DS d'informatique n°5 : Tri topologique de graphes

Durée : **2h**. Calculatrices non autorisées.
Le soin et la clarté de la rédaction pourront faire varier la note de ±1 point. Il n'est pas attendu que vous arrivez au bout du sujet (le barème dépassera 100 points).
Les fonctions devront être commentées lorsque cela est utile, et l'indentation devra être respectée.

Monsieur Cosinus aide régulièrement monsieur Sinus à apprendre les gestes simples de la vie quotidienne. Aujourd'hui, la leçon porte sur l'ordre dans lequel il faut mettre ses vêtements afin de s'habiller. Voici deux exemples de règles :

- Il n'est pas possible de mettre sa cravate avant sa chemise.
- Les chaussures doivent être mises après le caleçon, le pantalon et les chaussettes.

Pour aider son ami à comprendre les différentes règles, M. Cosinus décide d'utiliser une représentation sous forme de graphe orienté (voir la figure 1). Dans ce graphe, chaque arc représente une contrainte : s'il existe un arc entre le sommet A et le sommet B, cela signifie que le vêtement A doit être mis avant le vêtement B. En particulier, les deux règles énoncées ci-dessus se déduisent bien du graphe de la figure 1.

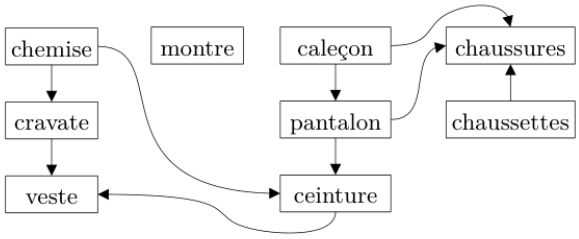

Figure 1

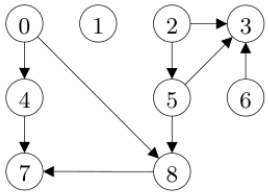

Figure 2

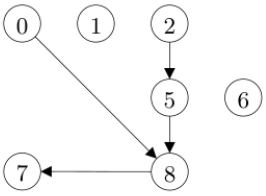

Figure 3

L'objectif de M. Sinus est, à partir de la figure 1, de déterminer l'ordre dans lequel mettre ses vêtements. Voici une possibilité :

(T_1) : chaussettes, caleçon, pantalon, chaussures, montre, chemise, ceinture, cravate, veste.

Afin de modéliser le problème, on considère un graphe orienté $G = (S, A)$ à n sommets numérotés de 0 à $n-1$ (la figure 2 montre une numérotation possible des sommets de la figure 1). Notre but est de trouver un *tri topologique* du graphe, c'est-à-dire déterminer une liste T qui vérifie les trois conditions suivantes :

(C1) T est de taille n.

(C2) T contient une et une seule fois chaque entier de $[\![0, n-1]\!]$.

(C3) Si G contient l'arc (u, v) alors le sommet u apparaît strictement avant v dans T.

Par exemple, le tri topologique T_1 correspond à la liste :

```
T1 = [6, 2, 5, 3, 1, 0, 8, 4, 7]
```

Partie I : vérifier si une liste est un tri topologique

Dans ce devoir, on représentera un graphe orienté $G = (S, A)$ par une liste d'adjacence `G` : pour tout sommet $i \in [\![0, n-1]\!]$, `G[i]` contient la liste des successeurs de i dans le graphe G.

1) Donner la liste d'adjacence du graphe de la Figure 2, qui sera noté `Gfig2` dans la suite du devoir.

2) **a)** Écrire une fonction `test1` qui prend en argument un entier n et une liste T et qui renvoie `True` si la condition **(C1)** est vérifiée, `False` sinon.

b) Quelle est la complexité de votre fonction `test1` ?

3) Écrire une fonction `test2` qui prend en argument un entier n et une liste T et qui renvoie `True` si la condition **(C2)** est vérifiée, `False` sinon.
Bonus : écrire cette fonction avec une complexité en $O(n)$. On ne demande pas une preuve de cette complexité. À savoir : les fonctions "toutes faites" qui nécessitent un parcours de la liste dans le pire cas (telles que `L.remove(...)`) ont en général une complexité au moins en $O(n)$...

4) Dans cette question, on considère une liste `T1` et on suppose que les instructions `test1(n,T1)` et `test2(n,T1)` renvoient toutes deux `True`.

a) Écrire une fonction `inverse` qui prend en argument une liste `T1` qui renvoie une liste `T2` de taille n telle que pour tout $u \in [\![0, n-1]\!]$, `T2[u]` soit égal à l'indice de `u` dans `T1`. Par exemple :

Si `T1 = [6, 2, 5, 3, 1, 0, 8, 4, 7]`, alors `T2 = [5, 4, 1, 3, 7, 2, 0, 8, 6]`

Bonus : écrire cette fonction avec une complexité en $O(n)$. On ne demande pas une preuve de cette complexité.

b) En déduire une fonction `test3` qui prend en argument une liste d'adjacence `G` d'un graphe et une liste `T1` et qui renvoie `True` si la condition **(C2)** est vérifiée pour `T1`, `False` sinon.

5) Écrire une fonction `estTriTopo` qui prend en argument un graphe `G` et une liste `T` et qui renvoie un booléen indiquant si `T` est un tri topologique de `G` ou non.

Partie II : quelques préliminaires

Soit G_0 un graphe. On dit que G est un sous-graphe de G_0 si G peut être obtenu en supprimant certains sommets dans G_0. Plus précisément, si on a $G_0 = (S_0, A_0)$, alors $G = (S, A)$ est un sous-graphe de G_0 si et seulement si :

$$S \subset S_0 \qquad \text{et} \qquad A = \{(u, v) \in A_0 \mid u \in S,\, v \in S\}$$

Par exemple, le graphe de la figure 3 est un sous-graphe de la figure 2, il est obtenu en supprimant les sommets 3 et 4. En Python, si le graphe initial G_0 contient n_0 sommets et est représenté par une liste d'adjacence `G0`, alors un sous-graphe G est représenté par un couple `G=(G0,S)` où `S` est une liste de taille n_0 telle que `S[u]` vaut True si et seulement si `u` est un sommet de `G`. Par exemple, si on note `Gfig3` le sous-graphe de `Gfig2` représenté en figure 3, on obtient :

```
Gfig3 = ( Gfig2, [True, True, True, False, False, True, True, True, True] )
```

6) Écrire une fonction `supprSommets` qui prend en entrée un sous-graphe `G=(G0,S)` ainsi qu'une liste `L`, et modifie `S` pour supprimer de `G` tous les sommets appartenant à `L`. On pourra supposer sans le vérifier que `L` ne contient que des sommets de `G`. Votre fonction doit modifier `S` et ne rien renvoyer.

7) Écrire une fonction `degEntrant` qui prend en entrée un sous-graphe `G=(G0,S)` avec `n` sommets et renvoie une liste `D` de taille `n` tel que `D[u]` est le degré entrant du sommet `u` dans `G`. Si `u` n'est pas un sommet de `G`, `D[u]` sera fixé à -1. Par exemple, pour `Gfig3`, on obtient :

```
[0,0,0,-1,-1,1,0,1,2]
```

Dans ce devoir, on parlera de *circuit* pour désigner un chemin de longueur strictement positive, dont les arcs sont distincts deux à deux et dont les sommets de départ et d'arrivée sont identiques.

8) Montrer que s'il existe un circuit dans un graphe `G` alors il n'existe pas de tri topologique pour `G`.

Dans la suite de ce devoir, on supposera que tous les graphes considérés ne contiennent pas de circuit.

Partie III : calcul d'un tri topologique par les degré entrants

Monsieur Sinus propose une méthode pour obtenir un tri topologique à partir du graphe de la figure 1. Il remarque tout d'abord que les sommets étiquetés par "chemise", "montre" , "caleçon" et "chaussettes" ont tous les quatre un degré entrant égal à 0. On peut donc les placer au début du tri topologique (dans un ordre arbitraire). En supprimant ces quatre sommets du graphe, les sommets "pantalon" et "cravate" ont pour degré entrant 0 et peuvent donc être placés à la suite dans le tri topologique. On continue ainsi jusqu'à ce que tous les sommets aient été supprimés du graphe. Avec cette procédure on obtient :

(T_2) : chemise, montre, caleçon, chaussettes, cravate, pantalon, chaussures, ceinture, veste.

9) Soit G_0 un graphe sans circuit et G un sous-graphe de G_0 avec au moins un sommet. Montrer qu'il existe au moins un sommet de G dont le degré entrant est 0.

10) En généralisant l'idée ayant permis de construire le tri topologique (T_2), écrire une fonction `triTopo1` qui renvoie un tri topologique de G.

Partie IV : la Tangente s'en mêle

Madame Tangente, l'amie en commun de M. Cosinus et de M. Sinus, pense que la solution de M. Sinus n'est pas optimale en termes de complexité. Elle propose donc d'accélérer la recherche des sommets ayant un degré entrant égal à 0. Son idée est d'utiliser trois variables :

- Une liste `T` initialement vide et qui contiendra le tri topologique à la fin de la procédure. Le sous-graphe G correspondra au sous-graphe de G_0 où on a supprimé les sommets déjà ajoutés à `T`.
- Une liste `D` contenant les degrés entrants des sommets dans le sous-graphe G (si le sommet a déjà été ajouté à `T`, on laisse la valeur correspondante dans `D` à 0).
- Une file `F` contenant les sommets ayant un degré entrant égal à 0, mais n'ayant pas encore été ajoutés à `T`. On pourra modéliser cette file par un deque, voir en Annexe.

À chaque étape, on défile `F` pour en retirer un sommet `u`. On ajoute `u` à la fin de `T` et on met à jour `D` en diminuant les degrés de tous les successeurs de `u` dans `G`. Par exemple, voici l'évolution des trois variables lors des quatre premières étapes de la procédure pour le graphe de la figure 2 :

	Étape 1	Étape 2	Étape 3	Étape 4	...
`T`	`[]`	`[0]`	`[0,1]`	`[0,1,2]`	...
`D`	`[0,0,0,3,1,1,0,2,2]`	`[0,0,0,3,0,1,0,2,1]`	`[0,0,0,3,0,1,0,2,1]`	`[0,0,0,2,0,0,0,2,1]`	...
`F`	`(6,2,1,0)`	`(4,6,2,1)`	`(4,6,2)`	`(5,4,6)`	...

11) Dans cette question, on s'intéresse à la procédure appliquée au graphe de la figure 2.

a) Donner le tableau pour les étapes 5, 6 et 7.

b) Quel tri topologique obtient-on à la fin de la procédure ?

12) Écrire une fonction `triTopo2` qui renvoie un tri topologique du graphe en entrée en suivant cette méthode. On garantira une complexité en $O(n+m)$ où n est le nombre de sommets dans G et m le nombre d'arcs. Pour la file `F`, on utilisera le type deque du module collections (voir ci-dessous).

Annexe. On rappelle que le module `collections` permet de gérer les deques.

- `Q = collections.deque()` crée un deque vide dans la variable `Q`.
- `Q.appendleft(...)` et `Q.append(...)` ajoutent un élément au début (respectivement à la fin) de `Q`.
- `v = Q.popleft()` et `v = Q.pop()` enlèvent un élément au début (respectivement à la fin) de `Q`, et stockent cet élément dans la variable `v`.