Tri topologique

Julie Parreaux

2018-2019

Référence du développement : Cormen [1, p.566]

Leçons où on présente le développement : 903 (Algorithmes de tri) ; 925 (Algorithmes de graphe).
Leçon où on peut l'évoquer : 927 (Correction et terminaison)

1 Introduction

Le tri topologique est un tri avec un ordre partiel donnée par un graphe orienté acyclique. En effet, un graphe orienté acyclique définit une relation d'ordre partielle. Cependant, avec une relation d'ordre partielle ne permet pas d'appliquer les algorithmes de tri classique. Le tri topologique tri en fonction des contraintes sur les données (introduites par le graphe) grâce au parcours en profondeur. Nous donnons ici l'algorithme du tri topologique, sa terminaison, sa correction et sa complexité.

Remarques sur le développement

1. Algorithme du tri topologique.
2. Terminaison de cet algorithme.
3. Complexité de cet algorithme.
4. Correction de cet algorithme.
5. Ajout à la détection de cycle si le temps le permet.

2 Présentation du tri topologique

Hypothèse : on considère un graphe orienté acyclique (un graphe orienté acyclique définit alors une relation d'ordre partielle sur les sommets du graphe).

Soit $G = (V, E)$ un graphe acyclique. On cherche L une liste de tous les éléments de V tels que $\forall (u, v) \in A$, u apparaît avant v dans L (on dit une liste topologique : on voit apparaître l'ordre partiel). Les algorithmes 1 et 2 réalisent un tri topologique. On n'utilise pas tout à fait l'algorithme donné dans [1] : on le déroule. On prend la partie du parcours en profondeur [1, p.558] qui nous intéresse (on enlève les notions de dates) et on ajoute le traitement spécifique du tri topologique.

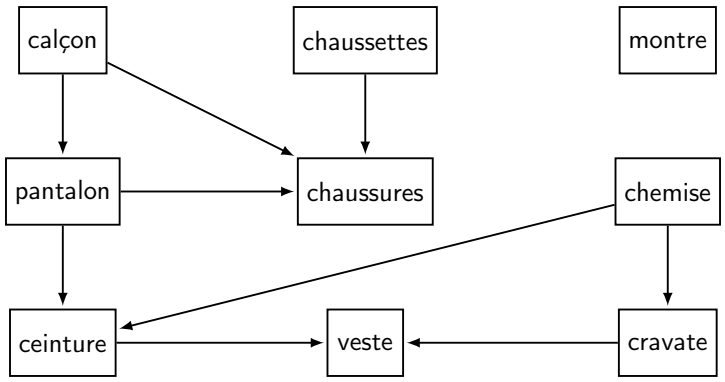


FIGURE 1 – Le tri topologique sur ce graphe nous donne : chaussettes, caleçon, pantalon, chaussures, montre, chemise, ceinture, cravate, veste.

Algorithm 1 Le tri topologique d'un graphe orienté acyclique.

```
1:  function TRI-TOPOLOGIQUE(G)
2:      L ← []                                  ▷ G est un graphe
3:      Colorier les sommets de G en blanc
4:      for tout u ∈ G do
5:          if u est en blanc then
6:              VISITE(G, u)
7:          end if
8:      end for
9:      Renvoie L
10: end function
```

Algorithm 2 Visiter un sommet : traiter ce sommet lors du parcours

```
1:  function VISITE(()G, u)
2:      Colorier u en gris
3:      for tout v ∈ Adj(u) do
4:          if v est de couleur blanc then
5:              VISITE(G, u)
6:          end if
7:      end for
8:      Colorier u en noir                      ▷ correction
9:      L ← u :: L
10: end function
```

Théorème (Terminaison). *L'algorithme du tri topologique (algorithme 1) termine.*

Démonstration. On effectue au plus $|V|$ appels à VISITE (algorithme 2). Comme le nombre d'appel à VISITE est borné, le nombre d'appel récursif l'est également d'où la terminaison. □

Remarque. Le nombre de sommets blancs décroît strictement. Donc le tri topologique termine.

Théorème (Correction). *L'algorithme du tri topologique (algorithme 1) est correct.*

Démonstration. Notons que l'on fait un parcours en profondeur. On va alors prouver trois propriétés permettant d'assurer la correction.

Propriété 1 : chaque sommet est ajouter une et une seule fois à la liste. En effet, on ajoute le sommet à la liste à la fin de VISITER. On appelle une seule fois VISITER pour chacun des sommets blanc qui sont directement colorier en gris au début de la fonction (ils ne sont plus jamais blanc ensuite).

Propriété 2 : à chaque début et fin de VISITER, L contient exactement les sommets noir. En effet, on les ajoute après les avoir mis en noir et avant de sortir de VISITER.

Propriété 3 : pour toute arête $(u, v) \in E$, v se retrouve avant u dans L (considération temporelle). Soit $(u, v) \in E$ explorée par le tri. Lors de l'appel à VISITER, trois cas sont possibles.

- Si v est noir, alors $v \in L$ donc v est dans L avant u.
- Si v est blanc, alors VISITE(G, v) se termine avant l'ajout de u dans L donc v est dans L avant u.
- Si v est gris, alors il existe un chemin de v à u (se montre par récurrence) et un arc de u à v. On a une contradiction avec l'hypothèse d'acyclicité.

Par ces trois propriété, on a montré la correction de l'algorithme du tri topologique. □

Complexité : on effectue essentiellement un parcours en profondeur (on n'ajoute aucune opération coûteuse), sa complexité est alors la même que le parcours en profondeur : $O(|V| + |E|)$ si G est en liste d'adjacence.

Remarque. On peut alors ajouter la détection de cycle : si le sommet visiter tombe sur un sommet gris on a un cycle.

Quelques applications du tri topologique :

- définition non circulaire de mots dans un dictionnaire s'utilisant mutuellement pour se définir (contre-exemple : algorithme - modèle de calcul - procédure effective ;)) ;
- unité d'enseignement (on essaye que les élèves on les prérequis nécessaire lorsqu'ils assistent à un cours) : plus généralement, la notion de prérequis ;
- l'outil make et plus généralement l'ordonnancement de tâches non circulaire ;
- implémenter comme pré-traitement d'un plus court chemin (remplace la file de priorité).

Remarque.
- Si le graphe est cyclique, la correction n'est plus assurée.
- Si le graphe n'est pas connexe alors l'algorithme de tri ne pose aucun problème.

3 Autours du parcours de profondeur

Stratégie du parcours : Parcourir l'ensemble des nœuds du graphe en privilégiant la profondeur : on va le plus loin possible dans le dit graphe.

Astuces : On utilise une coloration des nœuds du graphe afin de savoir quels sont les nœuds que nous avons déjà pris : pas étudier, en cours, fini. Cette coloration nous permet de nous assurer d'avoir parcouru tous les nœuds et de l'avoir fait une unique fois. On utilise aussi des dates de début et de fin de traitement pour chacun des nœuds. Ces dates jouent le même rôle que les couleurs et peuvent être exploité dans certaines applications comme le calcul de composantes fortement connexes. Ces deux astuces ne sont pas obligatoirement utiliser en même temps même si l'une ou l'autre peut ainsi simplifier les preuves de correction, de terminaison ou de complexité.

Applications du parcours en profondeur :

- Tri topologique : à la sortie de VISITER, on ajoute le sommet que l'on vient de visiter dans une liste.
- Composantes fortement connexes : on effectue deux passages de l'algorithme (un sur le graphe et le deuxième sur le graphe complémentaire dont l'ordre de l'étude des sommets est donné par les dates calculées lors du premier parcours.)

Présentation de l'algorithmique autours du parcours en profondeur de graphe Nous présentons ici un algorithme récursif (Algorithme 3) du parcours de graphe en profondeur [1, p.558]. Cet algorithme fait appelle à une fonction auxiliaire (qui porte la récursivité) VISITE (Algorithme 4) qui traite chacun des sommets du graphe et décide de l'ordre dans lequel on les traites. Nous prouverons également la correction et la terminaison de cet algorithme. Nous finirons par étudier sa complexité. La figure 2 nous donne un exemple d'application de l'algorithme de parcours en profondeur.

Algorithm 3 Le parcours en profondeur d'un graphe.

```
1:  function PARCOURS-PROFONDEUR(G)
2:      for tout u ∈ G.V do
3:          u.couleur ← blanc
4:          u.π ← NIL                    ▷ Prédécesseurs
5:      end for
6:      date ← 0
7:      for tout u ∈ G.V do
8:          if u.couleur = blanc then
9:              VISITE(G, u)
10:         end if
11:     end for
12: end function
```

Algorithm 4 Visiter un sommet : traiter ce sommet lors du parcours

```
1:  function VISITE(G, u)
2:      date ← date + 1          ▷ u juste découvert
3:      u.d ← date
4:      u.couleur = gris
5:      for tout v ∈ Adj(u) do
6:          if v.couleur = blanc then
7:              v.π ← u
8:              VISITE(G, u)
9:          end if
10:     end for
11:     u ← noir
12:     date ← date + 1
13:     u.f ← date
14: end function
```

Théorème (Terminaison)**.** *Soit $G = (V, E)$ un graphe. Le parcours en profondeur sur ce graphe termine.*

Démonstration. La terminaison provient du fait qu'on appel la fonction Visite au plus $|V|$. □

Remarque. L'ordre dans lequel les sommets sont examinés influe sur la sortie de l'algorithme. Mais en pratique, cet ordre nous importe peu car les sorties possibles sont équivalentes.

Théorème (Complexité)**.** *Soit $G = (V, E)$ un graphe. Le parcours en profondeur sur ce graphe se fait en $O(|V| + |E|)$.*

Démonstration. On applique la méthode de l'agrégat. On applique exactement $|V|$ fois la procédure Visite. De plus, pendant tout le parcours, la boucle dans Visite s'exécute $\sum_{v \in S} |Adj(v)| = \Theta(|E|)$. Le temps d'exécution du parcours en profondeur est $\Theta(|S| + |E|)$. □

Quelques propriétés que l'on peut déduire du parcours Le parcours en profondeur d'un graphe révèle quelques unes de ces structures.

Théorème (Théorème des parenthèses)**.** *Dans un parcours en profondeur d'un graphe $G = (V, E)$, pour deux sommets quelconques u et v, une et une seule des trois conditions suivantes est vérifiée :*

- *les intervalles $[u.d, u.f]$ et $[v.d, v.f]$ sont disjoints, et ni u ni v n'est un descendant de l'autre dans la forêt de parcours en profondeur ;*
- *l'intervalle $[u.d, u.f]$ est entièrement inclus dans l'intervalle $[v.d, v.f]$, et ni u est un descendant de v dans un arbre de parcours en profondeur ;*
- *l'intervalle $[v.d, v.f]$ est entièrement inclus dans l'intervalle $[u.d, u.f]$, et ni v est un descendant de u dans un arbre de parcours en profondeur.*

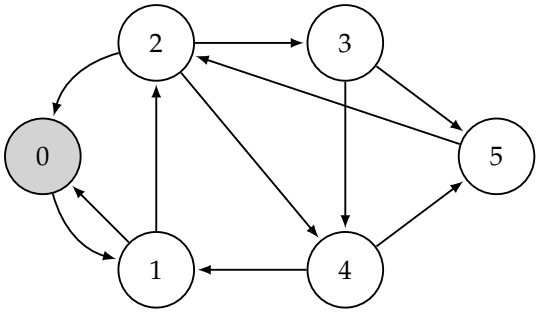
Démonstration. Supposons que $u.d < u.f$. On distingue alors deux sous-cas. Si $v.d < u.f$, alors v a été découvert pendant que u était encore gris (u est cours de traitement). Donc v est un descendant de u. Puisque la découverte de v est plus récente que u, le traitement de v se termine avant le traitement de u. Donc, l'intervalle $[v.d, v.f]$ est entièrement inclus dans l'intervalle $[u.d, u.f]$.

Dans l'autre sous-cas, $u.f < v.d$ ce qui implique que $u.d < u.f < v.d < v.f$ et donc les intervalles $[u.d, u.f]$ et $[v.d, v.f]$ sont disjoints. Comme les intervalles sont disjoints, aucun des deux sommets n'a été découvert pendant que l'autre était gris. Donc, aucun sommet n'est un descendant de l'autre.

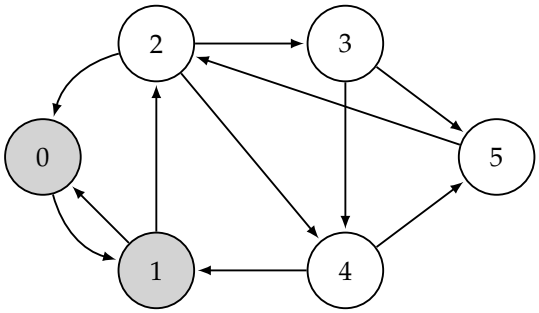
Dans le cas $v.d < u.d$, on raisonne de manière analogue en inversant les rôles de u et v. □

Corollaire (Imbrication des intervalles descendants)**.** *Le sommet v est un descendant propre du sommet u dans la forêt de parcours en profondeur d'un graphe G si et seulement si $u.d < v.d < v.f < u.f$.*

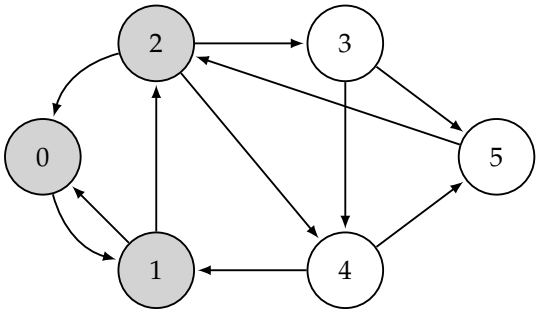
Démonstration. Immédiat par le théorème précédent. □



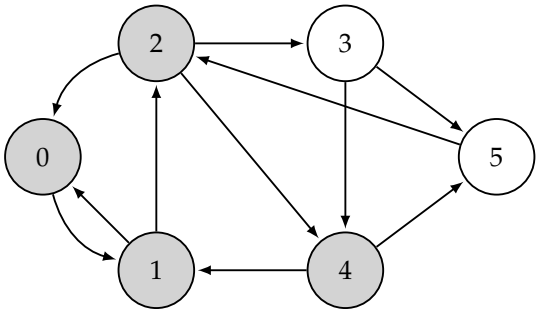
(a) Graphe sur lequel on applique le parcours en profondeur



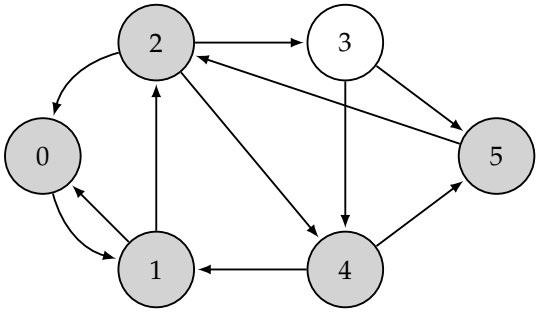
(b) Visite du sommet 1.



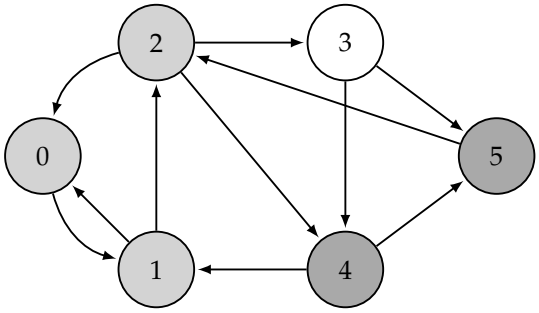
(c) Visite du sommet 2.



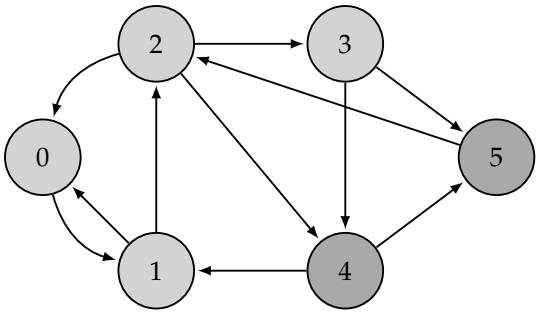
(d) Visite du sommet 4.



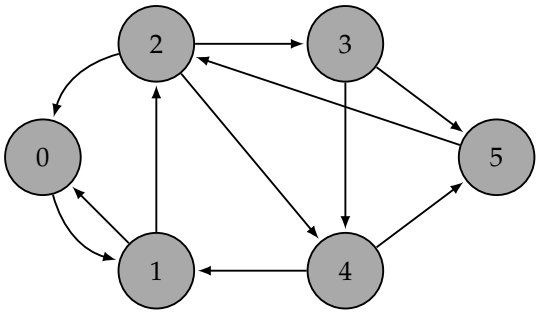
(e) Visite du sommet 5.



(f) Fin des visites des sommets 4 et 5.



(g) Visite du sommet 3.



(h) Fin des visites des sommets 3, 2, 1 et 0.

FIGURE 2 – Le parcours en profondeur d'un graphe. On obtient alors : 0, 1, 2, 4, 5 et 3.

Théorème (Théorème du chemin blanc)**.** *Dans une forêt de parcours en profondeur d'un graphe $G = (V, E)$, un sommet v est un descendant d'un sommet u si et seulement si, au moment $u.d$ où le parcours découvre u, il existe un chemin de u à v composé uniquement de sommets blancs.*

Démonstration. $\Rightarrow$: Si $v = u$, alors le chemin de u à v contient uniquement le sommet u, qui est encore blanc quand on définit la valeur de $u.f$. Supposons que v soit un descendant propre de u dans la forêt de parcours en profondeur. D'après ca qui précède, $u.d < v.d$ et donc v est blanc à l'instant $u.d$. Comme v peut être un descendant quelconque de u, tous les sommets situés sur le chemin simple unique entre u et v dans la forêt de parcours sont blanc à l'instant $u.d$.

$\Leftarrow$: Supposons qu'à l'instant $u.d$ il existe un chemin constitué de sommets blanc entre u et v, mais que v ne devient pas un descendant de u dans l'arbre de parcours en profondeur. Sans perte de généralité, on suppose que tous les autres sommets de ce chemin (sauf v) deviennent un descendant de u dans l'arbre. Si ce n'est pas le cas, on choisi le plus proche de u qui vérifie cette propriété. Soit w le prédécesseur de v dans ce chemine (u et w peuvent être le même chemin). D'après ce qui précède, $w.f \leq u.f$. Comme v doit être découvert après u, mais avant que le traitement de w soit terminé, on a $u.d < v.d < w.f \leq u.f$. Le théorème des intervalles implique alors que l'intervalle $[v.d, v.f]$ est entièrement inclus dans l'intervalle $[u.d, u.f]$. Donc, par le corollaire, v est un descendant de u. □

Classification des arcs Le parcours en profondeur peut permettre de classer les arcs du graphe $G = (V, E)$. Ce type peut fournir des informations sur le graphe.

Définition (Classification des arcs)**.** Soit $G = (V, E)$ un graphe. Le parcours en profondeur donne une classification des arcs du graphe.

1. Les arcs de liaison sont les arcs de la forêt de parcours en profondeur. L'arc (u, v) est un arc de liaison si v a été découvert la première fois pendant le parcours de l'arc (u, v).
2. Les arcs arrière sont les arcs (u, v) reliant un sommet u à un ancêtre v dans un arbre de parcours en profondeur. Les boucles (dans un graphes orienté) sont considérées comme des arcs arrière.
3. Les arcs avant sont les arcs (u, v) qui ne sont pas des arcs de liaison et qui relient un sommet u à un descendant v dans un arbre de parcours en profondeur.
4. Les arcs transverse sont tous les autres arcs. Ils peuvent relier deux sommets d'un même arbre de parcours en profondeur, du moment que l'un des sommets n'est pas un ancêtre de l'autre. Ils peuvent aussi deux sommets appartenant à des arbres de parcours en profondeur différents.

Lors du parcours en profondeur du graphe, on peut déterminer la nature de l'arc. Lorsque l'arc est exploré pour la première fois :

1. blanc indique un arc de liaison ;
2. gris indique un arc arrière ;
3. noir indique un arc avant ou de transverse.

Proposition. *Dans un parcours en profondeur d'un graphe non orienté G, chaque arête de G est soit un arc de liaison, soit un arc arrière.*

Démonstration. Soit (u, v) un arc de G tel que $u.d < v.d$. Alors, v doit être découvert et son traitement terminé avant le traitement de u (pendant que u est gris) (v se trouve dans la liste d'adjacence de u). Si l'arête (u, v) est d'abord exploré dans le sens u vers v, alors v n'a pas été découvert (blanc) jusqu'ici (sinon on aurait déjà exploré cet arête dans la direction de v vers u). Dans ce cas, (u, v) est un arc de liaison.

Si (u, v) est exploré dans l'autre sens, on a un arc arrière car u est encore gris lors de la première exploration. □

Proposition. *Un graphe G est acyclique si et seulement si un parcours en profondeur de G ne génère aucun arc arrière.*

Démonstration. $\Rightarrow$: Supposons qu'un parcours en profondeur produise un arc arrière. Alors, le sommet v est un ancêtre du sommet u dans la forêt de parcours en profondeur. Donc, il existe un chemin v à u dans G et l'arc arrière complète le cycle partant de v.

$\Leftarrow$: Supposons que G contienne un cycle c. On va montrer qu'un parcours en profondeur de G génère un arc arrière. Soit v le premier sommet découvert dans c et soit (u, v) l'arc précédent dans c. A l'instant $v.d$, les sommets de c forment un chemin entre v et u composé de sommets blanc. D'après le théorème du chemin blanc, le sommet u devient un descendant de v dans la forêt de parcours en profondeur. Donc (u, v) est donc un arc arrière. □

Structures de données et algorithmes affiliés Le parcours en profondeur peut être écrit de manière impérative à l'aide d'une pile. Dans le cas de l'algorithme récursif, la pile est caché dans le boucle pour et de la fonction Visite.

Références

[1] Rivest R. Stein C. Cormen T., Leiserson C. *Algorithmique, 3ème édition*. Dunod, 2010.