Filière MP : ENS de Cachan, Lyon, Rennes et Paris

Page de garde du rapport de TIPE - Session 2014

NOM : CABANNES Prénoms : Vivien, Albert, Léon

Lycée : Stanislas

Classe : MP*

Ville : PARIS

Concours auxquels vous êtes admissible dans la banque MP inter-ENS :

(les indiquer par une croix inscrite dans les cases ci-dessous)

ENS Cachan	MP - option MP	X	MP - option MPI	
	Informatique			
ENS Lyon	MP - option MP	X	MP - option MPI	
	Informatique - option M		Informatique - option P	
ENS Rennes	MP - option MP	X	MP - option MPI	
	Informatique			
ENS Paris	MP - option MP	X	MP - option MPI	
	Informatique			

Matière dominante du TIPE (la sélectionner d'une croix inscrite dans l'une des cases ci-dessous) :

Informatique		Mathématiques	X	Physique	

Titre du TIPE : Coloration de graphes

Nombre de pages (à indiquer dans les cases ci-dessous) :

Texte	T	3	Illustrations	I	7	Bibliographie	B	1

Dossier Annexes

Résumé (ou descriptif succinct) du TIPE (6 lignes, maximum) :

Initiation à la théorie des graphes à travers le coloriage
1ère partie théorique : premières manipulations et résultats : Euler - Kuratowski - 5 couleurs
2ème partie : programme - algorithmes de coloration
3ème partie : une application puissante - démonstration de Brewer à partir du lemme de Sperner

A Paris, le 12.06.2014

Signature du (de la) candidat(e)

Vivien Cabannes

Signature du professeur responsable de la classe préparatoire dans la discipline

Cachet de l'établissement

STANISLAS
enseignement privé sous contrat d'association
CPGE
22, Rue Notre-Dame des Champs
75279 PARIS CEDEX 06

Vivien Cabannes
MP

Coloration de graphes

Rapport de TIPE

Sommaire

Fiche synoptique

Réalisé avec Enzo Fiorentino

Motivation : La théorie des graphes m'était complètement inconnue. Elle offrait le support idéal pour ce Tipe :

- sujet *vaste* recoupant le thème *Transfert, Flux, Echanges*
- *liberté* des recherches vis-à-vis du cours
- *curiosité* de découvrir de nouveaux outils mathématiques

Plus tard, se concentrer sur la notion de coloriage fournissait un cadre plus restreint permettant de lier théorie, algorithmes et applications.

Objectif : S'initier à la théorie des graphes avec le concept de coloration

Démarches : Le dossier s'articule autour de trois axes et trois solutions retenues :

- apprendre à raisonner sur des graphes : Pour cela, notre attention s'est focalisée sur le théorème de Kuratowski, la formule d'Euler et un aperçu du théorème quatre couleurs à travers celui des cinq couleurs.

- faire un programme informatique : Connaissant quelques bornes théoriques d'optimalité de coloriage, nous voulions ensuite implémenter un algorithme de coloration avec retour graphique.

- trouver une application plus théorique : La théorie des graphes est un outil mathématique à part entière, c'est ce que nous souhaitons illustrer à travers une démonstration du théorème de Brouwer depuis le lemme de Sperner.

Bilan : Le sujet était très vaste, ce dossier ne peut qu'en constituer une introduction.

Les recherches théoriques donnèrent un petit aperçu sur la recherche fondamentale, qui offre beaucoup de joies mais demande beaucoup de ténacité. Les théorèmes simples et intuitifs aux démonstrations demandant trop de temps pour être bien comprises donnèrent naissance à plusieurs pistes abandonnées.

La partie algorithmique m'a permis de développer un mode de penser propre à l'informatique, notamment la programmation orientée objet. Elle demanda beaucoup de temps pour un résultat final relativement sobre, mais le travail et l'apprentissage furent vraiment agréables.

Le théorème de Brouwer permet de saisir la puissance de la théorie des graphes en dehors d'elle-même, et, par ailleurs, le lien très fort entre le continu et le discret. Ce dernier développement se révéla particulièrement bien adapté au niveau des classes de spéciales.

Définitions : Par souci de simplicité nous avons rejeté toutes les définitions en annexe, nous encourageons le lecteur à s'y reporter fréquemment.

Dossier

La théorie des graphes naît d'un papier de Léonard Euler (*Les Sept Ponts de Königsberg*, **1736**). Elle constitue un outil puissant pour résoudre un grand nombre de problèmes en se ramenant à l'étude d'un ensemble de sommets et d'arrêtes, et trouve de nombreuses applications dans les domaines liés aux notions de réseaux, flux et communications.

Ce dossier s'intéresse à la coloration de graphes, plus particulièrement au théorème des quatre couleurs et à quelques applications de la notion de coloriage : construction d'emplois du temps, approche combinatoire du théorème de Brouwer.

I. Vers un théorème de coloration

A. Premiers cheminements sur les graphes

Théorème de Kuratowski (*1930*)

Un graphe est planaire si et seulement s'il ne contient pas de subdivision de $K_{3,3}$ ou de K_5.

La démonstration de ce théorème se fait en deux temps, on montre d'abord que $K_{3,3}$ et K_5 ne sont pas planaires, puis qu'il ne peut pas exister de graphe non planaire qui ne contienne pas de subdivisions de l'un ou de l'autre.

Formule d'Euler (*1752*)

Toute représentation d'un graphe planaire connexe définit r régions selon :

$$n - a + r = 2$$

On en déduit, en notant v_i le nombre de sommets ayant i sommets adjacents, tout graphe planaire G vérifie l'inégalité, qui est une égalité seulement si G est triangulé :

$$\sum (6 - i) \cdot v_i \geq 12$$

B. Théorème des cinq couleurs

Historiquement, c'est en montrant l'erreur d'une preuve de Kempe (*1879*), que Heawood en sauva une démonstration du théorème des cinq couleurs (*1890*). Ayant d'abord choisi d'exposer cette démonstration historique, nous avons finalement opté de suivre volontairement le formalisme de l'actuelle démonstration du théorème des quatre couleurs.

Schéma de la preuve

On suppose, par l'absurde, qu'il existe un graphe planaire qui n'est pas 5-coloriable. On considère n le plus petit nombre tel qu'il existe un contre-exemple à n sommets et G un tel graphe.

Dans un premier temps, par des considérations algébriques, on montre que G contient nécessairement un élément appartenant à un ensemble de *configurations inévitables*.

Ensuite, on montre que toutes ces configurations sont *réductibles*, c'est à dire, s'il existe un contre-exemple contenant une de ces configurations, il en existe un ayant moins de sommets.

On conclut qu'il ne peut pas exister de contre-exemple minimal.

II. Applications pratiques du coloriage

A. Algorithmes de coloration

Description du problème

Etant donné un graphe *G*, écrire un algorithme pour colorier *G* à la fois rapide et utilisant peu de couleurs. Devant les lacunes théoriques, les algorithmes feront appel à l'heuristique. Ce problème a plusieurs applications pratiques comme la création d'emplois du temps.

Implémentation sur Python

Coloriage linéaire : On colorie les sommets en procédant selon leur degré.
Coloriage de proche en proche **:** On colorie d'abord le sommet de plus haut degré. Puis étant donné un état semi-colorié de *G*, on colorie le sommet ayant le plus grand nombre de voisins coloriés, en cas d'égalité, on regarde le degré. La complexité est quadratique.

Observations : les deux algorithmes sont rapides mais pas optimaux, comme le prouve le graphe de la Figure 1, le deuxième est plus performant (cf. Figure 2 et Test)

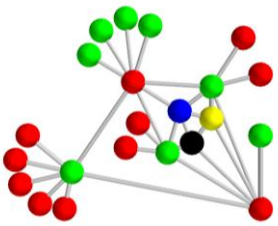

Figure 1 : non optimalité des deux algorithmes, dessin en 3D

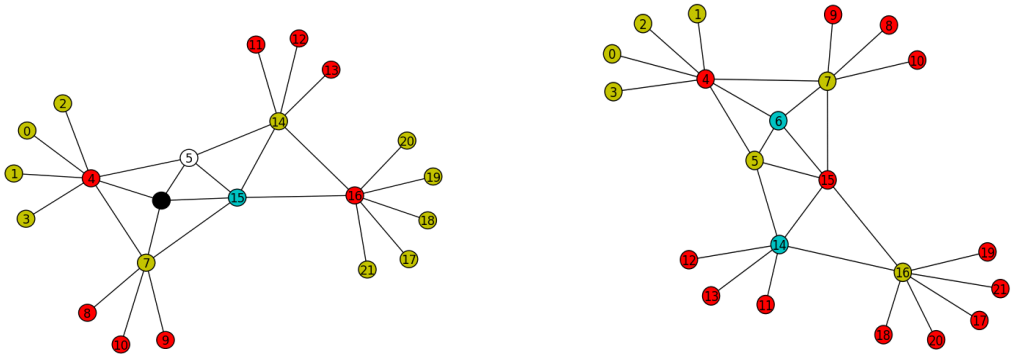

Figure 2 : le premier algorithme renvoie cinq couleurs, le second que trois, dessin en 2D

Tests : Nous avons testé nos deux algorithmes sur les 633 configurations de Robertson, Sanders, Seymour et Thomas (disponibles depuis l'article de G. Gonthier), voici les résultats :

	Coloriages non optimaux	Temps d'exécution (U.A.)
Coloration linéaire	38 ‰	2
Coloration proche en proche	0	3

B. Approche combinatoire du théorème de Brouwer

On cherche à ramener un problème continu de topologie, le théorème de Brouwer, à un problème discret de combinatoire, le lemme de Sperner. Cette démarche est propre à l'esprit des mathématiciens se rattachant à une axiomatique qu'ils maitrisent mieux ; l'illustrent Descartes et la géométrie algébrique, Euler et la topologie algébrique, Poincaré et la topologie combinatoire.

Lemme de Sperner (*1928*)

Toute coloration compatible d'un n-simplexe triangulé contient un nombre impair de cellules conformes. En particulier, elle en contient au moins une.

Schéma de la démonstration : On raisonne par récurrence sur n. Le cas $n = 1$ se résout avec une récurrence quasi-immédiate sur le nombre de sommets.

Hérédité : On considère un coloriage $\{0, 1, \ldots, n\}$ et on regarde les faces coloriées avec $\{0, 1, \ldots, n-1\}$.

Un dénombrement adéquat que l'on peut formaliser avec le graphe dual de la triangulation montre que le nombre de cellules conformes est de même parité que le nombre de faces conformes appartenant à la triangulation de la face coloriée avec $\{0, 1, \ldots, n-1\}$ du n-simplexe origine.

On utilise alors l'hypothèse de récurrence pour conclure que ce nombre est impair

Théorème de Brouwer (*1912*)

Toute application continue d'un compact convexe de $\mathbb{R}^n$ dans lui-même admet un point fixe.

Schéma de la démonstration :

1er cas : Si K est un n-simplexe et $f : K \to K$. En coordonnées barycentriques, on définit les ensembles :

$$\forall\, i \in \{0, \ldots, n\},\ F_i = \{\, \boldsymbol{v} \in K \mid f_i(\boldsymbol{v}) \leq v_i \text{ ET } v_i > 0 \,\}$$

Pour chaque triangulation de K, on peut définir un coloriage $\{0, 1, \ldots, n\}$ compatible et adapté aux F_i. En considérant une suite de triangulations de diamètres tendant vers 0, le lemme de Sperner nous permet d'en déduire une suite de cellules conformes dont le diamètre tend également vers 0 et comme K est compact, il existe une sous-suite convergeant vers un point $\boldsymbol{y}$ de K.

La continuité de f montre alors que $\boldsymbol{y} \in F_0 \cap F_1 \cap \ldots \cap F_n$, ce qui implique $f(\boldsymbol{y}) = \boldsymbol{y}$.

2ème cas : Les compacts convexes ont tous une structure très proche, une éventuelle relation d'homéomorphie permet de conclure, considérant K comme une déformation d'un n-simplexe.

Connaissant mal ces théorèmes, nous avons préféré inscrire le compact K dans un n-simplexe T puis définir π la projection de T sur K. Après avoir montré que π est bien défini et continu, en appliquant le cas précédent à $f \circ \pi$, on obtient un point fixe $\boldsymbol{y}$ pour $f \circ \pi$ mais comme f est à valeur dans K, $\boldsymbol{y} \in K$ donc $\boldsymbol{y}$ est un point fixe de f.

Annexes

Bibliographie

Documents de travail :

Ouvrage de vulgarisation :

Introduction to Graph Theory, Richard J. Trudeau, Dover Publications, **1993**, première édition en 1976.

Four color theorem, Wikipédia, dernière consultation le 18 mai 2014 :
http://en.wikipedia.org/wiki/Four_color_theorem

Source perdue :

Points fixes, ADS extrait du tétraconcours autour des points fixes, dont une partie portait sur le lien Brouwer Sperner, que l'on me donna à faire en SUP.

Pour aller plus loin :

Sujets à l'attention d'élèves étudiants l'informatique fondamentale :

Graphes planaires, Théophile Trunck, **2009**, à propos de la représentation des graphes, caractérisation et programmes de reconnaissance, disponible à partir de :
http://perso.ens-lyon.fr/eric.thierry/Graphes2009/sujets.html

Le théorème des quatre couleurs, Georges Gonthier, **2000**, pour mener tous les calculs montrant la réductibilité des 633 configurations inévitables, disponible à partir de :
http://www.enseignement.polytechnique.fr/informatique/profs/Georges.Gonthier/pi2000/

Images :

Réalisation personnelle sur Python 2.7.5, dessin en 2D et 3D avec les modules respectifs matplotlib et mayavi, travail à l'aide du module networkx :
http://networkx.github.io/index.html

Définitions

Première partie :

Graphe : ensemble de n sommets et a arêtes ; $G = (A, S)$,

$$S = \{ S_i, i \in \{1, ..., n\} \} \text{ ET } A \subset \{ \{ S_i, S_j \} \mid (S_i, S_j) \in S^2 \} \text{ ET } a = Card(A)$$

Adjacence : S_i et S_j sont liés par une arête ; $\{S_i, S_j\} \in A$

Arbre : graphe connexe qui ne contient pas de cycle ; $\forall k \in \mathbb{N}, \forall (U_0, U_1, ..., U_k) \in S^{k+1}$,

$$U_k \neq U_0 \text{ OU } \exists i \in \{0, ..., k\text{-}1\}, (U_i = U_{i+2} \text{ OU } \{U_i, U_{i+1}\} \notin A)$$

k-Coloration : application coloriant deux sommets adjacents différemment, $C \in \{1, ... , k\}^S$;

$$\forall (i, j) \in \{1, ..., n\}^2, \{S_i, S_j\} \in A \Rightarrow C(S_i) \neq C(S_j)$$

Connexité : il existe un chemin passant par tous les sommets ; $\exists (U_0, ..., U_k) \in S^{k+1}$,

$$\forall i \in \{0, ..., k\text{-}1\}, \{U_i, U_{i+1}\} \in A \text{ ET } S = \{U_0, U_1, ..., U_k\}$$

Degré : nombre de sommets adjacents ; $deg(S_i) = Card \{ j \in \{1, ..., n\} \mid \{S_i, S_j\} \in A \}$

$\boldsymbol{K_n}$ **:** graphe à n sommets tous adjacents

$\boldsymbol{K_{p,q}}$ **:** graphe constitué de deux parties P et Q ayant respectivement p et q sommets et tel que l'ensemble des arêtes de $K_{p,q}$ s'écrive $\{ \{i, j\}, (i, j) \in P \times Q \}$

Planéité : Graphe pouvant être représenté sur un plan sans que deux arêtes se coupent.

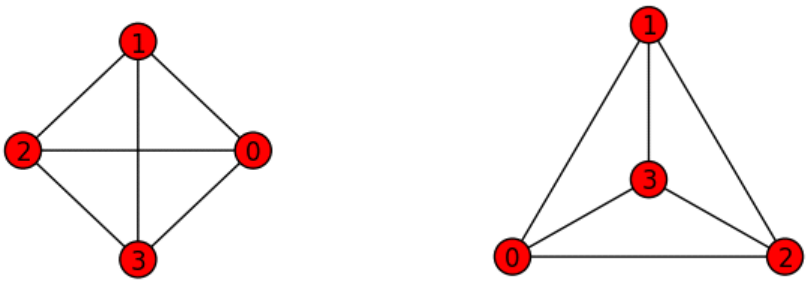

Figure 3 : Deux représentations du même graphe K_4 qui est planaire

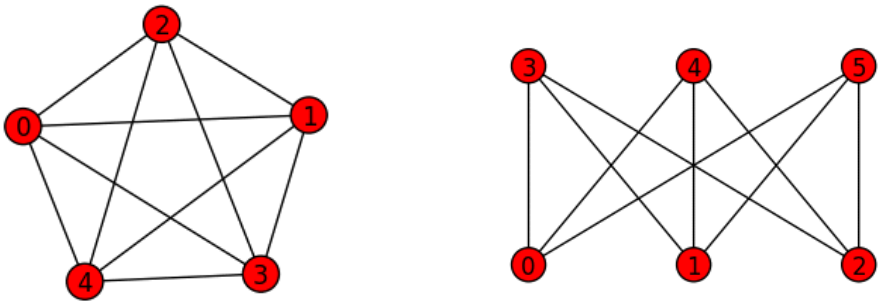

Figure 4 : Deux graphes non planaires K_5 à gauche et $K_{3,3}$ à droite

Région : Soit *F* la représentation planaire de *G*, une région est l'ensemble de points de $\mathbb{R}^2$ qu'il est possible de relier à un point donné sans entrecroiser d'arêtes de *F*.

Sous-graphe : Graphe $H = (E, V)$ vérifiant $E \subset A$ et $V \subset S$. On dit que *G contient H*.

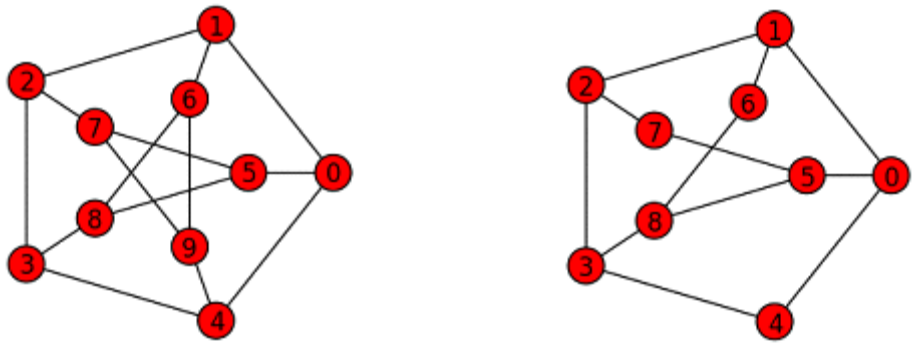

Figure 5 : Le graphe de Petersen, à gauche, contient le graphe H de droite

Subdivision : ajout de sommets de degré deux aux milieux d'arêtes.

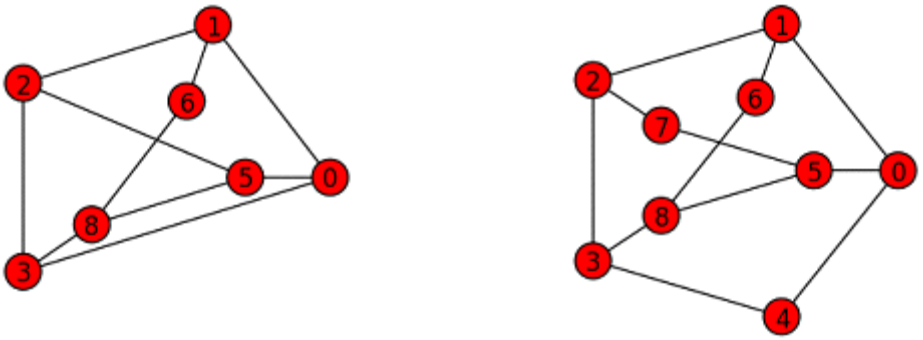

Figure 6 : Le graphe H, à droite, est une subdivision du graphe de gauche, qui en est une de $K_{3,3}$

Triangulation *:* Toutes les régions sont délimitées par trois arêtes.

2ème partie :

Coloriage adapté : on autorise le coloriage de $\boldsymbol{v}$ avec la couleur *i* si $\boldsymbol{v}$ appartient à *Fi*.

Compatibilité : le coloriage {0, 1, ..., *n*} d'un *n*-simplexe triangulé est dit compatible si les sommets du *n*-simplexe portent les *n*+1 couleurs et si tout sommet sur une grande face ne porte pas la couleur du sommet opposé.

Conformité : une cellule est conforme si elle porte toutes les couleurs

Coordonnées barycentriques : Pour tout point $\boldsymbol{v}$ d'un n-simplexe définit par $\{\boldsymbol{a_0}, \boldsymbol{a_1}, ..., \boldsymbol{a_n}\}$, il existe un unique couple $(v_0, v_1, ..., v_n)$ de $(\mathbb{R}^{+*})^n$, tel que :

$$\boldsymbol{v} = \sum v_i . \boldsymbol{a_i} \quad \text{ET} \quad \sum v_i = 1$$

Dualité : notion non intrinsèque mais propre à une représentation *F* du graphe étudié *G*. Le graphe dual a pour sommet les régions (ou cellules) de *F*, deux sommets sont adjacents si les régions correspondantes le sont.

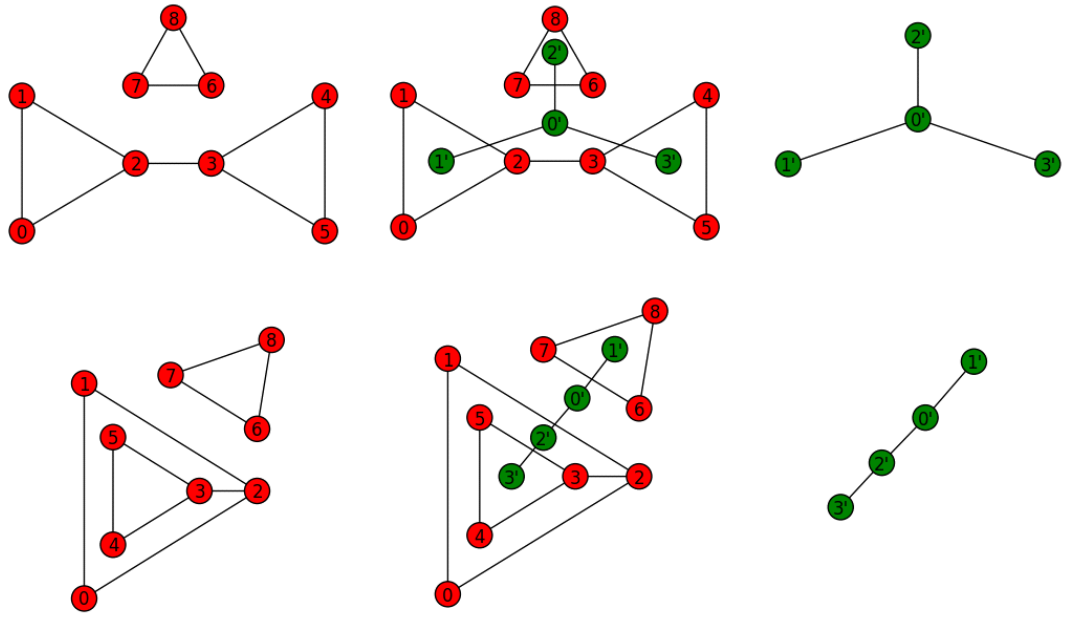

Figure 7 : deux représentations d'un même graphe entrainent différents graphes duaux

Enveloppe convexe : ensemble des barycentres à poids positifs.

Face : enveloppe convexe de *n* des points définissant le *n*-simplexe

***n*-simplexe :** enveloppe convexe de *n*+1 points n'étant pas un (*n*-1)-simplexe.

Triangulation *:* On considère un ensemble de sommets permettant de définir *m n*-simplexes disjoints au sens large, que nous appelons *cellules*, dont la réunion forme un nouveau *n*-simplexe. Nous appelons *diamètre* d'une triangulation le plus grand diamètre métrique de ses cellules.

Programme en Python

```
try : import networkx as nx
except ImportError:
    print("Networkx n'est pas sur votre ordinateur, pour l'installer :\n
          1. importez le sur internet\n
          2. extrayez le dossier zip\n
          3. placer le dossier dans C:\Python\Lib\site-packages\n
          4. rejouez ce fichier");
import matplotlib.pyplot as plt import
numpy as np
from mayavi import mlab
import random

#-----------------------------
# Fonctions auxillaires
#-----------------------------

def entrer_nombre(phrase):
    while True:
        try: n = eval(raw_input(phrase).replace(' ', '')); break
        except: print("Ceci n'est pas un nombre")
    return n

def ordonner(degres):
    liste = []
    for sommet in degres.keys():
        ctr = False
        for i in range(len(liste)):
            if degres[sommet] > degres[liste[i][0]]:
                liste[i:i], ctr = [[sommet, 0]], True; break if
        not ctr: liste.append([sommet, 0])
    return liste

def colorier_sommet(aretes, couleur, sommet):
    indice = -1
    while True:
        indice, ctr = indice + 1, True
        try:
              for sommet_colorie in couleur[indice]:
                if {sommet, sommet_colorie} in aretes: ctr = False; break if
            ctr: couleur[indice].append(sommet); break
        except IndexError: couleur.append([sommet]); break

def trier(degres, ordre):
    nouveau_ordre =  [] for
    sommet in ordre:
        ctr = False
        for i in range(len(nouveau_ordre)):
            if sommet[1] > nouveau_ordre[i][1]:
                nouveau_ordre[i:i], ctr = [sommet], True; break
            elif sommet[1] == nouveau_ordre[i][1]:
                  if degres[sommet[0]] > degres[nouveau_ordre[i][0]]:
                    nouveau_ordre[i:i], ctr = [sommet], True; break if
        not(ctr): nouveau_ordre.append(sommet)
      return nouveau_ordre

#-----------------------------
# Création d'un type Graphe
#-----------------------------

class Graphe(object):
    def __init__(self, nom = ''):
        self.nom, self.degres, self.aretes, self.position = nom, {}, [], {}
        self.plt_couleur = {0:'r', 1:'y', 2:'c', 3:'w', 4:'k', 5:'m', 6:'b',
                            7:'g'}
```

```
        self.mlab_couleur = {0 : (1, 0, 0), 1 : (0, 1, 0), 2 : (0, 0, 1),
                             3: (1,1,1), 4: (0, 0, 0), 5: (1, 1, 0),
                             6: (1, 0, 1), 7: (0, 1, 1)}

    def __add__(self, liste_aretes):
    """
        les aretes sont de la forme {sommet1, sommet2}
        """
        for arete in liste_aretes:
            if (len(arete) == 2) and (not(arete in self.aretes)):
                  self.aretes += [arete]
                  for i in arete: self.degres[i] = self.degres.get(i,0) + 1

    def __sub__(self, liste_aretes):
        for arete in liste_aretes:
            try: self.aretes.remove(aretes)
            except: pass

#-----------------------------
# Interface utilisateur
#-----------------------------

    def definir_position_sommet(self, sommet, dim = 2):
        print('sommet ' + str(sommet) + ' :')
        x = entrer_nombre('abscisse x : ') y
        = entrer_nombre('ordonnee y : ') if
        dim == 3:
            z = entrer_nombre('hauteur z : ')
            self.position[sommet] = [x, y, z]
        else: self.position[sommet] = [x, y]
    def definir_position(self, dim = 2):
        for sommet in self.degres.keys():
          self.definir_position_sommet(sommet, dim)

    def definir_couleurs_2D(self):
        n = entrer_nombre('combien de couleurs voulez-vous definir ? ')
        print("      =====   =======\n
                      Alias   Couleur\n
                      =====   =======\n
              'b'     bleu\n      'g'     vert\n      'r'     rouge\n
              'c'     cyan\n      'm'     magenta\n      'y'     jaune\n
              'k'     noir\n      'w'     blanc\n      =====   =======\n\n")
        for i in range(n):
              self.plt_couleur[i] = raw_input('couleur ' + str(i) + ' : ')

    def definir_couleurs_3D(self):
        n = entrer_nombre('combien de couleurs voulez-vous definir ? ')
        print("entrez un triplet de flottants entre 0 et 1,
              (rouge, vert, bleu)")
        for i in range(n):
              self.mlab_couleur[i] = eval(raw_input('couleur ' +
                                                    str(i) + ' : '))
#-----------------------------
# Coloriage
#-----------------------------

    def coloriage_proche_en_proche(self):
        ordre, couleur = ordonner(self.degres), []
        while len(ordre)!=0:
            colorier_sommet(self.aretes, couleur, ordre[0][0])
            for sommet_adjacent in ordre :
                  if {ordre[0][0], sommet_adjacent[0]} in self.aretes:
                    sommet_adjacent[1] += 1
            ordre.pop(0)
            ordre = trier(self.degres, ordre)
        return couleur
```

```
    def coloriage_lineaire_sommet(self):
        couleur, ordre = [], ordonner(self.degres)
        for sommet in ordre: colorier_sommet(self.aretes, couleur, sommet[0])
        return couleur

    def coloriage_lineaire_couleur(self):
        couleur, ordre = [], ordonner(self.degres)
        while len(ordre) != 0:
            index = -1
            couleur.append([])
            while index < len(ordre)-1:
                index, ctr = index + 1, True
                for sommet_adjacent in couleur[-1]:
                    if {ordre[index][0], sommet_adjacent} in self.aretes:
                        ctr = False
                        break
                if ctr:
                    couleur[-1].append(ordre[index][0])
                    ordre.pop(index)
                    index -= 1
        return couleur

#-----------------------------
# Dessin
#-----------------------------

     def dessin_2D(self, coloriage = 'proche_en_proche'):
        if coloriage.lower() == 'lineaire_sommet':
              couleur = self.coloriage_lineaire_sommet()
        elif coloriage.lower() == 'lineaire_couleur':
              couleur = self.coloriage_lineaire_couleur()
        else : couleur = self.coloriage_proche_en_proche()
        graphe, dic, liste = nx.Graph(self.aretes), {}, []
        for index in range(len(couleur)):
            for sommet in couleur[index]: dic[sommet] = index
        for i in graphe.nodes(): liste.append(self.plt_couleur[dic[i]])
        try : nx.draw(graphe, self.position, node_color = liste)
        except: nx.draw_spring(graphe, node_color = liste)
        plt.show()
        plt.close()

     def dessin_3D(self, coloriage = 'proche_en_proche'):
        if coloriage.lower() == 'lineaire_sommet':
              couleur = self.coloriage_lineaire_sommet()
        elif coloriage.lower() == 'lineaire_couleur':
              couleur = self.coloriage_lineaire_couleur()
        else : couleur = self.coloriage_proche_en_proche()
        graphe = nx.Graph(self.aretes)
        mlab.figure(1, bgcolor=(0.3, 0.2, 0.4))
        try :
            pos = self.position
            for i in range(len(couleur)):
                xyz=np.array([self.position[sommet] for sommet in couleur[i]])
                pts = mlab.points3d(xyz[:,0], xyz[:,1], xyz[:,2],
                                    scale_factor=0.1, resolution=20,
                                    color = self.mlab_couleur[i])
        except :
            pos = nx.spring_layout(graphe,dim=3)
            for i in range(len(couleur)):
                xyz=np.array([pos[sommet] for sommet in couleur[i]])
                pts = mlab.points3d(xyz[:,0], xyz[:,1], xyz[:,2],
                                    scale_factor=0.1, resolution=20,
                                    color = self.mlab_couleur[i])
        for (i,j) in self.aretes:
              mlab.plot3d(np.array([pos[i][0], pos[j][0]]),
                          np.array([pos[i][1], pos[j][1]]),
```

```
                            np.array([pos[i][2], pos[j][2]]),
                            tube_radius=0.01, color=(0.8, 0.8, 0.8))
        mlab.show()


 #----------------------------
# Lecture des 633 configurations de Robertson, Sanders, Seymour et Thomas
 #----------------------------

def lireConfig(nomDossier, pourcentageAffiche, coloriage = 'proche_en_proche'):
    dossier = [ line.rstrip() for line in open(nomDossier, "r").readlines() ]
    i, count = 0, 0
    while i < len(dossier):
        graphe = Graphe(dossier[i])
        i, aretes = i+3, []
        while dossier[i+1][3:8] != 'point' :
            i += 1
            aretes.append({eval(dossier[i][14:16]), eval(dossier[i][19:21])})
        graphe + aretes
        while dossier[i][3:9] != 'region': i += 1
        while dossier[i+1][0:3] != 'Fin' :
            i += 1
            graphe.position[eval(dossier[i][0:2])] = [eval(dossier[i][6:9]) ,
                                                      eval(dossier[i][10:13])]
        i += 3
        if coloriage.lower() == 'lineaire_sommet':
              couleur = graphe.coloriage_lineaire_sommet()
        elif coloriage.lower() == 'lineaire_couleur':
              couleur = graphe.coloriage_lineaire_couleur()
        else : couleur = graphe.coloriage_proche_en_proche()
        indice = random.random()
        if len(couleur) > 4:
              print(str(graphe.nom) + " n'a pas reussi le test : "
                    + str(len(couleur)) + ' couleurs')
              count += 1
        else: print(str(graphe.nom) + ' a reussi le test : ' + str(len(couleur))
+ ' couleurs')
        if indice < pourcentageAffiche:
            if indice < pourcentageAffiche/2 : graphe.dessin_3D(coloriage)
            else : graphe.dessin_2D(coloriage)
    return count

#----------------------------
# Exemples
#----------------------------

print('    ============================== PRESENTATION
==============================')
print("Introduction : Voici le module de coloriage de graphes,
      nous testerons d'abord nos algorithmes sur des graphes simples
      4-coloriables puis nous montrerons leurs différences sur deux
      exemples particuliers\n")
raw_input("On essaye notre algorithme sur une grande quantite de graphes,
         a l'aide du fichier configuration.txt :\n")
print("Le coloriage lineaire 'sommet' fut non optimal "
      + str(lireConfig('configurations.txt', 1.0/100, 'lineaire_sommet'))
      + ' fois\n')
print("Le coloriage lineaire 'couleur' fut non optimal "
      + str(lireConfig('configurations.txt', 1.0/100, 'lineaire_couleur'))
      + ' fois\n')
print("Le coloriage de proche en proche fut non optimal " +
      str(lireConfig('configurations.txt', 1.0/100)) + ' fois\n')

raw_input("Voici quelques cartes illustrant les limites de nos algorithmes :\n")
print('Premier graphe :
      les deux algorithmes renvoient plus que quatres couleurs')
```

```
Aretes_graphe_erreur = [{0, 'Paris'}, {'Paris','Londres'}, {'Paris','Pablo
Alto'}, {'Paris','El Paso'}, {'Paris','Ulm'}, {2,'Londres'}, {3,'Londres'},
{'Londres','Pablo Alto'}, {'Londres', 'Copenhague'}, {'Londres', 'Moscou'},
{'Pablo Alto','El Paso'}, {'Pablo Alto', 'Copenhague'}, {'El Paso','Ulm'}, {'El
Paso', 'Copenhague'}, {'Ulm',4}, {'Ulm',5}, {'Ulm', 'Copenhague'}, {'Ulm',
'Moscou'}, {'Caracas', 6}, {'Caracas', 7}, {'Caracas', 8}, {'Caracas', 9},
{'Caracas', 10}, {'Caracas', 'Moscou'}, {'Copenhague', 'Moscou'}, {'Moscou',
11}, {'Moscou', 12}, {'Moscou', 13}, {'Moscou', 1}]
##Aretes_graphe_erreur = [{0, 1}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {1, 10}, {2,
4}, {3, 4}, {4, 5}, {4, 16}, {4, 17}, {5, 6}, {5, 16}, {6, 7}, {6, 16}, {7, 8},
{7, 9}, {7, 16}, {7, 17}, {10, 11}, {10, 12}, {10, 13}, {10, 14}, {10, 15}, {10,
17}, {16, 17}, {17, 18}, {17, 19}, {17, 20}, {17, 21}]
Graphe_erreur = Graphe()
Graphe_erreur + Aretes_graphe_erreur
Graphe_erreur.position = {0: [0, 0.3], 'Paris': [0, 0], 2: [0.07, 0.7], 3: [0,
0.55], 'Londres': [0.2, 0.5], 'Pablo Alto': [0.2, 0.37], 'El Paso': [0.28,
0.27], 'Ulm': [0.38, 0.23], 4: [0.5, 0.35], 5: [0.55, 0.25],'Caracas': [0.77,
0.15], 6: [0.8, 0], 7: [0.9, 0.02], 8: [0.98, 0.1], 9: [1, 0.2], 10: [0.95,
0.3], 'Copenhague': [0.33, 0.4], 'Moscou': [0.52, 0.52], 11: [0.45, 0.75], 12:
[0.55, 0.78], 13: [0.65, 0.73], 1: [0.7, 0.63]}
##Graphe_erreur.position = {0: [0, 0.3], 1: [0, 0], 2: [0.07, 0.7], 3: [0,
0.55], 4: [0.2, 0.5], 5: [0.2, 0.37], 6: [0.28, 0.27], 7: [0.38, 0.23], 8: [0.5,
0.35], 9: [0.55, 0.25], 10: [0.77, 0.15], 11: [0.8, 0], 12: [0.9, 0.02], 13:
[0.98, 0.1], 14: [1, 0.2], 15: [0.95, 0.3], 16: [0.33, 0.4], 17: [0.52, 0.52],
18: [0.45, 0.75], 19: [0.55, 0.78], 20: [0.65, 0.73], 21: [0.7, 0.63]}
print('coloriage de proche en proche, illustration en 2D')
Graphe_erreur.dessin_2D()
for i in Graphe_erreur.position.keys():
        Graphe_erreur.position[i].append(0)
print('coloriage lineaire, illustration en 3D\n')
Graphe_erreur.dessin_3D('lineaire_sommet')

print('Deuxieme graphe : le deuxieme coloriage fonctionne mieux')
Aretes_graphe_erreur_2 =[{0, 4}, {1, 4}, {2, 4}, {3, 4}, {4, 5}, {4, 6}, {4, 7},
{5, 6}, {5, 14}, {5, 15}, {6, 7}, {6, 15}, {7, 8}, {7, 9}, {7, 10}, {7, 15},
{11, 14}, {12, 14}, {13, 14}, {14, 15}, {14, 16}, {15, 16}, {16, 17}, {16, 18},
{16, 19}, {16, 20}, {16, 21}]
Graphe_erreur_2 = Graphe('le deuxieme coloriage fonctionne mieux')
Graphe_erreur_2 + Aretes_graphe_erreur_2
print('coloriage lineaire, illustration en 2D')
Graphe_erreur_2.dessin_2D('lineaire_sommet')
print('coloriage de proche en proche, illustration en 2D\n')
Graphe_erreur_2.dessin_3D()
```