Rapport Mini Projet - Info Fondamentale (électif)

Préambule

Nous avons décidé d'utiliser la Programmation Orientée Objet (POO) car nous visualisons ce problème comme une organisation structurée (graphe) constituée de points et de segments reliant ces points entre eux. Ce projet a été codé sur *Spyder* en *python*.

Constructions générales nécessaires à l'élaboration des méthodes de coloration

Classe Point

Nous allons d'abord vous présenter la classe *point* utilisée par la suite dans la classe graphe pour réaliser les deux méthodes de coloration. Un point est défini par son nom, sa couleur et ses coordonnées. Les coordonnées seront utilisées pour la partie affichage des graphes. La couleur sera affectée par la suite dans les méthodes de coloration.

```
7   import pygame
8   import math
9
10  class point:
11
12      def __init__(self,nom):
13          self.nom=nom
14          self.couleur=""
15          self.coord=(0,0)
```

Classe Graphe

La classe graphe est quant à elle constituée de trois attributs. Le premier attribut **V** représente l'ensemble des **sommets du graphe** sous forme d'une liste de points. Le second attribut **E** représente les **arrêtes du graphe** et est sous la forme d'une liste d'arrêtes, sachant que chaque arrête est une liste de deux points. Enfin, le dernier attribut **C** représente l'ensemble des **couleurs qui peuvent être attribués aux sommets**. Il est sous forme de liste également.

```
23
24  class graphe:
25
26      def __init__(self,V,E,C):
27          self.V=V
28          self.E=E
29          self.C=C
30
```

Exercice 1 : 2-coloration

Complexité

- La complexité de notre méthode de 2-coloration est de n.
- Aucune optimisation est possible car la complexité n est le minimum pour parcourir une liste.

```python
def couleur2(self):#colorie le graphe en 2 couleurs
    for i in range (len(self.E)):
        if(self.E[i][0].couleur=="" and self.E[i][1].couleur==""):
            self.E[i][0].couleur=self.C[0]
            self.E[i][1].couleur=self.C[1]
        elif(self.E[i][0].couleur==self.C[0] and self.E[i][1].couleur==""):
            self.E[i][1].couleur=self.C[1]
        elif(self.E[i][0].couleur==self.C[1] and self.E[i][1].couleur==""):
            self.E[i][1].couleur=self.C[0]
        elif(self.E[i][1].couleur==self.C[0] and self.E[i][0].couleur==""):
            self.E[i][0].couleur=self.C[1]
        elif(self.E[i][1].couleur==self.C[1] and self.E[i][0].couleur==""):
            self.E[i][0].couleur=self.C[0]

    for i in range(len(self.E)):
        if(self.E[i][0].couleur==self.E[i][1].couleur):
            return False
    return True
```

O(n)

K*n

K*n

Explication du code

Dans le main, on a définit que la liste de couleurs C était composée de rouge et bleu. Donc C[0] représente la couleur rouge et C[1] représente la couleur bleue.

Cette méthode parcourt la liste d'arrêtes (E) avec la boucle for. Elle vérifie ensuite la coloration des deux points :

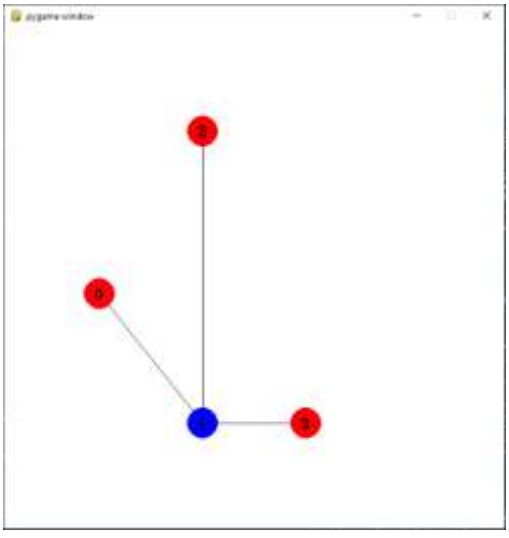
- Aucun des deux points n'est colorié => on colorie le premier point en rouge et le deuxième en bleu.
- Si l'un des deux est colorié => on colorie l'autre point avec la couleur qui n'est pas utilisée.

La seconde boucle for permet de vérifier que le graphe est correctement colorié :

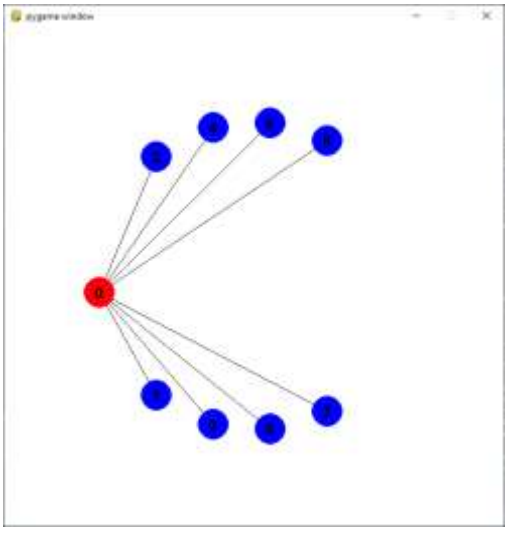
- Si les deux points sont coloriés de la même couleur => return false.
- Si les deux points ne sont pas coloriés de la même couleur => return true.

Cette méthode couleur2 est un booléen qui colorie et teste la coloration.

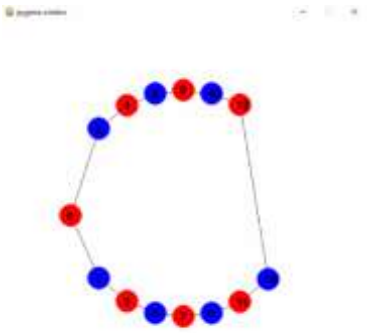
Graphe



Graphe avec 4 points



Graphe avec 9 points



Graphe avec 14 points

En choisissant les arrêtes de façon à ce qu'il n'y ait pas de sommets reliés entre eux sous forme de triangle, on peut constituer un graphe colorié en deux couleurs avec une infinité de points.

Exercice 2 : 3-coloration

Complexité

- La complexité de notre méthode de 3-coloration est de 3^n car la boucle for se répète 3 fois (la longueur de la liste C est de 3 couleurs) et la fonction est appelé n fois (nombre de point à colorier) de manière récursive dans la boucle for précédente.
- On ne voit pas d'optimisation possible.

```
77      def complet(self):      #test si tous les points ont une couleur
78          for i in range(len(self.V)):
79              if(self.V[i].couleur==""):
80                  return False
81          return True
```

O(n)

```
71      def verificateur(self):     #test si des point reliés sont de couleurs différentes
72          for i in range(len(self.E)):
73              if(self.E[i][0].couleur!="" and self.E[i][0].couleur==self.E[i][1].couleur):
74                  return False
75          return True
```

O(n)

```
58      def couleur3(self,i):#colorie le graphe en 3 couleurs
59          if(self.complet() and self.verificateur()):
60              return True
61          for j in range(len(self.C)):
62              self.V[i].couleur=self.C[j]
63              if(self.verificateur()):
64                  if(self.couleur3(i+1)):
65                      return True
66                  self.V[i].couleur=""
67              else:
68                  self.V[i].couleur=""
69          return False
```

K*n

3^n

O(3^n)

Explication du code

La méthode *complet* permet de tester si on a attribué une couleur à chaque point.

Elle est composée d'une boucle for qui parcourt la liste de sommets, puis teste :

- Si le point n'a pas de couleur : return false
- Return true sinon

La méthode *vérificateur* permet de tester si les points reliés sont tous d'une couleur différente.

Elle est composée d'une boucle for qui parcourt la liste d'arrêtes, puis teste :

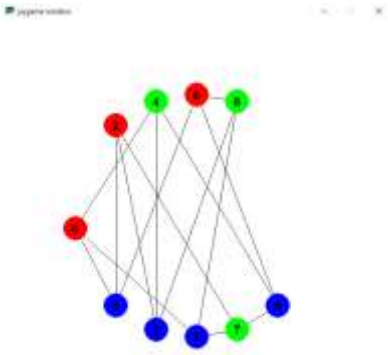
- Si le premier point à l'extrémité de l'arrête a une couleur et que les deux points aux extrémités sont de la même couleur : return false
- Return true sinon

La méthode *couleur3* colorie le graphe en 3 couleurs.

- Elle vérifie si chaque sommet a une couleur attribuée et si chaque segment relie deux points de couleurs différentes (en utilisant les méthodes *complet* et *vérificateur*).
- La variable i en paramètre de la fonction va parcourir la liste des sommets.
- La variable j de la boucle for parcourt la liste de couleurs
    - On affecte une couleur au sommet
    - On vérifie que les deux points reliés par un même segment sont de couleurs différentes, si c'est true on reboucle sur la méthode (récursivité) et on passe aux points suivants. Tant que les points suivants sont validés (*vérificateur* retourne true) on continue.
    - Si jamais la méthode *vérificateur* n'est plus validée, on réinitialise, on déboucle et on retourne en arrière au moment où le vérificateur était juste (=on déboucle d'une étape). Puis on change de couleur pour le point suivant si le vérificateur est validé on continue, sinon on déboucle, …
- Si le graphe n'est vérifié pour aucune succession de couleur, la méthode return false.

Graphe

Ce graphe est connu et s'appelle le graphe de Peterson. Il est 3-coloriable.



<u>Exercice 3 :</u> Programme vérificateur de la 3-coloration d'un graphe

Complexité

Ce programme est de complexité O(n).

```
71      def verificateur(self):     #test si des point reliés sont de couleurs différentes
            for i in range(len(self.E)):
                if(self.E[i][0].couleur!="" and self.E[i][0].couleur==self.E[i][1].couleur):
                    return False
75          return True
```

O(n)

Explication du code

La méthode *vérificateur* permet de tester si les points reliés sont tous d'une couleur différente.

Elle est composée d'une boucle for qui parcourt la liste d'arrêtes, puis teste :

- Si le premier point à l'extrémité de l'arrête a une couleur et que les deux points aux extrémités sont de la même couleur : return false
- Return true sinon

<u>Affichage</u>

La méthode *EquationCercle* permettra par la suite d'afficher les différents points du graphe selon l'équation d'un cercle. On utilise un cercle pour que les arrêtes entre les sommets ne croisent que les points qui nous intéressent.

La méthode Affiche Graphe permet d'afficher le graphe. Les lignes 83 à 94 permettent d'initialiser l'interface : initialiser la fenêtre pygame, lui donner une surface, initialiser les couleurs qu'on utilisera par la suite, définir une police, remplir la fenêtre en blanc et actualiser les changements.

```
75      def EquationCercle(x):
76          memoire=200**2-(x-325)**2
77          if(memoire<0):
78              memoire=-memoire
79          y=math.sqrt(memoire)+325
80          return y
81
82      def AfficheGraphe(graphe):#affiche le graphe
83          pygame.init()#initialise la fenetre pygame
84          tailleSurface=(650,650)
85          surface=pygame.display.set_mode(tailleSurface)
86          Blanc=(255,255,255)
87          Bleu=(0,0,255)
88          Rouge=(255,0,0)
89          Vert=(0,255,0)
90          Noir=(0,0,0)
91          police=pygame.font.Font(None,30)
92
93          surface.fill(Blanc)
94          pygame.display.flip()
95          x=124
96          diametre=400
97          while(diametre%((len(graphe.V)/2)+1)!=0):#commence l'affichage du graphe
98              diametre=diametre+1
99          memoire=diametre/((len(graphe.V)/2)+1)
100         compteur=0
```

1

1- La première boucle while permet d'adapter le diamètre du cercle au nombre de points afin que le pas en ordonnée soit un entier (le diamètre du cercle doit être divisible par le nombre de points du graphe).

2- La seconde boucle while calcule des coordonnées de chaque point.

- Boucle if : affecte des coordonnées au point le plus à gauche du cercle
- Boucle elif : affecte des coordonnées au point le plus à droite du cercle
- Boucle else : affecte aux autres points compris entre ces deux points des coordonnées
    - Affectation de coordonnées au point le plus en bas (y le plus grand)
    - Affectation de coordonnées point le plus en haut (y le plus petit)
- x+pas (pour positionner les points suivants)

NB : origine de l'axe en haut à gauche

3- Cette première boucle for permet d'afficher les arrêtes entre les points.

4- Cette seconde boucle for affiche les sommets au-dessus des intersections des arrêtes. Les sommets sont affichés de la couleur dont ils sont définis dans les méthodes de coloration. On affiche ensuite le nom des points (en ayant défini la police auparavant).

5- Cette dernière boucle garde la fenêtre et l'instance de pygame ouvertes jusqu'à ce qu'on appuie sur la croix rouge.

```
100         compteur=0
101         while compteur!=len(graphe.V):#calcule des coordonnees de chaque point
102             if(x==124):
103                 graphe.V[compteur].coord=(x,EquationCercle(x))
104                 compteur=compteur+1
105             elif(compteur==len(graphe.V)-1):
106                 graphe.V[compteur].coord=(x,EquationCercle(x))
107                 compteur=compteur+1
108             else:
109                 graphe.V[compteur].coord=(x,EquationCercle(x))
110                 compteur=compteur+1
111                 if compteur!=len(graphe.V):
112                     graphe.V[compteur].coord=(x,EquationCercle(x)-(EquationCercle(x)-324)*2)
113                     compteur=compteur+1
114                 else:
115                     break
116             x=x+memoire
117         for i in range (len(graphe.E)):#coloriage et affichage des points
118             pygame.draw.line(surface,Noir,graphe.E[i][0].coord,graphe.E[i][1].coord,1)
119         for i in range (len(graphe.V)):
120             if(graphe.V[i].couleur=="rouge"):
121                 pygame.draw.circle(surface,Rouge,graphe.V[i].coord,20)
122             elif(graphe.V[i].couleur=="vert"):
123                 pygame.draw.circle(surface,Vert,graphe.V[i].coord,20)
124             else:
125                 pygame.draw.circle(surface,Bleu,graphe.V[i].coord,20)
126             texte=police.render(graphe.V[i].nom,True,Noir)
127             surface.blit(texte,(graphe.V[i].coord[0]-6,graphe.V[i].coord[1]-8))
128         pygame.display.flip()
129
130         launched=True#boucle pour garder la fenetre pygame ouverte
131         while launched:
132             for event in pygame.event.get():
133                 if event.type == pygame.QUIT:
134                     launched=False
135         pygame.quit()
```

```
137 def AfficheErreur():#affiche un message d'erreur si le graphe n'est pas coloriable
138     pygame.init()#initialise la fenetre pygame
139     tailleSurface=(650,650)
140     surface=pygame.display.set_mode(tailleSurface)
141     Blanc=(255,255,255)
142     Noir=(0,0,0)
143     police=pygame.font.Font(None,30)
144     surface.fill(Blanc)
145     texte=police.render("Le graphe n'est pas valide",True,Noir)
146     surface.blit(texte,(200,270))
147     pygame.display.flip()
148 
149     launched=True#boucle pour garder la fenetre pygame ouverte
150     while launched:
151         for event in pygame.event.get():
152             if event.type == pygame.QUIT:
153                 launched=False
154     pygame.quit()
155 
```

Cette méthode *AfficheErreur* affiche un message d'erreur sur la console si le graphe n'est pas coloriable.

Main

```
168 #Main
169 if __name__ == '__main__':
170     point1= point("0")
171     point2= point("1")
172     point3= point("2")
173     point4= point("3")
174     point5= point("4")
175     point6= point("5")
176     point7= point("6")
177     point8= point("7")
178     point9= point("8")
179     point10= point("9")
180     point11= point("10")
181     point12= point("11")
182     point13= point("12")
183     point14= point("13")
184     point15= point("14")
185     couleur=["rouge","bleu","vert"]
186     graphe1= graphe([point1,point2,point3,point4,point5,point6,point7,point8,point9,point10
187     if(len(graphe1.C)==2 and graphe1.couleur2()):
188         AfficheGraphe(graphe1)
189     elif(len(graphe1.C)==3 and graphe1.couleur3(0)):
190         AfficheGraphe(graphe1)
191     else:
192         AfficheErreur()
```

- Création de points
- Création d'une liste de couleurs
- Création d'une instance de graphe
- Si on a colorié avec 2 couleurs dans la liste et que le graphe est bien 2-coloriable
    - On fait l'affichage du graphe colorié avec 2 couleurs
- Si on a colorié avec 3 couleurs dans la liste et que le graphe est bien 3-coloriable
    - On fait l'affichage du graphe colorié avec 3 couleurs
- On lance la méthode AfficheErreur() sinon