CI 2 : ALGORITHMIQUE & PROGRAMMATION

ALGORITHMES D'INFORMATIQUE

1 Recherches dans une liste

1.1 Recherche d'un nombre dans une liste

Pseudo Code

Algorithme : Recherche naïve d'un nombre dans une liste triée ou non

Données :
- n, int : un entier
- tab, liste : une liste d'entiers triés ou non triés

Résultat :
- un booléen : Vrai si le nombre est dans la liste, Faux sinon.

```
is_number_in_list(n,tab) :
    l ← longueur(tab)
    Pour i allant de 1 à l faire :
        Si tab[i] = n alors :
            Retourne Vrai
        Fin Si
    Fin Faire
    Retourne Faux
Fin fonction
```

python

```python
def is_number_in_list(nb,tab):
    """Renvoie True si le nombre nb est dans la liste
    de nombres tab
    Keyword arguments:
      * nb, int -- nombre entier
      * tab, list -- liste de nombres entiers
    """
    for i in range(len(tab)):
        if tab[i]==nb:
            return True
    return False
```

python

```python
def is_number_in_list(nb,tab):
    """Renvoie True si le nombre nb est dans la liste
    de nombres tab
    Keyword arguments:
      * nb, int -- nombre entier
      * tab, list -- liste de nombres entiers
    """
    i=0
    while i<len(tab) and tab[i]!=nb:
        i+=1
    return i<len(tab)
```

Remarque

Ces algorithmes sont modifiables aisément dans le cas où on souhaiterait connaître l'index du nombre recherché.

1.2 Recherche du maximum dans une liste de nombre

Pseudo Code

Algorithme : Recherche du maximum dans une liste de nombres

Données :
– tab, liste : une liste de nombres

Résultat :
– maxi, réel : maximum de la liste

```
what_is_max(tab) :
    n ← longueur(tab)
    i ← 2
    maxi ← tab[1]
    Tant que i < n faire :
        Si tab[i]>maxi alors :
            maxi ← tab[i]
        Fin si
        i ← i+1
    Fin tant que
    Retourner maxi
Fin fonction
```

python

```python
def what_is_max(tab):
    """
    Renvoie le plus grand nombre d'une liste
    Keyword arguments:
    tab -- liste de nombres
    """
    i=1
    maxi=tab[0]
    while i<len(tab):
        if tab[i]>maxi:
            maxi=tab[i]
        i+=1
    return maxi
```

1.3 Recherche par dichotomie dans un tableau trié

Pseudo Code

Algorithme : Recherche par dichotomie d'un nombre dans une liste triée ou non

Données :
– nb, int : un entier
– tab, liste : une liste d'entiers triés

Résultat :
– m, int : l'index du nombre recherché
– None : cas où nb n'est pas dans tab

```
is_number_in_list_dicho(nb,tab) :
    g ← 0
    d ← longueur(tab)
    Tant que g < d alors :
        m ← (g+d) div 2 alors :
            Si tab[m]=nb alors :
                Retourne m
            Sinon si tab[m]<nb alors :
                g ← m+1
            Sinon, alors :
                d ← m-1
        Fin Si
    Fin Tant que
    Retourne None
Fin fonction
```

python

```python
def is_number_in_list_dicho(nb,tab):
    """
    Recherche d'un nombre par dichotomie dans un
    tableau trié.
    Renvoie l'index si le nombre nb est dans la liste
    de nombres tab.
    Renvoie None sinon.
    Keyword arguments:
    nb, int -- nombre entier
    tab, list -- liste de nombres entiers triés
    """
    g, d = 0, len(tab)-1
    while g <= d:
        m = (g + d) // 2
        if tab[m] == nb:
            return m
        if tab[m] < nb:
            g = m+1
        else:
            d = m-1
    return None
```

2 Gestion d'une liste de nombres

2.1 Calcul de la moyenne

Pseudo Code

**Algorithme : ** Calcul de la moyenne arithmétique des nombres d'une liste

Données :
- tab, liste : une liste de nombres

Résultat :
- res, réel : moyenne des nombres

```
calcul_moyenne(tab) :
    n ← longueur(tab)
    res ← 0
    Pour i allant de1 à n faire :
        res ← res+tab[i]
    Fin faire
    Retourner res/n
Fin fonction
```

python

```python
def calcul_moyenne(tab):
    """
    Renvoie la moyenne des valeurs d'une liste de
    nombres.
    Keyword arguments:
    tab -- liste de nombres
    """
    res = 0
    for i in range(len(tab)):
        res = res+tab[i]
    return res/(len(tab))
```

2.2 Calcul de la variance

Soit une série statistique prenant les n valeurs x_1, x_2, ..., x_n. Soit m la moyenne de ces valeurs. La variance est définie par :

$$v = \frac{1}{n}\sum_{i=1}^{n}(x_i - m)^2$$

Pseudo Code

**Algorithme : ** Calcul de la variance des nombres d'une liste

Données :
- tab, liste : une liste de nombres
- m, réel : moyenne de la liste

Résultat :
- res, réel : variance

```
calcul_variance(tab) :
    n ← longueur(tab)
    res ← 0
    Pour i allant de 1 à n faire :
        res ← res+(tab[i]-m)**2
    Fin faire
    Retourner res/n
Fin fonction
```

python

```python
def calcul_variance(tab,m):
    """
    Renvoie la variance des valeurs d'un tableau.
    Keyword arguments:
    tab -- liste de nombres
    m -- moyenne des valeurs
    """
    res = 0
    for i in range(len(tab)):
        res = res+(tab[i]-m)**2
    return res/(len(tab))
```

2.3 Calcul de la médiane

Pseudo Code

Algorithme : Recherche de la valeur médiane d'une liste de nombres triés

Données :
– tab, liste : liste de nombres triés

Résultat :
– flt : valeur de la médiane

```
mediane(tab) :
    n ← Longueur(tab)
    Si n modulo 2 = 0 Alors :
        i ← n/2
        Retourner (tab[i] +tab[i+1])/2
    Sinon :
        i ← n div 2+1
        Retourner (tab[i])
Fin fonction
```

python

```python
def calcul_variance(tab):
    """
    Calcule la variance des éléments d'un tableau trié.
    Keyword arguments:
    tab -- liste de nombres
    """
    if len(tab)%2 == 0 :
        i=len(tab)//2
        return (tab[i-1]+tab[i])/2
    else :
        i = len(tab)//2
    return tab[i]
```

3 Chaînes de caractères

3.1 Recherche d'un mot dans une chaîne de caractères

Pseudo Code

python

```python
def index_of_word_in_text(mot, texte):
    """ Recherche si le mot est dans le texte.
    Renvoie l'index si le mot est présent, None sinon.

    Keyword arguments:
    mot -- mot recherché
    texte -- texte

    """
    for i in range(1 + len(texte) - len(mot)):
        j = 0
        while j < len(mot) and mot[j] == texte[i + j]:
            j += 1
        if j == len(mot):
            return i
    return None
```

4 Calcul numérique

4.1 Recherche du zéro d'une fonction continue monotone par la méthode de dichotomie

Pseudo Code

Début Fonction
  Données : f, a, b, ε
  $g \leftarrow a$
  $d \leftarrow b$
  $f_g \leftarrow f(g)$
  $f_d \leftarrow f(d)$
  tant que $(d-g) > 2\varepsilon$ **faire**
    $m \leftarrow (g+d)/2$
    $f_m \leftarrow f(m)$
    si $f_g \cdot f_m \leq 0$ **alors**
      $d \leftarrow m$
      $f_d \leftarrow f_m$
    sinon
      $g \leftarrow m$
      $f_d \leftarrow f_m$
    fin
  fin
  retourner $(g+d)/2$
Fin

python

```python
def rechercheDichotomique(f,a,b,eps):
    g = a
    d = b
    f_g = f(g)
    f_d = f(d)
    while (d-g) > 2*eps:
        m = (g+d)/2
        f_m =f(m)
        if f_g * f_m <= 0 :
            d = m
            f_d = f_m
        else :
            g  = m
            f_d = f_m
    return (g+d)/2
```

4.2 Recherche du zéro d'une fonction continue monotone par la méthode de Newton

Pseudo Code

Début Fonction
  Données : f, f', a, ε
  $g \leftarrow a$
  $c \leftarrow g - \dfrac{f(g)}{f'(g)}$
  tant que $|c-g| > \varepsilon$ **faire**
    $g \leftarrow c$
    $c \leftarrow c - \dfrac{f(c)}{f'(c)}$
  fin
  retourner c
Fin

python

4.3 Méthode des rectangles pour le calcul approché d'une intégrale sur un segment

4.3.1 Méthode des rectangles à gauche

Pseudo Code

Algorithme : Calcul d'intégrale par la méthode des rectangles à gauche
Données :
- f, fonction : fonction définie sur $[a,b]$
- a, réel : borne inférieure de l'intervalle de définition
- b, réel : borne supérieure de l'intervalle de définition, b≥a
- nb, entiers : nombre d'échantillons pour calculer l'intégrale

Résultat :
- res, réel : valeur approchée de $\int_a^b f(t)\mathrm{d}t$

```
integrale_rectangles_gauche(f,a,b,nb) :
    res ← 0
    pas ← (b-a)/nb
    x ← a
    Tant que x<b-pas : Faire
        res← res + pas * f(x)
        x ← x+pas
    Fin Tant que
    Retourner res
```

python

```python
def integrale_rectangles_gauche(f,a,b,nb):
    """
    Calcul de la valeur approchée de l'intégrale de f(x) entre a et b par la
    méthode des rectangles à gauche.
    Keywords arguments :
    f -- fonction à valeur dans IR
    a -- flt, borne inférieure de l' intervalle d'intégration
    b -- flt, borne supérieure de l' intervalle d'intégration
    nb -- int, nombre d'échantillons pour le calcul
    """
    res = 0
    pas = (b-a)/nb
    x = a
    while x<b-pas:
        res = res + pas *f(x)
        x = x + pas
    return res
```

4.3.2 Méthode des rectangles à droite

Pseudo Code

Algorithme : Calcul d'intégrale par la méthode des rectangles à droite
Données :
- f, fonction : fonction définie sur $[a,b]$
- a, réel : borne inférieure de l'intervalle de définition
- b, réel : borne supérieure de l'intervalle de définition, b≥a
- nb, entiers : nombre d'échantillons pour calculer l'intégrale

Résultat :
- res, réel : valeur approchée de $\int_a^b f(t)\mathrm{d}t$

```
integrale_rectangles_droite(f,a,b,nb) :
    res ← 0
    pas ← (b-a)/nb
    x ← a+pas
    Tant que x<b-pas : Faire
        res← res + pas * f(x)
        x ← x+pas
```

Pseudo Code

python

```
def integrale_rectangles_droite(f,a,b,nb):
    """
    Calcul de la valeur approchée de l'intégrale de f(x) entre a et b par la
    méthode des rectangles à droite.
    Keywords arguments :
    f -- fonction à valeur dans IR
    a -- flt, borne inférieure de l'intervalle d'intégration
    b -- flt, borne supérieure de l'intervalle d'intégration
    nb -- int, nombre d'échantillons pour le calcul
    """
    res = 0
    pas = (b-a)/nb
    x = a+pas
    while x<b-pas:
        res = res + pas *f(x)
        x = x + pas
    return res
```

4.3.3 Méthode des rectangles – Point milieu

Pseudo Code

Algorithme : Calcul d'intégrale par la méthode des rectangles – point milieu

Données :
- f, fonction : fonction définie sur $[a,b]$
- a, réel : borne inférieure de l'intervalle de définition
- b, réel : borne supérieure de l'intervalle de définition, b≥a
- nb, entiers : nombre d'échantillons pour calculer l'intégrale

Résultat :
- res, réel : valeur approchée de $\int_a^b f(t)\mathrm{d}t$

```
integrale_rectangles_droite(f,a,b,nb) :
    res ← 0
    pas ← (b-a)/nb
    x ← a
    Tant que x<b-pas : Faire
        res← res + pas *( f(x)+f(x+pas))/2
        x ← x+pas
    Fin Tant que
    Retourner res
```

```python
def integrale_rectangles_milieu(f,a,b,nb):
    """
```

```python
Calcul de la valeur approchée de l'intégrale de f(x) entre a et b par la
méthode du point milieu.
Keywords arguments :
f -- fonction à valeur dans IR
a -- flt, borne inférieure de l' intervalle d'intégration
b -- flt, borne supérieure de l' intervalle d'intégration
nb -- int, nombre d'échantillons pour le calcul
"""
res = 0
pas = (b-a)/nb
x = a+pas
while x<b-pas:
    res = res + pas *(f(x)+f(x+pas))/2
    x = x + pas
return res
```

4.4 Méthode des trapèzes pour le calcul approché d'une intégrale sur un segment

cf. méthodes des rectangles par la méthode du point milieu.

4.5 Méthode d'Euler pour la résolution d'une équation différentielle

4.5.1 Méthode d'Euler explicite

Résolution de l'équation différentielle :

$$y(t)+\tau\frac{dy(t)}{dt}=y_f$$

Pseudo Code

Algorithme : Méthode d'Euler explicite

Données :
- tau, réel : constante de temps
- y_0, réel : valeur initiale de y
- y_f, réel : valeur finale y
- t_f, réel : temps de la simulation numérique
- nb, entier : nombre d'échantillons pour calculer les valeurs de y

Résultat :
- res, liste : liste des couples (t,y(t)).

```
euler_explicite(tau,y_0,y_f,t_f,nb) :
    Initialiser res
    t ← 0
    y ← y_0
    pas ← t_f/nb
    Tant que t<t_f Faire :
        Ajouter (t,y) à res
        y ← y + pas *(y_f-y)/tau
        t ← t + pas
    Fin Tant que
    Retourner res
```

```python
def euler_explicite (tau,y0,yf,tf,nb):
    """
    Résolution d'une équation différentielle d'ordre 1 en utilisant la méthode
    d'Euler explicite.
    Keywords arguments :
    tau -- flt, constante de temps de l'équation différentielle
    y0 -- flt, valeur initiale de y(t)
    yf -- flt valeur finale de y(t)
    tf -- flt temps de fin de la simulation
    nb -- int, nombre d'échantillons pour la simulation
    """
    t = 0
    y = y0
    pas = tf / nb
    res = []
    while t < tf:
        res.append((t,y))
        y = y + pas*(yf-y)/tau
        t = t + pas
    return res
```

4.6 Algorithme de Gauss – Jordan

```python
def recherche_pivot(A,i):
    n = len(A) # le nombre de lignes
    j = i # la ligne du maximum provisoire
    for k in range(i+1, n):
        if abs(A[k][i]) > abs(A[j][i]):
            j = k # un nouveau maximum provisoire
    return j

def echange_lignes(A,i,j):
    # Li <-->Lj
    A[i][:], A[j][:]=A[j][:], A[i][:]

def transvection_ligne (A, i, j, mu):
    # L_i <- L_i + mu.L_j """
    nc = len(A[0]) # le nombre de colonnes
    for k in range(nc):
        A[i][k] = A[i][k] + mu * A[j][k]

def resolution (AA, BB):
    """Résolution de AA.X=BB; AA doit etre inversible"""
    A, B = AA.copy(), BB.copy()
    n = len(A)
    assert len(A[0]) == n
    # Mise sous forme triangulaire
    for i in range(n):
        j = recherche_pivot(A, i)
        if j > i:
            echange_lignes(A, i, j)
            echange_lignes(B, i, j)
        for k in range(i+1, n):
```

```python
            x = A[k][i] / float (A[i][i])
            transvection_ligne (A, k, i, -x)
            transvection_ligne (B, k, i, -x)
    # Phase de remontée
    X = [0.] * n
    for i in range(n-1, -1, -1):
        X[i] = (B[i][0]-sum(A[i][j]*X[j] for j in range(i+1,n))) / A[i][i]
    return X
```

5 Algorithmes de tris

5.1 Tri par sélection

```python
#Tri par sélection
def tri_selection (tab):
    for i in range(0,len(tab)):
        indice = i
        for j in range(i+1,len(tab)):
            if tab[j]<tab[indice]:
                indice = j
        tab[i],tab[indice]=tab[indice],tab[i]
    return tab
```

5.2 Tri par insertion

5.2.1 Méthode 1

Pseudo Code

Algorithme : Tri par insertion – Méthode 1
Données :
– tab, liste : une liste de nombres
Résultat :
– tab, liste : la liste de nombres triés

```
tri_insertion(tab) :
    n ← longueur(tab)
    Pour i de 2 à n :
        x ← tab[i]
        j ← 1
        Tant que j ≤ i-1 et tab[j]<x :
            j ← j+1
        Fin Tant que
        Pour k de i-1 à j-1 par pas de -1 faire :
            tab[k+1] ← tab[k]
        Fin Pour
        tab[j] ← x
    Fin Pour
```

```python
def tri_insertion_01(tab):
    """
    Trie une liste de nombre en utilisant la méthode
    du tri par insertion.
    En Python, le passage se faisant par référence, il
    n'est pas indispensable de retourner le tableau.
    Keyword arguments:
    tab -- liste de nombres
    """
    for i in range (1,len(tab)):
        x=tab[i]
        j=0
        while j<=i-1 and tab[j]<x:
            j = j+1
        for k in range(i-1,j-1,-1):
            tab[k+1]=tab[k]
        tab[j]=x
```

Estimation de la complexité

- Meilleur des cas, le tableau est trié à l'envers, la complexité est linéaire : $\mathcal{O}(n)$.
- Pire des cas, le tableau est trié, la complexité est quadratique : $\mathcal{O}(n^2)$.

5.2.2 Méthode 2

Pseudo Code

```
Algorithme : Tri par insertion – Méthode 2
Données :
 – tab, liste : une liste de nombres
Résultat :
 – tab, liste : la liste de nombres triés

tri_insertion(tab) :
    n ← longueur(tab)
    Pour i de 2 à n :
        x ← tab[i]
        j ← i
        Tant que j >1 et tab[j-1]>x :
            tab[j] ← tab[j-1]
            j ← j-1
        Fin Tant que
        tab[j] ← x
    Fin Pour
```

python

```python
def tri_insertion_02(tab):
    """
    Trie une liste de nombre en utilisant la méthode
    du tri par insertion .
    En Python, le passage se faisant par référence,
    il n'est pas indispensable de retourner le tableau.
    Keyword arguments:
    tab –– liste de nombres
    """
    for i in range (1,len(tab)):
        x=tab[i]
        j=i
        while j>0 and tab[j−1]>x:
            tab[j]=tab[j−1]
            j = j−1
        tab[j]=x
```

Estimation de la complexité

- Meilleur des cas, le tableau est trié, la complexité est linéaire : $\mathcal{O}(n)$.
- Pire des cas, le tableau est trié à l'envers, la complexité est quadratique : $\mathcal{O}(n^2)$.

5.3 Tri shell

python

```python
def shellSort (array ):
    "Shell sort using Shell's (original) gap sequence: n/2, n/4, ..., 1."
    "http://en.wikibooks.org/wiki/Algorithm_Implementation/Sorting/Shell_sort#Python"
    gap = len(array) // 2
    # loop over the gaps
    while gap > 0:
        # do the insertion sort
        for i in range(gap, len(array)):
            val = array[i]
            j = i
            while j >= gap and array[j − gap] > val:
                array[j] = array[j − gap]
                j −= gap
            array[j] = val
        gap //= 2
```

5.4 Tri rapide «Quicksort»

Pseudo Code

```
Algorithme : Tri Quicksort – Segmentation
Données :
– tab, liste : une liste de nombres
– i,j, entiers : indices de début et de fin de la segmentation à effectue
Résultats :
– tab, liste : la liste de nombre segmenté avec le pivot à sa place définitive
– k entier : l'indice de la place du pivot

Segmente(tab,i,j) :
    g ← i+1
    d ← j
    p ← tab[i]
    Tant que g ≤ d Faire
        Tant que d≥ 0 et tab[d]>p Faire
            d ← d-1
        Fin Tant que
        Tant que g≤ j et tab[g]≤p Faire
            g ← g+1
        Fin Tant que
        Si g<d alors
            Échange( tab,g,d )
            d ← d-1
            g ← g+1
        Fin Si
    Fin Tant que
    k← d
    Échange( tab,i,d )
    Retourner k
```

```
Algorithme : Tri Quicksort – Tri rapide
Données :
– tab, liste : une liste de nombres
– i,j, entiers : indices de début et de fin de la segmentation à effectue
Résultats :
– tab, liste : liste triée entre les indices i et j

tri_quicksort(tab,i,j) :
    Si g<d alors
        k← segmente(tab,i,j )
        tri_quicksort(tab,i,k-1 )
        tri_quicksort(tab,k+1,j )
    Fin Si
```

5.5 Tri fusion

6 Algorithmes classiques

6.1 Division euclidienne

Pseudo Code

Data : $a, b \in \mathbb{N}^*$
reste ← a
quotient ← 0
tant que *reste* ≥ *b* **faire**
  reste ← reste − b
  quotient ← quotient + 1
fin
Retourner quotient,reste

6.2 Algorithme d'Euclide

Cet algorithme permet de calculer le PGCD de deux nombres entiers. Il se base sur le fait que si a et b sont deux entiers naturels non nuls, $pgcd(a,b) = pgcd(b, a \bmod b)$.

Pseudo Code

Fonction PGCD : algorithme d'Euclide
Données : a et b : deux entiers naturels non nuls tels que a > b
Résultat : le PGCD de a et b

Euclide_PGCD(a,b)
  Répéter
    r ← a mod b
    a ← b
    b ← r
  Jusqu'à r == 0
  Retourner a

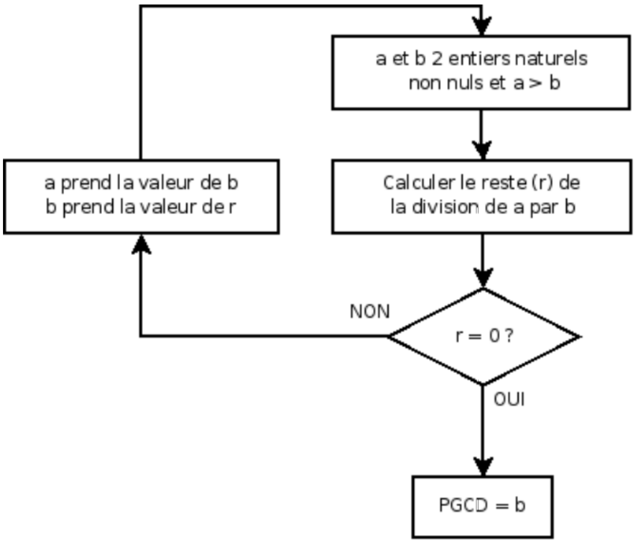
python

Codage en Pythonde l'algorithme d'Euclide :

```
def Euclide_PGCD(a,b):  # on définit le nom de la
                        # fonction et ses variables
                        # d'entrées/d'appel
    r=a%b               # on calcule le reste dans
                        # la division de a par b

    while r!=0:         # tant que r est non nul :
        a=b             # b devient le nouveau a
        b=r             # r devient le nouveau b
        r=a%b           # on recalcule le reste

    return(b)           # une fois la boucle terminée,
                        # on retourne le dernier b
print(Euclide_PGCDpgcd(1525,755))
                        # on affiche le résultat
                        # retourné par la fonction
```



6.3 Recherche des nombres premiers – Crible d'Ératosthène

6.4 Calcul de puissance

6.4.1 Algorithme naïf

6.4.2 Exponentiation rapide

7 Calcul d'un polynôme

7.1 Algorithme naïf

7.2 Méthode de Horner

Références

[1] Patrick Beynet, Cours d'informatique de CPGE, Lycée Rouvière de Toulon, UPSTI.

[2] Adrien Petri et Laurent Deschamps, Cours d'informatique de CPGE, Lycée Rouvière de Toulon, UPSTI.

[3] Damien Iceta, Cours d'informatique de CPGE, Lycée Gustave Eiffel de Cachan, UPSTI.

[4] Benjamin WACK, Sylvain CONCHON, Judicaël COURANT, Marc DE FALCO, Gilles DOWEK, Jean-Christophe FILLIÂTRE, Stéphane GONNORD, Informatique pour tous en classes préparatoires aux grandes écoles, Éditions Eyrolles.