-*- coding: utf-8 -*-

In [ ]:
```
"""
Created on Tue Feb  9 09:57:42 2021

@author: kourf
"""
```

In [ ]:
```
from graphviz import Graph, Digraph
from collections import deque
from PIL import Image
import os
```

In [ ]:
```
"""
Je suis parti sur la classe SommetColore mais il aurait probablement été plus simple d'indiquer l'état visité ou découvert dans le dictionaire des sommets :
        (decouvert: bool, visite: bool, successeurs: list).
"""
```

In [ ]:
```
COULEUR_INITIALE = '#FFFFFF'
COULEUR_DECOUVERT = '#FF7F59'
COULEUR_VISITE = '#9F1818'
```

In [ ]:
```
class SommetColore:
    def __init__(self, etiquette):
        self._etiquette = etiquette
        self._couleur = COULEUR_INITIALE
        self.decouvert = False
        self.visite = False

    def marquer_decouvert(self):
        self._couleur = COULEUR_DECOUVERT
        self.decouvert = True

    def marquer_visite(self):
        self._couleur = COULEUR_VISITE
        self.visite = True

    def est_decouvert(self):
        return self.decouvert

    def est_visite(self):
        return self.visite

    def etiquette(self):
```

```
        return self._etiquette

    def couleur(self):
        return self._couleur

    def __repr__(self):
        return self._etiquette
```

In [ ]:

```
```

In [ ]:

```
class GraphColore:
    """
    Crée un objet graphe coloré à partir d'une liste de sommets ou d'un
    graphe donné sous la forme d'un dictionnaire
    """
    def __init__(self, sommets):
        self.dico_sommets = {v : SommetColore(v) for v in sommets}
        if type(sommets) == list:   # liste de sommets -> graphe sans arête
            self.sommets = {self.dico_sommets[v]: [] for v in sommets}
        else:                       # sommets est un dictionnaire
            self.sommets = {self.dico_sommets[v]: [self.dico_sommets[s]
                                                   for s in sommets[v]]
                            for v in sommets}
        self.n = n = len(sommets)

    def get_sommets(self):
        return list(self.sommets.keys())

    def successeurs(self, s):
        if not isinstance(s, SommetColore):
            s = self.dico_sommets[s]
        return self.sommets[s]

    def voisins(self, s):
        "Identique à la méthode successeurs"
        return self.successeurs(s)

    def est_successeur(self, s1, s2):
        if not isinstance(s1, SommetColore):
            s1 = self.dico_sommets[s1]
        if not isinstance(s2, SommetColore):
            s2 = self.dico_sommets[s2]
        return s2 in self.sommets[s1]


    def est_oriente(self):
        for s1 in self.sommets:
            for s2 in self.sommets[s1]:
                if s1 not in self.sommets[s2]:
                    return True
        return False

    def dessiner_graphe(self, nombre_images=1, oriente=None):
```

```
        if oriente==None:
            oriente = self.est_oriente()
        nom_fichier = 'graph' + str(nombre_images)
        if oriente:
            g = Digraph('G', filename=nom_fichier, format='gif')
        else:
            g = Graph('G', filename=nom_fichier, format='gif')
        g.attr(rankdir="LR")
        for s1 in self.sommets:
            for s2 in self.sommets[s1]:
                if oriente or s2.etiquette() <= s1.etiquette():
                    g.node(s1.etiquette(), style='filled',
fillcolor=s1.couleur())
                    g.node(s2.etiquette(), style='filled',
fillcolor=s2.couleur())
                    g.edge(s1.etiquette(), s2.etiquette())
        g.render()
        return g
```

In [ ]:

In [ ]:
```
gtemp_1 = {'A': ['B', 'E', 'G'],
           'B': ['A', 'C', 'E', 'F'],
           'C': ['B', 'D', 'G'],
           'D': ['C', 'E', 'F'],
           'E': ['A', 'B', 'D', 'H'],
           'F': ['B', 'D', 'H'],
           'G': ['A', 'C'],
           'H': ['E', 'F']
           }
graphe_1 = GraphColore(gtemp_1)
```

In [ ]:
```
gtemp_2 = {'A': ['B', 'E', 'G'],
           'B': ['A', 'C', 'D'],
           'C': ['B', 'J'],
           'D': ['B', 'F'],
           'E': ['A'],
           'F': ['D', 'H', 'I'],
           'G': ['A'],
           'H': ['F'],
           'I': ['F'],
           'J': ['C'],
           }
graphe_2 = GraphColore(gtemp_2)
```

In [ ]:
```
gtemp_3 = {'A': ['B', 'G'],
           'B': ['C', 'E'],
           'C': ['D'],
           'D': ['F'],
           'E': ['A', 'D', 'H'],
           'F': ['B', ],
```

```
            'G': ['C'],
            'H': ['F']
            }
graphe_3 = GraphColore(gtemp_3)
```

Parcours en largeur

In [ ]:
```
def parcours_largeur(G, s):
    nombre_images = 1
    s = G.dico_sommets[s]  # conversion de s en instance de SommetColore
    file = deque([s])
    s.marquer_decouvert()
    while file:
        s = file.popleft()
        s.marquer_visite()
        G.dessiner_graphe(nombre_images)
        nombre_images += 1
        for v in G.voisins(s):
            if not v.est_decouvert():
                v.marquer_decouvert()
                file.append(v)
                G.dessiner_graphe(nombre_images)
                nombre_images += 1
```

Parcours en profondeur itératif

In [ ]:
```
def parcours_profondeur_iter(G, s):
    nombre_images = 1
    s = G.dico_sommets[s]  # conversion de s en instance de SommetColore
    pile = [s]
    s.marquer_decouvert()
    while pile:
        s = pile.pop()
        if not s.est_visite():
            s.marquer_visite()
            G.dessiner_graphe(nombre_images)
            nombre_images += 1
            for v in G.voisins(s):
                if not v.est_visite():
                    pile.append(v)
                    v.marquer_decouvert()
                    G.dessiner_graphe(nombre_images)
                    nombre_images += 1
```

Parcours en profondeur récursif

In [ ]:
```
def parcours_profondeur(G, s, nombre_images=None):
```

```
    if nombre_images==None:
        nombre_images=[1]
        if not isinstance(s, SommetColore):
            s = G.dico_sommets[s]
    s.marquer_visite()
    G.dessiner_graphe(nombre_images[0])
    nombre_images[0] += 1
    for v in G.voisins(s):
        if not v.est_visite():
            parcours_profondeur(G, v, nombre_images)
```

Parcours chemin donné

In [ ]:
```
def parcours_chemin(G, chemin):
    for i in range(len(chemin)):
        s = chemin[i]
        if not isinstance(s, SommetColore):
            s = G.dico_sommets[chemin[i]]
        s.marquer_visite()
        G.dessiner_graphe(i+1)
```

Parcours hamiltonien

In [ ]:
```
def parcours_ham(G, s, solutions, chemin=None):
    if chemin==None:
        chemin = [s]
    if len(chemin) == G.n:
        solutions.append(chemin)
    for v in G.voisins(s):
        if v not in chemin:
            parcours_ham(G, v, solutions, chemin + [v])
```

In [ ]:
```
def parcours_hamiltonien(G):
    solutions = []
    for s in G.get_sommets():
        parcours_ham(G, s, solutions)
    if solutions:
        parcours_chemin(G, solutions[0])
    else:
        print("Pas de chemin hamiltonien")
```

In [ ]:
```

```

Création du gif

```

```

In [ ]:
```
def creation_parcours_colore(graphe, parcours, sommet=None,
nom_fichier='graphe.gif'):
    """
    Crée un gif représentant le parcours choisi :
    - parcours_largeur, parcours_profondeur_iter ou parcours_profondeur :
        préciser le sommet de départ
    - parcours_chemin : indiquer un chemin sous forme d'une liste de sommets
        à la place du sommet de départ
    - parcours_hamiltonien : ne pas indiquer de sommet de départ

    """
    try:
        os.mkdir('dossier_temp')
    except:
        pass
    if os.path.exists("graphe.gif"):
        os.remove("graphe.gif")
    os.chdir('dossier_temp')
    if sommet==None:
        parcours(graphe)
    else:
        parcours(graphe, sommet)
    nombre_images = len(os.listdir())//2
    if nombre_images == 0:
        os.chdir('..')
        os.rmdir('dossier_temp')
        return
    images = [0]*nombre_images
    image_finale = Image.open('graph1.gif')
    for i in range(nombre_images):
        images[i] = Image.open('graph'+str(i+1)+'.gif')
    image_finale.save(nom_fichier, save_all=True, append_images=images,
optimize=False, duration=500, loop=0)
    image_finale.close()
    for i in range(nombre_images):
        images[i].close()
        os.remove('graph'+str(i+1)+'.gif')
        os.remove('graph'+str(i+1))
    os.rename('graphe.gif', '../graphe.gif')
    os.chdir('..')
    os.rmdir('dossier_temp')
```

In [ ]:
```
creation_parcours_colore(graphe_3,parcours_profondeur, sommet="A",
nom_fichier='graphe.gif')
```

In [ ]:
```

```