BÚSQUEDA Y ORDENAMIENTO

1. Algoritmos de Búsqueda
2. Algoritmos de Ordenamiento

SESIÓN **/03**

**INTRODUCCIÓN**

* La clase anterior revisamos las principales operaciones sobre un Arreglo unidimensional, conocido también como Vector.
* En la presente sesión, conoceremos los principales algoritmos para realizar operaciones de búsqueda y de ordenamiento sobre un Vector.
* Culminaremos analizando la eficiencia de estos algoritmos a fin de seleccionar la mejor opción.

1. **ALGORITMOS DE BÚSQUEDA**

Existen varios algoritmos de búsqueda, entre los más importantes tenemos los siguientes:

* 1. **Búsqueda Secuencial**

Este algoritmo consiste en recorrer el Vector, desde la posición cero, comparando cada elemento con el valor buscado. El algoritmo termina cuando se encuentra el valor buscado o cuando se llega al final del Vector (último elemento).

***EJEMPLO:***

***En el siguiente Vector, busca el número 70***



***El algoritmo empieza comparando el número buscado con el primer elemento (posición 0).***

***Como no son iguales, se compara el número buscado con el segundo elemento (posición 1).***

***Como no son iguales, se compara el número buscado con el tercer elemento (posición 2).***

***Como no son iguales, se compara el número buscado con el cuarto elemento (posición 3).***

***Como son iguales, se termina la búsqueda. Se realizaron 4 comparaciones.***

Eficiencia

La eficiencia se determina por el número de comparaciones a realizar. Para un Vector de tamaño **n**:

* Si el valor buscado se encuentra en la posición cero, entonces sólo se necesitará hacer una comparación. Esta situación representa **el mejor caso**.
* Si el valor buscado se encuentra en la posición **n-1** o si este no se encuentra, entonces se necesitará hacer **n** comparaciones. Ambas situaciones representan **el peor caso**.
  1. **Búsqueda Binaria**

Este algoritmo compara el valor buscado con el elemento de la posición central del Vector. Si no son iguales, se verifica si el valor buscado es menor al elemento de la posición central. De ser así, se buscará el valor en la parte izquierda del Vector; caso contrario, se lo buscará en la parte derecha del Vector. El algoritmo termina cuando se encuentra el valor buscado o cuando ya no quedan elementos de la posición central.

Para aplicar este algoritmo, los elementos del Vector deben estar ordenados.

***EJEMPLO:***

***En el siguiente Vector, busca el número 40***



***Se determina la posición central:*** 0 + 7 = 3 (sólo se toma la parte entera)

2

***Se compara el valor buscado con la posición 3.***

***Como no son iguales, se pregunta si el valor buscado es menor que el de la posición 3.***

***Como 40 no es menor a 24, entoces buscamos en la parte derecha del Vector.***

******

***Se determina la posición central:*** 4 + 7 = 5 (sólo se toma la parte entera)

2

***Se compara el valor buscado con la posición 5.***

***Como no son iguales, se pregunta si el valor buscado es menor que el de la posición 5.***

***Como 40 no es menor a 36, entoces buscamos en la parte derecha del Vector.***



***Se determina la posición central:*** 6 + 7 = 6 (sólo se toma la parte entera)

2

***Se compara el valor buscado con la posición 6.***

***Como no son iguales, se pregunta si el valor buscado es menor que el de la posición 6.***

***Como son iguales, se termina la búsqueda. Se realizaron 3 comparaciones.***

Eficiencia

La eficiencia se determina por el número de comparaciones a realizar. Para un Vector de tamaño **n**:

* Si el valor buscado se encuentra en la posición central, sólo se necesitará realizar una comparación. Esta situación representa **el mejor caso**.
* Si el valor buscado se encuentra en los extremos del Vector, o si este no se encuentra, entonces se necesitará realizar **log2n** comparaciones. Ambas situaciones representan **el peor caso**.
  1. **Búsqueda Hash**

Este algoritmo se basa en una función de trasformación o función hash, la cual convierte una clave dada en una dirección (índice) dentro del Vector.

Eficiencia

Este algoritmo aumenta la velocidad de búsqueda sin necesidad de tener los elementos ordenados. Además, el tiempo de búsqueda es independiente del tamaño del Vector.

1. **ALGORITMOS DE ORDENAMIENTO**

Existen varios algoritmos de ordenamiento, entre los más importantes tenemos los siguientes:

**2.1 Ordenamiento Burbuja**

Para implementar este Algoritmo, es necesario recorrer el Vector varias veces. El proceso de recorrer el Vector se llama rueda ó ronda. De esta manera, el ordenamiento se realiza en varias ruedas de comparaciones.



A

Rueda 1

Se compara el segundo elemento del Vector con los demás elementos. Cada vez que se encuentra un elemento menor al elemento de la posición 0, se intercambian sus posiciones.

……

A[n-1]

A[2]

A[1]

A[0]

Rueda 2

Se compara el segundo elemento del Vector con los demás elementos. Cada vez que se encuentra un elemento menor al elemento de la posición 1, se intercambian sus posiciones.

……

A[n-1]

A[3]

A[2]

A[1]

Rueda 3

Se compara el tercer elemento del Vector con los demás elementos. Cada vez que se encuentra un elemento menor al elemento de la posición 2, se intercambian sus posiciones.

A[2]

A[3]

A[4]

A[n-1]

……

Rueda n-1

Se compara el penúltimo elemento del Vector con el último elemento. Si el penúltimo elemento es mayor al último, se intercambian sus posiciones.

A[n-1]

A[n-2]

* 1. **Ordenamiento Quick Sort**

Es el algoritmo más veloz, consiste en lo siguiente: Se toma un elemento **x** del Vector. Se trata de ubicar a **x** en la posición correcta, de tal forma que los elementos que se encuentran a su izquierda sean menores y los elementos que se encuentran a su derecha sean mayores. Se repite el procedimiento para ambas partes del Vector.

* 1. **Ordenamiento Shell**

Es una versión mejorada del algoritmo de inserción directa, en el cual cada elemento se compara con los elementos que se encuentran en la parte izquierda del Vector. En el algoritmo Shell las comparaciones se efectuan con saltos de mayor tamaño pero con incrementos decrecientes; así, los elementos se ordenarán más rápido.

**CONCLUSIÓN**

En la sesión de hoy vimos que, para comparar la eficiencia entre los algoritmos de búsqueda, se analiza las situaciones que representan el mejor y el peor caso en cada algoritmo. En cuanto a la operación de ordenamiento, esta se puede realizar en forma ascendente o descendente; si no se especifica, se asume la primera forma.

En general, la eficiencia de los algoritmos de búsqueda y ordenamiento se mide por el número de comparaciones que se deben realizar.

**REFERENCIAS BIBLIOGRÁFICAS**

* Cairo, O.; Guardati, S. (2008). Estructuras de datos. 3ra. Edición. México D.F., México: McGraw Hill
* Instituto NIIT (2011). Data Structures and Algorithms. Student guide.