TD d'algorithmique avancée
Corrigé du TD : Graphe et Tri topologique

Jean-Michel Dischler

Un **tri topologique** d'un graphe orienté acyclique $G = (S, A)$ est un ordre linéaire des sommets de G tel que si G contient l'arc (u, v), u apparaît avant v. Le tri topologique d'un graphe peut être vu comme un alignement de ses sommets le long d'une ligne horizontale tel que tous les arcs soient orientés de gauche à droite.

Attention : le tri topologique d'un graphe orienté acyclique n'est pas forcément unique.

1. Proposez un tri topologique du graphe de la figure 1.

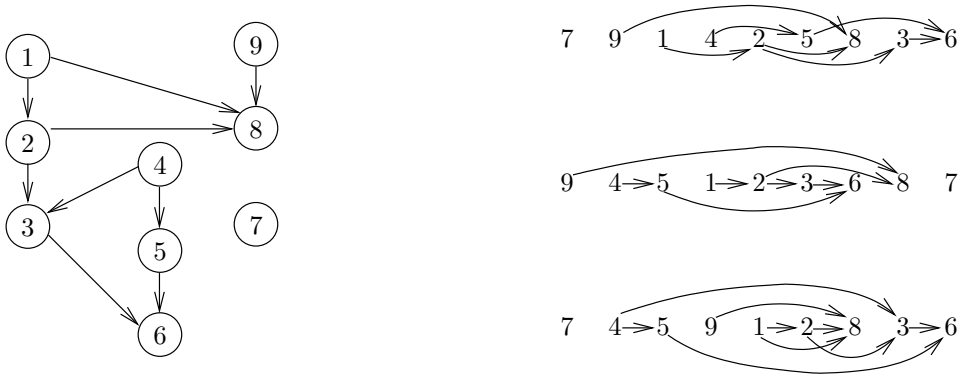

FIGURE 1 – Exemple de graphe orienté acyclique avec trois de ses tris topologiques possibles.

2. Modifiez l'algorithme de parcours en profondeur vu en cours pour qu'il calcule pour chaque nœud u sa date de fin de traitement *fin*$[u]$ (la date à laquelle lui et ses fils ont été traités).

```
PP(G)
  pour chaque sommet u de G faire couleur[u] ← BLANC
  pour chaque sommet u de G faire si couleur[u] = BLANC alors VISITER-PP(G, u, couleur)

VISITER-PP(G, s, couleur)
  couleur[s] ← GRIS
  pour chaque voisin v de s faire
    si couleur[v] = BLANC alors VISITER-PP(G, v, couleur)
  couleur[s] ← NOIR
  temps ← temps + 1
  fin[s] ← temps
```

3. Quel lien pouvez-vous faire entre les dates de fin de traitement et un tri topologique ?

   Si le graphe G contient l'arc (u, v), le traitement de u finira après *le traitement de v, et on aura donc fin$[u] >$ fin$[v]$. Trier les sommets par date de fin de traitements décroissantes nous fournit donc un tri topologique du graphe. La figure 2 présente le graphe de la figure 1 où les nœuds ont été annotés avec des fins de date de traitement lors d'un parcours en profondeur (¡¡ racines ¿¿ successives : nœuds 1, 9, 4 et 7). On retrouve alors le troisième des tris topologiques de la figure 1.*

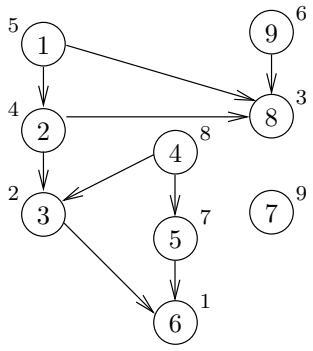

FIGURE 2 – Graphe annoté par les dates de fin de traitement d'un parcours en profondeur.

4. Proposez un algorithme de tri topologique. Quel est sa complexité ?

   On commence par parcourir le graphe en profondeur avec l'algorithme de la question 2, puis on trie les sommets par date de fin décroissante. La complexité de l'ensemble est donc celle du parcours $O(|S| + |A|)$ *plus celle du tri* $O(|S| \log(|S|))$*, soit* $O(|S| \log(|S|) + |A|)$*.*

5. Améliorez votre algorithme pour qu'il soit de complexité linéaire.

   PP(G)
   Soit P une pile initialement vide
   pour chaque sommet u de G **faire** *couleur*$[u] \leftarrow$ BLANC
   pour chaque sommet u de G **faire si** *couleur*$[u] =$ BLANC **alors** VISITER-PP(G, u, *couleur*, P)
   tant que non PILE-VIDE(P) **faire**
   $u \leftarrow$ DÉPILER(P)
   AFFICHER(u)

   VISITER-PP(G, s, *couleur*, P)
   couleur$[s] \leftarrow$ GRIS
   pour chaque voisin v de s **faire**
   si *couleur*$[v] =$ BLANC **alors** VISITER-PP(G, v, *couleur*, P)
   couleur$[s] \leftarrow$ NOIR
   EMPILER(P,s)

   On stocke dans une pile les éléments dans l'ordre dans lequel leur traitement se termine. On retirera les éléments de la pile dans l'ordre inverse de celui dans lequel ils ont été insérés (une pile fonctionne toujours sur le mode ¡¡ premier entré, dernier sorti ¿¿). Utiliser une pile plutôt que simplement les dates de fin de traitement nous évite d'avoir à trier ces dates (ce qui nous coûterait $O(|S| \log(|S|))$*. L'algorithme complet a ici un coût de* $O(|S| + |A|)$ *(le parcours est de coût* $O(|S| + |A|)$ *et la pile contient* $|S|$ *éléments, donc les dépiler est de coût* $O(|S|)$*).*

6. Proposez un autre algorithme de tri topologique, basé cette fois-ci sur le fait qu'un sommet de degré entrant nul peut être placé en tête d'un tri topologique. Quelle est la complexité de cet algorithme ?

   La figure 3 présente le second algorithme de tri topologique. Un sommet qui est placé dans la file, est un sommet dont tous les prédécesseurs ont déjà été ordonnés : on peut donc le placer à son tour dans l'ordre sans risquer de violer un arc. La complexité de l'ensemble est une fois de plus en $O(|S| + |A|)$*.*

```
TRI-TOPOLOGIQUE(G = (S, A))
  Soit F une file initialement vide
  pour chaque sommet u de G faire
    degré(u) ← degré entrant de u
    si degré(u) = 0 alors INSERTION(F, u)
  tant que non FILE-VIDE(F) faire
    u ← SUPPRESSION(F)
    AFFICHER(u)
    pour chaque voisin v de u faire
      degré(v) ←degré(v) − 1
      si degré(v) = 0 alors INSERTION(F, v)
```

FIGURE 3 – Deuxième algorithme de tri topologique.