Concours blanc IPT

Mardi 05 mai 2016

Le sujet est composé de 4 pages. **Les calculatrices sont interdites**.*La correction tiendra fondamentalement compte de la qualité de la rédaction et de la présentation.*

Si le candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Implémentation

Dans ce sujet, on adoptera la syntaxe du langage `Python`.

La complexité d'une fonction Python `proc` de paramètres $p_1, \dots, p_k$ est définie comme le nombre maximal d'opérations élémentaires exécutées par `proc` pour ces paramètres.

On rappelle qu'on dispose des opérations suivantes, dont on supposera qu'elles ont toutes une complexité constante (les listes `Python` étant des tableaux redimensionnables) :

- [] crée une liste vide ;
- si L est une liste, $\text{len}(L)$ renvoie la longueur de la liste L ;
- si L est une liste et $0 \leqslant i \leqslant \text{len}(L) - 1$, $L[i]$ renvoie le i-ième élément de L ;
- si L est une liste, e une expression et $0 \leqslant i \leqslant \text{len}(L) - 1$, $L[i] = e$ affecte le i-ième élément de L à e ;
- si L est une liste, $L.pop()$ renvoie la valeur du dernier élément de la liste **et** l'élimine de la liste ;
- si L est une liste et e une expression, $L.append(e)$ ajoute la valeur de e à la fin de la liste L ;
- `True` et `False` sont les deux valeurs booléennes *vrai* et *faux*.

Le candidat reste libre d'utiliser d'autres fonctions, pourvu qu'elles existent et qu'elles soient clairement spécifiées. **Par contre, on ne pourra faire aucune hypothèse de complexité sur ces autres fonctions et on ne pourra donc pas les utiliser dans les questions pour lesquelles on attend une complexité donnée**.

Enfin, le code écrit devra être sûr (pas d'accès invalide à une liste, pas de division par zéro, et le programme termine, notamment) pour toutes valeurs des paramètres vérifiant les conditions données dans l'énoncé. Pour toute autre valeur des paramètres, le comportement du code proposé n'aura aucune importance et il n'en sera pas tenu compte dans la notation.

Points fixes d'applications sur des ensembles finis

Dans tout le problème, on s'intéresse aux points fixes des applications $f : E \to E$, où E est un ensemble fini. Le calcul effectif et efficace des points fixes de telles applications 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 considèrera ici le cas d'un ensemble de la forme $E_n = \{0, 1, \dots, n-1\}$.

On représentera une application $f : E_n \to E_n$ par une liste Python `L` de longueur n, autrement dit $f(x) =$ `L[x]` pour tout x tel que $0 \leqslant x \leqslant n-1$.

Ainsi l'application f_0 qui à $x \in E_{10}$ associe $2x + 1$ modulo 10 est-elle représentée par la liste Python `L` suivante :

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

Pour lever toute ambiguité, cette liste serait créée avec l'instruction `L = [1,3,5,7,9,1,3,5,7,9]`.

Partie I – Échauffement

▶ **Question 1** Quelle application est représentée par la liste $[1,1,1,1]$? *Aucune justification n'est demandée.*

▶ **Question 2** Écrire une fonction Python `constante(a,n)` qui prend en argument un entier n et un entier $a \in E_n$ et renvoie la liste représentant l'application constante $\left\{ \begin{array}{ccc} E_n & \to & E_n \\ x & \mapsto & a. \end{array} \right.$

▶ **Question 3**

a. Écrire une fonction Python `max(L)` qui prend en argument une liste `L` d'entiers, non nécessairement triée (c'est-à-dire que les entiers peuvent figurer dans la liste dans n'importe quel ordre), et renvoie le maximum des éléments de `L`.

b. Écrire une fonction Python `recherche_dichotomique(a,Lt)` qui prend en argument un entier `a` et une liste **triée** d'entiers `Lt` (c'est-à-dire que les entiers sont rangés de manière croissante, comme par exemple `[1,3,4,6,7]`), et renvoie `True` si l'élément `a` est dans la liste `Lt` et `False` sinon, en utilisant l'algorithme de recherche dichotomique. Donner sans démonstration la complexité de cet algorithme.

Ces fonctions existent déjà en Python mais on demande de les reprogrammer.

Partie II – Recherche de point fixe : cas général

On rappelle que x est un point fixe de l'application f si et seulement si $f(x) = x$.

▶ **Question 4** Écrire une fonction Python `admet_point_fixe(L)` qui prend en argument une liste `L` et renvoie `True` si l'application $f : E_n \to E_n$ représentée par `L` admet un point fixe, `False` sinon (n est donc ici la longueur de L). Par exemple, `admet_point_fixe` devra renvoyer `True` pour la liste donnée en introduction, puisque 9 est un point fixe de l'application f_0 qui à x associe $2x+1$ modulo 10.

▶ **Question 5** Écrire une fonction Python `nb_points_fixes(L)` qui prend en argument une liste `L` et renvoie le nombre de points fixes de l'application $f : E_n \to E_n$ représentée par `L` (n est donc ici la longueur de L). Par exemple, `nb_point_fixes` devra renvoyer 1 pour la liste donnée en introduction, puisque 9 est le seul point fixe de f_0.

Dans la suite, on note f^k l'itérée k-ième de f, autrement dit l'application $f^k : \left\{ \begin{array}{ccc} E_n & \to & E_n \\ x & \mapsto & \underbrace{f(f(\dots f}_{k \text{ fois}}(x))\dots) \end{array} \right.$

▶ **Question 6** Écrire une fonction Python `itere(L,x,k)` qui prend en premier argument une liste `L` représentant une fonction $f : E_n \to E_n$ (n est donc ici la longueur de L), en deuxième et troisième arguments des entiers x, k de E_n, et qui renvoie $f^k(x)$.

▶ **Question 7** Écrire une fonction Python `nb_points_fixes_iteres(L,k)` qui prend en premier argument une liste `L` représentant une application $f : E_n \to E_n$ (n est donc ici la longueur de L), en deuxième argument un entier $k \geqslant 0$, et qui renvoie le nombre de points fixes de f^k.

Un élément $z \in E_n$ est dit attracteur principal de $f : E_n \to E_n$ lorsque z est un point fixe de f et que pour tout $x \in E_n$, il existe un entier $k \geqslant 0$ tel que $f^k(x) = z$. Lorsque c'est le cas, on admet que pour tout $x \in E_n$, on a $f^n(x) = z$ (ce qui se montre facilement à l'aide du principe des tiroirs).

Pour illustrer cette notion : la fonction f_1 représentée par la liste ci-dessous admet 2 comme attracteur principal. **Il n'est pas demandé de le montrer.**

i	0	1	2	3	4	5	6
L[i]	5	5	2	2	0	2	2

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 $k \geqslant 0$.

▶ **Question 8** Écrire une fonction Python `admet_attracteur_principal(L)` qui prend en argument une liste `L` et renvoie `True` si et seulement si l'application $f : E_n \to E_n$ représentée par `L` admet un attracteur principal, `False` sinon (n est donc ici la longueur de L). On n'impose ici aucune complexité particulière.

On suppose aux questions 9 à 11 que f admet un attracteur principal. Le **temps de convergence** de f en $x \in E_n$ est le plus petit entier $k \geqslant 0$ tel que $f^k(x)$ soit un point fixe de f. Pour l'application f_1 ci-dessus, le temps de convergence en 4 est 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 $\texttt{tc}(f, x)$ le temps de convergence de f en x.
On notera que, d'après la propriété admise, le temps de convergence de f en x est toujours inférieur à n.

▶ **Question 9** Écrire une fonction Python `temps_de_convergence(L,x)` qui prend en premier argument une liste `L` représentant une application $f : E_n \to E_n$ qui admet un attracteur principal (n est donc ici la longueur de L), en deuxième argument un entier x de E_n, et renvoie le temps de convergence de f en x.

▶ **Question 10** Écrire une fonction Python `temps_de_convergence_max(L)` qui prend en argument une liste `L` représentant une application $f : E_n \to E_n$ qui admet un attracteur principal, et renvoie $\max_{x \in E_n} \texttt{tc}(f, x)$.
On impose dans cette question un temps de calcul au plus quadratique en la longueur n de la liste (*i. e.* en $O(n^2)$). On ne demande pas de démonstration de la complexité de la solution proposée, mais il est impératif d'expliquer clairement le fonctionnement de votre fonction Python.

▶ **Question 11** Reprendre la fonction `temps_de_convergence_max(L)` de sorte que sa complexité soit **linéaire** en la longueur n de la liste (*i. e.* en $O(n)$). On ne demande pas de démonstration de la complexité de la solution proposée, mais il est impératif d'expliquer clairement le fonctionnement de votre fonction Python.
À titre d'indication, on pourra au besoin créer des listes intermédiaires au cours du calcul.

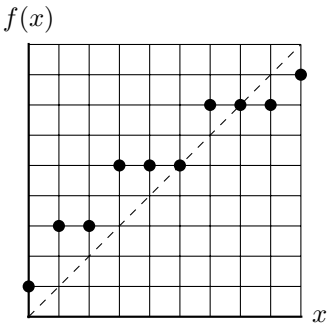
Partie III – Recherche logarithmique dans un cas particulier

Toute fonction Python `point_fixe(L)` retournant un point fixe d'une fonction arbitraire est de complexité au mieux linéaire en la longueur de L. Il existe des améliorations possibles de cette complexité lorsque la fonction considérée possède certaines propriétés spécifiques.

On se limitera ici au cas d'une application croissante de E_n dans E_n. On rappelle qu'une application $f : E_n \to E_n$ est croissante si et seulement si pour tous $x, y \in E_n$ tels que $x \leqslant y, f(x) \leqslant f(y)$.

On admet que pour toute partie $A \subset E_n$, une application croissante de A dans A admet toujours un point fixe.

À titre d'exemple, l'application dont la liste et le graphe sont donnés ci-dessous est croissante. Elle a deux points fixes, à savoir les entiers 5 et 7.



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

▶ **Question 12** Écrire une fonction Python `est_croissante(L)` qui prend en argument une liste `L` et renvoie `True` si l'application représentée par `L` est croissante, et `False` sinon. On impose un temps de calcul **linéaire** en la longueur n de la liste. On ne demande pas de démonstration du fait que le temps de calcul de la solution proposée est linéaire.

▶ **Question 13** Écrire une fonction Python `point_fixe(L)` qui prend en argument une liste `L` représentant une application croissante $f : E_n \to E_n$ (n est donc ici la longueur de L), et retourne un entier $x \in E_n$ tel que $f(x) = x$. On impose un temps de calcul **logarithmique** en la longueur n de la liste : pour ce faire, on pourra s'inspirer de la question 3.b. On ne demande pas ici de démonstration du fait que le temps de calcul de la solution proposée est logarithmique.

▶ **Question 14** Démontrer que la fonction Python de la question 13 termine.

▶ **Question 15** Justifier que le temps de calcul est logarithmique en la longueur n de la liste.

FIN DE L'ÉPREUVE