Algoritmos sobre Búsqueda y Ordenamiento (Análisis y Diseño)

Dra. María de Luz Gasca Soto
Departamento de Matemáticas,
Facultad de Ciencias, UNAM.

17 de marzo de 2020

Luz Gasca S

Capítulo 1

El Problema de Búsqueda

En este capítulo presentamos el problema de búsqueda, el cual consiste, de manera general, en encontrar un elemento x en una colección de objetos, parcialmente ordenada[1]. Considerando que siempre se realizan búsquedas sobre conjuntos, los cuales pueden estar o no ordenados, es necesario tener métodos que tomen en cuenta las características del problema que se desea resolver para así obtener procesos eficientes, ya que no es lo mismo efectuar búsquedas sobre conjuntos ordenados y desordenados. Además, también hay que considerar el tamaño del conjunto. Para mostrar esto, pensemos en los siguientes ejemplos:

1. Al pagar con tarjeta, de crédito o débito, la terminal tiene acceso a una base de datos y realiza una búsqueda para autentificar la tarjeta, es decir verifica que exista una cuenta asociada a tal tarjeta.

2. Buscar en el disco duro de una computadora un directorio o archivo.

3. La búsqueda que se realiza en algunas bibliotecas pequeñas para localizar los datos de un libro sobre ficheros ordenados alfabéticamente.

4. Al buscar un libro por tema en una biblioteca automatizada el sistema tendrá que filtrar la información de la base de datos que forma el catálogo para mostrar los libros que contienen el tema dado.

5. Al buscar, el ayudante, la tarea de un alumno en *montón* de tareas un curso, no es necesario que el conjunto de tareas esté organizado ni por número de tarea ni por nombre del alumno.

En general, la búsqueda localiza un objeto en un conjunto dado, la búsqueda se realiza considerando un atributo clave, es decir un atributo que lo identifica de los demás de forma única denominado llave.

Los métodos que describiremos en las siguientes secciones, están enfocados principalmente en búsquedas sobre secuencias ordenadas y se presenta un método para secuencias no necesariamente ordenadas.

[1]Es necesario que los elementos en la colección puedan ser comparados entre ellos.

Por simplicidad, supondremos que los conjuntos y secuencias usadas para describir e ilustrar los métodos son de tipo entero. Si los elementos no son enteros, siempre es posible construir una función biyectiva para asociar cada objeto del conjunto con un entero.

El resultado de la búsqueda dependerá de los requerimiento específicos del problema; por ejemplo, podría:

1. indicar si el dato está o no en el conjunto;
2. indicar en qué posición se encuentra el dato, si está, o en qué posición se encontraría, si no está en el conjunto;
3. regresar el primer dato, o al primero que se encuentre, que satisfaga las condiciones de la búsqueda;
4. regresar a todos los datos que satisfagan una propiedad dada.

Antes de iniciar con el tema daremos algúnos conceptos que usaremos, durante el desarrollo de este trabajo.

Objeto: Un objeto es la representación detallada, concreta y particular de un algo, que le permite ser identificado de otros. Para nuestro propósito a esta representación la llamaremos de forma indistinta "elemento u objeto".

Conjunto: Es una colección de objetos del mismo tipo, que no admite elementos repetidos. Consideraremos conjuntos para los cuales existe una relación de orden.

Secuencia: Es una sucesión de objetos del mismo tipo donde los objetos pueden o no estar repetidos, para los cuales existe una relación de orden.

Espacio de búsqueda: conjunto o secuencia de datos en la cual se realizará la búsqueda.

Un planteamiento más formal del problema es:

Problema de Búsqueda. Sea $S = \{s_1, s_2, s_3, \ldots, s_n\}$ una secuencia de números enteros, no necesariamente ordenada, no vacía, finita y de tamaño n.
Determinar se existe z en S, tal que $z = s_i$, para algún $i = 1, 2, 3 \ldots, n$.

A continuación presentamos cuatro técnicas empleadas para resolver este problema: La búsqueda secuencial, la binaria, la exponencial y la búsqueda por interpolación. Para cada una de ellas se presenta la estrategia general, el análisis de complejidad, en el peor de los casos (para algunos el análisis del caso esperado) y, al menos, un pseudocódigo, que podría ser recursivo o iterativo o ambos.

1.1. Búsqueda Secuencial

La Búsqueda secuencial es también llamada búsqueda lineal, es el método de búsqueda más intuitivo y simple, por lo que no se requiere ningún orden sobre los datos de entrada.

Estrategia

Esta técnica revisa uno a uno los elementos de la secuencia S hasta encontrar el dato deseado o llegar al último elemento de S, lo cual indicaría que el dato no está en S.

Para ilustrar el método consideremos los ejemplos de la Figura 1.1, ambos incisos se busca al número 7 y el índice i señala la posición que está siendo revisada.

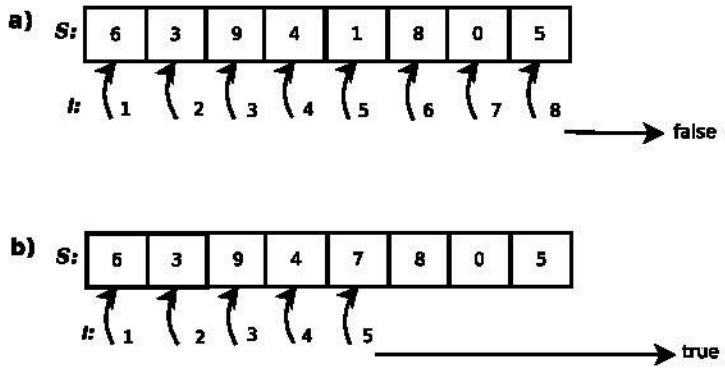

Figura 1.1: Ejemplos de búsqueda secuencial.

En la Figura 1.1**(a)** la secuencia es S={6,3,9,4,1,8,0,5}; aquí se tiene una búsqueda no exitosa, el proceso regresa false. La búsqueda termina cuando se llega al último elemento. Para el inciso **(b)** la secuencia es S={6,3,9,4,7,8,0,5}; ahora búsqueda es exitosa, el proceso regresa true, y termina al localizar el elemento en la posición $i = 5$ de S.

Análisis de Complejidad

En este apartado estableceremos el desempeño computacional tanto en el peor caso como en el caso esperado. Para tal propósito supondremos que la búsqueda se realiza sobre una secuencia S no necesariamente ordenada de tamaño n, donde $S = \{s_1, s_2, \ldots . s_n\}$.

Análisis del Peor Caso

Para establecer el desempeño computacional en el peor de los casos, es necesario que observemos que hay dos posibles escenarios; en el primero se tiene la situación de que el elemento buscado se encuentre al final de la secuencia, el segundo se da cuando el elemento no se encuentra en ella.

Para ambos casos se realizarán, necesariamente, n comparaciones por lo que el desempeño es exactamente el tamaño del ejemplar. Por tanto, en el peor caso la búsqueda secuencial requiere tiempo $O(n)$.

Análisis del Caso Esperado

En este caso también tenemos dos escenarios, en el primero la búsqueda siempre es exitosa, mientras que en el segundo consideramos que no lo es.

Para comenzar el análisis suponemos que la secuencia S ha sido obtenida bajo una distribución uniforme, entonces si el elemento sí está en S hay n localidades posibles para él, todas tienen la misma probabilidad de contenerlo, de esta forma la probabilidad de que el elemento buscado esté en la posición i, con $1 \leq i \leq n$, es de $1/n$. De manera más formal, sea x el elemento a buscar, entonces la probabilidad de que x esté en la posición i, denotado por $P_i[x]$, es: $P_i[x] = 1/n$.

Retomando la forma como el método resuelve el problema podemos observar que el número de comparaciones realizadas para ubicar al elemento, depende de su localización, por ello este número es simplemente su posición en la secuencia, de tal forma que si el elemento está en el primer lugar se realizará una comparación, si está en el segundo se efectuarán dos y así sucesivamente.

Entonces, podemos calcular el promedio de las comparaciones que esperamos realizar sumando el número de comparaciones efectuadas para cada localidad y multiplicando éstas por la probabilidad de que el elemento se encuentre en ella, esto es denotado por: $E\left[P_i\left[x\right]\right] = E\left[x\right]$, en específico:

$$E[x] = \sum_{i=1}^{n} i \cdot (P_i[x]) = \sum_{i=1}^{n} i \cdot \left(\frac{1}{n}\right) = \frac{1}{n} \cdot \sum_{i=1}^{n} i = \frac{1}{n} \cdot \frac{n(n+1)}{2} \in O(n).$$

Si incluimos la posibilidad de que el elemento no esté en la secuencia, tendremos ahora que considerar $(n+1)$ localidades por revisar y los $(n+1)$ posibles lugares son igualmente probables, entonces el número esperado de comparaciones ahora es:

$$E[x] = \frac{n+2}{2} = \frac{n+1}{2} + \frac{1}{2}.$$

Nótese que incluir la posibilidad de que el elemento pueda no estar en la secuencia sólo incrementa el caso esperado en $1/2$, cuando comparamos esta cantidad con el tamaño de la secuencia, esta cantidad es no significativa por lo que podemos concretarnos al valor $(n+1)/2$. Así, el número de comparaciones realizadas, en el caso esperado, por la búsqueda secuencial es lineal, es decir $O(n)$.

Este método de búsqueda no requiere ningún orden en los datos de entrada. Se puede usar en secuencias ordenadas, en este caso se recomienda modificar la condición de salida para terminar antes de recorrer toda la secuencia, aunque esto no influye en su desempeño global.

Algoritmo

A continuación se da el pseudocódigo para el algoritmo de búsqueda secuencial, presentado en el Listado 1. Suponemos que la secuencia S está contenida en un arreglo X, el cual no tiene datos repetidos.

Listado 1 Búsqueda Secuencial

```
// PreC: X es un arreglo no necesariamente ordenado, no vacio y finito de enteros
// PostC: X no sufre cambios;
//       regresa la posicion i, si el elemento z es encontrado;
//       regresa -1 si z no se encuentra en el conjunto.

int BSecuencial(int z; data_array  X) {

    int i;                          // indice de acceso al arreglo.
    i = 1;                          // inicializacion del indice.
    while (i < X.lenght) do {       // recorre el arreglo con el indice
         if (X[i]== z)
            return i;               // busqueda exitosa, regresa el indice
         else i=i+1
    }
    return -1;                      // busqueda no exitosa,
                                    // regresa una posicion no existente.
    }
```

Como se puede observar en el pseudocódigo se hace uso del índice i para recorrer el arreglo en el ciclo while, cuando la llave es encontrada (en caso de estar en el arreglo) se regresa i, la posición actual, lo que se interpreta como una búsqueda exitosa, ya que se obtiene la localidad en la que se encuentra el elemento buscado.

En caso de que se llegue al final del arreglo (condición en while) y no se encuentre el elemento, el algoritmo regresa -1, este valor fue escogido en caso de fracaso, pues estamos suponiendo que el índice inferior del arreglo es positivo.

1.2. Búsqueda Binaria

Este método pertenece al grupo de técnicas basadas en la estrategia de divide y vencerás. La Búsqueda Binaria requiere que el espacio de búsqueda esté ordenado. Así el nuevo planteamiento del problema es el siguiente:

Problema de Búsqueda. Sea $S = \{s_1, s_2, \ldots, s_n\}$ una secuencia ordenada y finita de números enteros, de tamaño n. Determinar si existe z en S tal que $z = s_i$, para algún i, $i = 1, 2, \ldots, n$.

Sin pérdida de generalidad, supondremos que la secuencia S está ordenada de forma ascendente y tiene longitud n_S. Denotaremos con m al elemento que divide a la secuencia en dos subsecuencias de tamaño similar y con z al elemento a buscar.

Estrategia

La estrategia empleada por esta técnica es más fácil de comprender si la vemos de manera recursiva, ésta consiste en dividir a la secuencia original en dos secuencias más pequeñas, de tamaño similar, donde el elemento que las divide m es comparado con z; en caso de ser igual, se considera una búsqueda exitosa y se da por concluida. En caso contrario, dependiendo de cómo sea la relación de orden entre el elemento m y el elemento buscado z, sin dejar de considerar, el tamaño de la secuencia S se tienen las siguientes situaciones:

Si $(m < z)$ **y** $(n_S > 1)$ se considera a la subsecuencia a la derecha del elemento m, como el nuevo espacio de búsqueda y se emplea nuevamente búsqueda binaria sobre esta subsecuencia.

Si $(m > z)$ **y** $(n_S > 1)$ se considera a la secuencia a la izquierda de m como el nuevo espacio de búsqueda y se aplica el método de la búsqueda binaria sobre ella.

Si ($(m < z)$ **y** $(n_S = 1)$) **o si** ($(m > z)$ **y** $(n_S = 1)$) se considera una búsqueda no exitosa y se da por terminada.

Para ilustrar el método, consideremos el ejemplo mostrado en la Figura 1.2(a). Se tiene la secuencia $S = \{2, 3, 5, 7, 9, 10, 13, 15, 16, 18\}$ y se busca al número $z = 13$.

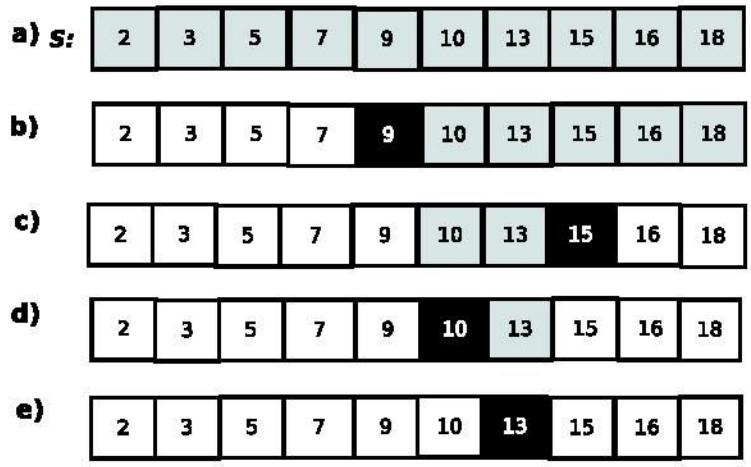

Figura 1.2: Ejemplo de búsqueda binaria

Como podemos ver en el inciso **(b)**, de la Figura 1.2, se tiene que el número de datos es $n_S = 10$; el elemento en la mitad, es $m = 9$ y está enmarcado con color negro; $S_1 = \{2, 3, 5, 7\}$ es la subsecuencia izquierda de m y $S_2 = \{10, 13, 15, 16, 18\}$ es la subsecuencia derecha. Ya que $z = 13 > 9 = m$ se toma como nuevo espacio de búsqueda a S_2, marcada en gris. Aplicamos la búsqueda binaria sobre ella, obteniendo $m = 15,\ n_S = 5$. Las nuevas

secuencias izquierda y derecha son $S_1 = \{10, 13\}$ y $S_2 = \{16, 18\}$, respectivamente, inciso **(c)**. Se tiene que $m = 15 \neq 13 = z$ y $z = 13 < 15 = m$, entonces se toma a la secuencia izquierda (marcada en gris) y se le aplica búsqueda binaria. Ahora, $m = 10$ y $n_S = 2$, dado que el espacio de búsqueda tiene solo dos elementos, se considera $S_1 = \{10\}$ y $S_2 = \{13\}$, como $m = 10 < 13 = z$, se aplica la búsqueda binaria sobre S_2, inciso **(d)**. Tenemos que $m = 13$, $n_S = 1$, ya que $m = 13$, es el número buscado, entonces se termina la búsqueda binaria con éxito.

Algo que es importante observar, con la misma secuencia S, es que si el número buscado hubiera sido el 12, se realizarían los mismos pasos pero al final como $m = 13 \neq 12 = z$ y $n_S = 1$ se terminaría con una búsqueda no exitosa, lo cual nos indica que al menos para este caso se realizán el mismo número de comparaciones para la búsqueda exitosa y la no exitosa, esto nos proporciona una pista del desempeño computacional de este algoritmo.

Complejidad Computacional

Calcularemos el desempeño computacional de este método de búsqueda tanto en el peor caso como en el caso promedio. Sin pérdida de generalidad y para facilitar el análisis consideraremos que S es de tamaño 2^k para $k \in \mathbb{Z}$ y $k \geq 1$.

Análisis del peor caso

Dado que el método siempre reduce la secuencia a la mitad y que el tamaño de la secuencia es: $n = 2^k$, entonces para cada llamada recursiva el tamaño de la secuencia revisada se puede expresar como $n' = 2^{k-1}$, de aquí podemos decir que el tamaño mínimo se da cuando $k = 1$. Por tanto, el método es usado k veces, entonces al resolver la ecuación $n = 2^k$ tenemos que $k = \log_2 n$, por lo que se realizan $\log_2 n$ comparaciones. Por lo tanto, la búsqueda binaria toma tiempo $O(\log_2 n)$, en el peor caso.

Análisis del Caso Esperado

Para realizar el análisis del caso esperado se requiere suponer que la secuencia fue obtenida bajo una distribución uniforme. Se revisan dos posibles escenarios: cuando el elemento a localizar está en la secuencia y cuando el elemento no está.

En el primer escenario se tienen n posibles posiciones para z, número a buscar, cada posición tiene la misma probabilidad de contener a z y ésta es de $1/n$, lo que denotaremos por: $P_i[z] = 1/n$. Si consideramos el árbol binario que representa a este proceso de búsqueda vemos que se requiere una comparación para encontrar el elemento raíz, que en el caso de la búsqueda es el elemento en la mitad, m, dos comparaciones para cualquier nodo del primer nivel y así sucesivamente; en general, se requieren i comparaciones para cualquier nodo en el nivel i. Sabemos además que un árbol binario tiene 2^{i-1} nodos en el nivel i, entonces cuando $n = 2^k$ se tiene un árbol con k niveles.

Entonces, podemos determinar el número de comparaciones hechas para cada posible localidad, esto lo conseguimos considerando el nivel en el que se encuentra el árbol binario que representa la búsqueda. Una vez que conocemos este dato, es posible calcular el

número promedio de comparaciones que se espera realizar para cada posición y al multiplicar esta cantidad por la probabilidad de que el elemento esté en ella y sumarlas todas obtenemos el análisis del caso esperado. Esto lo podemos escribir como:

$$E[P_i[z]] \;=\; E[z] \;=\; \frac{1}{n}\cdot\sum_{i=1}^{k} i\cdot\left(2^{i-1}\right) \;=\; \left(\frac{1}{n}\right)\cdot\frac{1}{2}\sum_{i=1}^{k} i\cdot\left(2^{i}\right)$$

$$=\; \frac{1}{n}\cdot\left(\frac{1}{2}\right)\cdot\left[(k-1)\,2^{k+1}+2\right] \;=\; \frac{1}{n}\cdot\left[(k-1)\,2^{k}+1\right]$$

$$=\; \frac{1}{n}\cdot\left[k\left(2^{k}\right)-2^{k}+1\right] \;=\; \frac{\left[k\left(2^{k}\right)-\left(2^{k}-1\right)\right]}{n}$$

$$=\; \frac{k\left(2^{k}\right)-n}{n} \;=\; \frac{k\left(2^{k}\right)}{n}-1 \;=\; \frac{k\cdot n}{n}-1 \;=\; k-1.$$

Ya que $n=2^k$ entonces $k-1 \;\approx\; \log(n)-1$, por lo tanto $E[z]$ es $O(\log n)$.

Ahora bien, si consideramos el segundo escenario en el cual el número buscado no está en la secuencia, seguimos teniendo las n posibilidades del escenario anterior, pero además hay que agregar la $(n+1)$ generada a partir del hecho de que el elemento no está, las cuales son:

$$\begin{array}{l} z \;<\; s_1. \\ z \;>\; s_1, \;\text{ pero }\; z \;<\; s_2. \\ \qquad\vdots \\ z \;>\; s_{n-1}, \text{ pero } z \;<\; s_n. \\ z \;>\; s_n. \end{array}$$

Así, para este caso se tiene en total $(2n+1)$ posibilidades, las cuales son equiprobables, por lo tanto, la probabilidad de que el elemento esté en una posición es $P_i\,[z] = 1/(2n+1)$. Para calcular el número de comparaciones esperadas hay que tomar en cuenta que para cada una de las $(n+1)$ nuevas localidades que agregamos se realizan k comparaciones, por tanto tenemos $(n+1)k$ comparaciones adicionales. Al poner estos datos juntos obtenemos que el número esperado de comparaciones a realizar, formalmente:

$$E[P_i] \;=\; E[z] \;=\; \frac{1}{2n+1}\left[\left(\sum_{i=1}^{k} i\left(2^{i-1}\right)\right)+(n+1)k\right]$$

$$=\; \left(\frac{1}{2n+1}\;\left(\frac{1}{2}\right)\sum_{i=1}^{k} i\left(2^{i}\right)\right)+\frac{(n+1)\,k}{2n+1}$$

$$=\; \left(\frac{1}{2n+1}\;\left(\frac{1}{2}\right)\left[(k-1)\,2^{k+1}\;+2\right]\right)\;+\frac{(n+1)\,k}{2n+1}.$$

Distribuyendo la suma y sustituyendo el valor de $n = 2^k$,

$$E[z] = \frac{\left[(k-1)\,2^k+1\right] + (n+1)\,k}{2n+1} = \frac{\left[(k-1)\,2^k+1\right] + \left(2^k+1\right)k}{2\,(2^k)+1}$$

$$= \frac{\left(k\left(2^k\right) - 2^k + 1\right) + \left(2^k\right)k + k}{2^{k+1}+1} = \frac{k\left(2^{k+1}\right) - 2^k + 1 + k}{2^{k+1}+1}$$

$$\approx \frac{k\left(2^{k+1}\right) - 2^k + k}{2^{k+1}} \approx \frac{k\left(2^{k+1}\right) - 2^k}{2^{k+1}} \approx k - \frac{1}{2}$$

Ya que $n = 2^k$ entonces $k - \frac{1}{2} \approx \log n - \frac{1}{2}$. Por lo tanto, la búsqueda binaria tiene desempeño computacional de orden $O(\log_2 n)$, en el caso esperado.

Algoritmo de Búsqueda Binaria

En esta subsección daremos el pseudocódigo para el algoritmo de búsqueda binaria, por cuestiones de simplicidad supondremos que los datos están contenidos en un arreglo y que no hay datos repetidos. Presentaremos dos versiones, una iterativa, Listado 2, y otra recursiva, Listado 3.

Listado 2 Búsqueda Binaria Iterativa

```
//PreC: X[a,b]  es un arreglo  ordenado, no vacio y finito de enteros
//PostC: X no sufre cambios;
//       regresa la posicion i, si el elemento z es encontrado;
//       regresa -1 si z no se encuentra en el conjunto.

int BBinaria(int z; data_array X){
    int max, mitad, min                    // variables temporales
       min = a;    max = b;
       while ( (max - min) > 1 ) do {
            mitad = (max + min) div 2;     // calcula mitad
            if (z < X[mitad])              // busca a la izquierda
              then max = mitad;
            else
              min = mitad;                 // busca a la derecha
       }
       if (X[mitad] == z)
         return mitad;                     // busqueda exitosa
       if (X[min] == z)
         return min;                       // exito
       else
         return -1;                        // sin exito
}//end BBinaria.
```

A pesar de que la explicación de la estrategia se realizó recursivamente, este pseudocódigo se implementa de manera iterativa para mostrar cómo se ve la técnica con otro enfoque, en este caso el ciclo while es el equivalente a determinar el tamaño de la secuencia a emplear, con el garantizamos que la última secuencia a usar es de un elemento.

Como también se puede apreciar, la selección de la subsecuencia a emplear se determina al actualizar los valores ya sea para max, índice superior del subarreglo, o para min, índice inferior del arreglo actual. El valor de -1 se escoge como valor a regresar en caso de búsqueda sea no exitosa, debido a que consideramos a los índices del arreglo son estrictamente positivos y el método regresa la posición donde se encuentra el elemento si se encuentra en el arreglo.

Listado 3 Búsqueda Binaria Recursiva

```
//PreC: X[a,b]  es un arreglo  ordenado, no vacio y finito de enteros
//       a <= b+1  el arreglo tiene al menos un elemento
//PostC: X no sufre cambios;
//       regresa true, si el elemento z es encontrado;
//       regresa false, si z no esta en el conjunto.
//       Encontro = (z esta_en X)

// Proceso principal
 BusquedaB(array X; int a,b; int z){
     Encontro = BB (X, a, b, z);
 }

// Proceso auxiliar
boolean BB(array X; int a,b; int z) {
    int  mid;                                  // posicion mitad

     if ( a > b )                              // arreglo vacio
       return false;                           // sin exito
       else
         mid := (a+b) div 2;                   // calcula la mitad
           if ( z = X[mid])
              return true;                     // elemento encontrado
           else if (z < X[mid])                // busca a la izquierda
              return BB(X, a, mid-1, z);
            else                               // busca a la derecha
               return BB(X, mid+1, b, z);
 }
 // end BB
```