Rapport de projet pour le cours Algorithmique et structures de données 2

Florian Lefebvre

May 29, 2022

Contents

1 Cahier des charges du projet

Nous allons ici présenter la problématique du projet et donner les objectifs que nous nous sommes fixés.

1.1 Le sujet du projet : La coloration de graphe

Le but de ce projet est de pouvoir colorier chaque sommet d'un graphe de telle manière que deux sommets voisins n'aient pas la même couleur. Dans ce contexte, on cherche souvent à utiliser le nombre minimal de couleurs. C'est ce que l'on appelle nombre chromatique.

1.2 Ma démarche globale

Pour offrir une ergonomie plus agréable à l'utilisateur, j'ai voulu dès le départ réaliser un éditeur graphique pour pouvoir dessiner le graphe. L'environnement que j'ai choisi d'utiliser est l'outil Qt Creator. Cet outil est un IDE très riche pour développer des interfaces homme-machine en C++. J'ai dû m'initier à

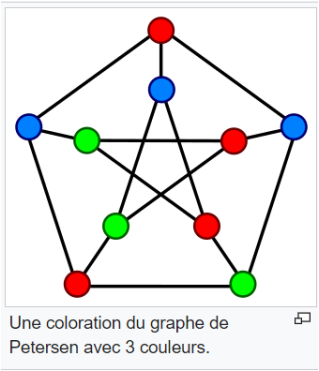

Figure 1: Exemple de graphe colorié extrait de wikipédia

cet environnement pour pouvoir développer un éditeur de graphe avec des fonctionnalités minimales. Une fois que cet éditeur a été fonctionnel, j'ai étudié la problématique de coloration de graphe. L'un des algorithmes a retenu mon attention et j'ai décidé de l'implémenter en complément de l'éditeur de graphe.

2 L'éditeur de graphe

2.1 Les étapes de création d'un graphe et de sa coloration

L'application se présente au départ sous la forme d'une fenêtre très simple avec une zone centrale permettant de créer les états du graphe de façon numériquement croissante. En cliquant sur le bouton de gauche de la souris, un état se crée avec une première valeur commençant à la valeur 1. Puis se succèdent d'autres états possible comme le montre la figure ci-dessous. Ensuite, il est possible de relier

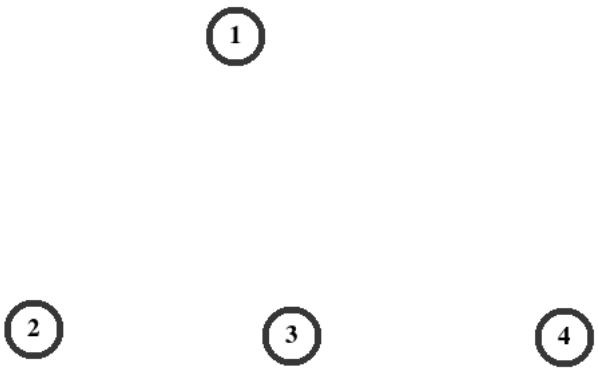

Figure 2: Création des états d'un graphe

les états en cliquant sur deux états différents. Le fait de cliquer deux fois sur le même état ne crée pas une transition sur l'état. Il est bien entendu possible de créer les états et de les relier dans n'importe quel sens. L'éditeur ne permet pas dans cette version de corriger ou d'effacer des éléments du graphe. Voici

ci-dessous l'exemple d'un graphe avec des liens entre les états. Ensuite avec

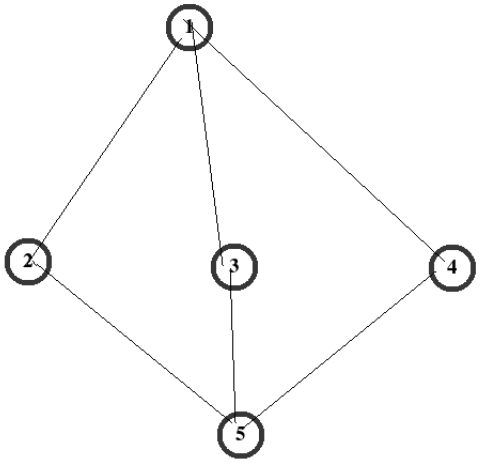

Figure 3: Exemple de graphe avec transitions

le bouton droit de la souris, on peut demander la coloration du graphe que je viens de créer. L'outil va choisir un nombre minimal de couleur en s'assurant bien entendu que deux états successifs aient la même couleur. Voici ci-dessous le résultat engendré. Il est possible d'étendre ce graphe et de demander une

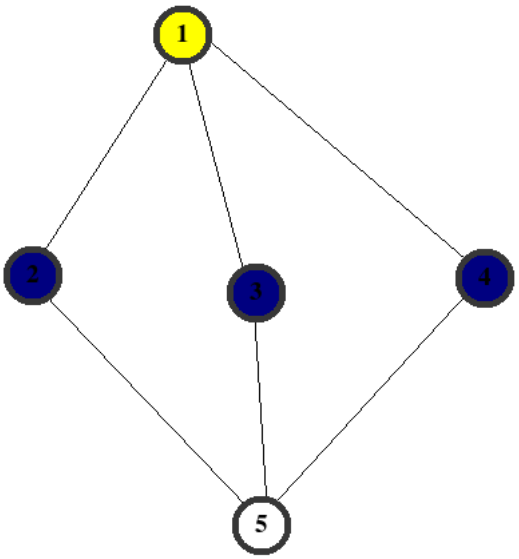

Figure 4: Colorisation d'un graphe

nouvelle coloration de celui-ci comme le montre la figure 5 :

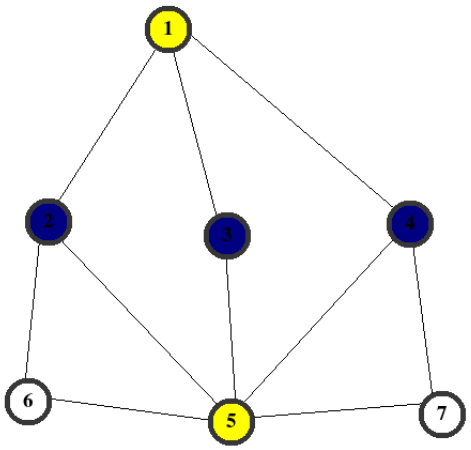

Figure 5: Extension d'un graphe

2.2 Les structures de données pour représenter un graphe

Au départ, pour stocker les états du graphe, j'utilise deux dictionnaires. Le premier se nommant "sommet" permet d'associer un numéro d'état à une position en x et y à l'écran. Un deuxième dictionnaire se nommant "etat_couleur" permet d'associer à un état, une couleur de départ qui est le blanc. A chaque fois que l'utilisateur clique à l'écran, cela crée un nouvel état. Si l'utilisateur clique sur un état, cela mémorise un début d'association avec un autre état. Lorsqu'il cliquera sur un autre état, cela reliera ces deux états. Pour sauvegarder cette relation entre deux états, j'utilise un autre dictionnaire qui s'appelle "relation" mais cette fois-ci avec la possibilité d'associer à une clé plusieurs valeurs. De même, je stocke dans le dictionnaire l'association dans les deux sens. Voici ci-dessous un exemple de graphe et le contenu du dictionnaire stockant les relations.

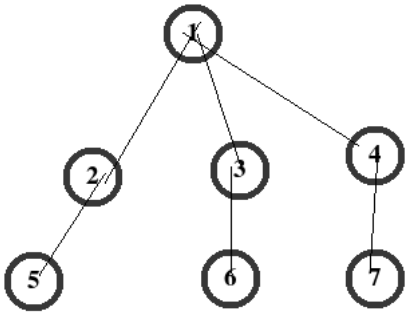

Figure 6: Exemple d'un graphe et sa représentation interne

- sommet 1 relié à sommet 4
- sommet 1 relié à sommet 2

- sommet 1 relié à sommet 3
- sommet 2 relié à sommet 1
- sommet 2 relié à sommet 5
- sommet 3 relié à sommet 1
- sommet 3 relié à sommet 6
- sommet 4 relié à sommet 1
- sommet 4 relié à sommet 7
- sommet 5 relié à sommet 2
- sommet 6 relié à sommet 3
- sommet 7 relié à sommet 4

La dernière étape va être de stocker dans le dictionnaire "état_couleur" les codes représentant des couleurs à partir du résultat de la coloration dont voici le contenu :

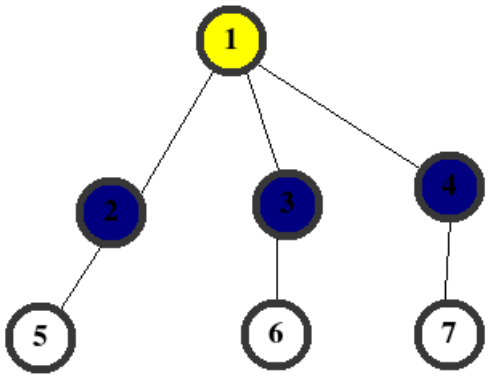

Figure 7: Exemple du dictionnaire d'un graphe colorié

- etat numero 1 couleur : 1
- etat numero 2 couleur : 2
- etat numero 3 couleur : 2
- etat numero 4 couleur : 2
- etat numero 5 couleur : 0
- etat numero 6 couleur : 0
- etat numero 7 couleur : 0

2.3 Exemple d'un graphe complexe avec coloration

La figue 8 ci-dessous montre un exemple de graphe avec de nombreux états et une optimisation du nombre de couleurs:

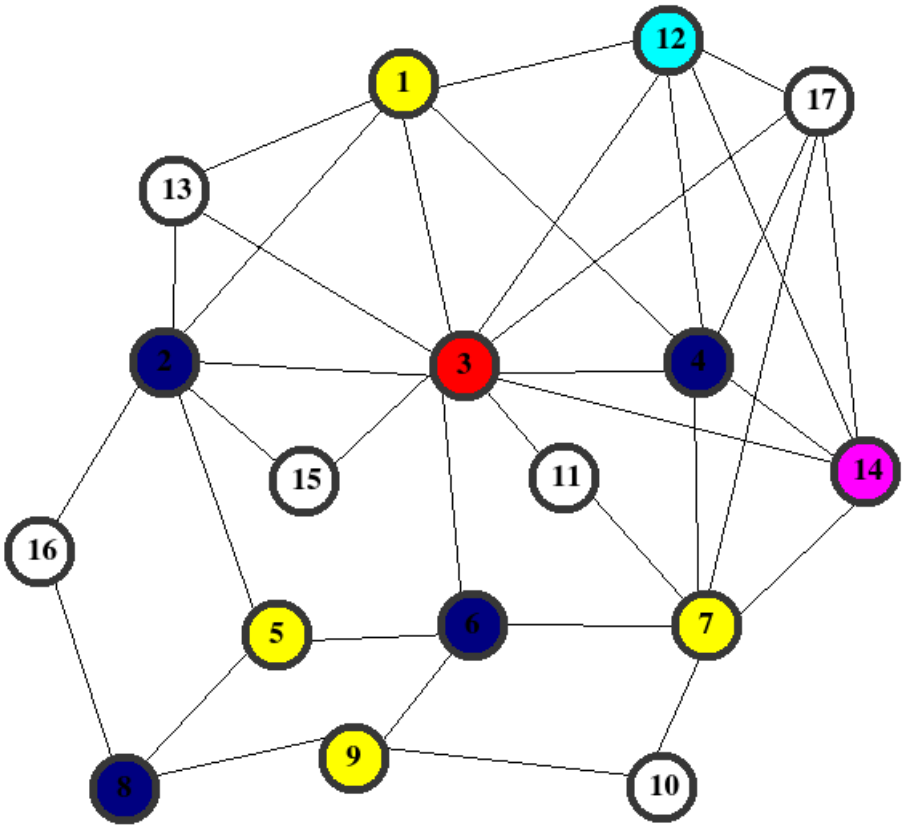

Figure 8: Exemple d'un graphe complexe et sa coloration

3 L'algorithme de coloration d'un graphe

L'objectif de cet algorithme est de colorier un graphe de telle manière que deux sommets n'aient pas la même couleur. Les couleurs elles-mêmes ne sont pas importantes. Seul compte le nombre de couleurs différentes utilisés. Une coloration propre d'un graphe utilisant le nombre minimal de couleurs est dit optimale. Au début, aucun sommet n'est colorié. La règle de coloration du sommet courant est la suivante : le sommet courant doit prendre la plus petite couleur parmi celles non utilisées par ses voisins déjà coloriés.

3.1 Présentation de l'algorithme de liste

Si nous prenons un exemple, la table des couleurs choisies est la suivante :

- blanc(0)

- jaune(1)
- bleu(2)
- rouge(3)
- cyan(4)
- magenta(5)
- vert(6)
- gris(7)

Soit le graphe suivant figure 9 :

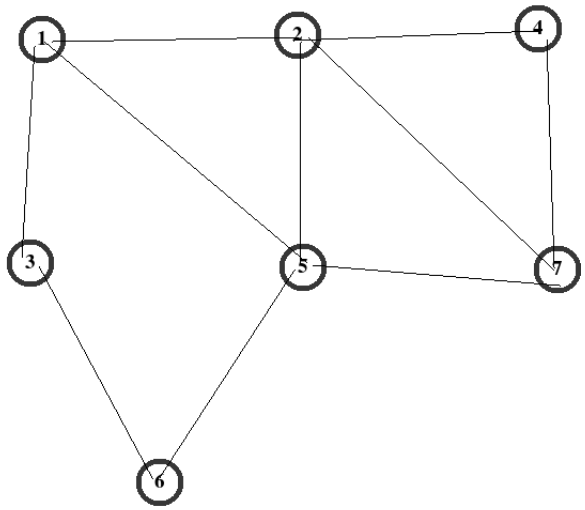

Figure 9: Exemple d'un graphe avant sa coloration

Voici le résultat de sa coloration avec entre parenthèse la couleur :

- Le sommet 1 est relié au sommet 2(0), 3(0), 5(0). Le sommet 1 va prendre la couleur 1 (jaune).
- Le sommet 2 est relié au sommet 4(0), 5(0), 7(0), 1(1). Le sommet 2 va prendre la couleur 2 (bleu).
- Le sommet 3 est relié au sommet 6(0), 1(1). Le sommet 3 va prendre la couleur 2 (bleu).
- Le sommet 4 est relié au sommet 2(2), 7(0). Le sommet 4 va prendre la couleur 1(jaune).
- Le sommet 5 est relié au sommet 1(1), 2(2), 7(0), 6(0). Le sommet 5 va prendre la couleur 3(rouge).
- Le sommet 6 est relié au sommet 3(2), 5(3). Le sommet 6 va prendre la couleur 0(blanc).

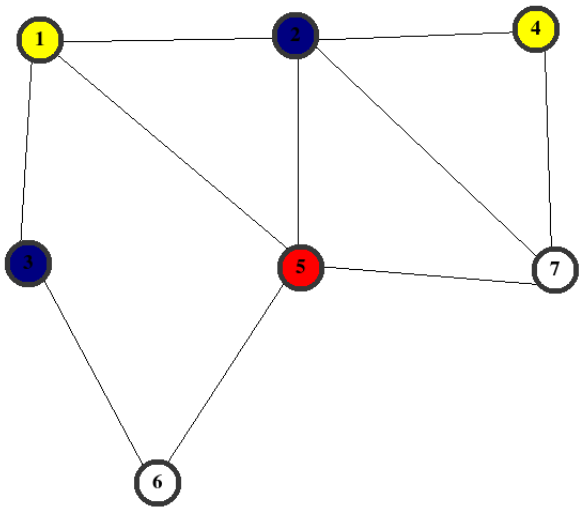

Figure 10: Coloration du graphe précédent

- Le sommet 7 est relié au sommet 2(2), 4(1), 5(3). Le sommet va prendre la couleur 0(blanc).

Le résultat final correspond bien au graphe de la figure 10.

3.2 Analyse de l'algorithme de liste

Cet algorithme construit toujours une coloration propre car à chaque étape, le sommet s traité prend nécessairement une couleur différente de ses voisins déjà coloriés. Avec la règle de coloration, chaque voisin de s non encore traité prendra plus tard lorsqu'il sera traité, une couleur différente de celle de s. Ainsi, chaque sommet s a une couleur différente de ses voisins. La coloration est donc propre. Si un sommet s a n voisins dans le graphe, le numéro de couleur que lui affecte l'algorithme sera toujours au plus n + 1. En effet, au moment où le sommet s est traité, au plus n de ses voisins ont déjà une couleur, en l'occurence, ceux qui ont déjà été traité avant s. Deux cas de figure peuvent se présenter :

- Toutes les couleurs entre 1 et n sont déjà prises par les voisins de s. La règle de coloration donne la couleur n + 1 à s.
- Une des couleurs entre 1 et n n'est prise par aucun voisin de s (parce que tous les n voisins n'ont pas encore été traités ou parce que deux voisins de s ont la même couleur). Dans ce cas, le sommet s reçoit la plus petite couleur non encore attribuée.

3.3 Un exemple d'application: gestion d'un planning

Pour illustrer l'intérêt de l'utilisation de la coloration d'un graphe, je vais prendre la situation de trois étudiants qui doivent suivre des cours parmi 5 thèmes. Chaque cours aura la même durée, c'est-à-dire une heure. Le problème est de

savoir si ces cours doivent être sur des créneaux parallèles ou différents. Plusieurs configurations sont possibles.

Première configuration : tous les étudiants doivent suivre tous les cours.

Il est logique que tous les cours seront sur des créneaux différents pour que tous les étudiants suivent les cours. Voici ci-dessous figure 11 le graphe représentant cette situation. Les états représentent une arête entre deux cours qui indique donc que ces deux cours ne peuvent pas voir lieu en même temps. C'est une contrainte qui doit être respectée.

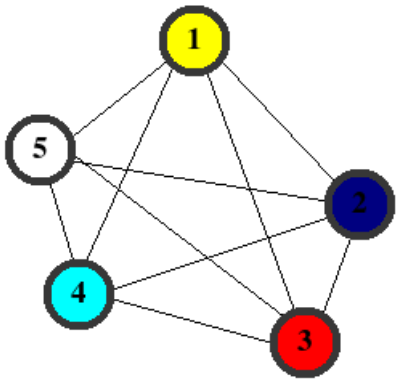

Figure 11: Gestion d'un planning: configuration 1

Deuxième configuration :

- L'étudiant E1 doit suivre les cours C1, C2 et C5
- L'étudiant E2 doit suivre les cours C2 et C3
- L'étudiant E3 doit suivre les cours C3, C4, et C5

Le résultat du graphe colorié nous montre qu'il faudrait trois créneaux horaires pour gérer cette configuration. Le premier créneau horaire pourrait gérer les cours 1 et 3. Le second créneau horaire pourrait gérer les cours 2 et 4. Enfin le dernier créneau serait le 5.

Troisième configuration :

- L'étudiant E1 doit suivre les cours C1, C2
- L'étudiant E2 doit suivre les cours C3 et C4
- L'étudiant E3 doit suivre les cours C5

Voici le résultat du graphe colorié figure 13 correspondant :

Dans cette configuration, deux créneaux sont seulement nécessaires, un créneaux pour le cours 1 et 3 et un deuxième créneau pour les cours 2, 4, 5.

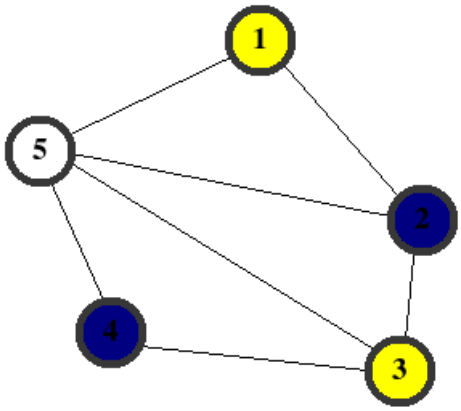

Figure 12: Gestion d'un planning: configuration 2

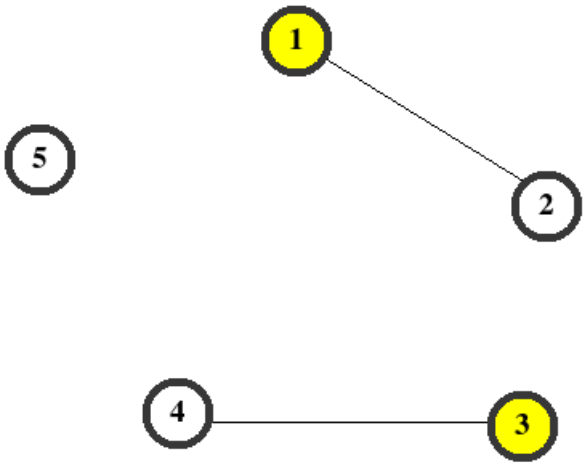

Figure 13: Gestion d'un planning: configuration 3

4 Conclusions

Ce projet m'a permis de découvrir une notion intéressante sur les graphes : la coloration. J'ai pu m'initier également à la bibliothèque Qt afin de pouvoir dessiner les graphes. La difficulté principale de ce projet était de trouver un algorithme me permettant de colorier un graphe. L'algorithme de liste a été simple à comprendre mais sa mise en oeuvre est plus délicate. Pour conclure, j'ai réalisé un prototype avec un objectif direct de coloration de graphes. Il pourrait être étendu par des fonctionnalités comme la modification, l'implémentation d'autres algorithmes. J'ai beaucoup apprécié le thème de ce projet.