Projet 2024-2025
Graphes et complexité
Coloration de graphes

Tous les membres du groupe ont participé à la classe VisualisationGraphe.

Victoria TANDAMBA : Méthode First fit
Yanis CHATAIGNER : Méthode DSatur
Morgan MONTIGNY : Méthode Largest Fit
Manon CATTANEO : Méthodes Backtracking et Smallest last

MÉTHODES ANNEXES :

GETAFFECTATIONCOULEUR :
Retourne l'affectation des couleurs du graphes.

OBTENIRCOULEURNOM:
Méthode pour obtenir le nom de la couleur du sommet à partir du numéro. Elle cherche dans le tableau COULEUR. On l'a utilisée pour la génération du fichier CSV pour le voir dans l'application Cytoscape.

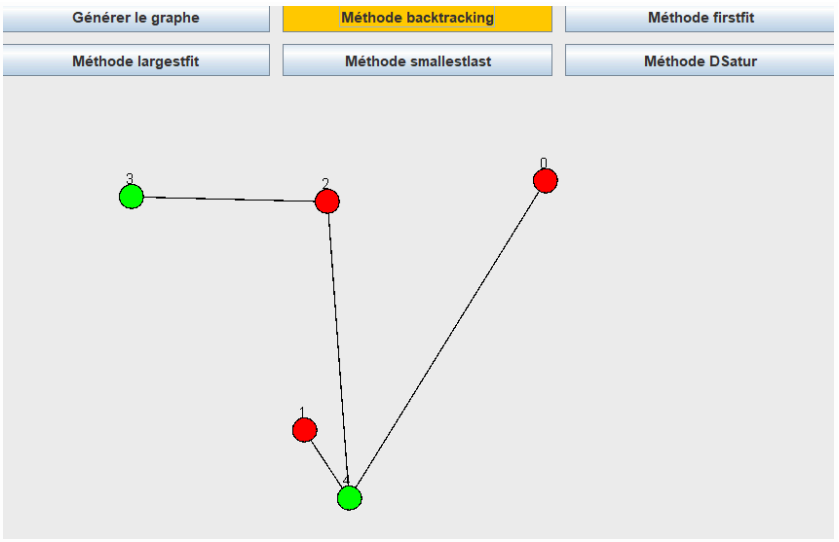
COULEURVALIDE:
Méthode vérifiant si la couleur est affectable au sommet.

MÉTHODE VISUALISATIONGRAPHE:

2 classes imbriquées permettant d'afficher le graphe dans une JFrame. On retrouve des boutons permettant de générer le graphe puis de choisir quelle méthode appliquer. Il faut simplement générer à nouveau un graphe afin de changer de méthode de génération (car chaque méthode vérifie que les sommets soient tous colorés, et ne peut donc pas se régénérer puisque les sommets sont déjà colorés).

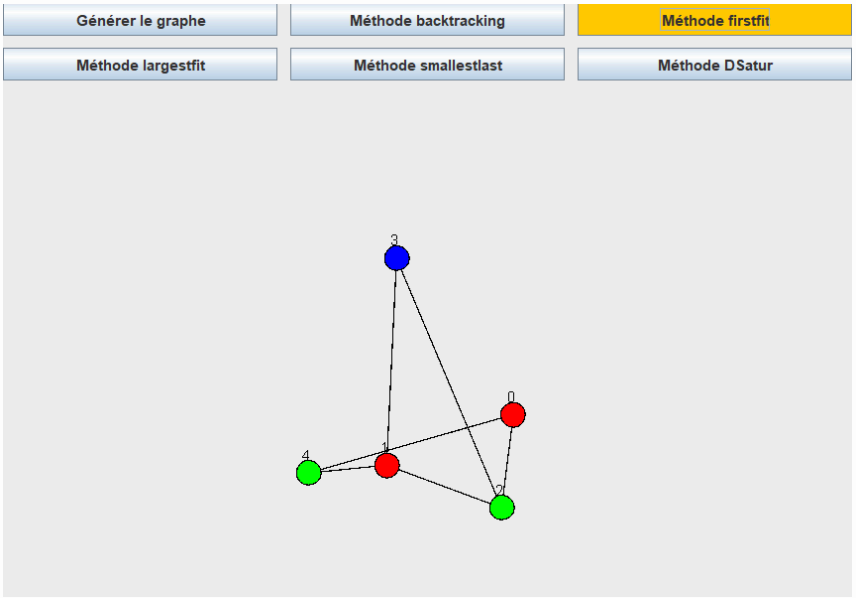
MÉTHODES PRINCIPALES DE COLORATION:

BACKTRACKING :
La méthode prend en argument d'entrée un INT qui symbolise le sommet par lequel on commence à colorier. Elle vérifie en premier, comme dans toutes les méthodes, si tous les sommets ne sont pas coloriés. Dans la boucle suivante, on cherche à colorier tous les sommets avec une couleur différente du voisin. On utilise la méthode définie au préalable (couleurValide(sommet couleur)) pour savoir si la couleur peut être affectée au sommet en vérifiant la couleur des voisins. Si effectivement on peut mettre la couleur I du tableau, on l'affecte grâce à la méthode affectationCouleur. Si la solution n'est pas possible, on enlève la couleur et on continue la boucle. S'il n'y a aucune solution, on renvoie false.



COLORATION FIRSTFIT:

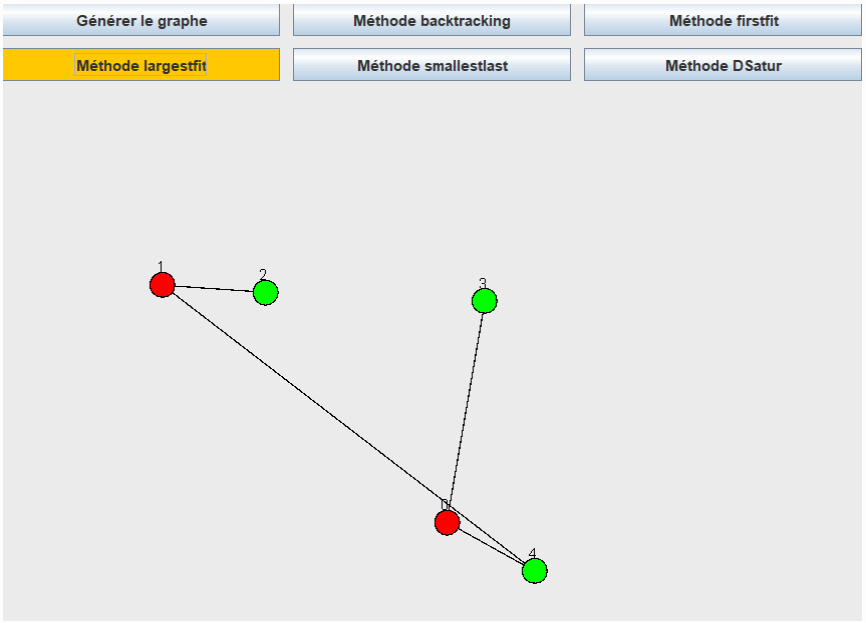
La méthode ne prend aucun argument d'entrée. Elle parcourt successivement tous les sommets du graphe. Pour chaque sommet traité, elle vérifie d'abord les couleurs déjà utilisées par ses sommets voisins directs. Ensuite, elle recherche la première couleur disponible qui n'a pas encore été utilisée par ces voisins, en parcourant les couleurs dans l'ordre croissant. Dès qu'une couleur valide est trouvée, elle est immédiatement attribuée au sommet concerné en mettant à jour la structure affectationCouleurs. Cette opération est répétée pour chacun des sommets jusqu'à coloration complète du graphe.



LARGEST FIT:

La méthode ne prend aucun argument d'entrée. Elle crée d'abord une liste de tous les sommets du graphe, puis trie ces sommets en ordre décroissant en fonction du

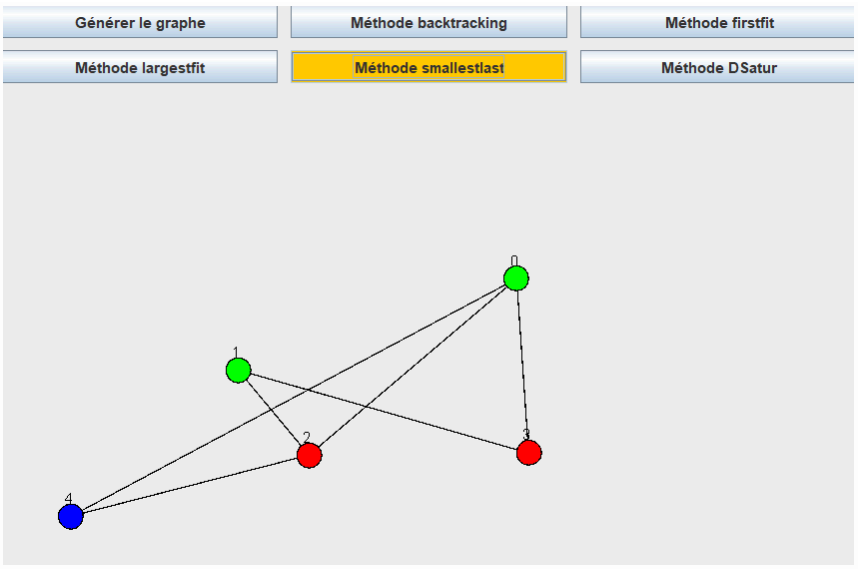
nombre de leurs voisins (degré). Ensuite, elle utilise une méthode auxiliaire (colorationFirstFitAvecOrdre) pour attribuer rapidement les couleurs en suivant l'ordre ainsi obtenu. Cette méthode auxiliaire parcourt chaque sommet dans l'ordre établi, vérifie les couleurs des sommets voisins déjà coloriés, et affecte au sommet courant la première couleur disponible différente de celles de ses voisins. L'attribution des couleurs se fait en mettant à jour la structure affectationCouleurs.



SMALLEST LAST :

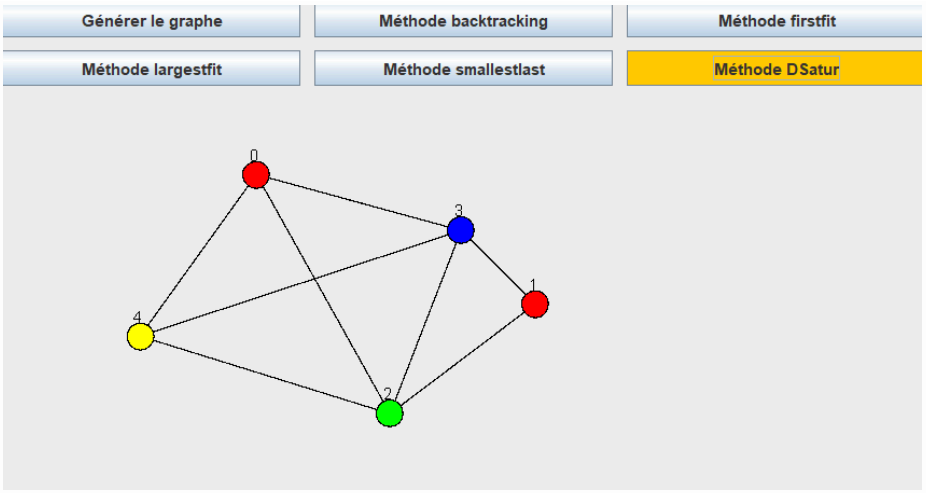
La méthode va utiliser une méthode externe colorationFirstFitAvecOrdre qui permet de d'attribuer les couleurs rapidement après avoir défini l'ordre des sommets dans la méthode. On a défini cette méthode afin d'éviter les répétitions dans le code entre LARGEST FIT et SMALLEST LAST.

La méthode ne prend pas d'arguments d'entrée. Elle crée dans un premier temps une liste vide pour stocker l'ordre des sommets et une liste de sommets restants. Elle utilise une boucle qui ne s'arrête pas tant qu'il reste des sommets à colorier. Elle commence à chercher le sommet qui a le moins de voisins, elle l'ajoute à la liste d'ordre et l'enlève des sommets restants. Une fois que la liste des sommets restants est vide, on inverse l'ordre de la liste de coloration et grâce à la méthode colorationFirstFitAvecOrdre, on colorie tous les sommets.



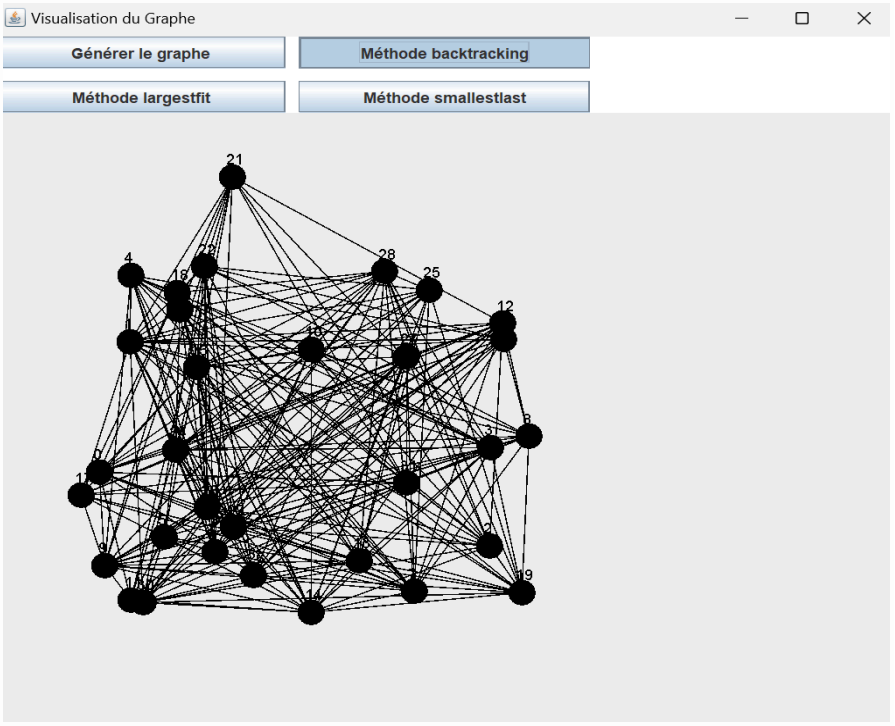
COLORATION DSATUR:

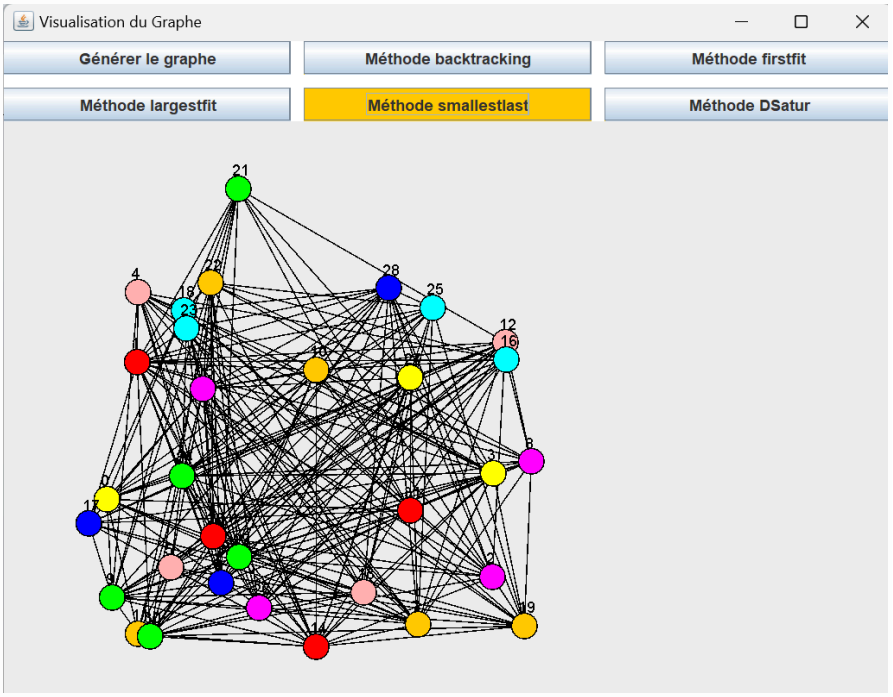
Pour cette méthode on utilise une Map<Integer, Integer> pour stocker la saturation de chaque sommet. La saturation d'un sommet est le nombre de couleurs différentes utilisées par ses voisins déjà coloriés. Elle est initialisée à 0 pour tous les sommets. Pour la sélection d'un sommet à colorier on s'appuie sur 2 conditions : Tant qu'il reste des sommets non coloriés, on choisit celui qui a la saturation la plus élevée.En cas d'égalité, on peut choisir un sommet avec un degré plus élevé (ce qui n'est pas explicitement fait dans le code). Par la suite pour le choix de la couleur on récupère toutes les couleurs déjà utilisées par les voisins du sommet sélectionné. Et on choisit la plus petite couleur non utilisée. Après ça on met à jour la saturation : Une fois le sommet colorié, la saturation de chacun de ses voisins non coloriés est augmentée de 1, car ils ont maintenant un voisin supplémentaire colorié.



RÉPONSES AUX QUESTIONS:

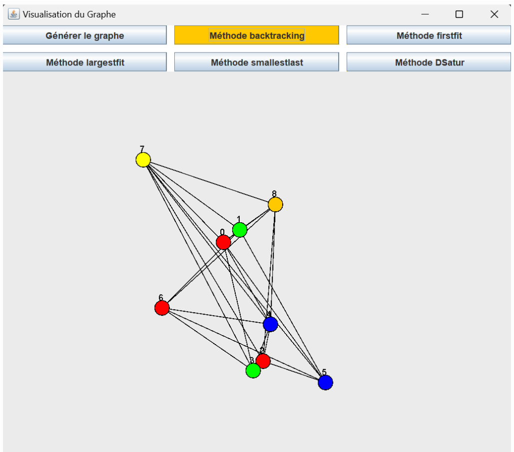
Dans l'algorithme de backtracking, on a remarqué qu'au-delà des 30 sommets la coloration ne se faisait pas (le temps d'attente pour l'afficher était élevé +5min) alors que pour les autres graphes on arrivait à obtenir un affichage, certes surchargé. On peut même voir que la JFrame ne s'est pas entièrement chargée.

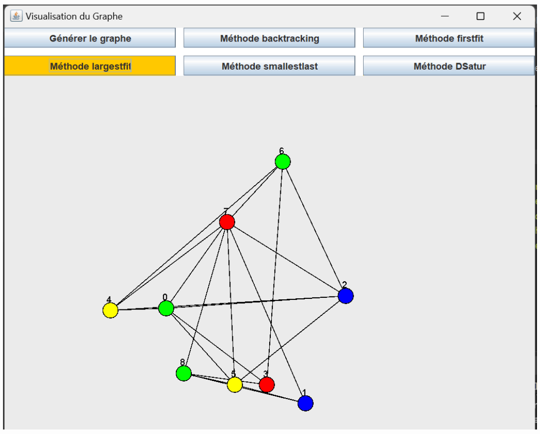


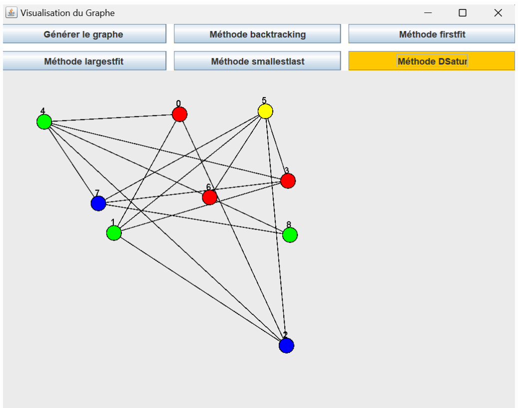


On a remarqué au fur et à mesure des tests, que les méthodes autres que celle de backtracking sont plus rapides avec une utilisation optimisée des couleurs.

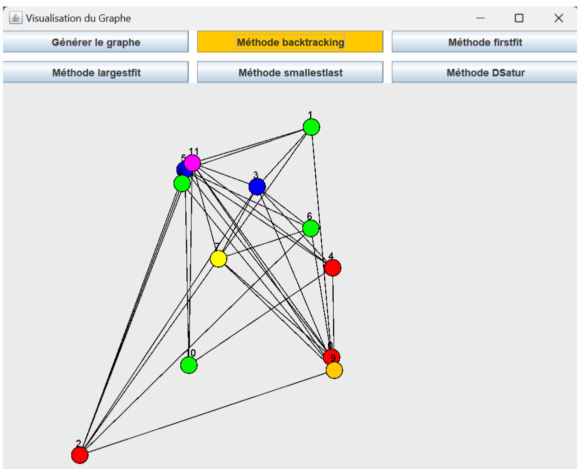
Pour un graphe ayant le même nombre de sommets, on remarque que la méthode backtracking utilise plus de couleurs (dans cet exemple on est sur 1 couleur supplémentaire)

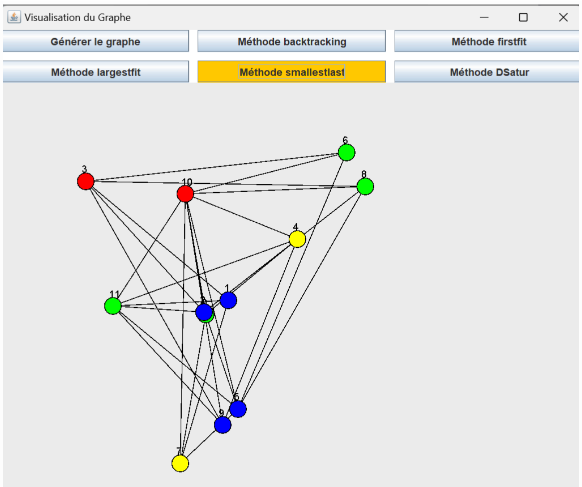






Ou même 2 dans cet exemple ci.





On réfléchit très rapidement à la question du sudoku et effectivement il serait possible d'envisager une grille de sudoku comme un graphe avec des sommets (les cases) et des contraintes, les lignes, colonnes et régions comme définissant les arêtes de ce graphe. On n'a cependant pas pu mettre en place une solution ou un exemple illustrant notre idée.