

6.5

Recherche dichotomique

NSI 1ÈRE - JB DUTHOIT

6.5.1 Principe

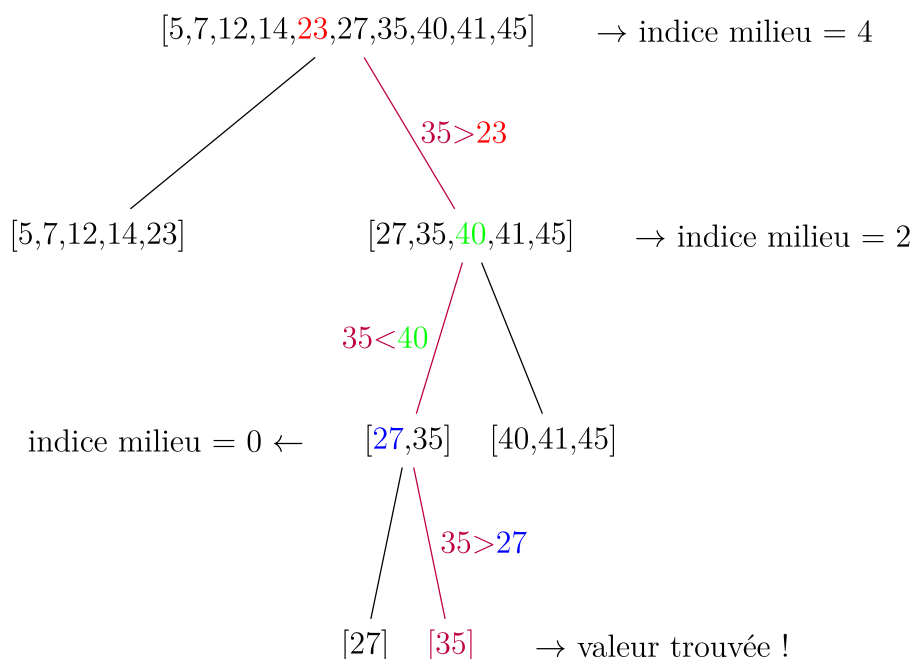
Considérons le tableau trié $[5, 7, 12, 14, 23, 27, 35, 40, 41, 45]$.

Si on veut trouver une valeur, 35 par exemple, il est possible d'utiliser l'algorithme de recherche d'une occurrence. Dans le pire des cas, on parcourt le tableau en entier. Dans notre exemple, il faut faire 7 comparaisons.

Mais comme la liste de départ est triée, la recherche dichotomique permet d'améliorer la performance de la recherche.

On souhaite rechercher l'entier 35 :

- Si la liste est vide, la recherche est finie et on renvoie False
- Sinon, on cherche la valeur la plus "centrale" dans la liste et on la compare avec l'élément recherché.
 - Si la valeur est celle recherchée, la recherche est finie, et on renvoie True
 - Si la valeur est strictement inférieure à l'élément recherché, reprendre la recherche avec la seconde moitié de la liste
 - Si la valeur est strictement supérieure à l'élément recherché, reprendre la recherche avec la première moitié de la liste.



☞ Il suffit ici de 3 tours de boucles.

6.5.2 Algorithme



Exercice 6.140

Créer une fonction `recherche__dichotomique(T,element)` qui prend en paramètre un tableau d'entiers et un entier, et renvoie `True` si l'élément est dans la liste, `False` sinon

6.5.3 Complexité

Etude théorique

Considérons un tableau de taille n .

☞ On admet que la partie entière de $\log_2(n)$ augmentée de 1 donne le nombre de chiffres de l'écriture binaire de n .

Exemples

- $\log_2(4) = 2$, et l'écriture binaire de 4 (100) contient 3 chiffres.
- $\log_2(64) = 6$, et l'écriture binaire de 64 (1000000) contient 7 chiffres.
- $\log_2(24) \simeq 4.58$, et l'écriture binaire de 24 (11000) contient 5 chiffres.

A chaque étape de la recherche dichotomique, on divise la longueur de la liste par 2 (à 1 près).

Or, diviser un entier par 2 revient à supprimer le bit de poids faible dans son écriture binaire (toujours à 1 près!)

- 24 est représenté en binaire par 11000. 12 est représenté en binaire par 1100. (on a supprimé le dernier chiffre).
- et 6 est représenté par 110 (on a supprimé le dernier chiffre).
- 3 est représenté par 11 (on a supprimé le dernier chiffre).
- si on continue le procédé en enlevant une nouvelle fois le dernier chiffre, on arrive à 1 comme écriture binaire, qui correspond au nombre 1. C'est bien la moitié de 3, à 1 près.

Ainsi, si la longueur du tableau est n , et puisqu'on divise par 2 la longueur du tableau à chaque étape, on enlève le bit de poids faible à l'écriture binaire de n à chaque étape.

Comme l'écriture binaire de n comporte $\log_2(n) + 1$ chiffres, il faudra au maximum $\log_2(n)$ étapes!

Ainsi, le nombre maximal d'étapes pour la recherche dichotomique dans un tableau de longueur n est $\log_2(n)$.

Autrement dit, le nombre maximum de comparaisons dans un tableau de longueur n est $\log_2(n)$.

on a donc une complexité temporelle proportionnelle à $\log_2(n)$. On parle de complexité logarithmique.

On note $\mathcal{O}(\log_2(n))$.

Illustration sur des exemples

● Exercice 6.141

Compléter le tableau suivant, vous comprendrez l'intérêt de la recherche dichotomique par rapport à une recherche séquentielle...Bien sûr, il faut que le tableau soit trié!!

Vous indiquerez les temps en nanoseconde !

⚠ Il faudra sûrement utiliser un chronomètre plus fin avec `perf__counter` ou `perf__counter__ns`

Taille de la liste	0	1	2	4	128	1 000	10 000	100 000	1 000 000	n
Recherche séquentielle										$\mathcal{O}(n)$
Recherche dichotomique										$\mathcal{O}(\log_2(n))$

6.5.4 Terminaison

La boucle TANT QUE peut poser problème, et il faut s'assurer que dans tous les cas, la boucle n'est pas infinie.

Considérons comme variant de boucle la longueur du tableau en cours.

● Exercice 6.142

| Montrer que ce variant décroît strictement et finira par être strictement inférieur à 1.