Université catholique de Louvain
Ecole polytechnique
Année académique 2013-2014

Détection de cycle dans un graphe dirigé

Conventions de représentations et théorie du problème

Groupe 9
LINGI1122 - Méthodes de conception de programmes
Titulaire : José Vander Meulen

Table des matières

Contexte

Problème: étant donné un graphe dirigé, déterminer si celui-ci contient ou non un cycle.

Algorithme: supprimer tous les nœuds qui n'ont pas d'arête entrante, ainsi que les arêtes dont ces nœuds sont l'origine. En répétant cette opération, deux cas peuvent survenir:

1. plus de nœud disponible $\rightarrow$ pas de cycle;
2. il ne reste que des nœuds avec au moins une arête entrante $\rightarrow$ au moins un cycle dans le graphe.

Théorie du problème I

Un **graphe dirigé**, ou orienté, est un triplet (V, E, ψ) où:

- V est un ensemble dont les éléments sont appelés sommets ou nœuds;
- E est un ensemble dont les éléments sont appelés arêtes;
- ψ est une fonction, dite fonction d'incidence, qui associe à chaque arête un couple de sommets. Ici, l'ordre au sein du couple de sommets a de l'importance, il signifie qu'un sommet est le nœud de départ de l'arête, l'autre étant le nœud d'arrivée.

Un **cycle** est un parcours fermé dont les sommets d'origine et intérieurs sont tous distincts. Un graphe qui ne contient pas de cycle est dit acyclique.

L'algorithme est correct I

Soit G un graphe possédant n nœuds v et m arêtes e. Procédons par l'absurde, et supposons que tous les nœuds de G possèdent une arête entrante, et que G est acyclique.

- On choisit arbitrairement le nœud v_i, par hypothèse le nœud v_i possède au moins une arête entrante.
- On choisit arbitrairement l'une des ces arêtes entrantes et on supprime les éventuelles autres arêtes entrantes de v_i, on arrive alors au nœud v_j, qui possède également au moins une arête entrante(par hypothèse).
  - Soit on arrive à un nœud déjà visité et la démonstration est finie.
  - Soit on arrive à un nœud qu'on n'avait pas encore visité et on réitère l'algorithme.
- Comme le nombre de nœuds de G est fini, on arrive au dernier nœud (puisqu'on n'a pas de cycle jusqu'à présent). Hors, par hypothèse ce dernier nœud possède également une arête entrante qui ne peut pointer vers un autre nœud qu'un de ceux visité $\Rightarrow$ il y a un cycle et contradiction.

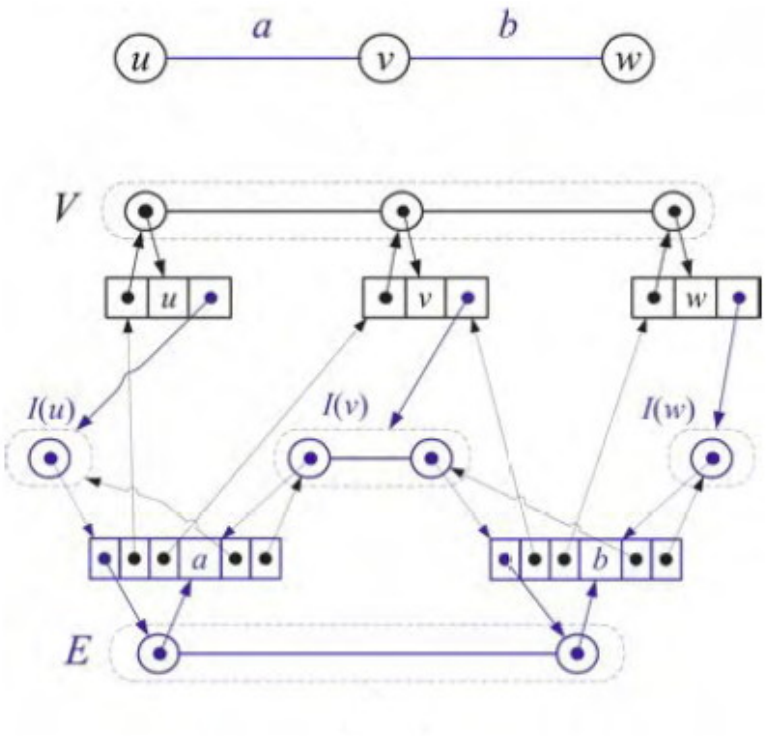
L'algorithme se termine I

Le graphe G possède un nombre fini de nœuds et d'arêtes.
A chaque itération

- Il ne reste aucun nœud $\Rightarrow$ Le programme s'arrête.
- Tous les nœuds ont au moins une arête entrante $\Rightarrow$ Le programme s'arrête.
- Il existe un nœud ne possédant pas d'arête entrante. Ce nœud (et ses arêtes) est retiré par l'algorithme. Comme le graphe G possède, par hypothèse, un nombre fini de nœuds, il finit par se retrouver dans l'une des situations précédentes $\Rightarrow$ Le programme s'arrête.

Conventions de représentation I

Adjacent List Stucture



Objet noeud :

- Référence vers une liste chainée contenant des références vers les arêtes sortantes du noeud
- Entier *incounter* contenant le nombre d'arêtes entrantes au noeud

Objet arête :

- Référence vers le noeud dont l'arête est la destination

Conventions de représentation II

Avantages :

- **Facilité d'implémentation**
- **Complexité** de l'algorithme en $\mathcal{O}(n+m)$ avec n le nombre de noeuds et m le nombre d'arêtes. C'est le meilleur choix possible car toutes les arêtes et tous les noeuds sont parcourus dans le pire des cas. De plus, la complexité est meilleure par rapport à
  - l'*edgelist structure* : *IncidentEdges* en $\mathcal{O}(m)$
  - l'*adjacency matrix structure* : *IncidentEdges* en $\mathcal{O}(n)$

  En effet, la complexité de *IncidentEdges* est dans notre cas de $\mathcal{O}(1)$