Concours Blanc : corrigé du TP

PTSI A et B Lycée Eiffel

9 juin 2017

I. Quelques exemples et fonctions élémentaires.

1. Il faut simplement vérifier que k est inférieur au nombre total d'éléments de la liste (ou plutôt $k+1$ puisqu'évidemment tout est numéroté à partir de 0), et renvoyer l'élément correspondant :

```
def image(L,k) :
    if k<len(L) :
        return L[k]
    else :
        return ('La valeur de k est incorrecte')
```

2. Il suffit donc de renvoyer une liste constituée de n éléments tous égaux à a.

```
def constante(n,a) :
    return (n*[a])
```

3. On va pour cela parcourir la liste et ajouter à notre liste réponse tous les éléments dont l'image est égale à k :

```
def antecedents(L,k) :
    ant=[]
    for i in range(len(L)) :
        if L[i]==k :
            ant.append(i)
    return ant
```

4. On recopie le programme vu en cours :

```
def fmax(L) :
    m=L[0]
    for i in L[1 :] :
        if i>m :
            m=i
    return m
```

5. Parmi les très nombreuses méthodes possibles, on peut utiliser la fonction **antecedents** écrite un peu plus haut : si un entier inférieur à n n'admet pas exactement un antécédent par l'application, alors elle n'est pas bijective.

```
def bijectif(L) :
    for i in range(len(L)) :
        if len(antecedents(L,i)) !=1 :
            return False
    return True
```

II. Recherche de point fixe : cas général.

1. Il suffit de regarder élément par élément si on trouve un tier k vérifiant $f(k) = k$:

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

2. On modifie légèrement la fonction précédente pour lui adjoindre un compteur du nombre de points fixes :

```
def nombrepointsfixes(L) :
    n=0
    for i in range(len(L)) :
        if L[i]==i :
            n=n+1
    return n
```

3. Une simple boucle for fera l'affaire :

```
def iteree(L,i,k) :
    z=k
    for j in range(i) :
        z=L[z]
    return z
```

4. Une méthode brutale mais efficace consiste à vérifier si f^n prend la même valeur pour tous les entiers :

```
def attracteur(L) :
    n=len(L)
    a=iteree(L,n,0)
    for k in range(1,n) :
        if iteree(L,n,k) !=a :
            return False
    return True
```

5. Pour savoir quel est la valeur de l'attracteur, on va simplement calculer $f^n(0)$. En suite, on itère f à partir de la valeur initiale k jusqu'à obtenir une valeur égale à celle de l'attracteur.

```
def tempsconvergence(L,k) :
    p=iteree(L,len(L),0)
    t,i=0,k
    while i !=p :
        t=t+1
        i=L[i]
    return t
```

6. Pour ne pas chercher à faire subtil, on peut simplement calculer un max classique à partir de la fonction précédente (c'est loin d'être optimal !) :

```
def tempsconvergencemax(L) :
    tmax=tempsconvergence(L,0)
    for i in range(1,len(L)) :
        t=tempsconvergence(L,i)
        if t>tmax :
            tmax=t
    return tmax
```

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

1. On compare simplement chaque élément de la liste au suivant :

```
def estcroissante(L) :
    for i in range(len(L)-1) :
        if L[i+1]<L[i] :
            return False
    return True
```

2. Procédons comme suggéré par l'énoncé :

```
def poinfixedicho(L) :
    a,b=0,len(L)-1
    if L[0]==0 :
        return 0
    elif L[b]==b :
        return b
    else :
        while a<b :
            c=(a+b)//2
            if L[c]<c :
                b=c
            elif L[c]>c :
                a=c
            else :
                return c
    return ('La liste proposée ne vérifie pas les hypothèses !')
```

3. Écrivons un petit programme pour créer une liste de n éléments alétoires rangés par ordre croissant :

```
from random import randint
def listealea(n) :
    l=[randint(0,n-1) for i in range(n)]
    l.sort()
    return l
```

Ensuite, on crée une telle liste (une seule, si on en recrée une pour chaque test, on va compter le temps de création de liste en plus du temps d'exécution de notre algorithme, ce n'est pas

le but), puis on cherche un point fixe dedans 10 000 fois de suite (par exemple pour une liste de 1 000 éléments) :

```
from time import time
liste=listealea(1000)
a=time()
for i in range(10000) :
        pointfixedicho(liste)
b=time()
print(b-a)
```

J'obtiens les temps suivants pour différentes valeurs de n (ça peut valoir le coup de tenter avec plusieurs listes différentes, car on peut assez facilement tomber sur une liste pour laquelle le point fixe va être trouvé très vite, notamment si 0 est point fixe, auquel cas on ne mesurera rien du tout), exprimés en millisecondes.

n	1000	10^4	10^5	10^6
temps	7	8	8	9

Les temps indiqués sont plus ou moins des moyennes après plusieurs essais. En fait, on a occasionnellement un temps d'exécution de l'ordre de 3 à 10 fois supérieur (en gros quand il n'y a pas de point fixe au bord) qui croit un peu plus rapidement que le temps moyen. Mais dans l'ensemble la construction de la liste aléatoire prend (très nettement) plus de temps que le traitement de l'algorithme lui-même.