TP 10 - Graphes - Partie 3
Colorations d'un graphe

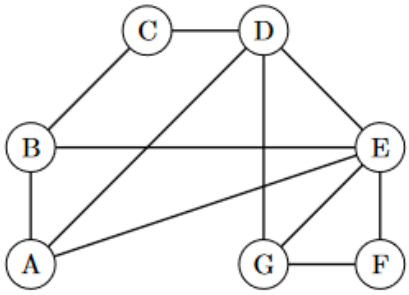
On considère un graphe connexe non orienté et sans boucle. Colorer ce graphe, c'est associer à chaque sommet une couleur. Le plus souvent, on s'intéresse à la coloration *propre* d'un graphe, qui est de telle sorte que deux sommets adjacents n'aient pas la même couleur. Le nombre minimal de couleurs nécessaires pour colorer proprement un graphe G est appelé son nombre chromatique, et il sera noté dans la suite $\chi(G)$.

Bien sûr, le nombre chromatique est majoré par le nombre de sommets du graphe, mais ce résultat n'a que peu d'intérêt. En fait, pour les graphes dits planaires, c'est à dire qui peuvent se représenter sans croisement entre les arêtes, le résultat le plus connu est le théorème des quatre couleurs qui stipule que le nombre chromatique est majoré par 4. Il est donc possible de colorer la carte de France en utilisant seulement 4 couleurs !

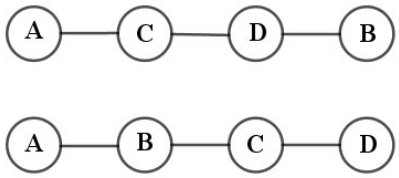
Dans ce TP, nous mettons en œuvre un algorithme glouton pour colorer (proprement) un graphe. Pour des raisons pratiques, les couleurs sont assimilées aux entiers naturels. L'algorithme est décrit ci-après.

> Pour chaque sommet s :
> - on regarde les couleurs attribuées aux voisins de s ;
> - on colore s en utilisant le plus petit entier naturel n'appartenant pas à l'ensemble précédent.

1. Calculer $\chi(K_{15})$, où K_{15} est un graphe complet d'ordre 15.
2. Appliquer à la main l'algorithme au graphe ci-dessous, en traitant les sommets dans l'ordre alphabétique. :



3. Appliquer l'algorithme aux deux graphes suivants, en traitant les sommets dans l'ordre alphabétique. Que remarque-t-on ? Quel est le nombre chromatique ?



4. Il s'agit en premier lieu d'écrire une fonction Python prenant en argument une liste d'entiers naturels (les couleurs), et qui renvoie le plus petit entier naturel n'appartenant pas à cette liste. Pour ce faire, si L est la liste en argument, on parcourt les entiers de 0 à $\max(L)+1$, et on renvoie l'entier i s'il n'est pas dans L. On pourra utiliser la commande `not in` qui permet de vérifier si un entier n'est pas dans L. Recopier et compléter le programme suivant :

```
def couleur_dispo(L):
    for i in .......
        if .......
            return .....
```

5. (a) Une deuxième fonction qui sera utile est la suivante, `liste_couleurs_voisins`, qui prend en argument la matrice d'adjacence G du graphe, le vecteur (ou la liste) des couleurs C, et un sommet s, et qui renvoie la liste des couleurs des voisins de s. Recopier et compléter le programme ci-dessous :

```
def liste_couleurs_voisins(G,C,s):
    n = len(G)
    L = []
    for i in range(.......):
        if ....... :  # Si i est voisin de s
            L.append(.......) # On rajoute la couleur de i
    return .......
```

   (b) Proposer une simplification du code de la question précédente en utilisant la syntaxe des listes définies en compréhension.

6. Il reste à conclure. La fonction `colore_graphe` ci-dessous prend en argument la matrice d'adjacence G du graphe, et renvoie un vecteur contenant la couleur de chaque sommet. On initialisera le vecteur couleurs avec des -1.

```
def colore_graphe(G):
    n = len(G)
    C = .......
    for i in range(n):
        C[i] =  .......
    return C
```

7. Reprendre les questions 1, 2 et 3 en faisant appel au programme Python créé dans les question précédentes.
8. *Bonus.* Quel est le nombre chromatique du graphe de la question 2 ?