Devoir d'informatique n° 01 (2 heures)

Ce devoir est constitué de deux petits problèmes. L'ordre des exercices ne correspond à aucun critère de difficulté ou de longueur : vous pouvez les traiter dans l'ordre que vous voulez. Veillez à soigner la copie tant pour l'écriture, la propreté que pour la rédaction, la rigueur et l'argumentation. Vous numéroterez vos copies et ferez apparaître clairement sur la première page le nombre de copies.
La calculatrice n'est pas autorisée.

EXERCICE I : Chiffre de César ou chiffrement par décalage

Soit n un entier tel que $n \leqslant 26$. On souhaite écrire un programme qui code un mot en « *décalant* »chaque lettre de l'alphabet de n lettres. Par exemple, pour $n = 4$, la lettre **a** est « *codée* »par la lettre **e**, la lettre **m** par la lettre **q**, la lettre **w** par la lettre **a**, la lettre **z** par la lettre **d**, le mot **lycee** par le mot **pcgii** et le mot **marceau** par le mot **qevgiey**.

1. Définir une chaine de caractères ***code*** contenant toutes les lettres dans l'ordre alphabétique (caractères en minuscules).
2. Écrire une fonction ***decalage*** d'argument un entier **n** renvoyant une chaine de caractères contenant toutes les lettres dans l'ordre alphalbétique décalées de **n**, comme indiquée ci-dessus.
3. Écrire une fonction ***indices*** comme arguments un caractère **x** et une chaine de caractères **phrase** renvoyant une liste contenant les indices de **x** dans **phrase** si **x** est une lettre de **phrase**, une liste vide sinon. **Attention** : on n'a pas le droit dans cette question d'utiliser la méthode .index sur les listes ou chaines de caractères.
4. Écrire une fonction ***codage*** d'arguments un entier **n** et une chaine de caractères **phrase** renvoyant **phrase** codé avec un décalage de **n** lettres.
5. Comment peut-on décoder un mot codé ? Que pensez-vous du cas particulier $n = 13$, qui était utilisé par les romains ?

EXERCICE II : Nombres

1. On considère le code Python de la fonction ***dd*** suivante :

```
def dd(n):
  L = [1]
  for nombre in range(2, n+1):
    if n % nombre == 0:
      L.append(nombre)
  return(L)
```

   Quel est le résultat de l'appel ***dd***(4) ? Puis de l'appel ***dd***(10) ? Que fait la fonction ***dd*** ?
2. Un diviseur non-trivial d'un entier naturel non nul n est un diviseur de n différent de 1 et n. Ecrire une fonction ***DNT*** d'argument **n** renvoyant la liste des diviseurs non-triviaux de l'entier **n**.
3. Écrire une fonction ***sommeCarresDNT*** d'argument **n** renvoyant la somme des carrés des diviseurs non-triviaux de l'entier **n**.
4. Écrire la suite d'instructions permettant d'afficher tous les nombres inférieurs à 1000 et égaux à la somme de leurs diviseurs non triviaux.
5. Quelle est la complexité de l'appel ***dd***(n) pour un entier n ?
6. Écrire une fonction ***ddbis*** effectuant le même travail que ***dd*** mais avec une complexité strictement inférieure (on n'exige pas que la liste retournée soit dans le même ordre pour les deux fonctions)

EXERCICE III : Vitesse de convergence

On considère le programme suivant :

```
def fct(n):
  if n==0 :
    return 1
  else :
    P = 1
    for k in range(1,n+1):
      P = P*k
    return(P)
```

1. On appelle le programme ***fct*** avec un argument appartenant à $\mathbb{N}$. Expliquer ce que ce programme nous permet de calculer.
2. On considère alors la suite $(S_n)_{n\in\mathbb{N}}$ définie pour tout $n \in \mathbb{N}$ par : $S_n = \sum_{k=0}^{n} \frac{1}{n!}$. On rappelle que $\lim_{n\to\infty} S_n = e$.
   (a) Construire un programme ***somme***, utilisant une boucle itérative et qui, pour un entier n donné, renvoie la valeur de S_n. On pourra faire appel à la fonction ***fct*** .
   (b) Déterminer un programme ***erreur*** ne prenant aucun argument et qui affiche l'erreur $|S_k - e|$ tant que $|S_k - e| > 0.001$, puis retourne le premier indice n vérifiant $|S_k - e| \leqslant 0.001$ ainsi que la valeur S_n correspondante.

PROBLEME

Ce problème constitue la première partie de l'épreuve donnée au concours de l'école polytechnique en 2013.

Points fixes de fonctions à domaine fini

Dans ce problème, 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 strictement positif $n > 0$ fixé et rangé dans une variable globale de même nom (dans la suite du sujet, on confondra l'entier n et la variable informatique `n`), et on pose $E_n = \{0, \ldots, n-1\}$. On représente une fonction $f : E_n \to E_n$ par un tableau `t` de taille n, autrement dit $f(x) =$ `t[x]` pour tout $x = 0, \ldots, n-1$. Ainsi la fonction f_0 qui à $x \in E_{10}$ associe $2x + 1$ modulo 10 est-elle représentée par le tableau `t` suivant [1] :

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

Les tableaux sont indexés à partir de 0 et la notation `t[i]` est utilisée dans les questions pour désigner l'élément d'indice `i` du tableau `t`. On suppose qu'il existe une fonction `allouer(n)` qui renvoie un tableau d'entiers de taille n, mais le contenu de ces cases est à priori quelconque.
On suppose les entiers machines signés, et on suppose que les entiers $-n, -n+1, \ldots, n-1, n$ ne débordent pas de la capacité des entiers machines — en d'autres termes, les entiers machines représentent fidèlement ces entiers. On suppose que les tableaux peuvent être passés en arguments . On note dans l'énoncé `True` et `False` les deux valeurs possibles d'un booléen. Enfin, le code écrit devra être sûr (pas d'accès invalide à un tableau, pas de division par zéro, et le programme termine, notamment) pour toutes valeurs des paramètres vérifiant les conditions données dans l'énoncé.
Le temps de calcul d'une procédure `proc` de paramètres $p_1, \ldots, p_k$ est défini comme le nombre d'opérations (accès en lecture ou écritures à une case d'un tableau ou à une variable, appel à une des primitives données dans l'énoncé) exécutées par proc pour ces paramètres ; on note $T(\texttt{prox}, n)$ le temps de calcul maximal pris sur tous les paramètres possibles pour n fixé. On dit que `proc` s'exécute en temps linéaire s'il existe des réesl $\alpha, \beta > 0$ et

1. Pour lever toute ambiguité, ce tableau serait créé avec l'instruction `t = [1,3,5,7,9,1,3,5,7,9]` .

un entier $N \geqslant 0$ tels que $\alpha n \leqslant T(\texttt{proc}, n) \leqslant \beta n$ pour tout $n \geqslant N$. De même, on dit que `proc` s'exécute en temps logarithmique s'il existe des réels $\alpha, \beta > 0$ et un entier $N \geqslant 0$ tels que $\alpha \log n \geqslant T(\texttt{proc}, n) \geqslant \beta \log n$ pour tout $n \geqslant N$.

Recherche de point fixe : cas général

On rappelle que x est un point fixe de la fonction f si et seulement si $f(x) = x$.

Question 1. Écrire une procédure `admet_point_fixe(t)` qui prend en argument un tableau `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 $2x + 1$ modulo 10.

Question 2. Écrire une procédure `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_point_fixes` devra renvoyer 1 pour le tableau donné en introduction, puisque 9 est le seul point fixe de f_0.

On note f^k l'itérée k-ième de f, autrement dit

$$\begin{aligned} f^k : E_n &\to E_n \\ x &\mapsto \underbrace{f(f(\dots f}_{k \text{ fois}}(x)) \dots) \end{aligned}$$

Question 3. Écrire une procédure `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, k de E_n et renvoie $f^k(x)$.

Question 4. Écrire une procédure `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 $k \geqslant 0$ et renvoie le nombre de points fixes de f^k.

Un élément $z \in E_n$ est dit *attracteur principal* de $f : E_n \to E_n$ si et seulement si z est un point fixe de f, et pour tout $x \in E_n$, il existe un entier $k \geqslant 0$ 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.

`i`	0	1	2	3	4	5	6
`t[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 5. Écrire une procédure `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, `False` sinon. On ne requiert pas ici une solution efficace.

On suppose à la question 6 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 la fonction 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.

Question 6. Écrire une procédure `temps_de_convergence` qui prend en premier argument un tableau `t` de taille n représentant 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. On pourra admettre que $\texttt{tc}(f, x)$ vaut 0 si x est un point fixe de f, et $1 + \texttt{tc}(f, f(x))$ si x n'est pas un point fixe de f.