Leçon d'informatique : algorithme dichotomique

S. Benlhajlahsen - PCSI$_1$

Sommaire

Extrait du programme :

Thèmes	Exemples d'activité, au choix du professeur et non exigibles des étudiants. Commentaires
Algorithmes dichotomiques.	Recherche dichotomique dans un tableau trié. Exponentiation rapide. *On met en évidence une accélération entre complexité linéaire d'un algorithme naïf et complexité logarithmique d'un algorithme dichotomique. On met en œuvre des jeux de tests, des outils de validation.*

I Recherche dichotomique sur un tableau trié

Tableau trié : On considèrera, ici, essentiellement des listes `L` triées de longueur $n \neq 0$, c'est-à-dire :

$$\forall i \in [\![0, n-2]\!],\ \texttt{L[i+1]} \geqslant \texttt{L[i]}$$

Par exemple, `[1,2,5,10,20]` est une liste triée mais `[1,2,5,30,20]` ne l'est pas.

Principe de la recherche dichotomique : La dichotomie est un algorithme permettant, par itérations successives, de situer un point dans un intervalle ou dans une liste. Voyons un exemple très simple.

- Alice[a] pense à un entier x entre 0 et 100, disons 33, et Bob doit le deviner
- Bob peut seulement proposer des valeurs, et Alice répond en précisant si le nombre est plus grand ou plus petit.

Pour imiter l'algorithme de dichotomie, Bob choisit de toujours proposer la valeur centrale de l'intervalle étudié.

Valeur proposée	Réponse d'alice	conclusion de Bob
50	trop grand	$x \in [\![0, 49]\!]$
24	trop petit	$x \in [\![25, 49]\!]$
37	trop grand	$x \in [\![25, 36]\!]$
30	trop petit	$x \in [\![31, 36]\!]$
33	bonne réponse !	$x = 33$

a. Il est d'usage international, en informatique, d'appeler "Alice" et "Bob" les deux personnes intervenant dans une communication, ce qui est un peu plus convivial que "A" et "B".

Remarque : On notera que cette recherche dichotomique n'est possible que sur une liste triée.

Implémentation de l'algorithme : On note `n` la longueur de la liste. On va travailler pour la recherche sur les indices par souci de simplicité.

- **Initialisation** : on crée deux variables `g` et `d` qui vont délimiter l'intervalle de recherche. Initialement, `g=0` et `d=n-1`.
- **Á chaque itération :**
  - on calcule l'indice médian `m`[a] qui délimitera deux sous-intervalle de recherche (voir figure 1) ;
  - On compare `x` avec l'élément d'indice `moy` dans la liste.
    - S'il est avant, on redéfinit le bord droit `d` pour limiter la recherche à la liste de gauche, puis on réitère.

- S'il est après, on redéfinit le bord gauche `g` pour limiter la recherche à la liste de droit, puis on réitère.

- **Conclusion :** On renvoie `True` ou `False` suivant l'appartenance ou non de `x` à la liste `L` (voir figure 2). On peut aussi renvoyer l'indice de `x` à la liste `L` (voir figure 3).

Comme on ne sait pas le nombre d'itérations nécessaires à la recherche, on utilise une boucle `while`.

a. qui doit être un entier donc on le calculera par $\left\lfloor \frac{n+g}{2} \right\rfloor$.

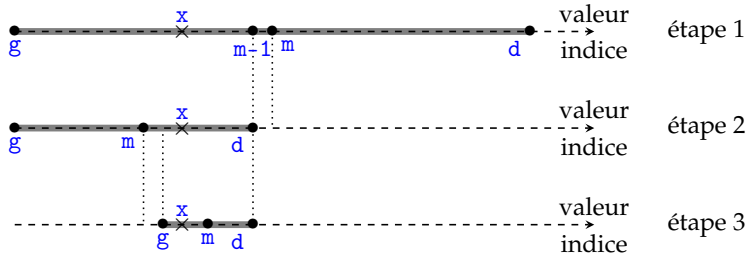


FIGURE 1

```python
def RechDicho1(L, x):
    # Renvoie True ou False
    n = len(L)
    g = 0 # indice de gauche
    d = n-1 # indice de droite
    while d>=g:
        m = (g+d)//2  # indice median
        if x ==L[m]:
            return True
        elif x>L[m]:
            g = m+1
        else:
            d = m-1
    return False
print(RechDicho1([1,2,4,6,10],6))
```

FIGURE 2 – Algorithme de recherche dichotomique

```python
def RechDicho2(L, x):
    # renvoie la position de x dans L
    n = len(L)
    g = 0 # indice de gauche
    d = n-1 # indice de droite
    while d>=g:
        m = (g+d)//2  # indice median
        if x ==L[m]:
            return m
        elif x>L[m]:
            g = m+1
        else:
            d = m-1
    return -1
print(RechDicho2([1,2,4,6,10],6))
```

FIGURE 3 – Algorithme de recherche dichotomique

II Complexité de la recherche dichotomique

Rappel : On rappelle que la complexité est une estimation du nombre d'opération élémentaire effectué par un algorithme.

II.A Rappel sur la complexité de la recherche linéaire

On rappelle que la recherche linéaire ou recherche séquentielle (voir figure 4) a une complexité en $O(n)$. Cette recherche ne nécessite pas que la liste soit triée.

II.B Complexité de la recherche dichotomique

Complexité logarithmique : Si on note n la taille de la liste triée, alors, pour n « suffisamment grand », le nombre d'opération varie comme $\log_2(n)$. On parle de complexité logarithmique en $O(\ln(n))$ ou en $O(\log_2(n))$.

Une démonstration rapide : Notons p le nombre total de tour de boucle. Initialement, la taille de la liste est $n = d - g + 1$. À chaque tour de boucle, la taille de la liste est au moins divisée par 2. La condition d'arrêt de la boucle montre que l'on

```python
# recherche de a dans L
def recherche(a, L):
    res = False
    for x in L :
        if x==a:
            res = True
    return res
print(recherche(1, [4,3,1,2]))
```

```python
def recherche(a, L):
    for x in L :
        if x==a:
            return True
    return False
print(recherche(1, [4,3,1,2]))
```

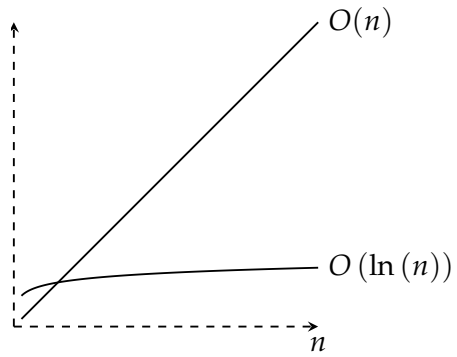
FIGURE 4 – Recherche séquentielle d'un élément dans une liste non nécessairement triée.

s'arrêtera lorsque $\frac{n}{2^p} \leqslant 1$ soit encore $n \leqslant 2^p$:

$$\ln(n) \leqslant \ln(2^p) \Rightarrow p \leqslant \frac{\ln(n)}{\log(2)} = \log_2(n) \Rightarrow p = O(\log_2(n))$$

Remarque :

Cet algorithme est donc plus efficace que la recherche séquentielle. En effet, pour n suffisamment grand, $\ln(n) \leqslant n$. C'est le principal avantage de cette méthode même si c'est imperceptible pour des petites listes.



II.C Vérification par le temps d'exécution

Pour vérifier expérimentalement, on se propose donc d'utiliser la méthode `time()` du module `time` qui permet de mesurer un instant t. On génère alors aléatoirement des listes d'entiers de taille n triée et on cherche un élément `a`. On mesure alors le temps d'exécution de la recherche linéaire et de la recherche dichotomique. On donne alors la représentation graphique en échelle linéaire en figure 5 et en échelle logarithmique en figure 6.

III Conclusion

Diviser pour régner - *Divide and conquer* : L'algorithme de recherche dichotomique est un exemple de méthode informatique appelée « diviser pour régner ». Cela consiste en :

- **diviser** : découper un problème initial en sous-problèmes ;
- **régner** : résoudre les sous-problèmes (récursivement ou directement s'ils sont assez petits).

Ici, on constate qu'on **divise** l'intervalle de recherche.

Autre exemple d'algorithmes dichotomiques :

- l'algorithme d'exponentiation rapide que l'on verra avec la récursivité ;
- la recherche des zéros d'une fonctions utiles pour le calcul numérique en science.
- les tris fusion et rapides vus dans le chapitre sur les tris.

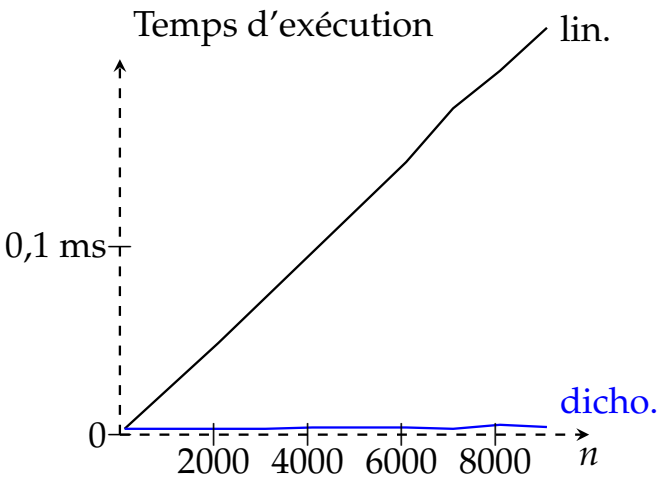


FIGURE 5 – Temps d'exécution en échelle linéaire.

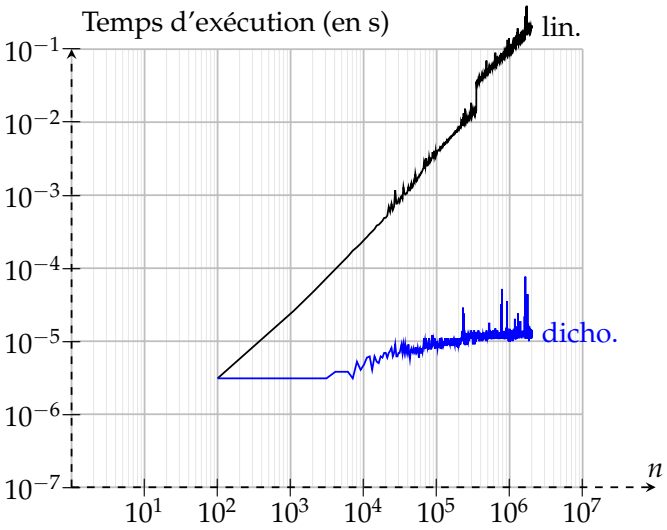


FIGURE 6 – Temps d'exécution en échelle logarithmique.