SAÉ S2.02 Exploration algorithmique d'un problème

Wittmann Gregory

Rouillon Tom

Welty Alexandre

Yifru Abenezer

UNIVERSITÉ DE LORRAINE | IUT Saint-Dié-des-Vosges

Sommaire

1. Introduction au projet

1.a) Présentation du projet

Tout d'abord, notre projet consiste à colorier une carte découpée en régions, avec des couleurs différentes entre régions adjacentes (c'est-à-dire avec une frontière commune), chaque couleur ayant un prix par unité de surface variable. Connaissant la superficie de chaque secteur, nous devons donner la solution la moins coûteuse à un instant donné.

1.b) Ressources utilisées

Suite aux différents cours de la ressource DOO/Développement orienté objets ainsi que la ressource Graphe nous avons acquis les compétences requises afin de mener à bien le projet et le sujet demandé ci-dessus.

En effet celui-ci nous demande qu'à partir d'un graphe, donc la structure est composée de sommets, certains reliés par une arête, que nous fassions en sorte que la coloration des sommets soit ordonnée pour que deux sommets adjacents aient une couleur différente.

Ce phénomène de coloration est expliqué et mis en application grâce à l'algorithme de Welsh-Powell qui permet d'obtenir un graphe avec une coloration plus ou moins optimale.

Nous devons ensuite donner à partir d'un prix de la couleur la solution la moins coûteuse pour colorier le graphe.

2. Développement de l'application

2.a) Répartition

Tout d'abord nous avons réparti notre code dans 4 fichiers qui sont dans le package graphe.
Le premier étant celui de la modélisation du graphe et de l'application de l'algorithme de Welsh-Powell.
Le deuxième celui de la gestion des couleurs et de leur prix.
Le troisième celui de la gestion des sommets et de ses voisins.
Puis le dernier qui va afficher le résultat obtenu à partir de l'association des 2 fichiers ci-dessus. C'est ce fichier qui comporte le main, et c'est donc celui-ci qui va nous permettre d'exécuter notre application.

2.b) Le fichier de la gestion des sommets

La classe sommet a comme paramètres un nom, une superficie, une *ArrayList* de ses voisins et une couleur associée qui ne sera pas initialisé par défaut afin que l'on puisse lui associer une couleur par la suite. Le constructeur va donc les contenir avec comme exceptions qu'un nom ne peut pas être vide et qu'une superficie ne peut pas être négative.

Cette classe va nous permettre grâce aux différentes méthodes d'obtenir la superficie d'un sommet, son nom ainsi que ses voisins.

En fonction du nombre de voisins que le sommet aura, un degré lui sera associé afin de mener à bien le bon déroulement de l'algorithme de Welsh-Powell dans le fichier *Graphe*.

De plus entre chaque sommet, et en fonction des voisins, une arête sera créée afin de définir la relation entre voisins.

Cette méthode ne peut être utilisée que dans un cadre de graphe non-orienté puisque si on ajoute un sommet s1 dans la liste des voisins de s2, alors s1 sera aussi dans la liste des voisins de s2, ce que nous ne voulons pas dans un cadre de graphe orienté.

Cette classe implémente l'interface comparable pour pouvoir être trier selon le degré du sommet.

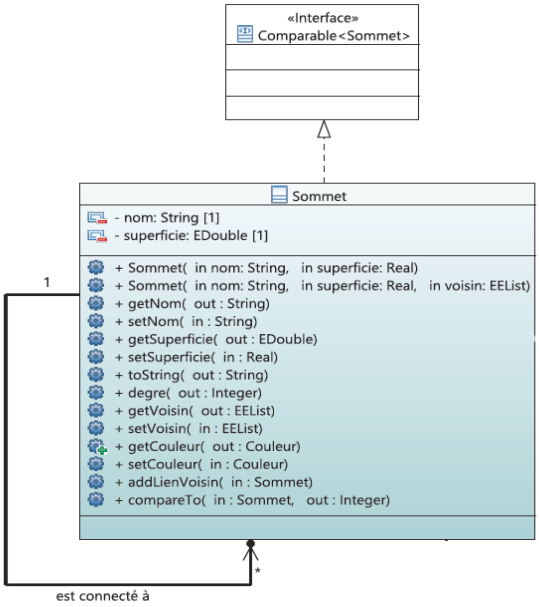

Figure 1: Diagramme UML - Sommet

2.c) Le fichier de la gestion des couleurs

Pour la création de ce fichier, nous sommes partis sur la même base de classe, c'est-à-dire l'initialisation des paramètres, ici le nom et le prix de la couleur. Puis le constructeur qui prend ces 2 paramètres avec la gestion des exceptions de si le nom de la couleur est vide ou pas et que le prix ne peut être négatif.

Toujours une méthode *toString* qui va permettre l'affichage de la couleur avec son prix. Puis une méthode *equals* qui va vérifier si la couleur est bien égale à la couleur passée en paramètre.

Enfin, cette classe implémente aussi l'interface *Comparable ;* la méthode compareTo va nous permettre de comparer 2 couleurs. Cette méthode sera utilisée lors du tri de la liste des couleurs par prix décroissant dans la classe *graphe.*

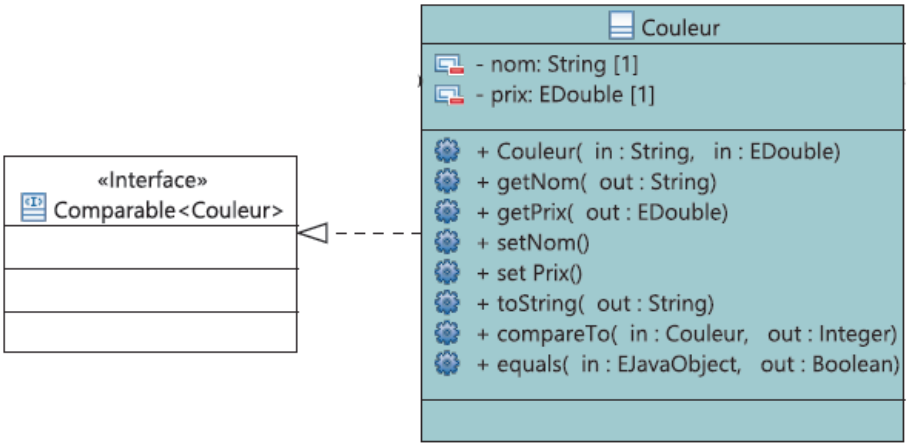

Figure 2: Diagramme UML - Couleur

2.d) Le fichier de la gestion du graphe

Afin de résoudre le problème, il est important de savoir que nous sommes dans le cas des graphes non-orientés.

Pour commencer nous avons créé une classe *Graphe* qui va contenir comme paramètre la liste de tous les sommets du graphe ainsi que la liste des couleurs qui sont sous la forme d'une *ArrayList.* Le constructeur du graphe va contenir ces 2 paramètres ce qui va nous permettre de créer un graphe de sommets.

Puis pour que l'application soit plus complète il faudrait vérifier que le graphe est vide ou non pour éviter les mauvaises surprises, ce qui a été intégré à celle-ci.

Afin de voir la liste des sommets et des couleurs lors de l'affichage nous avons fait des méthodes qui vont renvoyer la liste des sommets ainsi que de celle des couleurs.

De plus dans un graphe, chaque sommet a des voisins (autres sommets reliés), c'est pourquoi une méthode *toString* qui va dans un premier temps convertir les sommets en chaîne de caractères, puis qui va renvoyer la liste des sommets avec leurs voisins respectifs. Tout cela afin d'avoir un affichage correct.

Suite à cela, nous avons incorporé l'algorithme de Welsh-Powell qui trie les sommets selon leur degré décroissant puis attribue une couleur différente par sommets adjacents. Cet algorithme ne prend pas en compte la superficie des sommets.

Par conséquence, la suite du sujet nous demande de donner à partir d'un prix de la couleur, la solution la moins coûteuse pour colorier le graphe, c'est pourquoi, une fois qu'on a les couples (couleur attribuée, listes des sommets), on a pu faire le calcul de la superficie totale par couleur attribué. Pour le tri des couples (couleur, liste de sommets) selon une superficie totale décroissante, on a utilisé un *Comparator* anonyme.

Par la suite, il suffit juste d'affecter la couleur la moins coûteuse à la superficie totale la plus grande.

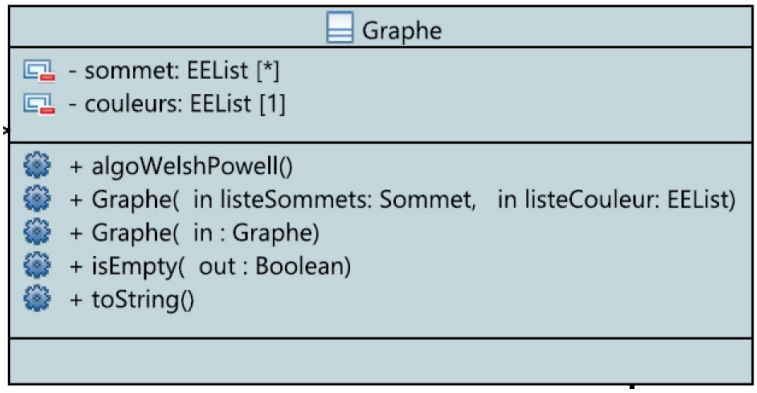

Figure 3: Diagramme UML - Couleur

2.e) Le fichier de la gestion du résultat

Le résultat est affiché grâce au fichier *Main* qui va nous permettre d'obtenir toutes les informations nécessaires pour respecter le sujet.

Donc celui-ci va premièrement utiliser un fichier externe au format *csv* (Comma Separated Values) qui va nous permettre de gagner du temps lors de l'initialisation et la réalisation du graphe.
Les informations dans ces fichiers sont du type :

Numéro du sommet, Nom du sommet, Superficie, Tous ses voisins

Pour lire dans ces fichiers, nous avons utilisé un BufferedReader, qui va nous permettre de récupérer toutes les lignes du fichier grâce à la méthode *readLine*. De plus, le fait que toutes les valeurs soient séparées par une virgule nous permet facilement de les récupérer grâce à la méthode *split.*

La seule contrainte qu'il y a est qu'il faut mettre les numéros des sommets dans l'ordre croissant, pour pouvoir créer correctement tous les liens entre les voisins.
Ce fichier va donc se coller et bien s'associer aux différents fichiers ci-dessus afin que tout reste coordonné ensemble.

Cette classe *Main* va donc dans un second temps afficher tous les sommets enregistrés, puis va créer 4 couleurs avec leur nom et leur prix. Cependant, le choix de 4 couleurs n'est pas hasardeux. En effet, il repose sur le théorème des 4 couleurs, qui nous indique qu'il est possible de colorier n'importe quelle carte découpée en régions connexes avec seulement 4 couleurs de telle sorte que deux régions adjacentes (c'est-à-dire avec une frontière en commun) ne soient pas de même couleurs.
Comme c'est exactement le cadre de notre sujet, donc nous avons décidé de n'utiliser que 4 couleurs, puisque de toute façon les autres n'auraient jamais servi.

Ensuite, la classe va afficher ces couleurs, puis va afficher le graphe non trié, c'est-à-dire qu'il va afficher le nom de chaque sommet et le nom de chacun de ses voisins, et après elle va afficher le graphe trié par ordre décroissant du degré des sommets, ce qui veut dire que le sommet avec le plus de voisins sera le premier afficher et le sommet avec le moins de sommet sera affiché en dernier.

Enfin, la méthode *algoWelshPowell* est appelée pour appliquer cet algorithme sur le graphe fraîchement créé.

Cette méthode va aussi afficher certaines choses, notamment le résultat final, c'est-à-dire qu'elle va donner, pour chaque sommet, son nom, le nom de la couleur associée, la superficie du sommet puis le prix de la couleur.

Puis après chaque changement de couleur, il y a un récapitulatif donnant le nom de la couleur, son prix puis la superficie totale de tous les sommets coloriés avec cette couleur.

3. Conclusion

Pour conclure, dans ce projet, nous avons vu les différentes phases du développement d'une application.

En effet nous avons vu comment créer le fichier qui nous a permis de gérer le graphe et comment le lier aux autres fichiers de gestion des sommets et de la couleur, pour pouvoir créer un tout qui fonctionne.

Puis à la fin, nous avons vu comment utiliser ces fichiers afin d'afficher le résultat voulu et toutes les informations avec.

Ce projet nous a permis d'acquérir de l'expérience concernant le développement dans le langage de programmation java, et d'appliquer ce que nous avons vu en cours de développement sur un exemple concret.

Si nous devions recommencer nous pensons qu'il faudrait réfléchir davantage au début du projet, notamment sur les structures de données utilisées pour choisir celles qui sont le plus efficace étant donné toutes les contraintes du projet. Cela nous éviterait de changer le code en plein milieu du développement parce que les structures utilisées ne nous permettent pas de remplir toutes les tâches demandées.