Coloration d'un graphe

Introduction

Vous pouvez voir ci-contre une carte de France représentant les 12 régions métropolitaines (sans la Corse pour des raisons de simplification du problème).
On se demande s'il est possible de colorier la carte en respectant les contraintes suivantes :

- Utiliser un **minimum** de couleurs ;
- Faire en sorte que deux régions limitrophes (ayant une frontière commune) soient coloriées de deux couleurs **différentes**.

La figure utilise 12 couleurs, une par région ; on peut faire mieux. Il y a des solutions optimales à 4 couleurs, d'autres qui s'en approchent. Nous allons voir plusieurs algorithmes résolvant cette problématique.

Partie 1 : Représentation et implémentation par un graphe

Pour tenter de résoudre ce problème, nous n'allons pas travailler directement sur la carte, mais sur un graphe associé à celle-ci.
Chaque région sera un sommet du graphe et, si deux régions sont limitrophes, il existera alors une arête entre ces deux sommets. Pour nommer les sommets nous utiliserons les codes suivants :

Région	Bretagne	Pays de Loire	Nouvelle Aquitaine	Occitanie	Provence-Alpes-Côte d'Azur	Auvergne-Rhône-Alpes	Bourgogne Franche-Comté	Centre-Val de Loire	Île de France	Grand-Est	Hauts-de-France	Normandie
Code	BRE	PLO	NAQ	OCC	PAC	ARA	BFC	CVL	IDF	GES	HDF	NOR

1. Représentez le graphe correspondant à la carte en utilisant les codes fournis pour les étiquettes des sommets.

2. Implémentez le graphe dans une variable `regions` à l'aide d'un dictionnaire.

3. Écrivez en Python une fonction `voisins` telle que `voisins(g,k)` renvoie une liste contenant les voisins du sommet `k` dans le graphe `g`.

Partie 2 : Coloration séquentielle

Une première approche pour colorer le graphe est de prendre ses sommets les uns après les autres afin de leur affecter une couleur, tout en veillant à ce que deux sommets adjacents n'aient jamais la même couleur : c'est l'algorithme de coloration séquentielle. On propose le code ci-dessous qui utilise la fonction `voisins` programmée à la question précédente :

```
def coloration(g) :
    """Renvoie une coloration du graphe G : version séquentielle"""

    couleur = ['Rouge', 'Bleu', 'Vert', 'Jaune', 'Noir', 'Blanc']
    coloration_sommets = {s_i: None for s_i in g}
    for s_i in g:
        couleurs_voisins_s_i = [coloration_sommets[s_j] for s_j in voisins(g, s_i)]
        k = 0
        while couleur[k] in couleurs_voisins_s_i:
            k = k + 1
        coloration_sommets[s_i] = couleur[k]
    return coloration_sommets
```

On modélise maintenant un graphe avec le dictionnaire `G2` ci-dessous :

```
G2 = {"A": ["C"], "B": ["D"], "C": ["A", "D"], "D": ["B", "C"]}
```

4. Représentez ce graphe.

5. Que va renvoyer l'appel de la fonction `coloration(G2)` ?

6. Montrez qu'il existe une solution avec moins de couleurs.

Partie 3 : L'algorithme de Welsh et Powell

Il s'agit maintenant d'utiliser l'algorithme de coloration séquentiel avec un ordre judicieux, en vue d'obtenir une coloration valide la plus « acceptable » possible. L'algorithme de Welsh et Powell consiste ainsi à colorer séquentiellement le graphe en visitant les sommets **par ordre de degré décroissant**. L'idée est que les sommets ayant beaucoup de voisins sont plus difficiles à colorer : il faut donc les colorier en premier.
On rappelle que le degré d'un sommet est le nombre d'arêtes issues de ce sommet, c'est-à-dire son nombre de voisins.
On souhaite implémenter en Python une fonction `tri_sommets` telle que `tri_sommets(g)` renvoie la liste des étiquettes des sommets du graphe `g` passé en paramètre triés par degrés décroissants.

7. Complétez les lignes de code suivantes :

```
def tri_sommets(g) :
    """Renvoie la liste des sommets, triée par degré décroissant"""
    sommets = [sommet for sommet in g]
    for i in range(...):
        i_sommet_max = i
        degre_sommet_max = len(g[sommets[i]])
        for j in range(..., len(sommets)):
            if len(g[sommets[j]]) > degre_sommet_max:
                i_sommet_max = ...
                degre_sommet_max = ...
        tmp = sommets[i]
        sommets[i] = sommets[i_sommet_max]
        sommets[i_sommet_max] = ...
    return sommets
```

8. Quelle sera la liste renvoyée par l'appel de fonction `tri_sommets(regions)` où l'on passe en paramètre le dictionnaire de la question **2** ?

9. Quel est le type de tri utilisé ?

10. Citer un exemple de tri plus efficace en temps.

11. Reprenez l'algorithme séquentiel fourni dans la partie pour implémenter l'algorithme de Welsh-Powell.