Détermination de cycles

Cycles. On considère un graphe orienté $G = (S, A)$, où $A \subset S \times S$.

Pour x et y sommets, on note $x \to y$ pour signifier que $(x, y) \in A$.

Un cycle est une famille $(x_0, ..., x_{p-1})$ de p sommets, avec $p \geq 2$ telle que

$$x_0 = x_{p-1} \quad \text{et} \quad x_0 \to x_1 \to ... \to x_{p-1}$$

Un graphe est dit acyclique ssi il ne contient aucun cycle.

En PYTHON, le graphe est codé par un dictionnaire G de sorte que les couples (*clé,valeur*) sont les $(s, G[s])$, où s est un sommet de G et $G[s]$ est la liste des successeurs de s.

1) Détermination d'un cycle dans un graphe orienté via un parcours en profondeur

Lors du parcours en profondeur, on utilise un marquage qui prend trois valeurs possibles :

- `M[s] = 0` signifie : s n'a pas encore été visité

- `M[s] = 1` signifie : s est visité mais l'arbre dont il est racine est en cours de visite

- `M[s] = 2` signifie : s est traité (c'est-à-dire tous ses descendants sont visités).

Il existe un cycle ssi on rencontre au cours du processus un sommet déjà visité mais non encore traité : en effet, entre le moment où un sommet s est visité et celui où il est traité (c'est-à-dire le sous-arbre issu de s est visité), les seuls sommets parcourus sont les successeurs de s, et lorsqu'on est amené à visiter à nouveau le sommet s dans ce laps de temps, on est sûr qu'il existe un cycle passant par x. Ainsi, pour déterminer s'il existe un cycle.

Remarque : En effet, le sens direct est immédiat. Réciproquement, supposons G cyclique. Il existe donc un cycle. Comme s n'a pas de prédécesseur, le cycle ne passe pas par s, donc est encore un cycle de G'.

Écrire une fonction `cycle(G)` de complexité linéaire qui étant donné un graphe G représenté par la liste des listes d'adjacence, renvoie `True` ssi G contient un cycle.

On utilise :

- un dictionnaire de marquage M dont les clés sont les sommets et les valeurs appartiennent à $\{0, 1, 2\}$

- une procédure récursive auxilaire `traite(s)` ; en particulier, on ne fait rien si $M[s] = 2$.

2) Détermination qu'un graphe orienté est acyclique par suppressions de sommets

Principe (admis ici) :

- Si un graphe G est acyclique, il existe un sommet x sans prédécesseur

- Si x est un sommet sans prédécesseur, alors G est acyclique ssi le graphe G' obtenu en supprimant dans G le sommet x et les arêtes associées est acyclique.

On appelle degré entrant d'un sommet s le nombre de prédécesseurs de s.

On va utiliser un dictionnaire d dont les couples (*clé,valeur*) sont les $(s, d[s])$ où s est un sommet et d le degré entrant de s (dans le graphe considéré, qui est modifié au cours de l'algorithme).

Pour obtenir une complexité linéaire en $O(n+m)$, on utilise le dictionnaire des degrés entrants, qu'on met à jour au fur et à mesure de la suppression de sommets sans prédécesseur :

Etape 1 :

- on initialise le tableau des degrés entrants
- on initialise la pile des sommets dont le degré entrant vaut 0
- on initialise à 0 un compteur qui dénombre le nombre de sommets supprimés

Etape 2 : Tant que la pile n'est pas vide :

- on supprime un élément s de la pile et on met à jour le compteur
- on met à jour le tableau des degrés entrants correspondant au graphe obtenu en supprimant x
- on ajoute à la pile les sommets dont le nouveau degré entrant vaut 0

Etape 3 : Le graphe initial est acyclique ssi *tous* les sommets ont été supprimés.

a) Écrire une fonction `acyclique(G)` qui étant donné un graphe représenté par la liste des listes d'adjacence, renvoie `True` ssi G est acyclique (c'est-à-dire sans cycle).

b) Justifier la correction de l'algorithme.