Ordre topologique dans un graphe orienté sans cycle

Soit $G = (S, A)$ un graphe orienté sans cycle. On note S l'ensemble des sommets.

On supposera les sommets de G numérotés de 0 à $(n-1)$, et G donné par la liste des listes des fils.

Ainsi, $G[x]$ est la liste des fils de i, c'est-à-dire la liste des j tels que (i, j) est une arête de G.

Un chemin dans G est une suite $(x_0, x_1, x_2, ..., x_m) \in S$ telle que $\forall k$, $(x_k, x_{k+1}) \in A$.

La longueur d'un chemin est le nombre de ses arêtes (ici m).

1) Caractérisation des graphes orientés acycliques

a) *Prop* : Les deux assertions suivantes sont équivalentes :

(i) Le graphe G est acyclique (c'est-à-dire sans cycle)

(ii) Tous les chemins sur G sont de longueur $\leq n-1$, où n est le nombre de sommets de G.

Preuve : Supposons (i). Considérons un chemin $x_0 \to x_1 \to ... \to x_p$. Les sommets de ce chemin sont nécessairement deux à deux distincts, car sinon, le graphe admettrait un cycle. Donc $p \leq n-1$.

Réciproquement, supposons qu'il existe un cycle $x_0 \to x_1 \to ... \to x_p = x_0$, où $p \geq 1$.
Alors il existe des chemins de longueur kp arbitrairement grand (en répétant k fois le cycle).

b) *Corollaire* : Dans tout graphe acyclique, il existe un sommet sans prédécesseur et il existe un sommet sans successeur.

Preuve : On déduit de a) qu'il existe un chemin $x_0 \to x_1 \to ... \to x_p$ de longueur p maximale.
Nécessairement, le sommet x_0 n'a pas de prédécesseur et le sommet x_p n'a pas de successeur.

2) Ordre topologique dans un graphe orienté acyclique

a) Soit G un graphe orienté. On note S l'ensemble des sommets.

Un ordre topologique sur G est un ordre *total* $\prec$ sur S compatible avec la relation de descendance définie sur G :
$\forall (x, y) \in S^2$, (y est un descendant de x) $\Rightarrow (x \prec y)$.
Remarque : Par transitivité, il faut et il suffit d'avoir : $\forall (x, y) \in S^2$, $(x \to y) \Rightarrow (x \prec y)$.

En pratique, un ordre topologique consiste à trouver une fonction $\sigma : S \to \mathbb{N}$ telle que

$$\forall (x, y) \in S^2, \quad (x \to y) \Rightarrow \sigma(x) < \sigma(y)$$

Remarque : En classant les sommets par valeur croissante de σ, on obtient un ordre topologique sur G.

On obtient alors $S = \{x_0, ..., x_{n-1}\}$ vérifiant : $(x_i \to x_j) \Rightarrow (i < j)$.

b) **Prop : Les deux assertions suivantes sont équivalentes :**

(i) Le graphe G est acyclique (c'est-à-dire sans cycle)

(ii) Il existe un ordre topologique sur G.

Preuve : Supposons (ii). Supposons par l'absurde qu'il exisun cycle $x_0 \to x_1 \to ... \to x_p = x_0$.

Alors $\sigma(x_0) < \sigma(x_1) < ... < \sigma(x_p) = \sigma(x_0)$, ce qui est absurde. Donc (i).

Supposons (i), c'est-à-dire G acyclique.

Première preuve (utilisé ici conduit à l'algorithme de Kakn, cf paragraphe 4) :

On raisonne par récurrence sur le nombre de sommets. Il existe un sommet x sans successeur. Considérons le sous-graphe G' obtenu en supprimant dans G le sommet x et les arêtes associées. Comme G est acylique, il en est a fortiori de même de G'. Par hypothèse de récurrence, il existe un ordre topologique sur G'. On prolonge σ sur G en associant à x de sorte qu'il aît le dernier numéro.

Seconde preuve : On appelle hauteur $h(x)$ d'un sommet la longueur maximale d'un chemin se terminant par x. Comme les chemins sont tous de longueur $< n$, la hauteur $h(x)$ est bien défini.

Par exemple, les sommets sans prédécesseur sont de hauteur $h(x) = 0$.

Pour obtenir un ordre topologique, il suffit de classer les sommets par **hauteur croissante**.

Remarque : Une variante consiste à classer les sommets par **profondeur décroissante**, où la profondeur $p(x)$ d'un sommet est la longueur maximale d'un chemin commençant par x.

c) *Remarques* :

- Les graphes filiforme (c'est-à-dire en chaîne) sont les seuls possédant un unique ordre topologique.
- Les graphes sans arête sont les seuls où tout ordre est un ordre topologique.

3) Obtention d'un ordre topologique par l'algorithme de Kahn

a) Principe de l'algorithme

On considère un graphe acyclique $G = (S, A)$.

Il existe nécessairement un sommet de degré entrant nul. Sinon, en partant d'un sommet arbitraire x_0, on pourrait trouver une suite infinie $(x_k)_{k\in\mathbb{N}}$ tel que $(x_{k+1}, x_k) \in A$ pour tout $k \in \mathbb{N}$. On obtient donc nécessairement un cyle (par le principe des tiroirs).

L'algorithme de Kahn est basé sur le principe suivant : si $(x_1, ..., x_{n-1})$ est un ordre topologique sur le sous-graphe $G' = (S', A')$, où $S' = S \setminus \{x_0\}$ et $A' = A \cap (S')^2$, alors $(x_0, x_1, ..., x_{n-1})$ est un ordre topologique sur le graphe G. On peut applique le principe à G', et itérer jusqu'à se ramener à un graphe vide.

Informatiquement, on va gérer une liste contenant tous les sommets entrants de degré entrant nul (pour les sous-graphes obtenus en éliminant peu à peu les sommets).

On va utiliser un parcours du style parcours en largeur en utilisant une file et en commençant par les éléments de degré entrant nul (il en existe nécessairement au moins un car G est acyclique)

Ensuite, on extrait un élément de la file, et **on met à jour le tableau des degrés entrants** de sorte qu'il corresponde au nouveau sous-graphe.

b) Code de l'algorithme de Kahn

```
def ordre_topologique(G) :
    ### Calcul des degrés entrants
    n = len(G) ; degre = [0]*n
    for x in range(n) :
        for y in G[x] :  degre[y] += 1
    ### Calcul des sommets de degré entrant 0
    pile = []    # pile des éléments à traiter
    for x in range(n) :
         if degre[x]==0 :  pile.append(x)
    ### Tri topologique
    ordre = []
    while pile :
         x = pile.pop() ; ordre.append(x)
         for y in G[x] :
             degre[y] = degre[y] - 1
             if d[y] == 0 :  pile.append(y)
    return ordre
```

Remarque : Le logique serait plutôt d'utiliser une file au lieu d'une pile, mais en PYTHON, il est plus pratique d'utiliser une pile.

Remarque : Un élément se trouve dans la pile ssi il n'admet aucun parent non traité.

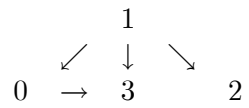
Remarque : S'il existe un ou plusieurs sommets dont le degré entrant ne décroît pas à 0, alors cela signifie qu'il apparaît sur un cycle. En fait, s'il existe un cycle, on obtiendra à un moment donné une pile vide alors qu'il reste des sommets à traiter.
On peut donc ajouter avant le return le test suivant : `assert len(ordre)<n , "existence d'un cycle"`

Complexité : Le calcul des degrés entrants s'effectue en temps $O(n+m)$, où $n = |S|$ et $m = |A|$.
L'identification des sommets de degré entrant 0 s'effectue en temps $O(n)$.
L'exploration d'un sommet x requiert un temps en $O(1 + d_x)$, où d_x est le degré entrant de x.
Puisque tous les sommets sont explorés, le temps total est donc en $O(n+m)$.

4) Obtention d'un ordre topologique par un parcours en profondeur

a) Considérons donc un graphe acyclique. Une méthode efficace pour obtenir σ est d'utiliser un parcours en profondeur **avec une numérotation postfixe** : tout sommet est numéroté après ses fils.

Le principe consiste à parcourir successivement, pour chaque sommet, l'arbre issu de ce sommet. Les sommets visités sont marqués de sorte à ne visiter qu'une fois chaque sommet.

Considérons par exemple le graphe suivant : 

On visite successivement les arbres issus des sommets 0, 1, 2 et 3.

L'arbre issu de 0 permet de visiter 0 et 3. On les classe dans l'ordre postfixe, c'est-à-dire $3, 0$.

Puis on visite les sommets non marqués de l'arbre issu de 1. On les classe dans l'ordre postfixe.

On obtient donc $[3, 0, 2, 1]$.

En inversant cette liste, on obtient un ordre topologique : $[1, 2, 0, 3]$.

Montrons que d'une façon générale, le parcours en profondeur postfixe donne bien un ordre topologique (en inversant la liste obtenue). Notons $f(x)$ le numéro attribué au sommet x.

Il s'agit de prouver que si y est un descendant de x, on a $f(y) < f(x)$: Lorsqu'on visite x, on numérote les descendants de x non encore numérotés avant de numéroter x. **Que y soit ou non déjà numéroté lors de la visite de x, le sommet y est donc numéroté avant x.**

Pour obtenir un ordre topologique, il suffit donc d'inverser l'ordre de la numérotation associée à f.

En pratique, on utilise un tableau auxiliaire de marquage pour indiquer les sommets déjà visités, d'où l'algorithme :

- On initialise le tableau de marquage sachant qu'aucun sommet n'est encore visité ;

- Pour tout sommet x, on effectue le traitement $T(x)$ séfini comme suit :

(1) Si x a déjà été visité, on ne fait rien ; sinon, on effectue les opérations suivantes

(2) On marque x comme visité dans le tableau de marquage

(3) On traite les fils de x, c'est-à-dire que pour tout fils y de x, on effectue $T(y)$

(4) On numérote x (en lui attribuant le numéro qui suit le dernier numéro attribué).

```
01.     def traite(x,G,M,L) :
02.         if M[x] == 0 :
03.             M[x] = 1 :   # le marquage se fait avant la boucle for
04.             for y in G[x] :  traite(y,G,M,L)
05.             L.append(x) ; M[x] = 2
06.     def ordre(G) :
07.         n = len(G) ; M = [0]*n ; L = []
08.         for x in range(n) :  traite(x,G,M,L)
09.         return reversed(L)
```

Remarque : `M[x]` vaut `0` lorsque x n'a pas encore été visité, `M[x]` vaut `1` lorsque x a été visité mais pas encore ajouté à la liste L, et `M[x]` vaut `2` si x est traité (c'est-à-dire ajouté à la liste L).

La complexité est linéaire en $O(n + m)$, où n est le nombre de sommets et m le nombre d'arêtes.

En effet, la fonction `traite(x,G,M,L)` est appelée une unique fois (puisqu'elle n'est appelée que si `M[x]` vaut `0`, et que `M[x]` est mis à `1` tout de suite après l'appel à `traite(x,G,M,L)`.

On en déduit aussi que le test de la ligne 02. est effectué une seule fois pour chaque arête (x, y).

Donc le test de la ligne 02 est effectué m fois. D'où une complexité en $O(n + m)$.

b) Variante pour décider (en temps linéaire) **de l'existence d'un cycle**

On considère ici un graphe orienté arbitraire. On lui applique l'algorithme précédent.

Il existe un cycle ssi on rencontre au cours du processus un sommet déjà visité mais non encore numéroté, c'est-à-dire un sommet x tel que $M[x] = 1$. En effet, entre le moment où un sommet x est visité et celui où il est numéroté, les seuls sommets parcourus sont les descendants de x, et lorsqu'on est amené à visiter à nouveau le sommet x dans ce laps de temps, on est sûr qu'il existe un cycle passant par x. Réciproquement, un cycle ne peut être parcouru sans qu'un de ses sommets ne soit visité deux fois avant d'être ajouté à la liste. Ainsi, pour déterminer s'il existe un cycle :

```
01.     def Testcycle(G) :
02.         def traite(x,G,M,L) :
03.             if M[x] == 1 :  cycle = True  # il existe un cycle
04.              elif M[x] == 0 :
05.                  M[x] = 1
06.                  for y in G[x] :  traite(y)
07.                  M[x] = 2
08.          n = len(G) ; M = [0]*n ; cycle = False
09.          for x in range(n) :  traite(x,G,M,L)
10.          return cycle
```