# TP de Théorie des Graphes

## Informations générales

- Section : Ingénierie Génie Logiciel (IGL3)

- Membres du groupe (Nom, Prénom) :

* Chaabane BOUSSADIA (IGL 3)
* Raed BETTAHER (IGL 3)
* Abderrahmen JEDIDI (IGL 3)

- Algorithme traité : Coloration des sommets (Algorithme glouton de Welsh et Powell)

- Langage utilisé : Python

## 2. Présentation de l’algorithme : Pourquoi on a besoin de tel algorithme ?

Soit une collection de n tâches, chacune de durée indivisible d'un jour. Une série E de contraintes spécifie que certaines paires de tâches ne peuvent pas être exécutées simultanément. L'objectif est de déterminer si toutes les tâches peuvent être exécutées dans un intervalle donné de k jours tout en respectant les contraintes, ou de trouver le plus petit intervalle de temps (nombre minimum de jours) nécessaire pour exécuter toutes les tâches. Ce problème peut être modélisé comme un problème de coloration de graphe, où chaque tâche est un sommet, chaque contrainte d'exclusion est une arête, et chaque couleur représente un jour où les tâches associées peuvent être exécutées simultanément.

L'algorithme de Welsh-Powell est une approche gloutonne pour résoudre ce problème de coloration de graphe. Il vise à assigner le plus petit nombre de couleurs (jours) possible tout en garantissant que deux sommets adjacents (tâches incompatibles) n'ont pas la même couleur.

* Fonctionnement général de l’algorithme de Welsh-Powell : soit un graphe G = (V , E) , où V est l’ensemble des tâches , et E contient une arête entre xi et xj si ces taches ne peuvent pas être exécutées simultanément .

1. Tri des sommets : Triez les sommets de G par ordre décroissant de leur degré (nombre de voisins, càd nombre des contraintes associées à chaque tâche).
2. Coloration gloutonne (d’une manière itérative jusqu’à ce que tous les sommets soient coloriés) :
   1. Prenez le premier sommet non coloré dans l'ordre trié et assignez-lui la première couleur disponible (la plus petite couleur non utilisée par ses voisins déjà colorés).
   2. Parcourez les sommets restants dans l'ordre, en assignant à chaque sommet non coloré la plus petite couleur possible qui n'est pas utilisée par ses voisins.

* Complexité des algorithmes :

Le tri des sommets par degré décroissant prend [dans des algorithmes de tri avancé comme Tri Fusion ou le Tim Sort utilisé en Python)

Lors de la coloration des graphes, le meilleur cas est de colorier un graphe complet, sa complexité est égale à . Dans le pire de cas, cette étape prend

La complexité totale est alors évaluée par l’expression

NB : L’algorithme de Welsh-Powell est simple et rapide (par rapport aux autre algorithmes), mais il ne permet pas toujours de donner le nombre chromatique d’un graphe G.

Nombre chromatique : noté par ꭓ(G) est le plus petit nombre des couleurs nécessaires pour colorer les sommets de G de telle sorte que deux sommets adjacentes (reliés par une arête) n’aient pas la même couleur.

## 3. Environnement et implémentation

- Langage de développement : Python (version 3.12) + Jupyter Notebook

- IDE : Visual Studio Code

- bibliothèques utilisés dans ce projet :

* networkx : Bibliothèque pour la création , la manipulation et de l’analyse de graphes et réseaux .
* matplotlib.pyplot : un module de bibliothèque Matplotlib pour créer des visualisations graphiques
* numpy : bibliothèque pour le calcul numérique , offrant des structures de données comme des tableaux multidimensionnels et des fonctions mathématiques performantes.
* Random : module standard de Python pour générer des nombre pseudo-aléatoires et effectuer des opérations aléatoires
* Pulp : bibliothèque pour la programmation linéaire, permettant de modéliser et résoudre des problèmes d’optimisation (Dans notre cas c’est la minimisation du nombre des couleurs dans un graphe)

- Structures de données principales :

Graphe : la bibliothèque NetWorkX représente un graphe comme dictionnaire des dictionnaires pour stocker les sommets, les arêtes et les attributs.

Composants de base de cette structure :

* Sommets (nœuds) : représentés par des clés dans un dictionnaires. Chaque sommet peut être associé à des attributs stockés comme paires clé-valeurs.
* Arêtes : stockées dans un dictionnaire imbriqué où chaque sommet est mappé à un dictionnaire de ses voisins, avec des attributs optionnels

- Explication des étapes clés de code :

* Développement et test des méthodes de création et affichage des structures des données : On a utilisé la bibliothèque networx pour créer des graphes à partir de nombre des sommets et une liste des tuples montrant les arêtes. Puis on a utilisé la bibliothèque matplotlib pour l’affichage des graphes
* Implémentation de l’algorithme de Welsh-Powell en utilisant le principe suivant : la fonction retourne la liste des couleurs de chaque nœud ainsi que le nombre des couleurs utilisés pour cette solution. On a aussi implémenté une fonction d’affichage de graphe colorié qui prend le graphe et les couleurs comme paramètres
* Test et évaluation des méthodes par des exemples
* Analyse des limites de l’algorithme de Welsh et Powell lors de détermination de nombre chromatique d’un graphe G avec un cas particulier et un exemple de code dont on a utilisé une bibliothèque de programmation linéaire pour l’implémenter.

## 4. Jeux de tests

(ICI LES EXEMPLES , IL FAUT RESPECTER LES INFORMATIONS)

Fournissez au moins 5 cas de tests avec les informations suivantes :

- Description du graphe (type, nombre de sommets/arêtes, pondérations...)

- Capture d'écran du graphe (visualisation ou matrice d’adjacence)

- Résultat de l’exécution de l’algorithme

- Commentaire sur la validité et l’interprétation du résultat

(FIN DE SECTION EXEMPLES)

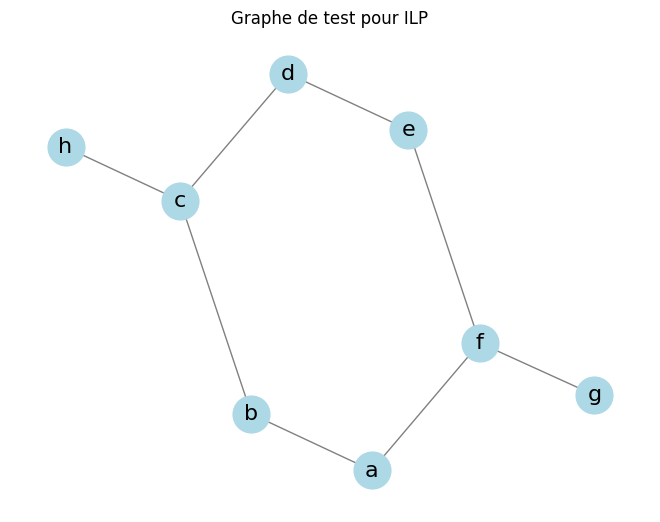
## 5. Discussion et limites

L’algorithme fonctionne dans tous les cas testés, tout en respectant les propriétés de chaque graphe mentionné (complet, biparti, cycle, ...) et le nombre des couleurs obtenu respecte l’encadrement théorique suivant

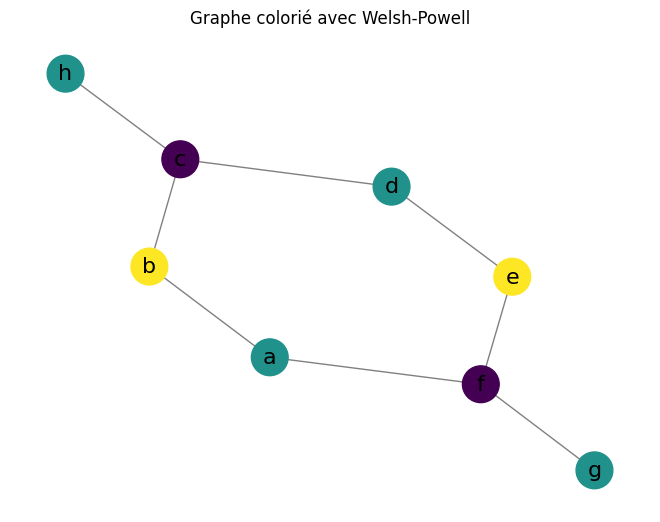
* Si on trouve un coloriage possible d’un graphe avec N couleurs, alors le nombre chromatique est nécessairement inférieur ou égal à N.
* Si le sommet de plus haut degré est r, alors le nombre chromatique est nécessairement inférieur ou égal à (r + 1)

Cependant, cet algorithme ne permet pas de donner le nombre chromatique, càd le nombre des couleurs minimum pour colorier un tel graphe. Un tel contre-exemple se présente comme le suivant :

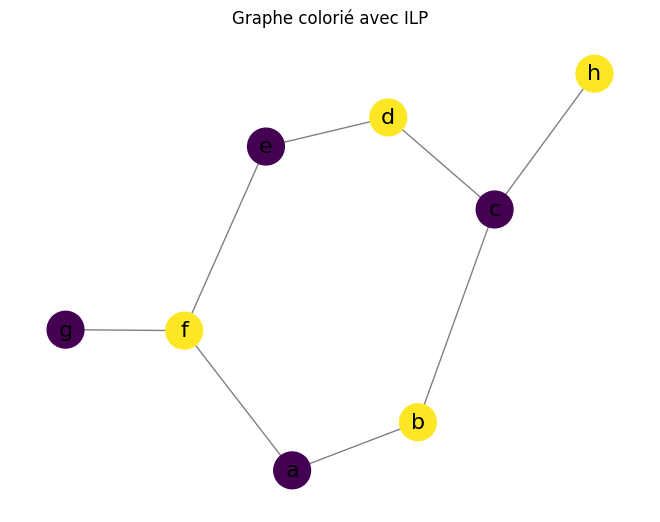
Considérons ce graphe suivant (résultat de code Python) :



Si on utilise l’algorithme de Welsh et Powell , on se trouve dans le cas suivant :



L’algorithme a utilisé 3 couleurs pour colorier le graphe. Par contre si on utilise une approche linéaire optimisé pour minimiser le nombre des couleurs mais couteuse en temps d’exécution avec une complexité théorique exponentielle on peut déduire le suivant :



Il a utilisé que 2 couleurs seulement pour colorier le graphe (ꭓ(G) = 2)

- Amélioration et choix de l’algorithme adéquat :

Le choix d’algorithme nécessaire pour colorier un graphe dépend de plusieurs facteurs, notamment la taille du graphe et sa structure.

Pour les petits graphes (n <= 50) : il est recommandé l’ILP ou une approche qui utilise le backtracking pour une solution exacte

Pour les graphes de taille moyenne (50 <= n <= 1000) : DSatur ou Welsh-Powell pour un bon compromis entre qualité et rapidité.

Pour les grands graphes : algorithmes gloutons ou métaheuristiques

Il faut aussi tenir compte de la structure des graphes qui peuvent faciliter la tâche. Par exemple, pour un graphe planaire on opte à utiliser un algorithme adapté aux 4 couleurs.

## 6. Conclusion

Dans ce TP, nous avons implémenté et analysé l’algorithme de Welsh-Powell pour la coloration des sommets d’un graphe, en utilisant Python. Cet algorithme glouton, qui trie les sommets par degré croissant avant d’assigner les couleurs de manière itérative, s’est relevé rapide et efficace lors des tests malgré qu’il ne donne pas le nombre chromatique exact dans certains cas.

Ce travail nous a permis de comprendre comment et pourquoi choisir un tel algorithme par rapport aux autres tous en étudiant la problématiques et les critères de graphe obtenu lors de la représentation de la structure de données.

## 7. Annexes

- Lien vers le dépôt du code : Vous trouverez le code ainsi que la trace d’exécution de ce TP dans ce répertoire GitHub suivante : https://github.com/Chaabane2k03/Graphes\_IGL3

- Instruction pour l’exécution de code :

* Assurer que vous avez installé Python dans votre machine ainsi que l’environnement de travail Jupiter.
* SI vous ne souhaitez pas l’installer ou vous avez rencontré des problèmes, vous pouvez exécuter le code à partir de la plateforme GoogleColab
* Exécuter le code, en cliquant sur Run All (l’installation des bibliothèques nécessaires est présente dans le code)

- Le rapport a été inspiré par l’article COLORATION DE GRAPHES : FONDEMENTS ET APPLICATIONS de RAIRO Operations Research