

# Algorithmes et structures de données 2

## Laboratoire n° 2 : Graphes orientés

30.09.2020

### Introduction

Ce laboratoire a pour but de travailler les graphes orientés, il est composé de deux parties distinctes :

1. Dans la première partie, vous résoudrez un problème d'ordonnancement de modules en fonction de leurs prérequis.
2. Dans la seconde partie, vous étudierez différentes caractéristiques d'un réseau social modélisé sous la forme d'un graphe.

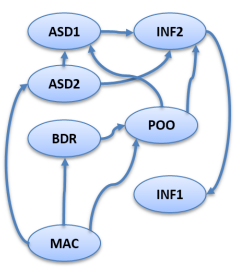
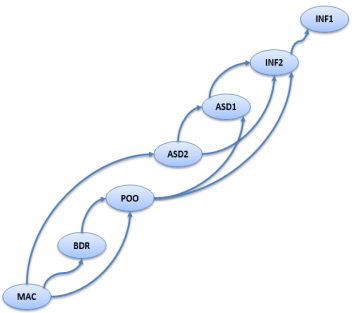
### Environnement de travail

- Pour expérimenter sur votre IDE, le plus simple est de télécharger les archives zip depuis Cyberlearn.
- Dans ce cas, faites attention à ce que les fichiers textes et d'images soient visibles par votre programme exécutable (dans CLion, ajouter le chemin du dossier de votre projet dans *Run -> Edit Configurations -> Working Directory*).
- Les fichiers textes utilisés pour construire les graphes doivent être passés en arguments de votre programme.

### Travail demandé

#### Exercice 1 – Ordonnancement de modules

- Le but de cet exercice est de résoudre un problème d'ordonnancement de modules en fonction de leurs prérequis.

Fichier prerequis.txt	Graphe orienté	Graphe trié (DFS + post-ordre)	Affichage inversé des sommets
ASD1, INF2 ASD2, ASD1, INF2 BDR, POO1 INF2, INF1 MAC, ASD2, BDR, POO1 POO1, ASD1, INF2			INF1 INF2 ASD1 ASD2 POO BDR MAC

- Les noms des modules et les prérequis sont fournis dans des fichiers texte, à partir desquels vous devrez construire des graphes de symboles (comme dans l'ex. 3 du labo 1).

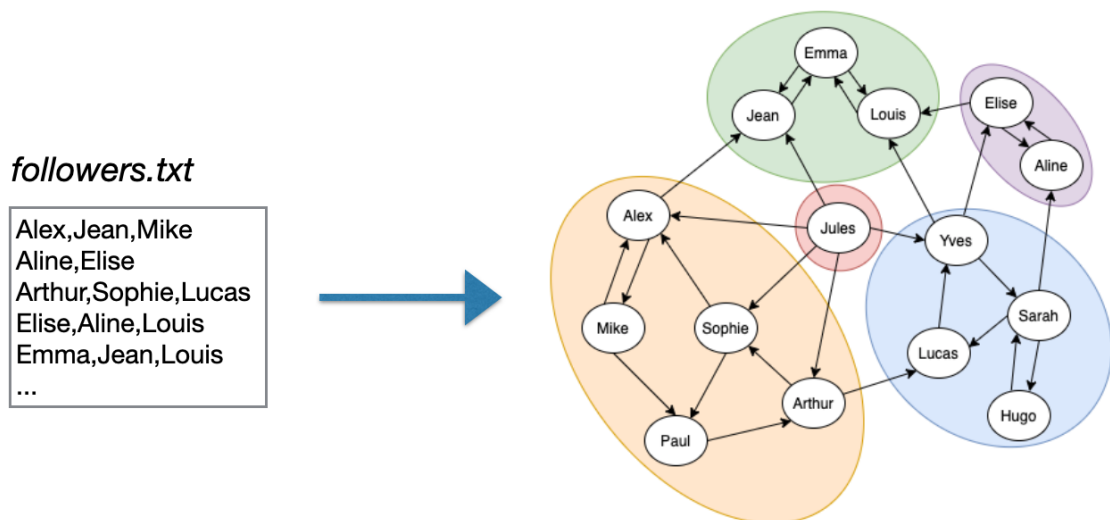
- Les connaissances nécessaires se trouvent dans les slides du cours « Graphes orientés » (surtout 23 à 25).
- Vous testerez le programme avec deux graphes fournis (fichiers `prerequis.txt` et `prerequis2.txt`), l'un ayant un cycle et l'autre non. Dans chaque fichier, chaque ligne contient un nom de module (première étiquette) suivi des modules prérequis.
- Vous devrez réutiliser la structure `SymbolGraph` du laboratoire précédent, soit dans votre propre implémentation, soit dans l'implémentation fournie dans ce labo (fichier `SymbolGraph.h`).
- On fournit la classe `DiGraph` implémentant un graphe orienté (*directed*) grâce à des listes d'adjacence.
- La détection de cycle dans un graphe se fera dans la classe générique `DirectedCycle`, dont certains éléments sont fournis et ne doivent pas être modifiés (voir le fichier `.h`). La détection devra afficher le premier cycle trouvé, ce qui nécessite l'adaptation de l'algorithme du slide 25.
- Le tri topologique se fera dans la classe générique `TopologicalSort`, dont certains éléments sont fournis et ne doivent pas être modifiés (voir le fichier `.h`).
- **Travail à faire :**
  1. Implémenter la détection d'un cycle dans `DirectedCycle.h` (éviter `.cpp` à cause des *templates*).
  2. Implémenter le tri topologique dans `TopologicalSort.h` (éviter `.cpp` à cause des *templates*).
  3. Écrire la fonction `main()` dans `main.cpp` pour obtenir des sorties comme ci-dessous (où `mod_X` seront des étiquettes de modules) en utilisant la méthode `checkOrder` fournie.
- **Résultat attendu :**

```
prerequis.txt est un DAG
Ordre topologique:
<mod_1> <mod_2> <mod_3> ...
Vérification réussie
```

```
prerequis2.txt n'est pas un DAG
Cycle trouve:
<mod_a> <mod_b> <mod_c> ... <mod_a>
```

## Exercice 2 – Réseau social

- L'objectif de cet exercice est d'analyser différentes caractéristiques d'un réseau social modélisé sous forme de graphe.
- Notre réseau social permet de suivre différentes personnes, une personne suivie ne nous suivant pas forcément. Il permet de lister les personnes que l'on suit ainsi que les personnes qui nous suivent. Il permet également de suivre de nouvelles personnes et de lister les personnes appartenant à notre groupe de relations. Ce groupe de relations correspond à la composante fortement connexe dans laquelle on se trouve.
- Les connaissances nécessaires se trouvent dans les slides du cours « Graphes orientés » (surtout 27 à 35). Pour mémoire, voici une représentation graphique de notre réseau social.



- Vous testerez le programme avec le graphe fourni (fichier `followers.txt`).
- Vous devrez réutiliser la structure `SymbolGraph` du laboratoire précédent, soit dans votre propre implémentation, soit dans l'implémentation fournie dans ce labo (fichier `SymbolGraph.h`). Attention à ce que le graphe `g` soit accessible depuis les sous-classes de `SymbolGraph`.
- On fournit la classe `DiGraph` implémentant un graphe orienté (*directed*) grâce à des listes d'adjacence.
- L'implémentation de l'algorithme de Kosaraju-Sharir se fera dans la classe `CFC`, dont certains éléments sont fournis et ne doivent pas être modifiés (voir le fichier `.h`).
- L'implémentation de notre réseau social se fera, elle, dans la classe `SocialNetwork`.
- **Travail à faire :**
  - Implémenter l'algorithme de Kosaraju-Sharir afin de détecter les cercles d'amis (correspond aux composantes fortement connexes).
  - Implémenter la classe `SocialNetwork` (sous-classe de `SymbolGraph`)
    - Constructeur

- `personsSubscribedBy(name)`
  - retourne les personnes auxquelles la personne "name" est abonnée
- `personsWhoFollows(name)`
  - retourne les personnes qui sont abonnées à la personne "name"
- `isFollowing(subscriber, personFollowed)`
  - détermine si la personne subscriber suit la personne personFollowed
- `addSubscription(subscriber, personToFollow)`
  - ajoute un nouvel abonnement au réseau social (subscriber suit personFollowed)
- `relationCircle(name)`
  - retourne les personnes faisant partie du cercle de relations de la personne "name"

## Modalités pratiques

- À rendre sur Cyberlearn au plus tard le dimanche **25.10.2020** à **23h59**.
- Pas de rapport à rendre pour ce laboratoire, mais soignez vos commentaires et entêtes.
- Travail à faire par groupes de trois étudiant-e-s. Durée : 6 périodes encadrées. Sources sur Cyberlearn.
- Respectez les consignes indiquées sur Cyberlearn.

**Bon travail !**