COURS DE THÉORIE DES GRAPHES
ORGANISATION PRATIQUE & PROJET

Pour le **lundi 23 octobre** 2017, chaque étudiant aura choisi les modalités d'examen le concernant :

1) projet d'implémentation, examen écrit (exercices et théorie vue au cours) ;
2) pas de projet, exercices et partie théorique étendue.

Les délégués des différentes sections fourniront, par mail, la **liste des choix retenus**.

La note ci-dessous détaille le projet d'implémentation.

1. Extrait de l'engagement pédagogique MATH-0499

[...] Un projet d'implémentation, par groupes de deux, intervient (pour ceux qui en font le choix) dans la note finale. Ce projet nécessite en plus de fournir un code C, la production d'un rapport écrit court devant faciliter la compréhension du code et la défense orale de celui-ci (questions individuelles). Sauf mention explicite, les différents groupes ne peuvent ni collaborer, ni s'inspirer du code d'un autre groupe. [...]

2. Le code source

Nous utilisons les mêmes **consignes** que le cours INFO-0030 (projet de programmation) et les résumons ci-dessous.

Le code source sera fourni en C "standard" et utilisera les bibliothèques usuelles. Votre code doit pouvoir être compilé, sans erreur (ni 'warning'), sous `gcc`. Si des options particulières sont nécessaires à la compilation, par exemple `--std=c99`, il est indispensable de le mentionner en préambule (ou de fournir un Makefile). Un code ne compilant pas entraîne une note de zéro au projet.

Quelques consignes qu'il est indispensable de respecter :

— Le choix des noms de variables et de sous-programmes doit faciliter la lecture et la compréhension du code.
— L'emploi de commentaires judicieux est indispensable : entrée/sortie des différents sous-programmes, points clés à commenter, boucles, etc.
— Enfin, l'indentation et l'aération doivent aussi faciliter la lecture de votre code en identifiant les principaux blocs.

Un code peu clair, même si le programme "tourne", sera pénalisé.

Une interface rudimentaire `graphes.c`/`graphes.h` est disponible en ligne sur `http://www.discmath.ulg.ac.be/`. Celle-ci est détaillée à la fin des notes de cours (chapitre V). Libre à vous de l'utiliser ou non, voire de l'améliorer.

3. Le rapport

Le rapport ne doit pas être un copier-coller du code source (ce dernier étant fourni par ailleurs). Le rapport, au format `pdf` et idéalement rédigé sous LaTeX, est court : maximum 5 pages. Il doit décrire la stratégie utilisée, les choix opérés, les grandes étapes des différentes procédures ou fonctions. Il pourra aussi présenter les difficultés/challenges rencontrés en cours d'élaboration ou reprendre certains résultats expérimentaux (benchmarking sur des exemples types ou générés aléatoirement).

4. La présentation orale

La présentation est limitée à **10 minutes** maximum. Sans que cela soit nécessaire, les étudiants ont le droit d'utiliser un ordinateur (pour faire tourner leur programme, pour présenter leur code, pour présenter leur travail avec un support type "power point"). Un projecteur vidéo est à disposition. Cette présentation se veut être une synthèse du travail fourni.

Elle est suivie par une **séance de questions**. Le but de ces questions est de déterminer la contribution et l'implication de chacun. Ainsi, des questions différentes seront posées individuellement et alternativement aux deux membres du groupe. L'ensemble présentation/questions ne devrait pas dépasser 20 minutes. Un ordre de passage des différents groupes sera déterminé.

5. Dates importantes

- **lundi 23 octobre 2015** : choix individuel des modalités d'examen, répartition en groupes et choix des sujets.
- **vendredi 15 décembre 2016** : dépôt du code et du rapport sous forme d'une archive envoyée par mail au titulaire du cours. Cette archive contiendra deux répertoires, un pour le code à compiler, l'autre pour le rapport.
- **semaine du 18 décembre 2016** — ordre de passage à déterminer : présentation orale.
- **janvier 2016** : examen écrit (commun pour tous).
- **janvier 2016** : examen oral pour les étudiants ayant choisi l'option 2 uniquement.

6. Les projets

Sauf problème, les étudiants proposent une **répartition par groupes** de deux (en cas d'un nombre impair d'étudiants, un unique groupe de 3 étudiants sera autorisé) et l'**attribution des sujets** aux différents groupes (un même sujet ne peut pas être donné à plus de 2 groupes — le sujet choisi par l'éventuel groupe de 3 étudiants ne peut être attribué à un second groupe). La répartition devra être validée par le titulaire du cours.

Si un accord entre les étudiants n'est pas trouvé, le titulaire procédera à un tirage au sort (des groupes et des sujets).

Le plagiat est, bien entendu, interdit : il est interdit d'échanger des solutions complètes, partielles ou de les récupérer sur Internet. Néanmoins, vous êtes encouragés à discuter entre groupes. En particulier, il vous est

loisible d'utiliser des fonctions développées par d'autres groupes (et qui ne font pas partie du travail qui vous est assigné). Mentionner les sources utilisées/consultées.

Les projets listés ci-dessous sont "génériques", il est loisible à chaque groupe d'aller plus loin, d'adapter et de développer plus en avant les fonctionnalités de son code (par exemple, meilleure gestion des entrées/sorties, optimisation des structures de données, fournir des exemples "types" dans un fichier, etc.).

Vérifiez que votre solution tourne même sur les cas pathologiques (par exemple, quel est le comportement attendu, si le graphe fourni n'est pas connexe, ne satisfait pas aux hypothèses, si le fichier est mal structuré, etc.). Essayez de construire un ensemble "témoin" de graphes "tests" sur lesquels faire tourner votre code. Tous les projets n'ont pas la même difficulté, il en sera tenu compte pour la cotation.

(1) *Noyau d'un graphe.* On donne un graphe simple et orienté $G = (V, E)$ sans cycle. Le noyau de G est l'unique sous-ensemble N de sommets vérifiant les deux propriétés suivantes
— stable : $\forall u, v \in N$, $(u, v) \notin E$ et $(v, u) \notin E$
— absorbant : $\forall u \notin N$, $\exists v \in N : (u, v) \in E$.
Etant donné un graphe, être en mesure de fournir son noyau. On appliquera l'algorithme à des graphes issus de la théorie des jeux : étant donnés deux entiers $k, \ell \geq 0$ considérés comme paramètres fournis par l'utilisateur, on considère le graphe dont les sommets sont les couples (x, y) avec $x \leq k$ et $y \leq \ell$. Il y a un arc de (x, y) à (x', y') si et seulement si $x = x'$ et $y' < y$ ou, $x' < x$ et $y = y'$. Dans un second temps, on ajoutera aux arcs précédents, des arcs de (x, y) à (x', y') avec la condition $x - x' = y - y' > 0$.

(2) *Recherche d'un "matching" dans un graphe biparti.* On présentera tout d'abord la notion de matching (définition, existence d'un matching parfait, applications, difficulté du problème) avant d'implementer un algorithme (par exemple, l'algorithme de Hopcroft–Karp) recherchant un matching de taille maximum au sein d'un graphe biparti.

(3) *Génération aléatoire de graphes, modèle de Barabási–Albert.* On présentera d'abord le modèle d'un point de vue théorique. Ensuite, on l'implémentera pour générer des graphes aléatoires. L'implementation devra permettre d'afficher graphiquement les graphes obtenus. Le but est de pouvoir générer des graphes de grande taille. On pourra, par exemple, générer une sortie utilisable par un utilitaire de visualisation de graphes, comme GraphViz `http://www.graphviz.org/`.

(4) *Le problème de l'isomorphisme de graphes.* On donne deux graphes (tous les deux orientés ou non) et on veut décider s'ils sont ou non isomorphes. En cas de réponse positive, on donnera un isomorphisme entre eux.

(5) *Enumération de circuits hamiltoniens - le parcours du cavalier.* Etant donné un échiquier de dimension $m \times n$, énumérer tous les circuits hamiltoniens réalisés en utilisant uniquement les mouvements du cavalier dans le jeu d'échecs. Même question, mais en énumérant le nombre de chemins hamiltoniens "ouverts" (origine et destination différentes).

(6) *Graphes sans cycle et tri topologique.* Etant donné un digraphe simple, déterminer si celui-ci est sans cycle et fournir, le cas échéant, un tri topologique des sommets. Si un graphe possède un cycle, être en mesure de lister ses circuits simples. Si on fournit un multigraphe orienté en entrée, le remplacer par un graphe orienté 'équivalent' (boucles supprimées et multi-arcs remplacés par un unique arc). Enfin, on soignera la "sortie" que ce soit l'affichage à l'écran de la solution proposée ou, sa sauvegarde dans un fichier structuré. Une seconde partie du travail consiste à renuméroter les sommets en respectant le tri topologique trouvé.

(7) *Algorithme de Prim.* Etant donné un graphe simple connexe pondéré, trouver un arbre de poids minimal en utilisant l'algorithme de Prim. Si on fournit un multigraphe en entrée, le remplacer par un graphe 'équivalent' (boucles supprimées et multi-arêtes remplacées par un unique arête de poids minimal). La première partie de ce travail nécessite une petite recherche bibliographique et le rapport expliquera l'algorithme. Dans la mesure du possible, si plusieurs arbres réalisent la meilleure borne, fournir ces alternatives. Effectuer une comparaison théorique (pas d'implémentation) avec l'algorithme de Kruskal).

(8) *Algorithme de Kruskal.* Etant donné un graphe simple connexe pondéré, trouver un arbre de poids minimal en utilisant l'algorithme de Kruskal. Si on fournit un multigraphe en entrée, le remplacer par un graphe 'équivalent' (boucles supprimées et multi-arêtes remplacées par un unique arête de poids minimal). La première partie de ce travail nécessite une petite recherche bibliographique et le rapport expliquera l'algorithme. Dans la mesure du possible, si plusieurs arbres réalisent la meilleure borne, fournir ces alternatives. Effectuer une comparaison théorique (pas d'implémentation) avec l'algorithme de Prim).

(9) *Recherche d'un chemin simple de longueur maximale.* On fournit un graphe (orienté ou non) et la question est de déterminer un chemin simple (ne passant pas deux fois par le même sommet) de longueur maximale. Dans un second temps, on considérera la version pondérée de ce problème. Quelles sont les applications possibles de cette recherche ?

(10) *Détection des arêtes de coupure et algorithme de Fleury.* Implémenter un test permettant de déterminer si une arête est ou non une arête de coupure et implémenter l'algorithme de Fleury vu au cours. Dans un second temps, pouvoir énumérer (lister) tous les circuits eulériens d'un graphe eulérien.

(11) *Ensemble de domination.* Soit $G = (V, E)$ un graphe simple non orienté. On dit que $W \subset V$ est un *ensemble de domination* de G, si tout sommet de V est un élément de W ou voisin d'un élément de W. Pour un

graphe G donné et un sous-ensemble de sommets W, tester si W est un ensemble de domination. Ensuite, pour un graphe G donné, fournir un ensemble de domination de taille minimale. (Attention : ce dernier problème est connu pour être NP-complet, il n'existe donc pas, à ce jour, de solution efficace.) Explorer des heuristiques possibles.

(12) *Couverture par des sommets.* Soit $G = (V, E)$ un graphe simple non orienté. On dit que $W \subset V$ est une *couverture* de G si toute arête de G possède au moins une extrémité dans W. Pour un graphe G donné et un sous-ensemble de sommets W, tester si W est une couverture. Ensuite, pour un graphe G donné, fournir une couverture de taille minimale. (Attention : ce dernier problème est connu pour être NP-complet, il n'existe donc pas, à ce jour, de solution efficace.) Explorer des heuristiques possibles.

(13) *Test de planarité.* Etant donné un graphe simple, déterminer si celui-ci est ou non planaire (on pourra utiliser le théorème de Kuratowski). S'il n'est pas planaire, mettre en évidence un sous-graphe homéomorphe à K_5 ou $K_{3,3}$. Le programme fournit donc une preuve de non-planarité.

(14) *Algorithme de Kosaraju—Sharir.* Implémentation de cet algorithme permettant de rechercher les composantes fortement connexes d'un graphe orienté. On peut se limiter au cas de graphes simples. On comparera, d'un point de vue théorique, cet algorithme avec celui de Tarjan.

(15) *Coloriage d'arêtes.* Etant donné un graphe simple non orienté, on demande de colorier les arêtes du graphe de façon telle que des arêtes adjacentes en un même sommet reçoivent des couleurs distinctes. Ainsi, si, le nombre de couleurs nécessaires est au moins égal à $\max_{v \in V} \deg(v)$. Voir par exemple, `http://www.cs.utexas.edu/users/psp/vizing.pdf`.

(16) *Détection de communautés.* Ce projet est plus ambitieux : comprendre et implémenter la méthode décrite dans le papier repris sur `http://arxiv.org/abs/0803.0476`.

(17) *Calcul de la "centralité" d'un sommet.* Comme le précédent, ce projet est ambitieux : comprendre et implémenter la méthode décrite dans le papier repris sur `http://www.inf.uni-konstanz.de/algo/publications/b-fabc-01.pdf`.

(18) *Cloture transitive.* Etant donné un graphe orienté, fournir de la façon la plus efficace possible la cloture transitive de ce dernier.

(19) *Construction d'un graphe réel de collaborations.* Pouvoir interroger une base de données comme `http://www.ams.org/mathscinet/` ou `orbi.ulg.ac.be` pour construire un graphe (pondéré non orienté) dont les sommets sont les auteurs et une arête existe entre deux auteurs s'ils ont une publication commune. Le poids d'une arête est fourni par le nombre de publications communes. En interrogeant ces bases de données, l'utilisateur pourra fournir certains critères (pour ne pas

obtenir un graphe trop gros) comme une liste de noms, certaines dates, etc. ou construire tous les sommets à distance au plus d d'un sommet donné.

Quelques conseils :

- Pensez à l'utilisateur qui teste votre programme : préparer un makefile, donner des conseils sur l'utilisation (fournir quelques fichiers de test), quelles entrées fournir, quelles sorties attendues ? Décrivez un exemple typique d'utilisation.
- Préparez une petite bibliographie, citer les sources utilisées (même les pages Wikipédia !). Si vous avez exploiter une source, un autre cours, mentionnez-le explicitement !
- Avez-vous tester votre programme sur de gros graphes ? De quelles tailles ? Eventuellement produire un petit tableau de "benchmarking" indiquant, sur une machine donnée, le temps de calcul en fonction des tailles de graphes testés.
- Relisez (et relisez encore) votre rapport ! Faites attention à l'orthographe (accords, conjugaison), au style.
- Si vous développez des heuristiques, avez-vous des exemples de graphes (ou de familles de graphes) qui se comportent mal par rapport à cette heuristique ?