Université
Paris Cité

Tri topologique et composantes fortement connexes

CM nº4 — Algorithmique (AL5)

Matěj Stehlík

7/10/2022

L'homme d'affaires pressé

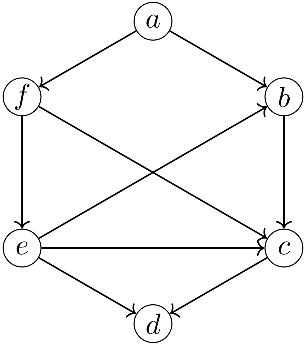
- Un homme d'affaire doit s'habiller très vite.
- Il ne doit bien sûr oublier aucun vêtement parmi les suivants :
  - caleçon
  - pantalon
  - ceinture
  - cravate
  - chemise
  - chaussettes
  - veste
  - montre
  - chaussures.
- Il ne peut bien sûr pas enfiler son pantalon ni mettre ses chaussures avant son caleçon.
- Les chaussettes doivent être mise avant les chaussures.
- La chemise doit être mise avant la cravate et la ceinture (qui ne peut être mise avant le pantalon).
- La veste ne peut être enfilée tant qu'il n'a pas sa cravate et sa ceinture.

Modélisation graphique du problème de l'homme d'affaires pressé

Tri topologique et les DAG (1/3)

Définition

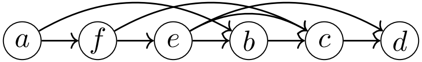
Un *tri topologique* d'un graphe orienté $G = (V, E)$ est un ordre total $\prec$ sur V tel que, pour tout arc $(u, v) \in E$, on a $u \prec v$.



Tri topologique et les DAG (1/3)

Définition

Un *tri topologique* d'un graphe orienté $G = (V, E)$ est un ordre total $\prec$ sur V tel que, pour tout arc $(u, v) \in E$, on a $u \prec v$.



Tri topologique et les DAG (2/3)

Théorème

Un graphe orienté G admet un tri topologique ssi G ne contient pas de circuits.

Démonstration (1/2)

- Si G contient un circuit, G n'admet évidemment pas de tri topologique.
- Inversement, supposons que G est un DAG.
- On définit l'ordre total $\prec$ sur les sommets de G comme suit :
- $u \prec v$ ssi $\text{post}(u) > \text{post}(v)$.

Tri topologique et les DAG (3/3)

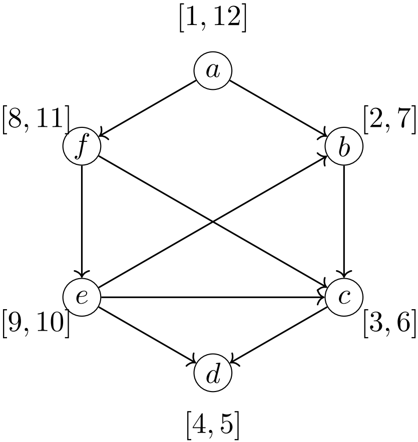
Démonstration (2/2)

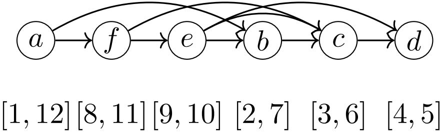
- Soit (u, v) un arc quelconque de G.
- Comme G est un DAG, v n'est pas un ancêtre de u.
- Donc, (u, v) n'est pas un arc retour.
- Par la remarque à la fin de la classification des arcs, $\text{post}(u) > \text{post}(v)$.
- Donc $u \prec v$.

Remarque

Pour trouver un tri topologique d'un DAG G, il suffit de faire un parcours en profondeur, et trier les sommets de G par ordre décroissant de $\text{post}(\cdot)$.

Exemple





Une conséquence du tri topologique

- Le premier sommet dans un tri topologique de G est une source (c'est-à-dire, aucun arc n'entre le sommet).
- De même, le dernier sommet est un puits (c'est-à-dire, aucun arc ne sorte du sommet).

Théorème

Tout graphe orienté acyclique contient au moins une source et au moins un puits.

Ce théorème est la base d'une autre approche au tri topologique :

- Trouver un sommet source de G et supprimer-le de G.
- Répéter jusqu'à ce que le graphe devienne vide.

Implémentation naïve de l'algorithme

Entrées : graphe orienté acyclique $G = (V, E)$

$L = \emptyset$

tant que $V \neq \emptyset$ **faire**

  trouver un sommet v t.q. $d^{-}(v) = 0$

  $G \leftarrow G - v$

  $L \leftarrow L + v$

return L

Quelques idées pour améliorer l'algorithme...

- Garder une file avec les sommets de degré entrant 0 pour ne pas avoir à rechercher ces sommets plusieurs fois.
- Le degré entrant d'un sommet ne change que lorsque l'un de ses voisins entrants (correspondant à des conditions préalables) ne soit supprimé.
- Garder une liste des degrés entrants des sommets pour ne pas avoir à modifier le graphe.

Algorithme de Kahn

Entrées : graphe orienté acyclique $G = (V, E)$

$L = \emptyset$

pour tous les $u \in V$ **faire**

  $d^-(u) \leftarrow$ degré entrant de u

créer file(Q)

pour tous les $u \in V$ **faire**

  si $d^-(u) = 0$ **alors**

    enfiler(Q, u)

tant que $Q \neq \emptyset$ **faire**

  $v \leftarrow$ défiler(Q)

  $L \leftarrow L + v$

  réduire le degré de tous les voisins sortants de v de 1

  pour tous les $u \in V$ **faire**

    si $d^-(u) = 0$ **alors**

      enfiler(Q, u)

return L

Complexité de l'algorithme de Kahn

- L'initialisation prend temps $O(n)$,
- La boucle **tant que** est parcourue $O(n)$ fois.
- Réduire le degré de tous les voisins sortants de v est de complexité $O(m)$.
- On obtient donc une complexité de $O(mn)$.

Complexité de l'algorithme de Kahn

- L'initialisation prend temps $O(n)$,
- La boucle **tant que** est parcourue $O(n)$ fois.
- Réduire le degré de tous les voisins sortants de v est de complexité $O(m)$.
- On obtient donc une complexité de $O(mn)$.
- Or, si on est attentif, on remarque que chaque arc n'est traité qu'une seule fois, donc finalement la boucle while prend temps $O(n+m)$.
- On conclut que l'algorithme de Kahn est de complexité $O(n+m)$, donc la même que si l'on utilise DFS.

Application du tri topologique aux tableurs

- Cellules dont les formules font référence à d'autres cellules ont des dépendances.
- On peut utiliser le tri topologique pour mettre à jour efficacement les cellules !

	A	B
1	1	
2	1	=A2/A1
3	=A1+A2	=A3/A2
4	=A2+A3	=A4/A3
5		=AVERAGE(B2:B4)

Application du tri topologique aux tableurs

- Cellules dont les formules font référence à d'autres cellules ont des dépendances.
- On peut utiliser le tri topologique pour mettre à jour efficacement les cellules !

	A	B
1	1	
2	1	=A2/A1
3	2	=A3/A2
4	=A2+A3	=A4/A3
5		=AVERAGE(B2:B4)

Application du tri topologique aux tableurs

- Cellules dont les formules font référence à d'autres cellules ont des dépendances.
- On peut utiliser le tri topologique pour mettre à jour efficacement les cellules !

	A	B
1	1	
2	1	1
3	2	`=A3/A2`
4	`=A2+A3`	`=A4/A3`
5		`=AVERAGE(B2:B4)`

Application du tri topologique aux tableurs

- Cellules dont les formules font référence à d'autres cellules ont des dépendances.
- On peut utiliser le tri topologique pour mettre à jour efficacement les cellules !

	A	B
1	1	
2	1	1
3	2	`=A3/A2`
4	3	`=A4/A3`
5		`=AVERAGE(B2:B4)`

Application du tri topologique aux tableurs

- Cellules dont les formules font référence à d'autres cellules ont des dépendances.
- On peut utiliser le tri topologique pour mettre à jour efficacement les cellules !

	A	B
1	1	
2	1	1
3	2	2
4	3	`=A4/A3`
5		`=AVERAGE(B2:B4)`

Application du tri topologique aux tableurs

- Cellules dont les formules font référence à d'autres cellules ont des dépendances.
- On peut utiliser le tri topologique pour mettre à jour efficacement les cellules !

	A	B
1	1	
2	1	1
3	2	2
4	3	1.5
5		`=AVERAGE(B2:B4)`

Application du tri topologique aux tableurs

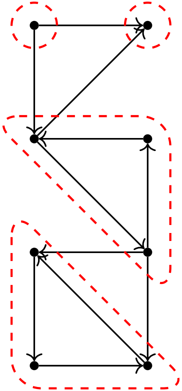
- Cellules dont les formules font référence à d'autres cellules ont des dépendances.
- On peut utiliser le tri topologique pour mettre à jour efficacement les cellules !

	A	B
1	1	
2	1	1
3	2	2
4	3	1.5
5		1.5

Connexité dans les graphes orientés

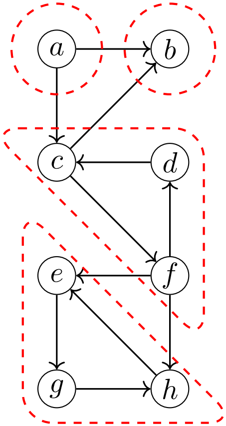
- Nous avons déjà vu la définition de graphes connexes et des composantes connexes.
- Intuitivement, un graphe est connexe s'il ne peut pas être "séparé" sans casser des arêtes.
- Pour les graphes orientés, la notion de connexité est un peu plus compliquée.
- Soit $\sim$ la relation suivante : $u \sim v$ ssi il existe un chemin de u à v et aussi un chemin de v à u.

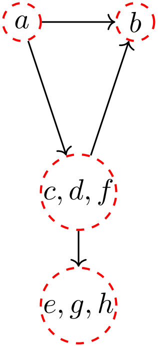
Composantes fortement connexes



- Il est facile de vérifier que $\sim$ est une relation d'équivalence :
  - $\sim$ est symétrique
  - $\sim$ est refléxive
  - $\sim$ est transitive.
- Les classes d'équivalence forment une partition de $V(G)$.
- On les appelle les *composantes fortement connexes*.

Contracter les composantes fortement connexes





Le graphe contracté est un DAG

Théorème

Soit G un graphe orienté quelconque, et soit G' le graphe obtenu en contractant chaque composante fortement connexe à un seul sommet. Alors, G' est un graph orienté acyclique (un DAG).

- Soit $C = (G_1, G_2, \ldots, G_k)$ un circuit dans G'.
- Alors, tous les sommets de G dans la composante fortement connexe G_i sont atteignable depuis tous les sommets de G_j, pour tous $i, j \in \{1, \ldots, k\}$.
- Donc, $V(G_1) \cup V(G_2) \cup \cdots \cup V(G_k)$ appartiennent à une seule composante connexe.
- Donc $k = 1$.

Composantes fortement connexes et parcours en profondeur

Propriété 1

Si la procédure `explorer` est lancée à partir du sommet u, elle se terminera précisément lorsque tous les sommets atteignables depuis u auront été visités.

- Donc, si u est un sommet dans une composante fortement connexe G_i qui est un puits dans le graphe contracté G' (tous les arcs incidents à G_i pointent vers G_i), alors $\texttt{explore}(u)$ va parcourir précisément les sommets de G_i.

Composantes fortement connexes et parcours en profondeur

Propriété 2

Soient G_i et G_j des composantes fortement connexes de G. S'il existe un arc d'un sommet de G_i à un sommet de G_j, alors

$$\max\{\textbf{post}(v) : v \in G_i\} \geq \max\{\textbf{post}(v) : v \in G_j\}.$$

- Il y a deux cas à considérer.
- Si le DFS visite la composante G_i avant la composante G_j, alors tous les sommets de G_i et G_j seront visités avant de coincer.
- Par conséquent, le nombre post du premier sommet visité dans G_i sera supérieur à celui de n'importe quel sommet de G_j.
- Si G_j est visité en premier, le DFS va coincer après avoir visité l'ensemble de G_j mais avant d'avoir visité l'ensemble de G_i.

Conséquence de la Propriété 2

Propriété 3
Le sommet avec la valeur maximum de post$(\cdot)$ dans une recherche en profondeur appartient à une composante fortement connexe de type source.

- On peut trier les composantes fortement connexes par ordre décroissant de leurs nombres post maximaux.

. . . et si on veut trouver un sommets dans un puit ?

- La Propriété 2 nous aide à trouver un sommet dans une composante fortement connexe de G de type « source ».
- Or, nous avons besoin d'un sommet dans une composante fortement connexe de G de type « puits ».
- Soit G^R le graphe orienté *inverse*, définit comme suit : $G^R = (V, E^R)$, où $(u, v) \in E^R$ ssi $(v, u) \in E$. C'est-à-dire, on reverse la direction des arcs.
- G^R a les même composantes fortement connexes que G.
- En effectuant un parcours en profondeur sur G^R, le sommet avec la valeur maximale de $\text{post}(\cdot)$ appartient à une composante fortement connexe de G^R de type « source », c'est-à-dire, à une composante fortement connexe de G de type « puits ».

Vers un algorithme de composantes fortement connexes

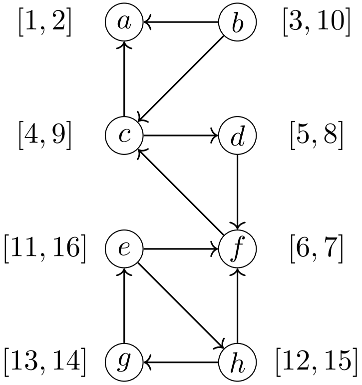
- Comment continuer après l'identification de la première composante fortement connexe de type puits ?
- Il suffit d'utiliser la Propriété 2.
- Une fois que nous avons trouvé la première composante fortement connexe G_1 et que nous l'avons supprimée du graphe, le sommet avec le nombre post maximum dans $G - V(G_1)$ appartiendra à une composante fortement connexe de $G - V(G_1)$.
- Par conséquent, nous pouvons continuer à utiliser les nombres $\mathrm{post}(\cdot)$ du parcours en profondeur initial sur G^R pour produire successivement la deuxième composante fortement connexe, la troisième composante fortement connexe, etc.

Un algorithme de composantes fortement connexes

Voici donc un algorithme pour déterminer les composantes fortement connexes de G :

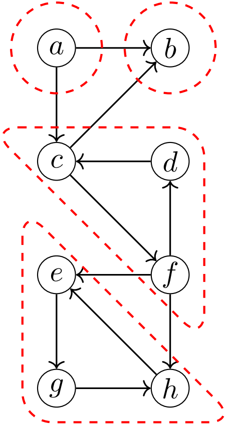
1. Exécuter un parcours en profondeur sur G^R et garder pour chaque sommet son valeur post.
2. Trier les sommets selon leurs valeurs post.
3. Exécuter un parcours en profondeur sur sur G selon l'ordre inverse. (En particulier, chaque fois que la procédure DFS appelle la procédure explore, commencer par la premier sommet non visité dans l'ordre inverse.)

Exemple



G^R

g, e, h, b, c, d, f, a



G

$\{g, e, h\}, \{b\}, \{c, d, f\}, \{a\}$

Applications du parcours en profondeur : résumé

Nous avons vu 3 applications du parcours en profondeur :

1. Détection de circuits dans les graphes orientés :
   - Un DFS révèle un arc retour ssi le graphe contient un circuit.
2. Tri topologique des graphes acycliques orientés :
   - Il suffit de faire un DFS qui garde trace des temps de visite, et ensuite trier les sommets par nombre post décroissant.
3. Calcul des composantes connexes d'un graphe orienté :
   - On fait DFS sur le graphe inverse G^R et on trie les sommets selon l'ordre décroissant de nombre post
   - Ensuite on fait DFS sur G en utilisant l'ordre trouvé dans la première étape.

Exercice

Deux ivrognes ont à se partager équitablement (c'est-à-dire par moitié) 8 litres de vin. Ils disposent de trois cruches : celle qui contient les 8 litres, et deux autres, de contenances respectives 5 litres et 3 litres. Aucune d'entre elles n'étant graduée, ils ont recours à une seule opération : verser le contenu d'une cruche dans une autre, en ne s'arrêtant que lorsque la crouche source soit vide ou que la cruche de destination soit pleine.

1. Modéliser ce problème comme un problème de graphe : donner une définition précise du graphe concerné et énoncer la question spécifique à ce graphe à laquelle il faut répondre.
2. Quel algorithme peut être appliqué pour résoudre le problème ?
3. Trouver la réponse en appliquant l'algorithme.