Master 1 – Modélisation, Graphes et Algorithmes
Mini-projet : 5-coloration des graphes planaires

1 Description générale

Les graphes planaires peuvent se représenter dans le plan sans qu'il n'y ait deux arêtes se croisent. Une k-coloration d'un graphe $G = (V, E)$ est une fonction $c : V \to \{1, 2, \ldots, k\}$ telle que $c(u) \neq c(v)$ pour toute arête $\{u, v\} \in E$. Autrement dit, il est possible d'utiliser k couleurs pour colorer les sommets du graphe sans que deux sommets voisins n'aient la même couleur.

Le fameux théorème des 4-couleurs [1] assure qu'un graphe planaire peut toujours être coloré avec seulement 4 couleurs. Dans ce projet nous serons plus modestes ; vous implémenterez un algorithme qui colore un graphe planaire avec au plus 5 couleurs.

2 Quelques briques

Commençons par fixer quelques notations techniques [2]. Soit $G = (V, E)$ un graphe. Nous notons $n = |V|$ et $m = |E|$. Le graphe $H = (V', E')$ est un sous-graphe de G si $V' \subseteq V$ et $E' \subseteq E$. On dit que $H = (V', E')$ est un sous-graphe induit de G si $E' = \{\{u, v\} : u \in V' \wedge v \in V' \wedge \{u, v\} \in E\}$. Etant donné un sommet $u \in V$ et un ensemble de sommets $X \subseteq V$, on note $d_{G\setminus X}(u) = |\{v \in (V \setminus X) : \{u, v\} \in E\}|$; ainsi $d_{G\setminus X}(u)$ représente le nombre de voisins de u dans $G \setminus X$. Si $L = [x_1, x_2, \ldots, x_i, \ldots, x_n]$ est une liste, on note par $L[i : n]$ la liste restreinte aux $n - i + 1$ éléments $[x_i, \ldots, x_n]$.

Les quelques briques suivantes sont utiles pour garantir qu'un graphe planaire peut effectivement être coloré avec au plus 5 couleurs. Elles sont aussi utiles à l'algorithme qui effectue cette 5-coloration.

Brique 1. Si G est un graphe planaire, alors il possède au moins un sommet de degré au plus égal à 5.

Brique 2. Si G est un graphe planaire, alors tout sous-graphe de G est également planaire.

Considérons un graphe planaire G et une coloration de G utilisant 5 couleurs. Nous notons $\{\alpha, \beta, \gamma, \delta, \epsilon\}$ ces 5 couleurs.

Brique 3. Soit $G(\alpha, \beta)$ le sous-graphe de G induit par les sommets de couleur α et β. Soit $H(\alpha, \beta)$ une composande connexe de $G(\alpha, \beta)$. En échangeant les couleurs α et β dans $H(\alpha, \beta)$, on obtient encore une coloration valide de G.

1. Voir Wikipédia pour des éléments historiques sur ce magnifique théorème.
2. En première lecture, vous pouvez sauter ce premier paragraphe et vous y référer ensuite.

Brique 4. Soit x un sommet de G ayant au plus 4 voisins. Si on dispose d'une coloration de $G - x$, alors il est possible de colorer x par l'une des couleurs non utilisées par ses voisins.

Brique 5. Soit x un sommet de G ayant exactement 5 voisins notés a, b, c, d, e. Soit $G' = G - x$. Si on dispose d'une coloration de G' telle qu'au plus 4 couleurs différentes soient utilisées pour colorer a, b, c, d, e, alors il est possible de colorer x par l'une des couleurs non utilisées par ses voisins.

Brique 6. Soit x un sommet de G ayant exactement 5 voisins. Soit $G' = G - x$. Si on dispose d'une coloration de G' telle que 5 couleurs différentes soient utilisées pour colorer les voisins de x, alors il est possible de colorer x de la façon suivante :

- On oriente le voisinage de x dans le sens trigonométrique. Les sommets rencontrés sont a, b, c, d, e, dans cet ordre.
- Notons respectivement $\alpha, \beta, \gamma, \delta, \epsilon$ les couleurs de a, b, c, d, e.
- Comme G est planaire, on ne peut pas avoir a et c dans une même composante connexe de $G'(\alpha, \gamma)$ et, *en même temps*, b et d dans une même composante connexe de $G'(\beta, \delta)$.
- On applique la brique 3 sur l'une de ces deux composantes connexes.
- On libère ainsi une couleur utilisée dans le voisinage de x pour colorer x.

3 L'algorithme

3.1 Version récursive

Grâce aux briques précédentes, nous pouvons écrire l'algorithme récursif `Coloring-rec` qui colorie un graphe planaire en utilisant au plus 5 couleurs.

Algorithme `Coloring-rec`(G)

si *G n'est pas vide* **alors**
- Soit x un sommet de degré ≤ 5
- `Coloring-rec`$(G - x)$
- Appliquer la brique 4, 5 ou 6 pour colorer x (et éventuellement recolorer des sommets de G)

3.2 Version itérative

Pour rendre cet algorithme itératif, nous pouvons étudier la *dégénérescence* du graphe :

Définition. Un graphe G est k-dégénéré si tout sous-graphe de G admet un sommet de degré au plus k.

Les briques 1 et 2 assurent que si $G = (V, E)$ est un graphe planaire, alors il est 5-dégénéré. Cela signifie qu'il est possible de trouver un sommet $v_1 \in V$ tel que $d_G(v_1) \leq 5$, puis un sommet $v_2 \in V \setminus \{v_1\}$ tel que $d_{G \setminus \{v_1\}}(v_2) \leq 5$, puis un sommet $v_3 \in V \setminus \{v_1, v_2\}$ tel que $d_{G \setminus \{v_1, v_2\}}(v_3) \leq 5$, puis ...

Brique 7. Si $G = (V, E)$ est un graphe planaire, alors on peut ordonner les sommets V dans une liste $L = [v_1, v_2, \ldots, v_n]$ telle que pour tout i $(1 \leq i \leq n)$, $d_{G \setminus (\bigcup_{1 \leq j < i} \{v_j\})}(v_i) \leq 5$.

On obtient l'algorithme itératif `Coloring-it` :

Algorithme `Coloring-it`(G)

Construire une liste L qui respecte la propriété de la brique 7
pour *i de n à 1 (par pas de -1)* **faire**
  Soit $\tilde{N}(v_i)$ les (au plus 5) voisins de v_i qui appartiennent à $L[i : n]$
  Appliquer la brique 4, 5 ou 6 pour colorer x (et éventuellement recolorer des sommets de $L[i : n]$)

4 Contraintes d'implémentation

Vous devez implémenter une fonction `Coloring` pour la 5-coloration d'un graphe planaire. Vous pouvez vous inspirer de l'algorithme récursif ou de sa version itérative. Les graphes utilisés dans votre implémentation devront être représentés par des listes d'adjacences (et non des matrices d'adjacences).

Vous développerez également une fonction `CheckColoring` qui testera que la coloration produite par votre implémentation est effectivement valide.

Pour l'implémentation, vous utiliserez le langage de votre choix parmi : Python, Caml, Java, C, C++. L'utilisation de tout autre langage nécessitera l'accord de votre chargé de TD.

4.1 Entrée

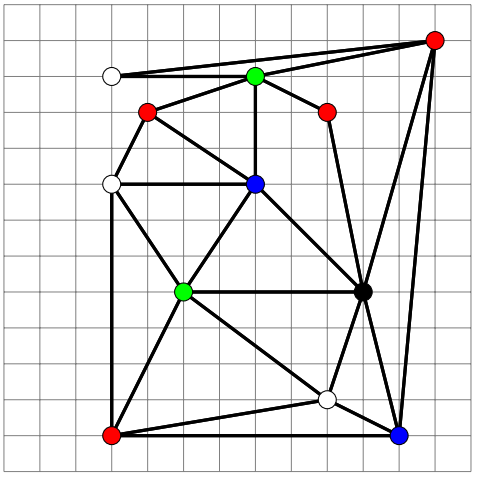
Afin de pouvoir tester votre algorithme, votre programme devra pouvoir lire un graphe sauvegardé dans un fichier `.graphe`. Un tel fichier contient le nombre de sommets du graphe, puis les listes d'adjacence de chaque sommet. Pour chaque sommet, la liste de ses voisins est triée dans le sens trigonométrique. Il est **impératif de respecter** ce format afin de tester ultérieurement votre algorithme.
Voici le format de l'entrée pour un graphe planaire à 12 sommets :

```
12
0: [1, 5, 7, 11]
1: [8, 4, 5, 0]
2: [4, 9]
3: [7, 4, 9, 6, 11]
4: [5, 1, 8, 2, 3, 7]
5: [0, 1, 4, 7]
6: [11, 3, 9]
7: [0, 5, 4, 3, 11]
8: [10, 9, 4, 1]
9: [6, 3, 2, 8, 10]
10: [9, 8]
11: [0, 7, 3, 6]
```

En bonus, le fichier `.coords` contient les coordonnées de chacun des sommets, au cas où vous souhaiteriez développer une sortie graphique pour dessiner le graphe. Pour un graphe à n sommets, chaque coordonnée est représentée par un couple d'entiers (x, y) tel que $1 \leq$

$x, y \leq n$. Toutes les coordonnées sont distinctes et il n'y a donc pas de point confondu. Voici le fichier pour l'exemple précédent :

```
12
0: (3, 1)
1: (11, 1)
2: (9, 10)
3: (7, 8)
4: (10, 5)
5: (9, 2)
6: (4, 10)
7: (5, 5)
8: (12, 12)
9: (7, 11)
10: (3, 11)
11: (3, 8)
```



4.2 Sortie

Votre programme enregistrera un fichier `.colors` contenant une coloration du graphe de l'entrée. À chaque sommet sera associé une couleur de l'ensemble (`blue`, `red`, `green`, `white`, `black`).

Voici un exemple de fichier en sortie :

```
12
0: red
1: blue
2: red
3: blue
4: black
5: white
6: red
7: green
8: red
9: green
10: white
11: white
```

5 Rapport de projet

Vous devez rendre un rapport d'au **maximum 5 pages**. Ce document devra :

— indiquer vos noms et prénoms ;
— proposer une analyse de la complexité en temps et en espace de votre implémentation ;
— indiquer comment exécuter ou compiler votre code (en quelques lignes ; on ne demande pas comment installer Java ou Python) ;
— donner le temps d'exécution de votre programme sur les graphes mis à disposition sur Celene ;
— expliquer vos choix d'implémentation, difficultés rencontrées, solutions proposées, ... ;
— si nécessaire, vous présenterez également le diagramme de classes et un pseudo-code simplifié des méthodes plus complexes.

6 Modalités

Vous réaliserez ce projet par groupes de 2 étudiants. Une soutenance de projet pourra éventuellement avoir lieu après le rendu du projet.

Vous devez rendre une archive (au format *.zip* ou similaire) contenant :
— le code source de votre programme ;
— votre rapport au format PDF.

L'archive doit porter le nom des auteurs (exemple : `LiedloffPerez.zip`). Vous devrez déposer cette archive sur la plateforme Celene.

`http://celene.univ-orleans.fr`

Le projet est à rendre avant le 13 décembre 2020, minuit.