Recherche dichotomique

Algorithmique

Résumé

Chercher un élément dans une collection de données est l'une des opérations les plus fréquentes en informatique : trouver un mot dans un dictionnaire, un contact dans un répertoire, un livre dans une bibliothèque. Ce cours présente deux algorithmes de recherche et compare leur efficacité.

① Recherche séquentielle

La méthode la plus naturelle pour chercher un élément dans une liste consiste à la parcourir du début à la fin en comparant chaque élément à la valeur cherchée. C'est la **recherche séquentielle** (ou recherche linéaire).

```
def chercher(liste, x):
    """Renvoie True si x est dans liste, False sinon."""
    for element in liste:
        if element == x:
            return True
    return False
```

Cette méthode fonctionne toujours, que la liste soit triée ou non. Mais elle peut être lente : dans le **pire des cas** (élément absent ou en dernière position), il faut examiner les n éléments de la liste. On dit que la complexité de la recherche séquentielle est en $O(n)$.

② Peut-on faire mieux ?

Imaginez que l'on vous demande de trouver le mot « parallélisme » dans un dictionnaire de mille pages. Vous n'allez pas commencer à la page 1 pour lire chaque entrée une par une. Vous ouvrez le dictionnaire vers le milieu, vous regardez où vous en êtes, puis vous continuez dans la bonne moitié. C'est exactement le principe de la **recherche dichotomique**.

⚠ **Condition indispensable.** La recherche dichotomique ne fonctionne que si les éléments sont **triés**. Dans un dictionnaire, les mots sont classés par ordre alphabétique ; dans une liste Python, les valeurs doivent être rangées dans l'ordre croissant.

2.1 Le jeu du nombre mystère

L'ordinateur choisit un nombre entre 1 et 100 et vous devez le deviner. À chaque essai, on vous dit « c'est plus » ou « c'est moins ».

Stratégie naïve : essayer 1, puis 2, puis 3… Au pire, 100 essais.

Stratégie dichotomique : toujours proposer le milieu de l'intervalle restant.

Supposons que le nombre mystère est **73** :

Essai	Proposition	Réponse	Intervalle restant
1	50	« C'est plus »	[51 ; 100]
2	75	« C'est moins »	[51 ; 74]
3	62	« C'est plus »	[63 ; 74]
4	68	« C'est plus »	[69 ; 74]
5	71	« C'est plus »	[72 ; 74]
6	73	Trouvé !	–

En six essais, on a trouvé parmi 100 possibilités. À chaque étape, on élimine **la moitié** des possibilités.

③ Algorithme de recherche dichotomique

3.1 Principe

On dispose d'une liste **triée dans l'ordre croissant**. On maintient deux indices, `debut` et `fin`, qui délimitent la zone de recherche. À chaque étape :

1. on calcule l'indice du milieu : `centre` = (`debut` + `fin`) ÷ 2 (division entière) ;
2. on compare la valeur `liste[centre]` à l'élément cherché :
   - si elle est **égale** : l'élément est trouvé ;
   - si elle est **inférieure** : on cherche dans la moitié droite (`debut` ← `centre` + 1) ;
   - si elle est **supérieure** : on cherche dans la moitié gauche (`fin` ← `centre` - 1) ;
3. on recommence tant que `debut` ≤ `fin`. Si cette condition n'est plus vérifiée, l'élément est absent.

3.2 Exemple pas à pas

Cherchons la valeur **14** dans la liste triée [1, 2, 5, 9, 10, 14, 17, 24, 41].

Étape 1. `debut` = 0, `fin` = 8, `centre` = 4. La valeur centrale est `liste[4]` = 10.

1	**2**	**5**	**9**	**10**	**14**	**17**	**24**	**41**
0	1	2	3	**4**	5	6	7	8

Comme 10 < 14, on cherche dans la moitié **droite** : `debut` ← 5.

Étape 2. `debut` = 5, `fin` = 8, `centre` = 6. La valeur centrale est `liste[6]` = 17.

14	**17**	24	41

Comme 17 > 14, on cherche dans la moitié **gauche** : `fin` ← 5.

Étape 3. `debut` $= 5$, `fin` $= 5$, `centre` $= 5$. La valeur centrale est `liste[5]` $= 14$. L'élément est **trouvé en trois étapes**.

3.3 Exemple avec un élément absent

Cherchons la valeur **7** dans la même liste.

1. `centre` $= 4$, valeur 10. Comme $10 > 7$, moitié gauche : `fin` $\leftarrow 3$.
2. `centre` $= 1$, valeur 2. Comme $2 < 7$, moitié droite : `debut` $\leftarrow 2$.
3. `centre` $= 2$, valeur 5. Comme $5 < 7$, moitié droite : `debut` $\leftarrow 3$.
4. `centre` $= 3$, valeur 9. Comme $9 > 7$, moitié gauche : `fin` $\leftarrow 2$.

Maintenant `debut` $= 3 >$ `fin` $= 2$: la boucle s'arrête. L'élément **7 est absent**. Quatre étapes ont suffi pour le déterminer.

④ Implémentation en Python

4.1 Version booléenne

```python
def recherche_dichotomique(liste, element):
    """Renvoie True si element est dans liste (triée), False sinon."""
    debut = 0
    fin = len(liste) - 1
    while debut <= fin:
        centre = (debut + fin) // 2
        if liste[centre] == element:
            return True
        elif liste[centre] < element:
            debut = centre + 1
        else:
            fin = centre - 1
    return False
```

4.2 Version renvoyant l'indice

```python
def indice_dichotomie(liste, element):
    """Renvoie l'indice de element dans liste (triée), ou None si absent."""
    debut = 0
    fin = len(liste) - 1
    while debut <= fin:
        centre = (debut + fin) // 2
        if liste[centre] == element:
            return centre
        elif liste[centre] < element:
            debut = centre + 1
        else:
            fin = centre - 1
    return None
```

5 Complexité

À chaque étape, la taille de la zone de recherche est **divisée par deux**. Après k étapes, il reste au plus $\frac{n}{2^k}$ éléments. La recherche se termine lorsque $\frac{n}{2^k} < 1$, soit $k > \log_2(n)$.

Ici, $\log_2(n)$ est le logarithme de base 2 de n, vu au chapitre Complexité : c'est l'exposant k tel que $2^k = n$. Par exemple, $\log_2(8) = 3$ et $\log_2(1\,000\,000) \approx 19{,}9$. La notation $\lfloor x \rfloor$ désigne la partie entière de x, que Python calcule avec `math.floor(x)` : $\lfloor 19{,}9 \rfloor = 19$.

Le nombre maximal d'étapes est donc $\lfloor \log_2(n) \rfloor + 1$.

Taille n	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	10^6
Séquentielle	1	2	4	8	16	32	64	128	10^6
Dichotomie	1	2	3	4	5	6	7	8	20

Pour une liste d'**un million** d'éléments, la dichotomie nécessite au plus **20 comparaisons** contre un million pour la recherche séquentielle.

6 Terminaison de l'algorithme

Considérons le **variant de boucle** $v = \texttt{fin} - \texttt{debut} + 1$, le nombre d'éléments restant à examiner. À chaque tour, soit on trouve l'élément (et on sort), soit on met à jour `debut` ou `fin` de sorte que v diminue strictement d'au moins 1. Comme v est un entier positif qui diminue strictement, la boucle se termine en un nombre fini d'étapes.

Résumé

Algorithme	**Prérequis**	**Complexité**	**Étapes ($n = 10^6$)**
Séquentielle	Aucun	$O(n)$	10^6
Dichotomie	Liste triée	$O(\log_2 n)$	20

Exercices

Exercice 1 *Déroulement à la main* Déroulez l'algorithme de recherche dichotomique pour trouver la valeur **24** dans la liste triée [3, 7, 11, 15, 19, 24, 28, 33, 40].

Pour chaque étape, indiquez les valeurs de `debut`, `fin`, `centre` et la valeur centrale, puis la décision prise.

Exercice 2 *Élément absent* Déroulez l'algorithme de recherche dichotomique pour chercher la valeur **20** dans la même liste [3, 7, 11, 15, 19, 24, 28, 33, 40].

Combien d'étapes sont nécessaires pour conclure que 20 est absent ?

Exercice 3 *Compléter le tableau de complexité* Compléter le tableau ci-dessous :

Taille n	**1**	**10**	**100**	**1 000**	**10 000**	**100 000**
Séquentielle						
Dichotomie						

Indication : utiliser `from math import log2` et `int(log2(n)) + 1` en Python.

Exercice 4 *Erreur dans le code* Un élève a écrit le code suivant. Il contient une erreur qui peut provoquer une boucle infinie. Trouvez l'erreur et corrigez-la.

```
def recherche(liste, element):
    debut = 0
    fin = len(liste) - 1
    while debut <= fin:
        centre = (debut + fin) // 2
        if liste[centre] == element:
            return True
        elif liste[centre] < element:
            debut = centre      # <-- ligne suspecte
        else:
            fin = centre - 1
    return False
```

Exercice 5 *Recherche dichotomique sur liste non triée* On applique l'algorithme de recherche dichotomique sur la liste **non triée** [8, 2, 15, 3, 7, 12, 1] pour chercher la valeur 15.

1. Déroulez l'algorithme. L'élément est-il trouvé ?

2. Pourtant, 15 est bien dans la liste. Expliquez pourquoi l'algorithme échoue.