Pas même un seul laissez-passer d'une journée, quand nous n'avons pas à chercher quelque chose dans notre vie quotidienne, clés de voiture, livres, stylo, chargeur de téléphone portable, etc. Il en va de même pour la vie d'un ordinateur, il y a tellement de données qui y sont stockées que chaque fois qu'un utilisateur demande des données, l'ordinateur doit rechercher dans sa mémoire pour rechercher les données et les mettre à la disposition de l'utilisateur.

**Les algorithmes de recherche** sont conçus pour vérifier un élément ou récupérer un élément à partir de n'importe quelle structure de données où il est stocké. Selon le type d'opération de recherche, ces algorithmes sont généralement classés en deux catégories :

1. **Recherche séquentielle** : dans ce cas, la liste ou le tableau est parcouru de manière séquentielle et chaque élément est vérifié. Par exemple : **recherche linéaire.**
2. **Recherche par intervalles**: ces algorithmes sont spécialement conçus pour effectuer des recherches dans des **structures de données triées**. Ces types d'algorithmes de recherche sont beaucoup plus efficaces que la recherche linéaire car ils ciblent à plusieurs reprises le centre de la structure de recherche et divisent l'espace de recherche en deux. Par exemple : **recherche binaire**.

**Recherche linéaire**

La recherche linéaire est un algorithme de recherche très basique et simple. Dans la recherche linéaire, nous recherchons un élément ou une valeur dans un tableau donné en parcourant le tableau depuis le début, jusqu'à ce que l'élément ou la valeur souhaitée soit trouvée.

Il compare l'élément à rechercher avec tous les éléments présents dans le tableau et lorsque l'élément est mis en correspondance avec succès, il renvoie l'index de l'élément dans le tableau, sinon il renvoie -1.

**La recherche linéaire est appliquée sur des listes non triées ou non ordonnées, lorsqu'il y a moins d'éléments dans une liste.**

Caractéristiques de l'algorithme de recherche linéaire

1. Il est utilisé pour une petite liste d'éléments non triés et non ordonnés.
2. Le temps dépend linéairement du nombre d'éléments, ce qui n'est pas mauvais, mais pas si bon non plus.
3. Il a une mise en œuvre très simple.

Implementation du code

FUNCTION linear\_search(tab : ARRAY\_OF INTEGER, elt : INTEGER) : INTEGER

VAR j : INTEGER;

BEGIN

j := 0;

WHILE (j< tab.length) DO

IF (tab[j] = elt) THEN

RETURN j; // element is found let's break the loop and return the index

END\_IF

j := j+1; // update the index

END\_WHILE

// we reached the end of array without finding the element

RETURN -1 ;// means that we did not find the element

END

**Recherche binaire**

**La recherche binaire** est utilisée avec un tableau ou **une liste triée**. Dans la recherche binaire, nous suivons les étapes suivantes :

1. Nous commençons par comparer l'élément à rechercher avec l'élément au milieu de la liste/tableau.
2. Si nous obtenons une correspondance, nous renvoyons l'index de l'élément du milieu.
3. Si nous n'obtenons pas de correspondance, nous vérifions si l'élément à rechercher a une valeur inférieure ou supérieure à celle de l'élément du milieu.
4. Si l'élément/le nombre à rechercher a une valeur supérieure au nombre du milieu, alors nous sélectionnons les éléments sur le côté droit de l'élément du milieu (comme la liste/le tableau est trié, donc sur la droite, nous aurons tous les nombres supérieurs au nombre du milieu), et recommencez à partir de l'étape 1.
5. Si l'élément/numéro à rechercher a une valeur inférieure à celle du numéro du milieu, nous sélectionnons les éléments du côté gauche de l'élément du milieu et recommençons à partir de l'étape 1.
6. La recherche binaire est utile lorsqu'il y a un grand nombre d'éléments dans un tableau et qu'ils sont triés. Donc, **une condition nécessaire pour que la recherche binaire fonctionne est que la liste/tableau doit être trié.**

Implementation du code

FUNCTION binary\_search(arr : ARRAY\_OF INTEGER) : INTEGER

VAR left, right, mid : INTEGER;

BEGIN

left := 0;

right := arr.length-1;

WHILE (left < right) DO

mid := left + (right - left)/2;

// check if x is present in the mid

IF (arr[mid] = x) THEN

RETURN mid;

END\_IF

// if x grater, ignore the left half

IF (arr[mid] < x) THEN

left := mid+1;

ELSE

// if x is smaller, ignore the right half

right := mid-1;

END\_IF

END\_WHILE

// if we reached here then the element is not present

RETURN -1 ;

END