Coloriage de graphes

Durée : 2 heures

Dans tout le problème, on considère un graphe *non orienté* $G = (S, A)$ où $S = \{0, 1, \dots, n-1\}$ est un ensemble de sommets et A un ensemble d'arêtes. Ces graphes seront représentés en Python par liste d'adjacence.

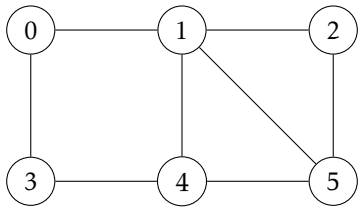
Partie I. Définitions

Si $k \in \mathbb{N}^*$ on appelle k-coloriage de G une application $c : S \to [\![0, k-1]\!]$ telle que :

$$\forall (u, v) \in A, \quad c(u) \neq c(v)$$

Autrement dit, un k-coloriage attribue une couleur (représentée par un entier de $[\![0, k-1]\!]$) à chaque sommet de sorte que deux sommets adjacents soient de couleurs différentes.

On note G_1 le graphe représenté ci-dessous :



Question 1. Donner la représentation Python de G_1 sous forme de liste d'adjacence.

Question 2. Recopier G_1 sur votre copie en ajoutant, à côté de chaque sommet, une couleur (autrement dit un entier) de façon à obtenir une 3-coloration.

Question 3. Justifier que G_1 ne possède pas de 2-coloration.

Dans toute la suite du problème, on représentera une k-coloration par une liste C de longueur n telle que $C[i]$ est la couleur (autrement dit un entier de $[\![0, k-1]\!]$) du sommet i.

Question 4. Rédiger une fonction `valide(G, C)` qui prend pour arguments un graphe G *représenté par liste d'adjacence* et une liste C et qui renvoie le booléen `True` ou `False` suivant que C est une coloration valide ou non de G.
On supposera sans avoir besoin de le vérifier que les listes G et C sont de même longueur.

Partie II. Degré

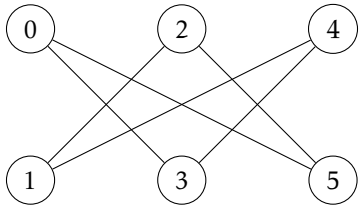
Dans cette partie, on détermine une majoration du nombre de couleurs nécessaires pour colorer un graphe.

Question 5. Écrire en Python une fonction `deg(G, s)` qui prend pour arguments un graphe G représenté par liste d'adjacence et un sommet s et qui renvoie le degré de s dans G.

Question 6. En déduire une fonction `degMax(G)` qui prend pour argument un graphe G représenté par liste d'adjacence et renvoie le degré maximum d'un sommet dans G. On note $\Delta(G)$ cette quantité.

Il existe un algorithme simple donnant une $(\Delta(G)+1)$-coloration pour un graphe G : considérer chaque sommet un par un et lui attribuer le plus petite couleur disponible, autrement dit la plus petite couleur qui n'est pas déjà la couleur d'un voisin de ce sommet.

Question 7. On note G_2 le graphe représenté ci-dessous :



Que donne cet algorithme lorsqu'on l'applique au graphe G_2, en traitant les sommets par ordre croissant (de 0 à 5)? Existe-t-il une coloration utilisant moins de couleur ?

Question 8. Écrire une fonction `deltaColor(G)` qui prend pour argument un graphe G représenté par liste d'adjacence et qui renvoie une $(\Delta(G)+1)$-coloration de G, autrement dit une liste C telle que $C[s]$ soit la couleur attribuée à s.

Partie III. Clique

Dans cette partie, on s'intéresse à une minoration du nombre de couleurs nécessaires pour colorer un graphe.

Une *clique* d'un graphe G est un ensemble de sommets contenant toutes les arêtes possibles entre deux sommets de cet ensemble. La *taille* d'une clique est le nombre de sommets qui le composent.
Par exemple, l'ensemble de sommets $\{1,2,5\}$ est une clique de taille 3 de G_1 puisque chacun de ces trois sommets est relié aux deux autres.

Question 9. Justifier que si G contient une clique de taille k alors G n'est pas $(k-1)$-coloriable.

Question 10. Donner un exemple de graphe connexe 2-coloriable ne contenant pas de clique de taille 3 (on pourra se contenter de le dessiner).

Question 11. Rédiger une fonction `clique(G, S)` qui prend pour arguments un graphe G représenté par liste d'adjacence et une liste de sommets S et qui renvoie le booléen `True` si S est une clique de G, et `False` sinon.

Une clique est dite *maximale* lorsqu'elle ne peut être incluse dans une clique de plus grande taille. Pour vérifier qu'une clique est maximale, il suffit de vérifier qu'aucun sommet ne peut lui être ajouté et former encore une clique.

Question 12. Rédiger une fonction `cliqueMax(G, S)` qui prend pour arguments un graphe G représenté par liste d'adjacence et une liste de sommets S et qui renvoie le booléen `True` si S est une clique maximale, et `False` sinon. On supposera sans qu'il soit nécessaire de le vérifier que S est une clique.

Partie IV. Graphe biparti

Un graphe *biparti* est un graphe connexe qui possède une 2-coloration. Pour déterminer si un graphe est biparti, on exécute un parcours depuis un sommet quelconque (par exemple le sommet 0) que l'on colorie avec la couleur 0, puis on colorie les voisins avec la couleur 1, puis les voisins des voisins avec la couleur 0, etc.
Si, à un moment du parcours, on doit colorier un sommet avec une certaine couleur alors qu'il a déjà été colorié d'une couleur différente, le graphe G n'est pas biparti.

Question 13. Rédiger une fonction `biparti(G)` qui prend pour argument un graphe G (que l'on supposera connexe) défini par liste d'adjacence et qui renvoie le booléen `True` si G est biparti, et `False` sinon.

Remarque. On pourra utiliser le fait que si $c \in \{0,1\}$ est une couleur, alors l'autre couleur est $1-c$.