p^m(v)$ pour un certain entier m. S'il existait un autre chemin de u à v, c'est que u serait aussi égal à $p^{m'}(v)$ pour un $m' \neq m$. Par symétrie, disons $m' > m$. On aurait alors $p^{m'-m}(u) = u$. Donc $m' = m$, par $(*)$: contradiction.

3. Par le point 2, on a deux chemins uniques $* = p^m(u) \to_T p^{m-1}(u) \to_T \cdots \to_T p(u) \to_T u$ et $* = p^n(v) \to_T p^{n-1}(v) \to_T \cdots \to_T p(v) \to_T v$. Leur plus grand préfixe commun est un chemin de $*$ à un sommet w tel que $w \to_T^* u$ et $w \to_T^* v$. Plus précisément, on a $w \stackrel{\text{def}}{=} p^{m-k}(u) = p^{n-k}(v)$, où k est l'entier le plus grand ($\leq \min(m,n)$) tel que $p^{m-k}(u) = p^{n-k}(v)$.

Si w' est n'importe quel sommet tel que $w' \to_T^* u$ et $w' \to_T v$. Alors w' s'écrit à la fois $p^i(u)$ et $p^j(v)$. De plus, il existe un entier k tel que $p^k(w) = *$. Donc le chemin $* = p^k(w) \to_T p^{k-1}(w) \to_T \cdots \to_T p(w) \to_T w$ est un préfixe de l'unique chemin de $*$ à w. En particulier, $w' \to_T^* w$.

4. Soient m et n des entiers tels que $u = p^m(w)$ et $v = p^n(v)$. Si $m \geq n$, alors $u = p^{m-n}(v)$ donc $u \to_T^* v$. Sinon, $v \to_T^* u$.

5. Il y a en fait un unique chemin $u = p^n(v) \to_T p^{n-1}(v) \to_T \cdots \to_T p(v) \to_T v$ de u à v, par le point 2. Comme $u \neq v$, on a $n \geq 1$, ce qui nous permet de conclure. □

Un *arbre couvrant* de G est un arbre qui forme un sous-graphe de G, et qui contient tous les sommets de G. La racine d'un arbre couvrant est nécessairement une source de G (donc son unique source $*$, vu notre hypothèse de départ sur G). arbre couvrant

2 Parcours

On rappelle que l'on suppose que G a une source fixée $*$.

Un *parcours* de G est une liste de sommets $L \stackrel{\text{def}}{=} [u_1, \cdots, u_n]$ de G telle que : parcours

— chaque sommet de G apparaît exactement une fois dans L ;

— $u_1 = *$;

— chaque sommet de L, sauf le premier, est successeur (dans G) d'un élément qui le précède dans L : autrement dit, pour tout j $(1 < j \leq n)$, il existe un i $(1 \leq i < j)$ tel que $u_i \to u_j$.

L'unique élément qui n'a pas de précédesseur doit donc être le premier ; autrement dit, on doit avoir $u_1 = *$.

Étant donné un parcours $L \stackrel{\text{def}}{=} [u_1, \cdots, u_n]$ de G, on notera L_k $(0 \leq k \leq n)$ le préfixe $[u_1, \ldots, u_k]$ formé des k premiers éléments de L. On a $L_n = L$. Les L_k, $1 \leq k \leq n$, sont des parcours partiels : par définition, un *parcours partiel* de G est une liste L_* de sommets $[u_1, \cdots, u_k]$ où $1 \leq k \leq n$, où $u_1 = *$, où chaque sommet de G apparaît *au plus* une fois dans L_*, et où chaque sommet sauf le premier est successeur d'un élément qui le précède dans L_*. Un parcours partiel à n éléments est un parcours.

parcours partiel

Si pour chaque indice j $(1 < j \leq n)$, on *choisit* un i $(1 \leq i < j)$ tel que $u_i \to u_j$, on dira que $u_i \to u_j$ est un *arc de liaison*. On peut organiser les choix des arcs de liaison (un pour chaque j) comme une fonction $p \colon S \setminus \{*\} \to S$, qui à u_j associe *un* u_i avec $i < j$ tel que $u_i \to u_j$. Un tel choix n'est pas unique. Mais, étant donné un parcours L de G, tout choix d'une telle fonction p définit un arbre couvrant de G. On obtient donc :

arc de liaison

Proposition 4.5 (BBC, page 128, modifiée) *Tout choix d'arcs de liaison d'un parcours de G constitue un arbre couvrant de G.*

Remarque 3 *C'est un arbre, et pas une forêt comme dans le BCC, parce que G a une source $*$. Si l'on ne suppose pas que G a une source, on définit un parcours de G comme étant une liste L telle que $* :: L$ soit un parcours de G_*. Tout choix d'arcs de liaison d'un tel parcours constitue un arbre couvrant de G_*, donc une* forêt couvrante *de G après en avoir supprimé la racine $*$. (Une forêt est une union disjointe finie d'arbres.)*

forêt couvrante

Un *sommet fermé* d'un parcours partiel L_* est un sommet de L_* dont tous les successeurs (dans G) sont aussi dans L_*. Tout sommet est fermé dans un parcours (non partiel) L.

sommet fermé

Un *sommet ouvert* de L_* est un sommet de L_* qui a au moins un successeur qui n'est pas dans L_*. Tout sommet de G est soit ouvert dans L_*, soit fermé dans L_*, soit hors de L_*.

sommet ouvert

Lemme 4 *Soit $L_* \stackrel{def}{=} [u_1, \cdots, u_k]$ un parcours partiel de G, $1 \leq k \leq n$. Si $k < n$, alors L_* contient un sommet ouvert.*

Démonstration. Supposons le contraire. L'ensemble $\{u_1, \cdots, u_k\}$ des sommets de L_* forme alors une famille de sommets de G contenant $*$ et stable par successeurs, puisque tous ses sommets sont fermés dans L_k. Cet ensemble contient donc l'ensemble $Reach(*)$ des sommets accessibles depuis $*$. Or $Reach(*)$ vaut S tout entier, par définition de la source $*$. Donc il y a n éléments dans $Reach(*) = \{u_1, \cdots, u_k\}$, ce qui est impossible puisque $k < n$. □

Corollaire 5 *Il existe au moins un parcours L de G.*

Démonstration. Par récurrence sur k, on montre que pour tout k avec $1 \leq k \leq n$, il existe un parcours partiel de G. Pour $k = 1$, $[*]$ est un tel parcours partiel. Si $1 < k \leq n$, par hypothèse de récurrence il existe un parcours partiel $L_* \stackrel{\text{def}}{=} [u_1, \cdots, u_{k-1}]$ à $k-1$ éléments de G. Par le lemme 4, L_* contient un sommet ouvert u_i $(1 \leq i < k)$. Par définition, il existe un arc $u_i \to u_k$, où u_k est un sommet de G qui n'est pas dans L_*. Donc $[u_1, \cdots, u_{k-1}, u_k]$ est un parcours partiel de longueur k. □

Si L est un parcours de G, il existe donc pour chaque j $(1 < j \leq n)$ un *plus petit* indice i $(1 \leq i < j)$ tel que u_i soit un sommet ouvert de L_{j-1}. On dit que L est un *parcours en largeur* s'il y a un arc de chacun de cet u_i vers u_j, pour chaque j. On choisit alors (u_i, u_j), avec i choisi ainsi, comme arc de liaison d'extrémité u_j, pour chaque j. L définit un arbre couvrant de G, par $p(u_j) \stackrel{\text{def}}{=} u_i$ (pour chaque j, i étant le plus petit tel que u_i soit ouvert dans L_{j-1}). On dit usuellement, mais plus informellement (voir le BBC) : dans un parcours en largeur, l'origine de tout arc de liaison est le *premier* sommet ouvert déjà visité.

parcours en largeur

Remarque 6 *Dans le cas d'un graphe de la forme G_*, où il y a un arc $* \to u$ pour tout sommet u autre que $*$, la notion de parcours en largeur est triviale : tout parcours est en largeur. Elle l'est moins pour d'autres graphes.*

Lemme 7 *Soit $L \stackrel{def}{=} [u_1, \cdots, u_n]$ un parcours de G. On suppose un choix d'arcs de liaison effectué, ce qui définit un arbre couvrant T de G.*

1. *Tout sommet u de G s'écrit u_i pour un unique i $(1 \leq i \leq n)$.*
   On appelle i le rang *$r(u)$ de u dans L.*
2. *Tout descendant v d'un sommet u dans T satisfait $r(v) \geq r(u)$.*
3. *Soit $1 \leq i \leq j \leq k \leq n$. Si u_i est un sommet de L_j qui est ouvert dans L_k, alors il est ouvert dans L_j.*

rang

Démonstration. 1. Évident.

2. Il suffit de le démontrer lorsque $u = p(v)$, le résultat s'en déduisant par récurrence sur la longueur de l'unique chemin de u à v dans T. Soit $j \stackrel{\text{def}}{=} r(v)$, c'est-à-dire $u_j = v$. Le sommet $u = p(v)$ est défini comme un u_i avec $i < j$ satisfaisant certaines conditions. Mais alors $r(u) = i < j = r(v)$.

3. Si u_i était fermé dans L_j, tous ses successeurs (dans G) seraient des u_ℓ avec $\ell \leq j$. En particulier, $\ell \leq k$, et donc u_i serait fermé dans L_k. □

3 Parcours en profondeur

Dans un parcours en profondeur, c'est le *dernier* sommet ouvert déjà visité qui est l'origine de tout arc de liaison. Autrement dit,

Définition 8 *Soit $L \stackrel{def}{=} [u_1, \cdots, u_n]$ un parcours de G. L est un* parcours en profondeur *si et seulement, pour chaque j $(1 < j \leq n)$, le* plus grand *indice i $(1 \leq i < j)$ tel que u_i soit un sommet ouvert de L_{j-1} est tel que $u_i \to u_j$.*

parcours en profondeur

On rappelle qu'un tel i existe toujours par le lemme 4. On définit ces arcs $u_i \to u_j$ comme étant les arcs de liaison, et ceci définit de nouveau un arbre couvrant de G.

Nous fixons désormais un parcours en profondeur $L \stackrel{\text{def}}{=} [u_1, \cdots, u_n]$ de G (avec $u_1 = *$), et nous notons $T \stackrel{\text{def}}{=} (S, *, p)$ l'arbre couvrant associé. Un *descendant* d'un sommet u est un sommet v accessible depuis u dans T, c'est-à-dire tel que $u \to_T^* v$, autrement dit $p^n(v) = u$ pour un certain $n \in \mathbb{N}$. On dit aussi que u est un *ascendant* de v dans T.

descendant

ascendant

La proposition suivante n'est vraie que des parcours en profondeur, et est le point clé.

Proposition 9 *Pour tout sommet u de G, l'ensemble des descendants $D_T(u)$ de u dans T est un* intervalle *de L, c'est-à-dire un ensemble de la forme $\{u_i, u_{i+1}, \cdots, u_j\}$, où $i = r(u)$. On appelle j le* temps de fin *$fin(u)$ de u dans L.*

intervalle

temps de fin

De plus, $j = fin(u)$ est le plus petit entier supérieur ou égal à i tel que tous les sommets u_i, u_{i+1}, …, u_j soient fermés dans L_j.

Démonstration. On note d'abord que, par le lemme 7 (2), tout descendant de u a un rang plus grand ou égal à $r(u)$. L'ensemble des descendants $D_T(u)$ de u dans T est donc un sous-ensemble de $\{u_i, u_{i+1}, \cdots, u_n\}$, où $i \stackrel{\text{def}}{=} r(u)$. Il contient aussi $u_i = u$.

Soit j le plus petit entier supérieur ou égal à i tel que tous les sommets u_i, u_{i+1}, …, u_j soient fermés dans L_j. Un tel j existe (et est inférieur ou égal à n) car tous les sommets sont fermés dans L_n. On montre que tous les sommets u_k ($i \le k \le j$) sont dans $D_T(u)$, par récurrence sur $k - i$:

- Si $k - i = 0$, $u_i = u$ est dans $D_T(u)$.
- Sinon, par minimalité de j, et comme $k - 1 < j$, il existe un sommet parmi u_i, u_{i+1}, …, u_{k-1} qui soit ouvert dans L_{k-1}. Par définition d'un parcours en profondeur, l'arc de liaison $u_\ell \to_T u_k$ est tel que ℓ est maximal parmi les indices tels que u_ℓ soit ouvert dans L_{k-1}. Ceci implique $\ell \ge i$. Ainsi, on peut utiliser l'hypothèse de récurrence, d'où l'on déduit que $u_\ell \in D_T(u)$. On a donc un chemin de u à u_ℓ dans T, auquel on peut ajouter l'arc de liaison $u_\ell \to_T u_k$, et on en conclut que u_k est lui aussi dans $D_T(u)$.

Pour montrer que $D_T(u)$ vaut exactement $\{u_i, u_{i+1}, \cdots, u_j\}$, on raisonne par l'absurde. Si ce n'est pas le cas, il existe un sommet u_k, avec $j < k \le n$, dans $D_T(u)$. Choisissons k minimal avec cette propriété. Comme $k > j \ge i$, u_k est différent de $u_i = u$. Donc le prédécesseur $u_\ell \stackrel{\text{def}}{=} p(u_k)$ de u_k dans T est bien défini. De plus, l'unique chemin de u à u_k dans T se décompose en un chemin de u à u_ℓ (dans T), suivi de l'arc de liaison $u_\ell \to_T u_k$ (Lemma 2 (5)). Il s'ensuit que u_ℓ est aussi dans $D_T(u)$.

- Il est impossible que $\ell > j$, à cause de la minimalité de k.
- Si $i \le \ell \le j$, le sommet u_ℓ, qui est ouvert dans L_{k-1} par la définition du parcours en profondeur, serait aussi ouvert dans L_j, par le lemme 7 (3). C'est impossible aussi, parce que tous les sommets u_i, u_{i+1}, …, u_j sont fermés dans L_j.
- Finalement, le cas $\ell < i$ est impossible lui aussi, puisque l'existence d'un chemin dans T de $u = u_i$ à u_ℓ implique $i \le \ell$, par le lemme 7 (2). □

Proposition 10 *Pour tout couple de sommets u et v dans G, les trois propriétés suivantes sont équivalentes :*

1. *v est un descendant de u dans T ;*
2. *$[r(v), fin(v)] \subseteq [r(u), fin(u)]$;*
3. *$r(u) \leq r(v)$ et $fin(v) \leq fin(u)$.*

Démonstration. $1 \Rightarrow 2$. Comme v est un descendant de u dans T, $D_T(v) \subseteq D_T(u)$. Par la proposition 9, $D_T(v) = \{u_{r(v)}, u_{r(v)+1}, \cdots, u_{fin(v)}\}$ et $D_T(u) = \{u_{r(u)}, u_{r(u)+1}, \cdots, u_{fin(u)}\}$. Donc les indices $r(v)$, $r(v)+1$, ..., $fin(v)$ sont tous dans l'ensemble $\{r(u), r(u)+1, \cdots, fin(u)\}$.

Il est clair que 2 et 3 sont équivalents.

$3 \Rightarrow 1$. Par la proposition 9, $D_T(u) = \{u_{r(u)}, u_{r(u)+1}, \cdots, u_{fin(u)}\}$. Comme $r(u) \leq r(v) (\leq fin(v)) \leq fin(u)$, $v = u_{r(v)}$ est dans $D_T(u)$. □

Proposition 11 *Pour tout couple de sommets u et v dans G, tels qu'aucun des deux n'est descendant de l'autre dans T, les intervalles $[r(u), fin(u)]$ et $[r(v), fin(v)]$ sont disjoints.*

Démonstration. S'ils ne l'étaient pas, il y aurait un indice i dans les deux. Par la proposition 9, u_i serait donc dans $D_T(u)$ et dans $D_T(v)$, autrement dit $u \rightarrow_T^* u_i$ et $v \rightarrow_T^* u_i$. Par le lemme 2 (4), u et v seraient comparables dans $\rightarrow_T^*$, ce qui contredirait l'hypothèse. □

Étant données deux sommets u_i et u_j, avec $i < j$, on n'a donc que deux possibilités : soit u_j est un descendant de u_i dans T (et $[i, fin(u_i)]$ contient $[j, fin(u_j)]$), soit les intervalles $[i, fin(u_i)]$ et $[j, fin(u_j)]$ sont disjoints, et donc en particulier $fin(u_i) < j$.

Ceci mène à la classification suivante des arcs $u \rightarrow v$ de G :

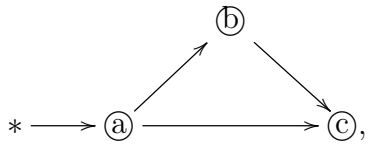
— c'est une *boucle* si $u = v$; boucle

— c'est un *arc avant* si $u \rightarrow_T^* v$ et $u \neq v$; arc avant

— c'est un *arc arrière* si $v \rightarrow_T^* u$ et $u \neq v$; arc arrière

— c'est un *arc transverse* sinon. arc transverse

Remarque 12 *Parmi les arcs avants, on trouve bien sûr les arcs $u \rightarrow_T v$ de l'arbre T. Mais il y en d'autres. Par exemple, dans le graphe :*



avec comme parcours en profondeur $[*, ⓐ, ⓑ, ⓒ]$, l'arc ⓐ → ⓒ est un arc avant qui n'est pas un arc de l'arbre T.

Lemme 4.6 (BBC, page 129) *Pour tout arc transverse $u \rightarrow v$, $r(u) > r(v)$.*

$u \to v$	$r(u)\ ?\ r(v)$	$fin(u)\ ?\ fin(v)$	intervalles
boucle	$=$	$=$	$[u = v \quad \cdots]$
avant	$<$	$\geq$	$[u \quad \cdots \quad [v \quad \cdots] \quad \cdots]$
arrière	$>$	$\leq$	$[v \quad \cdots \quad [u \quad \cdots] \quad \cdots]$
transverse	$>$	$>$	$[v \quad \cdots] \cdots [u \quad \cdots]$

TABLE 1 – Classification des arcs dans un parcours en profondeur

Démonstration. Imaginons que $r(u) \leq r(v)$, et donc $r(u) < r(v)$ puisque $u \neq v$, un arc transverse n'étant pas une boucle. Par la proposition 9, Notons $i \stackrel{\text{def}}{=} r(u)$, $k \stackrel{\text{def}}{=} r(v)$, de sorte que $u = u_i$, $v = u_k$, et $i < k$.

Par la première partie de la proposition 9, $D_T(u)$ est un intervalle $\{u_i, u_{i+1}, \cdots, u_j\}$, avec $j = fin(u)$. De même, $D_T(v)$ est un intervalle $\{u_k, u_{k+1}, \cdots, u_\ell\}$, avec $\ell = fin(v)$. Les intervalles $[i, j]$ et $[k, \ell]$ sont disjoints par la proposition 11. Comme $i < k$, on en déduit $j < k$.

Mais alors, l'arc $u \to v$ (c'est-à-dire $u_i \to u_k$) montre que u_i est un sommet ouvert dans L_j, ce qui contredit la deuxième partie de la proposition 9. □

Note 13 *La démonstration dans le BBC ne me satisfait pas, pour deux raisons. En premier, elle utilise un raisonnement temporel (« lorsque le sommet x est visité, le sommet y ne l'est pas encore ») qui est, au mieux, une bonne explication intuitive, mais ne constitue pas une preuve sérieuse. C'est malheureusement assez classique en algorithmique. En second, le point crucial (« y sera accessible* dans l'arborescence *à partir de x ») ne me semble pas justifié. Or c'est la proposition 9 qui permet de le justifier ; c'est le point clé des parcours en profondeur, et il n'est pas mentionné dans le BBC, ce qui est curieux.*

Pour un arc transverse $u \to v$, le fait que $r(u) > r(v)$ et que les intervalles $[r(u), fin(u)]$ et $[r(v), fin(v)]$ soient disjoints implique encore mieux : que $r(v) \leq fin(v) < r(u) \leq fin(u)$.

Tout ceci permet de classer les arcs comme montré en table 1.

Un *circuit* est un chemin $u \to \cdots \to u$ contenant au moins un arc. En particulier, les boucles sont des circuits. circuit

Lemme 4.7 (BBC, pages 129–130) *Le graphe G est sans circuit si et seulement si il n'a ni boucle ni arc arrière (par rapport au parcours en profondeur L).*

Démonstration. Si $u \to v$ est un arc arrière, alors $v \to_T^* u$ donc $v \to^* u$ (dans G), ce qui exhibe un circuit dans G.

Réciproquement, supposons qu'il n'y ait aucun arc arrière. Tous les arcs sont avant ou transverses. Ordonnons les sommets u par la forme suivante d'ordre lexicographique sur $(fin(u), r(u))$: $u \succ v$ si et seulement si $fin(u) > fin(v)$, ou bien $fin(u) = fin(v)$ et $r(u) < r(v)$. La table 1 montre que tout arc $u \to v$, qu'il soit avant ou transverse, satisfait

$u \succ v$. Comme $\succ$ est un ordre strict (transitif, irréflexif), l'existence d'un circuit de u à u impliquerait $u \succ u$, ce qui est impossible. □

Le BBC ne mentionne pas les boucles : et pour cause, il ne traite que des graphes sans boucles.

4 Calcul des rangs et des temps de fin

On peut construire un parcours $L \stackrel{\text{def}}{=} [u_1, \cdots, u_n]$ de G en construisant ses préfixes L_k, par récurrence sur k, comme suit. D'abord, $L_1 \stackrel{\text{def}}{=} [*]$. (Par commodité, on posera aussi $L_0 \stackrel{\text{def}}{=} []$.) Ensuite, pour tout k ($1 \leq k \leq n-1$), en supposant $L_k \stackrel{\text{def}}{=} [u_1, \cdots, u_k]$ construit, on note qu'il existe nécessairement un u_i ($1 \leq i \leq k$) et un arc $u_i \to v$, avec $v \notin \{u_1, \cdots, u_k\}$. En effet, sinon, L_k serait une liste de sommets contenant $*$ ($= u_1$) et stable par successeurs, et contiendrait donc $Reach(*)$. Or $Reach(*) = S$, donc le cardinal de L_k vaudrait au moins n, ce qui est impossible puisque $k \leq n-1$. On pose alors $L_{k+1} \stackrel{\text{def}}{=} [u_1, \cdots, u_k, v]$.

Si l'on souhaite prouve que $L = L_n$ est bien un parcours de G, il suffit de montrer les invariants :

— chaque sommet de G apparaît au plus une fois dans L_k ;

— $*$ est le premier sommet de L_k ;

— chaque sommet de L_k, sauf le premier, est successeur d'un élément qui le précède dans L_k.

On ne demande donc pas que tout sommet de G apparaisse dans L_k. C'est la seule différence par rapport à un parcours.

Revenons sur la façon donnée ci-dessus de construire L_k. Nous choisissons maintenant i *maximal* parmi les indices possibles. L est alors un parcours en profondeur. Pour le prouver, on considère l'invariant supplémentaire :

— chaque sommet u_j de L_k, sauf le premier, est successeur d'un sommet ouvert de L_{j-1}.

(On laissera au lecteur le soin d'adapter la notion de sommet ouvert à ce cadre.) Si parmi les indices i possibles on avait choisi le plus petit, alors L aurait été un parcours en largeur.

Supposons maintenant G représenté par liste de successeurs. On a donc une fonction `succ`: $S \to S$ `list` qui à tout sommet associe une liste (ordonnée) de ses successeurs. Nous supposerons que cette liste est sans répétition.

On peut contraindre encore davantage la construction de L en demandant que, à i fixé, v soit le *premier* élément de `succ`(u_i) qui ne soit pas dans $\{u_1, \cdots, u_k\}$. Le parcours en profondeur L est alors défini de façon unique.

L'algorithme de la table 2, ou plutôt l'appel `dfs_1 (*)`, calcule les rangs et les temps de fin de chaque sommet u d'un graphe G, avec source $*$. Le graphe est donné par une fonction `succ : 'a -> 'a list` donnant la liste des successeurs d'un sommet. Les variables `marked : 'a set` (ensemble des sommets déjà marqués), `dfsNum : int` (compteur maintenant le nombre de sommets déjà marqués), `r : 'a -> int ref` (fonction rang) et `fin :`

```
fun dfs_1 (u) =
    (* Pre I_1(dfsNum, marked, r, fin) et r(u) = dfsNum + 1 *)
    marked := marked ∪ {u};
    r(u) := ++dfsNum;
    for each v ∈ succ(u) do
        if v ∉ marked
           then dfs_1 (v);
    fin(u) := dfsNum;
    (* Post I_1(dfsNum, marked, r, fin) et dfsNum = fin(u) = fin(u) *)
```

TABLE 2 – Un algorithme de parcours en profondeur basique

`'a -> int ref` (fonction temps de fin) sont supposées globales, pour simplifier. De plus, `dfsNum` est initialisée à 0, et `marked` à l'ensemble vide.

La précondition, repérée par `Pre`, est une hypothèse faite sur les entrées de `dfs_1`. Si elle est vraie, on prétend que la postcondition, repérée par `Post`, est vraie en sortie. Elles se lisent comme suit.

L'invariant $I_1(k, \texttt{marked}, \texttt{r}, \texttt{fin})$ est la conjunction des propriétés suivantes :

(i) `marked` est l'ensemble $\{u_1, \cdots, u_k\}$ des éléments de $L_k = [u_1, \cdots, u_k]$;

(ii) pour tout i $(1 \leq i \leq k)$, $\texttt{r}(u_i) = i$ $(= r(u_i))$;

(iii) pour tout sommet fermé u de L_k, $\texttt{fin}(u) = fin(u)$.

La condition $r(\texttt{u}) = \texttt{dfsNum} + 1$, en conjonction avec l'invariant et la définition d'un parcours, implique que `u` n'est pas marqué (pas dans l'ensemble `marked`) en entrée.

La postcondition implique, elle, que `u` soit marqué, ainsi que tous ses descendants, dans $D_T(\texttt{u})$ — mais pas davantage : si k désigne la valeur de `dfsNum` en entrée et k' celle de `dfsNum` en sortie, alors $L_{k'}$ est exactement le préfixe de L obtenu en ajoutant à la fin de L_k les sommets de $D_T(\texttt{u})$.

À la fin du calcul de $\texttt{dfs\_1}(*)$, la postcondition implique que $\texttt{dfsNum} = fin(*)$. Or $fin(*) = n$, et l'invariant implique donc que `r`, `fin` envoient chaque sommet de G vers son rang et son temps de fin, respectivement.

On laisse la preuve de correction de `dfs_1` en exercice. Pour l'effectuer, en plus des résultats déjà connus sur les parcours, on a besoin de la proposition 15 ci-dessous, qui permet d'assurer que $\texttt{dfsNum} = \texttt{fin}(\texttt{u}) = fin(\texttt{u})$ en postcondition. On remarque que ceci donne une définition par récurrence sur $n - r(u)$ de $fin(u)$ (voir la table 1), et c'est ce calcul par récurrence qui est réalisé par `dfs_1`, car les arcs de laison $\texttt{u} \rightarrow_T \texttt{v}$ sont exactement ceux explorés lors de l'appel récursif `dfs_1 (v)`, c'est-à-dire lorsque $\texttt{v} \notin \texttt{marked}$.

Lemme 14 *Pour tout sommet u de G, en posant $j \stackrel{def}{=} fin(u)$:*

1. *u_j est une feuille de l'arbre T ;*
2. *pour tout sommet v de G tel que $u \rightarrow_T v \rightarrow_T^* u_j$, $fin(v) = fin(u)$.*

Démonstration. 1. Si u_j n'est pas une feuille de T, il a un successeur v dans T. Or $u_j \to_T v$ implique $j = r(u_j) < r(v)$, ce qui est impossible, puisque $u \to_T^* u_j \to_V v$ implique que v est dans $D_T(u)$, donc que $r(v) \leq fin(u) = j$, par la proposition 10 (3).

2. Toujours par la proposition 10 (3), $u \to_T v$ implique $fin(v) \leq fin(u) = j$. Pour l'inégalité réciproque, par cette même proposition $v \to_T^* u_j$ implique $fin(u_j) \leq fin(v)$. Mais par le point 1, u_j est une feuille de T, donc $D_T(u_j) = \{u_j\}$, autrement dit $fin(u_j) = r(u_j)$ $(= j)$, par la proposition 9. Donc $j \leq fin(v)$. □

Proposition 15 *Pour tout sommet u de G, $fin(u)$ vaut :*

— *$r(u)$ si u est une feuille de T ;*

— *$\max\{fin(v) \mid u \to_T v\}$ sinon.*

Démonstration. Soit $fin'(u)$ l'entier $r(u)$ si u n'a pas de successeur dans T, $\max\{fin'(v) \mid u \to_T v\}$ sinon. On montre par récurrence (forte) sur $n - r(u)$ que $fin'(u) \leq fin(u)$. D'abord, on a toujours $r(u) \leq fin(u)$. Ensuite, si $u \to_T v$ alors $fin(v) \leq fin(u)$ par la proposition 10 (3), et par hypothèse de récurrence $fin'(v) \leq fin(v)$. En prenant le max sur tous les v tels que $u \to_T v$, on obtient $fin'(u) \leq fin(u)$.

On démontre que, réciproquement, $fin(u) \leq fin'(u)$ comme suit. Soit $i \stackrel{\text{def}}{=} r(u)$, $j \stackrel{\text{def}}{=} fin(u)$. Si u est une feuille de T, alors $D_T(u) = \{u\}$, donc $fin(u) = r(u)$, par la proposition 9, or $r(u) \leq fin'(u)$.

Sinon, $D_T(u) = \{u_i, u_{i+1}, \cdots, u_j\}$, et nécessairement $i < j$ (sinon $u = u_i$ serait une feuille de T). De plus, il existe un chemin de u à u_j dans T, et comme $i < j$, ce chemin est non vide, donc de la forme $u \to_T v \to_T^* u_j$. Par le lemme 14 (2), $fin(v) = fin(u)$, et par définition $fin(v) \leq fin'(u)$. □

En représentant chaque fonction `r`, `fin` par des tableaux indexés par les sommets, en supposant qu'ils soient identifiés par des numéros, et en codant l'ensemble `marked` par un tableau de bits, la complexité de `dfs_1` est $O(m+n)$, où n est le nombre de sommets de G et m son nombre d'arcs (somme pour chaque sommet `u` d'une constante, plus coût du parcours de `succ(u)`).

On peut ensuite décider de l'existence d'un circuit dans G en testant, dans une deuxième passe, s'il existe un arc arrière, en comparant pour chaque arc $u \to v$ les rangs et temps de fin de u et de v (table 1). Ceci prend un temps additionnel $O(m + n)$.

Plutôt que d'effectuer une deuxième passe, on peut aussi directement décider de l'existence d'un circuit au sein de l'algorithme `dfs_1`, en ajoutant une branche `else` (s'appliquant donc si `v` $\in$ `marked`) :

```
...else if u=v or (r(u)>r(v) and fin(u)≤fin(v)) then circuit := true;
```

Pour ceci, on doit, avant de lancer `dfs_1`$(*)$, initialiser `fin`(v) à une valeur strictement plus grande que n, et `circuit` à `false`. L'invariant est raffiné en demandant que pour tout sommet ouvert u de L_k, `fin`$(u) = \infty$.

5 Tri topologique

Une *extension* $\sqsubseteq$ d'une relation d'ordre $\leq$ sur un ensemble E est une relation d'ordre telle que pour tous $x, y \in E$, si $x \leq y$ alors $x \sqsubseteq y$. (Autrement dit, $\leq \subseteq \sqsubseteq$.) Une *extension totale* est une extension qui est une relation d'ordre totale. Le théorème de Szpilrajn énonce que tout ordre a une extension totale. En voici une démonstration rapide, et totalement non constructive. La famille des extensions de $\leq$ est un ensemble ordonné inductif, et a donc un élément maximal par le lemme de Zorn. Si $\sqsubseteq$ est une extension de $\leq$ mais n'est pas totale, il existe deux éléments x et y tels que $y \not\sqsubseteq x$. La relation $\sqsubseteq'$ définie par $a \sqsubseteq' b$ ssi $a \sqsubseteq b$, ou bien $a \sqsubseteq x$ et $y \sqsubseteq b$, est alors une extension stricte de $\sqsubseteq$. Ceci montre que toute extension non totale est non maximale. Donc l'extension maximale trouvée par le lemme de Zorn est totale.

extension

extension totale

Dans le cas où E est un ensemble fini, une extension totale $\sqsubseteq$ de $\leq$ est souvent appelée un *tri topologique*. Ceci s'applique notamment à l'ensemble des sommets d'un graphe G sans circuit, ordonnés par $\rightarrow^*$. (On a besoin que G soit sans circuit, ou au moins que ses seuls circuits soient des boucles, pour que $\rightarrow^*$ soit antisymétrique.) Un tel tri topologique peut se présenter sous la forme d'une énumération $x_1 \sqsubset x_2 \sqsubset \cdots \sqsubset x_n$ des éléments de E, donc d'une numérotation des éléments de E : le numéro d'un élément x est l'unique i tel que $x = x_i$.

tri topologique

Le parcours $L \stackrel{\text{def}}{=} [u_1, \cdots, u_n]$ en profondeur de G est un *tri* des sommets de G, et l'on pourrait penser que c'est un tri topologique $u_1 \sqsubset \cdots \sqsubset u_n$. Ceci est faux. Comme la table 1 le suggère, ceci échoue dès qu'il existe un arc transverse.

La démonstration du lemme 4.7 (BBC) donne la solution. Elle établit que si G est sans circuit, alors on peut ordonner les sommets par un ordre strict $\succ$. Cet ordre est *total*, car toute combinaison lexicographique d'ordre totaux est totale. De plus, on y a démontré que $u \rightarrow v$ implique $u \succ v$. Donc $\succeq$ est un tri topologique de G. (La propriété $u \rightarrow v \Rightarrow u \succ v$ est en fait un peu plus forte, car elle exclut l'existence de boucle.)

On peut calculer une numérotation des sommets associée au tri topologique $\succ$ grâce au résultat suivant.

Lemme 16 *Soit rto: $S \to \mathbb{N}$ n'importe quelle fonction telle que, pour tout sommet u,*

1. *pour tout $v \in S$ tel que $u \rightarrow_T v$, $rto(u) > rto(v)$;*
2. *pour tout $v \in S$ tel que $fin(v) < r(u)$, $rto(u) > rto(v)$.*

Si G est sans circuit, alors pour tous $u, v \in S$, $u \rightarrow v$ implique $rto(u) > rto(v)$.

Démonstration. Il n'y a que des arcs avant et des arcs transverses.

Si $u \rightarrow v$ est un arc avant, alors $v \in D_T(u)$, et comme G n'a pas de boucle, on a $v \neq u$, donc $u \rightarrow_T^+ v$. Par la condition 1, itérée autant de fois qu'il y a d'arcs de liaison de u à v, $rto(u) > rto(v)$.

Si $u \rightarrow v$ est un arc transverse, alors $fin(v) < fin(u)$. De plus, comme u et v sont incomparables pour $\rightarrow_T^*$, la proposition 11 nous indique que les intervalles $[r(v), fin(v)]$ et $[r(u), fin(u)]$ sont disjoints. Donc $fin(v) < r(u)$. On en déduit que $rto(u) > rto(v)$, par la condition 2. □

```
fun dfs_2 (u) =
    (* Pre I_2(dfsNum, marked, r, fin, rto, revTopOrder) et r(u) = dfsNum + 1 *)
    marked := marked ∪ {u};
    r(u) := ++dfsNum;
    for each v ∈ succ(u) do
        if v ∉ marked
           then dfs_2 (v);
    fin(u) := dfsNum;
    rto(u) := ++revToporder; (* calcul de rto(u) *)
    (* Post I_2(dfsNum, marked, r, fin, rto, revTopOrder) et dfsNum = fin(u) = fin(u) *)
```

TABLE 3 – Tri topologique

Toujours en supposant G sans circuit, on peut définir une telle fonction rto par :

$$rto(u) \stackrel{\text{def}}{=} \max(\max\{rto(v) \mid v \in S \text{ tel que } u \to_T v\}, \max\{rto(v)) \mid v \in S \text{ tel que } fin(v) < r(u)\}) + 1, \quad (1)$$

où, le cas échéant, le max d'une famille vide est considéré égal à 0. Cette définition est une définition par récurrence sur u le long de l'ordre $\succ$ de la démonstration du lemme 4.7 (BBC), autrement dit par récurrence (forte) sur $(fin(u), r(u))$ ordonné par le produit lexicographique de $>$ et de $<$. Informatiquement parlant, les appels « récursifs » à $rto(v)$ sur le côté droit sont tous tels que $fin(u) > fin(v)$, ou bien $fin(u) = fin(v)$ et $r(u) < r(v)$.

L'algorithme de la table 3, qui est obtenu à partir de `dfs_1` en ajoutant une seule ligne, étiquetée « calcul de rto(u) », et calcule $rto(u)$, pour chaque sommet u, en le stockant dans le tableau `rto`. La variable globale `revToporder` est initialisée à 0. L'invariant $I_2(\texttt{dfsNum}, \texttt{marked}, \texttt{r}, \texttt{fin}, \texttt{rto})$ est la conjunction de $I_1(\texttt{dfsNum}, \texttt{marked}, \texttt{r}, \texttt{fin})$ et de :

(iv) pour tout sommet fermé v de L_k, $\texttt{rto}(v) = rto(v)$,

(v) $\texttt{revTopOrder} = \max\{rto(v) \mid v \text{ sommet fermé de } L_k\}$,

où rto est définie en (1). (On rappelle que le max d'un ensemble vide vaut 0 par convention.)

Pour la même raison que pour `dfs_1`, `dfs_2 (*)` est un algorithme en temps $O(m+n)$, où n est le nombre de sommets de G, et m son nombre d'arcs.

6 Composantes fortement connexes

La relation d'accessibilité $\to^*$ est un préordre (une relation réflexive et transitive). Tout préordre a une relation d'équivalence associée, dans ce cas la relation $\equiv$ définie par $u \equiv v$ si et seulement si $u \to^* v$ et $v \to^* u$.

On notera $[u]$ la classe d'équivalence de u pour cette relation d'équivalence : c'est la *composante fortement connexe* de u dans le graphe G. Les composantes fortement connexes

composante fortement connexe

sont deux à deux disjointes.

On peut alors former le quotient $S/\equiv$, dont les éléments sont les composantes fortement connexes de G. De plus, on peut définir une relation $\Rightarrow$ sur $S/\equiv$ par : $C \Rightarrow C'$ si et seulement si $C \neq C'$ et il existe $u \in C$ et $v \in C'$ tels que $u \to v$ soit un arc de G. Le graphe $(S/\equiv, \Rightarrow)$ est alors sans circuit : c'est la *condensation* de G. condensation

Remarque 17 *Dans un circuit $u_0 \to u_1 \to \cdots \to u_n = u_0$ ($n \geq 1$), tous les sommets appartienent à la même composante fortement connexe. En effet, pour tous indices i et j ($0 \leq i, j \leq n$), disons $i \leq j$, on a $u_i \to^* u_j$, mais aussi $u_j \to^* u_n = u_0 \to^* u_i$. Donc $u_i \equiv u_j$.*

Lorsque G est un graphe qui n'a pas forcément de source, on note que $\{*\}$ forme une composante fortement connexe dans le graphe G_* à elle seule. Les composantes fortement connexes de G_* sont donc celles de G, plus $\{*\}$. On peut donc, de nouveau, supposer que G a une source $*$, sans perdre de généralité, si l'on souhaite calculer ses composantes fortements connexes.

Définition 18 *Soit L un parcours en profondeur de $G \stackrel{def}{=} (S, A)$, et $T \stackrel{def}{=} (S, *, p)$ l'arbre couvrant associé. Pour toute composante fortement connexe C de G, le* point d'entrée *$*_C$ de C est le sommet de plus petit rang de C.* point d'entrée

Lemme 19 *Pour toute composante fortement connexe C de G, $C \subseteq D_T(*_C)$.*

Démonstration. Soit $i \stackrel{\text{def}}{=} r(*_C)$, $j \stackrel{\text{def}}{=} fin(*_C)$. Par la proposition 9, les sommets de C sont exactement ceux de rangs compris entre i et j.

On le démontre par récurrence sur la longueur d'un chemin de $*_C$ à v. Si cette longueur vaut 0, $v = *_C$ est évidemment dans $D_T(*_C)$. Sinon, ce chemin est de la forme $*_C \to^* u \to v$, où le chemin $*_C \to^* u$ est strictement plus court. De plus, comme $v \equiv *_C$, il existe un chemin de v à $*_C$ dans G, dont aussi un chemin $u \to v \to^* *_C$: comme $*_C \to^* u$ et $u \to^* *_C$, u est donc aussi dans C. On peut dès lors appliquer l'hypothèse de récurrence, et en déduire que u est dans $D_T(*_C)$. On regarde maintenant les différentes formes possibles de l'arc $u \to v$:

- boucle : alors $v = u$, et donc trivialement $v \in D_T(*_C)$;
- arc avant : alors $u \to^*_T v$, et comme $*_C \to^*_T u$ (puisque $u \in D_T(*_C)$), on a $*_C \to^*_T v$ donc $v \in D_T(*_C)$;
- reste le cas où $u \to v$ est un arc arrière ou transverse, qui est l'unique cas intéressant. On a alors $r(v) < r(u)$ (voir la table 1). On utilise maintenant le fait que, par la proposition 9, les sommets de C sont exactement ceux de rangs compris entre $i \stackrel{\text{def}}{=} r(*_C)$ et $j \stackrel{\text{def}}{=} fin(*_C)$. Comme $u \in D_T(*_C)$, on a donc $i \leq r(u) \leq j$. Comme $v \in C$ et $*_C$ soit de rang minimal dans C, on a $i \leq r(v)$. Donc $i \leq r(v) \leq j$, ce qui signifie que v est dans $D_T(*_C)$. □

Lemme 4.8 (BBC, page 130, reformulé) *Pour toute composante fortement connexe C de G, $T_{|C} \stackrel{def}{=} (C, *_C, p_{|C \setminus \{*_C\}})$ est un arbre couvrant C, de racine $*_C$, et un sous-graphe induit de T.*

Ce qui est important, c'est que ceci veut dire qu'il y a *exactement un* point d'entrée de T dans C, qui est $*_C$.

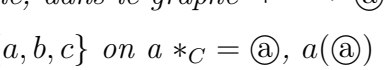
Démonstration. La seule chose qui ne soit pas une évidence est que, pour tout sommet $v \in C \setminus \{*_C\}$, $p(v)$ est encore dans C. Par le lemme 19, v est dans $D_T(*_C)$, autrement dit il y a chemin $*_C \to_T^* v$ (dans T, pas juste dans G). Ce chemin est unique, et comme $v \neq *_C$, il est de la forme $*_C \to_T^* p(v) \to_T v$.

Comme v et $*_C$ sont dans la même composante fortement connexe, on a aussi un chemin $v \to^* *_C$, donc un circuit $*_C \to_T^* p(v) \to_T v \to^* *_C$. Par la remarque 17, tous les sommets de ce circuit sont dans C, en particulier $p(v)$. □

Pour tout sommet u d'une composante fortement connexe C, le lemme 4.8 (BBC) montre que u est un sommet de l'arbre $T_{|C}$, et donc il existe un (unique) chemin $*_C \to_T^* u$. L'algorithme de Tarjan va calculer, pour chaque sommet u, un indice $\texttt{low}(u)$ qui est l'indice d'un sommet qu'on appelle son point d'attache $a(u)$. On pourrait penser que $a(u)$ est justement $*_C$, mais ce ne sera pas le cas en général ! En revanche, on verra que $u = a(u)$ si et seulement si $u = *_C$.

La définition du point d'attache est technique. Les circuits sans boucle partant de et arrivant en u (de composante fortement connexe $C \stackrel{\text{def}}{=} [u]$) sont tous de la forme $u \to_T^* v \hookrightarrow w \to^* u$, où $v \hookrightarrow w$ est le premier arc en arrière ou transverse du circuit. (On notera $\hookrightarrow$, en général, pour désigner un arc arrière ou transverse.) Il existe nécessairement un tel arc dans chacun de ces circuits : sinon on aurait $u \to_T^+ u$, et donc $r(u) < r(u)$, ce qui est absurde.

Lorsque l'on parcourt l'espace de tous les circuits de la forme $u \to_T^* v \hookrightarrow w \to^* u$ (avec ou sans boucle, peu importe), les sommets w ainsi obtenus forment un ensemble $AT(u)$. Le *point d'attache* $a(u)$ de u est le sommet $w \in AT(u) \cup \{u\}$ de plus petit rang. C'est donc le sommet de $AT(u)$ de plus petit rang s'il existe un $w \in AT(u)$ de rang inférieur à celui de u, sinon c'est u lui-même. point d'attache

Remarque 20 *La figure 4.8 du BBC peut donner l'impression que $a(u)$ est égal à $*_C$ pour tout sommet u de toute composante fortement connexe C. Ce n'est pas le cas. Par exemple, dans le graphe*  *, dans la composante fortement connexe $C \stackrel{def}{=} \{a, b, c\}$ on a $*_C = ⓐ$, $a(ⓐ) = a(ⓑ) = ⓐ$, mais $a(ⓒ) = ⓑ$.*

Lemme 21 *Pour toute composante fortement connexe C, pour tout sommet u de C, $a(u)$ est aussi dans C.*

Démonstration. C'est clair si $a(u) = u$. Sinon, $a(u)$ est un sommet w apparaissant dans un circuit $u \to_T^* v \hookrightarrow w \to^* u$, lequel ne contient que des sommets de C par la remarque 17. □

Lemme 4.9 (BBC, page 131) *Un sommet u est le point d'entrée $*_C$ de sa composante fortement connexe $C \stackrel{def}{=} [u]$ si et seulement si $u = a(u)$.*

Démonstration. Soit C la composante fortement connexe de u, et $i \stackrel{\text{def}}{=} r(u)$. Si $u = *_C$, alors tout circuit de la forme $u \to_T^* v \hookrightarrow w \to^* u$ ne contient que des sommets de C, par la

remarque 17. Comme w est dans C, et que $*_C$ est le sommet de plus petit rang dans C par définition, $i \leq r(w)$. Ceci étant vrai pour tous les sommets w ainsi obtenus, il s'ensuit que $a(u) = u$.

Réciproquement, supposons que $u = a(u)$, c'est-à-dire que dans tout circuit de la forme $u \rightarrow_T^* v \hookrightarrow w \rightarrow^* u$, on a $r(w) \geq i$. On va montrer que $u = *_C$.

En particulier, pour tout chemin de la forme $u \rightarrow_T^* v \hookrightarrow w \rightarrow^* *_C$, on a $r(w) \geq i$. En effet, un tel chemin induit un circuit $u \rightarrow_T^* v \hookrightarrow w \rightarrow^* *_C \rightarrow^* u$, puisque u est $*_C$ sont dans la même composante fortement connexe.

Comme u est dans C, il existe un chemin $u \rightarrow^* *_C$. Choisissons-en contenant le nombre minimal d'arcs arrières ou transverses. Si ce chemin $u \rightarrow^* *_C$ contient au moins un arc arrière ou transverse, alors on peut l'écrire $u \rightarrow_T^* v \hookrightarrow w \rightarrow^* *_C$. Dans ce cas, on a vu que $r(w) \geq i$. Comme $v \hookrightarrow w$ est arrière ou transverse, $r(w) < r(v)$ (voir la table 1). Par la proposition 9, les descendants dans T du sommet u sont exactement les sommets de rangs compris entre $r(u)$ et $fin(u)$; comme $u \rightarrow_T^* v$, on a donc $r(v) \leq fin(u)$. On en déduit que $r(w) < r(v) \leq fin(u)$; donc $i = r(u) \leq r(w) \leq fin(u)$, et donc, par la proposition 9 encore, $u \rightarrow_T^* w$. Mais alors $u \rightarrow_T^* w \rightarrow^* *_C$ est un chemin contenant un arc arrière ou transverse de moins, ce qui contredit la minimalité.

Il s'ensuit que le chemin $u \rightarrow^* *_C$ ne contient aucun arc arrière ou transverse. On en élimine les boucles, et il reste un chemin ne contenant que des arcs avant. Donc $u \rightarrow_T^* *_C$.

L'unique sommet de $T_{|C}$ dont $*_C$ est un descendant est $*_C$. Donc $u = *_C$. □

La proposition 22 ci-dessous fournit une définition équivalente de $r(a(u))$ (et donc de $a(u)$) par récurrence sur $n - r(u)$. En effet, pour tout arc avant $u \rightarrow v$, on a $r(u) < r(v)$ (voir la table 1).

Proposition 22 *Pour tout sommet u de G, le rang de $a(u)$ est l'entier le plus petit parmi :*

1. *$r(u)$;*
2. *$r(a(v))$, lorsque $u \rightarrow v$ parcourt les arcs de liaison d'origine u (autrement dit, $u \rightarrow_T v$) ;*
3. *$r(v)$, lorsque $u \rightarrow v$ parcourt les arcs arrières ou transverses d'origine u dont l'extrémité v est dans la même composante fortement connexe que u.*

Démonstration. Notons que dans le cas 2, on ne restreint pas v à être dans la même composante fortement connexe que u. Mais seuls les v qui sont dans la même composante fortement connexe que u comptent, car : $(*)$ si $u \rightarrow_T v$ et si u et v ne sont pas dans la même composante fortement connexe, alors $r(a(u)) < r(a(v))$. En effet, soit C la composante fortement connexe celle de v. Comme $u = p(v)$ n'est pas dans C, c'est que $v = *_C$, par le lemme 4.8 (BBC). Donc $r(u) < r(v) = r(*_C)$ (voir la table 1). Or, comme le point d'entrée $*_C$ est le sommet de rang minimal de C', et que $a(v)$ est dans C, on a $r(*_C) \leq r(a(v))$. Donc $r(u) < r(a(v))$. Or, par définition, $r(a(u)) \leq r(u)$, ce qui établit $(*)$.

Montrons le lemme. Soit C la composante fortement connexe de u.

Commençons par montrer que $r(a(u))$ est inférieur ou égal aux entiers mentionnés dans les cas 1, 2, et 3.

— Cas 1 : $r(a(u)) \leq r(u)$ est par définition du point d'attache.

— Cas 2. Soit $u \to_T v$ un arc de liaison. On prétend que $r(a(u)) \leq r(a(v))$. Si u et v ne sont pas dans la même composante fortement connexe, alors $r(a(u)) < r(a(v))$ par $(*)$. Sinon, $v \in C$, et l'on va construire un circuit d'origine et d'extrémité u, passant par v et $a(v)$, comme suit. D'abord, $r(a(v)) < r(v)$, sinon $a(v) = v$, ce qui impliquerait $v = *_C$, par le lemme 4.9 (BBC) ; mais $*_C$ étant la racine du sous-arbre $T_{|C}$, par le lemme 4.8 (BBC), $u \to_T v$ impliquerait que u ne serait pas dans C. Donc $r(a(v)) < r(v)$, et ceci implique qu'il existe un circuit $v \to_T^* v' \hookrightarrow a(v) \to^* v$, par définition du point d'attache. On peut alors former le circuit $u \to_T v \to_T^* v' \hookrightarrow a(v) \to^* v \to^* u$, où la dernière partie $v \to^* u$ vient du fait que u et v sont dans la même composante fortement connexe, C. Par la propriété de minimalité dans la définition du point d'attache de u, on a donc $r(a(u)) \leq r(a(v))$.

— Cas 3. Soit $u \to v$ un arc arrière ou transverse, avec $v \in C$. On prétend que $r(a(u)) \leq r(v)$. Comme $v \in C$, on a $v \to^* u$, ce qui nous permet de former un circuit $u \to v \to^* u$. Ceci est un circuit $u \to_T^* u \hookrightarrow v \to^* u$ (avec 0 arc de u à u), et la minimalité dans la définition du point d'attache de u implique donc que $r(a(u)) \leq r(v)$.

Il reste à montrer que $r(a(u))$ est supérieur ou égal à l'un des entiers des cas 1, 2, ou 3. Il sera donc égal au plus petit de ces entiers.

— Si $a(u) = u$ (c'est-à-dire si $u = *_C$), alors $r(a(u)) = r(u)$ est égal à l'entier du cas 1. Dans la suite, supposons que $a(u) \neq u$, donc $a(u)$ est un sommet w tel que $r(w) < r(u)$, et il existe un circuit de la forme $u \to_T^* v' \hookrightarrow w \to^* u$.

— Si $u = v'$, d'abord, par la remarque 17, v' est dans C. On pose $v \stackrel{\text{def}}{=} w$: c'est un sommet du cas 3. Et $r(a(u)) = r(w) \geq r(v)$, trivialement.

— Sinon, le chemin de u à v' est non vide, et notre circuit s'écrit donc $u \to_T v \to_T^* v' \hookrightarrow w \to^* u$. Le sommet v est un sommet du cas 2. Il ne reste qu'à montrer que $r(a(u)) = r(w)$ est supérieur ou égal à $r(a(v))$. Pour ceci, on fait « tourner le circuit d'un cran », et l'on produit le circuit $v \to_T^* v' \hookrightarrow w \to^* u \to_T v$. La minimalité dans la définition du point d'attache $a(v)$ de v implique alors que $r(a(v)) \leq r(w)$. □

La proposition 22 montre comment calculer $a(u)$, récursivement… à condition de savoir déterminer si u et v sont dans la même composante fortement connexe dans le cas 3. Ceci sera effectué dans l'algorithme de Tarjan en testant si v est sur une pile gardée dans une variable nommée `stk`. En fait, comme on le verra à la proposition 24, les sommets de cette pile sont ceux qui sont dans la même composante fortement connexe que v, et aussi ceux de $D_T(u)$. On pourrait exclure ce dernier cas en remplaçant le test '`v ∈ stk`' de la ligne 9 de l'algorithme à venir (table 4) par '`v ∈ stk and r(v)<r(u)`'. Mais ce test est inutile, grâce à l'affaiblissement suivant de la proposition 22.

Proposition 23 *Pour tout sommet u de G, le rang de $a(u)$ est l'entier le plus petit parmi :*

1. *$r(u)$;*
2. *$r(a(v))$, lorsque $u \to v$ parcourt les arcs de liaison d'origine u (autrement dit, $u \to_T v$) ;*
3. *$r(v)$, lorsque $u \to v$ parcourt :*

(a) *les arcs arrières ou transverses d'origine u dont l'extrémité v est dans la même composante fortement connexe que u ;*

(b) *et les arcs avants, ainsi que les boucles, d'origine u.*

Démonstration. Il suffit de montrer que les arcs avants (ou les boucles) d'origine u ne contribuent pour rien dans le minimum de la proposition 22. Il suffit pour cela d'observer que pour tout arc avant (ou boucle) $u \to v$, on a $u \to_T^* v$ donc $r(u) \leq r(v)$. Dès lors, les valeur $r(v)$ du cas 3(b) sont toutes plus grandes que la valeur du cas 1. □

Rappelons que nous supposons un parcours en profondeur $L \stackrel{\text{def}}{=} [u_1, \cdots, u_n]$. Pour tout sommet u du graphe G, on peut regarder la suite des sommets u, $p(u)$, ..., $p^i(u) = *$. Prise à l'envers, c'est l'unique chemin de $*$ à u dans T : $* = u_{i_1} \to_T u_{i_2} \to_T \cdots \to_T u_{i_\ell} = u$, avec $1 = i_1 < i_2 < \cdots < i_\ell = k$. On appellera ce chemin la *branche* menant au sommet u. Par définition, l'un des sommets u_{i_j} est le point d'entrée $*_C$ de la composante fortement connexe de u. branche

Pour tout k ($0 \leq k \leq n$), notons π_k la sous-liste de L_k formée des éléments u_i qui sont dans la même composante fortement connexe qu'un des éléments de la branche $* = u_{i_1} \to_T u_{i_2} \to_T \cdots \to_T u_{i_\ell} = u_k$ menant à u_k. Ce n'est pas l'union des composantes fortement connexes de u_{i_1}, u_{i_2}, ..., u_{i_ℓ}, mais c'est l'intersection de cette union avec l'ensemble des éléments de L_k, listée dans le même ordre que dans L_k.

On appellera π_k la *pile à profondeur k*. Ce sera ce qui sera stocké dans la pile `stk` de l'algorithme de Tarjan plus bas. pile à profondeur k

Proposition 24 *Soit u un sommet de G, $i \stackrel{def}{=} r(u)$, et $k \in [i, fin(u)]$. Pour tout arc $u \to v$ de source u :*

1. *si $u \equiv v$ et $v \in L_k$, alors v est dans π_k ;*
2. *si v est dans π_k, alors v est dans L_k, et $u \equiv v$ ou bien $u \to_T^* v$.*

Démonstration. Notons $* = u_{i_1} \to_T u_{i_2} \to_T \cdots \to_T u_{i_\ell} = u_k$ la branche menant à u_k (donc $i_\ell = k$). Par la proposition 9, et comme $k \in [i, fin(u)]$, u_k est dans l'ensemble $D_T(u)$ des descendants de u dans T. Donc $u = p^m(u_k)$ pour un certain entier $m \in \mathbb{N}$, d'où l'on déduit que $i = r(u) = i_{\ell-m}$. En d'autres termes, $u = u_{i_j}$ pour un certain entier j, $1 \leq j \leq \ell$.

1. Si $u \equiv v$, alors v est dans la même composante fortement connexe que u_{i_j}, et comme il est dans L_k, il est donc dans π_k.

2. Supposons maintenant que v est dans π_k. Alors $v \equiv u_{i_{j'}}$ pour un certain j', $1 \leq j' \leq \ell$. Choisissons j' minimal. En notant C la composante fortement connexe de v, $u_{i_{j'}}$ est alors la racine $*_C$ de l'arbre $T_{|C}$ couvrant C, par le lemme 4.8 (BBC). En effet, sinon $p(u_{i_{j'}}) = u_{i_{j'-1}}$ serait aussi dans $T_{|C}$, donc dans C, ce qui contredirait la minimalité de j'.

Si $j' \geq j$, on a un chemin $u = u_{i_j} \to_T^+ u_{i_{j'}} = *_C \to_T^* v$ (puisque v est dans C, donc apparaît comme un sommet de T_C).

Si $j' < j$, alors on a $v \to^* u_{i_{j'}}$ (puisque $v \equiv u_{i_{j'}}$) $\to_T^* u_{i_j} = u$. Or, comme v est un successeur de u dans G, on a aussi $u \to v$, donc $u \equiv v$. □

```
   fun scc (u) =
   (* Pre I3(dfsNum, marked, r, stk, low) et r(u) = dfsNum + 1 et Pre(stk, u) *)
1        marked := marked ∪ {u};
2        r(u) := ++dfsNum;
3        low(u) := dfsNum;
4        push(stk,u);
5        for each v ∈ succ(u) do
6            if v ∉ marked
7               then (scc (v);
8                     low(u) := min(low(u),low(v)));
9            else if v ∈ stk then low(u) := min(low(u),r(v));
10       if r(u)=low(u) (* composante trouvée *)
11          then do v := pop(stk); c(v):=u until v=u; (* on dépile la composante *)
   (* Post I3(dfsNum, marked, r, stk, low) et dfsNum = fin(u) et Post(stk, u) *)
```

TABLE 4 – L'algorithme de Tarjan

7 Calcul des composantes fortement connexes

algorithme de Tarjan

L'*algorithme de Tarjan* du calcul des composantes fortement connexes de G (toujours supposé avec origine $*$) consiste à appeler `scc (*)`, où `scc` est défini en table 4. Il remplit un tableau `c`. À la fin du calcul, pour tout sommet u de G, `c`(u) vaudra le point d'entrée $*_C$ de sa composante fortement connexe C. Ceci permettra de décider ensuite très rapidement si deux sommets u et v sont dans la même composante fortement connexe : ils suffira de tester si `c`$(u) =$ `c`(v).

La procédure `scc` est une modification de `dfs_1`, où :

- d'abord, on ne calcule plus les temps de fin (mais on pourrait), et surtout,
- on calcule un nouveau tableau `low`, qui à chaque sommet va associer le rang dans L de son point d'attache ;
- on maintient une pile `stk`, sur laquelle on empile `u` (ligne 4) ; crucialement, on ne dépile pas `u` à la fin de la boucle des lignes 5–9 : le but est d'accumuler tous les sommets de la composante fortement connexe [`u`] de `u` à la suite de `u` dans `stk` ;
- on ne dépile `stk` qu'aux lignes 10–11, lorsqu'on peut garantir que *tous* les sommets de [`u`] ont été empilés à la suite de `u` dans `stk` ;
- on aura besoin de tester rapidement l'appartenance d'un sommet à la pile `stk` en ligne 9 (non, on n'effectuera pas un parcours linéaire des éléments de `stk` ; voir plus bas).

La pile `stk` peut être implémentée par :

- un tableau `s` de sommets, de taille n ;
- un compteur `k` du nombre d'éléments dans `s` ;

— un tableau `onStk` de bits, de taille n ;

Ce dernier tableau sera tel que `onStk`$[u]$ sera vrai, pour tout sommet u, si et seulement si u est sur la pile. La pile est initialisée à `k` := 0, et toutes les entrées de `onStk` sont mises à 0. Ceci prend un temps $O(n)$. L'empilement `push(stk,u)` s'implémente par : `s[k++] := u; on[u]:=true`. Le dépilement `pop(stk)` s'implémente par `let v = s[--k] in (on[v] := false; v)`. Le test `v` $\in$ `stk` s'implémente par `onStk[v]`. Ces trois opérations sont en temps constant.

Une pile `stk` dénote une liste de sommets P, la pile vide dénotant la liste vide [], et le résultat de l'empilement de u sur une pile dénotée par $P \stackrel{\text{def}}{=} [u_1, \cdots, u_i]$ étant $P :: u \stackrel{\text{def}}{=} [u_1, \cdots, u_i, u]$. Par $I_3(\texttt{dfsNum}, \texttt{marked}, \texttt{r}, \texttt{stk}, \texttt{low})$, on entend l'invariant conjonction des propriétés **(i)** et **(ii)** utilisées pour définir I_1 (la condition **(iii)** n'a plus de sens, puisque le tableau `fin` n'est plus présent ici), et de :

(vi) pour tout sommet $u \in$ `marked`, `low`(u) est l'entier le plus petit parmi :

1. $r(u)$;
2. `low`(v), lorsque $u \rightarrow_T v$ parcourt les arcs de liaison d'origine u *et* dont l'extrémité v est dans `marked` ;
3. $r(v)$, lorsque $u \rightarrow v$ parcourt les arcs arrières ou transverses d'origine u dont l'extrémité v est dans la même composante fortement connexe que u. (On notera que v est alors nécessairement dans `marked`, car $r(v) < r(u)$, voir la table 1.)

On demande de plus qu'en entrée la formule $Pre(\texttt{stk}, \texttt{u})$ suivante soit satisfaite :

— si P est la pile dénotée par `stk`, alors $P :: \texttt{u} = \pi_k$, où $k \stackrel{\text{def}}{=} r(\texttt{u})$.

En sortie, on demande que la formule $Post(\texttt{stk}, \texttt{u})$ suivante soit satisfaite :

— si $\texttt{u} = a(\texttt{u})$, alors $Pre(\texttt{stk}, \texttt{u})$;

— sinon, alors `stk` dénote la pile π_k, où k est le rang maximal d'un sommet v dans la même composante fortement connexe que `u` et tel que $\texttt{u} \rightarrow_T^* v$.

Voici un aperçu sommaire de l'argument de correction.

En ligne 7, l'arc `u` $\rightarrow$ `v` est nécessairement un arc de liaison, comme dans `dfs_1`. En ligne 9, le test `v` $\in$ `stk` réussit si et seulement si `v` est dans π_k, où (grâce à la postcondition) k est dans un intervalle permettant d'utiliser la proposition 24, et donc, les sommets `v` qui réussissent ce test sont exactement ceux qui sont dans la même composante fortement connexe que `u`. À la fin de la boucle des lignes 5–9, on a donc établi l'invariant **(vi)** pour le sommet $u \stackrel{\text{def}}{=}$ `u`.

En ligne 10, l'invariant **(vi)** se simplifie, car tous les sommets v tels que $\texttt{u} \rightarrow_T v$ sont dans `marked`. Ceci est dû au fait que `dfsNum` $= fin(v)$ à la fin de l'appel récursif `scc`(v) (ligne 7), ceci étant dû à la postcondition de `scc`, et au fait que I_3 assure que `marked` contient tous les sommets de L_k avec $k \stackrel{\text{def}}{=}$ `dfsNum`. On reconnaît alors en l'invariant **(vi)** les trois cas de la proposition 23. En conséquence, en ligne 10, `low`(`u`) est égal à $r(a(\texttt{u}))$. Comme dans l'algorithme `dfs_1`, `r`(`u`) $= r(\texttt{u})$. Donc le test `r(u)=low(u)` de la ligne 10 réussit si et seulement si `u` est le point d'entrée $*_C$ de sa composante fortement connexe C.

La propriété $Post(\texttt{stk}, \texttt{v})$ assurée par l'appel récursif de la ligne 7 (pour chaque `v` tel que $\texttt{u} \to \texttt{v}$ par un arc avant, donc) permet de conclure qu'à la ligne 10, `stk` représente la pile π_k où k est le rang maximal d'un sommet v dans la même composante fortement connexe que `u` et tel que $\texttt{u} \to_T^* v$. On trouve donc en suffixe de π_k tous les descendants de `u` dans T, par le lemme 4.8 (BBC). Si le test de la ligne 10 réussit, la pile π_k représentée par `stk` contient donc comme suffixe tous les sommets de C, avec $*_C$ comme premier sommet. Ceci assure la validité de $Post(\texttt{stk}, \texttt{u})$ en fin de procédure, que le test de la ligne 10 réussisse ou non.

Finalement, la complexité de `scc` est la même que `dfs_1`, à savoir $O(m+n)$, où n est le nombre de sommets de G et m son nombre d'arcs, à laquelle on doit rajouter la complexité induite par les lignes 10–11. Comme les opérations de pile sont en temps constant, cette complexité additionnelle est proportionnelle au nombre de sommets jamais dépilés de `stk` au cours de toute la procédure, et ceci vaut n, chaque composante fortement connexe C étant dépilée exactement une fois, et chaque sommet `v` de C étant dépilé exactement une fois.

In fine, la complexité de `scc` est, encore une fois, de $O(m+n)$.