Projet 1 - Projet en Java
L3IF - ENS Lyon 2013-2014

Petru Valicov <petru.valicov@ens-lyon.fr>

Préliminaires

Date de rendu (stricte cette fois-ci) : 7 Janvier 2013 à 23h59
AUCUN RETARD NE SERA ACCEPTE !

Généralités

- Les projets sont à faire obligatoirement en binôme.
- Le rendu du projet se fait par mail au responsable sous la forme d'une archive `.tar.gz` ou `.zip` qui contiendra l'intégralité du code ainsi qu'un fichier INSTALL/README contenant les indications pour compiler et exécuter votre programme... IL NE FAUT PAS inclure les fichiers `.class` dans votre archive !
- L'objet du mail doit contenir l'expression suivante : [Projet1][Java]
- L'archive tar.gz ou .zip portera vos deux noms de famille accolés. Par exemple, si les noms sont Dupont et Martin, cette dernière s'appellera :

  `DupontMartin.tar.gz` OU `MartinDupont.tar.gz`

- Vous allez également devoir rendre un rapport (entre 5 et 10 pages).
  - Vous y expliquerez ce que votre projet fait, ainsi que la liste des options implémentées.
  - Vous justifierez les choix que vous aurez effectués durant l'élaboration de votre programme (structures de données, algorithmes...) et discuterez les limites de votre implémentation.
  - Le rapport devra être rédigé en LaTeX et exporté ensuite en `.pdf` . Aucun rapport dans un autre format ne sera accepté !

Remarque
La note du projet prendra en compte :
- La propreté et la lisibilité du code ainsi que tout ce qui facilitera sa compréhension par les correcteurs (noms des variables, commentaires, modularité, etc).
- La facilité d'utilisation du code (le fichier INSTALL/README est-il lisible et suffit-il pour utiliser le code ?). La compilation de votre code devrait marcher sur tout type de machine - pensez à tester avant la rendue finale.
- La correction et la performance du code
- La modélisation objet - si vous avez une seule classe c'est 0 !
- La qualité de la rédaction du rapport

Instructions

- Le projet doit répondre à toutes les instructions obligatoires (les Instructions 1-4) et au moins une des deux instructions optionnelles (Instruction 5 ou Instruction 6).
- Vous pouvez aussi implémenter des bonus que vous auriez imaginés. Il est cependant conseillé de venir en parler avec votre responsable de projet afin d'en discuter.
- Il est **strictement interdit** de copier du code du projet des vos collègues. Attention : l'obfuscation du code n'est pas une solution, c'est très facile à détecter !

Ce que fait un bon programmeur : rappels

Même si cela peut paraître évident, voici quelques règles indispensables à la survie d'un programmeur.

- **Toute fonction écrite doit être testée ! ! ! ! ! !**

- Un test contient généralement la ligne de commande avec ses arguments pour lancer le test AINSI QUE le résultat qui s'affiche à l'écran. Si c'est un affichage graphique, une impression d'écran conviendra.

- Chaque test devra être ajouté au dossier /test qui contient les tests de toutes les fonctions que vous aurez testées au cours de la réalisation du projet (même les fonctions écrites pendant la période TP dont vous vous resservez).

- Le code doit être organisé en différents fichiers au contenu cohérent.

CONSEIL : pensez à utiliser des environnements de développement intégré (IDE) tel que Eclipse par exemple. La complétion automatique, la refactorisation du code (renommage des variables, des fonctions ou des classes) ou encore l'indentation automatique sont seulement quelques uns des avantages de ces outils.

Le couplage maximum

1 Comment trouver un maximum de couples heureux ?

Le problème du couplage maximum consiste, étant donné un graph non-orienté, à trouver le plus grand ensemble d'arêtes indépendantes (les extrémités sont toutes distinctes). Il s'agit d'un problème d'optimisation classique et dont l'algorithme polynomial a été publié par Jack Edmonds en 1965.

L'enjeu du projet est de réaliser un programme qui permette de trouver un couplage maximum dans un graphe saisi par l'utilisateur avec la possibilité de visualisation de la solution sur une interface graphique. Une option est de visualiser le déroulement de l'algorithme sur cette interface. Elle est en bonus.

Pour réaliser ce projet, plusieurs étapes relativement indépendantes sont nécessaires :

- la mise en place d'une architecture pour représenter vos structures de données
- l'implémentation de l'algorithme résolvant le problème lui-même
- l'ajout d'une couche graphique pour pouvoir représenter le graphe et la solution dans une fenêtre

- Bonus : l'implémentation d'un mode interactif où l'utilisateur peut choisir de dérouler l'algorithme à la main et visualiser chaque étape. Dans ce cas, étant assez méchant, il pourra modifier éventuellement la solution courante.

2 Les chemins pleins de fleurs dans la forêt

Voici quelques définitions permettant d'exprimer plus formellement le problème. Tout au long du sujet vous serez amenés à faire la distinction entre un ensemble maximal et maximum.

Définition 2.1. *Un* graphe non-orienté *G est une paire $G = (V, E)$ où :*

- *V est un ensemble de points, que nous appellerons* sommets *ou* nœuds
- *E est un ensemble de liens, aussi appelés* arêtes*, qui sont des paires de sommets. Pour deux sommets adjacents u et v, on notera uv l'arête qui les relie.*

Dans ce qui suit on considérera uniquement des graphes simples – entre deux sommets u et v il ne peut pas y avoir deux arêtes distinctes.

Définition 2.2. *Un couplage M d'un graphe G est un sous-ensemble d'arêtes de G ($M \subseteq E$) telles que $\forall (uv, u'v') \in M^2$ avec $u \neq u'$, $\{u, v\} \cap \{u', v'\}' = \emptyset$. Un exemple est donné dans la Figure 1.*

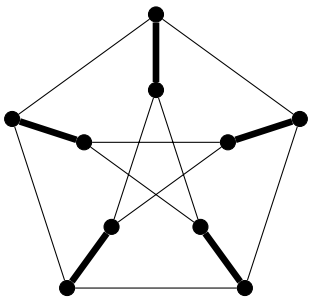


FIGURE 1 – Un exemple de graphe avec un couplage

Définition 2.3. *Un* chemin *est une suite de sommets $x_1, x_2, \ldots, x_k$ telle que $(x_i, x_{i+1}) \in E$ pour chaque $1 \leq i \leq k-1$. La longueur d'un chemin et son nombre d'arêtes.*

Définition 2.4. *Étant donnée un couplage M d'un graphe G, un* chemin alternant *dans G est un chemin dont les arêtes sont alternativement dans M et hors de M (donc si une arête sur deux est dans M). On dit qu'un sommet v est* couplé *s'il existe une arête uv dans M, sinon le sommet est dit* libre*. Finalement, un* chemin augmentant *de M est un chemin alternant dont les extrémités sont libres.*

Par la suite on utilisera souvent le résultat suivant :

Théorème 2.5 (Berge, 1957)**.** *Un couplage M est maximum si et seulement si il ne contient pas de chemin augmentant.*

Définition 2.6. *On dit qu'un graphe est* connexe *si pour tous sommets u et v, tels que $u, v \in V$, il existe un chemin de u à v.*

Définition 2.7. *Un* cycle *est un chemin $x_1, x_2, \ldots, x_k$ dont tous les sommets sont distincts, sauf les extrémités qui sont égales : $x_1 = x_k$. La longueur d'un cycle est donnée par le nombre de ces sommets (ou de ces arêtes).*

Définition 2.8. *La contraction d'une arête uv d'un graphe G est l'opération d'identification des sommets u et v tel que le nouveau sommets a comme voisins l'ensemble des voisins de u et de v. Contracter un chemin $x_1, x_2, \ldots, x_k$ (ou tout autre sous-graphe) de G, revient à contracter successivement les arêtes du chemin jusqu'à obtenir un seul sommet y qui a comme voisins l'ensemble des voisins de chaque sommet $x_1, x_2, \ldots, x_k$.*

Définition 2.9. *Un* arbre *est un graphe connexe sans cycle. Une* forêt *est un graphe tel que toutes ces composantes connexes sont des arbres.*

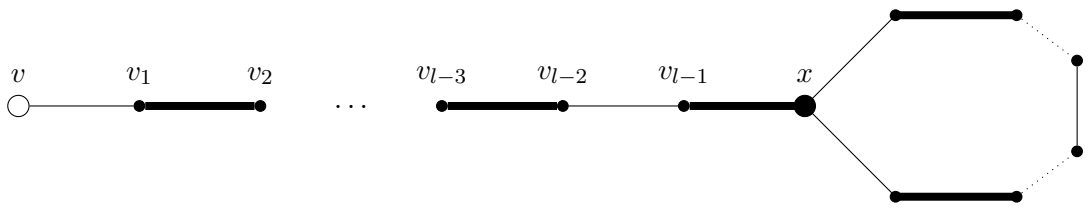
Le Théorème 2.5 mène à l'algorithme suivant pour le calcul d'un couplage maximum :

Algorithme 1 : Algorithme *MM*

Input : $G = (V, E)$ un graphe
Output : Un couplage maximum
1 Choisir aléatoirement un couplage maximal M.
2 S'il existe un chemin augmentant, mettre à jour le couplage M
3 Sinon, le couplage M est maximum

Un couplage maximal peut être obtenu avec un simple algorithme glouton : on part d'un ensemble vide d'arêtes et tant qu'il existe une arête qui ne touchent aucune autre arête de l'ensemble courant, on la rajoute dans l'ensemble. Naturellement, le problème principal est de trouver un chemin augmentant. Pour cela l'originalité de Jack Edmonds nous a offert la notion de *fleur* :

Définition 2.10. *Soit M un couplage d'un graphe G. Une* fleur *est un cycle de longueur $2k+1$ contenant k arêtes du couplage tel que ce cycle contient un sommet x qui se trouve sur un chemin alternant de longueur paire $v, v_1, v_2, \ldots, v_{l-2}, v_{l-1}, x$ (la tige) où v est un sommet libre. Ci-dessous un exemple.*



Lemme 2.11 (Le lemme de contraction de cycle)**.** *Soit G un graphe, M un couplage de G et B une fleur de M. Soit G' et M', le graphe et respectivement le couplage, obtenus à partir de G en contractant le cycle B. Alors, il existe dans G un chemin augmentant de M si et seulement s'il existe dans G' un chemin augmentant de M'.*

La démonstration de ce lemme est laissée à titre d'exercice. Ce dernier résultat est la dernière (enfin presque ☺) pièce du puzzle. Maintenant on est prêt à décrire l'algorithme pour trouver un chemin augmentant :

Algorithme 2 : Algorithme *CheminAugmentant*

Input : $G = (V, E)$ un graphe, M un couplage
Output : P un chemin augmentant

```
1  début
2      explorer le graphe en profondeur (DFS) et en même temps
3      effectuer les tests suivant
4      si chemin augmentant trouvé alors
5          retourner ce chemin
6      si fleur trouvée alors
7          Construire G' en contractant la fleur
8          P ← CheminAugmentant(G',M')
9          si P = ∅ alors
10             retourner ∅
11         sinon
12             Augmenter P dans G
13 fin
```

Un algorithme de parcours en profondeur d'un graphe est décrit dans ce qui suit :

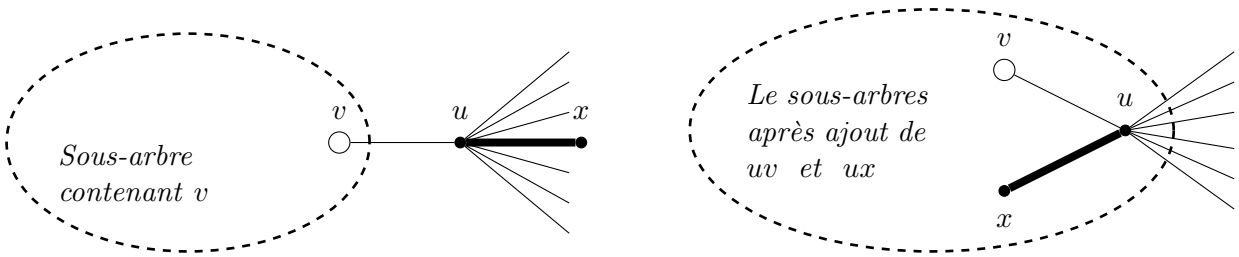
Algorithme 3 : Algorithme *DFS*

Input : $G = (V, E)$ un graphe, un sommet s
Output :

```
1  début
2      Marquer s
3      pour tout voisin v de s faire
4          si v est non marqué alors
5              DFS(G,v)
6  fin
```

L'algorithme *CheminAugmentant* explore le graphe en profondeur en marquant les sommets comme indiqué dans l'algorithme *DFS*. Sauf que la procédure de marquage est très particulière.

Les sommets reçoivent des étiquettes P (pair) ou I (impair). Au début tous les sommets libres sont pairs. Notez qu'il ne peut pas y avoir deux sommets pairs adjacents dans l'étape initiale car sinon le couplage M ne serait pas maximal. Ainsi au début on a une forêt de sommets pairs. Ensuite, on cherche une arête ayant comme extrémités un sommet v étiqueté P et un sommet u non-étiqueté et non-libre (i.e. il existe un sommet x tel que ux appartient à M). Si uv existe, on ajoute les arêtes uv et ux dans l'arbre (la composante connexe) contenant v et on étiquette u avec I et x avec P.



À la fin de la procédure, on a marqué tous les sommets. Que se passe-t-il s'il existe une arête dans G avec deux sommets étiquetés P ? Plusieurs cas de figure :

- Si les deux sommets appartiennent aux arbres distinctes de la forêt alors il existe un chemin augmentant entre les racines des deux arbres.
- Si les deux sommets appartiennent au même arbre, alors il existe une fleur. Prouvez le !

Les explications de la procédure de marquage et comment l'algorithme DFS est utilisé dans l'algorithme *CheminAugmentant* ne sont pas détaillées car c'est principalement là la difficulté technique !

Ready, steady, go !

Instruction 1. *Choisissez et implémentez une modélisation orientée objet de la structure de données Graphes.*

Instruction 2. *Implémentez l'algorithme MM.*

Instruction 3. *Créez vos propres jeux de données, dans le format qui convient le mieux à votre implémentation. Une méthode pourrait être de générer des graphes aléatoirement (à discuter lors des séances de TP avec le responsable du groupe). Les instances seront stockées dans des fichiers afin de pouvoir être réutilisées.*

3 Et si on dessinait un peu ?

Une interface utilisateur ergonomique est une des options clés de tout programme informatique. L'idée c'est de réaliser une mini-interface graphique (une fenêtre JFrame de la bibliothèque Swing) qui permet de visualiser la solution.

Instruction 4. *Créez une fenêtre qui affiche le graphe avec les arêtes du couplage marquées.*

Pour ceux qui s'ennuient

Instruction 5. *Modifiez votre interface graphique afin de permettre à l'utilisateur de visualiser le déroulement de l'algorithme et de modifier la solution courante ou finale. Il faudrait penser à une version adaptative de l'algorithme.*

Instruction 6. *Adaptez la méthode d'affichage de votre graphe pour qu'il soit plus lisible. Vous pouvez par exemple décider de changer la représentation en fonction du nombre de sommets et/ou en fonction de la densité du graphe.*