

Compte Rendu PROJET ALGO

Julien D'aboville - 22107316

2023

Sujet : Scénario 3 : Sujet de concours grandes écoles (Coloriage de graphes)*

Sommaire

Sommaire.....	2
1 Coloriage.....	4
Question 1.....	4
Question 2.....	4
Question 3.....	6
Question 4.....	7
Conclusion :.....	8
2 2-coloriage.....	9
Question 5.....	9
Question 6.....	10
Conclusion :.....	11
3 Algorithmes gloutons.....	12
Question 7.....	12
Question 8.....	13
Question 9.....	14
Question 10.....	14
Question 11.....	14
Question 12.....	15
Conclusion :.....	16
4 Algorithme de Wigderson.....	17
Question 13.....	17
Question 14.....	17
Question 15.....	18
Question 16.....	20
Question 17.....	22
Question 18.....	24
Question 19.....	25
Conclusion :.....	25
Conclusion Générale :.....	26

Introduction :

Dans le domaine de l'informatique théorique et des mathématiques discrètes, le problème du coloriage de graphes se distingue comme un sujet d'étude à la fois complexe et intrigant. Ce projet, basé sur un scénario de concours pour grandes écoles, explore divers aspects du coloriage de graphes, un domaine qui a des applications pratiques significatives, notamment en théorie des réseaux, en optimisation, et dans des domaines aussi variés que la planification de ressources ou la théorie des codes.

Le cœur de ce projet repose sur l'analyse de plusieurs algorithmes et méthodologies pour le coloriage de graphes, allant des concepts de base à des techniques plus avancées comme les algorithmes gloutons et l'algorithme de Wigderson. Chaque section de ce rapport traite d'un aspect spécifique du coloriage de graphes, en s'efforçant non seulement de résoudre des problèmes pratiques mais aussi d'apporter une compréhension théorique plus profonde des algorithmes et de leurs complexités.

En annexe de ce compte rendu, vous trouverez des fichiers Python contenant les codes sources des algorithmes discutés.

1 Coloriage

Question 1

Pour qu'un graphe soit colorié, il faut que pour chaque sommet du graphe, les sommets adjacents n'aient pas la même couleur.

C'est le cas seulement pour le second graphe.

Question 2

Le nombre chromatique pour le graphe de Petersen est de 3.

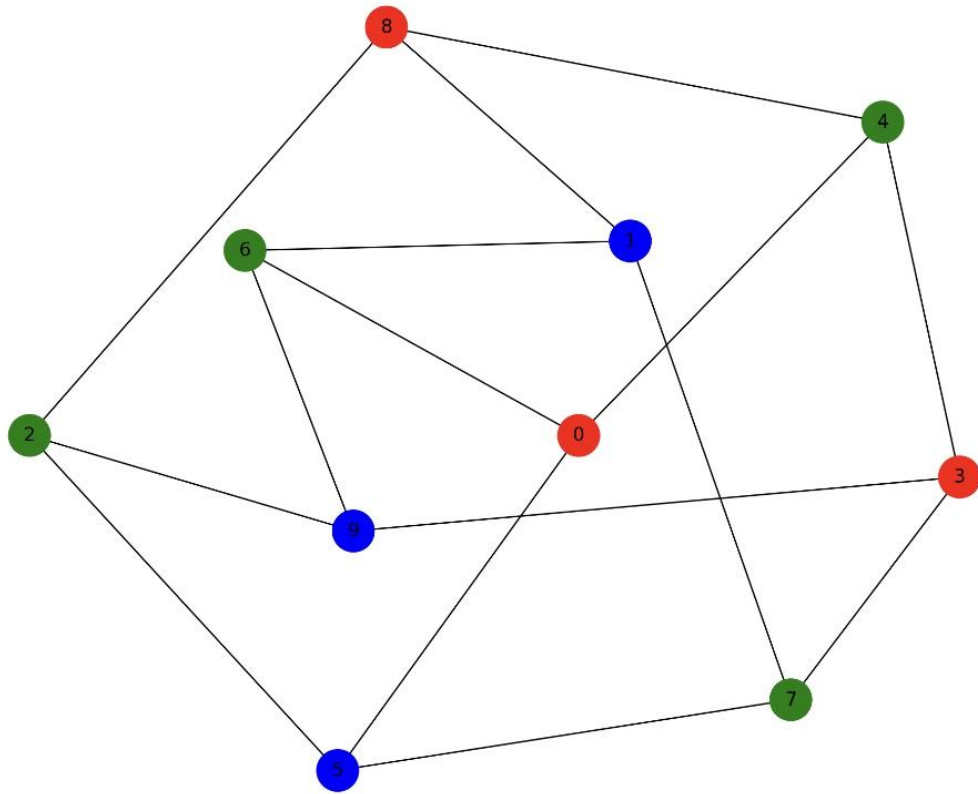
Un exemple de coloriage pourrait être:

Colorier en bleu les sommets 5, 1, 9

Colorier en vert les sommets 2, 4, 6, 7

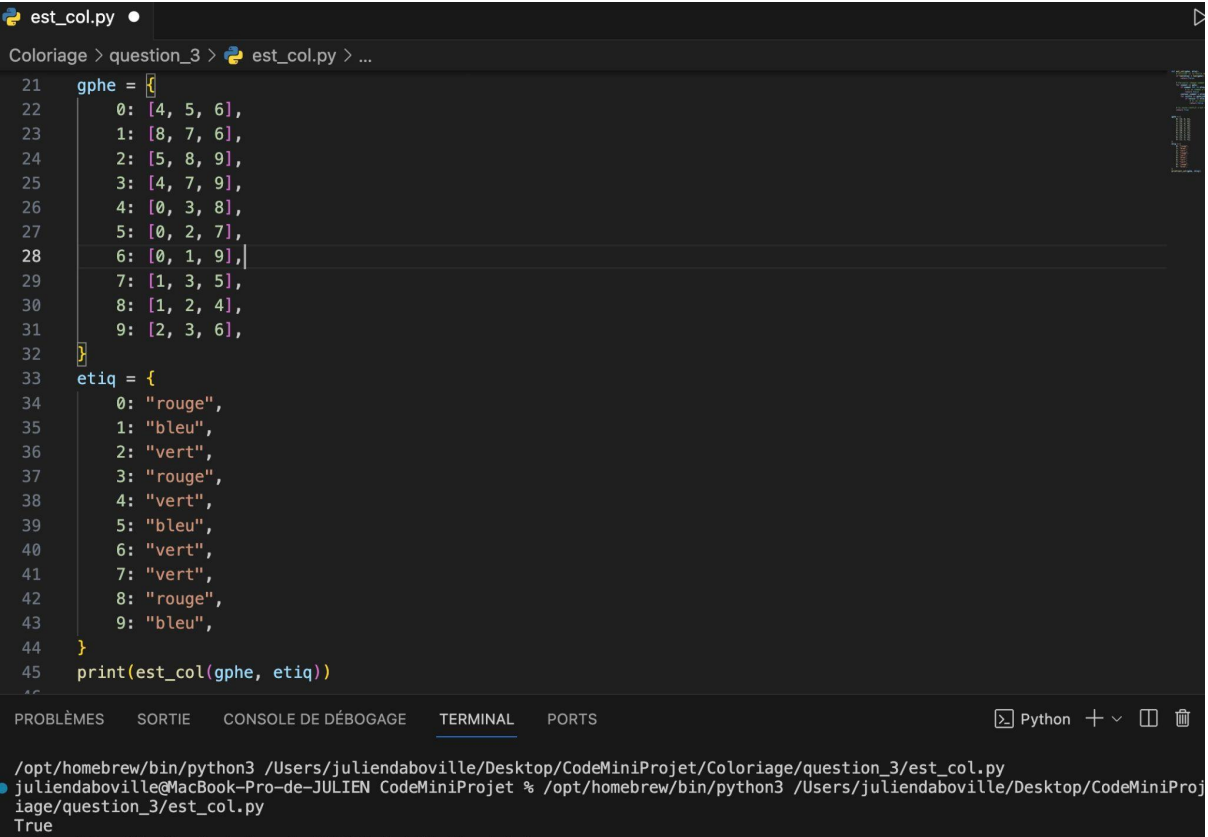
Colorier en rouge les sommets 0, 3, 8

Représentation graphique correspondante :



Question 3

Résultat de la fonction `est_col(graphe,etiquette)`:



```
est_col.py
Coloriage > question_3 > est_col.py > ...

21  gphe = {
22      0: [4, 5, 6],
23      1: [8, 7, 6],
24      2: [5, 8, 9],
25      3: [4, 7, 9],
26      4: [0, 3, 8],
27      5: [0, 2, 7],
28      6: [0, 1, 9],
29      7: [1, 3, 5],
30      8: [1, 2, 4],
31      9: [2, 3, 6],
32  }
33  etiq = {
34      0: "rouge",
35      1: "bleu",
36      2: "vert",
37      3: "rouge",
38      4: "vert",
39      5: "bleu",
40      6: "vert",
41      7: "vert",
42      8: "rouge",
43      9: "bleu",
44  }
45  print(est_col(gphe, etiq))

PROBLÈMES  SORTIE  CONSOLE DE DÉBOGAGE  TERMINAL  PORTS
Python + - []

/opt/homebrew/bin/python3 /Users/juliendaboville/Desktop/CodeMiniProjet/Coloriage/question_3/est_col.py
juliendaboville@MacBook-Pro-de-JULIEN CodeMiniProjet % /opt/homebrew/bin/python3 /Users/juliendaboville/Desktop/CodeMiniProjet/question_3/est_col.py
True
```

Conclusion : Le résultat **True** de cette fonction nous confirme bien que le graphe de Petersen est bien coloriable avec ces étiquettes.

Question 4

Exploration de tous les coloriage possibles: Pour trouver le nombre chromatique, il est nécessaire de considérer tous les coloriage possibles du graphe. Si nous avons n sommets et c couleurs, il y a c^n façons différentes de colorier le graphe (chaque sommet ayant c choix de couleur). Même si nous fixons le nombre de couleurs, la croissance de ce nombre en fonction de n est exponentielle.

Vérification de la validité de chaque coloriage : Pour chaque coloriage, nous devons vérifier si deux sommets adjacents sont colorés de la même couleur. Cette vérification a une complexité qui dépend du nombre d'arêtes dans le graphe, mais puisqu'elle doit être effectuée pour chaque coloriage possible, la complexité totale de l'algorithme reste dominée par le nombre de coloriages à vérifier.

Minimisation du nombre de couleurs : Le nombre chromatique est le nombre minimum de couleurs nécessaires. Cela signifie que nous devons non seulement trouver un coloriage valide, mais aussi vérifier qu'il est impossible de colorier le graphe avec moins de couleurs. Cela implique de tester avec 1, 2, 3, n couleurs jusqu'à trouver le minimum valide, ce qui ajoute une autre couche d'opérations exponentielles.

Conclusion :

Le calcul du nombre chromatique d'un graphe, essentiel pour la coloration optimale des graphes, présente une complexité exponentielle par rapport au nombre de sommets, reflétant la difficulté croissante du problème avec l'augmentation de la taille du graphe.

2 2-coloriage

Question 5

La démonstration qu' un graphe est biparti si et seulement s'il est 2-coloriable repose sur deux parties fondamentales :

Partie 1 : Graphe Biparti est 2-Coloriable

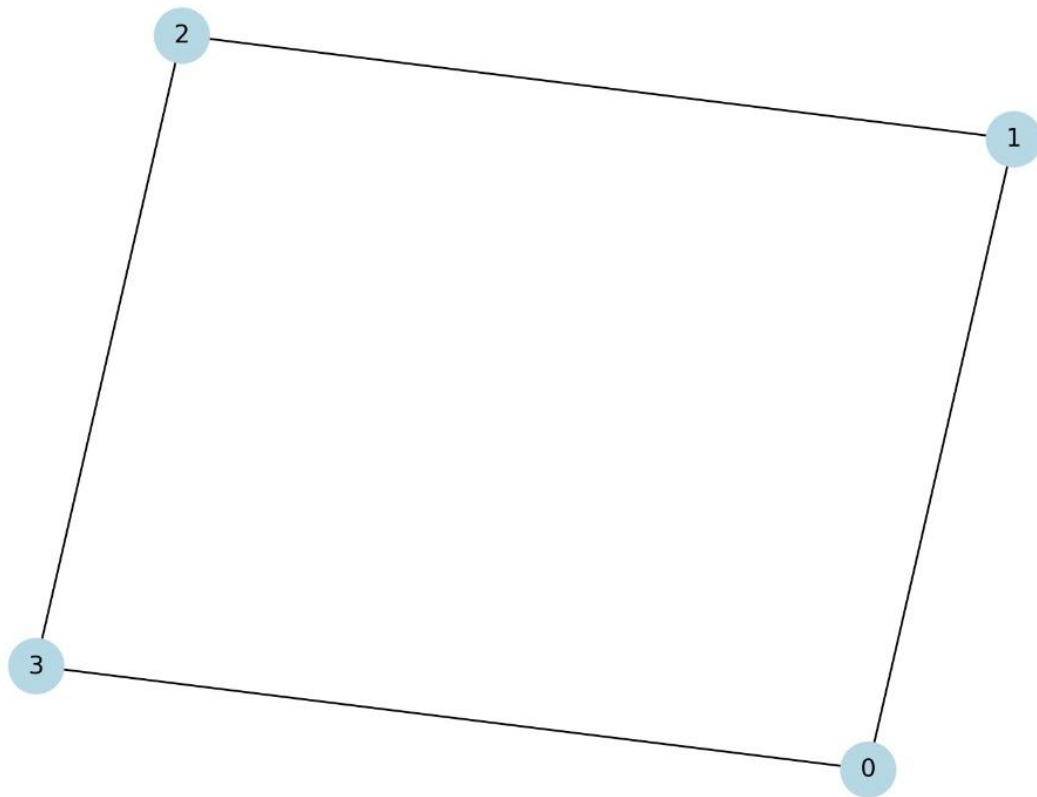
- Un graphe biparti est défini par la division de ses sommets en deux ensembles disjoints T et U, reliés uniquement par des arêtes entre T et U.
- En attribuant la couleur 0 à T et la couleur 1 à U, chaque arête relie des sommets de couleurs différentes, respectant ainsi la 2-colorabilité.
- Par conséquent, un graphe biparti peut être 2-colorié avec seulement deux couleurs.

Partie 2 : Graphe 2-Coloriable est Biparti

- Supposons qu'un graphe soit 2-coloriable, c'est-à-dire que ses sommets puissent être coloriés avec deux couleurs, sans sommets adjacents partageant la même couleur.
- En formant deux ensembles, T (couleur 0) et U (couleur 1), aucun sommet dans T ou U n'est adjacent à un autre de la même couleur.
- Chaque arête relie ainsi un sommet de T à un sommet de U, confirmant que le graphe est biparti.

Question 6

Graphe avec lequel j'ai testé l'algo :



Resultats obtenues de la fonction deux_col(graphe) :

```
20
21
22 # Exemple d'utilisation
23 gphe = {0: [1, 3], 1: [0, 2], 2: [1, 3], 3: [0, 2]}
24 coloriage = deux_col(gphe)
25 print(coloriage)
26
```

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE **TERMINAL** PORTS Python + - [] [] ... ^

juliendaboville@MacBook-Pro-de-JULIEN CodeMiniProjet % /opt/homebrew/bin/python3 /Users/juliendaboville/Desktop/CodeMiniProjet/2-Col
oriage/question_6/deux_col.py
[0, 1, 0, 1]

Conclusion : Le résultat de cette fonction retourne une liste des couleurs des sommets du graphe (0 et 1) ce qui nous confirme que le graphe est bien 2-coloriable

Conclusion :

L'approche du 2-coloriage, adaptée aux graphes bipartis, représente une méthode simplifiée et pratique.

Cet algorithme emploie un parcours en profondeur pour attribuer alternativement deux couleurs différentes aux sommets, tout en maintenant une complexité quadratique par rapport au nombre de sommets.

Cette technique se révèle particulièrement efficace pour les graphes bipartis, bien que son application soit restreinte à cette catégorie spécifique de graphes.

3 Algorithmes gloutons

Question 7

Pour num1 = [1, 3, 4, 0, 2, 6, 5, 9, 8, 7]:

Sommet 0: Couleur 0
Sommet 1: Couleur 0
Sommet 2: Couleur 0
Sommet 3: Couleur 0
Sommet 4: Couleur 1
Sommet 5: Couleur 1
Sommet 6: Couleur 1
Sommet 7: Couleur 2
Sommet 8: Couleur 2
Sommet 9: Couleur 2

Résultat : Nombre de couleurs = 3

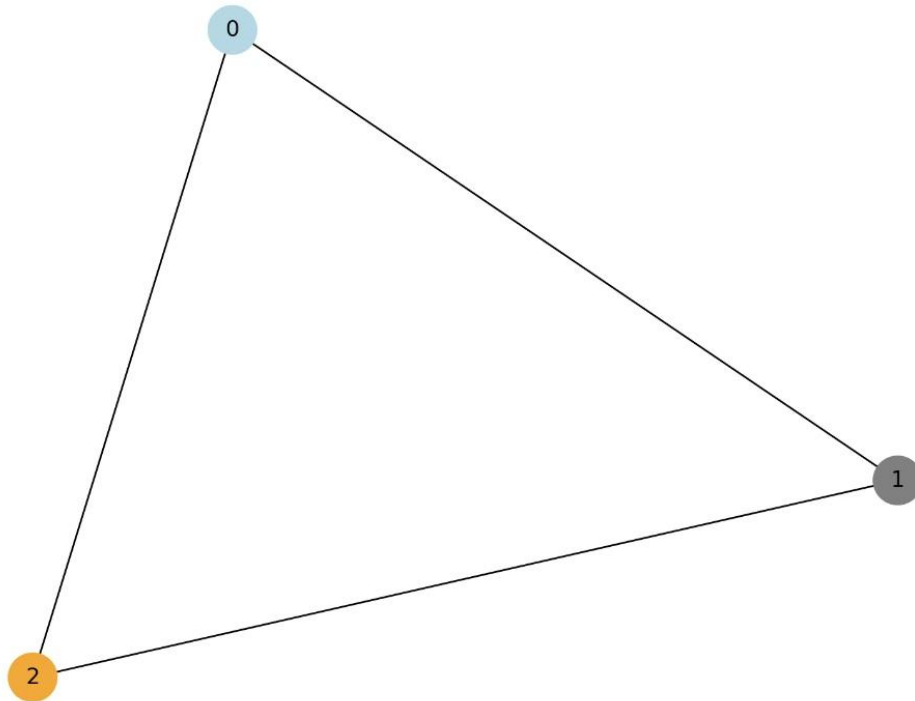
Pour num2 = [0, 7, 2, 5, 4, 6, 8, 1, 3, 9]:

Sommet 0: Couleur 0
Sommet 1: Couleur 3
Sommet 2: Couleur 0
Sommet 3: Couleur 2
Sommet 4: Couleur 1
Sommet 5: Couleur 1
Sommet 6: Couleur 1
Sommet 7: Couleur 0
Sommet 8: Couleur 2
Sommet 9: Couleur 3

Résultat : Nombre de couleurs = 4

Question 8

Graphe avec lequel j'ai testé l'algo :



Resultats obtenus de la fonction min_couleur_possible(graphe,etiquettes,sommet) :

```
19 # Exemple d'utilisation
20 gphe = {0: [1, 2], 1: [0, 2], 2: [0, 1]}
21 etiq = [-1, 0, 1] # Initialement, aucun sommet n'a de couleur
22 s = 0 # On veut trouver la plus petite couleur pour le sommet 0
23 print(min_couleur_possible(gphe, etiq, s))
24
```

Ports

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL PORTS

```
juliendaboville@MacBook-Pro-de-JULIEN question_8 % ls
min_couleur_possible.py
juliendaboville@MacBook-Pro-de-JULIEN question_8 % python3 min_couleur_possible.py
2
juliendaboville@MacBook-Pro-de-JULIEN question_8 %
```

Conclusion : Le test montre bien son succès, sachant que tous les sommets sont reliés (3 sommets) et que la première couleur commence à 0.

Comme le sommet 0 n'est pas colorié , la prochaine couleur (minimale) est donc bien de 2.

Question 9

On retrouve bien les résultats précédents.

```
juliendaboville@MBP-de-JULIEN question_9 % python3 glouton.py
coloriage glouton de [1, 3, 4, 0, 2, 6, 5, 9, 8, 7] : [0, 0, 0, 0, 1, 1, 1, 2, 2, 2]
coloriage glouton de [0, 7, 2, 5, 4, 6, 8, 1, 3, 9] : [0, 3, 0, 2, 1, 1, 1, 0, 2, 3]
juliendaboville@MBP-de-JULIEN question_9 %
```

Question 10

Construction d'un coloriage :

- L'algorithme parcourt tous les sommets du graphe dans un ordre spécifique, sans importance.
- Pour chaque sommet, l'algorithme choisit la plus petite couleur non utilisée par ses voisins déjà colorés.
- Cette approche garantit que chaque sommet sera coloré à la fin de l'exécution de l'algorithme, évitant ainsi la possibilité qu'un sommet reste sans couleur.

Utilisation au plus $d + 1$ couleurs :

- Supposons qu'un sommet v ait au maximum d voisins, où d est le degré maximal du graphe.
- Lorsque nous colorons le sommet v , chacun de ses voisins peut avoir une couleur différente. Dans le pire des cas, chaque voisin a une couleur unique.
- Il y a d voisins, ce qui signifie potentiellement d couleurs différentes parmi ces voisins.
- Pour colorer v , nous aurons besoin d'une couleur qui n'est pas utilisée par ses voisins. Même dans le pire des cas, il y aura toujours au moins une couleur disponible, différente des d couleurs utilisées par les voisins.
- Ainsi, nous n'aurons besoin que de $d + 1$ couleurs au maximum pour garantir que v et tous les autres sommets du graphe peuvent être colorés de manière à ce que deux sommets adjacents n'aient jamais la même couleur.

Question 11

Considérons un coloriage L d'un graphe G , où les sommets sont numérotés de manière à ce que les couleurs soient attribuées dans l'ordre croissant des numéros. Étant donné que deux sommets de même couleur ne peuvent pas être adjacents, lorsque nous utilisons la fonction

min_couleur_possibile pour un sommet s , seuls les sommets voisins de s ayant une couleur strictement inférieure à celle de s , c'est-à-dire $L(s)$, seront pris en compte.

Cela signifie que la couleur choisie, notée $L'(s)$, ne peut pas être strictement supérieure à $L(s)$: nous avons toujours $L'(s) \leq L(s)$ pour tout sommet s .

Si nous commençons avec un coloriage optimal pour établir la numérotation des sommets, cela nous conduit à un nouveau coloriage dont la couleur maximale est bornée par celle du coloriage optimal initial. En conséquence, ce nouveau coloriage sera également optimal.

Question 12

Pour le tri dans `tri_degre`, j'ai utilisé l'algorithme de tri intégré de Python (`sorted`). Cet algorithme est basé sur Timsort, un algorithme de tri hybride qui combine le meilleur du tri par insertion et du tri fusion. Timsort est très efficace pour les données déjà partiellement triées et a une complexité moyenne en $O(n \log n)$, où n est le nombre de sommets.

Resultats obtenus de la fonction `welsh_powell(graphe)` :

```
51
52
53 # Exemple de graphe
54 graphe = {
55     0: [4, 5, 6],
56     1: [8, 7, 6],
57     2: [5, 8, 9],
58     3: [4, 7, 9],
59     4: [0, 3, 8],
60     5: [0, 2, 7],
61     6: [0, 1, 9],
62     7: [1, 3, 5],
63     8: [1, 2, 4],
64     9: [2, 3, 6],
65 }
66
67 # Tester l'algorithme de Welsh-Powell
68 coloration_welsh_powell = welsh_powell(graphe)
69 print(coloration_welsh_powell)
70
```

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE **TERMINAL** PORTS Python + - □ □ ... ^

```
juliendaboville@MacBook-Pro-de-JULIEN CodeMiniProjet % /opt/homebrew/bin/python3 /Users/juliendaboville/Desktop/CodeMiniProjet/AlgorithmsGloutons/question_12/welsh_powell.py
{0: 0, 1: 0, 2: 0, 3: 0, 4: 1, 5: 1, 6: 1, 7: 2, 8: 2, 9: 2}
```

Conclusion: Le test effectué sur le graphe de Petersen retrouve également 3 couleurs.

Conclusion :

En conclusion, les algorithmes gloutons pour la coloration de graphes, bien que ne garantissant pas toujours le résultat optimal, offrent une solution rapide et suffisamment efficace pour de nombreuses applications.

Leur performance dépend de l'ordre de numérotation des sommets, et des optimisations comme celle de Welsh-Powell qui peuvent améliorer significativement l'efficacité en allant jusqu'à $O(n \log n)$.

4 Algorithme de Wigderson

Question 13

1. Définition de $k+1$ -coloriable : Un graphe est dit $k+1$ -coloriable s'il est possible de colorier tous ses sommets avec au plus $k+1$ couleurs de telle manière que deux sommets adjacents aient toujours des couleurs différentes.

2. Considérons un Sommet Spécifique s dans G : Supposons que G soit colorié avec $k+1$ couleurs. Le sommet s aura une des $k+1$ couleurs, disons la couleur C .

3. Sous-graphe Induit par $V(s)$: Le sous-graphe induit par $V(s)$, qui est l'ensemble des voisins de s , exclut le sommet s lui-même.

4. Coloriage de $V(s)$ sans s : Puisque s est exclu du sous-graphe induit par $V(s)$, la couleur C n'est plus utilisée dans ce sous-graphe.

5. k -Coloriable : Le sous-graphe induit par $V(s)$ peut donc être colorié avec les k couleurs restantes. Puisque G était $k+1$ -coloriable et que s avait une couleur unique parmi $k+1$ couleurs, en enlevant s , les voisins de s peuvent être coloriés avec au maximum k couleurs différentes.

6. Conclusion : Ainsi, le sous-graphe induit par $V(s)$ est k -coloriable, car il peut être colorié avec les k couleurs restantes une fois que la couleur de s a été retirée de la palette.

Question 14

L'algorithme de Wigderson garantit le coloriage d'un graphe 3-coloriable en utilisant $O(\sqrt{n})$ couleurs, où n est le nombre de sommets du graphe. Il procède en deux étapes principales :

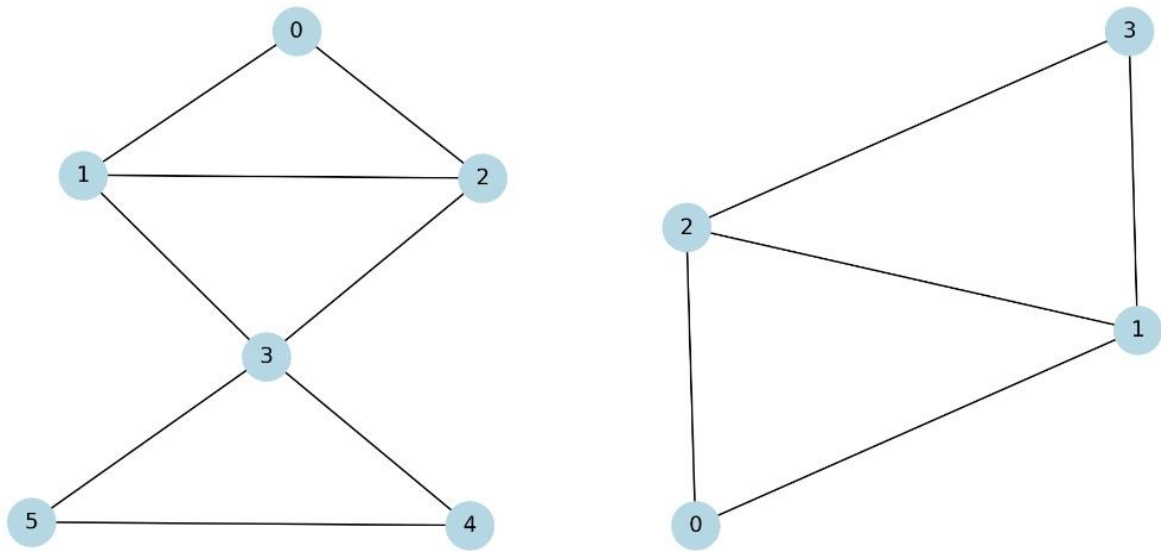
1. Coloriage local : Chaque sommet avec au moins \sqrt{n} voisins non coloriés est le point central pour un 2-coloriage des voisins non coloriés, employant deux nouvelles couleurs. Cela ne peut se produire plus de \sqrt{n} fois, car à chaque fois, au moins \sqrt{n} sommets sont coloriés, contribuant ainsi au coloriage progressif du graphe.

2. Coloriage glouton : Une fois que tous les sommets restants ont moins de \sqrt{n} voisins non coloriés, l'algorithme glouton est utilisé pour colorier les sommets restants. Puisque ces sommets forment un sous-graphe avec un degré maximal inférieur à \sqrt{n} , le glouton utilise au plus $\sqrt{n} + 1$ couleurs.

En combinant les couleurs utilisées dans les deux étapes, l'algorithme reste dans la limite de $O(\sqrt{n})$ couleurs. Cette efficacité est assurée par la réduction systématique du nombre de sommets non coloriés et le contrôle du nombre de couleurs utilisées à chaque étape.

Question 15

Graphe avec lequel j'ai testé l'algo ainsi que son sous-graphe obtenu :



Resultats obtenus de la fonction sous_graphe(graphe,indicesSommetSousGraphe) :

```
18     # Exemple d'utilisation
19     gphe = [
20         [False, True, True, False, False, False],
21         [True, False, True, True, False, False],
22         [True, True, False, True, False, False],
23         [False, True, True, False, True, True],
24         [False, False, False, True, False, True],
25         [False, False, False, True, True, False],
26     ]
27
28     sg = [0, 1, 2, 3] # Les indices des sommets du sous-graphe
29     sous_gphe = sous_graphe(gphe, sg)
30
31     # Affichage du sous-graphe
32     for ligne in sous_gphe:
33         print(ligne)
34     pass
35
```

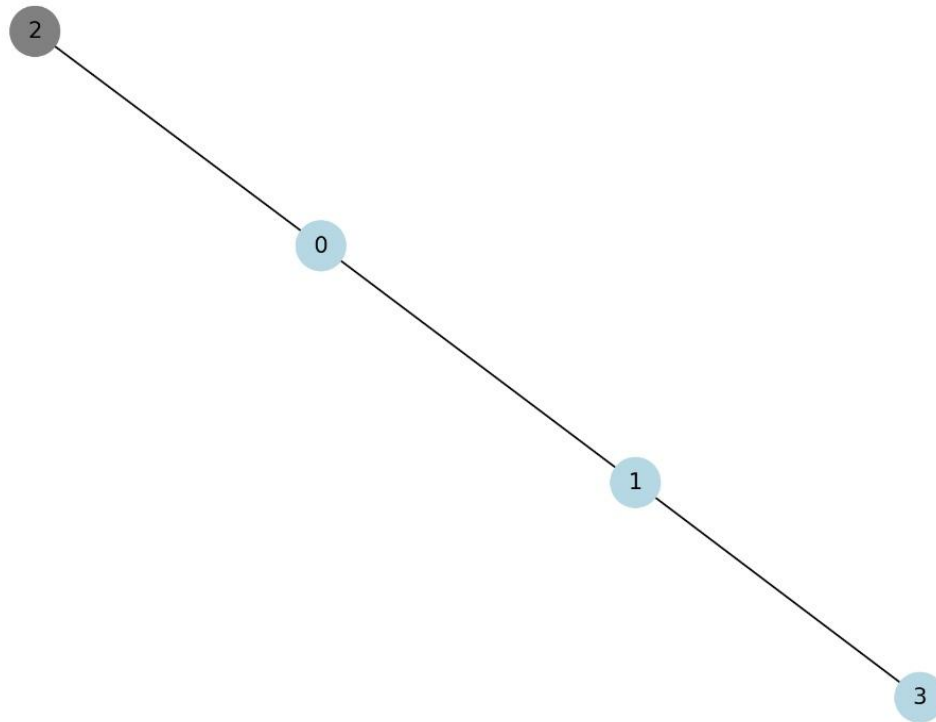
PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL PORTS

```
juliendaboville@MacBook-Pro-de-JULIEN question_15 % python3 sous_graphe.py
[False, True, True, False]
[True, False, True, True]
[True, True, False, True]
[False, True, True, False]
```

Conclusion: Le test effectué sur le graphe renvoie bien le sous-graphe correspondant.

Question 16

Graphe colorié avec lequel j'ai testé l'algo



Resultats obtenus de la fonction voisins_non_colories(graphe,etiquettes,sommet) et degre_non_colories(graphe,etiquettes,sommet) :

```
13     gphe = [  
14         [False, True, True, False], # Matrice d'adjacence pour un graphe  
15         [True, False, False, True],  
16         [True, False, False, False],  
17         [False, True, False, False],  
18     ]  
19     etiq = [-1, -1, 0, -1] # -1 indique qu'un sommet n'est pas encore colorié  
20  
21     # Liste des voisins non coloriés du sommet 1  
22     print(voisins_non_colories(gphe, etiq, 1))  
23  
24     # Degré des voisins non coloriés du sommet 1  
25     print(degre_non_colories(gphe, etiq, 1))  
26  
27     pass  
28
```

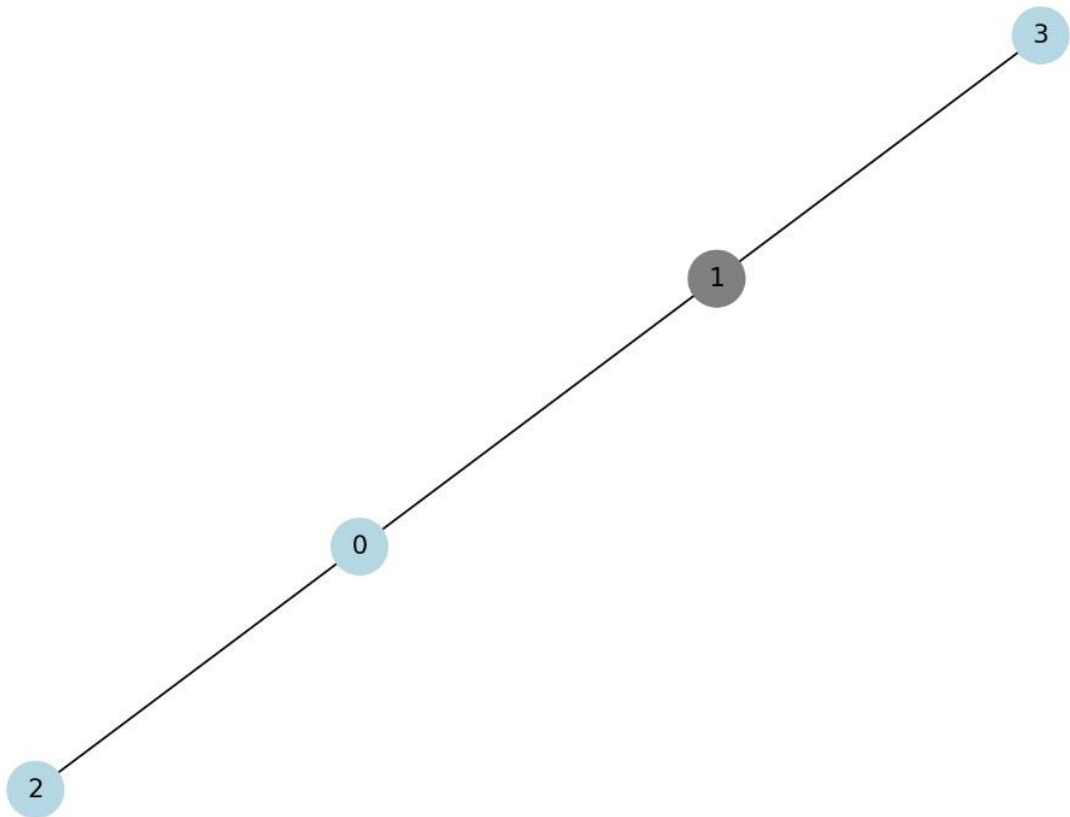
PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL PORTS

```
• juliendaboville@MacBook-Pro-de-JULIEN question_16 % python3 voisins_non_colories.py  
[0, 3]  
2
```

Conclusion : Le test montre bien la cohérence avec la représentation graphique c'est-à-dire que le sommet 1 possède 2 voisins non coloriés qui sont 0 et 3.

Question 17

Graphe colorié avec lequel j'ai testé l'algo



Resultats obtenus de la fonction non_colories(graphe,etiquettes) :

```
6 # Exemple d'utilisation
7 gphe = [
8     [False, True, True, False], # Matrice d'adjacence pour un graphe
9     [True, False, False, True],
10    [True, False, False, False],
11    [False, True, False, False],
12 ]
13 etiq = [-1, 0, -1, -1] # -1 indique qu'un sommet n'est pas encore colorié
14
15 # Liste des sommets non coloriés
16 print(non_colories(gphe, etiq))
17
```

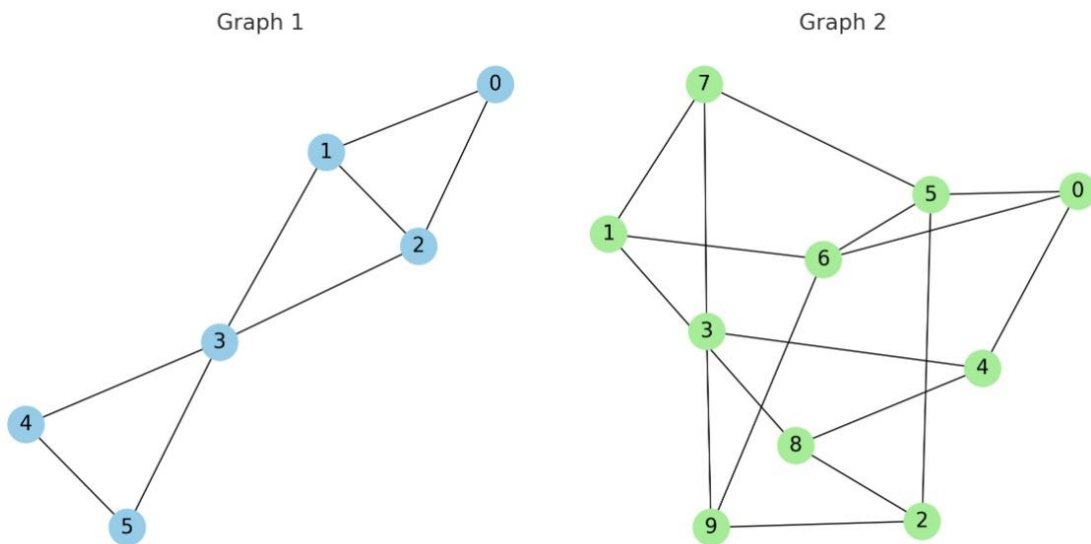
PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL PORTS

```
juliendaboville@MacBook-Pro-de-JULIEN question_17 % ls
non_colories.py
juliendaboville@MacBook-Pro-de-JULIEN question_17 % python3 non_colories.py
[0, 2, 3]
juliendaboville@MacBook-Pro-de-JULIEN question_17 %
```

Conclusion : Le test montre bien la coherence avec la representation graphique, cest -à-dire que le graphe possède les sommets 0, 2 et 3 non coloriés.

Question 18

Graphes sur lesquels j'ai testé l'algo :



L'algo retourne les résultats suivants :

```
juliendaboville@MBP-de-JULIEN question_18 % python3 wigderson.py  
coloration graphe1 : [0, 2, 1, 0, 1, 2]  
coloration graphe2 : [0, 0, 0, 0, 1, 1, 2, 2, 2, 1]  
juliendaboville@MBP-de-JULIEN question_18 %
```

Conclusion: L'application de l'algorithme de Wigderson sur deux graphes spécifiques a fourni des résultats concluants. Dans chaque cas, l'algorithme a réussi à identifier et à colorier les graphes comme étant 3-coloriables (couleurs 0, 1 et 2). Ce résultat démontre non seulement l'efficacité de l'algorithme pour identifier la colorabilité dans un cadre contraint, mais aussi sa capacité à fournir un coloriage valide dans les limites de complexité attendues.

Question 19

L'algorithme de Wigderson, initialement conçu pour des graphes 3-coloriables, peut être étendu à des graphes avec un nombre chromatique supérieur à 3 en adaptant sa stratégie de coloriage local. Cette extension nécessiterait une utilisation plus flexible du nombre de couleurs dans le coloriage initial, ajustée en fonction du nombre chromatique connu du graphe.

Par exemple, au lieu de se limiter à un 2-coloriage, l'algorithme pourrait appliquer un k -coloriage local pour les sous-graphes, où k dépend du nombre chromatique spécifique. De plus, l'intégration d'une approche hybride, combinant différentes stratégies de coloriage en fonction de la densité des connexions dans différentes parties du graphe, pourrait améliorer l'efficacité de l'algorithme pour ces graphes plus complexes.

Conclusion :

Pour les graphe 3-coloriable, l'algorithme de Wigderson implémenté pour la coloration de graphes a donc démontré une performance qui respecte la propriété clé de cet algorithme : l'utilisation d'un nombre de couleurs qui est en $O(\sqrt{n})$, où n est le nombre de sommets du graphe.

1. Pour le graphe 1 (6 sommets), le coloriage résultant utilise 3 couleurs (0, 1, 2). Puisque $\sqrt{6}$ est approximativement 2.45, l'utilisation de 3 couleurs est en accord avec l'ordre de grandeur $O(\sqrt{n})$.

2. Pour graphe 2 (10 sommets), le coloriage utilise également 3 couleurs. Ici, $\sqrt{10}$ est environ 3.16, ce qui signifie que l'utilisation de 3 couleurs reste dans les limites de $O(\sqrt{n})$.

Ce constat montre clairement que l'algorithme de Wigderson est efficace pour la coloration de graphes, utilisant un nombre de couleurs qui ne dépasse pas significativement la racine carrée du nombre de sommets.

Conclusion Générale :

Ce projet a exploré le domaine du coloriage de graphes à travers une série de problématiques et d'algorithmes: de l'analyse de la colorabilité 2-coloriage et bipartie aux techniques plus sophistiquées comme les algorithmes gloutons et l'algorithme de Wigderson.

Les découvertes réalisées dans ce projet soulignent l'importance de l'approche algorithmique dans la résolution de problèmes de coloriage de graphes. Bien que certains algorithmes offrent des solutions optimales dans des cas spécifiques, d'autres, tels que les algorithmes gloutons, fournissent des solutions approximatives qui peuvent être plus pratiques dans certaines situations, malgré leurs limites. L'efficacité de ces méthodes dans des contextes variés montre leur potentiel applicatif dans des domaines tels que la planification, l'optimisation, et la théorie des réseaux.