Tri topologique

Clarence Kineider

Leçons : 903, 925, 927

Référence(s) : Cormen, Leiserson, Rivest, Stein, *Introduction à l'algorithmique.*

Le problème du tri topologique est un problème d'ordonnancement des tâches : on a une liste de choses à faire, certaines ont besoin d'être faites avant d'autres. On représente ces contraintes par un graphe orienté dont les sommets sont les tâches et les arêtes sont les couples (u, v) tels que la tâche u doit être effectuée avant la tâche v. On cherche alors un ordre sur les sommets tel que si (u, v) est une arête, alors u apparait avant v dans l'ordre. Le problème n'a pas de solution s'il y a un cycle dans le graphe, on suppose donc le graphe acyclique. On peut formaliser le problème comme suit :

Tri topologique	Entrée : Un graphe orienté acyclique $G = (V, E)$ Sortie : Une liste L constituée des sommets de G telle que si $(u, v) \in E$, alors u apparaît avant v dans L

Remarque :
Il n'y a pas unicité de la solution (contrairement à un tri classique sur des valeurs distinctes).

Une façon de réaliser un tri topologique de G est de faire un parcours en profondeur, et de placer les sommets dans la liste à la fin de leur visite (contrairement au parcours en profondeur qui les place dans la liste au début de leur visite). Ce développement consiste à donner cet algorithme et à justifier sa correction. Sa terminaison et sa complexité sont élémentaires et seront données également.

On commence par donner l'algorithme principal, qui utilise la fonction auxiliaire **Visite** donnée immédiatement après. Pour alléger les notations, on supposera que **Visite** a accès aux variables définies dans le corps de **Tri_topo** (il suffit de définir **Visite** à l'intérieur de **Tri_topo**).

Algorithme 1 : Tri_topo

```
Tri_topo(G) :
    #G = (V, E)
    L ← [ ]
    Colorier tous les sommets en blanc
    #Par exemple avec un tableau de couleurs indexé par les sommets de G
    For u ∈ V do
        If (u est blanc) then
            Visite(u)
    return L
```

Algorithme 2 : Visite

```
Visite(u) :
    Colorier u en gris
    For v ∈Adj(u) do
        If (v est blanc) then
            Visite(v)
    Colorier u en noir
    L ← u :: L
```

Proposition : L'algorithme **Tri_topo** termine et a une complexité en $\Theta(|V| + |E|)$.

Démonstration : Le nombre de sommets blancs diminue strictement à chaque appel à **Visite**, donc l'algorithme termine. De plus, chaque sommet est visité exactement une fois, et chaque arête est traitée exactement une fois, d'où une complexité en $\Theta(|V| + |E|)$. □

Pour démontrer la correction, on donne d'abord quelques constatations :

Lemme :
- Pour tout $v \in V$, v est visité une unique fois, donc v est ajouté une unique fois à L.
- Au début et à la fin de chaque appel à **Visite**, L contient exactement l'ensemble des sommets noirs.
- Si au moment de l'appel à **Visite**(v), le sommet u est gris, alors il existe un chemin de u à v dans G.

Démonstration : Seul le troisième point n'est pas immédiat. Pour le démontrer proprement, faire une récurrence sur la profondeur de l'appel à **Visite**(v) dans l'exécution de **Visite**(u). □

Théorème : L'algorithme **Tri_topo** est correct.

Démonstration : Soit L la liste renvoyée par **Tri_topo**(G), et soit $(u, v) \in E$. Au moment de l'unique appel à **Visite**(u), il y a trois possibilités :
- Si v est noir, alors v est déjà dans L. Donc u est placé avant v dans L.
- Si v est gris, alors il existe un chemin de v à u dans G, et $(u, v) \in E$ ce qui contredit l'acyclicité de G. Absurde.
- Si v est blanc, alors v sera visité lors de la boucle sur les voisins de u. L'appel à **Visite**(v) sera donc terminé à la sortie de la boucle, donc v sera déjà dans L au moment où u y sera ajouté. Donc u est avant v dans L.

□

Remarques :
- On peut voir le tri topologique comme un "vrai" algorithme de tri : on trie un ensemble muni d'un ordre partiel.
- Le tri topologique est utilisé par le logiciel Make en compilation.
- On peut intégrer la vérification que le graphe donné en entrée est acyclique dans l'algorithme : il suffit de renvoyer une erreur "le graphe contient un cycle" lorsque l'on trouve un voisin gris lors d'un appel à visite (deuxième cas dans la dernière démonstration).