```python
import numpy as np
from collections import deque

##Exercice 1: matrices et listes d'adjacences
M1=np.array([[0,0,0,0,0,0,1],[0,0,1,1,1,0,0],[0,1,0,0,1,1,0],[0,1,0,0,1,0,0],
             [0,1,1,1,0,1,0],[0,0,1,0,1,0,0],[1,0,0,0,0,0,0]])

L1=[[6],[2,3,4],[1,4,5],[1,4],[1,2,3,5],[2,4],[0]]

M2=np.array([[0,0,0,0,1],[1,0,1,0,0],[0,1,0,0,1],[0,1,1,0,0],[0,0,0,1,0]])

L2=[[4],[0,2],[1,4],[1,2],[3]]

##Exercice 2: Conversion matrices/listes d'adjacences
def MatriceToListe(M):
    n=len(M)
    L=[]
    for i in range(n):
        liste_voisins=[]
        for j in range(n):
            if M[i,j]==1:
                liste_voisins.append(j)
        L.append(liste_voisins)
    return L

def ListeToMatrice(L):
    n=len(L)
    M=np.zeros((n,n))
    for i in range(n):
        for j in L[i]: #la lsite des voisins du sommet i
            M[i,j]=1
    return M

##Exercice 3: degré et voisins d'un sommet
def degré1(M,s):
    '''M est la matrice d'ajacence du graphe'''
    n=len(M)
    deg=0
    for j in range(n):
        deg+=M[s,j]
    if M[s,s]==1:#cas d'une boucle sur le sommet s
        deg+=1
    return deg


def degré2(L,s):
    '''L est la liste d'ajacence du graphe'''
    if s in L[s]: #cas d'une boucle sur le sommet s
        return len(L[s])+1
    else:
        return len(L[s])

def voisins1(M,s,t):
    '''M est la matrice d'ajacence du graphe'''
    return M[s,t]==1

def voisins2(L,s,t):
    '''L est la liste d'ajacence du graphe'''
    return t in L[s]
```

```python
##Exercice 4: parcours en profondeur

def parcours_profondeur(L,d):
    '''L est la liste d'adjacence d'un graphe, d est le noeud de départ'''
    visités=[]
    pile=[d]
    while len(pile)!=0:
        s=pile.pop()
        if s not in visités:
            visités.append(s)
            for x in L[s]: #on parcourt les voisins de s
                if x not in visités:
                    pile.append(x)
    return visités

#amélioration avec les marquages des sommets visités
def parcours_profondeurv2(L,d):
    '''L est la liste d'adjacence d'un graphe, d est le noeud de départ'''
    visités=[]
    pile=[d]
    n=len(L) #le nb de sommets
    marquage=[False for i in range(n)]
    while len(pile)!=0:
        s=pile.pop()
        if marquage[s]==False: #si on n'a pas encore visité ce sommet
            visités.append(s)
            marquage[s]=True
            for x in L[s]: #on parcourt les voisins de s
                if marquage[x]==False:
                    pile.append(x)
    return visités

##Exercice 5: Vérification de la connexité d'un graphe
def connexe(L):
    visités=parcours_profondeur(L,0) #peu importe le sommet de départ
    return len(L)==len(visités)

##Exercice 6: Parcours en largeur
def parcours_largeur(L,d):
    '''L est la liste d'adjacence d'un graphe, d est le noeud de départ'''
    n=len(L)
    visités=[d]
    file=deque([d])
    marquage=[False for i in range(n)]
    marquage[d]=True
    while len(file)!=0:
        s=file.popleft()
        for x in L[s]: #on parcourt les voisins de s
            if marquage[x]==False: #on s'occupe seulement des non visités
                    file.append(x)
                    visités.append(x)
                    marquage[x]=True

    return visités

##Exercice 7: Distance à partir d'un sommet sur un graphe non pondéré
def distance(L,d):
    '''L est la liste d'adjacence d'un graphe, d est le noeud de départ
    On calcule la distance entre d et tous les autres'''
    n=len(L)
```

```
    dist=[-1 for i in range(n)] #la distance de d à tous les autres sommets
    #visités=[d]
    dist[d]=0
    file=deque([d])
    marquage=[False for i in range(n)]
    marquage[d]=True
    while len(file)!=0:
        s=file.popleft()
        for x in L[s]: #on parcourt les voisins de s
            if marquage[x]==False: #on s'occupe seulement des non visités
                    dist[x]=dist[s]+1
                    file.append(x)
                    #visités.append(x)
                    marquage[x]=True

    return dist

##Exercice 9: Déclaration d'un graphe pondéré
L3=[[(1,1),(2,3)],[(0,1),(2,1),(3,2)],[(0,3),(1,1),(4,4)],[(1,2),(4,2),(5,6)],[(2,4),(

##Exercice 10: Algorihtme de Dijkstra pour un graphe pondéré
def dijkstra(L,s):
    n=len(L)
    S=list(range(n)) #la liste de tous les sommets
    dist=[np.inf for i in range(n)]
    dist[s]=0
    while len(S)!=0:

        #recherche de la distance minimum parmi les sommets de S
        v_min=S[0]
        d_min=dist[v_min]
        for u in S:
            if dist[u]<d_min:
                d_min=dist[u]
                v_min=u
        v=v_min

        if dist[v]==np.inf:
            return dist


        for couple in L[v]: #les voisins de v avec leur poids
            if couple[0] in S:
                dist[couple[0]]=min(dist[couple[0]],dist[v]+couple[1])
        S.remove(v)

    return(dist)
```