Informatique

4
Algorithmes dichotomiques

TD 4-1
Recherche dans un tableau trié

Exercice 1: Recherche par dichotomie dans un tableau trié

Méthode Python

Python propose la recherche d'un terme dans une liste à l'aide de l'instruction : `x in L`
Ce n'est évidemment pas celle que nous allons utiliser. Vous devrez créer les algorithmes renvoyant les mêmes résultats.

Recherche simple

Soit une liste de N entiers distincts L triés par ordre croissant. On souhaite déterminer la présence d'un élément x dans L.

Question 1: Proposer une fonction $recherche\_simple(x, L)$ qui renvoie le booléen indiquant la présence (True) ou non (False) de x dans L par recherche itérative sur tous les éléments de L

Question 2: Donner la complexité en temps de cette fonction dans le meilleur et le pire des cas

Question 3: Si ce n'est pas le cas, faites-en sorte que votre fonction soit en 0(1) dans le meilleur des cas

Recherche par dichotomie

Le principe de la recherche par dichotomie dans une liste triée est le suivant :

- Retourner **False** si L est vide
- Définir les indices de recherche notés $I_g = 0$ (indice gauche) et $I_d = N - 1$ (indice droite)
- Tant que $I_g \leq I_d$ répéter la procédure suivante :
  - $I_m = \text{int}\left(\frac{I_g + I_d}{2}\right)$
  - $T_m = L[I_m]$
  - Si $x = T_m$, retourner **True**
  - Si $x < T_m$, $I_d = I_m - \mathbf{1}$
  - Si $x > T_m$, $I_g = I_m + \mathbf{1}$
- A la sortie de la boucle, retourner **False**

Remarques :

- Il est possible que $I_g > I_d$, il faut donc bien faire le test $\leq$ et non $\neq$ (Cf. remarque 1 en fin de sujet)
- Il est important de bien exclure le terme du milieu s'il n'est pas égal à x avec les $-\mathbf{1}$ et $+\mathbf{1}$ (Cf. remarque 2 en fin de sujet)
- Préférer inclure le test $x = T_m$ dans la boucle comme proposé ci-dessus afin de simplifier la rédaction

Question 4: Proposer une fonction itérative $recherche\_dicho(x, L)$ qui renvoie le booléen indiquant la présence (True) ou non (False) de x dans L par dichotomie

Question 5: Vérifiez votre code en cherchant 1, 2, 3 et 4 dans [1,2,3].

Complexité

On donne le code suivant :

```
L = [1, 9, 5, 4, 7]
L.sort()
print(L)
```

Question 6: Recopier ce code, l'exécuter et identifier le rôle de la méthode sort sur les listes

Question 7: Proposer une évolution de la fonction recherche_dicho en une fonction recherche_dicho_it(x,L) afin qu'elle renvoie, en plus de Vrai ou Faux, le nombre d'itérations de la boucle while et la valeur de $\lfloor \log_2(N) \rfloor$ (N étant le nombre d'éléments de la liste)

Remarques :

- Utiliser floor pour avoir la partie entière
- Importer log du module math puis regarder l'aide à l'aide de help(log) pour savoir l'utiliser

Question 8: Utiliser cette fonction pour rechercher -1 dans une liste de N entiers choisis aléatoirement dans [0, 100*N] et triée. Modifier la valeur de N et comparer le nombre d'itérations avec l'ordre de grandeur de $\log_2$(N)

Question 9: Pourquoi a-t-on choisi x=-1 ? Que peut-on conjecturer sur la complexité de l'algorithme de dichotomie ?

Question 10: Donner la complexité en temps de cette fonction dans le meilleur et dans le pire des cas

Validation de la fonction

Nous allons mettre en place une fonction qui vérifie que votre fonction de recherche fonctionne, en supposant que L est bien triée. Pour cela, il faut connaître toutes les situations possibles lors de l'appel de la recherche :

- 1 : L est vide
- x est dans L :
    - Nombre d'éléments de L pair
        - 2 : x est en première place
        - 3 : x est en dernière place
        - 4 : x est dans la liste, sans être ni premier, ni dernier
    - Nombre d'éléments de L impair
        - 5 : x est en première place
        - 6 : x est en dernière place
        - 7 : x est dans la liste, sans être ni premier, ni dernier
- x n'est pas dans L :
    - Nombre d'éléments de L pair
        - 8 : x est plus petit que le premier élément de L
        - 9 : x est plus grand que le dernier élément de L
        - 10 : x est compris entre le premier et le dernier élément de L, sans toutefois être présent dans L
    - Nombre d'éléments de L impair
        - 11 : x est plus petit que le premier élément de L
        - 12 : x est plus grand que le dernier élément de L
        - 13 : x est compris entre le premier et le dernier élément de L, sans toutefois être présent dans L

Question 11: Proposer une fonction test(f) qui vérifie que votre fonction de recherche f en argument renvoie le bon résultat dans toutes les configurations possibles, auquel cas elle renverra True, sinon False

Question 12: Vérifier que votre fonction recherche_dicho est correcte

Remarque 1

Avec la version de la fonction de recherche qu'il est proposé de réaliser, il est possible que $I_g > I_d$ à la dernière itération. En effet, nous avons programmé un « milieu gauche » avec $i_m = (i_g + i_d)//2$. Imaginons rechercher 0 dans $[1,2]$:

Itération	0	1
$i_g = i_m + 1$	0	0
$i_d = i_m - 1$	1	-1
i_m	0	FIN
$L[i_m]$	$1 > 0$	

Ce serait identique en recherchant 3 dans $[1,2]$ avec un « milieu droite » $i_m = (i_g + i_d + 1)//2$, ce qui conduirait à la fin à :

Itération	0	1
$i_g = i_m + 1$	0	2
$i_d = i_m - 1$	1	1
i_m	1	FIN
$L[i_m]$	$2 < 3$	

J'ai proposé à la fin du corrigé trois autres versions de la recherche qui permettent d'avoir $i_g = i_d$ à la fin du while, si cela vous intéresse.

A RETENIR : En dichotomie, bien mettre le test $\leq$ et non $!=$

Remarque 2

Lorsque l'on n'exclut pas les milieux de la recherche suivante, trois choix peuvent induire un comportement problématique du programme :

- Indice milieu : Soit n le nombre de termes de L
  - $i_m = (i_g + i_d)//2$: « Milieu gauche » car si n impair (ex n=3 : (0+2)//2=1), c'est bien le milieu, mais si n pair (ex n=2 : (0+1)//2=0), c'est le terme à gauche du milieu
  - $i_m = (i_g + i_d + 1)//2$: « Milieu droite »
- Inclusion de i_m dans i_g et i_d, que i_g ou que i_d
- Inclusion ou non du test $x = L[i_m]$ et choix du test des conditions $x \geq L[i_m]$ ou $x > L[i_m]$ si le test $x = L[i_m]$ n'est pas réalisé

Illustrons l'un des problèmes pouvant être rencontrés, par exemple en cherchant $x = 3$ dans la liste $[1,2,3]$ avec un milieu gauche :

Itération	0	1	2	3
i_g	0	1	1	…
i_d	2	2	2	…
i_m	1	1	1	…
$L[i_m]$	2	2	2	…

Ce serait identique en recherchant 1 dans $[1,2,3]$ avec un « milieu droite » $i_m = (i_g + i_d + 1)//2$:

Itération	0	1	2	3
i_g	0	0	0	…
i_d	2	1	1	…
i_m	1	1	1	…
$L[i_m]$	2	2	2	…

La solution consiste donc, puisque l'élément recherché n'est pas en position de l'indice milieu, de l'exclure de l'intervalle de recherche suivant… De chaque côté !

A RETENIR : En dichotomie, préférer exclure le milieu des sous listes et préférer inclure le test $x = L[i_m]$