TP 19 graphes (suite)

Télécharger sur Cahier de Prépa le fichier nommé "TP 19 : module graphe à télécharger", enregistrez-le dans votre dossier personnel sous le nom `graphe.py`*. Dans Spyder, ouvrez ce fichier, et ouvrez aussi le fichier* `TP18.py` *que vous aviez sauvegardé de la dernière fois.*

1. Le module `graphe` a été modifié par rapport à la semaine dernière. Lisez la nouvelle fonction `__init__` et comprenez son fonctionnement.
2. Dans votre script `TP18.py` importez le module `graphe`, puis créez un graphe ayant 5 noeuds `'a'`, `'b'`, `'c'`, `'d'`, `'e'` avec les liens $a \to b, a \to c, c \to e, d \to c, d \to e, e \to a, e \to b$.
3. Lisez et comprenez la méthode `ajout_noeud` du module `graphe`. Utilisez la pour ajouter un noeud `'f'` à votre graphe, avec les liens $e \to f, f \to a, f \to d$.
4. Créez une méthode `ajout_liens(self, liens_depart, liens_arrivee)` qui ajoute des liens au graphe sans changer les noeuds du graphe. `liens_depart` et `liens_arrivee` seront des listes de même taille contenant les noeuds de départ et d'arrivée des liens à ajouter.
5. Testez la méthode `est_oriente` sur votre graphe.
6. Calculez à l'aide de la matrice de votre graphe le nombre de chemins de longueur 5 reliant `'c'` à `'d'`. Vous pourrez vous aider de la commande `matrix_power` du module `numpy.linalg` pour calculer les puissance de la matrice d'adjacence.
7. Dans le module `graphe`, créez une nouvelle classe `graph_pondere(graph)` qui gère les graphes pondérés en héritant de la classe `graph`. Pour ce faire, dans le dictionnaire `dico` associé à un graphe pondéré, les entrées seront des dictionnaires au lieu d'ensembles. Par exemple un graphe pondéré sera crée par la commande suivante :

```
G = graphe.graph_pondere({'a' : {'b' : 3, 'c' : 2}, 'b' : {'c' : 1}, 'c' : {}})
```

   Il faut adapter au moins la méthode `__init__`.
8. Adaptez aussi la méthode `ajout_noeud` et la méthode `matrice`.
9. *On cherche maintenant le poids total du plus court chemin partant d'un sommet donné et allant à un autre sommet du graphe. Pour ceci on va coder l'algorithme de Bellman-Ford. On commence avec un dictionnaire nommé π contenant les distances du sommet fixé (appelons le s) aux autres sommets. La distance de s à s est bien sûr 0, et on met la distance aux autres sommets à $+\infty$ au départ : on utilise pour ça* `float('inf')` *qui se comporte comme l'infini pour Python : c'est un objet supérieur strictement à tout réel, et si on ajoute un réel à* `float('inf')` *on obtient toujours* `float('inf')`*. On parcourt ensuite plusieurs fois tous les liens du graphe en mettant à jour au fur et à mesure les distances obtenues.*

   pour tous les sommets k du graphe :
   $\quad \pi(k) \leftarrow +\infty$
   $\pi(s) \leftarrow 0$
   $\pi_0 \leftarrow \{\}$
   tant que $\pi \neq \pi_0$:
   $\quad \pi_0$ est une copie de π
   $\quad$ pour chaque lien u - v du graphe :
   $\quad\quad$ si $\pi(v) > \pi(u) + poids(u, v)$:
   $\quad\quad\quad \pi(v) \leftarrow \pi(u) + poids(u, v)$
   retourner π

   Dans votre module `graphe`, codez une fonction `bellman(graph, source)` qui renvoie le dictionnaire `pi` donné par cet algorithme. Testez le sur le graphe de la question précédente.
   Vous aurez besoin de la méthode `deepcopy` *du module* `copy` *pour faire une copie de π.*

Formulaire :

Vous aurez besoin des commandes suivantes (dans le désordre)

```
### LES MODULES USUELS
from math import *
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rd

### LES VARIABLES ALEATOIRES
rd.randint(a,b,N)
rd.random(N)
rd.binomial(n,p,N)
rd.geometric(p,N)
rd.poisson(lambda,N)

## LES LISTES
np.linspace(a,b,n)
for compteur in liste :
for compteur in range(a,b):
[ f(compteur) for compteur in liste]
[ f(compteur) for compteur in range(a,b)]

### COMMANDES CLASSIQUES
while condition:
def nom_fonction( variables ) :
return resultat
print( 'message', variable )
if condition :
else :
log(x)
plt.plot( abscisses , ordonnees )

### CLASSES EN PYTHON
class nom_de_classe :
class nom_de_classe(classe_d_heritage):
def __init__(self, autres_parametres):
def __str__(self):

### MATRICES EN PYTHON
np.array([[...],...,[...]])
A.shape
np.zeros([p,n])
np.eye(n)
np.ones([p,n])
A.dot(B)
np.dot(A,B)
A.transpose()
A.all()
```