# Projet Algorithme

## Coloration de graphes triangulés

Ce rapport a pour but de présenter le projet réalisé en Algorithme, en détaillant notamment nos choix de structures de données, et la façon dont nous avons implémentés les algorithmes. Nous justifierons nos choix, en détaillant avec un diagramme des classes la conception de ce projet, qui a pour but de minimiser la coloration d'un graphe.

**Plan :**

#### I°) Besoin et cahier des charges

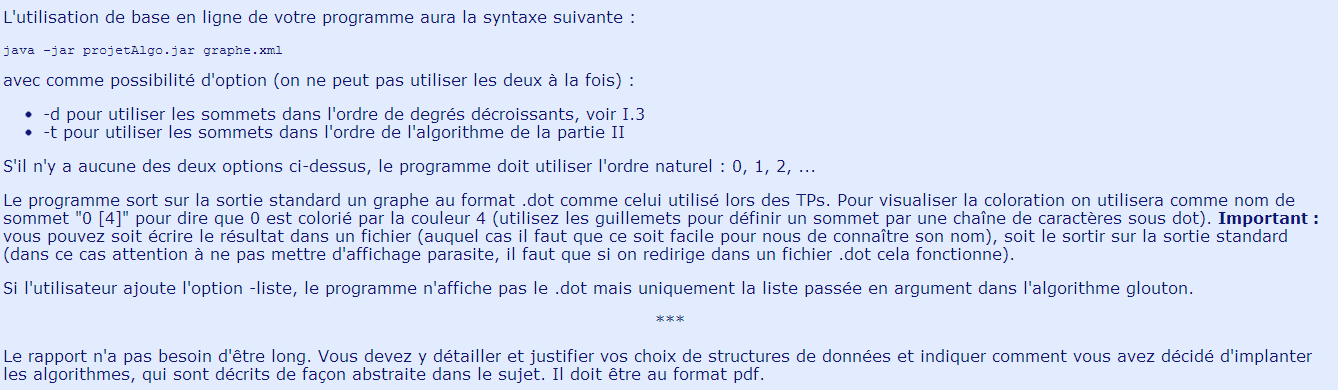
#### II°) Choix des structures de données

#### III°) Implémentations des algorithmes

### I°) Besoin et cahier des charges

Ce projet d'algorithme a pour but de nous faire implémenter un algorithme glouton de coloration de graphes, prenant en paramètre un ordre sur les sommets, et permettant de minimiser le nombre de couleurs d'un graphe. Une fois implémentée, nous utiliserons deux algorithmes pour calculer l'ordre sur les sommets, qui sont Welsh et Powell ainsi qu'un algorithme optimisant l'ordre pour les graphes triangulés.

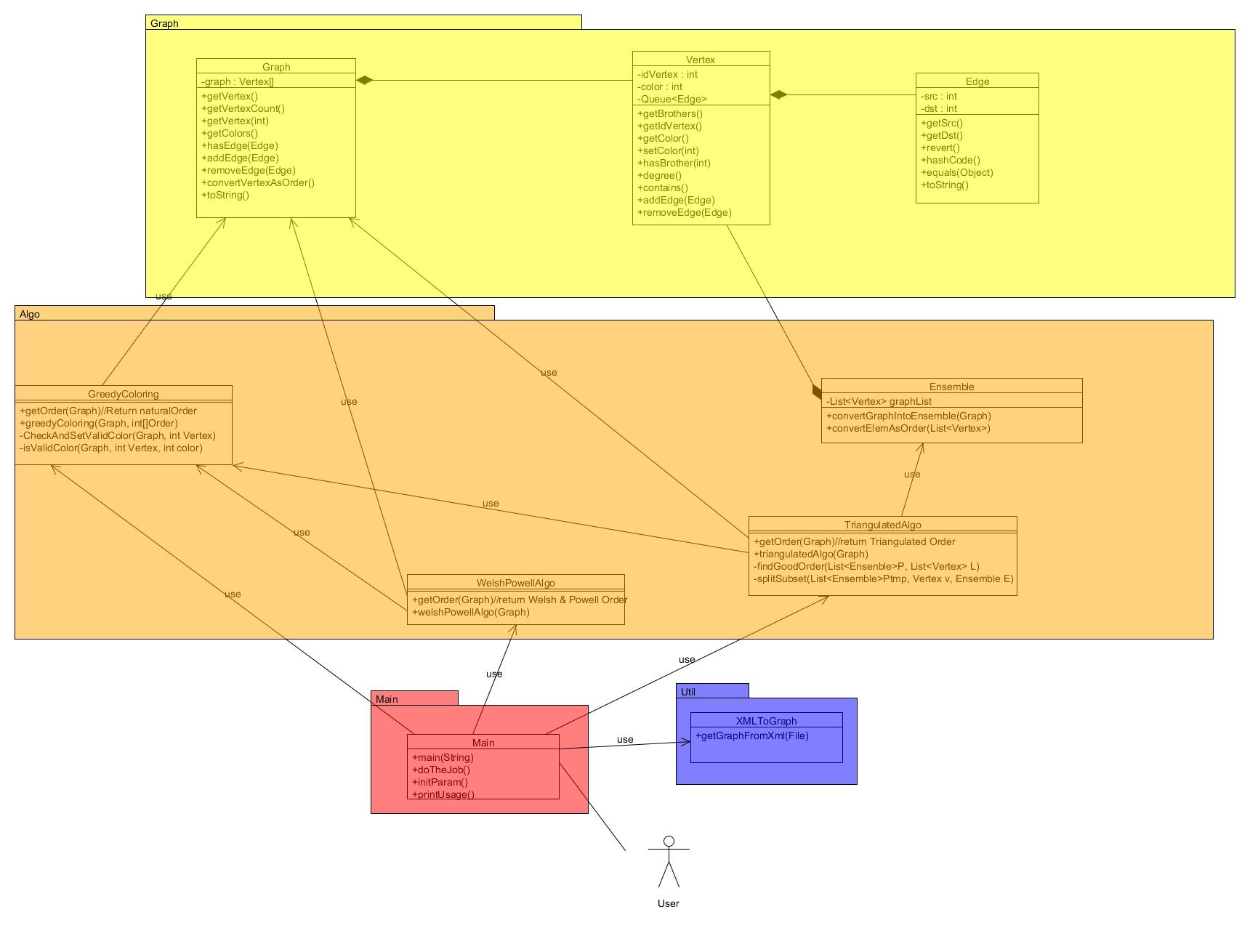
Tout au long du projet, nous ne considérerons que des graphes non-orientés, connexes, et dont les arêtes ne sont pas values. De plus, aucun sommet d'aura de boucle vers lui même.

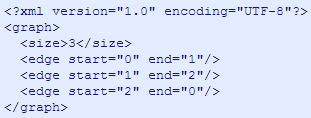
Voici le cahier des charges correspondant : 

### II°) Choix des structures de données

Pour le choix de notre structure de données, nous avons était au plus proche du concept de graphe : Ainsi, un graphe est représenté par un tableau de Vertex, qui lui même possède une couleur(nulle par défaut), un ID, ainsi qu'une Queue d'Edge, pour ainsi ne pas avoir de doublons d'Edge (chemins). Un chemin est donc représenté par deux entiers, qui correspondent aux deux ID des vertes composant le chemin.

Les algorithmes implémentées sont tous statiques et ne dépendent d'aucune instanciation d'objet. Ainsi, on a pas besoin d'instancier de classe pour appliquer les différents algorithmes aux graphes. On obtient, au final, le diagramme de classes suivants :



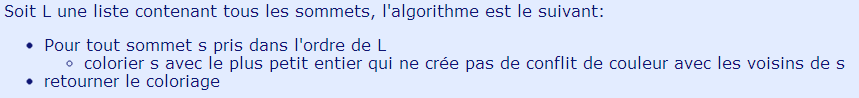
 Ainsi, grâce à cette structure de données, nous séparons la partie algorithmique de la partie structure de données. Ce programme permet ainsi à un utilisateur de réaliser plusieurs opérations :

* Par défaut, il faut Obligatoirement préciser un nom de fichier xml, prenant la syntaxe suivante :
* L'algorithme utilisé sera toujours l'algorithme glouton de coloration de graphes, mais avec des ordres sur les sommets différents.
* Par défaut, on utilise comme ordre l'ordre naturel ( 0 1 2 3 ...), et on affiche le graphe coloré sous format dot sur la sortie standard.
* Si l'argument précisé est **-d** ou **--decrease**, on utilise comme ordre sur les sommets ceux-ci classés par degrés décroissants, ou le degré d'un sommet est le nombre d'arêtes qui lui sont reliés.
* Si l'argument précisé est **-t** ou **--triangulated**, on utilise comme ordre sur les sommets l'optimisation pour les graphes triangulés.
* Si l'argument précisé est **-l** ou **--list**, on affiche uniquement la liste d'ordre sur les sommets, et on n'affiche pas le graphe.

### II°) Implémentations des algorithmes

Pour l'implémentation de chaque algorithme, il a été choisi de réaliser une décomposition par sous méthodes, pour ainsi clarifier l'algorithme, et le réaliser par couches successives.

#### A ) Algorithme Glouton de Coloration

Le choix a été fait de séparer l'algorithme en 3 méthodes, dont chaque méthode réalise une tâche. Ainsi, on obtient, comme implémentation de l'algorithme suivant : 

**greedyColoring (Graph G, OrderVertex order) :**

**Pour chaque sommet s de G :**

**checkAndSetValidColor(G, order[s])**

**CheckAndSetValidColor(Graph G, Vertex s) :**

**Pour i de 0 à |G| -1**

**Si isValidColor(G, s, i)**

**break;**

**isValidColor(Graph G, Vertex s, int color) :**

**Pour chaque successeur t de s**

**Si la couleur de t est égale à color**

**retourner faux**

**retourner vrai**

#### B ) Algorithme de Welsh et Powell

Pour implémenter l'algorithme de Welsh et Powell, il a suffit d'utiliser la méthode Sort de la classe Array, pour trier un graphe en fonction de l'ordre sur les degrés de chaque sommet, et d'en sortir un ordre sur les sommets, grâce à l'ID de chaque sommet. On obtient l'algorithme suivante :

**welshPowellAlgo(Graph G) :**

**retourner greedyColoring(G, getOrder(G))**

**getOrder(G) :**

**Graph Gtmp = copie (G)**

**Trie Gtmp en fonction de l'ordre sur les degrés de chaque sommet.**

**orderVertex = Convertie Sommets en ordre(Gtmp)**

**retourner orderVertex**

#### C ) Coloration de graphes triangulés

Un graphe triangulé, c'est un graphe non orienté tel que tout circuit de taille 4 ou plus contient une corde, c'est à dire une arrêt qui n'est pas dans le circuit et relie deux sommets du circuit, et permet donc de former un circuit plus court. (circuits minimaux de taille au plus 3).

Pour implémenter le calcul d'ordre qui optimise le nombre de couleurs sur un graphe triangulé, il a été choisi de séparer par couches l'algorithme. De plus, on a représenté un Ensemble par une InnerClass, représenté par une liste de sommets. On obtient ainsi l'implémentation suivante :

**triangulatedAlgo(Graph G) :**

**retourner greedyColoring(G, getOrder(G))**

**getOrder(G) :**

**P = list d'ensemble initialisé.**

**L = list de sommets initialisé.**

**Ajout à P l'ensemble (G)**

**Tant que n'est pas vide**

**findGoodOrder(P, L)**

**retourner convertie Ensemble ordre(L)**

**findGoodOrder(list<Ensemble>P,list<Vertex>L)**

**Ptmp = list d'ensemble initialisé.**

**Vertex = P.get(0).remove(0)**

**Si la taille de P.get(0) est 0**

**p.remove(0)**

**Pour tout ensemble E de P**

**splitSubset(Ptmp, v, E)**

**P = copie(Ptmp)**

**Ptmp.clear()**

**splitSubset(list<Ensemble>Ptmp, Vertex v, Ensemble E)**

**Ensemble V, W = init**

**Pour chaque sommet s de E**

**Si s est voisin de v**

**V.add(s)**

**Sinon**

**W.add(s)**

**Si V et W ne sont pas vides**

**Ptmp.add(V)**

**Ptmp.add(W)**

**Sinon**

**Ptmp.add(E)**

L'ensemble des algorithmes ont été implémentés, et testées avec les deux fichiers proposées avec le sujet. De plus, des tests unitaires JUnit ont été réalisés pour valider l'implémentation de notre structure de données, ainsi que le résultat des algorithmes.

Ce projet nous aura permis d'implémenter des algorithmes pour colorer un graphe en minimisant le nombre de couleurs. Ainsi, nous avons mis en place un algorithme glouton de coloration de graphes, que nous avons optimisés grâce à un ordre sur les grandeurs. En jouant avec cet ordre, nous avons implémentées deux algorithmes (Welsh & Powell ainsi qu'un algorithme d'optimisation pour les graphes triangulés) pour minimiser ce nombre de couleurs.