Colorer un graphe

La coloration d'une carte de pays consiste à attribuer une couleur à chacun des pays de manière à ce que deux pays voisins soient de couleurs différentes.
Étudier ces techniques de coloration revient de façon plus abstraite à travailler sur des graphes.
Le champ d'applications de la coloration de graphes est très vaste et couvre des domaines aussi variés que le problème de l'attribution de fréquences dans les télécommunications, la conception de puces électroniques ou l'allocation de registres en compilation.

Soulignons que tous les graphes considérés dans ce sujet sont non-orientés.
Quelques rappels de syntaxe Python figurent dans l'**annexe 2**.

Partie I - Des algorithmes pour colorer un graphe

I.1 - Introduction sur un exemple

On cherche à colorer une carte de pays avec comme seule contrainte que deux pays ayant une frontière commune ne peuvent être de la même couleur.
Comme les pays, les couleurs sont numérotées à partir de zéro.

À titre d'exemple, on considèrera la carte suivante (**figure 1**), comportant 8 pays numérotés de 0 à 7,

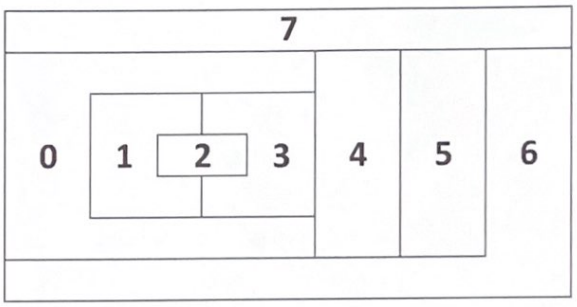

Figure 1 - Exemple d'une carte de pays

que l'on représentera par le graphe G_{ex} donné en **figure 2** :

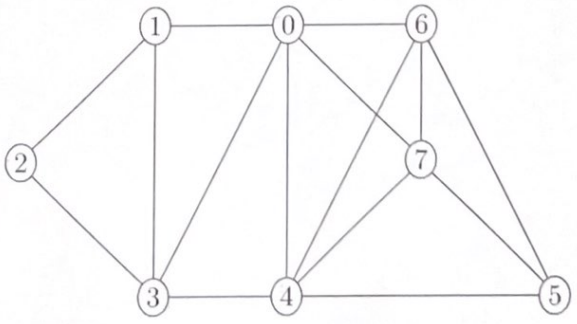

Figure 2 - Graphe G_{ex} associé à la carte de pays de la **figure 1**

Ainsi, sur le graphe de la **figure 2**, deux pays sont voisins si et seulement si les sommets correspondants sont reliés par une arête.

Q1. Un graphe peut être représenté par une matrice d'adjacence.
Le **DR** contient la matrice d'adjacence pré-remplie du graphe G_{ex}.
La compléter et expliquer le processus de construction.

Q2. Une autre manière de représenter en mémoire un graphe est d'utiliser une liste d'adjacence.

La liste d'adjacence d'un graphe (constitué de sommets numérotés de 0 à $n-1$) est une liste `LA`, telle que `LA[i]` est la liste des sommets voisins du sommet `i`. Par convention, les voisins énumérés dans `LA[i]` seront listés dans l'ordre croissant.

Donner la liste d'adjacence du graphe G_{ex} présenté en exemple.

Q3. Donner un avantage et un inconvénient d'une représentation par matrice d'adjacence.

Donner un avantage et un inconvénient d'une représentation par liste d'adjacence.

Q4. On rappelle que le degré d'un sommet `s` d'un graphe `G` est le nombre de voisins du sommet `s`, c'est-à-dire le nombre de sommets reliés à `s`.

Le **DR** fournit le tableau des degrés des différents sommets du graphe G_{ex}.

Le compléter.

I.2 - Tester si une coloration est valide

Q5. Écrire une fonction `voisins` avec trois arguments, deux nombres entiers distincts i et j et une liste d'adjacence `LA` représentant un graphe, qui renvoie `True` si les sommets numérotés i et j sont reliés par une arête et `False` sinon.

On considère une carte de pays, représentée par un graphe `G` pour laquelle on a une proposition de coloration donnée par une liste `C`. Ainsi, `C[i]` donne le numéro de la couleur attribuée au sommet `i`. On souhaite déterminer si la coloration proposée est valide (deux sommets du graphe reliés par une arête ne peuvent pas être de la même couleur).

Q6. Écrire la fonction `coloration_valide` avec pour arguments une liste d'adjacence `LA` et une liste de couleurs `C`, qui renvoie `True` si la coloration est valide, `False` sinon.

Q7. Pour un graphe comportant `n` sommets, quelle est la complexité temporelle dans le pire des cas de la fonction `coloration_valide` ?

I.3 - Un algorithme intuitif de coloration

L'attribution des couleurs à chaque sommet est caractérisée par une liste `C` où `C[i]` est la couleur attribuée au sommet `i` ; `C[i]` vaut -1 si la couleur n'est pas encore attribuée. La liste `C` ne contient que des -1 au départ et ses valeurs sont modifiées progressivement au fur et à mesure que les couleurs sont attribuées.

Q8. On suppose (dans cette question seulement) que `n` est une constante déjà définie. Écrire la ou les instruction(s) permettant de créer une liste initiale `C` composée de `n` éléments valant -1.

Q9. Compléter la fonction `colore_sommet` ayant trois arguments, la liste `C` des couleurs attribuées, le numéro `s` du sommet à colorer et la liste d'adjacence `LA` caractérisant le graphe.

Cette fonction ne renvoie rien mais modifie la liste `C` en donnant à `C[s]` la plus petite couleur possible, en fonction des couleurs des sommets voisins qui sont déjà colorés.

Par exemple, pour le graphe G_{ex}, prenons `C=[0,1,-1,-1,-1,-1,-1,-1]`. Les sommets 0 et 1 ont donc déjà été colorés avec les couleurs `C[0]=0` et `C[1]=1`.

L'appel `colore_sommet(C,2,LA)` modifie la liste `C` en `C=[0,1,0,-1,-1,-1,-1,-1]`. Cela veut dire que le sommet 2, adjacent au sommet 1 mais pas au sommet 0, a reçu la même couleur que le sommet 0.

Q10. À l'aide de **Q9**, écrire une fonction `colorer1` avec pour argument une liste `LA` caractérisant un graphe, qui crée et renvoie la liste `C` des numéros des couleurs attribuées en colorant les sommets un par un par ordre croissant de leurs numéros.
Par exemple, l'application de la fonction `colorer1` au graphe G_{ex} renverra la liste de couleurs `[0,1,0,2,1,0,2,3]`.

Q11. L'ordre de coloration imposé à la question précédente est arbitraire. On souhaite maintenant colorer le graphe en traitant les sommets selon un ordre arbitraire donné en argument.
Écrire une fonction `colorer2` analogue à `colorer1` et avec un argument supplémentaire, une liste `ordre` fixant l'ordre de coloration des pays.
Par exemple `colorer2([0,2,4,6,1,3,5,7],LA)` colorera le graphe G_{ex} en commençant d'abord par le sommet `0`, puis en continuant par les sommets `2,4,6`...

Q12. Donner la liste des couleurs renvoyée par `colorer2` pour colorer le graphe G_{ex} donné en exemple en prenant `ordre=[7,6,5,4,3,2,1,0]`.
Combien de couleurs ont-elles été utilisées ?

La méthode que nous venons de décrire est rapide et fonctionne plutôt bien. Cependant, si on cherche à utiliser le nombre minimum de couleurs, l'efficacité de l'algorithme proposé ci-dessus dépend en grande partie de l'ordre dans lequel on choisit de colorer les sommets du graphe.

L'objectif des sous-parties suivantes est d'affiner la stratégie pour mieux choisir cet ordre de coloration.

I.4 - Variante de Welsh-Powell

Une alternative est donnée par la variante de Welsh-Powell. L'idée est de parcourir l'ensemble des sommets du graphe par ordre décroissant de leurs degrés.
Comme le degré d'un sommet est un entier positif, il est possible d'écrire un algorithme de tri efficace (dit par répartition).

Q13. Écrire une fonction `degre` avec pour argument la liste d'adjacence `LA` d'un graphe quelconque, qui renvoie la liste des degrés des sommets du graphe.
Par exemple, pour un graphe de liste d'adjacence `LA=[[1,2],[0,2,3],[0,1,3],[1,2,4],[3]]`, la fonction `degre` renverra la liste `[2,3,3,3,1]`.

Q14. Écrire une fonction `init` avec pour argument un entier `n`, qui renvoie une liste de listes `R` de taille `n`, telle que `R[i]` soit une liste vide.
Par exemple, `init(3)` renverra `[[],[],[]]`.

Q15. Écrire une fonction `ranger` avec pour argument une liste d'adjacence `LA`, qui renvoie une liste `R` de même taille que `LA`, telle que `R[i]` soit la liste des sommets de degré `i`.
Ainsi, pour l'exemple de la question **Q13**, l'appel `ranger(LA)` renverra la liste `[[],[4],[0],[1,2,3],[]]`.

Q16. Écrire une fonction `renverse` avec pour argument une liste `L`, qui crée et renvoie une nouvelle liste obtenue en lisant `L` dans l'ordre inverse.
Par exemple, `renverse([1,2,3,4])` renverra `[4,3,2,1]`.

Q17. Écrire une fonction `trier_sommets` avec pour argument une liste d'adjacence `LA`, qui renvoie la liste des sommets triés dans l'ordre décroissant de leur degré.
Par exemple, pour un graphe de liste d'adjacence `LA=[[1,2],[0,2,3],[0,1,3],[1,2,4],[3]]`, la fonction `trier_sommets` renverra la liste de sommets `[1,2,3,0,4]`.

Q18. Pour un graphe à `n` sommets, quelle est la complexité temporelle de la fonction `trier_sommets` dans le pire des cas ?

Q19. Écrire la fonction `colorer3` avec pour argument une liste d'adjacence `LA`, qui crée et renvoie une liste de couleurs `C`, telle que `C[i]` soit la couleur à attribuer au sommet numéro `i`, les sommets étant colorés dans l'ordre décroissant de leur degré.

Quelle est la complexité de `colorer3` dans le pire des cas pour un graphe à n sommets ?

Q20. Pour le graphe G_{ex} de la **figure 2**, donner la liste `C` des couleurs renvoyée par la fonction `colorer3`.

L'amélioration proposée donne de bons résultats dans un grand nombre de cas. Cependant, il reste quelques cas où la coloration obtenue utilise trop de couleurs.

La méthode suivante, proposée en 1979 par Danier Brélaz de l'École Polytechnique Fédérale de Lausanne, raffine la détermination de l'ordre de coloration. La priorité de coloration est ainsi recalculée après chaque traitement d'un sommet et non plus une fois pour toute au départ. Au final, cette approche fournit rapidement une coloration optimale dans un très grand nombre de cas.

I.5 - Algorithme DSATUR

Lorsque l'on colore un graphe, le degré de saturation d'un sommet est le nombre de couleurs différentes déjà attribuées à ses différents sommets voisins. Évidemment, ce degré de saturation est susceptible d'évoluer à chaque fois qu'un nouveau sommet est coloré.

Q21. Écrire une fonction `degre_satur` avec 3 arguments, une liste d'adjacence `LA`, un sommet `s` du graphe, une liste `C` de couleurs. Cette fonction renvoie le degré de saturation du sommet `s`. On rappelle que le sommet `i` est coloré si et seulement si `C[i]` est différent de -1.

Q22. Écrire une fonction `liste_satur` avec deux arguments, une liste d'adjacence `LA`, la liste `C` des couleurs des sommets, qui renvoie la liste des sommets non colorés du graphe ayant un degré de saturation maximum parmi les sommets non colorés.

On notera qu'il s'agit d'une liste car plusieurs sommets peuvent avoir le même degré de saturation. On supposera de plus qu'il reste au moins un sommet non coloré.

Q23. Écrire une fonction `pas_fini` avec pour argument une liste `C`, qui renvoie `True` si cette liste contient la valeur `-1`, `False` sinon.

Q24. Compléter la fonction `colorer4` ayant pour argument une liste d'adjacence `LA`, qui renvoie une liste `C` constituant une coloration du graphe. Cette fonction procède de la façon suivante.

Tant qu'il reste un sommet non coloré :

- déterminer parmi les sommets non colorés ceux de degré de saturation maximale ;
- si plusieurs sommets non colorés ont un degré de saturation maximale, en choisir un parmi ceux-ci qui soit de degré maximal ;
- colorer le sommet choisi en lui attribuant la couleur disponible ayant la plus petite valeur.

Il n'est pas facile d'être certain d'avoir utilisé le nombre minimum de couleurs. La sous-partie suivante propose une piste.

I.6 - Un minorant du nombre de couleurs nécessaires

Considérons un graphe `G` et un sous-ensemble `K` de sommets de ce graphe. On dit que `K` forme une clique si et seulement si pour chaque paire de sommets de `K`, il existe une arête les reliant.
Par exemple, sur le graphe G_{ex} de la **figure 2**, `(4, 5, 6, 7)` constitue une clique de 4 sommets et `(4, 5, 6)` une clique de 3 sommets. Par contre, `(0, 4, 5, 6, 7)` ne forme pas une clique puisque le sommet 0 n'est pas relié au sommet 5.
Enfin, on note n_c le cardinal de la plus grande clique de `G`.

Q25. Justifier que le nombre minimum de couleurs pour colorer un graphe est supérieur ou égal au nombre n_c.
Montrer également que : $n_c \leqslant 1 + \max\{\deg s,\ s \in G\}$ où $\deg s$ désigne le degré du sommet s dans le graphe G.

Dans ce qui suit, on suppose toujours que les sommets d'un graphe à n sommets sont numérotés de 0 à $n-1$.

Q26. Écrire une fonction `est_clique` avec deux arguments, la liste d'adjacence `LA` d'un graphe et un tuple `K` de sommets de ce graphe, qui renvoie `True` si `K` est une clique, `False` sinon.

Q27. La fonction `combinations` du module `itertools`, présentée à l'**annexe 2**, fournit toutes les combinaisons de taille fixée d'une liste.
Sur le **DR**, compléter la fonction `minoration_nb_couleurs` ayant pour argument la liste d'adjacence `LA` d'un graphe, qui renvoie le cardinal de la plus grande clique du graphe considéré.

Ainsi, si le nombre de couleurs utilisées pour colorer le graphe est égal à n_c, on est sûr d'avoir utilisé le nombre minimum de couleurs.

Partie II - Interrogation d'une base de données géographiques

On dispose d'une base de données composée de quatre tables (**annexe 1**) :
– La table `Continents` constituée des champs suivants :
- *nom :* nom du continent (chaîne de caractères) ;
- *surface :* surface du continent en kilomètres carrés (entier).

– La table `Pays` constituée des champs suivants :
- *nom :* nom du pays (chaîne de caractères) ;
- *code_pays :* identifiant unique du pays (chaîne de caractères) ;
- *capitale :* capitale administrative du pays (chaîne de caractères) ;
- *population :* nombre d'habitants du pays (entier).

– La table `Inclusion` constituée des champs suivants :
- *code_pays :* identifiant unique du pays (chaîne de caractères) ;
- *continent :* nom du continent auquel appartient le pays (chaîne de caractères).

– La table `Frontieres` constituée des champs suivants :
- *code_pays1 :* identifiant unique du premier pays (chaîne de caractères) ;
- *code_pays2 :* identifiant unique du second pays (chaîne de caractères) ;
- *longueur :* longueur en kilomètres de la frontière entre *pays1* et *pays2* (nombre flottant strictement positif).

On a toujours *code_pays1*<*code_pays2* pour l'ordre lexicographique, ce qui assure que chaque frontière n'apparaît qu'une fois dans la table `Frontieres`.

Q28. Écrire une requête SQL permettant de récupérer le code du pays nommé 'France'.

Q29. Écrire une requête SQL permettant d'obtenir les noms des pays appartenant au continent 'Europe'.

Dans les deux dernières questions, on admet que le code du pays nommé 'France' est 'F' et on peut utiliser librement cette information.

Q30. Écrire une requête SQL permettant de récupérer la longueur totale de la frontière du pays nommé 'France'.

Q31. Écrire une requête SQL permettant de récupérer les noms des pays frontaliers du pays nommé 'France'.

ANNEXE 1 - Base de données géographiques

Table `Continents`

nom	surface
'Asie'	44579000
'Europe'	9938000
...	...

Table `Pays`

nom	code_pays	capitale	population
'Albanie'	'AL'	'Tirana'	3088385
'Algerie'	'DZ'	'Alger'	44487616
...	...	...	...

Table `Inclusion`

code_pays	continent
'AL'	'Europe'
'DZ'	'Afrique'
...	...

Table `Frontieres`

code_pays1	code_pays2	longueur
'AL'	'GR'	282.8
'AL'	'MK'	151.5
...	...	...

ANNEXE 2 - Rappels de syntaxe Python

Test d'appartenance	`>>> 2 in [1,2,3,4]`
	`True`
	`>>> 2 in [1,3,4]`
	`False`

Définir une liste	`>>> L=[1,2,3]`
	`>>> L[0]`
	`1`

Ajouter un élément à la fin d'une liste	`>>> L=[1,2,3]`
	`>>> L`
	`[1,2,3]`
	`>>> L.append(5)`
	`>>> L`
	`[1,2,3,5]`

Ajouter tous les éléments d'une liste `L1` à la fin d'une liste `L`	`>>> L=[6,8,7]`
	`>>> L1=[-1,-2]`
	`>>> L.extend(L1)`
	`>>> L`
	`[6,8,7,-1,-2]`

Obtenir les combinaisons d'une taille donnée d'une liste	`>>> from itertools import combinations`
	`>>> for C in combinations([0,1,2,3,4],3):`
	`  print(C)`
	`(0, 1, 2)`
	`(0, 1, 3)`
	`(0, 1, 4)`
	`(0, 2, 3)`
	`(0, 2, 4)`
	`(0, 3, 4)`
	`(1, 2, 3)`
	`(1, 2, 4)`
	`(1, 3, 4)`
	`(2, 3, 4)`

FIN

IMPRIMERIE NATIONALE – 24 1114 – D'après documents fournis