Algorithmique avancée **Master 1, Informatique**

Mini-projet — Tri topologique d'un graphe orienté sans circuits

16 janvier 2022

Le mini-projet, comptant pour la moitié du contrôle continu (coefficient 2 sur 12 ! comme le contrôle de novembre dernier) porte sur la programmation de l'algorithme du tri topologique tel qu'on l'a étudié en TD (feuille 7, exercice 4). **Ce projet est à rendre pour le 16 janvier 2022, délai de rigueur.**

Travail à faire et contraintes de programmation

Quelques indications sur ce qui est demandé et certaines contraintes à respecter :

— les graphes que l'on manipule sont des graphes orientés, linéaires (sans arcs parallèles) et sans boucles ils seront représentés par listes d'adjacence (plus précisément, les listes de successeurs des sommets) ;

— les sommets d'un graphe d'ordre n seront les entiers de 1 à n (et pas de 0 à $n-1$, attention !) ;

— un programme devra lire le graphe sur son entrée standard ou dans un fichier, dans un format **strict** donné plus bas, puis calculer un tri topologique de ce graphe, s'il en existe un, ou bien identifier un circuit dans le graphe qui interdit le tri topologique du graphe ; cette dernière contrainte signifie très précisément que si votre programme détecte un circuit dans le graphe, il doit fournir ce circuit sous la forme d'une suite de sommets x_1, x_2, x_3, ..., x_k (si le circuit est de longueur k), l'arc (x_k, x_1) refermant ce circuit.

Le format d'entrée des graphes est une suite de nombres entiers constituée de la manière suivante :

— un premier entier, n, qui est l'ordre du graphe considéré ;

— puis n suites de nombres entiers structurées ainsi :
  — un entier x compris entre 1 et n, représentant un sommet,
  — une suite d'entiers compris entre 1 et n, représentant la suite de successeurs du sommet x,
  — l'entier 0 indiquant la fin de la suite de successeurs en question.

Par exemple, la description :

```
1000
1 2 9 11 4 6 0
2 3 7 13 0
3 0
. . . . . .
999 1 4 8 16 0
1000 3 5 7 9 11 13 15 17 19 0
```

définit un graphe d'ordre 1000, dont la suite de successeurs du sommet 1 est la suite $(2, 9, 11, 4, 6)$, celle du sommet 2 est $(3, 7, 13)$, celle du sommet 3 est vide, etc., celle du sommet 999 est $(1, 4, 8, 16)$ et enfin, celle du sommet 1000 est $(3, 5, 7, 9, 11, 13, 15, 17, 19)$

Tout autre format sera rejeté immédiatement par le programme[1] ; l'intérêt d'un tel format est que les graphes exemples peuvent être utilisés par tous les programmes sans avoir besoin de se préoccuper de telle ou telle conversion de format[2]. Si vous devez afficher un graphe, pour telle ou telle raison, vous le ferez dans ce format-là (si vous voulez des affichages plus sexy, vous pourrez en ajouter, mais vous devrez au moins avoir celui-ci).

La programmation devra être soignée, chaque entité devant être soigneusement spécifiée et décrite (dans un document annexé au compte-rendu que j'appelle « rapport » dans la suite). Vous pouvez écrire ce programme en Ocaml, C++ ou en Java (si vous préférez un autre langage, demandez notre accord). **Vous n'êtes pas autorisés à utiliser des composants tout faits, qui ne sont pas de votre cru** (sauf autorisation préalable). Votre programme doit, a minima, avoir un mode de fonctionnement dans lequel il lit le graphe sur l'entrée standard (dans le format décrit plus haut),

1. Et par le correcteur !
2. Et d'ailleurs, je vous fournirai une collection de graphes tout faits décrits dans ce format-là.

fait son calcul et affiche le résultat du tri topologique sur la sortie standard (s'il y a un circuit dans le graphe, le circuit sera affiché sur la sortie standard et un message d'erreur sera affiché sur la sortie standard d'erreur).

Le travail est à faire en binôme de préférence, prévenez-nous si vous le faites seul ; lorsque vous rendez le travail, un seul rendu est exigé pour les deux membres du binôme. Aucun travail fait à trois ne sera accepté.

Travail à rendre

Vous rendrez une archive zip ou tar ou tar.gz, au choix, à l'exclusion de tout autre format (des points seront défalqués si l'archive n'est pas dans l'un de ces formats). Cette archive devra se décompresser pour créer un unique répertoire (portant les noms de famille en minuscules des deux membres du binôme, séparés par un tiret, et dans l'ordre alphabétique [par exemple, `bergey-naudin`]) dans lequel se trouvera tout votre travail sous une forme immédiatement testable par nous. L'archive aura le même nom que le répertoire principal, avec l'extension idoine.

Dans ce répertoire principal, il devra y avoir, a minima, la racine de la hiérarchie de fichiers sources que nous pourrons compiler, accompagnée d'un makefile ou d'un shell-script permettant de construire l'application ; nous n'utiliserons aucun exécutable tout fait, il faut donc que nous puissions compiler sur nos machines (Java en version 8 maximum). Si vous utilisez un IDE, vous prendrez soin de nettoyer l'archive que vous nous envoyez de tous les fichiers spécifiques à votre IDE.

Le rapport de projet peut également apparaître dans ce répertoire principal, joint à l'archive des sources Java, il devra être fourni en format PDF (ou texte brut) à l'exclusion de tout autre format plus ou moins exotique (dans le cas contraire, nous considérerons qu'il n'y a pas de rapport).

Comme vous l'avez compris, il s'agit de programmer l'algorithme que nous avons étudié en TD (exercice 8 de la feuille 7), une partie de votre rapport devra être consacré à une rédaction personnelle de la solution de cet exercice.

Considérations annexes

Je vous fournis (ce sera déposé bientôt sur la page du cours) un programme (Java) qui construit des graphes aléatoires sans circuit dont vous pouvez choisir l'ordre. Je vous le fournis sous la forme d'une archive Jar valide pour la distribution JDK 8, vous ne pourrez pas utiliser cette archive Jar avec une autre version de Java[3]. Voilà comment on peut utiliser ce programme de génération de graphes aléatoires sans circuit (exemple exécuté sur un système de type Unix) :

```
$ java -jar randomdag.jar 10 0.3
10
1 0
2 0
3 2 0
4 8 0
5 7 10 9 8 2 0
6 0
7 4 6 10 3 0
8 0
9 0
10 3 9 0
```

On a demandé un graphe sans circuit d'ordre 10 avec une probabilité d'existence d'arc qui est de 0.3, et on voit le résultat fourni par le programme[4].

3. C'est du JDK version Oracle, pas du Open JDK. Pour l'instant, je n'envisage pas de fournir une version du Jar pour d'autres versions de Java (il me semble que Java 8 est installé sur toutes les machines du BE).

4. Attention : il faut un peu ajuster la probabilité en fonction de l'ordre du graphe si on veut avoir des graphes intéressants ; typiquement si vous demandez un graphe d'ordre 10 avec une probabilité d'existence d'arc de 10 %, vous vous retrouverez fort probablement avec un graphe ayant 4 ou 5 arcs. Avec une probabilité de 0.3, comme ici, on devrait avoir environ une quinzaine d'arcs, dans l'exemple il y en a 14.