

**Instituto Tecnológico de Tijuana**

**Subdirección Académica**

**Departamento de Sistemas y Computación**

**Semestre agosto-diciembre 2018**

**Carrera:** Ing. En Sistemas Computacionales Serie SC1A

**Materia:** Estructura de Datos

**Unidad 4 –Algoritmos de Búsqueda**

**Alumno:**15211883 - Angeles Valadez Jonathan

**Profesor:** Ray Brunett Parra Galaviz

Siendo optimistas, una secretaria podría perder tan sólo uno o dos minutos para encontrar el archivo de uno de los clientes de la compañía para la cual trabaja, esto, asumiendo que los archivos estén perfectamente ordenados y catalogados. Ahora, con una computadora en su oficina, la búsqueda completa puede hacerse en tan sólo unos segundos y la secretaria no tiene que encargarse de volver a buscar el lugar correcto para almacenar el archivo, de eso se encargan los programadores que desarrollan la aplicación de la cual la secretaria toma ventaja. Este es tan sólo un ejemplo de la importancia de las operaciones de búsqueda en un computador, las cuales se realizan a todos los niveles y con infinidad de implementaciones distintas.

La búsqueda de un elemento dentro de un array es una de las operaciones más importantes en el procesamiento de la información, y permite la recuperación de datos previamente almacenados. El tipo de búsqueda se puede clasificar como interna o externa, según el lugar en el que esté almacenada la información (en memoria o en dispositivos externos). Todos los algoritmos de búsqueda tienen dos finalidades:

* Determinar si el elemento buscado se encuentra en el conjunto en el que se busca.
* Si el elemento está en el conjunto, hallar la posición en la que se encuentra.

**Búsqueda lineal (secuencial)**

Consiste en recorrer y examinar cada uno de los elementos del array hasta encontrar el o los elementos buscados, o hasta que se han mirado todos los elementos del array.

Este es el método de búsqueda más lento, pero si nuestra información se encuentra completamente desordenada es el único que nos podrá ayudar a encontrar el dato que buscamos. El siguiente algoritmo ilustra un esquema de implementación del algoritmo de búsqueda secuencial:  
  
for(i=j=0;i<N;i++)  
  if(array[i]==elemento)  
   {  
     solucion[j]=i;  
     j++;  
   }

Este algoritmo se puede optimizar cuando el array está ordenado, en cuyo caso la condición de salida cambiaría a:

for(i=j=0;array[i]<=elemento;i++)  
  
o cuando sólo interesa conocer la primera ocurrencia del elemento en el array:  
  
for(i=0;i<N;i++)  
  if(array[i]==elemento)  
    break;

En este último caso, cuando sólo interesa la primera posición, se puede utilizar un centinela, esto es, dar a la posición siguiente al último elemento de array el valor del elemento, para estar seguro de que se encuentra el elemento, y no tener que comprobar a cada paso si seguimos buscando dentro de los límites del array:

array[N]=elemento;  
for(i=0;;i++)  
  if(array[i]==elemento)  
    break;

Si al acabar el bucle, i vale N esto indica que no se encontró el elemento. El número medio de comparaciones que hay que hacer antes de encontrar el elemento buscado es de (N+1)/2.

**Búsqueda binaria (dicotómica)**

Si los elementos sobre los que se realiza la búsqueda están ordenados, entonces podemos utilizar un algoritmo de búsqueda mucho más rápido que el secuencial, la búsqueda binaria. El algoritmo consiste en reducir paulatinamente el ámbito de búsqueda a la mitad de los elementos, basándose en comparar el elemento a buscar con el elemento que se encuentra en la mitad del intervalo y en base a esta comparación:

* Si el elemento buscado es menor que el elemento medio, entonces sabemos que el elemento está en la mitad inferior de la tabla.
* Si es mayor es porque el elemento está en la mitad superior.
* Si es igual se finaliza con éxito la búsqueda ya que se ha encontrado el elemento.

Se puede aplicar tanto a datos en listas lineales (Vectores, Matrices, etc.) como en árboles binarios de búsqueda. Los prerrequisitos principales para la búsqueda binaria son:

* La lista debe estar ordenada en un orden especifíco de acuerdo al valor de la llave.
* Debe conocerse el número de registros.

La búsqueda binaria consiste en dividir el array por su elemento medio en dos subarrays más pequeños, y comparar el elemento con el del centro. Si coinciden, la búsqueda se termina. Si el elemento es menor, debe estar (si está) en el primer subarray, y si es mayor está en el segundo. Por ejemplo, para buscar el elemento 3 en el array {1,2,3,4,5,6,7,8,9} se realizarían los siguientes pasos:

Se toma el elemento central y se divide el array en dos:

{1,2,3,4}-5-{6,7,8,9}

Como el elemento buscado (3) es menor que el central (5), debe estar en el primer subarray: {1,2,3,4}

Se vuelve a dividir el array en dos:

{1}-2-{3,4}

Como el elemento buscado es mayor que el central, debe estar en el segundo subarray: {3,4}

Se vuelve a dividir en dos:

{}-3-{4}

Como el elemento buscado coincide con el central, lo hemos encontrado.

Si al final de la búsqueda todavía no lo hemos encontrado, y el subarray a dividir está vacío {}, el elemento no se encuentra en el array. La implementación sería:

int desde,hasta,medio,elemento,posicion; // desde y hasta indican los límites del array que se está mirando.

int array[N];

// Dar valor a elemento.

for(desde=0,hasta=N-1;desde<=hasta;)

{

if(desde==hasta) // si el array sólo tiene un elemento:

{

if(array[desde]==elemento) // si es la solución posición=desde; // darle el valor.

else // si no es el valor:

Posición=-1; // no está en el array.

break; // Salir del bucle.

}

medio=(desde+hasta)/2; // Divide el array en dos.

if(array[medio]==elemento) // Si coincide con el central:

{

posicion=medio; // ese es la solución

break; // y sale del bucle.

}

else

if(array[medio]>elemento) // si es menor:

hasta=medio-1; // elige el array de la izquierda.

else // y si es mayor:

desde=medio+1; // elige el array de la derecha.

}

**Bibliografía**

[En Línea]: <http://artemisa.unicauca.edu.co/~nediaz/EDDI/cap02.htm#ancla2_3>