# Tri topologique

*(inspiré de la version de Clarence Kineider)*

**Leçons : 903 ; 925 ; 927**

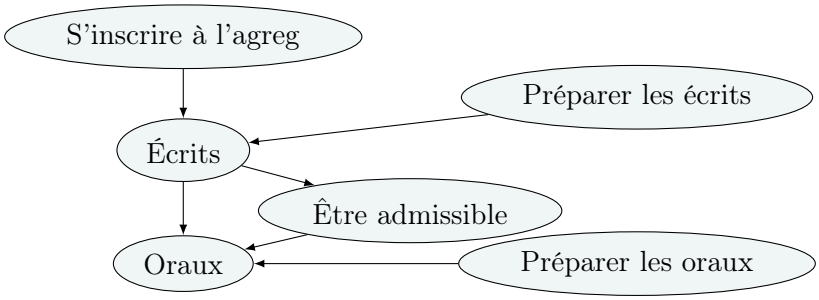
**Références : CORMEN–LEISERSON–RIVEST–STEIN, *Introduction à l'algorithmique* (p.566) [?] et LESESVRE–MONTAGNON–LE BARBENCHON–PIERRON, *131 développements pour l'oral* (p. 743) [?]**

**Introduction :**

Le problème du tri topologique est un problème d'ordonnancement des tâches : on a une liste de choses à faire, certaines ont besoin d'être faites avant d'autres. On représente ces contraintes par un graphe orienté dont les sommets sont les tâches et les arêtes sont les couples $(u, v)$ tels que la tâche $u$ doit être effectuée avant la tâche $v$. On cherche alors un ordre sur les sommets tel que si $(u, v)$ est une arête, alors $u$ apparait avant $v$ dans l'ordre. Le problème n'a pas de solution s'il y a un cycle dans le graphe, on va donc supposer que le graphe est acyclique.

**But :** Ordonnancer des tâches dont certaines doivent être faites avant d'autres.

**Exemple :**




Ordre possible : [Préparer les écrits, S'inscrire à l'agreg, Préparer les oraux, Écrits, Être admissible, Oraux]

**Cadre :** Un graphe acyclique orienté $G = (S, A)$

**Tri topologique**
- Entrée : Un graphe orienté acyclique $G = (S, A)$
- Sortie : Une liste $L$ constituée des sommets de $G$ telle que si $(u, v) \in A$, alors $u$ apparaît avant $v$ dans $L$

**Remarques :**

Il n'y a pas unicité de la solution (contrairement à un tri classique sur des valeurs distinctes).

Une façon de réaliser un tri topologique de $G$ est de faire un parcours en profondeur, et de placer les sommets dans la liste dans l'ordre inverse de l'ordre de visite. Ce développement consiste à donner cet algorithme et à justifier sa correction. Sa terminaison et sa complexité sont élémentaires et seront données également.

On commence par donner l'algorithme principal, qui utilise la fonction auxiliaire **Visite** donnée immédiatement après. Pour alléger les notations, on supposera que **Visite** a accès aux variables définies dans le corps de **TriTopo** (il suffit de définir **Visite** à l'intérieur de **TriTopo**).

**Algorithme 1 : TriTopo**

```
Procédure TriTopo(G)
    L ← [ ]
    Colorier tous les sommets en blanc
    [Par exemple avec un tableau de couleurs indexé par les sommets de G]
    Pour u ∈ V faire
        Si (u est blanc) alors
            Visite(u)
        Fin Si
    Fin Pour
    Retourner L
Fin
```

**Algorithme 2 : Visite**

```
Procédure Visite(u)
    Colorier u en gris
    Pour (u, v) ∈ A faire
        Si (v est blanc) alors
            Visite(v)
        Fin Si
    Fin Pour
    Colorier u en noir
    L ← u :: L
Fin
```

**Proposition 1.** L'algorithme **TriTopo** termine et a une complexité en $\Theta(|S| + |A|)$.

*Démonstration.* Le nombre de sommets blancs diminue strictement à chaque appel à **Visite**, donc l'algorithme termine. De plus, chaque sommet est visité exactement une fois, et chaque arête est traitée exactement une fois aussi, d'où une complexité en $\Theta(|S| + |A|)$. □

Pour démontrer la correction, on va utiliser le lemme suivant.

**Lemme 1.**
- Pour tout $v \in S$, $v$ est visité une unique fois, donc $v$ est ajouté une unique fois à $L$.
- Au début et à la fin de chaque appel à **Visite**, $L$ contient exactement l'ensemble des sommets noirs.
- Si au moment de l'appel à **Visite**$(v)$, le sommet $u$ est gris, alors il existe un chemin de $u$ à $v$ dans $G$.

*Démonstration.* Seul le troisième point n'est pas immédiat. Pour le démontrer proprement, faire une récurrence sur la profondeur de l'appel à **Visite**$(v)$ dans l'exécution de **Visite**$(u)$. □

**Théorème 1.** L'algorithme **TriTopo** est correct.

*Démonstration.* Soit $L$ la liste renvoyée par **TriTopo**$(G)$, et soit $(u, v) \in A$. Au moment de l'unique appel à **Visite**$(u)$, il y a trois possibilités :

— Si $v$ est noir, alors $v$ est déjà dans $L$. Donc $u$ est placé avant $v$ dans $L$.

— Si $v$ est gris, alors il existe un chemin de $v$ à $u$ dans $G$, et $(u, v) \in A$ ce qui contredit l'acyclicité de $G$. Absurde.

— Si $v$ est blanc, alors $v$ sera visité lors de la boucle sur les voisins de $u$. L'appel à **Visite**$(v)$ sera donc terminé à la sortie de la boucle, donc $v$ sera déjà dans $L$ au moment où $u$ y sera ajouté. Donc $u$ est avant $v$ dans $L$.

□

**Remarques :**

On peut voir le tri topologique comme un “vrai" algorithme de tri : on trie un ensemble muni d'un ordre partiel.

**Astuces de l'agrégatif :**

Souvent, l'exemple classique de cet algorithme est l'ordre dans lequel il faut s'habiller [1]. Dans la version de Clarence Kineider, il avait représenté une recette de cuisine. Pour ma part, sous l'influence de Lilian Besson, j'ai mis un exemple méta pour l'agrégation.

**Questions possibles :**

- Donner la solution que l'algorithme **TriTopo** renvoie sur l'exemple en partant de « S'inscrire à l'agrégation ».

1. il faut mettre son caleçon avec de mettre son pantalon, mais on peut mettre son tshirt avant/après le pantalon, etc.