PINSON Daniel
MOURELON Yann

07/05/2022

Rapport Projet Graphe 2022

Introduction :

Objectif 1 : Réaliser un graphe à partir d'un fichier SNAP ou KONECT

Objectif 2 : Calculer la dégénérescence de celui-ci et l'afficher.

Objectif 3 : Implanter au moins une extension parmis celles proposée sur le sujet :

- Algorithme de Matula & Beck (dégénérescence)
- Comparaison entre la dégénérescence et le nombre chromatique d'un graphe
- Dessiner de façon "joli" le graphe sous forme de cercle

Pour ce projet, nous avons décidé de travailler sur python en utilisant les librairies matplotlib pour l'affichage du graphe, numpy pour récupérer la valeur de π et utiliser les méthodes cosinus et sinus pour dessiner le graphe sous forme de cercle, sys pour récupérer le fichier snap ou konect qu'on passe en argument du lancement de notre programme et copy pour pouvoir copier plus simplement notre graphe (on s'en sert pour calculer la dégénérescence du graphe)

Partie 1 : Génération du graphe

Pour générer notre graphe, nous devons nous servir d'un fichier SNAP ou KONECT, dont le graphe est représenté comme suit : une paire "A B" qui nous donne un arc du noeud A vers le noeud B pour un graphe orienté ou bien une arête entre le sommet A et le sommet

B pour un graphe non-orienté (ou orienté symétrique), A et B étant des nombres. On parcourt alors le fichier pour connaître le nœud ayant la plus grande valeur (qu'on va appeler nMax), puis on crée les nœuds de 0 à nMax qui seront contenus dans une liste. Ensuite on parcourt notre fichier une seconde fois pour connaître toutes les différentes arêtes et les stocker dans une autre liste sous forme de tuple. Tout est fait dans notre méthode *generateGraphFromSnap()*.

Partie 2 : Calcul de la dégénérescence

Pour calculer la dégénérescence du graphe, on fait une copie de notre graphe pour pouvoir travailler car notre algorithme détruit le graphe or il est nécessaire de le garder pour calculer la coloration ou l'afficher. On crée un tableau de "centre" qui va stocker, pour chaque nœud, son centre qu'on initialise à -1 ainsi qu'une variable k qui représente la dégénérescence de notre graphe. Dans une boucle infinie, on regarde le degré des différents nœuds de notre graphe et si le degré est inférieur ou égal à k alors on supprime le nœud de notre graphe ce qui réduit le degré de ses voisins de 1, et on met à jour le tableau de centres qui vas stocker à l'indice noeud, son centre. Une fois tous les noeuds enlevé, on vérifie que le graphe est non-vide (qu'il reste encore des noeuds) et si tel est le cas, alors on incrémente k de 1, puis on recommence jusqu'à ce que tous les noeuds aient été supprimés. On retourne alors la valeur de dégénérescence du graphe et le tableau des centres.

Partie 3 : Implémentation d'une extension

Pour ce projet, nous avons décidé d'implémenter 2 extensions :

- La 3b) on doit comparer la dégénérescence et le nombre chromatique de plusieurs graphes et noter nos observations sur ceux-ci
- La 3c) On utilise les numéros de centres pour obtenir un "joli" dessin du graphe en disposant les sommets le long de plusieurs cercles en fonction de leurs centres.

Partie 3b : Colorisation du graphe avec glouton dans l'ordre des k-centres décroissants

Pour trier le graphe en ordre des centres décroissant, on utilise un tableau *centres* contenant les centres de chaque sommet (centre[i] = 2 signifie que le sommet *i* a pour centre 2) dans la fonction *sortNodes(centres)*. On crée une liste *inter* contenant autant de listes vides que la valeur maximale du tableau *centres* (centres = [1, 2, 3, 4] créée 4 sous-listes dans *inter*). On parcourt ensuite la liste de centres en fonction des sommets et on met dans la sous-liste *k-1* le sommet *i*, avec *i* le numéro de sommet et *k* le numéro de centre. Ensuite, il suffit d'aplatir *inter* et pour l'avoir dans l'ordre des centres décroissant, il faut inverser le

résultat. On inverse les sous-listes afin d'avoir les sommets dans l'ordre tel qu'il est dans *centre*.
Pour colorer le graphe, non créer une liste *color* qui vaut -1 pour tous les sommets (color[i] = 2 signifie que le sommet i est de couleur 2). Ensuite on parcourt le graph dans l'ordre des centres décroissants. Pour chaque sommet on crée une liste contenant toutes les couleurs de ses voisins que l'on trie par la suite. Puis on donne la première couleur qui n'est pas dans la liste de ses voisins en partant de 1 et en incrémentant à chaque fois qu'un voisin est de cette couleur.

Observation : La coloration d'un graphe est en moyenne à +-1 de la dégénérescence de celui-ci.

Partie 3c : Disposition des sommets pour un joli graphe:

Il est difficile d'afficher correctement un graphe pour que celui-ci soit lisible, et c'est ce pour quoi on va disposer les sommets de nos différents graphe sur des cercles en fonction de leurs centres. Pour ce faire, on a créé un tableau nommé *position* qui stock la position de chaque nœud comme suit : [n, (x,y)] avec n le nœud et (x,y) ses coordonnées. On a alors créé une méthode appelé *modifPos(G, degenGraph, centres)* qui prend en argument le graphe G sur lequel on travaille, la dégénérescence de celui-ci et les centres de ses nœuds. On utilise alors la formule $\{x = r * cos(\theta),\ y = r * sin(\theta)\}$. Puis pour afficher le graphe, on utilise matplotlib qui nous offre la possibilité d'afficher comme on souhaite nos graphes.
Pour le rayon, on met la valeur du centre du noeud que l'on souhaite afficher, puis pour θ, on a créé 2 dispositions différentes. La première ne concerne que les nœuds qui ont un centre égal à la dégénérescence du graphe : on les dispose le long d'un cercle de rayon 1. Pour tous les autres nœuds, on les dispose tous sur un cercle de rayon variable en fonction de leurs centres.

Limitations / Extensions à apporter :

Au niveau des limitations de notre programme, nous pensons que celui-ci peut être amélioré de façon à ce qu'il soit plus optimisé et plus rapide sur de gros graphes. Lorsqu'on le teste sur de petits graphes, le temps de calcul est correct. En revanche, si on le teste sur de plus gros graphes, le temps de calcul devient conséquent. Aussi peut-on améliorer l'affichage. Nous pouvons faire en sorte d'agrandir les rayons des cercles pour des gros graphes et a contrario, réduire celui-ci pour de plus petits graphes.

Au niveau des extensions, nous n'avons pas fait la partie 3a) l'implémentation de Matula&Beck, Aussi pouvons-nous par exemple implanter une méthode qui peut calculer les cliques maximales et maximum ainsi que les stables maximales et maximum du graphe.