Recherche dichotomique

Spé NSI - Lycée du parc

Année 2020 - 2021

Introduction

Lorsque l'on recherche un élément dans un tableau, si celui-ci est dans un ordre aléatoire, il faut examiner chaque élément.

Par contre, si le tableau est trié, on peut en tirer profit pour aller beaucoup plus vite : c'est l'algorithme de recherche dichotomique qui fait l'objet de ce chapitre.

I Rappel pour la recherche séquentielle

Exercice 1

`L` est une liste quelconque (non triée) d'entier. Écrire une fonction `appartient(L, x)` qui renvoie l'indice de la première occurence de `x` dans `L` si l'élément `x` apparait dans `L` et `None` dans le cas contraire.

On peut vérifier que pour de grandes listes, la réponse est loin d'être immédiate :

```python
from time import time
from random import randint

L = [randint(0, 10**6) for _ in range(10**6)]
t = time()
appartient(L, 0)
print(time()-t)  # L'unité est la seconde
```

```
0.042778968811035156
```

Exercice 2

En vous inspirant de l'exemple précédent, déterminer la moyenne sur une série de 10 des temps d'exécution de la fonction précédente pour des listes de n entiers aléatoires compris entre 0 et n. Pour des valeurs de $n = 10^4$, 10^5, 10^6 et 10^7.

II L'algorithme de recherche dichotomique

Exercice 3 : Intermède ludique

Reproduisez le code ci-dessous :

```python
from random import randint
def devinette(n):
    x = randint(1, n)
    print('Python vient de choisir un nombre entre 1 et', n)
    r = int(input('Devinez-le : '))
    while r != x:
        if r < x:
            print('Trop petit !')
        else:
            print('Trop grand !')
        r = int(input('Essayez encore : '))
    print('Bravo, vous avez trouvé !')
```

Faite quelques parties pour des valeurs assez grande de n (entre 10^3 et 10^5). Vous pouvez utiliser la calculatrice et noter des valeurs.

Décrivez la meilleur stratégie (celle qui permet de trouver avec le moins d'essai possible).

Exercice 4

On suppose maintenant que la liste `L = [`$x_0, x_1, \ldots, x_{n-1}$`]` est triée en ordre croissant, c'est à dire :

$$x_0 \leqslant x_1 \leqslant \cdots \leqslant x_{n-1}$$

On désigne par `L[i..j]` les éléments de L dont les indices sont compris entre i (inclu) et j (inclu). Ne pas confondre avec le tranchage `L[i:j]`

On suppose que parmi les éléments de `L[g..d]`, il y a un élément x. Et on considère un indice m tel que $g \leqslant m \leqslant d$.

Si `L[m]`<x, dans quelle nouvelle zone est-on sûr que x se trouve ?

Si `L[m]`>x, dans quelle nouvelle zone est-on sûr que x se trouve ?

Que peut-on affirmer si aucune des deux conditions précédentes n'est valide ?

Exercice 5

Compléter le code suivant selon la spécification présente dans la docstring.

```python
def rech_dicho(L, x):
    """ La liste L est supposée triée dans l'ordre croissant.
    Renvoie un indice d'une occurence de x dans L si x est présent
    et None sinon """

    g, d = 0, len(L) - 1

    while g <= d:
        m = (g + d) // 2
        if L[m] < x:
            .........
        elif L[m] > x:
            .........
        else:
            .........

    return None
```

Exercice 6

On rappelle que la méthode `.sort()` permet de trier une liste (effet de bord).

Refaire les tests de l'exercice 2 pour la fonction de recherche dichotomique (sans oublier de trier, sinon ça ne marchera pas)

Que constatez-vous ?

Exercice 7

Modifier le code de la fonction de recherche dichotomique pour afficher le nombre de tours de boucle effectué.

Exercice 8

Écrire un programme qui joue au jeu de devinette mais en trichant : la valeur n'est pas fixée au début mais les réponses données sont toujours cohérentes.