Mathématiques

# Informatique Semestre 4

### Première partie

# Travaux dirigés machine

```
import random
import time
#Recherche dichotomique :
def alÃ©atoire():
    x = random.randint(0,1000)
    return x

def EstPlusGrandQue(n):
    c+=1
    if x>n:
        return True
    return False

def RechercheDichotomique1():
    x=alÃ©atoire()
    c=0
    a=0
    b=1000
    while (b-a) > 1:
        c+=1
        if x > int(a+(b-a)//2):
            a=int(a+(b-a)//2)
        else:
            b=int(a+(b-a)//2)
    if b==x:
        print(c,b)
        return True
    return False

def RechercheDichotomique2(L,i,j,X):
    if X>L[j]:
        return j+1
    while i!=j :
        k =(i+j)//2
        if X <=L[k]:
            j=k
```

```
        else:
            i=k+1
    return i

#Fonction de tri

def Insertion(e,L_triÃ©e):
    pos=0
    e=len(L_triÃ©e)
    while L_triÃ©e[pos]<e and pos < e:
        pos+=1
    L_triÃ©e.insert(pos.e)
    return

def Tri():
    L = [random.random() for i in range(100)]
    L_triÃ©e=[]
    for e in L :
        Insertion(e,L_triÃ©e)
    return L_triÃ©e

def TriInsertion(a): #complexitÃ© n^2
    L= [random.random() for i in range(a)]
    for i in range(1,len(L)):
        if L[i] < L[i-1]:
            for k in range(0,i):
                if L[i]<L[k]:
                    X=L.pop(i)
                    L.insert(k,X)
    return L

def TriDichotomie(L):
    for i in range(1,len(L)):
        if L[i]<L[i-1]:
            k=RechercheDichotomique2(L,0,i-1,L[i])
            X=L.pop(i)
            L.insert(k,X)
    return[L]

#QuickSort
```

```
from random import randint
B=[8,1,0,6,2,4,9,7,5,3]
C=[[1],[2],[5]]
D=[randint(0,1000) for i in range (10000)]

def quickSort(L, dÃ©but=0, fin=None):
    if fin==None:
        fin=len(L)-1
    if dÃ©but>=fin:
        return
    pivot=L[randint(dÃ©but,fin)]
    seuil=partitionner(L,dÃ©but,fin ,pivot)
    quickSort(L,dÃ©but,seuil-1)
    quickSort(L,seuil,fin)

def partitionner(L,dÃ©but,fin,pivot):
    i=dÃ©but
    j=fin
    while i<j:
        while L[i]<pivot:
            i=i+1
        while L[j]>pivot:
            j=j-1
        if i<j:
            a=L[i]
            L[i]=L[j]
            L[j]=a
    return i


#Merge Sort
def fusion(L1,L2):
    i=0
    j=0
    L=[]
    while i<len(L1) and j<len(L2):
        if L1[i]<L2[j]:
            L.append(L1[i])
            i=i+1
        else:
```

```
            L.append(L2[j])
            j=j+1
    while len(L1)>i:
        L.append(L1[i])
        i=i+1
    while len(L2)>j:
        L.append(L2[j])
        j=j+1
    return L

def casser(L):
    L_c=[]
    i=0
    while len(L)>0:
        x=L.pop(i)
        L_c=L_c+[[x]]
    return L_c

def mergeSort(L):
    cpt=0
    L=casser(L)
    while len(L)>2:
        x=L.pop(0)
        y=L.pop(1)
        L.append(fusion(x,y))
    if len(L)==2:
        x=L.pop(0)
        y=L.pop(0)
        return fusion(x,y)


#Comparaison
def comparaison(L):
    L_x=L
    tps1=time.clock()
    x=TriDichotomie(L_x)
    tps2=time.clock()
    print(TriÃ©e(L_x),tps2-tps1  ,"Tri dichotomique terminÃ©")
    L_y = L
    tps3=time.clock()
```

```
    y=mergeSort(L_y)
    tps4=time.clock()
    print(TriÃ©e(L_y),tps4-tps3  ,"MergeSort_terminÃ©")
    L_z=L
    tps5=time.clock()
    z=quickSort(L_z)
    tps6=time.clock()
    print(TriÃ©e(L_z),tps6-tps5  , "QuickSort_terminÃ©")



def TriÃ©e(L):
    for i in range(1,len(L)):
        if L[i]<L[i-1]:
            return False
    return True

from bibgraphes import *
from palette import*

europe=ouvrirGraphe("europe.dot")
petersen=ouvrirGraphe("petersen.dot")
fr=sommetNom(europe,'France')
pr=sommetNom(europe,'Portugal')
esp=sommetNom(europe,'Espagne')
sue=sommetNom(europe,'Suede')
nor=sommetNom(europe,'Norvege')
ger=sommetNom(europe,'Allemagne')
slo=sommetNom(europe,'Slovaquie')
ir=sommetNom(europe,'Irlande')

#Question 2.1
def toutColorier(G,c):
    L=listeSommets(G)
    for i in range(nbSommets(G)):
        s=L[i]
        colorierSommet(s,c)
    return G

#Question 2.2
def existeCouleur(G,c):
    L=listeSommets(G)
```

```
    for i in range(nbSommets(G)):
        s=L[i]
        if couleurSommet(s)==c:
            return True
    return False

#Question 2.3
def toutCouleur(G,c):
    L=listeSommets(G)
    for i in range(nbSommets(G)):
        s=L[i]
        if couleurSommet(s)!=c:
            return False
    return True

#Question 2.4
def toutLaMemeCouleur(G):
    L=listeSommets(G)
    c=couleurSommet(L[0])
    for i in range(nbSommets(G)):
        s=L[i]
        if couleurSommet(s)!=c:
            return False
    return True

#Question 2.5
def nbSommetsCouleur(G,c):
    n=0
    L=listeSommets(G)
    for i in range(nbSommets(G)):
        if couleurSommet(i)==c:
            n=n+1
    return n

#Question 2.6
def nbSommetsColores(G):
    n=0
    for i in listeSommets(G):
        if couleurSommet(i)!=('white'):
            n=n+1
```

```
    return n

#Question 3.1
def sontVoisins(s1,s2):
    for i in listeVoisins(s1):
        if s1==s2:
            return True
    return False

#Question 3.2
def listeVoisinsCommuns(s1,s2):
    L=[]
    for i in listeVoisins(s1):
        for j in listeVoisins(s2):
            if i==j:
                L.append(i)
    return(L)

#Question 3.3
def degreMax(G):
    dg=0
    for i in listeSommets(G):
        if dg<degre(i):
            dg=degre(i)
    return dg

def degreMin(G):
    dg=nbSommets(G)
    for i in listeSommets(G):
        if dg>degre(i):
            dg=degre(i)
    return dg

#Question 3.4
def nbSommetsDegre(G,d):
    n=0
    for i in listeSommets(G):
        if degre(i)==d:
            n=n+1
    return n
```

```
#Question 3.5 somme des degres = 2 fois le nombre d'arrÃ¨tes
def nbAretes(G):
    d=0
    for i in listeSommets(G):
        d=d+degre(s)
    return d//2

#Question 3.6
def existeIsole(G):
    return degreMin(G)==0

#TD 6 - Encore des graphes
#ConnexitÃ©
#Question 1.1
def toutDemarquer(G):
    for i in listeSommets(G):
        demarquerSommet(i)

def sommetAccessible(G):
    for i in listeSommets(G):
        if estMarqueSommet(i)==False:
            for j in listeVoisins(i):
                if estMarqueSommet(j):
                    return i
                else:
                    return None

def marquerAccessibles(G,s):
    marquerSommet(s)
    while sommetAccessible2(G)!=None:
        marquerSommet(sommetAccessible2(G))

#Question 1.2
def sommetsTousMarques(G):
    for i in listeSommets(G):
        if estMarqueSommet(i)==False:
            return False
    return True
```

```
def estConnexe(G):
    toutDemarquer(G)
    L=listeSommets(G)
    s=L[0]
    marquerAccessibles(G,s)
    return sommetsTousMarques(G)

#Question 1.3
def nbComposantesConnexes(G):
    toutDemarquer(G)
    n=0
    for s in listeSommets(G):
        if estMarqueSommet(s)==False:
            marquerSommet(s)
            marquerAccessibles(G,s)
            n=n+1
    return n

#Question 1.4
def estAccessibleDepuis(G,s,t):
    marquerAccessibles(G,s)
    return estMarqueSommet(t)

#Question 1.5
def toutDemarquer2(G):
    for s in listeSommets(G):
        demarquerSommet(s)
        for a in listeAretesIncidentes(s):
            demarquerArete(a)
def sommetAccessible2(G):
    for s in listeSommets(G):
        if estMarqueSommet(s):
            for a in listeAretesIncidentes(s):
                t=sommetVoisin(s,a)
                if estMarqueSommet(t)==False:
                    marquerArete(a)
                    return t
#Trace plein de chemin au pif

#Question 1.6
```

```
def chemin(G,s,t):
    toutDemarquer2(G)
    return dessinerChemin(G,s,t)

def dessinerChemin(G,s,t):
    L=[s]
    marquerSommet(s)
    if s==t:
        return [s]
    for v in listeVoisins(s):
        if estMarqueSommet(v)==False:
            x=  dessinerChemin(G,v,t)
            if x!=None:
                return L+x
    return None

#Question 2.1
def bienColorie(G):
    for s in ListeSommets(G):
        for a in listeAretesIncidentes(s):
            v=sommetVoisin(s,a)
            if couleurSommet(v)==couleurSommet(s):
                return False
    return True

#Question 2.2
#1
def effacerCouleurs(G):
   for s in listeSommets(G):
       colorierSommet(s,'white')

#2
def sommetColoriable(G):
    for s in ListeSommets(G):
        for a in listeAretesIncidentes(s):
            v=sommetVoisins(s,a)
            if couleurSommet(s)=='white':
                if couleurSommet(v)!='white':
                    return s
    return None
```

```
#3
def monoCouleurVoisins(s):
    for t in listeVoisins(s):
        if couleurSommet(t)!='white':
            c=couleurSommet(t)
    for t in listeVoisins(s):
        if couleurSommet(t)!='white':
            c2=couleurSommet(t)
            if c2!=c:
                return None
    return c

#4
def deuxColoration(G,c1,c2):
    effacerCouleurs(G)
    L=ListeSommets(G)
    colorierSommet(L[0],c1)
    for t in listeSommets(G):
        while sommetColoriable(G)!=None:
            if monoCouleurVoisins(s)!=None:
                colorierSommet(t,c2)
            else:
                return G

#Algorithmes d'exploration de graphes
#Question 1.1
def parcoursEnLargeur(G,s):
    effacerCouleurs(G)
    distance={}
    pÃ¨re={}
    F=[s]
    colorierSommet(s,'grey')
    distance[s]=0
    pÃ¨re[s]="NIL"
    #ligne 5 Ã  9 inutiles
    while F!=[]:
        v=F[0]
        for w in listeVoisins(v):
            if couleurSommet(w)=='white':
```

```
                colorierSommet(w,'grey')
                distance[w]=distance[v]+1
                pÃ¨re[w]=v
                F=F+[w]
        del F[0]
        colorierSommet(v,'black')

def parcoursEnLargeur2(G,s):
    toutDemarquer(G)
    distance={}
    pere={}
    F=[s]
    marquerSommet(s)
    distance[s]=0
    pere[s]="NIL"
    while F!=[]:
        v=F[0]
        for w in listeVoisins(v):
            if not estMarqueSommet(w):
                marquerSommet(w)
                distance[w]=distance[v]+1
                pere[w]=v
                F=F+[w]
        del F[0]
        colorierSommet(v,'black')

def colorierParPalette(G,s,p):
    toutDemarquer(G)
    distance={}
    F=[s]
    marquerSommet(s)
    colorierSommet(s,p[0])
    distance[s]=0
    while F!=[]:
        v=F[0]
        for w in listeVoisins(v):
            if not estMarqueSommet(w):
                marquerSommet(w)
                distance[w]=distance[v]+1
                colorierSommet(w,p[distance[w]])
```

```
                F=F+[w]
        del F[0]

def ParcoursEnLargeur3(G,s):
    toutDemarquer(G)
    distance={}
    pere={}
    F=[s]
    marquerSommet(s)
    distance[s]=0
    pere[s]="NIL"
    while F!=[]:
        v=F[0]
        for i in listeAretesIncidentes(v):
            w=sommetVoisin(v,i)
            if not estMarqueSommet(w):
                marquerSommet(w)
                marquerArete(i)
                distance[w]=distance[v]+1
                pere[w]=v
                F=F+[w]
        del F[0]

#Test + TP
#/usr/bin/python3
# -*-coding:utf-8 -*

#Question 1
def estRegulier(G):
    """Retourne True si le graphe G est rÃ©gulier (tous les sommets sont d'un mÃªme degrÃ©
    L=listeSommets(G)
    d=degre(L[0])
    for s in L:
        if degre(s)!=d:
            return False
    return True

#Question 2
def nbAretesIncidentesMarquees(s):
    """Retourne le nombre d'arÃªtes marquÃ©es incidentes d'un sommet s."""
```

```
    n=0
    for a in listeAretesIncidentes(s):
        if estMarqueeArete(a):
            n=n+1
    return n

#Question 3
def estLibre(s):
    """Retourne True si le sommet s est libre, c-Ã -d n'a aucune arÃªte incidente marquÃ©e
    if nbAretesIncidentesMarquees(s)==0:
        return True
    return False

#Question 4
def estSature(s):
    """Retourne True si le sommet s est saturÃ©, c-Ã -d il a une et une seule arÃªte incid
    if nbAretesIncidentesMarquees(s)==1:
        return True
    return False

#Question 5
def estCouplage(G):
    """Retourne True si les arÃªtes marquÃ©es forment un couplage, c-Ã -d chaque sommet de
    (le nombre d'arÃªtes incidentes marquÃ©es est au plus 1 pour tout sommet), False sinon
    L=listeSommets(G)
    for s in L:
        if estSature(s)==False and estLibre(s)==False:
            return False
    return True

#Question 6
def areteIncidenteDisponible(s):
    if estLibre(s)==False:
        return None
    for a in listeAretesIncidentes(s):
        if estLibre(sommetVoisin(s,a))==True:
            return a
    return None

#Question 7
```

```
def areteDisponible(G):
    for a in listeSommets(G):
        if areteIncidenteDisponible(a)!=None:
            return areteIncidenteDisponible(s)
    return None

#Question 8
def construireCouplage(G):
    toutDemarquer2(G)
    for s in listeSommets(G):
        if estLibre(s)==True:
            while areteIncidenteDisponible(s)!=None:
                marquerArete(areteIncidenteDisponible(s))
```

FIN.