INFOF-F203

Oudahya Ismaïl - 000479390

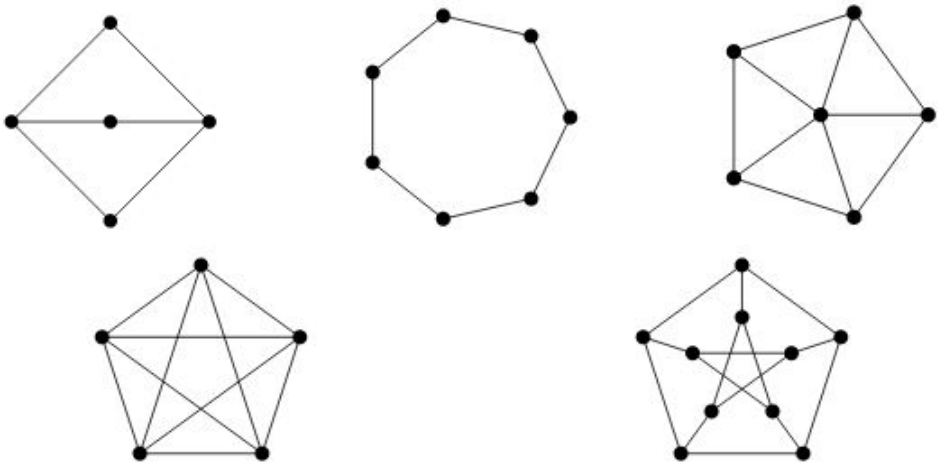
30 Avril 2021

Contents

1 Introduction

Pour ce projet, nous devons implémenter la coloration de graphe. Dans un graphe, deux sommets adjacents, ne sont pas colorés avec une même couleur. Dépendant de l'algorithme utilisé on peut avoir un graphe avec un minimum de nombre chromatique mais cela est un problème NP-COMPLET.

Lors de la coloration d'un graphe, un ordre de couleur est donné dépendamment des sommets adjacents, une coloration ne peut être similaire à une autre si les deux sommets sont reliés par un arrêt. Etant donnée une couleur k, nous devons trouver le minimum nombre chromatique pour un graphe, dans ce projet nous avons implémenté différentes façons d'arriver à une coloration correcte ainsi que développer les problèmes rencontrés pour trouver la bonne façon d'obtenir le nombre chromatique optimal.



Levin, O. (s.d.). Discrete Mathematics: An open introduction.
http://discrete.openmathbooks.org/dmoi2/sec$_c$*oloring.html*

2 Partie 1

2.1 Présentation de l'algorithme

Algorithm 1 Compare(int k)

```
j=0 for dernier sommet do
    if matriceAdjacence[k][j] = 1 ∧ colorlist[j] = colorlist[k] then
        return false
    end
end
return true
```

Cette méthode nous permet de vérifier si le sommet k est adjacent, et de comparer les couleurs assignées à tout autre sommet adjacent à k

Algorithm 2 findColoring(int k)

```
if k = dernier sommet then
    count ← 0
    i=0 for dernier sommet do
        if colorlist[i] > count then
            count ← colorlist[i]
        end
    end
    if count ≤ couleurmax then
        colorcomparing ← min(colorlist)
    end
    return
end
i=0 for couleur max do
    colorlist[k] ← i
    if Compare(k) then
        findColoring(k+1)
    end
end
colorlist[k] ← 0
```

Pour cet algorithme , on dispose de deux listes qui gèrent les couleurs, la liste colorcomparing se met à jour lorsqu'on trouve une liste de couleur dont le nombre chromatique est le minimum. Nous disposons d'une condition de fin qui stipule, que si on arrive au dernier graphe c'est qu'on a bien pu avoir tous les graphes colorés.

La deuxième partie du code, après la boucle allant de $0 \leftarrow couleurmax$ est la partie récursive de l'algorithme.

Le parcours en backtracking parcourt tous les cas possibles pour avoir une coloration minimale et il a donc une complexité exponentielle. Si on trouve la bonne couleur, on passe au prochain sommet, sinon on change de couleur et cela se repète pour chaque sommet avec différentes couleurs jusqu'à ce que l'on trouve le nombre chromatique minimal.

2.2 Quelque exemple d'exécution

Pour cette partie nous montrerons différents types de graphe et leur exécution

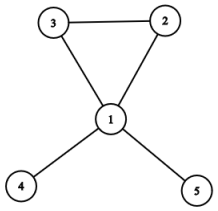


Figure 1: Graphe 1

$Sommet : 1 \rightarrow couleur3$ $Sommet : 2 \rightarrow couleur2$ $Sommet : 3 \rightarrow couleur1$ $Sommet : 4 \rightarrow couleur2$
$Sommet : 5 \rightarrow couleur2$

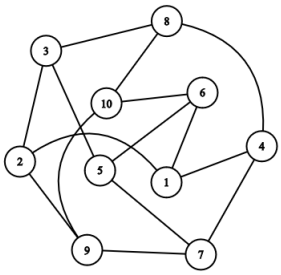


Figure 2: Graphe 2

$Sommet : 1 \rightarrow couleur3$ $Sommet : 2 \rightarrow couleur2$ $Sommet : 3 \rightarrow couleur3$ $Sommet : 4 \rightarrow couleur2$
$Sommet : 5 \rightarrow couleur2$ $Sommet : 6 \rightarrow couleur1$ $Sommet : 7 \rightarrow couleur3$ $Sommet : 8 \rightarrow couleur1$
$Sommet : 9 \rightarrow couleur1$ $Sommet : 10 \rightarrow couleur3$

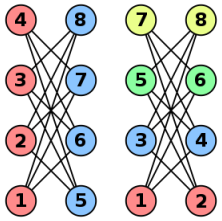


Figure 3: Graphe couronne
Graphe couronne. (2021, janvier 14). Dans Wikipedia, l'encyclopédie libre. Repéré le 15 avril 2020 à https://fr.wikipedia.org/wiki/Graphe$_c$*ouronne*

Définition graphe couronne : En théorie des graphes, une branche des mathématiques, un graphe couronne à 2 n sommet est un graphe non orienté comportant deux jeux de sommets u_i et v_i reliés par une arête de u_i à v_j $\forall$ $i \neq$j.
(*Graphecouronne*, 2021)

Pour le graphe de droite et gauche :
$Sommet : 1 \rightarrow couleur2$ $Sommet : 2 \rightarrow couleur1$ $Sommet : 3 \rightarrow couleur2$ $Sommet : 4 \rightarrow couleur1$
$Sommet : 5 \rightarrow couleur2$ $Sommet : 6 \rightarrow couleur1$ $Sommet : 7 \rightarrow couleur2$ $Sommet : 8 \rightarrow couleur1$

On remarque bien qu'avec le backtracking on retrouve la même coloration pour les deux différents graphes couronne, contrairement à un algorithme glouton qui suit le principe de ne pas trouver toutes les combinaisons possibles et qui n'est pas très efficace. Elle est simple à implémenter mais ne permet pas de trouver le nombre chromatique optimal.

2.3 Question 1

On utilise l'algorithme de parcours en backtracking pour trouver le nombre chromatique optimal, on retrouve la complexité de $O(n.k^n)$

où n représente le nombre de sommet et k représente le nombre de couleurs.

Dans la méthode findColoring ,

1. boucle dans la condition de fin qui est en $O(n)$
2. seconde boucle dans la deuxième partie du code , qui parcourt les couleurs $O(k)$
3. nous avons une condition qui fait appel à Compare(k) $O(n)$
4. enfin, nous avons l'appel récursif qui est en $O(k^n)$

Pour ce qui est de la complexité $O(k^n)$ de l'appel récursif, on utilise le concept mathématique de l'arrangement avec répétitions. La formule est donnée par $B_n^k = n^k$ Démonstration : On doit effectuer k choix, et pour chacun de ceux-ci, on a n possibilités. Le nombre de façons d'effectuer ces choix est donc simplement n...n = n^k.

(Radu,2014)

Grâce à l'application de la formule suivante :

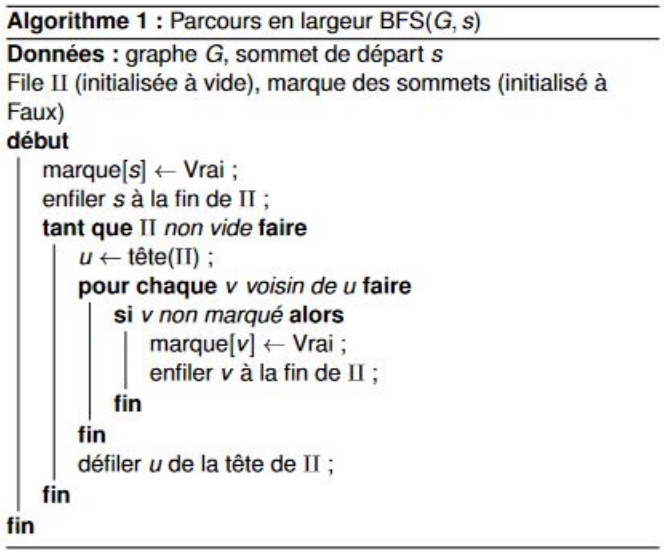
$f_1 = O(g_1)$ and $f_2 = O(g_2)$ $\rightarrow$f$_1$ + $f_2 = O(max(g_1, g_2))$

(Big O Notation,2021)

On obtient $O(n).O(n^k)$ ce qui donne $\rightarrow$O(n.k^n)

2.4 Question 2

Pour résoudre un graphe simple non dirigé, on peut utiliser la vue d'un graphe bipartite, un algorithme vérifie si le graphe que l'on reçoit est un graphe bipartite. Si c'est un graphe bipartite alors on peut avoir un nombre chromatique en k = 2.Si ce n'est pas un graphe bipartite alors la résolution chromatique en k = 2 n'est pas valable.Pour vérifier si le graphe est bipartite nous devons utiliser le parcours BFS soit l'algorithme de parcours en largeur.



Algorithme 1 : Parcours en largeur BFS(G, s)

Données : graphe G, sommet de départ s
File Π (initialisée à vide), marque des sommets (initialisé à Faux)
début
  marque[s] ← Vrai ;
  enfiler s à la fin de Π ;
  tant que Π *non vide* **faire**
    u ← tête(Π) ;
    pour chaque v *voisin de* u **faire**
      si v *non marqué* **alors**
        marque[v] ← Vrai ;
        enfiler v à la fin de Π ;
      fin
    fin
    défiler u de la tête de Π ;
  fin
fin

Figure 4: parcours BFS
Kardos, F. (s.d.). Algorithmique des graphes. https://www.labri.fr/perso/fkardos/cours3.pdf

Pour chaque sommet découvert , il faut visiter les enfants. Lorsqu'on finit de visiter les enfants, le prochain enfant devient le parent et on visite ainsi les enfants de ce parent et ainsi de suite.
La complexité est constante et est donnée en ordre polynomial de $O(n + m)$

Voici la méthode de résolution :

1. On suppose que l'on dispose de deux couleurs, bleu et rouge
2. On colorie le premier sommet en bleu
3. On fait le parcours en BFS , lorsqu'on découvre un sommet, on le colorie de la couleur opposée au parent et on le marque
4. Si le sommet a déjà été marqué, on vérifie si sa couleur correspond à celle du parent, si oui alors le graphe n'est pas bipartite et n'est donc pas coloriable en k=2

(Sghiouer et AL.,2010)

2.5 Question 3

Puisque le problème est un NP-COMPLET, trouver un algorithme optimal pour k = 3 s'avère compliqué. Il yen a plusieurs mais qui ne sont pas à 100% optimaux , l'algorithme de Wigderson,greedy,Welsh Powel et de DSATUR.

Nous allons plutôt nous focaliser sur l'algorithme de DSATUR qui utilise une implémentation en algorithme glouton.

L'ordre de chaque sommet est choisi dynamiquement, c'est-à-dire qu'il se base sur le nombre de couleurs qui ne peuvent être utilisé lors des chevauchements entre les sommets précédents. Les sommets ayant le moins de couleur disponible sont colorés en premier pour éviter tout conflit.

Il a donc une coloration optimale dans plus de 90% des cas. Mais il y a un graphe pour lequel une solution n'est pas exacte.

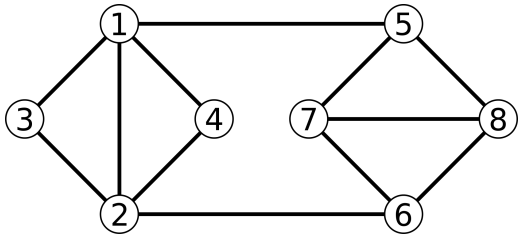


Figure 5: Graphe 3

DSATUR.(2019, septembre 12). Dans Wikipedia, l'encyclopédie libre. Repéré le 20 avril 2021 à https://fr.wikipedia.org/wiki/DSATUR

Pour ce graphe , on trouve un nombre chromatique de 4, alors que lorsqu'on lance ce graphe avec l'algorithme en backtracking, on a une solution optimale de 3 couleurs. On perçoit que l'algorithme DSATUR ne trouve pas le nombre chromatique optimal. (Furini et AL.,s.d)

Avec l'algorithme en backtracking :

$Sommet : 1 \rightarrow couleur3$ $Sommet : 2 \rightarrow couleur2$ $Sommet : 3 \rightarrow couleur1$ $Sommet : 4 \rightarrow couleur1$
$Sommet : 5 \rightarrow couleur2$ $Sommet : 6 \rightarrow couleur2$ $Sommet : 7 \rightarrow couleur3$ $Sommet : 8 \rightarrow couleur1$

Pour conclure , l'algorithme de DSATUR est en complexité polynomiale de $O(n^2)$

3 Partie 2

3.1 Présentation de l'algorithme

Algorithm 3 addMatrixAdj

```
i ← 1 for sommet-1 do
    j ← i + 1 for sommet do
        if matrix[j][1] ≤ matrix[i][0] ∨matrix[i][1] ≤ matrix[j][0] then
            pass
        end
        else
            addMatrix(i,j)
        end
    end
end
```

On trie les tâches que l'on reçoit, ensuite on effectue un double parcours. Pour chaque tâche i, on vérifie les tâches i+1 →dernière tâche.

La condition : $\text{Fin}_j \leq Début_i$ *ou* $Fin_i \leq Début_j$

Si nous supposons que la condition est vraie, alors la tâche n'a aucun conflit avec la tâche suivante.Par contre, si la condition est fausse, c'est qu'il ya bien un conflit avec la tâche.

Pour repérer les tâches qui sont en conflit, on les rajoute dans une matrice d'adjacence.

Algorithm 4 findColoring ord

```
result[firstelement] ← 1
i ← 1 for sommet do
    color ← 1
    j ← 1 for sommet do
        if matrixAdjacence[i][j] = 1 then
            if result[j] ≠ −1 then
                res[j] ← result[j]
            end
        end
    end
    while comparing res and color do
        color ← color + 1
        if color > colorMax then
            break;
        end
    end
    if color > colorMax then
        break;
    end
    result[i] ← color
end
```

On effectue un parcours brute, on donne la première couleur au premier sommet. Si dans la matrice d'adjacence les différents sommets sont en conflit, alors on itère la couleur jusqu'à trouver la bonne couleur,

si la couleur itèrée dépasse le nombre maximal k couleur chromatique, alors on arrête car il n'est pas possible de trouver un sommet plus grand que la coloration k.

3.2 Exemple d'exécution

```
18
1 0 3
2 0 7
3 0.6 1
4 2.5 6
5 3 7
6 3.9 9
7 5 12
8 5.2 10
9 6.9 19
10 8 11
11 9 18
12 9.4 15
13 10 18.5
14 12 16
15 14 17
16 15.3 18
17 16 18.8
18 18 21
```

Figure 6: Suite d'ordonnancement

$tache : 1 \rightarrow machine1$ $tache : 2 \rightarrow machine2$ $tache : 3 \rightarrow machine3$ $tache : 4 \rightarrow machine3$
$tache : 5 \rightarrow machine1$ $tache : 6 \rightarrow machine4$ $tache : 7 \rightarrow machine5$ $tache : 8 \rightarrow machine6$
$tache : 9 \rightarrow machine3$ $tache : 10 \rightarrow machine1$ $tache : 11 \rightarrow machine2$ $tache : 12 \rightarrow machine4$
$tache : 13 \rightarrow machine6$ $tache : 14 \rightarrow machine1$ $tache : 15 \rightarrow machine5$ $tache : 16 \rightarrow machine4$
$tache : 17 \rightarrow machine1$ $tache : 18 \rightarrow machine2$

3.3 Question 1

Dans cet exercice, nous recevons un ensemble de tâches ayant un début et une fin. On ne peut effectuer plusieurs tâches à la fois dans une seule même machine. Ce qu'il faudrait faire c'est assigner une nouvelle machine pour pouvoir gérer une tâche qui occupe une autre machine.

Afin de simuler la situation des tâches dans un graphe, nous devons considérer un sommet pour chaque tâche, et lorsque les tâches se chevauchent dans le temps, une arête relie les deux sommets. Le graphe ainsi construit s'appelle un graphe d'intervalle. Cette stratégie est utilisé pour la résolution de l'algorithme pour la deuxième partie.

Dans la partie du code les tâches qui se chevauchent sont représentées dans une matrice d'adjacence. Dans un graphe d'intervalles , il n'existe pas de sommet avec plus de 3 arêtes.(Yahia Sadi,2011)

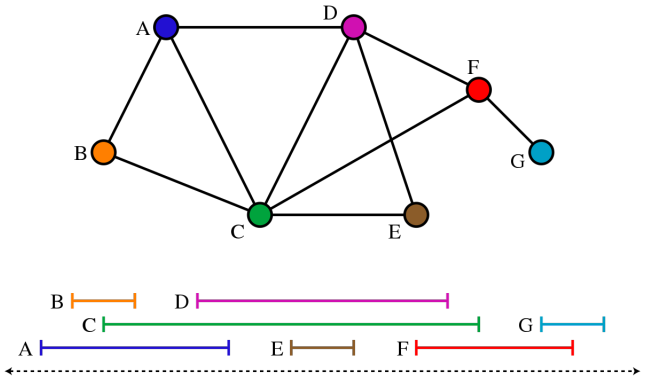


Figure 7: Graphe intervalle
Graphes d'intervalles. (2021, avril 10). Dans Wikipedia, l'encyclopédie libre. Repéré le 15 avril 2020 à https://fr.wikipedia.org/wiki/Graphe$_d$

Ce problème revient donc à colorer les sommets du graphe d'intervalles associé avec la nécessité que deux sommets reliés ne peuvent pas recevoir la même couleur (s'exécuter sur une même machine).

Pour résoudre le problème et avoir un nombre minimum de couleurs.

1. On commence par trier par ordre croissant les sommets c'est-à-dire les tâches par la valeur de début
2. On parcourt ensuite les sommets dans l'ordre de trie
3. On colorie le premier sommet donc on l'assigne à une machine
4. Si le $sommet_i$ peut être introduit dans la machine créée, alors on le rajoute à la machine. Ici on le met à la même couleur
5. Sinon, on le place dans une nouvelle machine, donc une nouvelle couleur est assignée

(Cyril,2011) Puisqu'on a un graphe d'intervalle , et que les tâches sont séquentielles, on peut implémenter un algorithme glouton en temps polynomial. On n'est pas dans l'obligation de chercher tous les parcours possibles pour avoir la bonne coloration. Un parcours en force brute suffit pour avoir une coloration optimale. Or pour la première partie , tout dépend de la couleur assignée $max(k)$
On avait dit que si $k = 2$, alors le problème pouvait être résolu en temps polynomial. Mais si $k > 3$ alors le temps devenait exponentiel. Pour k=3 il y a plusieurs solutions en temps polynomial mais elles ne sont pas toujours optimales.

3.4 Question 2

Pour la résolution de cet exercice, on a utilisé un algorithme de type glouton. On retrouve la complexité de $O(n^2.k)$

Où n représente le nombre de sommet et k représente le nombre de couleur.

Dans la méthode findColoring ord ,

1. boucle passant les différentes tâches qui est en $O(n)$
2. seconde boucle imbriquée à la première boucle similaire à la précédente qui est en $O(n)$
3. un while qui parcourt pour trouver la couleur parfaite pour la tâche qui est aussi imbriqué à la première boucle. Elle est en $O(n.k)$

Nous avons donc les deux parcours en boucle qui sont en $O(n) + O(n.k)$

Grâce à la formule données précédemment dans la partie 1, nous nous retrouvons avec $O(n.k)$. $O(n)$ ce qui donne $\rightarrow$ au pire des cas $O(n^2.k)$

4 Conclusion

Pour conclure, trouver un nombre chromatique optimal revient à avoir une complexité en temps exponentiel, sauf pour k=2 et quelque exceptions que l'on a détaillé.
Les difficultés que j'ai rencontré , étaient de trouver un algorithme en temps polynomial pour k = 3.

5 Source

Big O Notation. (2021, avril 28). Dans Wikipedia, *l'encyclopédie libre*. Repéré le 15 avril 2020 à https://en.wikipedia.org/wiki/Big_O_notation

Yahia, S. & Sadi, B. (2011, octobre 19). *Algorithmes Gloutons Optimaux pour les graphes d'indifférence.* [Mémoire de master en recherche opérationnelle ; Université Mouloud Mammeri]. https://dl.ummto.dz/bitstream/handle/ummto/3272/Yahia%2C%20Souhila.pdf?sequence=1&isAllowed=y

Graphes d'intervalles. (2021, avril 10). Dans Wikipedia, *l'encyclopédie libre*. Repéré le 15 avril 2020 à https://fr.wikipedia.org/wiki/Graphe_d%27intervalles

Graphe couronne. (2021, janvier 14). Dans Wikipedia, *l'encyclopédie libre*. Repéré le 15 avril 2020 à https://fr.wikipedia.org/wiki/Graphe_couronne

Furini, F., Gabriel, V. & Ternier, I.C (s.d). An improved DSATUR-based Branch and Bound for the Vertex Coloring Problem. https://tel.archives-ouvertes.fr/tel-01867961/document

Sghiouer, K., Li, Y., Lucet, C., & Moukrim, A. (2010, Mai 10). *Somme coloration de graphe*. In 8e Conférence Internationale de Mondialisation et Simulation-MOSIM'10. (p. 276). https://hal.archives-ouvertes.fr/hal-00527271/document

Levin, O. (s.d.). *Discrete Mathematics: An open introduction.* http://discrete.openmathbooks.org/dmoi2/sec_coloring.html

Kardos, F. (s.d.). Algorithmique des graphes. https://www.labri.fr/perso/fkardos/cours3.pdf

Radu, N. (2014, décembre 8). Dénombrement. Repéré le 10 avril 2020 à https://www.mathraining.be/chapters/35?type=10

Cyril, B.(2011, janvier 24). Analyse d'intervalles pour l'ordonnancement d'activités. Repéré le 16 avril 2020 à

https://tel.archives-ouvertes.fr/tel-00558925/document