Corrigé TP3 Dictionnaire et parcours linéaire

September 27, 2021

1 Recherche séquentielle dans une liste

Q1

[2]:
```
def recherche(x,L):
    n=len(L)
    for k in range(n):
        if L[k]==x:
            return True
    return False #si après avoir comparé tous les termes on ne trouve pas x, on renvoie False. Attention à l'indentation du return, qui stoppe la fonction

##Test
L=[3,4,17,5,2,1]
print(recherche(2,L))
print(recherche(6,L))
```

```
True
False
```

Q2 Le meilleur des cas est lorsque l'on trouve directement x dès le premier terme de la liste ; le pire des cas est lorsque le terme recherché est à la toute fin. La durée moyenne de recherche est donc proportionnelle à la taille de la liste.

Q3

[4]:
```
def compte(x,L):
    N=0
    n=len(L)
    for k in range(n):
        if L[k]==x:
            N+=1#on incrémente le compteur si on trouve une occurrence de x
    return N #on retourne le compteur
```

Le coût sera linéaire car on balaie l'intégralité de la liste, et donc le nombre d'opération est bien une fonction affine de n.

Q4

[5]:
```
def indices(x,L):
    lpos=[]#on définit une liste vide
    n=len(L)
    for k in range(n):
        if L[k]==x:
            lpos.append(k)#on conserve la position de l'une des occurrences de x
    return lpos #on retourne a liste

print(indices(3,[1,2,3,2,1,2,3,1,2]))
```

```
[2, 6]
```

2 Recherche du maximum d'une liste

Q5

[7]:
```
def recherche_maxi(L):
    maxprov=L[0]# on initialise le maximum avec le premier terme de la liste,␣
 ↪pas avec 0 car on pourrait imaginer des valeurs toutes négatives pour L !
    for k in range(1,len(L)):#on démarre à L[1]
        if L[k]>maxprov:
            maxprov=L[k]#on affecte le nouveau max provisoire
    return maxprov

L1=[1.2,1.5,1.2,12,15,14,15,17] #max est le dernier
L2=[19,1.2,1.5,1.2,12,15,14,15] #max est le premier
L3=[1.2,1.5,1.2,17,12,15,14,17,15] #max est en double
L4=[-1.2,-1.5,-0.7,-17,-12,-15,-14,-17,-15] #max est négatif
L5=[1.2,1.5,1.2,12,17,15,14,15] #max est à l'intérieur
print(recherche_maxi(L1),recherche_maxi(L2),recherche_maxi(L3),recherche_maxi(L4),recherche_ma
```

```
17 19 17 -0.7 17
```

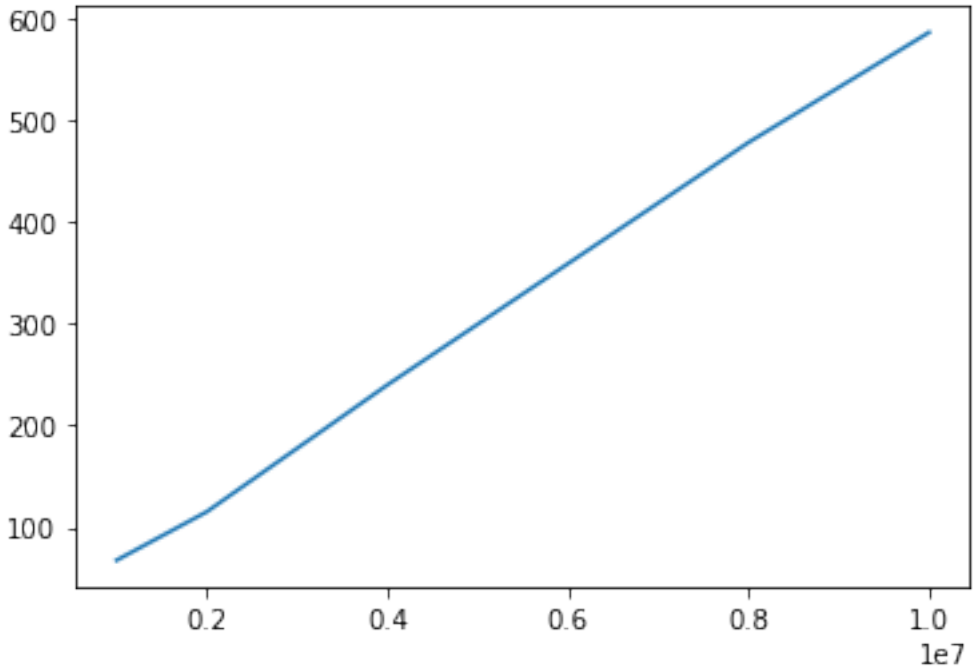
Q6 Il faut toujours balayer toute la liste pour savoir si le maximum est bien atteint, il n'y a donc pas de distinction entre meilleur et pire des temps. La complexité est bien linéaire en temps, car les opérations à effectuer sont bien proportionnelles à la longueur de la liste !

Q7

[21]:
```
import random as rd
from time import time
for nliste in (10**6,2*10**6,4*10**6,8*10**6,10**7):
    Lalea=[rd.uniform(-60,80) for k in range(nliste)]
    t1=time()
    recherche_maxi(Lalea)
    t2=time()
    print(nliste,"Le temps mis pour exécuter le code est de {}ms".
 ↪format(round((t2-t1)*1000)) )
```

```
1000000 Le temps mis pour exécuter le code est de 61ms
2000000 Le temps mis pour exécuter le code est de 109ms
4000000 Le temps mis pour exécuter le code est de 234ms
8000000 Le temps mis pour exécuter le code est de 476ms
10000000 Le temps mis pour exécuter le code est de 616ms
```

[22]:
```python
##Pour tracer un graphique
import matplotlib.pyplot as plt
x=[10**6,2*10**6,4*10**6,8*10**6,10**7]
y=[]
for nliste in (10**6,2*10**6,4*10**6,8*10**6,10**7):
    Lalea=[rd.uniform(-60,80) for k in range(nliste)]
    t1=time()
    recherche_maxi(Lalea)
    t2=time()
    y.append((t2-t1)*1000)
plt.plot(x,y)
plt.show()
```



On constate bien que l'évolution temporelle est affine de la longueur de la liste, conformément à ce qui était attendu !

3 Dictionnaire

Q8

[57]:
```
produits = {"pomme" : 25, "poire" : 43, "orange": 27,"noix" : 5}
print(produits)
L=produits.keys()
print(type(L)) #ce n'est pas une liste au sens où on peut la manipuler !
print(produits.values())
print(produits.items())
```

```
{'pomme': 25, 'poire': 43, 'orange': 27, 'noix': 5}
<class 'dict_keys'>
dict_values([25, 43, 27, 5])
dict_items([('pomme', 25), ('poire', 43), ('orange', 27), ('noix', 5)])
```

Q9

[58]:
```
produits["pomme"]
```

[58]: 25

Q10

[59]:
```
produits["noix"]=10
produits
```

[59]: {'pomme': 25, 'poire': 43, 'orange': 27, 'noix': 10}

Q11

[60]:
```
L=produits.items()
for couple in L: #boucle sur la liste, couple prend les valeurs successives de␣
 ↪la liste (mais L[k] vous renverra une erreur !)
    print(couple[0],' : ',couple[1])
```

```
pomme  :  25
poire  :  43
orange  :  27
noix  :  10
```

Q12

[61]:
```
def augmente(dico,fruit):
    if dico.get(fruit)!=None: #le fruit est dans le dictionnaire
        dico[fruit]+=1
    else:
        dico[fruit]=1
for k in range(18):
    augmente(produits,"banane")
produits
```

[61]: {'pomme': 25, 'poire': 43, 'orange': 27, 'noix': 10, 'banane': 18}

Q13

[71]:
```
def maxid(dico):
    maxi=0#pour initialiser, on suppose ici uniquement des valeurs positives,␣
 ↪impossible de prendre dico[dico.keys[0]] !
    cle_maxi=0
    for cle in dico:
        if maxi<dico[cle]:
            maxi=dico[cle]
            cle_maxi=cle
    return cle_maxi
print(maxid(produits))
```

```
poire
```

4 Utilisation d'un dictionnaire pour compter des éléments ou faire une étude fréquentielle

Q14

[100]:
```
chaine="ladifficileadaptationdelavieadesconditionsextremes"

def comptage_lettres(chaine):
    Nlettres={}
    for k in range(len(chaine)):
        augmente(Nlettres,chaine[k])
    return Nlettres
dicolettre=comptage_lettres(chaine)
print(dicolettre)
```

```
{'l': 3, 'a': 6, 'd': 5, 'i': 7, 'f': 2, 'c': 2, 'e': 7, 'p': 1, 't': 4, 'o': 3,
'n': 3, 'v': 1, 's': 3, 'x': 1, 'r': 1, 'm': 1}
```

Q15

[102]:
```
chaine="ladifficileadaptationdelavieadesconditionsextremes"

def freq_lettres(chaine):
    dico={}
    for k in range(len(chaine)):
        if dico.get(chaine[k])!=None: #le fruit est dans le dictionnaire
            dico[chaine[k]]+=1/len(chaine)
        else:
            dico[chaine[k]]=1/len(chaine)
    return dico
Nlettres=freq_lettres(chaine)
print(Nlettres)
```

```
{'l': 0.06, 'a': 0.12000000000000001, 'd': 0.1, 'i': 0.14, 'f': 0.04, 'c': 0.04,
'e': 0.14, 'p': 0.02, 't': 0.08, 'o': 0.06, 'n': 0.06, 'v': 0.02, 's': 0.06,
'x': 0.02, 'r': 0.02, 'm': 0.02}
```

[103]:
```
f = open("le_tour_du_monde_en_80_jours_utf8.txt", "rt", encoding="utf8")
texte = f.read()
f.close()
texte=texte.lower()
import string
texte2=texte.translate(str.maketrans('','',string.punctuation))
listedesmots=texte2.replace("\n"," ").split(" ") #on crée une liste de mots
listedesmots=list(map(str.strip,listedesmots)) #suppression des espaces␣
 ↪superflus
listedesmots= [i for i in listedesmots if i != '']
print(listedesmots[0:50])
```

```
['le', 'tour', 'du', 'monde', 'en', 'quatrevingts', 'jours', 'par', 'jules',
'verne', 'i', 'dans', 'lequel', 'phileas', 'fogg', 'et', 'passepartout',
'sacceptent', 'réciproquement', 'lun', 'comme', 'maître', 'lautre', 'comme',
'domestique', 'en', 'lannée', '1872', 'la', 'maison', 'portant', 'le', 'numéro',
'7', 'de', 'savillerow', 'burlington', 'gardens', 'maison', 'dans', 'laquelle',
'sheridan', 'mourut', 'en', '1814', 'était', 'habitée', 'par', 'phileas',
'fogg']
```

[104]:
```
Nmots=freq_lettres(listedesmots)
maxid(Nmots)
```

[104]:
```
'de'
```

[ ]:
```

```