Mathématiques & Informatique - F. Gaunard
`http://frederic.gaunard.com`
Lycée Voltaire, Paris 11e.

# 3 Travaux Pratiques : Dictionnaires et graphes

**Exercice 1.** ***Solution*** **Coloriage en 4 couleurs**

L'objectif de cet exercice est de trouver un coloriage en 4 couleurs de la carte des régions de France métropolitaine.

La carte des régions de France est alors assimilé à un graphe non orienté dont les sommets sont les régions et deux sommets sont voisins si les régions sont adjacentes.

**1.** Implémenter ce graphe que l'on nommera `regions`.
*On aura récupéré le fichier en ligne : `https://frederic.gaunard.com/2425/regions.py`.*

Le principe de l'algorithme et le suivant :

- ✗ les couleurs seront représentées par des entiers numérotés à partir de 0;
- ✗ on prend un sommet du graphe au hasard, on regarde les couleurs déjà données à ses voisins, et on lui donnera comme couleur la plus petite valeur non-affectée à un de ses voisins.

**2.** Écrire une fonction `min_couleur_dispo` prenant trois arguments :

- ✗ `G` : un dictionnaire, représentant le graphe
- ✗ `couleurs` : un dictionnaire, représentant des sommets déjà coloriés
- ✗ `v` : une valeur, représentant un sommet du graphe

et qui renvoie la valeur de couleur la plus petite non déjà utilisée dans `couleur` pour colorier les voisins de `v` dans `G`.

*On commence avec une liste de couleurs non disponibles vide. On parcourt les voisins de v, si un voisin est déjà colorié (c'est à dire dans le dictionnaire `couleurs`), sa couleur associée dans le coloriage est ajoutée à la liste des couleurs qui ne sont pas disponible.*
*Une fois cette liste remplie, on va récupérer le plus petit entier qui n'est pas dans la liste en parcourant les entiers. Easy.*

```
def min_couleur_dispo(G, couleurs, v):
    pas_dispo=[]
    for voisin in G[v]:
        if voisin in couleurs:
            pas_dispo.append(couleurs[voisin])
    k=0
    while k in pas_dispo :
            k=k+1
    return k
```

**3.** En reprenant l'algorithme de parcours en largeur d'un graphe, écrire une fonction `coloriage_graphe` qui prend en argument un graphe et une région de départ et renvoie un *tuple* contenant :

- ✗ le nombre de couleurs utilisées ;
- ✗ un dictionnaire affectant à chaque région sa couleur.

*Il s'agit de reprendre l'algorithme de parcours en largeur. Au lieu des files à voir, on les nomme à colorier.*
*Le dictionnaire des sommets coloriés prend pour clés les sommets mais pour valeur associé la couleur du coloriage.*

*Lorsqu'on ajoute un sommet dans la liste des sommets déjà coloriés, on le colorie avec la "plus petite" couleur disponible.*

```
def color(D, s):
    FileAcolorier=[s]
    DejaColorie={}
    DejaColorie[s]=0
    while len(FileAcolorier)>0:
        s2=FileAcolorier.pop(0)
        for v in D[s2]:     #D[s2] est la liste des sommets voisins de s2
            if not v in DejaColorie:
                FileAcolorier.append(v)
                couleur=min_couleur_dispo(D, DejaColorie, v)
                DejaColorie[v]=couleur
    return (max(DejaColorie.values())+1, DejaColorie)
```

4. Pouvez-vous donner un coloriage utilisant exactement 4 couleurs ? Quid de la carte des 48 états américains ?
*Il se trouve que oui !*

```
>>> color(regions, 'Aquitaine')
(4, {Aquitaine: 0, Auvergne: 3, Bourgogne: 1, Bretagne: 0, Centre: 2, Grand-Es
t: 2, HdF: 1, IdF: 0, Loire: 1, Normandie: 3, Occitanie: 1, PACA: 0})
>>> color(USA, 'Washington')
(4, {Alabama: 2, Arizona: 1, Arkansas: 0, California: 0, Colorado: 2, Connecti
cut: 1, Delaware: 0, Florida: 1, Georgia: 0, Idaho: 2, Illinois: 0, Indiana: 2
, Iowa: 1, Kansas: 1, Kentucky: 1, Louisiana: 2, Maine: 1, Maryland: 3, Massac
husetts: 2, Michigan: 1, Minnesota: 0, Mississippi: 1, Missouri: 2, Montana: 0
, Nebraska: 0, Nevada: 3, New Hampshire: 0, New Jersey: 2, New Mexico: 0, New
York: 0, North Carolina: 1, North Dakota: 1, Ohio: 0, Oklahoma: 3, Oregon: 1,
Pennsylvania: 1, Rhode Island: 0, South Carolina: 2, South Dakota: 2, Tennesse
e: 3, Texas: 1, Utah: 0, Vermont: 1, Virginia: 0, Washington: 0, West Virginia
: 2, Wisconsin: 2, Wyoming: 1})
```

## Exercice 2. *Solution* — Correction automatique

Un enseignant propose des QCM à ses élèves. La solution de ce QCM est représentée par un dictionnaire `solution` dont les clés sont des entiers (représentant les questions) de 1 à 20 et les valeurs un caractère.
Une bonne réponse rapporte un point, une mauvaise réponse enlève un quart de point. Une absence de réponse n'ajoute ni n'enlève aucun point. La note finale est un nombre entre 0 et 20.

Ainsi, chaque *copie* rendue est aussi représentée par un dictionnaire définie de la même manière que précédemment sauf que si l'élève ne répond pas à une question, la valeur correspondante n'apparait pas dans le dictionnaire.
Enfin, l'ensemble des copies est un nouveau dictionnaire `paquet` dont les clés sont les étudiants (représentés par des chaînes de caractère) et les valeurs les dictionnaires modélisant leurs copies.

Écrire une fonction `correction` qui prend en argument les dictionnaires `solution` et `paquet` et renvoie un dictionnaire qui associe chaque étudiant à sa note.

```
def correction(solution, paquet):
    notes={}
    for etudiant in paquet :
        N=0
        for question in paquet[etudiant] :
            if paquet[etudiant][question]==solution[question]:
                N=N+1
            else :
                N=N-0.25
        notes[etudiant]=max(0, N)
    return notes
```

## Exercice 3. *Solution* — Suite de Syracuse

Pour tout entier $p \in \mathbb{N}^*$, on définit la suite de Syracuse associée par $u_0 = p$ et $\forall n \in \mathbb{N}, u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ pair} \\ 3u_n + 1 & \text{sinon} \end{cases}$.

On admet que pour toute valeur raisonnable de $p$ (disons $p \leq 5 \times 2^{60}$), la suite finit par boucler sur $1, 4, 2$.
On note $f(p)$ le plus petit $n$ tel que $u_n = 1$.

**1.** Quelle est la plus grande valeur de $f(p)$ pour $1 \leq p \leq 10000$ ?
*On calcule les termes de la suite de Syracuse à l'aide d'un dictionnaire jusqu'à tomber sur* 1 *en faisant varier le premier terme entre* 1 *et* 10000*. On regarde ensuite si le nombre de termes à calculer est plus grand que le plus grand calculé jusqu'à alors (stocké dans la variable* `f` *et si oui, on écrase. On obtient* 350*.*

```
f=1
F={}
for p in range(1, 10**5+1):
    d={0 : p}
    k=0
    while d[k] != 1 :
        if d[k]%2==0 :
            d[k+1]=d[k]//2
        else :
            d[k+1]=3*d[k]+1
        k=k+1
    f=max(f,k)
    F[p]=k
print(f)
```

**2.** Combien d'entiers $p \leq 10^5$ vérifient $f(p) \leq 30$ ?

```
L=[]
for p in F :
    if F[p] <= 30 :
        L.append(p)
print(len(L))
```

*Cette exécution affiche une valeur de* 1879*.*

## Exercice 4. *Solution* — Secret Santa

*Idéalement on ne se fait pas de cadeau à soi-même...*

```
import numpy.random as rd
def secret_santa(L):
    cadeaux={}
    liste1=L.copy() # gens qui doivent faire un cadeau
    liste2=L.copy() # gens qui doivent recevoir un cadeau
    while len(liste2)>0 :
        n=len(liste2)
        pere_noel=liste1[-1]
        k=rd.randint(0,n)
        enfant_gate=liste2[k]
        while pere_noel == enfant_gate :
            k=rd.randint(0,n)
            enfant_gate=liste2[k]
        cadeaux[pere_noel]=enfant_gate
        liste1.pop()
        liste2.pop(k)
    return cadeaux
```