**Rapport de projet d’informatique fondamentale**

**Introduction**

Pour conclure notre cours d’informatique fondamentale, nous avons dû réaliser un projet en lien avec la théorie des graphes. Plus précisément sur l’aspect de la n-coloration d’un graphe. Vous pouvez retrouver l’énoncé de notre projet dans le dossier fourni.

Afin de mener à bien ce projet, nous avons commencé par faire un état de l’art. Sans étonnement, ce projet a déjà été fait plusieurs fois. Ce projet était donc très réalisable. Après s’en être donné une idée, nous nous sommes documentés sur les graphes à travers des articles et des vidéos pour renforcer nos acquis de cours.

Nous avons ensuite réfléchi au langage de programmation à utiliser. Nous faisons beaucoup de C# en cours, un langage orienté objet. Nous aurions voulu y faire une interface graphique pour nos graphes. Après quelques recherches, nous avons remarqué que manipuler et afficher des graphes en C# était plus complexe qu’en python. En effet, nous n’avons trouvé qu’une librairie, et peu de documentation sur les graphes dans ce langage. Par conséquent, nous aurions dû coder les différentes classes.

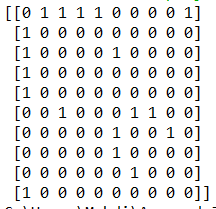
Bien que cela soit possible de réaliser le projet en C#, il nous a finalement semblé plus approprié de le faire avec python.

Nous avons donc finalement bien opté pour python qui possède la librairie networkx. C’est une puissante libraire qui permet de manipuler aisément des graphes. Cela nous a aussi permis de nous former sur python, un langage que nous n’avons que très peu pratiqué.

**Explication Networkx et de la première partie de nos algorithmes :**

Networkx est une librairie assez complète sur la manipulation des graphes. Elle nous permet d’effectuer diverses opérations dessus. Le point important de la librairie est le fait que l’on puisse les générer avec différents tableaux d’éléments. En effet, on peut manipuler les graphes avec des tableaux de sommets et des tableaux d’arêtes. On a aussi la possibilité d’accéder aux paramètres des sommets, tels que leur taille à l’affichage ou leur couleur par exemple.

On manipule aussi des matrices d’adjacences. Ce sont des matrices carrées et symétriques composées de 0 et de 1, et dont l’utilité est de voir quels sont les sommets reliés entre eux. 1 signifie que 2 sommets sont reliés, et les 0 signifient qu’ils ne le sont pas.

Ci-dessous une matrice d’adjacence générée par la librairie dans notre algorithme :

Les indices de 0 à n correspondent aux sommets des graphes (sommet 0, sommet 1 etc).

Nous avons donc utilisé des boucles for pour parcourir ces matrices d’adjacences, et étudier la couleur des sommets reliés par une arête.

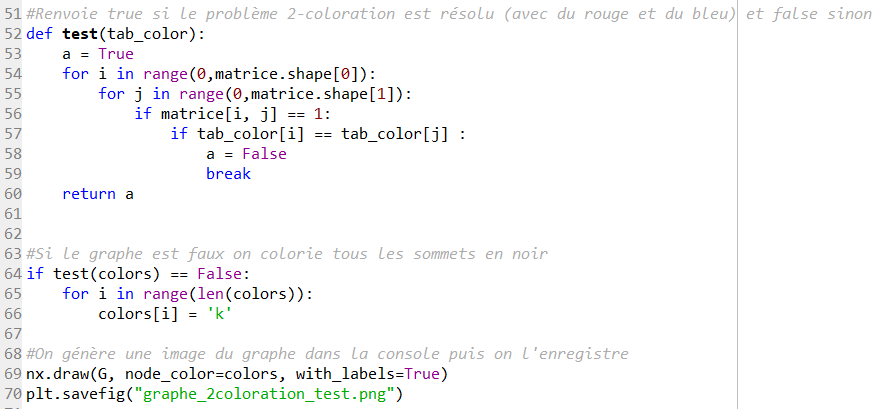
Nos algorithmes se décomposent globalement en sept parties :

* Importation des librairies networkx et matplotlib que l’on utilisera plus tard dans l’algorithme
* Création du graphe
* Création d’un tableau de couleur d’une taille égale au nombre de sommet du graphe
* Génération de la matrice d’adjacence
* Boucle for pour parcourir la matrice d’adjacence et étudier la couleur de chaque sommet rattaché par une arête
* Test pour vérifier la résolution du problème
* Partie optionnelle où nous générons une image du graphe

**Exercice 1 :**

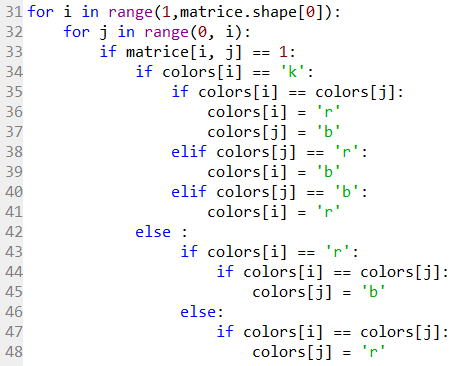






1. La matrice d’adjacence est symétrique, carrée et de hauteur et largeur n. On se place aussi dans le cas où la matrice est uniquement composée de 1.

Ci-dessous la première boucle trouvée pour la 2-coloration.



Les calculs de complexité (de calcul) des boucles for dans le cas d’une matrice nulle valent 2(n – 1) j dans tous les cas.

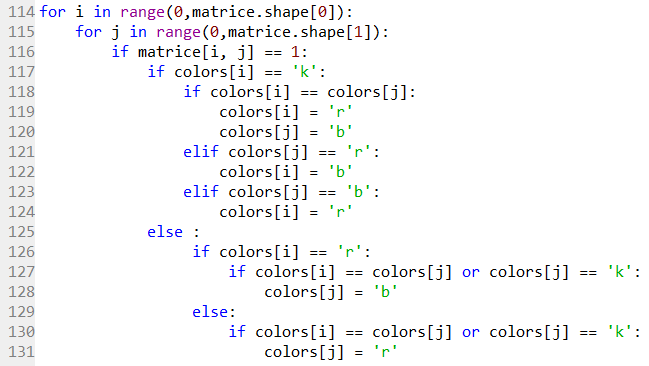
Calculs de complexité (de calcul) des boucles for dans le cas d’une matrice composée de 1 :

Meilleur cas : 6(n - 1)j soit O(nj)

Moyen cas : 13(n - 1)j/2 soit O(nj)

Pire cas : 7(n - 1)j soit O(nj)

1. Ci-dessous le premier algorithme trouvé pour la 2-coloration qui parcourt toute la matrice d’adjacence et étudie chacune des liaisons 2 fois :



Les calculs de complexité (de calcul) des boucles for dans le cas d’une matrice nulle valent 2n² dans tous les cas.

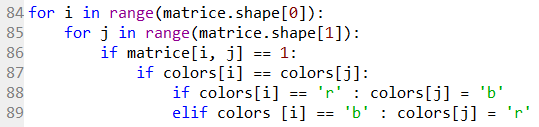
Calculs de complexité (de calcul) des boucles for dans le cas d’une matrice composée de 1 :

Meilleur cas : 6n² soit O(n²)

Moyen cas : 13n²/2 soit O(n²)

Pire cas : 7n² soit O(n²)

Ci-dessous une 3ème idée de boucle trouvée pour la 2-colorations, à l’issue de la recherche de la 3-coloration. Il y a moins de conditions, mais nous sommes obligés de parcourir toute la matrice afin de bien la colorer.



Les calculs de complexité (de calcul) des boucles for dans le cas d’une matrice nulle valent 2n² dans tous les cas.

Calculs de complexité (de calcul) des boucles for dans le cas d’une matrice composée de 1 :

Meilleur cas : 5n² soit O(n²)

Moyen cas : 11n²/2 soit O(n²)

Pire cas : 6n² soit O(n²)

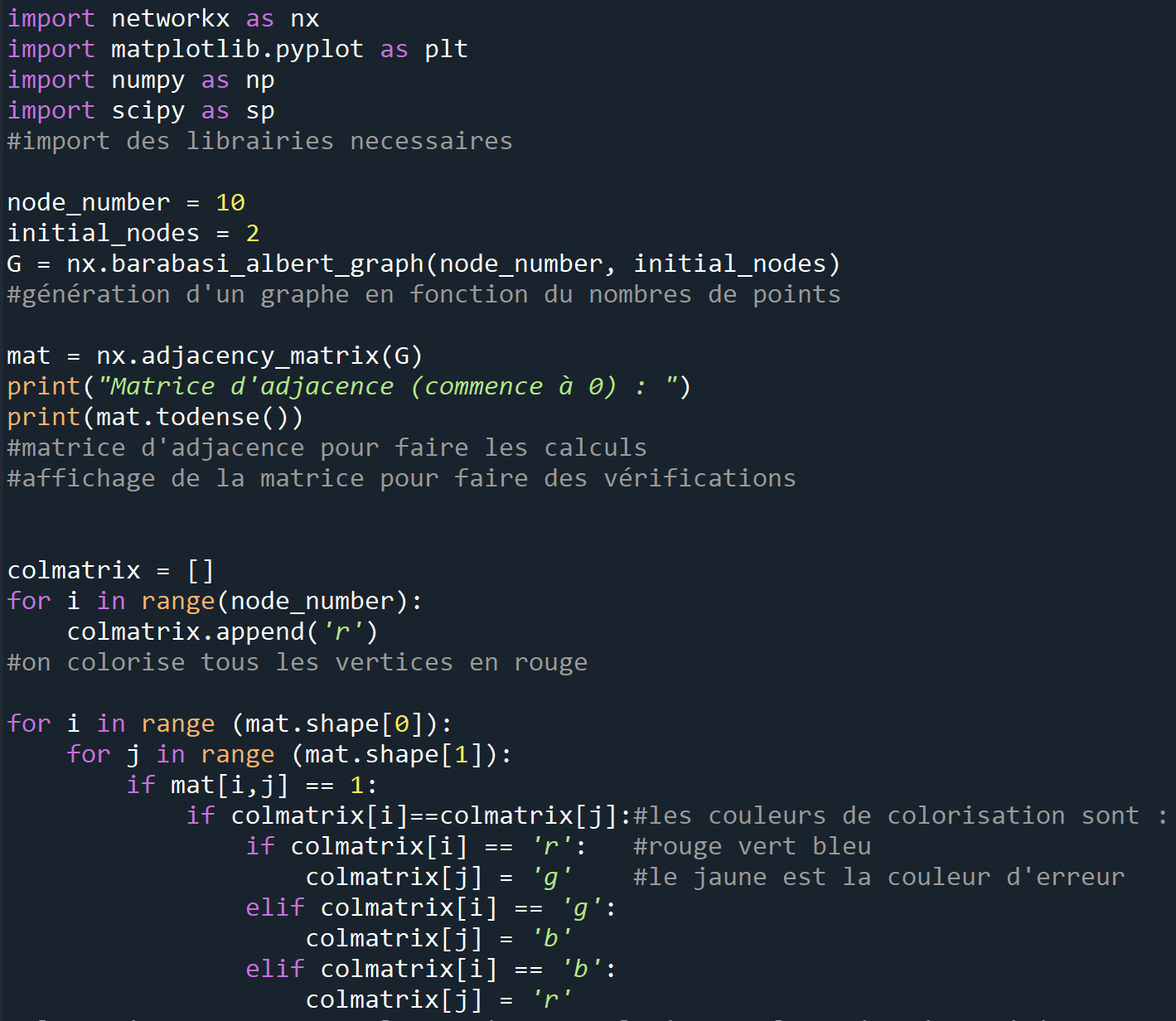
Dans les 3 cas énoncés au-dessus, la complexité de l’algorithme est polynomiale. Seul l’algorithme montré dans la question 2 et 1 a une complexité de O(nj) car nous avons divisé la taille de la matrice par 2 tout en retirant aussi l’étude de la diagonale. Cela permet de diviser le nombre d’opérations à effectuer par 2, moins n (la hauteur et la longueur de la matrice). La complexité de l’algorithme est donc réduite.

Dans les autres cas, la complexité des algorithmes des boucles for est O(n²), soit polynomiale.

1. On remarque que l’algorithme marche pour certaines dispositions. Mais si 3 nœuds sont reliés les uns entre les autres, l’algorithme ne marche pas. En effet, cette disposition n’est pas valable pour seulement 2 couleurs. Il en faudrait une 3ème pour qu’on puisse colorer un graphe où 3 sommets sont reliés les uns aux autres.

**Exercice 2 :**





1. Le programme comporte :

* 2 boucles for l’une dans l’autre qui parcourent toute la matrice
* 2 boucles if contenant une comparaison chacune
* Un choix entre 3 boucles if contenant chacune une comparaison et une assignation de variable

Les calculs de complexité (de calcul) des boucles for dans le cas d’une matrice nulle valent 2n² dans tous les cas.

Calculs de complexité (de calcul) des boucles for dans le cas d’une matrice composée de 1 :

Meilleur cas : 5n² soit O(n²)

Moyen cas : 6n² soit O(n²)

Pire cas : 7n² soit O(n²)

On peut donc dire que la complexité du programme est de O(n²), c’est une complexité polynômiale.

1. J’ai essayé de diviser le parcours de la matrice par 2 par rapport à la boucle for montrée dans la question 1.

En effet la matrice est symétrique, par conséquent, elle comporte 2 fois toutes les informations. Cependant, après avoir fait des tests, on remarque que le programme fonctionne moins bien. Parfois il ne colorise pas ce qu’il peut coloriser. J’ai aussi pensé à utiliser la fonction : all\_neighbours de networkx, qui retourne immédiatement tous les voisins d’un nœud. L’idée aurait été de colorer un sommet en fonction de ses voisins, puis « clear » cette donnée. Cela nous permettrait de traiter efficacement un grand nombre de sommets, plutôt que de devoir garder en mémoire toute une matrice, et de diminuer la complexité spatiale.

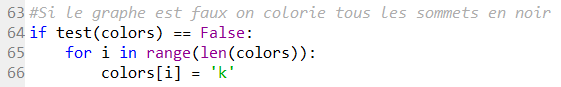
La complexité de calcul reste à déterminer. En effet, étant donné que Python est un langage haut niveau, beaucoup de fonctions sont déjà faites. Il n’est donc pas forcément facile d’accéder au contenu de tous les algorithmes pour calculer leur complexité.

1. Passé un certains nombre d’arêtes, il n’est plus possible de 3 coloriser. Le programme s’exécute quand même, mais il affiche en jaune les points qui ne peuvent pas être colorés.

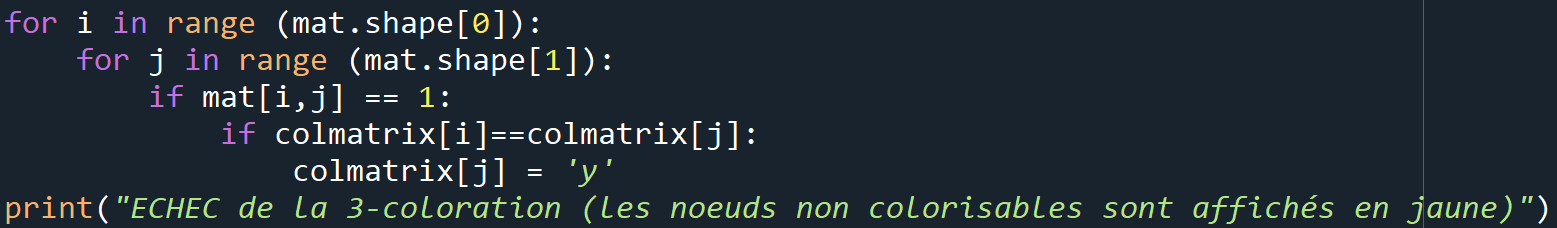
Quand on augmente le nombre de points, la complexité temporelle augmente grandement

**Exercice 3 :**

Idée 1 utilisée pour la 2-coloration :



Idée 2 utilisée pour la 3-coloration :



**Bibliographie**

Documentation Python :

Documentation de recherche et quelques librairies :

<https://www.python-course.eu/graphs_python.php>

<https://pypi.org/project/graph-theory/>

<https://bioinfo-fr.net/python-dessine-moi-un-graphe>

<https://hal.archives-ouvertes.fr/cel-01385792/document>

<https://marcarea.com/weblog/2019/02/17/parcours-de-graphes-en-python>

Documentation de la librairie networkx :

<http://www.cmap.polytechnique.fr/~gobet/DemoPython/ColoringRandomGraphs.html>

<https://stackoverrun.com/fr/q/6864738>

<https://stackoverrun.com/fr/q/6217276>

<https://he-arc.github.io/livre-python/networkx/index.html>

<https://pypi.org/project/networkx/>

<https://networkx.org/>

<https://networkx.org/documentation/stable/news.html#networkx-2-5>

<https://stackoverrun.com/fr/q/2792637>

<https://networkx.org/documentation/stable//reference/functions.html>

<https://stackoverflow.com/questions/27030473/how-to-set-colors-for-nodes-in-networkx>

<https://networkx.org/documentation/stable/reference/generated/networkx.drawing.nx_pylab.draw_networkx_nodes.html>

<https://iut-info.univ-reims.fr/users/coutant/TP3_Coloration.html>

<https://www.youtube.com/watch?v=Czbj_K9LGik&list=PL2VXyKi-KpYsjoY2rx9NiWJyfmmD79H97&index=5&ab_channel=PythonTutorialsforDigitalHumanities>

<https://networkx.org/documentation/stable/tutorial.html#drawing-graphs>

Documentation C# :

Librairie C# :

<https://www.developpez.net/forums/d704503/dotnet/langages/csharp/librairie-faire-graphes/>

<https://archive.codeplex.com/?p=quickgraph>

<https://www.codeproject.com/Articles/5603/QuickGraph-A-100-Csharp-Graph-Library-with-Graphvi>

Code en C++ du voyageur de commerce :

<https://codes-sources.commentcamarche.net/source/view/45437/1138843#browser>

Une explication pour ajouter des librairies en C# :

<https://openclassrooms.com/forum/sujet/c-creer-et-utiliser-une-librairie-visual-studio>

Documentation sur les graphes :

<https://books.google.fr/books?hl=fr&lr=&id=lI17AgAAQBAJ&oi=fnd&pg=PA160&dq=traveling+salesman+problem+algorithm&ots=0MkWfaR07s&sig=dfpHjFZD2Z104jvPhswc2BK86ao&redir_esc=y#v=onepage&q=traveling%20salesman%20problem%20algorithm&f=false>

Autres manières de résoudre le problème du voyageur de commerce :

<https://codes-sources.commentcamarche.net/source/54907-probleme-de-voyageur-de-commerce>

<https://khayyam.developpez.com/articles/algo/voyageur-de-commerce/genetique/>