Rechercher

INFORMATIQUE COMMUNE - TP n° 1.5 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ rechercher un élément dans un tableau séquentiellement ou par dichotomie itérative
- ☞ évaluer le temps d'exécution d'un algorithme avec la bibliothèque time
- ☞ générer un graphique légendé avec la bibliothèque matplotlib

A Recherche d'un élément dans un tableau

On considère une liste `L` contenant des éléments de type `int`. Cette liste est **triée** par ordre croissant de ses éléments. On veut savoir si un élément `x` est présent dans `L` et comparer les performances des approches séquentielles et la dichotomiques. On dispose des algorithmes 1, 2 et 3.

Algorithme 1 Recherche séquentielle d'un élément dans un tableau

```
1 : Fonction RECHERCHE_SÉQUENTIELLE(t, elem)
2 :     n ← taille(t)
3 :     pour i de 0 à n − 1 répéter
4 :         si t[i]= elem alors
5 :             renvoyer i                    ▷ élément trouvé, on renvoie sa position dans t
6 :     renvoyer l'élément n'a pas été trouvé
```

Algorithme 2 Recherche d'un élément par dichotomie dans un tableau trié

```
1 : Fonction RECHERCHE_DICHOTOMIQUE(t, elem)
2 :     n ← taille(t)
3 :     g ← 0
4 :     d ← n-1
5 :     tant que g ⩽ d répéter                ▷ ⩽ cas où valeur au début, au milieu ou à la fin
6 :         m ← (g+d)//2                      ▷ Division entière : un indice est un entier!
7 :         si t[m] < elem alors
8 :             g ← m + 1                     ▷ l'élément devrait se trouver dans t[m+1, d]
9 :         sinon si t[m] > elem alors
10 :            d ← m - 1                     ▷ l'élément devrait se trouver dans t[g, m-1]
11 :        sinon
12 :            renvoyer m                    ▷ l'élément a été trouvé
13 :    renvoyer l'élément n'a pas été trouvé
```

Algorithme 3 Recherche d'un élément par dichotomie dans un tableau trié, renvoyer l'indice minimal en cas d'occurrences multiples.

```
1 : Fonction RECHERCHE_DICHOTOMIQUE_INDICE_MIN(t, elem)
2 :     n ← taille(t)
3 :     g ← 0
4 :     d ← n-1
5 :     tant que g < d répéter                    ▷ attention au strictement inférieur!
6 :         m ← (g+d)//2                          ▷ Un indice de tableau est un entier!
7 :         si t[m] < elem alors
8 :             g ← m + 1                         ▷ l'élément devrait se trouver dans t[m+1, d]
9 :         sinon
10 :            d ← m                             ▷ l'élément devrait se trouver dans t[g, m]
11 :    si t[g] = elem alors
12 :        renvoyer g
13 :    sinon
14 :        renvoyer l'élément n'a pas été trouvé
```

A1. Coder l'algorithme de recherche séquentielle d'un élément dans un tableau. Lorsque l'élément n'est pas présent dans le tableau, la fonction Python renvoie `None`. Sinon, elle renvoie l'indice de l'élément trouvé dans le tableau. Vérifier que cet algorithme fonctionne sur un tableau d'entiers de 20 éléments rempli aléatoirement. Dans le pire des cas, quel est le coût d'une recherche séquentielle en fonction de la taille du tableau ?

A2. Coder l'algorithme 2. Lorsque l'élément n'est pas présent dans le tableau, la fonction Python renvoie `None`. Sinon, elle renvoie l'indice de l'élément trouvé dans le tableau. Vérifier que cet algorithme fonctionne sur un tableau d'entiers de 20 éléments rempli aléatoirement et trié.

A3. Coder l'algorithme 3. Vérifier que cet algorithme fonctionne sur un tableau d'entiers de 20 éléments rempli aléatoirement et trié et que l'indice renvoyé est bien l'indice minimal de la première occurrence de l'élément recherché.

A4. On suppose que la longueur de la liste est une puissance de 2, c'est à dire $n = 2^p$ avec $p \geqslant 1$. Combien d'étapes l'algorithme 3 comporte-t-il ? En déduire le nombre de comparaisons effectuées, dans le cas où l'élément est absent, en fonction de p puis de n, et comparer avec l'algorithme de recherche séquentielle.

> **Solution :** Supposons que la taille de la liste soit une puissance de 2 : $n = 2^p$. Soit k, le nombre de tours de boucle nécessaires pour terminer l'algorithme. À la fin de l'algorithme, le tableau ne contient qu'un seul élément. Comme on divise par deux la taille du tableau à chaque tour de boucle, à la fin de l'algorithme on a nécessairement :
>
> $$1 = \frac{n}{2^k} = \frac{2^p}{2^k}$$
>
> On en déduit que $k = \log_2 n$. On dit que cet algorithme est de complexité logarithmique en $O(\log n)$ (cf. cours du deuxième semestre)

A5. Tracer le graphique des temps d'exécution des algorithmes précédent en fonction de n. Les tracés sont-ils cohérents avec les calculs des coûts effectués précédemment ?

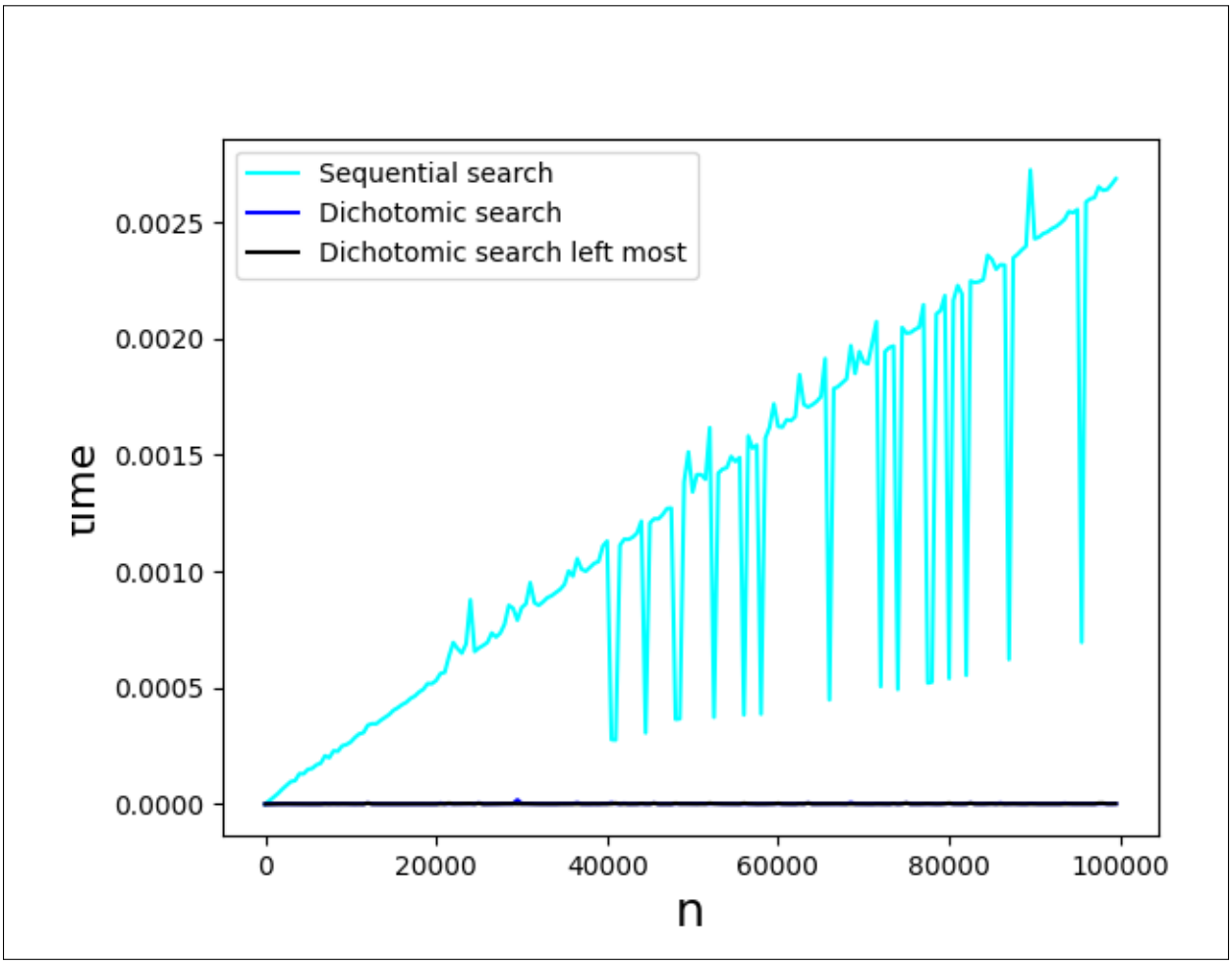
A6. La recherche dichotomique fonctionne-t-elle sur les listes non triées ? Donner un contre-exemple si ce n'est pas le cas.

Solution :

Code 1 – Recherche un élément dans un tableau

```
import time
from random import randint


def seq_search(t, elem):
    for i in range(len(t)):
        if t[i] == elem:
            return i
    return None


def dichotomic_search(t, elem):
    g = 0
    d = len(t) - 1
    while g <= d:
        m = (d + g) // 2
        # print(g, m, d)
        if t[m] < elem:
            g = m + 1
        elif t[m] > elem:
            d = m - 1
        else:
            return m
    return None


def dichotomic_search_left_most(t, elem):
    g = 0
    d = len(t) - 1
    while g < d:
        m = (d + g) // 2
        # print(g, m, d)
        if t[m] < elem:
            g = m + 1
        else:
            d = m
    if t[g] == elem:
        return g
    else:
        return None


def search_timing():
    sizes = [i for i in range(10, 100000, 500)]
    M = 5 * max(sizes)
    results = []
    for i in range(len(sizes)):
        t = sorted([randint(0, M) for _ in range(sizes[i])])
        mem_t = t[:]
```

```
            results.append([])
            for method in [seq_search, dichotomic_search,
                dichotomic_search_left_most]:
                t = mem_t[:]
                tic = time.perf_counter()
                method(t, M // 4)
                toc = time.perf_counter()
                results[i].append(toc - tic)
            print(f"#{i} - {sizes[i]} -> {results}")

        seq = [results[i][0] for i in range(len(sizes))]
        dics = [results[i][1] for i in range(len(sizes))]
        dicslm = [results[i][2] for i in range(len(sizes))]

        from matplotlib import pyplot as plt

        plt.figure()
        plt.plot(sizes, seq, color='cyan', label='Sequential search')
        plt.plot(sizes, dics, color='blue', label='Dichotomic search')
        plt.plot(sizes, dicslm, color='black', label='Dichotomic search left most')
        plt.xlabel('n', fontsize=18)
        plt.ylabel('time', fontsize=16)
        plt.legend()
        plt.show()

    # MAIN PROGRAM
    t = [0, 1, 2, 4, 7, 7, 9, 13, 17, 65, 99, 99, 99, 99, 101, 111, 111, 111, 113]

    print(t)
    for value in t:
        print(value, seq_search(t, value), dichotomic_search(t, value),
            dichotomic_search_left_most(t, value),
              dichotomic_search(t, value) == dichotomic_search_left_most(t, value))

    search_timing()
```


Sequential search
Dichotomic search
Dichotomic search left most
time
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
20000
40000
60000
80000
100000
n