corrigé

# Une chasse au trésor

**Question 1.**

```
u = [42]
for _ in range(999999):
    u.append((19999999 * u[-1]) % 19999981)
```

**Question 2.**

```
def v(k, p):
    if u[k] % 10000 < p:
        return 1
    return 0
```

```
def nb_aretes(n, p):
    s = 0
    for k in range(n * (n - 1) // 2):
        s += v(k, p)
    return s
```

**Question 3.**

```
def graphe(n, p):
    k = 0
    a = [[] for _ in range(n)]
    for i in range(n):
        for j in range(i + 1, n):
            if v(k, p) == 1:
                a[i].append(j)
                a[j].append(i)
            k += 1
    return a
```

On réalise un parcours (en profondeur) du graphe à partir du sommet 0 en comptant le nombre de sommets rencontrés :

```
def nb_sommets(n, p):
    a = graphe(n, p)
    s = 1
    dejavu = [False] * n
    dejavu[0] = True
    pile = [0]
    while len(pile) > 0:
        i = pile.pop()
        for j in a[i]:
            if not dejavu[j]:
                s += 1
                pile.append(j)
                dejavu[j] = True
    return s
```

**Question 4.** Pour obtenir le nombre de composantes connexes, on réalise un parcours en profondeur à partir de tous les sommets n'ayant pas encore été rencontrés lors d'un parcours précédent.

```python
def nb_composantes(n, p):
    a = graphe(n, p)
    dejavu = [False] * n
    s = 0
    for k in range(n):
        if dejavu[k]:
            continue
        dejavu[k] = True
        pile = [k]
        s += 1
        while len(pile) > 0:
            i = pile.pop()
            for j in a[i]:
                if not dejavu[j]:
                    pile.append(j)
                    dejavu[j] = True
    return s
```

**Question 5.** On ajoute les arêtes indiquées pour rendre le graphe connexe :

```python
def graphe2(n, p):
    a = graphe(n, p)
    for i in range(n - 1):
        if i + 1 not in a[i]:
            a[i].append(i + 1)
            a[i + 1].append(i)
    return a
```

Un parcours en largeur permet de calculer la distance entre deux sommets :

```python
def dist(a, i, j):
    n = len(a)
    dist = [n + 1] * n
    dist[i] = 0
    dejavu = [False] * n
    dejavu[0] = True
    atraiter = [i]
    while True:
        k = atraiter.pop(0)
        for v in a[k]:
            if not dejavu[v]:
                dist[v] = dist[k] + 1
                if v == j:
                    return dist[v]
                dejavu[v] = True
                atraiter.append(v)
```

Il reste à suivre la stratégie gloutonne :

```python
def glouton(n, p, k):
    a = graphe2(n, p)
    pieces = [False] * (n - k) + [True] * k
    dist_totale = 0
    i = 0
    for _ in range(k):
        dmin = n + 1
        for j in range(n - k, n):
            if pieces[j]:
                d = dist(a, i, j)
```

```
                if d < dmin:
                    dmin = d
                    suivant = j
        dist_totale += dmin
        i = suivant
        pieces[suivant] = False
    return dist_totale
```