Cours : Algorithmes et méthodes à connaître

ECG 1

Décembre 2021

Table des matières

1 Méthodes générales sur les listes et les tableaux	2
1.1 Création d'une liste	2
1.2 Parcours séquentiel d'une liste, avec indice	2
1.3 Parcours séquentiel d'une liste, sans indice	2
1.4 Parcours séquentiel, avec interruption	2
Avec un while et un drapeau ▲	2
2 Algorithmes à connaître sur les listes	2
2.1 Maximum et minimum	2
2.1.1 Position	2
2.1.2 Valeur avec range	3
2.1.3 Valeur sans range	3
2.1.4 Adaptations à savoir faire	3
2.1.5 Adaptations avancées ▲	3
2.2 Deuxième maximum	3
2.2.1 Adaptations à savoir faire	3
2.3 Occurrences	3
2.3.1 Fonctions déjà présentes dans python	4
2.3.2 Première occurrence	4
Avec for	4
Avec While et drapeau ▲	4
Adaptations à savoir faire	4
2.3.3 Présence d'un élément	4
For sans range	4
While▲	5
Adaptations à savoir faire	5
2.3.4 Nombre d'occurrence	5

Version sans range	5
Version avec range	5
Liste des occurrences ▲	5
3 Algorithmes plus avancés sur les listes, boucles imbriquées	5
3.1 Recherche par dichotomie dans une liste triée	5
3.1.1 Adaptation à savoir faire	6
3.2 Trouver les deux éléments les plus éloignés	6
3.2.1 Valeur	6
3.2.2 Positions	6
3.2.3 Adaptations ▲	6
3.3 Recherche d'un motif ▲	6
3.3.1 Adaptations	7
4 Autres algorithmes	7
4.1 Suite définies par récurrence	7
Adaptations à savoir faire	7
Adaptations avancées ▲	7
4.2 Calcul de somme de produit	7
4.2.1 Somme et produit simple	7
Structure	7
Exemples	8
4.2.2 Utilisation de boucles imbriquées	8
4.2.3 Utilisation en statistiques	8
Moyenne des éléments d'une liste	8
Variance des éléments d'une liste	8
Adaptation	9
4.3 Algorithmes gloutons	9
4.3.1 Rendu de monnaie	9
4.3.2 Allocation de salle ▲	9

Remarques

Le symbole ▲ indique les parties que vous pouvez ignorer lors de la première lecture de ce document.

1 Méthodes générales sur les listes et les tableaux

1.1 Création d'une liste

La principale façon de créer une liste est de partir d'une liste vide et de rajouter des éléments à l'aide d'une boucle `for` ou `while` et la méthode `.append()`.

```
L=[]
for .....:
    L.append(.....)
```

1.2 Parcours séquentiel d'une liste, avec indice

Dans ce type d'algorithme, on doit parcourir l'ensemble de la liste, et on s'intéresse à l'indice des éléments. On utilise

```
for i in range(len(L)):
    ...#pour accéder à l'élément on utilise L[i]
```

Exemples

- Recherche de l'indice de l'élément maximum, de l'élément minimum.
- Recherche du nombre d'occurrence d'un élément x dans la liste, de la liste des indices d'un éléments.
- Calcul d'une somme, d'un produit d'une moyenne.

1.3 Parcours séquentiel d'une liste, sans indice

Dans ce type d'algorithme, on doit parcourir l'ensemble de la liste, et on s'intéresse à la valeur des éléments. On utilise

```
for x in L:
    ...#pour accéder à l'élément on utilise x
```

Exemples

- Recherche de la valeur de l'élément maximum, de l'élément minimum.
- Recherche du nombre d'occurrence d'un élément x dans la liste .
- Calcul d'une somme, d'un produit d'une moyenne

1.4 Parcours séquentiel, avec interruption

Dans ce type d'algorithme on parcourt la liste jusqu'à une certaine condition soit réalisée. On trouve souvent le terme "premier élément" dans l'énoncé.

Exemples

- Première occurrence d'un élément x dans une liste L.
- Présence ou absence d'un élément x dans une liste L.
- La liste est elle triée dans l'ordre croissant ?

```
def f() :
    for i in range (len(L)) :
        if :
            return # Le return arrete la boucle et la
                ↪ fonction
    return # valeur si la Liste a été parcourue
                ↪ en entier
```

Avec un while et un drapeau ▲

```
drapeau=False
i=0
while i<len(L) and drapeau=False:
    if .... :
        drapeau =True
    i=i+1
```

2 Algorithmes à connaître sur les listes

2.1 Maximum et minimum

2.1.1 Position

```

def max(L):
    ''' position du maximum, on suppose la liste non
    ↪ vide renvoie le plus petit entier i tel que
    ↪ L[i]= Max(L)'''
    resultat=0
    for i in range(len(L)):
        if L[i]>L[resultat]:
            resultat=i
    return resultat

```

2.1.2 Valeur avec range

```

def max(L):
    ''' valeur du maximum, on suppose la liste non
    ↪ vide renvoie la valeur de Max(L)'''
    resultat=L[0]
    for i in range(len(L)):
        if L[i]>resultat:
            resultat=L[i]
    return resultat

```

2.1.3 Valeur sans range

```

def max(L):
    ''' valeur du maximum, on suppose la liste non
    ↪ vide renvoie la valeur de Max(L)'''
    resultat=L[0]
    for x in L:
        if x>resultat:
            resultat=x
    return resultat

```

2.1.4 Adaptations à savoir faire

En changeant l'inégalité en $<$, $\geq$, $\leq$ vous devez pouvoir écrire une fonction qui :

- Donne l'indice du maximum, et si il y a plusieurs maxima , renvoie l'indice le plus grand.
- Donne l'indice le plus grand ou le plus petit, ou la valeur du minimum.

2.1.5 Adaptations avancées ▲

- Valeur et position du maximum.
- Valeur du maximum et nombre de fois où le maximum apparaît.
- Liste des positions des maxima.

2.2 Deuxième maximum

```

def deuxiememax(L):
    ''' renvoie la valeur du deuxième maximum, c'est
    ↪ à dire la deuxième valeur la plus grande de
    ↪ L. On suppose la liste contient deux éléments
    ↪ et que tous les éléments sont distincts'''
    if L[0]>L[1]:
        max1=L[0]
        max2=L[1]
    else:
        max2=L[0]
        max1=L[1]

    for i in range(2, len(L)):
        if L[i]>max1:
            max2=max1
            max1=L[i]
        elif L[i]>max2:
            max2=L[i]
    return max2

```

2.2.1 Adaptations à savoir faire

- Version sans range
- Deuxième minimum
- Position du deuxième minimum et du deuxième maximum

2.3 Occurrences

Sont regroupés dans cette partie tous les algorithmes permettant de compter le nombre d'occurrence, de connaître la présence d'un élément dans une liste.

2.3.1 Fonctions déjà présentes dans python

- méthode **count()** compte le nombre de fois ou un élément apparaît dans une liste.

```
L=[1,2,3,2,1]
print(L.count(5))
print(L.count(2))
```

Les valeurs affichées sont 0 et 2

- Opérateur **in** teste la présence ou l'absence d'un élément dans une liste, le résultat est un booléen.

```
L=[1,2,3,2,1]
print(2 in L)
print('a' in L)
```

Les valeurs affichées sont True et False

- (hors programme mais utile) la méthode **index** renvoie la première position d'un élément dans une liste, si l'élément est absent une exception *ValueError* est soulevée.

```
L=[1,2,3,2,1]
print(L.index(2))
```

et

```
L=[1,2,3,2,1]
print(L.index('a'))
```

renvoie le message d'erreur " `print(L.index('a'))` *ValueError : 'a' is not in list*"

Même si ces fonctions existent et que nous les utiliseront, il faut être capable d'écrire les fonctions suivantes de façon basique.

2.3.2 Première occurrence

Avec for

```
def occurrence(L,x):
    '''renvoie le plus petit indice i tel que L[i]=c,
    ↪ si l'élément est absent la valeur renvoyée
    ↪ est None'''
    for i in range(len(L)):
        if L[i]==x:
            return i #interruption de la fonction
    return None
```

Avec While et drapeau ▲

```
def occurrence(L,x):
    '''renvoie le plus petit indice i tel que L[i]=c,
    ↪ si l'élément est absent la valeur renvoyée
    ↪ est None'''
    drapeau=False
    i=0
    while drapeau==False and i<len(L):
        if L[i]== x:
            drapeau=True
            i=i+1+colorbox

    if i==len(L):#la liste a été parcourue en entier
        return None
    else :
        return i-1
```

Adaptations à savoir faire

- Utiliser un for sans range.
- changer la valeur renvoyée en cas d'absence en -1.

2.3.3 Présence d'un élément

For sans range

```
def presence(L,x):
    '''renvoie True si x est dans L, False sinon'''
    for e in L:
        if x==e:
            return True
    return False
```

While▲

```
def presence(L, x):
    '''renvoie True si x est dans L, False sinon'''
    i=0
    drapeau=False
    while i<len(L) and drapeau== False:
        if L[i]== x:
            drapeau=True
            i=i+1
    return drapeau
```

Adaptations à savoir faire

— version for avec range.

2.3.4 Nombre d'occurrence

Version sans range

```
def compte(L, x):
    '''compte le nombre d'occurrence de x dans L'''
    nb=0
    for e in L:
        if e==x:
            nb=nb+1
    return nb
```

Version avec range

```
def compte(L, x):
    '''compte le nombre d'occurrence de x dans L'''
    nb=0
    for i in range(len(L)):
        if L[i]==x:
            nb=nb+1
    return nb
```

Liste des occurrences▲

```
def liste_occurrence(L, x):
    '''renvoie la liste des indices i tels que
    ↪ L[i]=x'''
    R=[]
    for i in range(len(L)):
        if L[i]==x:
            R.append(i)
    return R
```

3 Algorithmes plus avancés sur les listes, boucles imbriquées

3.1 Recherche par dichotomie dans une liste triée

On suppose que la liste L est triée dans l'ordre croissant et on cherche la position de x dans L, on renvoie -1 si x n'est pas présent. On utilise un algorithme de dichotomie (voir le TP).

```
def dichotomie(L, x):
    '''retourne la position de x -1 si x est
    ↪ absent'''
    debut=0
    fin=len(L)-1#position du dernier élément
    while debut<fin:
        milieu=(debut+fin)//2 # quotient division
        ↪ euclidienne
        if L[milieu]==x:
            return milieu
        elif L[milieu]>x:
            fin=milieu
        else:
            debut=milieu+1
    if L[debut]==x:
        return debut
    else:
        return -1
```

Pour tester si la liste est bien triée

```
def esttriee(L):
    '''teste si la liste est triée dans l'ordre
    ↪ croissant'''
    for i in range(1, len(L)): #attention au départ
        if L[i-1] > L[i]:
            return False
    return True
```

3.1.1 Adaptation à savoir faire

- Renvoie juste True/False.
- Cas où la liste est triée dans l'ordre décroissant.
- Tester si la liste est bien triée dans l'ordre décroissant.

3.2 Trouver les deux éléments les plus éloignés

3.2.1 Valeur

On dispose d'une liste d'au moins deux éléments on cherche à calculer la valeur maximale de $|L[i]-L[j]|$ pour i, j deux indices

```
def eloignes(L):
    '''L a au moins deux éléments, calcule la valeur
    ↪ maximale de |L[i]-L[j]| pour i, j deux
    ↪ indices'''
    resultat = abs(L[0]-L[1])
    for i in range(len(L)):
        for j in range(i+1, len(L)):
            if abs(L[i]-L[j]) > resultat:
                resultat = abs(L[i]-L[j])
    return resultat
```

3.2.2 Positions

```
def eloignes(L):
    '''L a au moins deux éléments, calcule la
    ↪ position du couple (k,l) qui réalise la
    ↪ valeur maximale de |L[i]-L[j]| par
    ↪ construction k < l et sont minimaux'''
    k, l = 0, 1 #initialisation
    distance = abs(L[0]-L[1])
    for i in range(len(L)):
        for j in range(i+1, len(L)):
            if abs(L[i]-L[j]) > distance:
                distance = abs(L[i]-L[j])
                k, l = i, j
    return k, l
```

3.2.3 Adaptations ▲

- Renvoyer les valeurs des éléments les plus éloignés
- Renvoyer le couple des positions des éléments les plus proches qui ne sont pas égaux.

3.3 Recherche d'un motif ▲

On cherche à savoir si une sous liste (ou motif apparaît dans liste) Par exemple si $L=[1,2,3,4,5]$ les motifs $[1,2]$ et $[4,5]$ sont dans L mais pas $[5,4]$ et $[1,1,2]$

```

def motif(liste,motif):
    '''détermine si motif est une sous-liste de
    ↪ liste'''
    L=len(liste)# longueur de la liste
    l=len(motif)#longueur du motif
    drapeau=False
    i=0
    j=0
    while i+1 <=L and drapeau ==False:
        j=0
        drapeau2=True
        while j<l and drapeau2==True:
            drapeau2=(liste[i+j]==motif[j])
            j=j+1
        drapeau=drapeau2
        i=i+1
    return drapeau

```

3.3.1 Adaptations

- Remplacer les boucle while par des boucles for.
- Nombre d'occurrence du motif.
- Position du motif.
- Liste des positions de motif dans liste.

4 Autres algorithmes

4.1 Suite définies par récurrence

On veut calculer les termes successifs d'une suite définie par :

$$u_0 \in \mathbb{R}, \quad \forall n \in \mathbb{N} \quad u_{n+1} = u_n$$

On ne se préoccupe pas de l'ensemble de définition de f . Voici un exemple :

$$u_0 \in \mathbb{R}, \quad \forall n \in \mathbb{N} \quad u_{n+1} = u_n^2 - 1$$

```

def f(x):
    return x**2-1

def suite(u0,n):
    u=u0
    for i in range(n):
        u=f(u)
    return u

```

Adaptations à savoir faire

- Remplacer la boucle for par une boucle while, avec une condition portant sur la valeur de u_n .
- Sauvegarder toutes les valeurs calculées dans une liste.
- calculer le somme des termes de la suite (voir partie suivante).

Adaptations avancées ▲

- La fonction f devient un argument
- Tracer le graphe de la fonction et de la suite avec "l'escargot".

4.2 Calcul de somme de produit

4.2.1 Somme et produit simple

Structure

```

def somme(...):
    S=0
    for i in ...:
        S=S+....
    return S

```

```

def produit(...):
    P=1
    for i in ...:
        P=P*....
    return P

```

Il faut bien faire attention au décalage dans le range.

Exemples Pour calculer

$$\sum_{i=1}^n i^3$$

```
def somme(n):
    S=0
    for i in range(n+1):
        S=S+i**3
    return S
```

Pour calculer

$$\prod_{i=1}^n \frac{1}{i^2 + 1}$$

```
def produit(n):
    P=1
    for i in range(1,n+1):
        P=P*1/(i**2+1)
    return P
```

4.2.2 Utilisation de boucles imbriquées

Pour calculer des sommes doubles, on utilise

```
def somme(...):
    S=0
    for i in range(...):
        for j in range(...):
            S=S+
    return S
```

$$S_1 = \sum_{1 \leq i, j \leq n} ij \quad S_2 = \sum_{1 \leq i \leq j \leq n} i^j \quad S_3 = \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq p}} (i + j)$$

```
def S1(n):
    S=0
    for i in range(n+1):
        for j in range(n+1):
            S=S+i*j
    return S

def S2(n):
    S=0
    for i in range(n+1):
        for j in range(i+1,n+1):
            S=S+i**j
    return S

def S3(n,p):
    S=0
    for i in range(n+1):
        for j in range(p+1):
            S=S+i+j
    return S
```

4.2.3 Utilisation en statistiques

Moyenne des éléments d'une liste

```
def moyenne(L)
    '''moyenne des éléments de la liste, on suppose
    ↪ la liste non vide'''

    S=0
    for i in range(len(L)):
        S=S+L[i]
    return S/len(L)
```

Variance des éléments d'une liste

```

def variance(L):
    '''variance des éléments de la liste, on suppose
    ↪ la liste non vide, utilise la fonction
    ↪ précédente'''
    S=0
    for i in range(len(L)):
        S=S+L[i]**2
    return S/len(L)-moyenne(L)

```

Adaptation

- Sans utilisation de la formule de König-Huygens.
- Autre moments
- Calcul de l'écart-type

4.3 Algorithmes gloutons

4.3.1 Rendu de monnaie

On dispose d'une liste $L = [m_0, m_1, \dots, m_{n-1}]$ des valeurs des pièces et des billets d'un système monétaire. On suppose que ces valeurs sont triées dans l'ordre strictement croissant. Par exemple pour l'euro, si on ne se préoccupe pas des centimes, le système de monnaie est le suivant $L=[1,2,5,10,20,50,100,200,500]$. On cherche à donner une somme de monnaie avec un minimum de billets et de pièces.

```

def monnaie(L,m):
    '''Système représente la système de monnaie et
    ↪ montant le montant à décomposer et N la liste
    ↪ représentant le rendu'''
    N=[0]*len(L)
    for i in range(len(L)-1,-1,-1):
        quotient,reste=m//L[i],m%L[i]
        N[i]=quotient
        m=reste

    return(N)

```

4.3.2 Allocation de salle ▲

On cherche à gérer l'occupation d'une seule salle que plusieurs personnes veulent occuper, mais la salle ne peut accueillir qu'une personne à la fois. On cherche à créer un planning qui permette au plus grand nombre de personnes successives d'utiliser la salle.

- Chaque personne va être identifiée par une étiquette P1, P2...
- Chaque personne fournit un intervalle sur lequel elle souhaite occuper la salle, de la forme $I_n = [debut_n, fin_n]$. Cet intervalle ne peut pas être modifié.
- On calcule une liste donnant l'ordre des personnes occupant de la salle, tel que la salle n'est jamais occupée par deux personnes et qui maximise le nombre d'intervenants.

```

def prochain(souhait, debut):
    '''renvoie le numero de l'intervenant dont
    ↪ l'intervalle est
    compatible avec debut et dont l'heure de fin est
    ↪ minimale
    Si aucun intervenant n'est compatible la valeur
    ↪ renvoyée est -1
    '''
    intervenant=-1
    fin_min=float('inf')
    for i in range(len(souhait)):
        if souhait[i][0]>=debut and
            ↪ souhait[i][1]<fin_min:
            intervenant=i
            fin_min=souhait[i][1]
    return(intervenant)

```

```

def planning(souhait):
    resultat=[]
    temps=0
    intervenant=prochain(souhait,temps)
    while intervenant != -1:
        resultat.append(intervenant)
        temps=souhait[intervenant][1]
        intervenant=prochain(souhait,temps)
    return(resultat)

```