```python
#Correction_info9

#EX1

from numpy import *
G=array([[0,1,1,0,0,0],[1,0,0,1,0,0],[1,0,0,1,0,0],[0,1,1,0,1,1],[0,0,0,1,0,0],
[0,0,0,1,0,0]])

#On va parcourir le tableau er regarder la ième ligne.
def voisins(M,i):
    """voisins(M:array,i:int)->list"""
    p,q=shape(M)
    assert 0<=i<=p
    L=[]
    for j in range(0,q):
        if M[i,j]!=0: #si le poids est non nul, j est voisin de i
            L.append(j) #on stocke alors j
        else:
            pass
    return L

#On peut utiliser le programme précédent et additionner le nombre de voisins de
chaque sommet... le nombre d'arêtes sera alors immédiat.
def degretotal(M):
    """degretotal(M:int)->int"""
    p,q=shape(M)
    S=0
    for i in range(0,p):
        S=S+len(voisins(M,i))
    return S

def aretes(M):
    """aretes(M:int)->int"""
    deg=degretotal(M)
    return int(deg/2)

#EX2

#On va parcourir le tableau avec une double boucle pour repérer les relations entre
deux sommets.
def matricetoliste(M):
    """matricetoliste(M:array)->list"""
    p,q=shape(M)
    L=[[] for k in range(0,p)]
    for i in range(0,p):
        for j in range(0,q):
            if M[i,j]!=0: #si le poids est non nul, c'est un voisin
                L[i].append(j) #on le stocke alors dans la liste des voisins du
sommet i
            else:
                pass
    return L

#On va parcourir la liste et pour chaque sous-liste, on placera des poids égaux à 1
ou 0 en fonction de la présence du sommet.
def listetomatrice(L):
    """listetomatrice(L:list)->array"""
    p=len(L)
    M=zeros((p,p),dtype=int) #j'ai ajouté une option à la demande Charly ;) pour
garder des entiers dans le tableau
    for i in range(0,p):
        for j in range(0,p):
            if j in L[i]:
                M[i,j]=1 #on repère les sommets présents et seulement eux
            else:
                pass
    return M
```

```python
#EX3
def degremax(D:dict)->int:
    """renvoie le degré sortant maximal"""
    assert len(D)!=0
    M=len(D[0])
    for x in D:
        if len(D[x])>M:
            M=len(D[x])
        else:
            pass
    return M

def grapheinv(D:dict):
    assert len(D)!=0
    n=len(D)
    #on construit un dictionnaire avec les bons sommets prêt à recevoir les chemins
inverses
    d={}
    for k in range(0,n):
        d[k]=[]
    #on va alors ajouter les arêtes inverses à chacun des sommets
    for k in range(0,n):
        for x in D:
            if k in D[x]:
                d[k].append(x)
            else:
                pass
    return d

def colorationvalide(D:dict,L:list)->bool:
    #on va simplement tester si deux sommets voisins n'ont pas la même couleur...
    assert len(D)!=0
    for x in D:
        for k in D[x]:
            if L[x]==L[k]:
                return False
    return True


#Le pire des cas se produit par exemple lorsque le graphe est bien colori é et qu’il
faut donc tout tester. On passe alors N fois dans la premi`ere boucle et pour chaque
passage dans cette boucle, on passe au plus M fois dans la seconde boucle. Au total,
on a donc au maximum NM tests d’ égalite.
```