```python
# -*- coding: utf-8 -*-

# ============================================================ #
# ============================================================ #
# Programme déterminant toutes les chaînes de longueur LC donnée #
# entre deux sommets donnés d'un graphe simple non orienté.      #
# JANVIER 2015 / Modifié DECEMBRE 2016                           #
# ============================================================ #
# ============================================================ #

# N'importons de numpy que ce dont nous avons réellement besoin !
from numpy import zeros
from numpy.linalg import matrix_power

# Fonction pour la construction de la matrice d'adjacence.
# Les arguments sont :
# (1) L'ordre du graphe
# (2) La liste des arêtes du graphe
def fMatAdj(n,LA):
    # n est le nombre de sommets du graphe
    # LA est une liste dont chaque élément est une liste comportant deux entiers
    # i et j si l'arête [S(i)S(j)] existe.
    Adj = zeros((n,n))
    # Construction de la matrice d'adjacence Adj. La validité de la liste LA est
    # vérifiée au fur et à mesure.
    for k in range(len(LA)):
        i, j = LA[k][0], LA[k][1]
        if len(LA[k]) > 2 or i == j or Adj[i,j] != 0:
            return ('Liste non valide !')
        else:
            Adj[i,j], Adj[j,i] = 1, 1
    return Adj

# Fonction de détermination des chaînes souhaitées.
# Les arguments sont :
# (1) Le numéro du premier sommet
# (2) Le numéro du second sommet
# (3) La longueur des chaînes souhaitées
def chain(s1,s2,LC):
    if LC == 1:
        if MA[s1,s2] == 0:
            # On se trouve dans une impasse !
            return
        else:
            # On a obtenu une chaîne (s1 et s2 sont adjacents).
            # On met à jour la liste C...
            C[-2] = s1
            # ... et on l'affiche !
            print(C)
            return
    else:
        C[LCinit - LC] = s1
        for i in range(n):
            if MA[s1,i] != 0:
                # Les sommets s1 et i sont adjacents. On cherche alors les
                # chaînes de longueur LC-1 entre i et s2 via un appel récursif !
                chain(i,s2,LC-1)

# PROGRAMME PRINCIPAL
# ===================

# Définition du graphe par ses arêtes.
# On travaille ici avec un petit graphe d'ordre 5.
n = 5
LA = [[0,2],[2,3],[3,1],[0,4],[2,4],[1,2]]

# Construction de la matrice d'adjacence.
```

```
MA = fMatAdj(n,LA)

# Saisie des données du problème par l'utilisateur
s1 = -1
while s1 < 0 or s1 >= n:
    s1 = int(input('Veuillez saisir le numéro du premier sommet : '))
s2 = -1
while s2 < 0 or s2 >= n:
    s2 = int(input('Veuillez saisir le numéro du deuxième sommet : '))
LCinit = 0
while LCinit < 1:
    LCinit = int(input('Veuillez saisir la longueur des chaînes entre ces sommets :
'))

# Détermination des chaînes cherchées
C = [s1 for i in range(LCinit)]
C.append(s2)
nc = matrix_power(MA,LCinit)[s1,s2]
if nc == 0:
    if LCinit == 1:
        if s1 == s2:
            print('\nPas de boucle entre le sommet',s1,'et lui-même !')
        else:
            print('\nLes sommets',s1,'et',s2,'ne sont pas adjacents !')
    else:
        print('\nAucune chaîne de longueur ' + str(LCinit) + ' ne relie ces deux
sommets.')
else:
    if nc == 1:
        print ('\nLa seule chaîne reliant ces deux sommets est : \n')
    else:
        print('\nLes ' + str(int(nc)) + ' chaînes sont : \n')
    chain(s1,s2,LCinit)

# ================
# FIN DU PROGRAMME
```