Algorithmes et Pensée Computationnelle

Algorithmes de recherche - Exercices avancés

Le but de cette séance est de se familiariser avec les algorithmes de recherche. Dans cette série d'exercices, nous manipulerons des listes et collections en Java et Python. Nous reviendrons sur la notion de récursivité et découvrirons les arbres de recherche. Au terme de cette séance, l'étudiant sera en mesure d'effectuer des recherches de façon efficiente sur un ensemble de données.
Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier **Ressources**.

1 Recherche binaire

Question 1: (*10 minutes*) Recherche binaire (Python)

Soient une liste d'entiers triés **L** et 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 = [123, 321, 328, 472, 549]
e = 328

Résultat attendu : 2

```
1  def recherche_binaire(liste,e):
2      first = 0
3      last = len(L)-1
4
5      #complètez ici
6
7
8  L = [123, 321, 328, 472, 549]
9  e = 328
10 recherche_binaire(L,e)
```

Conseil

Inspirez-vous des conseils des exercices précédents (exercices basiques).

Solution

Python :

```
1  def recherche_binaire(liste,e):
2      first = 0 #correspond à l'index du premier élément de la liste
3      last = len(liste) #correspond à l'index du dernier élément de la liste
4      #SOLUTION
5      while first <= last:
6          mid = int((first+last)/2) #l'élément du milieu. La fonction int() permet d'obtenir un entier dans le cas ou
            "first + last" est un nombre impair
7          print(mid)
8          if liste[mid] == e:
9              return mid #Si la condition est juste, la fonction retourne l'index de la valeur recherchée dans la liste
10         else:
11             if liste[mid] > e:
12                 last = mid-1
13             else:
14                 first = mid+1
15     return -1 # Si la condition de la ligne 8 n'est jamais remplie, la fonction retourne -1
16
17 L = [123, 321, 328, 472, 549]
18 e = 328
19 recherche_binaire(L,e)
```

Question 2: (*15 minutes*) Recherche binaire - plus proche élément (Python)

Soient une liste d'entiers **triés** `L` et un entier `e`. Écrivez un programme retournant la valeur dans `L` la plus proche de `e` en utilisant une recherche binaire (binary search).

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


L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63]
e = 41
print(plus_proche_binaire(L,e))
# Résultat attendu : 56\\
```

Conseil

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

Solution

Python :

```
def plus_proche_binaire(liste, n):
    # SOLUTION
    min = 0
    max = len(liste)
    found = False # boolean variable
    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:
        if abs(liste[mid-1]-n) < abs(liste[mid]-n) :
            return liste[mid-1]
        elif abs(liste[mid+1]-n) < abs(liste[mid]-n) :
            return liste[mid+1]
        else :
            return liste[mid]


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

Question 3: (*20 minutes*) Recherche dans une matrice (Python)
Considérez une matrice ordonnée `m` et un élément `l`.

Pour rappel, 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, votre programme retournera `(-1, -1)`.

Exemples

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 matrice m.

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


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

Conseil

Pour cet exercice, il est nécessaire d'utiliser des boucles **for** imbriquées, c'est-à-dire : une boucle **for** dans une autre boucle **for**. Cela permet de parcourir tous les éléments d'une liste (ou d'un tableau) à deux dimensions (dans notre cas une matrice).

Solution

Python :

```
def recherche_matricielle(m,l):
    #complètez ici
    for ligne in range(len(m)): # "Pour chaque ligne de la matrice" ou "Pour chaque liste de la liste"
        for colonne in range(len(m[ligne])): # "Pour chaque colonne de la matrice" ou "Pour chaque élément de la liste"
            if m[ligne][colonne] == l:
                return (ligne,colonne)
    return (-1,-1)

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