STRUCTURES DE DONNÉES ET ALGORITHMES · CHAPITRE 13

Éliminer la moitié d’un intervalle trié

Construisez une recherche dichotomique avec bornes incluses, invariant et trace. Testez absence, doublons et intervalle vide.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Réduire un intervalle candidat dans une collection triée.
- La dichotomie exige un ordre et un accès central adapté.
- Les bornes incluses gardent toutes les positions encore candidates ; le milieu déjà comparé est exclu au tour suivant.

Prérequis. Avoir étudié les chapitres DSA précédents ; connaître tableaux, invariants et récursivité.

Partir d’un besoin

Une suite croissante permet de comparer la cible à une valeur centrale. Si la cible est plus grande, pourquoi peut-on écarter toute la moitié gauche ?

Définir la structure et les conventions

La **recherche dichotomique** suppose une suite triée en ordre croissant au sens large, avec des valeurs comparables. Elle maintient un intervalle de candidats **inclusif** [low, high]. Son invariant est : si la cible existe, au moins une de ses occurrences se trouve dans l’intervalle restant.

Le milieu vaut `low + floor((high – low)/2)`. Si la valeur centrale est trop petite, toutes les positions jusqu’au milieu sont trop petites : low devient middle + 1. Si elle est trop grande, high devient middle - 1. Sinon, la recherche renvoie cette occurrence. Le contrat n’exige pas la

première occurrence en cas de doublons.

Quand low dépasse high, l'intervalle est vide : la cible est absente. À chaque échec de comparaison, le milieu est exclu, ce qui garantit la progression. Les bornes initiales d'une suite vide satisfont déjà low > high.

Suivre les états

Chercher 9 dans [1, 3, 5, 7, 9] en indices Python

low	high	middle	Décision
0	4	2 (valeur 5)	low devient 3
3	4	3 (valeur 7)	low devient 4
4	4	4 (valeur 9)	Trouvé

Comparer les réalisations

La cible 9 est trouvée à l'indice 4 en Python, 5 en MATLAB. Enregistrez la version MATLAB dans binary_search_demo.m. La recherche n'effectue ni copie ni tri de l'entrée : son ordre est une précondition.

Pseudo-code

```
// example.pseudo
// binary_search
low <- 0
high <- length(values) - 1
WHILE low <= high DO
    middle <- low + floor_divide(high - low, 2)
    IF values[middle] == target THEN
        RETURN middle
    ELSE IF values[middle] < target THEN
        low <- middle + 1
    ELSE
        high <- middle - 1
    END IF
END WHILE
RETURN -1
```

Python

```
# binary_search.py
def binary_search(values, target):
    low = 0
    high = len(values) - 1
    while low <= high:
        middle = low + (high - low) // 2
        if values[middle] == target:
            return middle
        if values[middle] < target:
            low = middle + 1
        else:
            high = middle - 1
    return -1
values = [1, 3, 5, 7, 9]
position = binary_search(values, 9)
print(position)
```

MATLAB

```
% binary_search_demo.m
values = [1, 3, 5, 7, 9];
position = binary_search(values, 9);
disp(position);
function position = binary_search(values, target)
    low = 1;
    high = numel(values);
    position = 0;
    while low <= high
        middle = low + floor((high - low) / 2);
        if values(middle) == target
            position = middle;
            return;
        end
        if values(middle) < target
            low = middle + 1;
        else
            high = middle - 1;
        end
    end
end
```

Justifier et compter

Le pire cas demande O(log n) comparaisons pour n ≥ 2 et la version itérative utilise Θ(1) mémoire auxiliaire. Cela suppose l'accès central Θ(1), comme dans un tableau. Une liste chaînée ne fournit pas cet accès gratuitement. Trier d'abord une entrée coûte aussi du travail : pour une seule recherche, un parcours linéaire peut être préférable.

Prévoir puis expérimenter

Vérifiez une cible absente entre deux valeurs, une cible plus petite ou plus grande que toutes, puis le vide et [7]. Avec [7, 7, 7], acceptez toute position contenant 7 selon le contrat actuel.

Exercice corrigé

▼ Pourquoi low = middle peut-il empêcher l'arrêt ?

Sur un intervalle d'un seul élément trop petit, middle vaut low. Réaffecter la même borne ne réduit pas l'intervalle. middle + 1 exclut la position déjà vérifiée.

À retenir

- La dichotomie exige un ordre et un accès central adapté.
- Les bornes incluses gardent toutes les positions encore candidates ; le milieu déjà comparé est exclu au tour suivant.
- Avec des doublons, la version étudiée renvoie une occurrence, sans promettre la première.

Version web : https://www.clearcode.fr/courses/programming/dsa/search-sort/binary-search