corrigé

Révision sur les graphes

Question 1.

```
u = [42]
for _ in range(100000):
    u.append((u[-1] * 1022) % 65533)
```

Rappelons que `u[-1]` désigne la valeur de la dernière case d'une liste `u` non vide.

Question 2.

```
def graphe_M(n, k, p):
    M = [[0 for j in range(n)] for i in range(n)]
    for i in range(n):
        for j in range(n):
            if u[i + u[k + j]] % p == 0:
                M[i][j] = 1
    return M
```

Question 3. Dans une représentation par matrice d'adjacence, le degré entrant du sommet j est égal au nombre de 1 dans la colonne indexée par j. D'où la fonction :

```
def deg_entrant(M, j):
    n = len(M)
    d = 0
    for i in range(n):
        d += M[i][j]
    return d
```

```
def deg_max(M):
    demax = 0
    for k in range(n):
        de = deg_entrant(M, k)
        if de > demax:
            demax = de
    return demax
```

Question 4.

```
def graphe_L(n, k, p):
    L = [[] for i in range(n)]
    for i in range(n):
        for j in range(n):
            if u[i + u[k + j]] % p == 0:
                L[i].append(j)
    return L
```

Question 5. Un puit est un sommet dont la liste des voisins est vide. D'où la fonction :

```
def puit(L):
    n = len(L)
    for i in range(n):
        if L[i] == []:
            return True
    return False
```

```
def nb_puits(n, p):
    s = 0
    for k in range(1000):
        if puit(graphe_L(n, k, p)):
            s += 1
    return s
```

Question 6. Un graphe est non orienté lorsque pour tout $i \in [\![0, n-1]\!]$ et tout $j \in \mathrm{L}[i]$, $i \in \mathrm{L}[j]$. D'où la fonction :

```
def non_oriente(L):
    n = len(L)
    for i in range(n):
        for j in L[i]:
            if i not in L[j]:
                return False
    return True
```

```
def premier_non_oriente(n, p):
    k = 0
    while not non_oriente(graphe_L(n, k, p)):
        k += 1
    return k
```

Question 7. On compte le nombre de sommets accessibles au fur et à mesure du parcours en profondeur :

```
def nb_accessibles(L):
    n = len(L)
    dejaVus = [False] * n
    dejaVus[0] = True
    aTraiter = [0]
    nb = 0
    while len(aTraiter) > 0:
        s = aTraiter.pop()
        nb += 1
        for v in L[s]:
            if not dejaVus[v]:
                aTraiter.append(v)
                dejaVus[v] = True
    return nb
```

```
def tous_accessibles(n, p):
    k = 0
    while nb_accessibles(graphe_L(n, k, p)) < n:
        k += 1
    return k
```

Question 8. Un parcours en largeur permet de calculer la distance minimale de la source aux sommets accessibles :

```
def distance(L):
    n = len(L)
    dejaVus = [False] * n
    dejaVus[0] = True
    aTraiter = [0]
    dist = ['inf'] * n
    dist[0] = 0
    while len(aTraiter) > 0:
        s = aTraiter.pop(0)
        for v in L[s]:
            if not dejaVus[v]:
                aTraiter.append(v)
                dejaVus[v] = True
                dist[v] = dist[s] + 1
    return dist[n - 1]
```