IR2 - Algorithmique des graphes

Fiche 4 - Graphes orientés acycliques

L'objectif de ce TP est d'implanter des algorithmes classiques sur les graphes orientés sans cycle (DAG). La structure de donnée à choisir est celle de **liste d'adjacence**. Le langage de programmation est le Java.

Exercice 1. Détection de DAG

1. Écrire une méthode qui teste si un graphe orienté est un DAG.
2. Évaluer la complexité temporelle et spatiale de l'algorithme utilisé.

Exercice 2. Tri topologique

Les DAG peuvent représenter des étapes d'un processus (plus ou moins) complexe : un arc partant du sommet x allant au sommet y signifie « x *doit être fait avant* y ». Par exemple, pour s'habiller le matin, le Professeur Cosinus met :

- son pantalon avant sa ceinture ;
- ses chaussettes avant ses chaussures ;
- sa chemise avant sa ceinture ;
- son caleçon avant son pantalon ;
- son pantalon avant ses chaussures ;
- sa chemise avant sa cravate ;
- sa cravate avant son pull ;
- son pull avant son chapeau ;
- sa montre n'importe quand.

1. Représenter le graphe qui modélise les relations décrites ci-dessus.
2. Un *tri topologique* d'un DAG à n sommets est une liste de sommets $(x_1, x_2, \ldots, x_n)$ telle que, pour tout $1 \leq i < j \leq n$, il n'existe pas de chemin de x_j à x_i. Autrement dit, c'est une façon d'ordonner les sommets du DAG qui respecte les étapes du processus modélisé. Bien entendu, il peut exister plusieurs tris topologiques pour un même graphe. Écrire une méthode qui retourne un tri topologique d'un DAG en s'inspirant de l'algorithme suivant qui est une variante de l'algorithme de parcours en profondeur :

Algorithm 1 TriTopologique(G)

Require: un graphe orienté acyclique G.
Ensure: retourne la liste L formant un tri topologique de G.

```
1: L ← ListeVide()
2: for tout sommet x de G do
3:    c(x) ← blanc
4: end for
5: for tout sommet x de G do
6:    if c(x) = blanc then
7:       Visiter(G, x)
8:    end if
9: end for
10: return L
```

Algorithm 2 VISITER(G, x)

Require: un graphe orienté G, x un sommet de G.
Ensure: visite les sommets de G depuis le sommet x.

```
1: c(x) ← gris
2: for tout sommet y successeur de x dans G do
3:    if c(y) = blanc then
4:       VISITER(G, y)
5:    end if
6: end for
7: c(x) ← noir
8: L ← x · L
```

Exercice 3. Nombre de chemins
On s'intéresse à compter le nombre de chemins entre deux sommets s et t d'un DAG.

1. En remarquant que le nombre de chemins de s à t est égal à la somme pour chaque successeur s' de s du nombre de chemins entre s' et t, écrire une méthode qui répond au problème donné.
2. Évaluer sa complexité temporelle et spatiale.