Détection de cycle dans un graphe dirigé :
conventions de représentations et théorie du problème

LINGI1122 - Méthodes de conception de programmes

SEDDA	Mélanie	2246-11-00
SLUYSMANS	Benoît	6957-11-00
VAN DEN EECKHAUT	Kim	7561-11-00
VICO	Nicolas	3271-09-00
VOLON	Julien	???

1 Introduction

Dans le cadre du cours de méthode de conception de programmes, il nous a été demandé de résoudre le problème suivant : étant donné un graphe dirigé, déterminer si celui-ci contient ou non un cycle. L'algorithme utilisé doit donc ressortir une réponse booléenne.

L'algorithme que nous devons utiliser est le suivant : supprimer du graphe 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 : soit il n'y a plus de nœud disponible, et nous pouvons conclure qu'il n'existe pas de cycle dans le graphe de base, soit il ne reste que des nœuds avec au moins une arête entrante, auquel cas il existe au moins un cycle dans le graphe.

2 Conventions de représentations

2.1 Présentation de la convention retenue

L'algorithme prend un graphe dirigé en entrée et retourne un booléen, qui indique si le graphe possède un cycle ou non. Il nous faut donc définir une convention de représentation d'un graphe dirigé sur lequel notre algorithme va s'appliquer.

Le graphe sera représenté en utilisant des listes d'adjacence (adjacency list structure). Dans cette structure, chaque objet noeud contient une référence vers la liste des arêtes qui sortent de celui-ci (mais pas celles entrantes). De même, chaque objet arête, contient une référence au noeud vers lequel elle se dirige. La Figure 1 illustre cette structure dans le cas d'un graphe non dirigé où V est l'ensemble des sommets $I(v)$ représente la liste des arêtes adjacentes au noeud v et E est l'ensemble des arêtes. Nous adaptons un peu ce schéma pour le cas d'un graphe dirigé sur lequel on appliquera notre algorithme en disant qu'un noeud ne doit connaître que la liste des arêtes sortantes et qu'une arête ne doit connaitre que sa destination. Nous ajoutons également à la classe noeud une variable d'instance incounter comptant le nombre d'arêtes entrantes au noeud.

2.2 Motivations

Le fait de représenter un graphe avec la structure décrite précédemment possède divers avantages. Ceux-ci sont les suivants :

- **Facilité d'implémentation** : dans l'algorithme à implémenter, les actions principales à effectuer sur le graphe sont de trouver les arêtes sortantes d'un noeud, les noeuds vers lesquels elles se

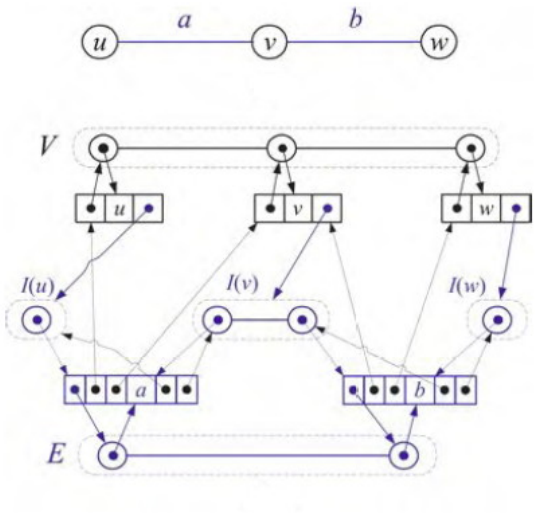

FIGURE 1 – Liste d'adjacence

dirigent et de diminuer leur incounter. L'implémentation de ces deux fonctionnalités est très facile car chaque objet contient les références nécessaires.

- **Complexité satisfaisante** : nous prouverons plus tard que la complexité temporelle de l'algorithme avec notre structure de données est en $\mathcal{O}(n+m)$ où n et m sont respectivement le nombre de noeud et d'arêtes du graphe. Nous ne pourrions pas faire mieux étant donné qu'il va au pire effectivement falloir passer par tous les noeuds et les arêtes. Deux autres représentations de graphe existent : la edge list structure et la adjacency matrix structure. La première a une complexité en $\mathcal{O}(m)$ pour trouver les arêtes adjacentes à un sommet alors qu'avec notre structure la complexité est en $\mathcal{O}(1)$. La matrice d'adjacence présente aussi des désavantages à ce niveau là car passer par toutes les arêtes est en $\mathcal{O}(n^2)$ au lieu de $\mathcal{O}(m)$.

3 Théorie du problème

3.1 Définitions

Un **graphe** 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 sommet ou une paire de sommets.

On peut représenter un graphe par un diagramme. Plusieurs diagrammes peuvent représenter le même graphe.

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 **parcours** est une suite $v_0e_1v_1e_2...e_nv_n$, où $v_1; v_2; ...$ sont des sommets, et $e_1; e_2; ...$ sont des arêtes. La longueur du parcours est son nombre d'arêtes n. Le sommet d'origine est v_0, le sommet de destination v_n. Les autres sommets sont dits intérieurs. Un parcours est fermé si $v_0 = v_n$.

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.

3.2 L'algorithme utilisé est correct

Démonstration : L'algorithme est correct.

Hypothèses : Soit G un graphe possédant n nœuds v_i avec $i = 1, ..., n$ et m arêtes e_j avec $j = 1, ..., m$. Procédons par l'absurde, et supposons que G possède un sous-graphe H dont tous les noeuds possèdent une arête entrante, et que G est acyclique.

- On choisit arbitrairement le nœud v_i appartenant à H. Par hypothèse le nœud v_i possède au moins une arête entrante.
- On choisit arbitrairement l'une des ces arêtes entrantes et on prend le noeud v_j qui est l'origine de cette arête et 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 le point précédent.
- Si tous les noeuds de H ont été parcourus, comme le nombre de nœuds de H est fini, on arrive au dernier nœud. 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 donc il y a contradiction avec l'hypothèse de départ.

Démonstration : L'algorithme se finit toujours.

Hypothèse : Le graphe G possède un nombre fini de nœuds et d'arêtes.
A chaque itération, on observe plusieurs cas possibles :
- 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.