Projet de programmation orientée objet

PG220

2020-2021

Introduction

Le but de ce projet est d'implémenter un programme en Java permettant de colorer des graphes planaires avec un minimum de couleur. Le programme devra prendre en paramètre un fichier au format JSON décrivant un ensemble de graphes, et devra renvoyer la coloration de ces même graphes sous le même format.

Pour faciliter le travail des groupes, qui seront des trinômes, le projet est découpé en quatre étapes de difficultés croissante. Chaque étape fera l'objet d'une démonstration et d'une rapide revue de code durant le début de chaque séance de suivi de projet (sauf la première, dédiée à la clarification du sujet et au lancement du projet).

Quelques définitions

Un graphe G est une pair (V, E), où V est un ensemble de sommets, et E est un ensemble d'arête, une arête étant simplement une paire de sommets. Si une arête relie deux sommets, on dit que ces sommets sont adjacents. On peut voir les graphes comme un ensemble de point(les sommets) reliés entre eux par des traits (les arêtes).

Colorier les sommets d'un graphe, c'est assigner à chaque sommet une couleur (un entier), tel que deux sommets adjacents n'ont pas la même couleur. Le but est souvent de colorier un graphe avec le minimum de couleur possible. Nous ne nous interesserons ici qu'à un type de graphe particulier: les graphes palanires: *i.e.* les graphes que l'on peut dessiner dans le plan sanq su'il n'y ait de croisement d'arête.

Le fameux théorème des quatre couleurs nous indique que les sommets d'un graphe planaire sont coloriable avec au plus 4 couleurs. Ce nombre de couleurs est néanmoins difficile à obtenir algorithmiquement, et nous allons donc nous donner une couleur supplémentaire.

Etape 1 : Une coloration avec autant de couleurs que de sommets

L'objectif de cette première étape est de développer une application prenant en argument un fichier au format JSON décrivant un ensemble de graphes (décrit comme la liste de leurs arêtes) et de retourner un fichier *resultat.json* au format JSON décrivant ces graphes avec les informations listées en Figure 1.

Figure 1: Format des informations du fichier resultat.json

Afin de pouvoir extraire facilement les informations depuis le fichier qui vous est donné, il vous sera nécessaire de "parser" ce dernier afin de pouvoir aisément le parcourir. Pour ce faire, il existe plusieurs bibliothèques, l'une des plus populaires est org.json, vous pouvez cependant utiliser celle que vous préférez. Vous pouvezr etrouver sa documentation sur le site suivant : `https://stleary.github.io/JSON-java`.

Pour cette première étape, vous travaillerez sur le fichier *etape_1.json*. Le champs label décrit en Figure 1 peut être interprété comme la couleur que l'on va donner au sommet dans la coloration. Le but de cette première étape est donc que tous les sommets, et toutes les arêtes aient des labels différents dans le fichiers de résultat.

Attention : certaines structures dans le fichier etape_1.json ne correspondent pas à des graphes simples (arêtes multiples, arête qui bouclent sur un seul sommet, arêtes qui relient plus de deux sommets, arêtes qui relient un sommet qui n'existe pas etc...). Votre programme devra indiquer en plus dans le fichier *resultat.json* que ces strutures ne sont pas des graphes en utilisant le format illustré en Figure 2.

Figure 2: Format pour indiquer qu'une structure n'est pas un graphe

Etape 2 : Une coloration avec au plus $\Delta + 1$ couleurs

Le but de cette étape est de donner une coloration avec au plus $\Delta + 1$ couleurs où Δ est le nombre maximum de voisins que peut avoir un sommet dans notre graphe. Le principe est simple, il consiste à colorier les sommets du graphe un par un:

- On prend un sommet u non coloré du graphe.
- On regarde les couleurs déjà utilisées par ses voisins: ces couleurs deviennent interdites pour u.
- On colore u avec la plus petite couleur disponible.

Une description de la précédente méthode est illustrée sur la Figure 3. Au final, il est clair que cette méthode n'utilise globalement que $\Delta + 1$ couleurs.

Figure 3: Coloration glouttone

Etape 3 : Une coloration d'arêtes avec au plus $2 \times \Delta - 1$ couleurs

Cette fois le client veut aussi pouvoir colorer les arêtes de son graphe de telle façon à ce que dexu arêtes qui partagent un même sommet n'ont pas la même couleur. Par une méthode similaire à la précédente coloration de sommet, il est possible de colorer l'ensemble des arêtes du graphes avec au plus $2 \times \Delta - 1$ couleurs.

Le but de cette partie est d'implémenter la méthode qui permet d'obtenir une telle coloration.

Etape 4 : Une coloration avec au plus 6 couleurs

Une propriété des graphes planaires est qu'ils ont toujours au moins un sommet qui possède au plus 5 voisins. Or si on a une graphe planaire G qui possède une sommet avec au plus 5 voisins, lorsqu'on suprimme u de G, on obtient aussi un graphe planaire G', et donc celui-ci possède aussi un sommet avec au plus 5 voisins. Ce processus peut être appliqué jusqu'à ce que notre graphe n'ai plus aucun sommet. On a donc une méthode simple pour n'utiliser au plus que 6 couleurs pour notre graphe:

- On "épluche" le graphes en enlevant un à un les sommets avec au plus 5 voisins: ça nous donne une liste de sommet $(u_1, \cdots, u_n)$.
- On utilise l'ordre inverse $(u_n, \cdots, u_1)$ pour colorer u_i avec la plus petite couleur qui n'est pas déjà utilisé par ses voisins.
- Comme u_i a au plus 5 voisins déjà colorés, seules 6 couleurs au maximum seront necéssaire pour colorer u_i et tous ses voisins.

Etape 5 : Une coloration avec au plus 5 couleurs

Il est temps maintenant de s'attaquer à la couleur 6! Le principe clef de l'algorithme est la notion de chaine de Kempe introduite en 1865 par Alfred Kempe dans sa tentative infructueuse de prouver le théorème des 4 couleurs. Lorsque l'on a un graphe déjà (partiellement) coloré, on peut modifier par une opération élémentaire cette coloration pour en obtenir une autre, avec potentiellement moins de couleurs.

Si on considère un graphe G déjà coloré, un sommet u de ce graphe qui est coloré 1 et qui possède un voisin v coloré 2, on peut changer la couleur de u en 2 et celle de v en 1. Un problème peut alors arriver si v possèdait d'autres voisins coloriés 1, or on peut changer la couleur de ces sommets en 2, et répeter l'opération jusqu'à ce que cette "chaîne" de 1 et de 2 s'arrête: cette chaîne est appelée $(1,2)$-chaîne de Kempe contenant u, et l'opération qui consiste à échanger les deux couleurs sur cette chaîne est appelée un $(1,2)$-chagement de Kempe sur u.

Comment peut-on éliminer une couleur avec cette méthode? Si on considère un graphe planaire G auquel j'applique l'algorithme pour le colorer avec 6 couleurs, et qu'on arrive sur un sommet u qui possède déjà 5 voisins u_1, u_2, u_3, u_4, u_5 colorés par des couleurs différentes $1, 2, 3, 4, 5$, je devrais colorer u avec la couleur 6. Or avec des chaînes de Kempe il est possible de recolorer mon graphe pour que deux voisins de u aient finalement la même couleur.

La planarité du graphe nous assure qu'il existe une paire de couleur i, j telle que la (i,j)-chaîne de Kempe contenant u_i ne contient pas u_j, il suffit donc de tester pour toutes les paires de couleurs (i,j) : si la (i,j)-chaîne de Kempe contenant u_i ne contient pas u_j, il suffit d'échanger les couleurs sur cette chaîne pour qu'il y ait une couleur inférieure à 5 disponible pour u.

Une description visuelle de cette méthode est présentée en Figure 4.

Figure 4: Le principe des chaînes de Kempe

Évaluation

Ce projet est à réaliser par groupe de 2 (3 si besoin) étudiants. Vous serez évalués sur :

- La conception des classes et l'architecture de votre programme.
- La modularité et l'extensibilité du programme (minimiser l'effort pour inclure de nouveaux type de coloration (coloration des arêtes, coloration avec d'autres contraintes etc), d'autres classes de graphes).
- L'utilisation des concepts fondamentaux de la programmation orientée objet.
- La robustesse de la façon dont vous avez implémenté les fonctionnalités.
- La gestion des erreurs.

Fonctionnalités supplémentaires

Des points bonus pourront être obtenus pour chaque implémentation d'une fonctionnalité proposée dans cette liste:

- Détection de planarité : si un graphe n'est pas 5-colorable avec notre méthode, c'est qu'il n'est pas planaire.
- Coloration des arêtes du graphe avec au plus $2 \times \Delta - 1$ couleurs.
- Application du principe des chaînes de Kempe sur les colorations d'arêtes. (TODO: exemple d'application)

Contraintes sur le code

Vous devez absolument respecter ces contraintes. Une pénalité de 2 points sera appliquées sur chaque projet qui ne respecte pas:

- Votre projet contient plusieurs paquetages.
- Les noms des paquetages sont intégralement en minuscules.
- Les noms des classes et interfaces commencent par une majuscule.
- Les noms des méthodes, des attributs et des variables commencent par une minuscule.
- Utilisez le CamelCase pour les noms des classes, interfaces, méthodes et attributs.
- Choisissez une langue pour tout votre projet (de préférence l'anglais), et gardez ce même standard pour tout le projet.
- Les attributs ont une visibilité *private* ou *protected*. Des accesseurs permettent si nécessaire d'accéder en lecture et/ou écriture aux attributs.
- La visibilité des méthodes dépend de leur utilisation.
- Les constructeurs sont définis uniquement s'ils sont nécessaires.
- Le code des méthodes compliquées doit être commenté.

Livrable

Il sera demandé aux étudiants de fournir une archive contenant le code source du programme. À la racine de l'archive, il doit y avoir un fichier README spécifiant les étapes qui ont été complétées et celles qui ne l'ont pas été, ainsi que les fonctionnalités supplémentaires qui ont été implémentées. L'archive doit aussi contenir un fichier java archive coloration.jar qui permet de lancer le jeu à l'aide de la commande java -jar coloration.jar. Le schéma de nommage de l'archive doit être le suivant (pénalitéde 3 points en cas de non respect) : pg220-2020-LOGINS.zip, où LOGINS est la liste des logins des membres du groupe, classée par ordre alphabétique et séparés par le symbole "_". L'archive est à déposer, avant le X XXXX 2020 à 23h59, via la plateforme `http://moodle.ipb.fr`.