Algorithmes et Pensée Computationnelle

Semaine 6

Algorithmes de recherche

1 Contenu des exercices

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.

2 Recherche séquentielle (ou recherche linéaire)

Une recherche séquentielle scanne un élément après l'autre, de gauche à droite, pour trouver un élément x dans un tuple, une liste ou un dictionnaire.

Le but est de comparer x à chaque élément de la liste et de retourner l'index de l'élément correspondant.

Sa complexité dans le pire des cas est de O(n) et dans le meilleur des cas de O(1).

Question 1: (*5 minutes*) langage : **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 = [3,55,6,8,3,5,56,33,6,5,3,2,99,53,532,75,21,963,100,445,56,45,12,56,24]
x = 100

seq_search(liste, x)

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

```

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.

Conseil

TODO : Créez votre fonction de recherche linéaire en utilisant une boucle `for` ou une boucle `while`.
Aide à la compréhension de l'algorithme séquentielle : Cliquez ici

Solution

```
def seq_search(list, x):
    #complètez ici
    for i in range(len(list)): # i est considéré comme un index
        if list[i] == x:
            print("X present in list at index: ", i)
            return i
            break
```

```
        else: #only used when loop not terminated by a break statement
        #if i == len(list)-1 and list[i] != x:  #also possible
            print("X not present in list")
            return -1

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
seq_search(liste, x)

%%time
print(seq_search(liste, x))

```

Question 2: (*5 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** le plus proche de **e** en utilisant une recherche séquentielle.

Exemple :
L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63]
e = 41
Résultat attendu : 56

```
def find_closest_seq(list,n): #calcul diff avec tout element et n
diff = None #Initialisation de la variable
resultat = None
#complètez ici

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

find_closest_seq(L,e)

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

```

Conseil

Aide à la compréhension de de l'algorithme séquentielle : Cliquez ici

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 index, elem in enumerate(L):
    #complètez ici

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

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

```

Conseil

Aide à la compréhension de de l'algorithme séquentielle : Cliquez ici

3 Recherche binaire

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), très performante. 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.

Question 4: (*10-15 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))

```

Conseil

TODO : Utilisez les variables à votre disposition et trouvez un moyen d'identifier le milieu de l'intervalle.

Rappel : Une fonction récursive est une fonction qui fait appelle à elle-même.

```
# 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))

```

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

Conseil

TODO : Utilisez une boucle **while**.

Aide à la compréhension de l'algorithme de recherche binaire (itérative) : Cliquez ici

Question 5: (*10 minutes*) language : **Python**

Considérez une liste d'entiers non 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 non 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

4 Arbre de recherche binaire