corrigé

Percolation d'un tableau

Question 1. Je commence par créer un tableau contenant toutes les valeurs de u_n pour $0 \leqslant n \leqslant 1\,000\,000$:

```
m = 2 ** 31 - 1
u0 = (3 ** 20) % m
u = [u0]
for i in range(1000000):
    u.append((16807 * u[i] + 17) % m)
```

```
for i in (5, 2500, 1000000):
    print("u[{}] mod 10000 = {}".format(i, u[i] % 10000))
```

Question 2.

```
v, w = u[0], u[1]
for n in range(999):
    v, w = w, w - v + n ** 2
print("v[1000] mod 10000 = ", w % 10000)
```

Question 3.

```
def nbindices(a, b):
    s = 0
    mini = None
    for i in range(a, b):
        if 2 * (u[i - 3] + u[i - 2] + u[i - 1]) <= u[i]:
            s += 1
            if mini is None or u[i] < mini:
                mini = u[i]
    return (s, mini % 10000)
```

```
for (a, b) in [(10, 1500), (2000, 1000000)]:
    (x, y) = nbindices(a, b)
    print("nb d'indices : {}, ui minimal : {}".format(x, y))
```

Question 4. Je crée un tableau de booléen qui mémorise les éléments de $[\![0, q-1]\!]$ qui sont rencontrés, tout en comptant ceux qui le sont.

```
def Nq(q):
    vu = [False] * q
    s = 0
    n = -1
    while s < q:
        n += 1
        w = u[n] % q
        if not vu[w]:
            vu[w] = True
            s += 1
    return n
```

```
for q in [5, 100, 1000, 10000]:
    print("q = {}, Nq = {}".format(q, Nq(q)))
```

Question 5. Je commence par définir une fonction qui construit la grille :

```
def grille(n, q):
    t = [[0 for j in range(n)] for i in range(n)]
    for i in range(n):
        for j in range(n):
            if u[i * n + j] % 1000 < q:
                t[i][j] = 1
    return t
```

```
def nombreDe1(t):
    n = len(t)
    s = 0
    for i in range(n):
        for j in range(n):
            s += t[i][j]
    return s
```

```
for (n, q) in [(5, 500), (200, 300)]:
    print("nombre de 1 :", nombreDe1(grille(n, q)))
```

Question 6.

```
def ligneMax(n, q):
    t = grille(n, q)
    s, imin = sum(t[0]), 0
    for i in range(1, n):
        if sum(t[i]) > s:
            s, imin = sum(t[i]), i
    return imin
```

```
for (n, q) in [(5, 500), (200, 300)]:
    print("ligne maximale :", ligneMax(n, q))
```

Question 7. Je commence par rédiger une fonction qui calcule T_1 en fonction de T_0 :

```
def nbVoisins(t, i, j):
    n = len(t)
    return t[(i + 1) % n][j] + t[(i - 1) % n][j] + t[i][(j + 1) % n] + t[i][(j - 1) % n]
```

```
def nouvelleGrille(t0):
    n = len(t0)
    t1 = [[0 for j in range(n)] for i in range(n)]
    for i in range(n):
        for j in range(n):
            if t0[i][j] == 1:
                t1[i][j] = 1
            elif nbVoisins(t0, i, j) > 1:
                t1[i][j] = 1
    return t1
```

```
for (n, q) in [(5, 500), (10, 300), (50, 600), (1000, 100)]:
    print("Nombre de 1 de T1 :", nombreDe1(nouvelleGrille(grille(n, q))))
```

Question 8. Je réécris la fonction précédente en y ajoutant un indicateur booléen qui indique si la grille T_1 est différente de T_0 :

```
def nouvelleGrilleBis(t0):
    n = len(t0)
    t1 = [[0 for j in range(n)] for i in range(n)]
    b = False
    for i in range(n):
        for j in range(n):
            if t0[i][j] == 1:
                t1[i][j] = 1
            elif nbVoisins(t0, i, j) > 1:
                t1[i][j] = 1
                b = True
    return t1, b
```

```
def iteration(n, q):
    t0 = grille(n, q)
    t1, b = nouvelleGrilleBis(t0)
    n = 0
    while b:
        t0 = t1
        t1, b = nouvelleGrilleBis(t1)
        n += 1
    return n
```

```
for (n, q) in [(5, 500), (10, 300), (50, 600), (1000, 100)]:
    print("N minimal :", iteration(n, q))
```

Question 9. Un algorithme naïf convient pour les trois premières valeurs de n. Pour obtenir la réponse en un temps raisonnable pour la dernière de ces quatre valeurs, il faut utiliser deux observations : la première est que si $T_0[i,j] = 1$ pour une certaine valeur de q, on aura toujours $T_0[i,j] = 1$ pour la valeur $q+1$; il n'est donc pas nécessaire de recalculer T_0 lorsqu'on change de valeur de q, il suffit de garder la grille en cours. La seconde observation consiste, lors de la percolation, à ne prendre en compte que les cases qui vont passer de la valeur 0 à la valeur 1. C'est le rôle de la variable `transition` de mon algorithme :
Les lignes 5-11 ont pour objet d'initialiser cette variable pour $q = 2$. La percolation s'exécute entre les lignes 13 et 24 : tant que cette variable n'est pas vide, on fait passer de 0 à 1 chacune de ses cases, en ajoutant à `transition` les voisins qui sont maintenant susceptibles de varier à leur tour.
À la ligne 25, la percolation est terminée. Si la grille contient encore des 0, on augmente la valeur de q, ce qui fait passer à 1 un certain nombre de nouvelles cases (lignes 28-34), les voisines de ces nouvelles cases pouvant éventuellement alimenter l'ensemble `transition` (lignes 25-43) et percoler de nouveau.

```python
def percolation(n):
    q = 2
    t = grille(n, q)
    nbzeros = 0
    transition = set()
    for i in range(n):
        for j in range(n):
            if t[i][j] == 0:
                nbzeros += 1
                if nbVoisins(t, i, j) > 1:
                    transition.add((i, j))
    while True:
        while len(transition) > 0:
            (i, j) = transition.pop()
            t[i][j] = 1
            nbzeros -= 1
            if t[(i + 1) % n][j] == 0 and nbVoisins(t, (i + 1) % n, j) > 1:
                transition.add(((i + 1) % n, j))
            if t[(i - 1) % n][j] == 0 and nbVoisins(t, (i - 1) % n, j) > 1:
                transition.add(((i - 1) % n, j))
            if t[i][(j + 1) % n] == 0 and nbVoisins(t, i, (j + 1) % n) > 1:
                transition.add((i, (j + 1) % n))
            if t[i][(j - 1) % n] == 0 and nbVoisins(t, i, (j - 1) % n) > 1:
                transition.add((i, (j - 1) % n))
        if nbzeros == 0:
            return q
        q += 1
        nouveaux = []
        for i in range(n):
            for j in range(n):
                if t[i][j] == 0 and u[i * n + j] % 1000 < q:
                    nouveaux.append((i, j))
                    t[i][j] = 1
                    nbzeros -= 1
        for (i, j) in nouveaux:
            if t[(i + 1) % n][j] == 0 and nbVoisins(t, (i + 1) % n, j) > 1:
                transition.add(((i + 1) % n, j))
            if t[(i - 1) % n][j] == 0 and nbVoisins(t, (i - 1) % n, j) > 1:
                transition.add(((i - 1) % n, j))
            if t[i][(j + 1) % n] == 0 and nbVoisins(t, i, (j + 1) % n) > 1:
                transition.add((i, (j + 1) % n))
            if t[i][(j - 1) % n] == 0 and nbVoisins(t, i, (j - 1) % n) > 1:
                transition.add((i, (j - 1) % n))
```

```python
for n in [5, 10, 50, 800]:
    print('q0 =', percolation(n))
```