Algorithmes et Pensée Computationnelle

Semaine 6

Algorithmes de recherche

1. Recherche séquentielle
2. Recherche binaire
3. Arbres de recherche binaire

Le but de cette séance est de se familiariser avec les algorithmes de recherches.

1 Recherche séquentielle (ou recherche linéaire)

1.1 Définition

Une recherche **linéaire** (ou **séquentielle**) est une méthode permettant de trouver un élément dans une liste, un tableau ou un dictionnaire. Elle vérifie un à un chaque élément de gauche à droite jusqu'à ce qu'une correspondance soit trouvée ou que toute la liste ait été recherchée.

Si l'élément recherché est trouvé, l'algorithme **renvoie l'index**, c'est-à-dire la position, de l'élément dans la liste donnée.

Sa complexité dans le pire des cas est de **O(n)**, la longueur de la liste, et dans le meilleur des cas de **O(1)**, lorsque l'élément se trouve en première position. Dans la pratique, l'algorithme de recherche séquentielle n'est pas souvent utilisé en vue de sa complexité élevée et des alternatives de recherche plus efficaces comme la recherche binaire.

1.2 Exercices

Question 1: (🕓 *5 minutes*) language : **Python**

A partir des éléments ci-dessous, écrivez une fonction qui cherche **x** dans la liste.
La fonction doit retourner l'index de l'élément correspondant de la liste si **x** est dans la liste et "-1" si **x** n'est pas dans la liste (avec **x = 100**)

```
def seq_search(list, x):
    #complètez ici

liste = [3,55,6,8,3,5,56,33,6,5,3,2,99,53,532,75,21,963,100,445,56,56,24]
x = 100

seq_search(liste, x)

%%time #Permet d'afficher le temps d'exécution de l'algorithme
print(seq_search(liste, x))
```

Conseil

Définissez votre fonction de recherche linéaire en utilisant une boucle **for** ou une boucle **while**.

Attention : la fonction doit retourné l'index de la valeur et non pas la valeur. Pour cela, veillez à utiliser **range(len(list))** avec la boucle **for** et une incrémentation **"i = i+1"** avec la boucle **while**.

La fonction **print()** vous permet d'afficher l'index lorsqu'il a été trouvé et un autre message le cas échéant.

Solution question 1

```
#Définition de la fonction
def seq_search(list, x):
    for i in range(len(list)): #i représente l'index
        if list[i] == x:
            print("X est présent dans la liste à l'index :", i)
            return i

        else: #Utilisez "else" s'il n'y a pas de "break"
            print("X n'est pas présent dans la liste")
            return -1

#Déclaration de la liste et de la variable x
liste = [3,55,6,8,3,5,56,33,6,5,3,2,99,53,532,75,21,963,100,445,56,45,12,56,24]
x = 100

#Exécution de l'algorithme
seq_search(liste, x)

%%time #Permet d'afficher le temps de calcul
print(seq_search(liste, x)
```

Question 2: (*10 minutes*) language : **Python**

Considérez une liste d'entiers non triée **L** ainsi qu'un entier **e**. Écrivez un programme qui retourne l'élément de la liste **L** dont la valeur est la plus proche de **e** en utilisant une recherche séquentielle.

Exemple :
L = [16, 2, 25, 8, 12, 31, 2, 56, 58, 63]
e = 50
Résultat attendu : 56

```
#Definition de la fonction ayant pour argument une liste et un nombre
def find_closest_seq(list,n):
    diff = None #Initialisation de la variable pour la différence
    resultat = None #Initialisation de la variable pour le résultat

    #Complètez ici

#Déclaration de la liste et de la variable e
L = [16, 2, 25, 8, 12, 31, 2, 56, 58, 63]
e = 50

#Exécution de la fonction
find_closest_seq(L,e)
```

```
%%time
print(find_closest_seq(L,e))
```

Conseil

Complétez la fonction **find_closest_seq** et exécutez le code.

Veillez à utiliser les valeurs absolues pour comparer les différences, la plus petite pouvant être positive ou négative. La fonction **abs()** retourne la valeur absolue. Exemple : **abs(3−10) retourne 7.**

Étant donné que la liste est **non triée**, l'algorithme doit obligatoirement la parcourir intégralement.

L'algorithme doit calculer la différence entre **e** et chaque élément de la liste **L** en gardant toujours la plus petite différence trouvée. À la fin, il retourne l'élément de la liste correspondant à la plus petite différence.

Solution question 2

```
#Definition de la fonction ayant pour argument une liste et un nombre
def find_closest_seq(list,n):
    diff = None #Initialisation de la variable pour la différence
    resultat = None #Initialisation de la variable pour le résultat

    #Solution
    for i in list:
        if not diff or abs(i-n) < diff: #old diff
            diff = abs(i-n) #new diff #
            resultat = i

    return resultat, diff

#Déclaration de la liste et de la variable e
L = [16, 2, 25, 8, 12, 31, 2, 56, 58, 63]
e = 50

#Exécution de la fonction
find_closest_seq(L,e)

%%time
print(find_closest_seq(L,e))
```

Question 3: (*5 minutes*) language : **Python**

Considérez une **liste d'entiers triés L** ainsi qu'un entier **e**. Écrivez un programme qui retourne l'index de l'élément **e** de la liste **L** en utilisant une recherche séquentielle. Si **e** n'est pas dans **L**, retournez -1.

Exemple :
L = [1231321,3213125,3284016,4729273,5492710]
e = 3284016
Résultat attendu : 2

```
def linear_search(L,e):
    for i in L: #Ici, i correspond à la valeur et non l'index.
```

```
    #complètez ici

L = [1231321,3213125,3284016,4729273,5492710]
e = 3284016
linear_search(L,e)

%%time
print(linear_search(L,e))
```

Conseil

Une liste triée permet une recherche plus efficace à l'aide d'un algorithme plus simple.

Veillez à retourner l'index de la valeur dans la liste. La fonction **index()** retourne l'index d'un élément au sein d'une liste.
Exemple et syntaxe : **lst.index(i)** va indiqué la position de l'élément **i** dans la liste **lst**.

Solution question 3

```
def linear_search(L,e):
    for i in L:
        #Solution
        if i == e:
            return L.index(i)
    return -1

L = [1231321,3213125,3284016,4729273,5492710]
e = 3284016
linear_search(L,e)

%%time
print(linear_search(L,e))
```

2 Recherche binaire

2.1 Définition

Le but de la recherche binaire est de trouver l'élément **x** plus rapidement. Pour cela, il est nécessaire d'utiliser **une liste d'éléments triée**.

La complexité de l'algorithme de recherche binaire est **O(log n)**. Cependant, il ne faut pas oublier le coût lié à l'obtention d'une liste triée à partir d'une liste non triée.

L'algorithme de recherche binaire divise l'intervalle de recherche par deux à chaque coup jusqu'à ce qu'il trouve l'élément **x** ou que l'intervalle soit vide.

Ainsi, si **x** est plus petit que l'élément du milieu, l'algorithme va choisir la moitié de gauche comme intervalle de recherche et ainsi de suite. Si **x** est plus grand que l'élément du milieu la recherche va se faire dans la moitié droite de l'intervalle.

La recherche binaire regarde les comparaisons d'ordre, alors que la recherche séquentielle regarde les comparaisons d'égalité pour trouver x.

2.2 La récursivité

Une fonction récursive est une fonction qui s'appelle elle-même pendant son exécution. Vous trouverez ci-dessous un exemple de fonction récursive utilisée pour le calcul factoriel.

```
def factoriel(n):
    if n == 1:
        return n
    else:
        return n * factoriel(n - 1)

factorielle(4)
#La fonction calcul 4 factoriel et retourne le résultat
#Rappel : 4! = 4 x 3 x 2 x 1 = 24
```

Les fonctions récursives sont courantes en informatique car elles permettent aux programmeurs d'écrire des programmes efficaces en utilisant une quantité minimale de code. L'inconvénient est qu'elles peuvent provoquer des boucles infinies et d'autres résultats inattendus si elles ne sont pas écrites correctement. Si la fonction n'inclut pas les cas appropriés pour arrêter l'exécution, la récursivité se répétera à l'infini, provoquant le plantage du programme ou, pire encore, l'arrêt de tout le système informatique.

2.3 Exercices

Question 4: (🕓 *20 minutes*) language : **Python**

A partir de la liste d'éléments **triée** ci-dessous, écrivez premièrement **une fonction récursive** puis **une fonction itérative** qui cherche **x** dans la liste. La fonction doit retourner **l'index** de l'élément correspondant de la liste si **x** est dans la liste et **−1** dans le cas contraire.
Ici, **x = 5**.

```
#Version récursive

def rec_bin_search(list,s,r,x):
    #complètez ici

liste=[1,3,4,5,7,8,9,15]
s = 0
```

```
r = len(liste)
x = 5
rec_bin_search(liste,s, r, x)

%%time
print(rec_bin_search(liste,s, r, x))



#Version itérative

def it_bin_search(list,s,r,x):
  #complètez ici

liste = [1,3,4,5,7,8,9,15]
s = 0
r = len(liste)-1
x = 7
it_bin_search(liste,s, r, x)

%%time
print(it_bin_search(liste,s, r, x))
```

Conseil

Complétez la fonction récursive **rec_bin_search** et la fonction itérative **it_bin_search**.

Détails sur les arguments de la fonction :
list : La liste dans laquelle nous effectuons la recherche
s : Le premier élément de la liste (index [0])
r : Le dernier élément de la liste (index [longueur de la liste -1])
x : La valeur recherchée.

Ainsi, à chaque itération, vos fonction vont modifier les valeurs de base données en argument pour resserrer l'intervalle jusqu'à trouver la valeur recherchée.
Pour définir le milieu d'un intervalle qui contient un nombre pair ou impair d'éléments, utilisez la fonction **int()**. Exemple :
liste1 = [1,2,3,4,5] s = 0 r = len(liste1) Calcul du milieu de l'intervalle :
- **(s+r)/2) = (0+5)/2 = 2.5** #Pas de correspondance
- **int((s+r)/2) = int((0+5)/2) = int(2.5) = 2** #Arrondi vers le bas

Pour la version itérative, il est conseillé d'utiliser une boucle **while**.

Solution question 4

```
#Version récursive
def rec_bin_search(list,s,r,x): #s as first index and r as last index of list
    #SOLUTION
    mid = int((s+r)/2) #int arrondi vers le bas
    print(mid)
    if list[mid] < x:
        return rec_bin_search(list, mid, r, x)
    elif list[mid] > x:
        return rec_bin_search(list, s, mid, x)
    elif list[mid] == x:
        print("X in list at index: ", mid)
        return mid
```

```
    else:
        print ("X not in list")

liste=[1,3,4,5,7,8,9,15]
s = 0
r = len(liste) #8 --> first mid = 4 -->"7" in liste --> 7>5 --> new mid = (0+4)/2 = 2,
x = 5
rec_bin_search(liste,s, r, x)


%%time
print(rec_bin_search(liste,s, r, x))


################################################


#Version itérative
def it_bin_search(list,s,r,x):
    #SOLUTION
    while s <= r:
        mid = int(s + (r-s)/2)
        print(mid)
        if list[mid] == x:
            print("X presents in list at index: ", mid)
            return mid
        elif list[mid] < x:
            s = mid+1
        else:
            r= mid-1
    print("X not present in list")
    return -1
liste = [1,3,4,5,7,8,9,15]
s = 0
r = len(liste)
x = 15
it_bin_search(liste,s, r, x)

%%time
print(it_bin_search(liste,s, r, x))
```

Question 5: (🕓 *15 minutes*) language : **Python**

Considérez une liste d'entiers **triés** **L** ainsi qu'un entier **e**. Écrivez un programme qui retourne l'élément de la liste **L** le plus proche de **e** en utilisant une recherche binaire.

On vous donne une liste d'entiers **triés** **L** ainsi qu'un entier **e**. Écrivez un programme retournant la valeur dans **L** la plus proche de **e** en utilisant une recherche binaire (binary search).

Résultat attendu : 56

```
def find_closest_bin(liste,n):
    #complètez ici


L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63]
e = 41
```

```
find_closest_bin(L,e)

%%time
print(find_closest_bin(L,e))
```

Conseil

Veillez à définir des variables **min** et **max** délimitant l'intervalle de recherche et une variable booléenne **found** initialisée **false** et qui devient **true** lorsque l'algorithme a trouvé la valeur la plus proche de **e**.

Solution question 5

```
def find_closest_bin(liste,n):
    #SOLUTION

    min = 0
    max = len(liste)
    found = False
    while min <= max and not found: # 0<10 and true puis 6<10 and true, etc.
        mid = (max+min)//2 # mid = 5 --> 16 in list
        print(mid)
        if n > liste[mid]: # 41>16
            min = mid + 1 # min = 5+1=6
        elif n < liste[mid]:
            max = mid -1
        else:
            found = True
    if found:
        return n
    else:
        return liste[mid]

L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63]
e = 41
find_closest_bin(L,e)

%%time
print(find_closest_bin(L,e))
```

Question 6: (*10 minutes*) language : **Python**

Considérez une liste d'entiers triés **L** ainsi qu'un entier **e**. Écrivez un programme qui retourne l'index de l'élément **e** de la liste L en utilisant une recherche binaire. Si **e** n'est pas dans **L**, retournez **–1**.

Exemple :

L = [1231321,3213125,3284016,4729273,5492710]

e = 3284016

Résultat attendu : 2

```
def binary_search(L,e):
first = 0
last = len(L)-1
```

```
#complètez ici


L = [1231321,3213125,3284016,4729273,5492710]
e = 3284016
binary_search(L,e)

%%time
print(binary_search(L,e))
```

Conseil

Inspirez-vous des exercices et des conseils précédents.

Solution question 6

```
def binary_search(L,e):
first = 0
last = len(L)-1
#SOLUTION
while first <= last:
    mid = int((first+last)/2)
    print(mid)
    if L[mid] == e:
        return mid
    else:
        if randoms[mid] > e:
            last-=1
        else:
            first +=1
return False

L = [1231321,3213125,3284016,4729273,5492710]
e = 3284016
binary_search(L,e)

%%time
print(binary_search(L,e))
```

Question 7: (*10 minutes*) language : **Python**

Matrice en Python
Considérez une matrice ordonnée m et un élément l.

Une matrice ordonnée répond aux critères suivants :
[i][j]<=m[i+1][j] (une ligne va du plus petit au plus grand)
[i][j]<=m[i][j+1] (une colonne va du plus petit au plus grand)

Écrivez un algorithme qui retourne la position de l'élément l dans m. Si l n'est pas présent dans m alors il faut retourner (-1, -1)

Exemple 1 : si m=[[1,2,3,4],[4,5,7,8],[5,6,8,10],[6,7,9,11]] et que l=7. Nous souhaitons avoir la réponse (1,2) OU (3,1) (l'une des deux, pas besoin de retourner les deux résultats).

Exemple 2 : si m=[[1,2],[3,4]] et que l=7. Nous souhaitons avoir la réponse (-1,-1) car 7 n'est pas dans la matrix m.

```
def search_in_matrix(m,l):
#complètez ici


m=[[1,2,3,4],[4,5,7,8],[5,6,8,10],[6,7,9,11]]
l=7
search_in_matrix(m,l)
```

Conseil

Pour cet exercice il est nécessaire d'utiliser une double boucle **for**, c'est-à-dire : une boucle **for** dans une boucle **for**. Cela permet de parcourir tout les éléments d'un tableau à deux dimensions à l'exemple d'une matrice.

Solution question 7

```
def search_in_matrix(m,l):
#complètez ici
for i in range(len(m)): #4
    for j in range(len(m[i])):
        if m[i][j] == l:
            return (i,j)#
return (-1,-1)

m=[[1,2,3,4],[4,5,7,8],[5,6,8,10],[6,7,9,11]]
l=7
search_in_matrix(m,l)
```

3 Arbre de recherche binaire

3.1 Définition

Un arbre de recherche binaire est une structure de donnée au même titre que les listes, tuples, dictionnaires, etc. Leur particularité est qu'au lieu d'ordonner les éléments les uns à la suite des autres, les arbres de recherche binaire enregistrent les valeurs de manière relationnelle.

En effet, l'arbre est divisé en branches qui peuvent elles-même contenir deux branches (enfants) et ainsi de suite. Lorsqu'une branche n'a pas de branches subséquentes (enfants), on parle de feuille.

Les branches peuvent donc contenir de 0 à 2 autres branches. On parle alors de la branche à gauche et de celle à droite. La branche de gauche est forcément plus petite que la branche parente et celle de droite est forcément plus grande.

De cette façon, on garantit que l'arbre est toujours ordonné même en rajoutant ou en retirant des éléments

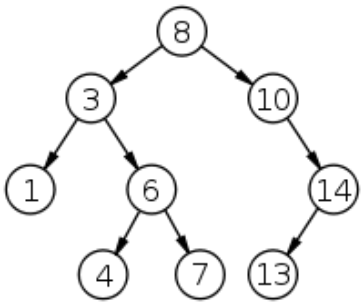

FIGURE 1 – Exemple d'arbre binaire

3.2 Exercices

Question 8: (🕓 *15 minutes*) language : **Python**

Dans l'exercice suivant nous vous donnons un arbre binaire avec les mêmes valeur que le schéma précédent et dont la racine est la variable root.

Ecrivez une fonction qui retourne **True** si la valeur valeur est dans l'arbre et **False**, sinon. Vous pouvez utiliser la variable value pour afficher la valeur d'une node et les variables **left** et **right** sur les **nodes** pour accèder aux branches enfantes, **node.value**, **node.left** et **node.right**.

Complétez la fonction **tree_search**.

```
import sys
import traceback
from Arbre import Arbre

def tree_search(node, value):
    # Complétez ici



def handle_error(): #Fonction appelée dans le bloc "except" qui gère les erreurs
    _, _, tb = sys.exc_info()
    traceback.print_tb(tb)
    tb_info = traceback.extract_tb(tb)
    filename, line, func, text = tb_info[-1]
```

```
    print('Une erreur s\'est produite sur la ligne {} dans la déclaration {}'.format(l

try: #"try" permet de tester le code pour détecter les erreurs.

    tree = Arbre(8, 3, 1, 6, 4, 7, 10, 14, 13)
    root = tree.root
    assert tree_search(root, 4) == True # devrait trouver
    assert tree_search(root, 7) == True # devrait trouver
    assert tree_search(root, 18) == False # ne devrait pas trouver
    assert tree_search(root, 21) == False # ne devrait pas trouver

    print("Bonne réponse avec le premier arbre")
except AssertionError: #"except" permet de gérer l'erreur
    handle_error()
    print("Mauvaise réponse")

try:
    tree = Arbre(1, 3, 9, 6, 14, -17, 110, 124, 13, -1, 9, 1, 40, -98, 120, 23)
    root = tree.root
    assert tree_search(root, 14) == True # devrait trouver
    assert tree_search(root, 110) == True # devrait trouver
    assert tree_search(root, 39) == False # ne devrait pas trouver
    assert tree_search(root, 18) == False # ne devrait pas trouver
    print("Bonne réponse avec le deuxième arbre")
except AssertionError as err:
    handle_error()
    print("Mauvaise réponse")
```

Conseil

Indice : Utilisez une fonction récursive.

Solution question 8

```
def tree_search(node, value):
    #Solution
    if node == None:
        return False

    if node.value == value:
        return True
    if node.value > value:
        return tree_search(node.left,value)
    else:
        return tree_search(node.right,value)
```