In [20]:

```
R = [
    [0,1,1,1,0,0,0,0],
    [1,0,1,0,1,0,0,0],
    [1,1,0,1,1,0,0,0],
    [1,0,1,0,1,0,1,1],
    [0,1,1,1,0,1,1,0],
    [0,0,0,0,1,0,1,1],
    [0,0,0,1,1,1,0,1],
    [0,0,0,1,0,1,1,0],
]
```

Q1

In [4]:

```
print(R[4][2], R[1], len(R[2]), len(R))
```

```
1 [1, 0, 1, 0, 1, 0, 0, 0] 8 8
```

Q2.a

In [21]:

```
def voisines(i, j, R):
    return R[i][j]==1
```

Q2.b

In [26]:

```
def list_voisines(i, R):
    L = []
    for j in range(len(R)):
        if R[i][j] == 1 :
            L.append(j)
    return L
```

Q3

In [27]:

```
def degre(i, R):
    return len(list_voisines(i, R))
print(degre(3,R),degre(0,R),degre(2,R))
```

```
5 3 4
```

Q4

In [34]:

```
def list_degres(R):
    D = []
    for i in range(len(R)):
        D.append((i,degre(i,R)))
    return D
```

Q5.a

In [36]:

```
def tri_degres(D):
    for i in range(len(D)):
        for j in range(i+1,len(D)):
            if D[i][1] < D[j][1]:
                D[i] , D[j] = D[j] , D[i]
```

Q5.b

In [38]:

```
def tri_villes(R):
    D = list_degres(R)
    tri_degres(D)
    L = []
    for a in D :
        L.append(a[0])
    return L
```

Q6

In [46]:

```
def premier_entier(L):
    mx = len(L)+1
    for i in range(1, mx):
        if i not in L:
            return i
    return mx
```

Q7

In [59]:

```
def couleurs_voisines(k, C, R):
    V = list_voisines(k, R)
    L = []
    for i in range(len(V)) :
        L.append(C[V[i]])
    return L
```

Q8

In [64]:

```
def couleurs_villes(R):
    V = tri_villes(R)
    C = [0]*len(V)
    for k in V :
        # obtenir la couleur des voisins de la ville i
        v = couleurs_voisines(k, C, R)
        # obtenir la première couleur possible
        col = premier_entier(v)
        # Affecter la couleur
        C[k] = col
    return C
```

Q9

In [68]:

```
def chemin_valide(T, R):
    # condition c1
    if(len(T)<2) : return False
    # condition c2
    for i in range(len(T)-1):
        if not voisines(T[i], T[i+1], R):
            return False
    return True
```

Q10

In [71]:

```
def chemin_simple(T, R):
    if not chemin_valide(T, R) : return False
    for i in T :
        if T.count(i)>1 : return False
    return True
```

Q11

In [87]:

```
def pc_chemins(i,j, R):
    chemins = liste_chemins(i,j,R)
    L = []
    for ch in chemins :
        if chemin_simple(ch, R) :
            if len(L) > 0 :
                n = len(L[-1])
                if len(ch)==n :
                    L.append(ch)
                elif len(ch)<n :
                    L = [ch]
            else :
                L.append(ch)
    return L
```

Q12

In [83]:

```
def produit_matriciel(A, B):
    n = len(A)
    C = zeros((n, n))
    for i in range(n):
        for j in range(i, n):
            s = 0
            for k in range(n):
                s += A[i][k]*B[k][j]
            C[i][j] = C[j][i] = s
    return C
```

Q13

In [84]:

```
def puissance(R, n):
    if n == 1:
        return R
    elif n % 2 == 0:
        return puissance(produit_matriciel(R, R), n//2)
    else:
        return produit_matriciel(R, puissance(produit_matriciel(R, R), (n-1)//2))
```

Q14

Table villes : Clé primaire : code Table chemins: Clé primaire : chemin Clés étrangères : ville_depart, ville_arrivee

Q15

R1 = Selection(villes, couleur= 1 or couleur = 2) R2 = Projection(R1, code, nom)

Q16

select sum(nb_routes)/2 from chemins where nb_routes=1

Q17

select max(couleur) from villes

Q18

selec nom, count(*) as degre from villes, chemins where villes.code = chemins.ville_depart and nb_routes=1 groupe by chemins.ville_depart having degre between 4 and 9 order by degre DESC

Q19

update chemins set cout = cout + 50 where chemin like "%0-1%"

Q20

select min(cout) from chemins where nb_routes = (select count(*) from villes) - 1