Partie I Recherche de point fixe : cas général

▶ **Question 1**

```
def admet_point_fixe(t):
    for i in range(len(t)):
        if t[i]==i:
            return(True)
    return(False)
```

▶ **Question 2**

```
def nb_point_fixe(t):
    res=0
    for i in range(len(t)):
        if t[i]==i:
            res=res+1
    return(res)
```

▶ **Question 3**

```
def itere(t,x,k):
    res=x
    for i in range(k):
        res=t[res]
    return(res)
```

▶ **Question 4**

```
def nb_points_fixes_iteres(t,k):
    res=0
    for x in range(k):
        if itere(t,x,k)==x:
            res=res+1
    return(res)
```

▶ **Question 5** Pour répondre à cette question, on procède de la manière suivante : k est un attracteur principal si et seulement si $f^n(x) = k$ pour toute valeur de x. Par ailleurs, s'il existe un attracteur principal, alors il s'agit nécessairement de l'unique point fixe. On commence donc par trouver le point fixe, puis on vérifie que tout point se retrouve dessus. La démonstration est élémentaire : si pour une valeur de x on a $f^n(x) \neq k$, alors on regarde les différent $f^i(x)$ pour i allant de 0 à n. Il s'agit de $n+1$ valeurs comprises entre 0 et $n-1$, donc il y a au moins une valeur en double : $f^i(x) = f^j(x)$ pour $i \neq j$. Il s'agit alors d'un cycle (car $f^{i+1}(x) = f^{j+1}(x)$, etc.) et k ne sera pas dans ce cycle (sinon on y resterait, car k est un point fixe).

```
def admet_attracteur_principal(t):
    n=len(t)
    ptfixe = -1
    for i in range(n):
        if t[i] == i:
            ptfixe = i
    if ptfixe == -1:
        # Il n'y a pas de point fixe dans ce tableau
        return(False)
    for i in range(n):
        if itere(t,x,n)!=ptfixe:
            # On n'est pas au point fixe en n itérations, on n'y sera
            # jamais.
            return(False)
    return(True)
```

▶ **Question 6**

```
def temps_de_convergence(t,x):
    temps=0
    value=x
    while (t[value] != value):
        value = t[value]
        temps=temps+1
    return(temps)
```

▶ **Question 7** On utilise un tableau `temps` qui va contenir les temps des différentes valeurs. Initialement, on met -1 pour toutes les valeurs et 0 pour un point fixe.

On utilise aussi un tableau `aux` comme accumulateur de valeurs. Si on traite l'élément i, on va inscrire dans le tableau, successivement, $i, f(i), f^2(i), f^3(i), \ldots f^k(i)$ jusqu'à arriver à une valeur dont on connait le temps t. On donnera alors à $f^{k-1}(i)$ le temps $t+1$, puis à $f^{k-2}(i)$ le temps $t+2$ et ainsi de suite. Ainsi, s'il y a un long chemin pour arriver au point fixe, tous les éléments de ce chemin auront leur temps fixé au bout du compte.

```
def temps_de_convergence_max(t):
    n = len(t)
    temps = allouer(n)
    for i in range(n):
        if t[i] == i:
            temps[i] = 0
        else:
            temps[i] = -1
    aux = allouer(n)
    for i in range(n):
    # Traitement de l'élément i
        aux[0] = i
        k=0
        # On remplit le tableau aux
        while temps[aux[k]]==-1:
            aux[k+1] = t[aux[k]]
            k=k+1
        # On le vide
        while k>0:
            k=k-1
            temps[aux[k]] = temps[aux[k+1]]+1
    # Enfin, on trouve le maximum de temps...
    max = 0
    for i in range(n):
        if temps[i] > max:
            max = temps[i]
    return(max)
```

Partie II Recherche efficace de points fixes

Premier cas

► **Question 8**

```
def est_croissante(t):
    n = len(t)
    for i in range(n-1):
        if t[i+1] < t[i]:
            return(False)
    return(True)
```

► **Question 9** L'idée générale, c'est que la fonction $g : i \mapsto f(i) - i$ est une fonction à valeurs entières, qui ne peut décroitre qu'au plus de 1 en 1. $g(0) \geqslant 0$ (et en cas d'égalité, on a un point fixe). $g(n) \leqslant 0$ (et en cas d'égalité, on a un point fixe). Il s'agit donc de faire une recherche dichotomique.

On notera dans le programme `a` et `b` les bornes entre lesquelles on cherche le point fixe. On aura tout au long de la procédure `t[a]>a` et `t[b]<b`.

```
def point_fixe(t):
    a=0
    b=len(t)-1
    while(b-a>1):
        c=(a+b)//2
        if t[c]==c:
            return(c)
        if t[c] > c:
            a=c
        else:
            b=c
    if t[a]==a:
       return(a)
    else:
       return(b)
```

► **Question 10** Chaque boucle prend un temps constant. À chaque itération, l'entier `b-a` est divisé par deux, et le programme s'arrête quand il vaut 1. Initialement, `b-a` vaut n, donc il est réduit à 1 en au plus $\lg(n)$ opérations, d'où la complexité attendue.

Deuxième cas

► **Question 11** Soit m un plus petit élément au sens de $\preccurlyeq$, k un entier tel que $f^k(m)$ est un point fixe, et soit x un autre point fixe. Comme m est un plus petit élément, on a $m \preccurlyeq x$. Par croissance de f, on a alors $f(m) \preccurlyeq f(x) = x$, et par une récurrence immédiate, $f^k(m) \preccurlyeq f^k(x) = x$. Ainsi, pour tout x point fixe, on a $f^k(m) \preccurlyeq x$, donc $f^k(m)$ est un plus petit élément parmi les points fixes de f.

▶ **Question 12** Montrons d'abord la remarque du sujet. En fait, il faut pour le montrer regarder $f(1)$. Quelle que soit cette valeur, on a $1|f(1)$. Du coup, par une récurrence immédiate, la suite des $(f^k(1))_k$ est une suite croissante (au sens de la divisibilité), donc elle aboutit nécessairement à un point fixe, qui est minimal d'après la question 11.

Rappel : au sens de la divisibilité, 0 est le *plus grand* élément de E_n....

Tout d'abord, on note x_1 le plus petit point fixe de l'ensemble, et p le pgcd de $(x_1, \dots x_m)$. On a $p|x_1$.

On a $\forall i, p|x_i$ donc $\forall i, f(p)|f(x_i) = x_i$, donc $f(p)$ divise chaque x_i, donc il divise leur pgcd, ie $f(p)|p$. Ainsi, la suite des $(f^k(p))_k$ est décroissante au sens de la divisibilité, donc elle aboutit à un point fixe $f^k(p)$, qui divise p. Ce point fixe est un multiple de x_1 (car x_1 est le plus petit point fixe), donc on a $x_1|f^k(p)|p$, donc $p = x_1$ (par antisymétrie de la divisibilité).

▶ **Question 13**

```
def pgcd_point_fixes(t):
    res=1
    while(t[res] !=res):
        res=t[res]
    return(res)
```

En fait, d'après les questions précédentes, on sait que :

— 1 est un plus petit élément au sens de la divisibilité.

— S'il existe k tel que $f^k(1)$ est un point fixe, alors il s'agit du plus petit point fixe au sens de la divisibilité (d'après la question 11).

— Or la suite des $(f^k(1))$ est croissante au sens de la divisibilité, donc elle est stationnaire, donc un tel k existe.

— Ce plus petit point fixe au sens de la divisibilité est bien le pgcd des points fixes de f d'après la question 12

Ce qui assure la correction de l'algorithme, et sa finitude.

▶ **Question 14** On note tout d'abord que le nombre d'opérations effectué dans chaque boucle est constant, donc la complexité du programme est dominée par le nombre de passages dans la boucle. On appelle res_i les différentes valeurs prises par la variable `res` au cours du programme. À l'initialisation, $\text{res}_0 = 1$. Comme on l'a vu, on sait que pour tout i, $\text{res}_i|\text{res}_{i+1}$. On note k la valeur telle que res_k soit renvoyé.

Si $\text{res}_k \neq 0$, alors on peut en déduire que $\text{res}_k \leqslant n$, et pour chaque valeur précédente, on a $\text{res}_i < {}^{\text{res}_{i+1}}/_2$. Par une récurrence immédiate, on a $1 = \text{res}_0 < {}^{\text{res}_k}/_{2^k} < {}^n/_{2^k}$ d'où $2^k < n$ d'où $k < \lg(n)$, et le programme fonctionne bien en $O(\lg n)$.

Si $\text{res}_k = 0$, alors on peut appliquer le même raisonnement à la valeur précédente res_{k-1}, qui est nécessairement atteinte en $O(\lg(n))$ itérations au maximum, et l'itération finale ne change pas cette complexité.