UNIVERSITé DE RENNES I
U.F.R. DE MATHÉMATIQUES

UNIVERSITÉ DE RENNES 1

Licence 2 Mathématiques et Applications **2018-2019**
U.E. Outils Informatiques B

TD 2 - Recherche de points fixes

Dans ce TD on s'intéresse aux points fixes des fonctions $f : E \to E$, où E est un ensemble fini. Le calcul effectif et efficace des points fixes de telles fonctions est un problème récurrent en informatique (transformation d'automates, vérification automatique de programmes, algorithmique des graphes, etc) et admet différentes approches selon la structure de E et les propriétés de f. On suppose par la suite un entier $n > 0$ fixé et on pose $E_n = \{0, 1, ..., n-1\}$. On représente une fonction $f : E_n \to E_n$ par un tableau ou une liste t de taille n, autrement dit $f(x) = t[x]$ pour tout $x = 0, 1, \cdots, n-1$. Ainsi, la fonction f_0 qui à $x \in E_{10}$ associe $2x + 1\text{modulo}10$ est- elle représentée par le tableau :

1	3	5	7	9	1	3	5	7	9
t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]	t[9]

Toutes les fonctions demandées devront être écrites en Python.

Partie 1 - Recherche de point fixe, cas général

1. Écrire une fonction `admet_point_fixe(t)` qui prend en argument un tableau ou une liste `t` de taille `n` et renvoie `True` si la fonction $f : E_n \to E_n$ représentée par `t` admet un point fixe, `False` sinon. Par exemple, `admet_point_fixe` devra renvoyer `True` pour le tableau donné en introduction, puisque 9 est un point fixe de la fonction f_0 qui à x associe $2x + 1\text{modulo}10$.

2. Écrire une fonction `nb_points_fixes(t)` qui prend en argument un tableau `t` de taille `n` et renvoie le nombre de points fixes de la fonction $f : E_n \to E_n$ représentée par `t`. Par exemple, `nb_points_fixes` devra renvoyer 1 pour le tableau donné en introduction, puisque 9 est le seul point fixe de f_0.

3. On note f^k l'itérée $k$ième de f, autrement dit, $f^k = \underbrace{fof \cdots fof}_{k \text{ fois}}$

   Écrire une fonction `itere(t, x, k)` qui prend en premier argument un tableau `t` de taille `n` représentant une fonction $f : E_n \to E_n$, en deuxième et troisième arguments des entiers `x` et `k` de E_n, et renvoie $f^k(x)$.

4. Écrire une fonction `nb_points_fixes_iteres(t, k)` qui prend en premier argument un tableau `t` de taille `n` représentant une fonction $f : E_n \to E_n$, en deuxième argument un entier $k \geq 0$, et renvoie le nombre de points fixes de f^k.

5. Un élément $z \in E_n$ est dit attracteur principal de $f : E_n \to E_n$ si et seulement si z est un point fixe de f , et pour tout $x \in E_n$, il existe un entier $k \geq 0$ tel que $f^k(x) = z$.
   Afin d'illustrer cette notion, on pourra vérifier que la fonction f_1 représentée par le tableau ci-dessous admet 2 comme attracteur principal.

5	5	2	2	0	2	2
t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]

   En revanche, on notera que la fonction f_0 donnée en introduction n'admet pas d'attracteur principal, puisque $f_0^k(0) \neq 9$ quel que soit l'entier $k \geq 0$.
   Écrire une fonction `admet_attracteur_principal(t)` qui prend en argument un tableau `t` de taille `n` et renvoie `True` si et seulement si la fonction $f : E_n \to E_n$ représentée par `t` admet un attracteur principal, verb=False= sinon. On ne requiert pas ici une solution efficace.

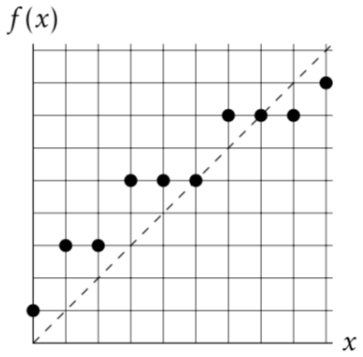
6. On suppose que f admet un attracteur principal. Le temps de convergence de f en $x \in E_n$ est le plus petit entier $k \geq 0$ tel que $f^k(x)$ soit un point fixe de f. Pour la fonction f_1 ci-dessus, le temps de convergence en 4 est égal à 3. En effet,

$$f_1(4) = 0,\ f_1^2(4) = 5, f_1^3(4) = 2$$

   et 2 est un point fixe de f_1. On note $tc(f, x)$ le temps de convergence de f en x.
   Écrire une fonction `temps_de_convergence(t, x)` qui prend en argument un tableau `t` de taille `n` reprsentant une fonction $f : E_n \to E_n$ qui admet un attracteur principal, en deuxième argument un entier `x` de E_n, et renvoie le temps de convergence de f en x.

Partie 2 - Recherche de point fixe, cas général

Toute fonction `point_fixe(t)` retournant un point fixe d'une fonction arbitraire est de complexité au mieux linéaire en n. On s'intéresse maintenant à des améliorations possibles de cette complexité lorsque le fonction considérée est croissante. On rappelle qu'une fonction $f : E_n \to E_n$ est croissante si et seulement si pour tout $(x, y) \in E_n^2, x \geq y \Rightarrow f(x) \geq f(y)$. On admet qu'une fonction croissante de E_n dans E_n admet toujours un point fixe. Á titre d'exemple, la fonction dont le tableau et le graphe sont donnés ci-dessous est croissante. Elle a deux points fixes, à savoir les entiers 5 et 7.



1	3	3	5	5	5	7	7	7	8
t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]	t[9]

1. Écrire une fonction `est_croissante(t)` qui prend en argument un tableau `t` et renvoie `True` si la fonction représentée par `t` est croissante, `False` sinon. On impose un temps de calcul linéaire en la taille n du tableau. On ne demande pas de démonstration du fait que le temps de calcul de la solution proposée est linéaire.
2. Écrire une fonction `point_fixe(t)` qui prend en argument un tableau `t` de taille n représentant une fonction croissante $f : E_n \to E_n$ et retourne un entier $x \in E_n$ tel que $f(x) = x$. On impose un temps de calcul logarithmique [1] en la taille n du tableau. On ne demande pas de démonstration du fait que le temps de calcul de la solution proposée est logarithmique.

1. Voir annexe

Annexe : extrait de wikipedia

Supposons que le problème posé soit de trouver un nom dans un annuaire téléphonique qui consiste en une liste triée alphabétiquement. On peut s'y prendre de plusieurs façons différentes. En voici deux :

- Recherche linéaire : parcourir les pages dans l'ordre (alphabétique) jusqu'à trouver le nom cherché.
- Recherche dichotomique : ouvrir l'annuaire au milieu, si le nom qui s'y trouve est plus loin alphabétiquement que le nom cherché, regarder avant, sinon, regarder après. Refaire l'opération qui consiste à couper les demi-annuaires (puis les quarts d'annuaires, puis les huitièmes d'annuaires, etc.) jusqu'à trouver le nom cherché.

Pour chacune de ces méthodes il existe un pire des cas et un meilleur des cas. Prenons la méthode 1 :

- Le meilleur des cas est celui où le nom est le premier dans l'annuaire, le nom est alors trouvé instantanément.
- Le pire des cas est celui où le nom est le dernier dans l'annuaire, le nom est alors trouvé après avoir parcouru tous les noms.

Si l'annuaire contient 30 000 noms, le pire cas demandera 30 000 étapes. La complexité dans le pire des cas de cette première méthode pour n entrées dans l'annuaire fourni est $O(n)$, ça veut dire que dans le pire des cas, le temps de calcul est de l'ordre de grandeur de n : il faut parcourir tous les n noms une fois.

Le second algorithme demandera dans le pire des cas de séparer en deux l'annuaire, puis de séparer à nouveau cette sous-partie en deux, ainsi de suite jusqu'à n'avoir qu'un seul nom. Le nombre d'étapes nécessaire sera le nombre entier qui est immédiatement plus grand que $\log_2 n$ qui, quand n est 30 000, est 15. La complexité (le nombre d'opérations) de ce second algorithme dans le pire des cas est alors $O(\log_2\ n)$. On peut écrire aussi bien $O(\ln\ n)$ ou $O(\log_2\ n)$, car $\ln\ n$ et $\log_2\ n$ ont le même ordre de grandeur.