DS n°6

Important : vous pouvez utiliser les fonctions des questions précédentes, même si vous ne les avez pas toutes implémentées.

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)`

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.

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,

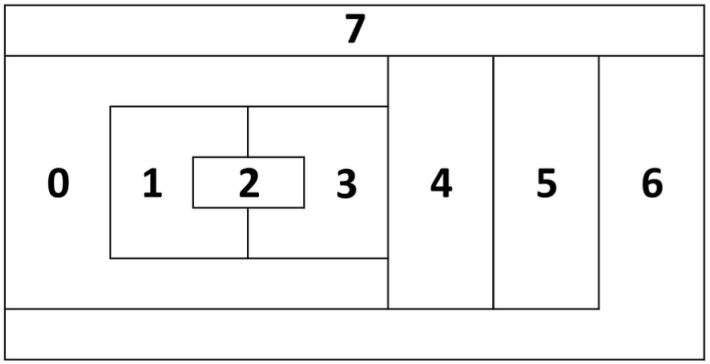

Figure 1 - Exemple d'une carte de pays

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

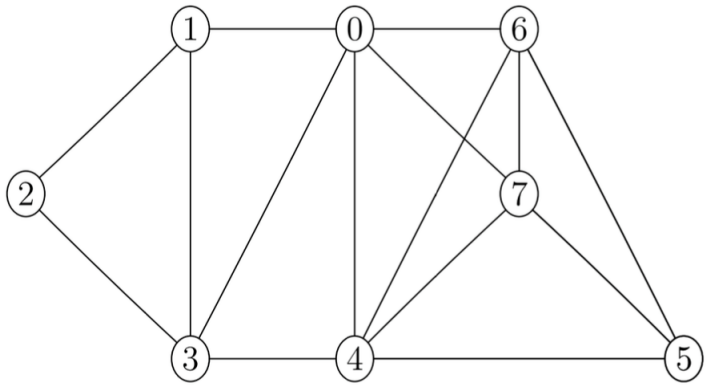

Figure 2 - Graphe G_{ex} associé à la carte de pays de la **figure 1**

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

1. Un graphe peut être représenté par une matrice d'adjacence. On donne ci-contre la matrice d'adjacence pré-remplie du graphe G_{ex}. La compléter et expliquer le processus de construction.

$$M = \begin{pmatrix} & & & & 1 & 0 & 1 & 1 \\ & & & & 0 & 0 & 0 & 0 \\ & & & & 0 & 0 & 0 & 0 \\ & & & & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

2. 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 ici 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.

3. 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.

4. Rappeler la définition du degré `s` d'un sommet d'un graphe `G`. Recopier le tableau ci-dessous des degrés du graphe G_{ex}.

Sommet	0	1	2	3	4	5	6	7
Degré								

I.2 Tester si une coloration est valide

5. 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).

6. Ecrire 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 et `False` sinon.

7. 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.

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

9. 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.

```
def colore_sommet(C,s,LA):
    # on determine la liste des couleurs des voisins
      de s deja colores
    coul_vois = []

    # coul_vois est maintenant determinee et on
      recherche la plus petite couleur, notee num_coul,
       absente de coul_vois

    # la valeur num_coul trouvee devient
    la couleur du sommet s
```

10. A 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]`
11. 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...

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 cidessus 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).

12. Ecrire 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]`
13. Ecrire une fonction `degre_max(LA)` renvoyant le maximum des degrés de la liste d'adjacence `LA`.
14. 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 proposé, l'appel `ranger(LA)` renverra la liste `[[],[4],[0],[1,2,3]]`. On pourra utiliser le fait que
    `[ 4*[] ] = [[],[],[],[],[]]`
15. Ecrire 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 `[3,2,1,0,4]` ou `[1,2,3,0,4]`.
16. Pour un graphe à `n` sommets, quelle est la complexité temporelle de la fonction `trier_sommets` dans le pire des cas ?
17. Ecrire 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 ?
18. Pour le graphe G_{ex} de la **figure 2**, donner la liste `C` des couleurs renvoyée par la fonction `colorer3`.