import numpy as np

### Exercice 1 : fonctions élémentaires

G1=[]

ligne = [0,1,1,0,1]

G1.append(ligne)

ligne = [1,0,1,0,0]

G1.append(ligne)

ligne = [1,1,0,1,1]

G1.append(ligne)

ligne = [0,0,1,0,1]

G1.append(ligne)

ligne = [1,0,1,1,0]

G1.append(ligne)

G1=np.array(G1)

print("La matrice d'adjacence de G1 sous forme de liste est :\n",G1)

print("La matrice d'adjacence de G1 sous forme de tableau Numpy est :\n",G1)

def sont\_egaux(A,B):

 n,p=len(A),len(A[0]) # respectivement : nb de lignes, nb de colonnes

 for i in range(n):

 for j in range(p):

 if A[i,j]!=B[i,j]:

 return False # joue le rôle d'un "break".

 return True # structure classique en Python : le True à la fin.

def est\_complet(M):

 n=len(M)

 for i in range(n):

 if M[i][i]!=0:

 return False

 for j in range(n):

 if i!=j and M[i][j]!=1: # attention, par "or", mais "and"

 return False

 return True

# OU, si l'on s'autorise les tableaux préprogrammés :

def est\_complet(M):

 n=len(M)

 J=np.ones(n)-np.eye(n)

 return sont\_egaux(M,J)

""" Remarque : la fonction de test d'égalité" de 2 matrices numpy A et B est : (A==B).all() ou bien np.equal(A,B).

"""

### Exercice 2 : conversion liste d'arcs -> matrice d'adjacence

def conversion(n,A):

 M=np.zeros((n,n))

 for couple in A:

 M[couple[0],couple[1]]=1

 M[couple[1],couple[0]]=1

 return M

def conversion2(n,A):

 #M=np.array([[0]\*n]\*n) à ne pas faire, à cause des pbs d'adressage.

 M=np.zeros((n,n))

 for [i,j] in A:

 M[i,j]=1

 M[j,i]=1

 return(M)

# ou bien encore :

def conversion3(n,A):

 M=np.zeros((n,n))

 for c in A:

 [i,j]=c # et pas c=[i,j], qui serait une redéfinition de c.

 M[i,j]=1

 M[j,i]=1

 return M

A2=[[0,1], [1,0], [1,2], [2,3], [3,1]]

G2=conversion(5,A2)

print("La matrice d'adjacence de A2 est :\n",G2)

### Exercice 3 : pondération

def estPondere(M):

 n=len(M)

 for i in range(n):

 for j in range(n):

 if M[i,j]!=0 and M[i,j]!=1: # if not M[i,j] in {0,1}

 # OU : if M[i,j] not in {0,1}

 return True

 return False

print("Pondération de G1 :",estPondere(G1))

G3=np.array([[0,10,20,0,30,0],[10,0,30,50,10,0],[20,30,0,10,0,30],[0,50,10,0,0,10],[30,10,0,0,0,0], [0,0,30,10,0,0]])

print("Pondération de G3 :",estPondere(G3))

### Exercice 4 : Fonction degré - chaîne et cycle eulérien

def degre(M,i):

 #si sum est permis : return sum(M[i])

 #sinon avec un compteur :

 c=0

 for k in M[i]:

 if k!=0: # mieux que k==1, car envisagerait le cas où le graphe est pondéré.

 c=c+1

 return c

""" Voisins """

def voisins(M,i):

 L=[]

 for j in range(len(M[0])):

 if M[i,j]!=0:

 L.append(j)

 return L

"""- chaîne et cycle eulériens :"""

def cycleeulerien(M):

 for k in range(len(M)): # parcours des numéros de sommets

 if degre(M,k)%2 != 0:

 return False

 return True

# ou, plus compact... : sum([degre(M,x)==1 for x in range(ordre(M))])==0

print("Cycle eulérien pour la matrice T1 :", cycleeulerien(G1))

def chaineeulerienne(M):

 compteur=0

 for k in range(len(M)): # parcours des numéros de sommets

 if degre(M,k)%2==1:

 compteur=compteur+1

 return (compteur==2)

# ou : sum([degre(M,x)%2 for x in range(len(M))])==2

# famille : complexité dans le pire des cas en n\*\*n !!!

def famille(M,i):

 L=[i]

 n=len(M)

 for k in range(n-1):

 for x in L:

 L=L+voisins(M,x)# pb : des sommets vont apparaître plusieurs fois

 return list(set(L)) # pour supprimer les redondances (pas de répétition dans un ensemble)

def famille2(M,i):

 L=[i]

 n=len(M)

 for k in range(n-1):

 for x in L:

 for y in voisins(M,x):

 if y not in L: # pour prévoir le pb précédent

 L.append(y)

 return L

# test de connexité : l'ensemble des sommets reliés à 0 est-il le grpahe tout entier ? (O compris puisqu'il est voisin de son voisin)

def est\_connexe(M):

 return len(famille(M,0))== len(M) # expéditif ! Y auriez-vous pensé ?

""" Remarquez qu'on peut éviter le "long" programme :

def est\_connexe(M):

 n=len(M)

 if len(famille(M,0))== n:

 return True

 else:

 return False

"""

print(famille(G2,0))

print(famille2(G2,0))

print(est\_connexe(G2))

print(est\_connexe(G1))