Concours Blanc : TP d'Informatique

PTSI A et B Lycée Eiffel

9 juin 2017

Dans tout ce TP, il est **interdit** de recourir aux fonctions max, sum ou autres fonctions rendant manifestement évidents la programmation des fonctions demandées dans l'énoncé.

On s'intéresse dans ce problème à l'étude de points fixes d'applications $f : E_n \to E_n$, où on note, pour tout entier $n \geqslant 1$, $E_n = \{0, 1, \ldots, n-1\}$ (ensemble des n premiers entiers naturels). Une telle application f sera représentée en Python par la liste L des images par f des éléments de E_n. Par exemple l'application f définie sur E_6 par les images suivantes :

x	0	1	2	3	4	5
$f(x)$	3	5	2	1	3	4

sera représentée par la liste $[3, 5, 2, 1, 3, 4]$.

I. Quelques exemples et fonctions élémentaires.

1. Écrire une fonction Python **image(L,k)** qui prend comme argument une liste L représentant une fonction f et un entier k et ressortant la valeur de $f(k)$. Si k n'appartient pas à l'ensemble E_n correspondant à la fonction f, la fonction devra renvoyer un message d'erreur.
   Exemple : image([3,5,2,1,3,4],4) doit renvoyer la valeur 3, mais image([3,5,2,1,3,4],8) doit renvoyer un message d'erreur.
2. Écrire une fonction Python **constante(n,a)** qui renvoit la liste correspondant à l'application constante égale à a définie sur E_n.
3. Écrire une fonction Python **antécédents(L,k)** qui prend comme argument une liste L représentant une fonction f et un entier k et qui renvoit la liste des antécédents de k par f (dans le cas où il n'y a pas d'antécédent, la fonction renvoit simplement une liste vide).
4. Écrire une fonction Python **max(L)** qui renvoit la plus grande valeur prise par la fonction f représentée par L (qui est donc simplement la plus grande valeur de la liste L, cette question est une question de cours).
5. Écrire une fonction Python **bijectif(L)** qui détermine si la fonction f représentée par L est une bijection de l'ensemble E_n (méthode au choix, on ne cherchera pas à faire une fonction efficace). La fonction renverra True ou False.

II. Recherche de point fixe : cas général.

Dans toute la suite de l'énoncé, on identifie la liste L et l'application f pour alléger un peu les énoncés. Un point fixe de l'application f (ou, donc, de la liste L) est simplement un entier k vérifiant $f(k) = k$.

1. Écrire une fonction Python **admetpointfixe(L)** qui détermine si la liste L admet ou non un point fixe (et renvoit True ou False).
   Exemple : admetpointfixe([3,5,2,1,3,4]) doit renvoyer True (2 est un point fixe de f), mais admetpointfixe([3,5,3,1,0,4]) doit renvoyer False.

2. Écrire une fonction Python **nombrepointfixes(L)** qui renvoit le nombre de points fixes de L.
3. Pour un entier $i \geqslant 1$, on note f^i la composée i fois de f par elle-même : $f^2 = f \circ f$, $f^3 = f \circ f \circ f$ etc. Écrire une fonction **iteree(L,i,k)** qui renvoit la valeur de $f^i(k)$.
4. Un élément $p \in E_n$ est dit **attracteur principal** de la fonction f si, pour tout entier $k \in E_n$, il existe un entier i tel que $f^i(k) = p$. On **admet** que, si p est un attracteur principal, on aura $f^n(k) = p$ pour tout entier $k \in E_n$ (autrement dit, la valeur se stabilise en un nombre d'itérations inférieur ou égal au nombre d'éléments de l'ensemble).

   Écrire une fonction **attracteur(L)** qui détermine s'il existe un attracteur principal pour la liste L. On renverra True ou False.
5. On suppose pour les deux dernières questions de cette partie que la liste L admet un attracteur principal p. Pour tout entier $k \in E_n$, on appelle alors **temps de convergence** de k le plus petit entier i pour lequel $f^i(k) = p$. Écrire une fonction Python **tempsconvergence(L,k)** qui calcule le temps de convergence de k pour la liste L.
6. Écrire une fonction Python **tempsconvergencemax(L)** qui calcule le temps de convergence maximal pour la liste L.

III. Recherche rapide de point fixe dans une liste triée.

On va décrire dans cette dernière partie un algorithme de recherche de point fixe plus rapide dans le cas où la liste L correspond à une fonction f croissante (au sens large, si f est strictement croissante, tout le monde est point fixe, c'est beaucoup plus facile).

1. Écrire une fonction Python **estcroissante(L)** qui détermine si une liste L quelconque est effectivement croissante. La fonction doit renvoyer True ou False.
2. On va maintenant chercher un point fixe dans une liste croissante (qui en possède forcément un) en procédant par dichotomie : quel que soit l'entier $k < n$, si $f(k) > k$, alors f admet un point fixe entre $k+1$ et $n-1$, et si $f(k) < k$, alors f admet un point fixe entre 0 et $k-1$ (on ne demande pas de démontrer ces résultats). À l'aide de cette constatation, écrire un alorithme dichotomique **pointfixedicho(L)** qui prend comme argument une liste croissante et calcule un point fixe de la liste.
3. Étudier la rapidité d'exécution de l'algorithme précédent en fonction de la taille n de la liste donnée. On pourra créer une liste constituée d'un grand nombre d'entiers aléatoires à l'aide de la fonction **randint(a,b)** du module **random** (cette fonction renvoit un entier aléatoire compris entre a et b inclus), on aura le droit de trier cette liste avec la méthode **sort** pour la rendre croissante, puis tester dessus l'algorithme. On rappelle que, pour tester le temps d'exécution de l'algorithme (ça devrait aller assez vite, utilisez de très grosses listes ou, mieux, faites tourner 1 000 ou 10 000 fois de suite l'algorithme), on dispose de la fonction **time()** du module **time** qui mesure le temps au moment où elle est executée. On pourra bien sûr commenter les résultats obtenus.