Capítulo 8

Algoritmos fundamentales

Capítulo 8

Algoritmos fundamentales

Objetivo

Este capítulo tiene como objetivo brindar una comprensión profunda de los algoritmos fundamentales que son la base de la mayoría de las soluciones computacionales. Se explorará en detalle el funcionamiento de estos algoritmos, sus ventajas y desventajas, y su aplicación en diferentes escenarios. Se analizarán los diferentes tipos de algoritmos básicos de búsqueda y ordenamiento. A través de la explicación de diferentes situaciones se brindará una introducción a la complejidad algorítmica subyacente en cada uno.

Al finalizar este capítulo, estarás capacitado para utilizar los algoritmos fundamentales de forma efectiva en tus programas. Habrás adquirido una comprensión profunda de los dife-rentes tipos de algoritmos, su funcionamiento y una introducción práctica a la complejidad computacional. Además, estarás en condiciones de elegir el algoritmo adecuado para cada caso y diseñar soluciones eficientes a problemas diversos.

Algoritmos

En nuestra disciplina, específicamente en el área de programación, llamamos algoritmos "fundamentales" a aquellos algoritmos que son esenciales y se utilizan con frecuencia para resolver una gran variedad de problemas.

Estos algoritmos también son importantes porque su eficiencia y complejidad son un factor crítico en el rendimiento de los programas y su capacidad para manejar grandes cantidades de datos.

En este contexto diremos que la eficiencia se refiere a la capacidad de un algoritmo para realizar una tarea utilizando la menor cantidad de recursos posible, como el tiempo de ejecución y la cantidad de memoria. Un algoritmo eficiente es aquel que puede completar una tarea de manera rápida y utilizando además la mínima cantidad de recursos.

Cuando mencionemos la complejidad algorítmica, por ahora, diremos que se refiere a la cantidad de recursos que un algoritmo necesita para completar una tarea en función del tamaño de la entrada.

La complejidad se utiliza para medir la eficiencia de un algoritmo y predecir su comportamiento a medida que aumenta el tamaño de la entrada.

Entonces podemos advertir que la eficiencia de un algoritmo está directamente relacionada con su complejidad. Un algoritmo con baja complejidad generalmente será más eficiente que un algoritmo con alta complejidad.

La complejidad temporal mide la cantidad de tiempo que un algoritmo necesita para completar una tarea y se expresa en términos de unidades de tiempo, como por ejemplo segundos o inclusive en términos de operaciones.

La complejidad espacial mide la cantidad de memoria que un algoritmo necesita para completar una tarea y se expresa en términos de unidades de memoria, como bytes o kilobytes.

De acá hasta el final cuando hablemos de complejidad lo haremos refiriéndonos a la temporal, cuánto tiempo le cuesta, en función de la entrada, al algoritmo terminar.

En definitiva, parte de los algoritmos fundamentales que veremos son los que se relacionan con la búsqueda y el ordenamiento. Conocer estos algoritmos y entender cómo funcionan es fundamental para cualquier programador que desee escribir programas eficaces y sobre todo programas eficientes.

> La eficiencia es la capacidad de un algoritmo para utilizar de la mejor manera los recursos, como el tiempo de ejecución y la cantidad de memoria, para lograr el resultado. Teniendo en cuenta además que el algoritmo no pierda legibilidad y fundamentalmente pueda escalar.

Búsqueda

Los algoritmos de búsqueda son aquellos que permiten encontrar un elemento particular en una colección de elementos. Si bien existen varios tipos de búsqueda, dependiendo el problema y datos involucrados, nos centraremos en las más elementales: la búsqueda lineal y la búsqueda binaria.

[-] lineal

Para comprender cómo funciona la búsqueda lineal podemos imaginar que estamos buscando un libro en una estantería llena de libros.

Iniciamos desde el principio de la estantería, verificamos el primer libro leyendo el título y lo comparamos con el que buscamos. Si es el libro que buscábamos entonces finalizamos

la búsqueda, ahora si no lo es continuamos con el siguiente y volvemos a repetir el procedimiento anterior. La búsqueda la continuamos hasta el final o hasta que encontremos en el camino el libro.

Entonces, dado que ya entendemos la lógica de funcionamiento podemos decir que la búsqueda lineal comienza en el primer elemento de la colección y lo compara con el elemento que buscamos. Si no son iguales, se pasa al siguiente elemento y se repite la comparación. Este proceso continúa hasta que se encuentra el elemento o hasta que llegamos al final de la lista.

Podemos ver esto de forma gráfica. Buscamos en un vector de 6 elementos aquel cuyo valor sea 0.

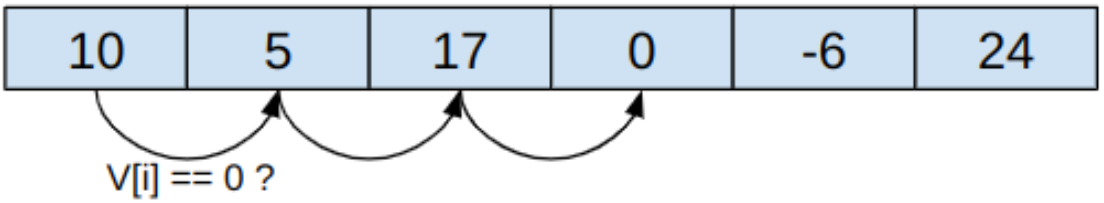

Fig.25 - búsqueda lineal o secuencial.

La búsqueda lineal es útil cuando la lista de elementos es relativamente pequeña, dado que si es grande podemos tardar mucho tiempo en determinar su existencia, o peor aún, su inexistencia.

Veamos como podría ser la implementación de una función que busca linealmente y retorna la posición del elemento si lo encuentra.

```
//
public static int busqueda(int[] vector, int e) {

   //
   for (int i=0; i < vector.length; i++){

       if(vector[i] == e){
           return i;
       }
   }
```