**Outline du rapport :**

* Principe général de la méthode, espace de recherche, fonction objective
* Algorithmes utilisés
* Explication du code
* Tests et résultats
* Commentaires

## 1°/ Heuristique du MAX STABLE :

### Principe général :

il est difficile, voire impossible, de trouver un k-coloring d'un grand graphe G (par exemple |V | ≥ 1000) avec k proche de χ(G) en appliquant directement un algorithme donné α sur G. Pour colorier les grands et très grands graphes en appliquant directement un algorithme de coloration α donné sur G. Dans ce cas, une approche possible est d'appliquer un prétraitement pour extraire i grands ensembles indépendants (i stables maximales) du graphe afin d'obtenir un graphe résiduel beaucoup plus petit qui devrait être plus facile à colorier que le graphe initial. Un algorithme de coloration α est ensuite invoqué pour colorier le graphe résiduel.

Puisque chacun des i ensembles forme une classe de couleur, 1 plus le nombre de couleurs nécessaires pour le graphe résiduel donne une borne supérieure pour χ(G).

Les méthodes conventionnelles pour la phase de prétraitement opèrent de manière greedy en extrayant à chaque fois un ensemble indépendant du graphe G.

Puis supprimer de G les sommets de l'ensemble indépendant ainsi que leurs arêtes incidentes. Nous pouvons affiner le choix de l'ensemble indépendant à supprimer et

en prenant l’ensemble indépendant qui est connecté au plus grand nombre possible de sommets du graphe restant.

Un k-coloring légal d'un graphe donné G = (V, E) correspond à une partition de V en k ensembles indépendants. Supposons que l'on veuille colorier G avec k couleurs. Supposons maintenant que nous extrayons t < k ensembles indépendants I1, ..., It de G. Il est évident que si nous pouvions maximiser le nombre de sommets couverts par les t ensembles indépendants (c'est-à-dire que |I1 ∪ ... ∪ It | est aussi grand que possible), nous obtiendrions un graphe résiduel avec moins de sommets lorsque I1 ∪ ... ∪ It sont retirés de G. Cela pourrait à son tour entraîner une diminution du nombre de sommets et en se ramenant à une borne supérieure de coloration inférieure ou égale à k - t couleurs, car il reste moins de sommets dans le graphe résiduel.

### Définition d’un max stable et complexité algorithmique :

Nous avons utilisé une heuristique qui se base principalement sur la relation entre le nombre chromatique et le cardinal de l’ensemble indépendant maximal.

χ(G) + α(G) <= n+1 où n est le nombre de sommets du graphe.

Preuve par Induction : On remarque qu'un seul sommet a χ(G) + α(G) = 2.En ajoute ensuite chaque sommet un par un. Remarquez qu'en ajoutant un seul sommet, on ne peut les augmenter que d'une unité. En faisant cela, nous convertissons G en G\*. Disons que le nouveau sommet v est connecté aux sommets 1 à r de notre G précédent et n'est pas connecté aux sommets r+1 à n. Pour que l'inégalité soit résolue conformément à notre hypothèse d'induction, les deux doivent être augmentés en un point.

Pour que le χ(G) augmente, les sommets 1 à r, v est connecté aux r couleurs dans toute coloration correcte de G. Ainsi |r| >= χ(G). De plus, l'ensemble indépendant maximum dont v doit faire partie doit être contenu dans G - r. Donc |G - r| >= α(G) En ajoutant les deux inégalités, on obtient |G| = n >= χ(G) + α(G)

Maintenant les deux augmentent, montrant que n+2>=χ(G\*) + α(G\*) Ainsi χ(G\*) + α(G\*) <= |G\*| + 1 CQFD La borne est valide pour les graphes complets et les graphes nuls.

Nous comprenons aussi que ce problème est NP-complet, mais il existe un algorithme séquentiel très efficace (en terme de temps d’exécution) permettant de calculer ce stable. La complexité est clairement O(m) où m est l’ordre du graphe, or d’un point de vue technique cet algorithme peut très bien être parallélisé et donc optimisé pour les graphes très larges.

### Algorithme d’extraction du stable maximum :

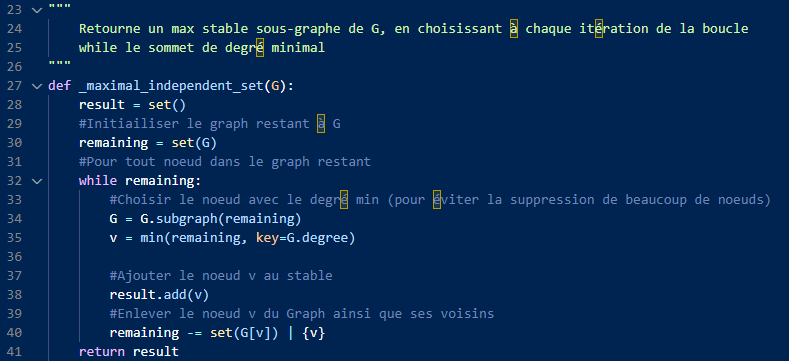
Étant donné un graphe G(V,E), il est facile de trouver un seul MIS en utilisant l'algorithme suivant :

1. Initialiser I à un ensemble vide
2. Tant que V n'est pas vide :

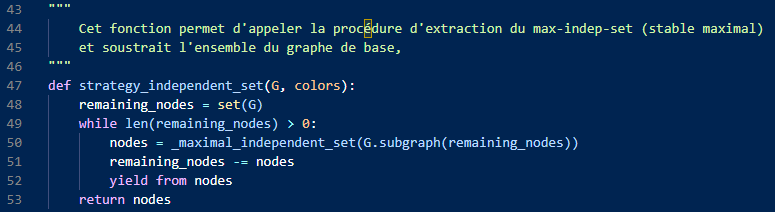
* Choisir un nœud v∈V ;
* Ajouter v à l'ensemble I ;
* Retirer de V le nœud v et tous ses voisins.

1. Retourner I

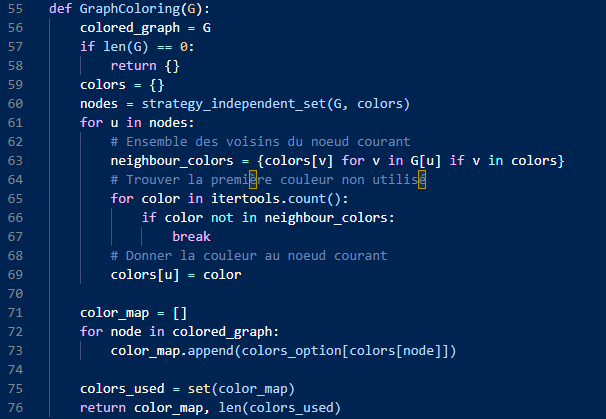
## 2°/ Explication du code :



* Ligne 28 : initialisation d’un ensemble vide qui contiendra
* Ligne 30 : Initialisation de l’ensemble des sommets du sous-graphe restant au graph initial
* Lignes 32 - 40 : tant que le graph restant n’est pas encore vide, nous choisissons à chaque itération le sommet de degré minimal min(remaining, key=G.degree), nous optons pour ce degré pour permettre à l’algorithme d’explorer plus de sommets en maximisant le stable, une fois v calculé, on l'enlève du graph ainsi que ses voisins.
* Ligne 41 : nous retournons ainsi l’ensemble des sommets qui n’ont aucune relation d’adjacence entre eux.



* Ligne 48 : initialisation de l’ensemble des sommets restants
* Lignes 49 - 52 : Tant qu’il y en a encore des sommets dans le graphes, extraire le stable maximum, enlever les sommets de ce dernier du graph orignal
* Ligne 53 : nous retournons uniquement les sommets du stable, car chacun des sommets restant se verra attribuer la première couleur non utilisée (greedy coloring) sur ce dernier.



* Lignes 61 - 69 : pour chacun des sommets retourné par la fonction strategy\_independent\_set, qui représente le stable maximal, nous lui attribuons la première couleur non utilisée par ses voisins.
* Lignes 71 - 73 : pour tous les sommets, s’il appartient au stable alors on lui a déjà attribué une couleur dans la ligne 69, sinon lui attribuer une nouvelle couleur.