On se donne de plus un élément u_0 de S quelconque. On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par l'application répétée de f, c'est-à-dire $u_{n+1} = f(u_n)$ pour tout $n \geq 0$. Comme l'ensemble S est fini, la suite ne peut prendre une infinité de valeurs distinctes : à partir d'un certain rang, celle-ci est donc périodique. On note dans la suite r le plus petit indice à partir duquel la suite est périodique, et p sa période. Plus précisément, r et p sont définis comme suit :

$$r = \min\{n \geq 0 \mid \exists k > n, u_k = u_n\} \qquad \text{et} \qquad p = \min\{k > 0 \mid u_{r+k} = u_r\}$$

Avec f la fonction de la figure 4, on a par exemple :

— si $u_0 = 7$, les valeurs de la suite sont $7, 4, 4, 4, \ldots$, on a donc $r = p = 1$;

— si $u_0 = 8$, les valeurs de la suite sont $8, 0, 6, 3, 1, 6, 3, 1, 6, \ldots$, on a donc $r = 2$ et $p = 3$;

— si $u_0 = 1$, les valeurs de la suite sont $1, 6, 3, 1, 6, \ldots$, on a donc $r = 0$ et $p = 3$.

Question 7. 1. Écrire une fonction `indice(L,x)` prenant en entrée une liste `L` et un élément `x`, et renvoyant le plus petit indice `i` tel que `L[i]` = `x` s'il existe, -1 sinon.

```
>>> indice([3, 4, 1, 0], 3)
0
>>> indice([3, 4, 1, 0], 2)
-1
```

2. Donner sa complexité en fonction de la taille n de la liste, en supposant que le test d'égalité s'effectue en temps constant.

Question 8. Écrire une fonction `rang_periode(f,u0)` prenant en entrée une fonction $f : S \to S$ où S est un ensemble fini, $u_0 \in S$, et renvoyant une liste contenant les entiers r et p définis défini ci-dessus. *Attention :* l'ensemble S n'est pas précisé : on sait simplement que f est bien une fonction d'un ensemble fini vers lui-même. On pourra stocker des termes de la suite dans une liste, et utiliser la fonction précédente. Avec `f` la fonction de l'exemple, on a :

```
>>> rang_periode(f,7)
[1, 1]
>>> rang_periode(f,8)
[2, 3]
>>> rang_periode(f,1)
[0, 3]
```

Question 9. Estimer la complexité de `rang_periode(f,u0)`, en fonction des entiers r et p calculés. On suppose qu'un appel à la fonction f s'effectue avec une complexité constante, de même que le test d'égalité entre deux éléments de S.

C. Calcul efficace : algorithme de Floyd

Pour le jeu de la vie, l'approche précédente demande de stocker en mémoire beaucoup d'univers, si bien qu'elle est inapplicable en pratique dès que N est grand. De plus, les appels à `evolue` et le test d'égalité sur des univers ont une complexité qui dépend également de l'entier N.

On cherche à diminuer à la fois la complexité de la fonction précédente, ainsi que l'espace mémoire utilisé. On propose donc une autre approche. Voici le squelette d'une fonction calculant les entiers r et p, que l'on va compléter.

```
def rang_periode(f,u0):
    u=u0
    v=u0
    t=0
    """ stocke dans t le plus petit élément non nul de A, dans u la valeur u_t et v la valeur v_t """
    while [... à compléter (question 11) ....]:
        u=f(u)
        v=f(f(v))
        t+=1
    """ Ici, t est le plus petit élément non nul de A, u contient u_t, v contient v_t. On calcule p """
    [... à compléter (question 12) ...]
    """ Calcul de r """
    u=u0
    w=u0
    [... à compléter (question 13.1) ...]
    """ Ici on a w=u_p, u=u_0 """
    [.... à compléter (question 13.2) ...]
    return [r,p]
```

Question 10. Soit $(v_n)_{n\in\mathbb{N}}$ la suite définie par $v_n = u_{2n}$ pour tout $n \in \mathbb{N}$. Montrer que l'ensemble $A = \{n \in \mathbb{N} \mid u_n = v_n\}$ est constitué de l'entier 0 et des multiples de p supérieurs ou égaux à r.

Dans la suite, vous avez **interdiction** d'utiliser des listes : le but est de ne stocker qu'un nombre fini d'éléments de S.

Question 11. On note $t = \min(A\backslash\{0\})$. On remarque que $v_0 = u_0$ et $v_{n+1} = f(f(v_n))$ pour tout $n \geq 0$. Compléter la condition de la boucle `while` du script pour que celle-ci permette de stocker dans la variable `t` la valeur t précédemment définie, ainsi que les valeurs u_t dans `u` et v_t dans `v`).

Question 12. Pour calculer la période p, il suffit d'itérer f sur u_t jusqu'à retomber sur $u_t = u_{t+p}$. Indiquer comment compléter le code permettant de stocker p dans la variable `p`.

Question 13. Considérons la suite $(w_n)_{n\in\mathbb{N}}$ définie par $w_n = u_{n+p}$ pour tout $n \in \mathbb{N}$. On admet que le premier indice k tel que $w_k = u_k$ est précisément r.

1. Compléter le code (question 13.1) pour stocker dans la variable `w` la valeur $w_0 = u_p$.
2. Enfin, compléter le code (question 13.2), pour stocker dans la variable `r` la valeur r définie dans l'énoncé.

Question 14. Que pouvez-vous dire de la complexité de cette deuxième version, en fonction des entiers r et p calculés (on supposera ici qu'un appel à f se fait en complexité constante, de même que le test d'égalité de deux éléments de S) ? Justifier.

D. Les univers ayant les plus longues périodes

À l'aide des fonctions des sections précédentes, on a pu générer des univers aléatoires, et calculer les périodes des suites démarrant par ces univers. Ces paires sont stockés sous la forme de listes à 2 éléments `[U, p]` où `U` est un univers, et `p` est la période de la suite de premier terme `U`. On a construit une liste de telles listes de taille 2. On souhaite la trier par période décroissante. On veut faire usage du tri fusion, on a pour cela écrit la fonction suivante :

```
def tri(L):
    """ Tri d'une liste non vide de listes [U,p] """
    if len(L)==1:
        return [L[0]]
    else:
        L1, L2 = separation(L)
        return fusion(tri(L1), tri(L2))
```

Question 15. Écrire la fonction `separation(L)` prenant en entrée une liste `L` ayant au moins deux éléments, et renvoyant une liste `[L1, L2]` avec `L1` et `L2` de même taille à un élément près, telles que les éléments de `L` soient répartis dans `L1` et `L2`.

Question 16. Écrire la fonction `fusion(L1, L2)`, prenant en entrée deux listes contenant des éléments de la forme `[U, p]`, supposées triées par période décroissante, et renvoyant une liste contenant tous les éléments de `L1` et `L2`, triée par période décroissante. On impose une complexité linéaire en la somme des tailles de `L1` et `L2`.

Question 17. Quelle est alors la complexité de `tri(L)` ?

E. SQL

On a sélectionné certains univers menant à des situations intéressantes, et regroupé ceux-ci dans une base de données à deux tables.

- La table `univers` comporte quatre champs de type entier, et un champ de type chaîne de caractères.
  - `id` : identifiant l'univers, c'est la clé primaire de la table ;
  - `n` : l'univers est de taille $n \times n$;
  - `p` et `r` : période et rang d'attraction de la suite de premier terme l'univers ;
  - `nom` : un petit nom donné à l'univers, comme `'planeur'` ou `'canon'`.
- La table `cellules` comporte également trois champs de type entier :
  - `idu` : un identifiant d'univers, c'est une clé étrangère qui référence le champ `id` de `univers` ;
  - `i` et `j` : servent à référencer les cellules vivantes de l'univers.

Ces tables sont telles qu'un univers d'identifiant `id` possède en (i, j) une cellule vivante si et seulement si `(id, i, j)` est dans la table `cellules`.

Question 18. On demande de répondre à chacune des questions suivantes par une unique requête SQL.

1. Donner le nombre d'entrées dans la table `univers`.
2. Donner la liste des couples (i, j) d'indices de cellules vivantes de l'univers appelé `'canon'`. On suppose qu'un tel univers est présent dans la table `univers` et que ce nom est unique.
3. Donner, pour chaque période présente dans `univers`, le nombre d'univers de la table ayant cette période.
4. Donner la liste des couples d'identifiants d'univers différents de même période et même rang d'attraction.

F. Stockage dans un fichier

On souhaite communiquer nos meilleurs univers à des collègues. Pour cela, il s'agit de les écrire dans des fichiers !

Question 19. Écrire une fonction `imprime(U, nom)`, prenant en entrée un univers `U` est une chaîne de caractères `nom` (par exemple `'planeur.txt'`), et créant un fichier de nom `nom` dans le répertoire courant, et y écrivant l'univers. Chaque ligne de l'univers sera associée à une ligne du fichier, on écrira les caractères `T` et `F` à la place de `True` et `False`, et on séparera les entrées par des points-virgules. Par exemple, si `U` vaut

```
[[True, False, False, False], [False, False, True, True], [False, False, False, False],
[True, False, True, False]]
```

les lignes du fichier seront :

```
T;F;F;F
F;F;T;T
F;F;F;F
T;F;T;F
```