Travaux pratiques 1

Recherche séquentielle, dictionnaires, comptage

Informatique tronc commun MPSI et PCSI

Certains exercices, parmi les 6 premiers, ont déjà été traités dans d'autres T.P.
vous pourra les traiter rapidement ou les passer si vous savez vraiment bien les faire.

I Recherche simple dans une liste ou un texte

I.1 Recherche d'un maximum

Une liste est une collection ordonnée (par les indices) de valeurs. Les indices commencent à 0 et sont des entiers naturels successifs.
On peut parcourir une liste par ses indices comme par exemple dans le programme suivant qui cherche la valeur maximale d'une liste de *float*.

```
def monMax(L):
    """ENTREE: une liste de flottants
       SORTIE: la plus grande valeur de la liste"""
    max_var = L[0] # initialisation
    n = len(L)
    for i in range(n):    # On utilise les indices
        if L[i] > max_var:
            max_var = L[i]
    return max_var
```

On peut parcourir une liste par ses valeurs comme par exemple dans le programme suivant qui cherche la valeur maximale d'une liste de *float*. On notera que l'ordre importe alors peu.

```
def monMaxVal(L):
    """ENTREE: une liste de flottants
       SORTIE: la plus grande valeur de la liste"""
    max_var = L[0] # initialisation
    for x in L:    # On utilise les valeurs
        if x > max_var:
            max_var = x
    return max_var
```

Exercice 1 - indice max
Écrire une fonction qui renvoie l'indice d'un maximum d'une liste L de nombres.

Exercice 2 - deuxième max
Écrire une fonction qui renvoie le deuxième maximum d'une liste de nombres c'est-à-dire le terme le plus grand strictement moins grand que le maximum de la liste.

I.2 Recherche d'un élément

Cherchons maintenant si un élément est présent dans une liste
On peut utiliser

```
8 in L
```

par exemple pour chercher si un élément est présent dans une liste mais ceci ne permet pas de bien comprendre le fonctionnement de la machine. La complexité de l'instruction `8 in L` est en $\mathcal{O}(n)$, c'est-à-dire à peu près proportionnelle à la longueur n de la liste L.

Exercice 3 - test d'apparition
Sans utiliser l'instruction **in** directement, écrire une fonction qui renvoie `True` si l'élément x est un terme de la liste L et qui renvoie `False` sinon.
Quelle est la complexité dans le pire des cas du programme proposé ?

Exercice 4 - test d'apparition dans une chaine
Sans utiliser l'instruction **in** directement, écrire une fonction qui renvoie `True` si la lettre x est une lettre du texte `tex` et qui renvoie `False` sinon.
Quelle est la complexité dans le pire des cas du programme proposé ?

Exercice 5 - rang d'apparition
Écrire une fonction qui renvoie l'indice de la première occurrence de l'élément x si x est un terme de la liste L et qui renvoie `False` sinon.

Exercice 6 - nombre d'apparitions
Écrire une fonction qui renvoie le nombre d'apparitions de l'élément x si x est un terme de la liste L et qui renvoie 0 sinon.

I.3 Inventaire des éléments d'une liste

Si on considère un nombre écrit sous la forme d'une chaine de caractères, comme par exemple `nbc='31415926535'` on peut compter le nombre d'apparitions de chaque chiffre.
Ici il y a zéros 0, deux 1, un 2, $\cdots$.
On pourrait regrouper cela dans la liste `chiffres =[0, 2, 1, 2, 1, 3, 1, 0, 0, 1]` ; le programme suivant effectue ce travail en remplissant au fur et à mesure les "cases" de la liste `chiffres`.

```
def inventaire(nombre):
    """ENTREE: une chaine de caractère composée
              de chiffres exclusivement
       SORTIE: la liste des nombres d'apparitions
              de chaque chiffre dans la chaine"""
    chiffres = 10*[0] #au départ le total de chaque chiffre est à
       0
    for x in nombre:
        chiffres[int(x)] += 1
        # remarque : x est une chaine, on la convertit en entier
    return chiffres
```

Le problème se corse si on veut dénombrer le nombre d'apparition de chaque lettre dans une phrase par exemple.

Exercice 7 - ACGT
Écrire une fonction inspirée de la précédente et qui renvoie le nombre d'apparitions de chaque lettre A, C, G, T dans une chaine de caractères constituée uniquement des lettres A, C, G, T.
On se rend compte de la difficulté de cette façon de compter. Cela peut motiver l'introduction des dictionnaires.

II Les dictionnaires

II.1 Un peu de cours

Une liste est une façon de construire une collection d'objets mis l'un après l'autre.

Définition 1 : dictionnaire

Un dictionnaire est une collection de valeurs quelconques indexées par des clés (on peut parler de structure de données associative).

À retenir : à propos des dictionnaires

- Les valeurs sont des objets quelconques ; penser aux définitions dans un dictionnaire de langue par exemple.
- Les clés sont des objets quelconques non mutables (donc les clés ne sont pas des listes ni des dictionnaires) ; penser aux mots à définir, aux entrées dans un dictionnaire de langue.
- Une clé n'apparait qu'une fois dans un dictionnaire.
- On peut modifier la valeur associée à une clé.
- On peut ajouter une clé ou la supprimer.
- On peut compter les entrées (clés) d'un dictionnaire.
- On peut parcourir un dictionnaire.
- Il n'y a pas d'ordre dans un dictionnaire. (l'analogie avec le dictionnaire de langue ne convient plus ici)

Il peut sembler difficile pour la machine de retrouver une clé (et la valeur associée) puisque le dictionnaire n'est pas ordonné (contrairement à un dictionnaire de langue). Cet accès est facilité par l'usage d'une fonction de hachage (ou hashage). Cette fonction calcule pour chaque entrée (clé)

un nombre entier qui permet de retrouver rapidement la valeur associée à la clé dans la mémoire de la machine. Le coût de la recherche d'une clé est constant. (il est linéaire pour la recherche d'une valeur dans une liste).
On n'entrera pas dans les détails mais on retiendra que la recherche d'une clé n'est alors pas guère plus coûteuse que la recherche d'un indice d'une liste.

À retenir : un peu de syntaxe

- Création d'un dictionnaire vide

```
mon_dico = dict()
```

- Ajout d'une clé `'prem'` associée à la valeur 2

```
mon_dico['prem'] = 2
```

- Modification de la valeur associée à la clé `'prem'`

```
mon_dico['prem'] = 4
```

- Ajout d'une clé 2 associée à la valeur `[2, 'g']`

```
mon_dico[2]= [2, 'g']
```

- Suppression de l'entrée de clé `'prem'`

```
del mon_dico['prem']
```

Exercice 8 - parcours de dictionnaire
Créer un dictionnaire `dico` avec quelques clés puis essayer le programme suivant.

```
for x in dico:
    print(x)
    print(dico[x])
```

Quel texte s'affiche ?

II.2 Inventaire des lettres à l'aide d'un dictionnaire

Exercice 9 - dicoACGT
Ecrire une fonction qui renvoie le nombre d'apparitions de chaque lettre A, C, G, T dans une chaine de caractères constituée uniquement des lettres A, C, G, T en utilisant un dictionnaire indexé par les clés A, C, G et T.

Exercice 10 - quelles lettres ?
Écrire une fonction `inventaireLettre` qui renvoie un dictionnaire qui compte le nombre d'apparitions de chaque lettre dans un texte.

Pour tester l'apparition d'une clé dans un dictionnaire, on utilisera `in` qui a une complexité constante (donc faible) grâce à l'utilisation des tables de hachage.
Par exemple `a in mon_dico` renvoie `True` si `a` est une clé du dictionnaire `mon_dico` et `False` sinon.

Exercice 11 - apparition de lettre
Ecrire une fonction qui teste l'apparition d'une lettre dans un texte. On pourra utiliser la fonction `inventaireLettre` de l'exercice 10.

Exercice 12 - plus fréquente
Ecrire une fonction qui trouve la (ou les) lettres qui apparaissent le plus souvent dans un texte.

II.3 Recherche d'inverse modulo N

On se donne une liste d'entiers notée L de longueur p et un nombre entier naturel non nul N.
On cherche si il existe un couple d'entiers (i, j) entre 0 et $p-1$ qui vérifie $L[i]+L[j] = N$ et $i < j$.
On testera les fonctions avec `L0=[10, 3, 4, 6, 8, 5,2,2]`

Exercice 13 - existence naïve
Écrire une première fonction qui renvoie un de ces couples si il existe, elle renverra 0 sinon. Par exemple pour `L0` et $N = 12$ on pourra obtenir $(0, 6)$. On impose dane cette question de ne pas utiliser de dictionnaire.
On observera que la complexité est $\mathcal{O}(n^2)$. (elle utilise deux boucles imbriquées).

Exercice 14 - existence avec dictionnaire
Écrire une deuxième fonction qui utilise les dictionnaires et dont la complexité est $\mathcal{O}(n)$. Pour cela on pourra commencer par créer un dictionnaire qui prend pour clé les valeurs x de la liste et pour valeur un des indices i de la liste pour lesquels `L[i]=x`. Sur l'exemple on obtiendrait par exemple `{2: 7, 3: 1, 4: 2, 5: 5, 6: 3, 8: 4, 10: 0}`.
Le dictionnaire nous permet ici d'échanger les rôles des indices et des valeurs

On cherche ensuite tous les couples d'entiers (i, j) entre 0 et $p-1$ qui vérifient `L[i] + L[j] = N` et $i < j$, par exemple pour `L0` et $N = 12$ on cherche à obtenir `[[0, 6], [0, 7], [2, 4]]`.

Exercice 15 - liste naïve
Écrire une première fonction qui donne la liste de ces couples, qui n'utilise pas de dictionnaire et dont la complexité est $\mathcal{O}(n^2)$. (elle utilise deux boucles imbriquées).

Exercice 16 - liste avec dictionnaire
Écrire une deuxième fonction qui utilise les dictionnaires et dont la complexité est $\mathcal{O}(n)$. Pour cela on pourra commencer par créer un dictionnaire qui prend pour clé les valeurs x de la liste et pour valeur la liste des indices i de la liste pour lesquels `L[i]=x`.
Sur l'exemple on obtiendrait
`{8 : [4], 2 : [6,7], 3 : [1], 4 : [2], 6 : [3], 5 : [5], 10 : [0]}`.

II.4 Inventaire des mots à l'aide d'un dictionnaire

On utilisera le fichier `littleMermaid.txt` qui est en anglais ce qui évite d'avoir des accents ; il est, de plus, purgé de sa ponctuation, il ne reste donc que les mots et les espaces.

Exercice 17 - liste des mots
Écrire les instructions qui permettent de créer la liste des mots contenus dans le texte.
Pour obtenir la liste des mots, on utilisera la méthode de `str` qui se nomme `split` (penser à utiliser l'aide de Python avec `help`).

Exercice 18 - inventaire des mots
Créer le dictionnaire des occurrences des mots du texte qui associe à chaque mot son nombre d'apparitions.

Exercice 19 - nombre d'apparitions
Créer un dictionnaire dont les clés sont les nombres n d'apparitions des mots et dont la valeur associée à la clé n est la liste des mots apparaissant n fois
Quels mots apparaissent 31 fois ?
Quelles fréquences inférieures à 31 n'apparaissent pas ?

II.5 Anagrammes

On reprend la liste des mots de la partie précédente.
Pour un mot donné, si on ordonne ses lettres dans l'ordre alphabétique, on obtient sa signature ; par exemple `'cehin'` est la signature de `'chien'` mais aussi de son anagramme `'niche'`.
On peut utiliser la fonction `sorted` qui donne la liste ordonnée des lettres de la chaîne passée en paramètre : `sorted(truc)` donne `['c', 'r', 't', 'u']`.
On doit ensuite recoller les lettres, cela peut se faire par la méthode `.join` :
`'a'.join(['c','r','t','u'])` donne le mot `'caratau'` : la chaîne `'a'` est placée entre toutes les chaînes de la liste. Un recollement simple se fait en appliquant `.join` à la chaîne vide : `''.join(sorted('truc'))` donne le mot `'crut'`.

Exercice 20
Créer un dictionnaire dont les clés sont les signatures des mots de la liste et les valeurs sont les listes des mots associés.
À la clé `'cehin'` pourra être associée la liste `['chien', 'niche']` par exemple.

Exercice 21
Créer une liste des listes d'anagrammes qui ont une longueur au moins 2.

Solutions

Solution de l'exercice 1 - indice max

```
def monMaxInd(L):
    """ENTREE: une liste de flottants
       SORTIE: un indice de la plus grande valeur de la liste"""
    max_var = L[0] #initialisation
    indice=0
    n = len(L)
    for i in range(n):
        if L[i] > max_var:
            max_var = L[i]
            indice=i
    return indice
```

Solution de l'exercice 2 - deuxième max

```
def secondMax(L):
    """ENTREE: une liste de flottants
       SORTIE: la deuxième plus grande (strictement) valeur de la
          liste"""
    n = len(L)
    if L[1]>L[0]:
        max_var = L[1] #initialisation
        sec_max = L[0]
    else:
        max_var = L[0]
        sec_max = L[1]
    for i in range(2,n):
        if L[i] > max_var:
            sec_max = max_var
            max_var = L[i]
        elif L[i] > sec_max and L[i] != max_var:
            sec_max = L[i]
    if max_var == sec_max: #cas ou il n'existe pas de second max
       strict
        return 'liste constante'
    return sec_max
```

Solution de l'exercice 3 - test d'apparition

```
def estDans(x,L):
    """ENTREE: un élément quelconque et une liste quelconque
    SORTIE: un booléen True si et seulement si x est une valeur de
        la liste"""
    for y in L:
        if x==y:
            return True #sortie définitive
    return False
```

Solution de l'exercice 4 - test d'apparition dans une chaine

```
def estDansChaine(lettre,tex):
    """ENTREE: lettre est une chaine de caractère de longueur 1
    et tex une chaine de caractère
    SORTIE: un booléen True si et seulement si lettre est un
       caractère de tex"""
    for y in tex:
        if lettre==y:
            return True
    return False
```

Solution de l'exercice 5 - rang d'apparition

```
def indOcc(x,L):
    """ENTREE: x est un élément quelconque et L une liste
       quelconque
    SORTIE: l'indice d'apparition de la valeur x dans la liste L (
       False sinon)"""
    n=len(L)
    for i in range(n):
        if L[i]==x:
            return i #sortie définitive
    return False #cas où L[i] toujours différent de x
```

Solution de l'exercice 6 - nombre d'apparitions

```
def nbOcc(x,L):
    """ENTREE: un élément quelconque et une liste quelconque
    SORTIE: le nombre d'apparitions de x dans la liste L"""
    compteur=0
    for y in L:
        if y==x:
            compteur+=1
    return compteur
```

Solution de l'exercice 7 - ACGT

```
def ACGT(texte):
    """ENTREE: une chaine de caractère composée
               des lettres A, C, G, T exclusivement
       SORTIE: la liste des nombres d'apparitions
               de chaque lettre dans la chaine"""
    compteur=[0, 0, 0, 0]# première case pour le nombre de A....
    lettres=['A', 'C', 'G', 'T']
    for x in texte:
        if x == lettres[0]:
            compteur[0] += 1
        elif x == lettres[1]:
            compteur[1] += 1
        elif x = =lettres[2]:
            compteur[2] += 1
        else:
            compteur[3] += 1
    return compteur
```

Solution de l'exercice 8 - parcours de dictionnaire

Solution de l'exercice 9 - dicoACGT

```
def dicoACGT(texte):
    """ENTREE: une chaine de caractère composée
    des lettres A, C, G, T exclusivement
    SORTIE: la liste des nombres d'apparitions
    de chaque lettre dans  la chaine en utilisant un dictionnaire
       """
    compteur=dict()
    compteur['A']=0
    compteur['C']=0
    compteur['G']=0
    compteur['T']=0
    for x in texte:
        compteur[x]+=1
    return compteur
```

Solution de l'exercice 10 - quelles lettres ?

```
def inventaireLettre(texte):
    """ENTREE: une chaine de caractère : texte
    SORTIE: la liste des nombres d'apparitions
    de chaque lettre dans la chaine en utilisant un dictionnaire
       """
    compteur=dict()
    for x in texte:
        if x in compteur:
            compteur[x]+=1
        else:
            compteur[x]=1
    return compteur
```

Solution de l'exercice 11 - apparition de lettre

```
def appartenanceLettre(lettre,texte):
    """ENTREE: une chaine (lettre) contenant un seul caractère
    et une chaine de caractère (texte)
    SORTIE: un booléen vrai si et seulement si lettre est dans
       texte"""
    dico=inventaireLettre(texte)
    return lettre in dico
```

Solution de l'exercice 12 - plus fréquente

```
def frequenceMax(texte):
    """ENTREE: une chaine de caractère : texte
    SORTIE: les lettres qui apparaissent le plus souvent dans
       texte"""
    dico=inventaireLettre(texte)
    maxi=0
    lettres=[]
    for x in dico:
        if maxi < dico[x]:
            maxi = dico[x]
            lettres=[x]
        elif maxi == dico[x]:
            lettres.append(x)
    return lettres
```

Solution de l'exercice 13 - existence naïve

```
def unInverse(liste,N):
    """ENTREE: une liste de nombres : liste et un entier : N
    SORTIE: un couple d' indices (i,j) tels que liste[i]+liste[j]=
       N """
    p=len(liste)
    for i in range(p):
        for j in range(i,p): #i<j
            if liste[i]+liste[j]==N:
                return (i,j)
    return 0
```

Solution de l'exercice 14 - existence avec dictionnaire

```
def unInverseAvecDico(liste,N):
    """ENTREE: une liste de nombres : liste et un entier : N
    SORTIE: un couple d' indices (i,j) tels que liste[i]+liste[j]=
       N
    en utilisant un dictionnaire"""
    p=len(liste)
    dico=dict()
    # on cree d abord le dico des valeurs
    for i in range(p):
        dico[liste[i]]=i
    #on cherche alors un couple solution
    for x in dico:
        if N-x in dico:
            return [dico[x],dico[N-x]] #fin de boucle dès qu'on a
               une réponse
    return 0
```

Solution de l'exercice 15 - liste naïve

```
def inverseAvecListe(liste,N):
    """ENTREE: une liste de nombres : liste et un entier : N
    SORTIE: la liste des couples d' indices (i,j) tels que liste[i
       ]+liste[j]=N """
    p=len(liste)
    solu=[] #initialisation
    for i in range(p):
        for j in range(i+1,p): #i<j
            if liste[i]+liste[j]==N:
                solu.append([i,j])
    return solu
```

Solution de l'exercice 16 - liste avec dictionnaire

```
def inverseAvecDico(liste,N):
    """ENTREE: une liste de nombres : liste et un entier : N
    SORTIE: la liste des couples d' indices (i,j) tels que liste[i
       ]+liste[j]=N
    en utilisant un dictionnaire"""
    p=len(liste)
    dico=dict()
    # on cree d abord le dico des valeurs
    for i in range(p):
        if liste[i] in dico:
            dico[liste[i]].append(i)
        else:
            dico[liste[i]]=[i]
    # on cherche alors les couples solutions
    solu=[]
    for x in dico:
       if N-x in dico:
           for i in dico[x]:
               for j in dico[N-x]:
                   if i<j:
                       solu.append([i,j])
    return solu
```

Solution de l'exercice 17 - liste des mots

```
nomDeFichier = "littleMermaid.txt"
fichier = open(nomDeFichier,'r') # ouvrir le fichier
listeContenu = fichier.readlines() # son contenu dans une liste
fichier.close() # fermer le fichier
mots = listeContenu[0].split(' ') # liste des mots
```

Solution de l'exercice 18 - inventaire des mots

```
dicomots = dict() # initialisation, dictionnaire vide
for x in mots:
    if x in dicomots:
        dicomots[x] += 1
    else:
       dicomots[x]=1
```

Solution de l'exercice 19 - nombre d'apparitions

```
dicoFreq=dict() #initialisation
for x in dicoMots:
    if dicoMots[x] in dicoFreq:
        dicoFreq[dicoMots[x]].append(x)
    else:
         dicoFreq[dicoMots[x]]=[x]

absents = []
for i in range(1,31):
    if i in dicoFreq:
        pass # on ne fait rien
    else:
        absent.append(i)
```

Solution de l'exercice 20

```
def anagramme(dicoMots):
    anag=dict()
    for x in dicoMots:
        xord = ''.join(sorted(x)) # xord est la signature de x
        if xord in anag:
            anag[xord].append(x)
        else:
            anag[xord]=[x]
    return anag
```

Solution de l'exercice 21

```
def listeAnag(dicoMots):
    anag=anagramme(dicoMots)
    liste=[]
    for x in anag:
        if len(anag[x]) > 1:
            liste.append(anag[x])
    return liste
```