1 Problématique

Rechercher un élément dans un tableau est une opération courante. Cette tâche a un coût qui dépend de la taille du tableau. Cependant, si le tableau est déjà trié il est possible d'accélérer grandement la recherche.

Comment implémenter une recherche efficace dans un tableau trié ?

2 Recherche classique dans un tableau

2.1 Génération des données

On demande à dix mille personnes de penser à un nombre entre 0 et 1000000 et on stocke les résultats dans un tableau au fur et à mesure des réponses.

Activité 1 : Construire par compréhension un tableau de dix mille entiers compris entre 0 et 1000000.

2.2 Recherche dans les données

Pour vérifier la présence d'une valeur dans les données, il faut parcourir le tableau élément par élément.

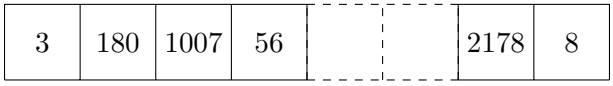

FIGURE 1 – Parcours séquentiel

À retenir

Dans le pire des cas la complexité temporelle de la recherche dépend du nombre d'éléments.

$$O(n)$$

Activité 2 :

1. Écrire la fonction `recherche_classique(tab: list, cherche: int)` → `bool` qui renvoie `True` si l'entier `cherche` est présent dans le tableau.
2. Tester la fonction : vérifier si le nombre 575000 a été choisi par une personne.
3. À l'aide de la méthode `time` de la bibliothèque `time` mesurer la durée d'exécution de la fonction.

3 Recherche dans un tableau trié

3.1 Des données ordonnées

Imaginons maintenant la même expérience mais prenons la peine de trier les éléments au fur et à mesure de leur ajout dans le tableau de données.

Christophe Viroulaud

FIGURE 2 – Tableau trié

Activité 3 : Pour simplifier nous allons utiliser la méthode `sort` pour trier les données.

1. Construire par compréhension un tableau de dix mille entiers compris entre 0 et 1000000.
2. Trier le tableau.

3.2 Recherche dichotomique

Les données étant triées, le principe de la dichotomie, pour chercher la présence d'un élément, consiste à :

— couper le tableau en deux parties égales,

— ne garder que la partie contenant l'élément,

— répéter l'opération jusqu'à trouver l'élément ou avoir une partie vide.

Cherchons 765 dans le tableau suivant :

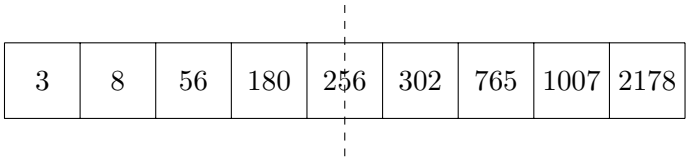

FIGURE 3 – Séparons les données en deux parties

FIGURE 4 – 256 n'est pas le nombre recherché et il est inférieur à 765

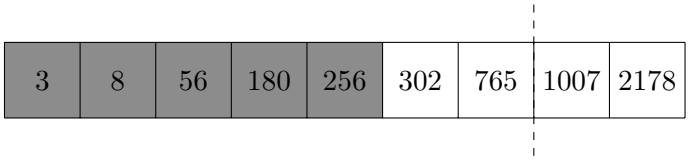

FIGURE 5 – Séparons les données restantes en deux parties

FIGURE 6 – Nous pouvons éliminer la partie supérieure à 765

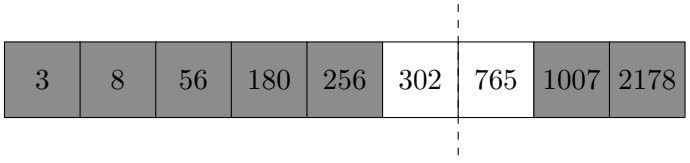

FIGURE 7 – Dernière séparation

Christophe Viroulaud

FIGURE 8 – 765 a été trouvée en trois itérations

Activité 4 : Écrire la fonction `recherche_dicho(tab: list, cherche: int)` → `bool` qui applique le principe de la dichotomie. Pour séparer les données en deux parties (à peu près) égales il faudra calculer l'indice médian de la partie encore valide.

3.3 Efficacité

À chaque itération la quantité de données (notée **n**) à étudier est divisée par deux. Dans le pire des cas, on divise jusqu'à ce que la taille de la partie restante soit inférieure ou égale à 1.

$$\frac{n}{2^x} = 1$$

$$\Leftrightarrow n = 2^x$$

Activité 5 :

1. Encadrer la valeur de x entre deux entiers, si le tableau contient $n = 10000$ éléments.
2. Mesurer la durée d'exécution de la fonction et la comparer à celle de la recherche classique.

À retenir

La complexité temporelle de la recherche dichotomique est :

$$\log_2 n = x$$

Christophe Viroulaud