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

python

```python
def what_is_max(tab):
    """
    Renvoie le plus grand nombre d'une liste
    Keyword arguments:
    tab -- liste de nombres
    """
    i=1
```

```python
    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
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
```

```python
        else :
            d = m-1
    return None
```

2 Gestion d'une liste de nombres

2.1 Calcul de la moyenne

```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 :

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

```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

Estimation de la complexité

3 Chaînes de caractères

3.1 Recherche d'un mot dans une chaîne de caractères

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
```

Estimation de la complexité

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
```

Précision du calcul

Rapidité

Comparaison à zéro

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

Précision du calcul

Rapidité

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 la dérivée

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 la dérivée

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
```

python

4.3.3 Méthode des rectangles – Point milieu

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 la dérivée

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
```

Pseudo Code

```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
```

python

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

$$f(t)+\tau\frac{df(t)}{dt}=\omega_f$$

Algorithme : Méthode d'Euler explicite

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 la dérivée

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
```

Pseudo Code

5 Algorithmes de tris

5.1 Tri par sélection

python

```
#Tri par sé lection
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]
```

python

```python
    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
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»

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

Data : $a, b \in \mathbb{N}^*$
$x \leftarrow a$
$y \leftarrow b$
tant que $y \neq 0$ **faire**
    $r \leftarrow$ reste de la division euclidienne de x par y
    $x \leftarrow y$
    $y \leftarrow r$
fin
Afficher x.

python

Codage en Pythonde l'algorithme d'Euclide :

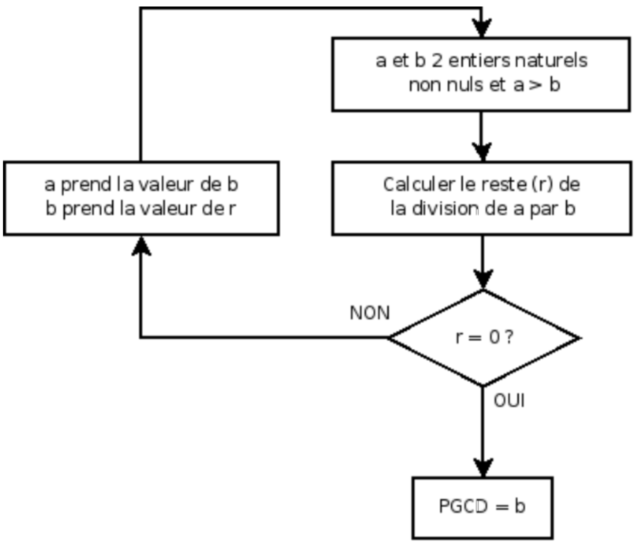
```
def Euclide_PGCD(a,b):  # on définit le nom de la
                        # fonction et ses variables
                        #d'entrées/d'appel
    r=a%b               # on calcule le reste dans
                        # la division de a par b

    while r!=0:         # tant que ce reste 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(pgcd(1525,755))   # on affiche le résultat
                        # retourné par la fonction
```

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



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.