Université de Carthage Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul Département Mathématique	IPEIN المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2023/2024 Filière : Informatique Niveau d'étude : 2 année Semestre : 1 Nombre de pages : 4 Date : 16/12/2023 Durée : 2h

CORRIGE D EXAMEN

Problème POO

Partie 1 **(5 pts= 1pt pour chaque question)**

1) **(1 pt)**

```
Class Matrice :
          def __init__(self ,n,m) :
                    self.n=n
                    self.m=m
                    self.A=[]
```

2) **(1 pt)**

```
          def saisie_mat(self ,deb,fin) :
                    for i in range(self.n) :
                              l=[]
                              for j in range(self.m) :
                                        while 1:
                                                  try:
                                                            a=int (input('donner a_'+str(i)+"_"+str(j))
                                                            if a in range(deb,fin+1) :
                                                                      break
                                                  except:
                                                            continue
                                        l.append(a)
                              self.A.append(l)
```

3) **(1pt)**

```
          def __str__(self ) :
                    ch=''
                    for i in range(self.n) :
                              for j in range(self.m):
                                        ch += str(self.A[i][j]) + "  "
                              ch+='\n'
                    return ch
```

4) **(1 pt)**

```
          def __add__(self ,M) :
                    SM=Matrice(self.n,self.m)
                    for i in range(self.n) :
                              l=[]
                              for j in range(self.m) :
```

```
                        l.append(self.A[i][j]+M.A[i][j])
                SM.A.append(l)
        return SM
```

5) (1 pt)

```
def __mul__(self ,M) :
        from copy import copy
        MM=Matrice(self.n,M.m)
        Ligne=[0]*MM.m
        M0 = [ligne] * MM.n
        MM.A=copy(M0)
        for i in range(MM.n) :
                for k in range(MM.m) :
                        c=0
                        for j in range(self.n):
                                c+=self.A[i][j]+M.A[j][k]
                MM.A[i][k]=c
        return MM
```

Partie 2 **(6 pts=1 pour chaque question)**

1) (1 pt)

```
Class Graphe :
        def __init__(self ,n) :
                self.n=n
                self.M=Matrice(self.n,self.n)
```

2) (1 pt)

```
def saisie_Gr(self ) :
        self.M.saisie_mat(0,1)
```

3) (1pt)

```
def __getitem__(self ,num):
        L=[]
        for i in range(self.n) :
                if self.M.A[num][i]==1:
                        L.append(i)
        return L
```

4) (1 pt)

```
def Degre(self ,num) :
                return len( self[num] )  #ou len( self.__getitem__(num) )
```

5) (1 pt)

```
def Nbr_arete(self ) :
        Ln=[self.Degre(num) for num in range(self.n)]
                return sum(Ln)/2
```

6) (1 pt)

```
def __setitem__(self, s, t):
        self.M.A[s][t]=1
        self.M.A[t][s]=1
```

Partie 3 (9 pts=2.5+2.5+2+2)

1) **(2.5 pts=0.25 +0.25+0.5 +0.5+0.5+0.5)**

```
Class Pile :
        def __init__(self ) :
                self.L=[]
        def vide_P(self ) :
                return self.L==[]
        def sommet(self ) :
                if not self.vide_P():
                     return self.L[-1]
        def empiler(self,x ) :
                self.L.append(x)
        def depiler(self ) :
                if not self.vide_P():
                     return self.L.pop()
        def __contains__(self,x ) :
        # version 1 (courte) :
                return  (x  in self.L)
        # ou bien version 2 (plus longue):
                P1=Pile()
                While x!=self.sommet() and not self.vide_P():
                      P1.empiler(self.depiler())
               Test= not self.vide_p()
               While not P.vide_P():
                      self.empiler(P.depiler())
                return Test
```

2) **(2.5 pts= 0.25 +0.25+0.5 +0.5+0.5+0.5)**

```
Class File :
        def __init__(self ) :
                self.L=[]
        def vide_F(self ) :
                return self.L==[]
        def sommet(self ) :  # tête
                if not self.vide_F():
                      return self.L[-1]
        def enfiler(self,x ) :
                self.L.insert(0,x)
        def defiler(self ) :
                if not self.vide_F():
                     return self.L.pop()
        def __contains__(self,x ) :
                return  (x  in self.L)
```

3) **(2pt)**

```
def Parcours_larg (G,numS ) :
    if numS not in range(G.n)
        return None
    else:
        F=File()
        F.enfiler(numS)
        Liste_sommets=[]
        While not F.vide_F():
            S=F.sommet()
            for v in G[S]:      #ou for v in ( G.__getitem__(S) )
                if v not in Liste_sommet and not(v in F):
                    F.enfiler(v)
            Liste_sommets.append(F.defiler())
        return Liste_sommets
```

4) **(2 pt)**

```
def Parcours_prof (G,numS ) :
    if numS not in range(G.n)
        return None
    else:
        P=Pile()
        P.empiler(numS)
        Liste_sommets=[]
        While not P.vide_P():
            S=P.depiler()
            Liste_sommets.append(S)
            For v in G[S]:      #ou for v in ( G.__getitem__(S) )
                if v not in Liste_sommet and not(v in P):
                    P.empiler(v)
        return Liste_sommets
```