

RECHERCHE DANS UN TABLEAU

I. Recherche séquentielle

On considère la fonction suivante, qui prend en argument un élément et une liste et qui renvoie **True** si l'élément se trouve dans la liste, **False** sinon :

```
def appartient(e, L):
    for i in range(len(L)):
        if L[i] == e:
            return True
    return False
```

et on souhaite évaluer la complexité de cette fonction.

- Description de la taille de l'entrée : nombre n d'éléments de L
- Opération élémentaire caractéristique de l'algorithme : comparaison avec un élément de la liste
- Dans le pire des cas : n itérations, chacune comportant une comparaison : complexité en $O(n)$

II. Recherche dichotomique dans une liste triée

1) Algorithme et implémentation

On souhaite savoir si un élément appartient à une liste de nombres triés dans l'ordre croissant.

On peut alors procéder par dichotomie ; on compare l'élément recherché avec l'élément du milieu de la liste :

- s'il y a égalité, on renvoie **True**,
- s'il est plus petit, on continue la recherche dans la moitié gauche de la liste,
- s'il est plus grand, on continue la recherche dans la moitié droite de la liste

Exemple

Recherche de 12 dans la liste [3, 5, 7, 9, 12, 15, 16]

Comparaison de 12 avec 9

3	5	7	9	12	15	16
---	---	---	---	----	----	----

Comparaison de 12 avec 15

3	5	7	9	12	15	16
---	---	---	---	----	----	----

Comparaison de 12 avec 12

3	5	7	9	12	15	16
---	---	---	---	----	----	----

Seules trois comparaisons ont été nécessaires.

Utiliser des sous-listes serait coûteux (copie des portions de liste). On va donc délimiter la portion du tableau dans laquelle rechercher l'élément à l'aide de deux indices **deb** (indice de début de la portion à prendre en compte) et **fin** (indice de fin).

On obtient la fonction Python suivante :

```
def recherche_dichotomique(e, L):
    deb, fin = 0, len(L)-1
    while deb <= fin:
        milieu = (deb + fin)//2
        if e == L[milieu]:
            return True
        if e < L[milieu]: #on continue à gauche
            fin = milieu - 1
        else: #on continue à droite
            deb = milieu + 1
    return False
```

2) Complexité

Notons d_k et f_k les valeurs respectives de **deb** et **fin** après k itérations.

On va montrer que $f_k - d_k < \frac{n}{2^k}$ où n est le nombre d'éléments de la liste.

- Avant l'entrée dans la boucle, $d_0 = 0$ et $f_0 = n - 1$ donc $f_0 - d_0 < n = \frac{n}{2^0}$.
- Si $f_k - d_k < \frac{n}{2^k}$ et si on effectue une itération supplémentaire et que la recherche continue :

— ou bien on continue à gauche : $d_{k+1} = d_k$ et $f_{k+1} = \left\lfloor \frac{d_k + f_k}{2} \right\rfloor - 1$, donc

$$f_{k+1} - d_{k+1} \leq \frac{d_k + f_k}{2} - 1 - d_k < \frac{f_k - d_k}{2} < \frac{n}{2^{k+1}}$$

— ou bien on continue à droite : $d_{k+1} = \left\lfloor \frac{d_k + f_k}{2} \right\rfloor + 1$ et $f_{k+1} = f_k$, donc

$$f_{k+1} - d_{k+1} \leq f_k - \frac{d_k + f_k}{2} = \frac{f_k - d_k}{2} < \frac{n}{2^{k+1}}$$

Supposons qu'on a effectué $k \geq \log_2(n)$ itérations. Alors $2^k \geq n$, donc $\frac{n}{2^k} \leq 1$. Par conséquent, $f_k - d_k \leq 0$ et on effectue au plus une dernière itération.

La complexité de la recherche dichotomique est donc en $O(\log(n))$

3) Correction

Montrons que la propriété suivante est un invariant de boucle :

pour tout $i \in \llbracket 0, n-1 \rrbracket$, si $i < d_k$ alors $L[i] < e$, et si $i > f_k$ alors $L[i] > e$

— Avant l'entrée dans la boucle, $d_0 = 0$ et $f_0 = n-1$ donc la propriété est vérifiée.

— On suppose qu'après k itérations, la propriété est vérifiée.

Soit $m_k = \left\lfloor \frac{d_k + f_k}{2} \right\rfloor$. Si on effectue une itération supplémentaire :

— si $L[m_k] = e$, d_k et f_k ne sont pas modifiés, donc la propriété reste vraie ;

— si $L[m_k] > e$, $d_{k+1} = d_k$ et $f_{k+1} = m_k - 1$, donc pour tout $i \in \llbracket 0, n-1 \rrbracket$

— si $i < d_{k+1}$, alors $i < d_k$ donc $L[i] < e$,

— si $i > f_{k+1} = m_k - 1$, alors $i \geq m_k$, donc comme la liste est triée dans l'ordre croissant, $L[i] \geq L[m_k] > e$;

— si $L[m_k] < e$, $d_{k+1} = m_k + 1$ et $f_{k+1} = f_k$, donc pour tout $i \in \llbracket 0, n-1 \rrbracket$

— si $i < d_{k+1} = m_k + 1$, alors $i \leq m_k$, donc $L[i] \leq L[m_k] < e$,

— si $i > f_{k+1}$, alors $i > f_k$ donc $L[i] > e$.

En sortie de boucle,

— soit on est sorti par un « **return True** » et l'élément est dans la liste

— soit $f_k < d_k$ et on renvoie **False** ; or pour tout $i \in \llbracket 0, n-1 \rrbracket$

— ou bien $i > f_k$, alors $L[i] > e$;

— ou bien $i \leq f_k$, alors $i < d_k$ et $L[i] < e$

Dans les deux cas, $L[i] \neq e$.

Donc l'élément n'est pas dans la liste.

III. Résumé

— Recherche séquentielle :

— Pas de contrainte sur la liste.

— Complexité en $O(n)$ (complexité linéaire)

— Recherche dichotomique :

— Dans une liste triée uniquement.

— Complexité en $O(\ln n)$ (complexité logarithmique)

— Implémentation : ne pas utiliser de sous-listes (slicing), sinon la complexité explose.

La recherche dichotomique est plus efficace mais nécessite que la liste soit triée.

Les tris par comparaison ayant au mieux une complexité en $O(n \ln n)$ (voir spé), il n'est pas intéressant de trier la liste juste pour une recherche.