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

1.3. Aplicaciones de Búsqueda Binaria

En esta sección revisaremos tres problemas que se resuelven aplicando la estrategia de la Búsqueda Binaria.

Secuencia cíclica

Una secuencia $S = x_1, x_2, ..., x_n$ se dice que está cíclicamente ordenada si el número más pequeño en la secuencia es el elemento x_i para algún i desconocido y la secuencia S' está ordenada en orden creciente, con $S' = x_i, x_{i+1}, ...x_n, x_1, x_2, ..., x_{i-1}$.

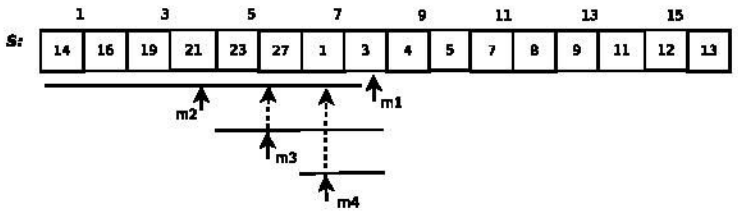
Por ejemplo, $S = \{14, 16, 19, 21, 23, 27, 1, 3, 4, 5, 7, 8, 9, 11, 12, 13\}$ es una secuencia cíclicamente ordenada, con $i = 7$.

Problema : Dada una lista cíclicamente ordenada, encontrar la posición del mínimo elemento en la lista. Supondremos, por simplicidad, que tal posición es única.

Estrategia

Para encontrar el mínimo elemento x_i en la secuencia, usamos la estrategia de la búsqueda binaria para eliminar la mitad de la secuencia en una comparación. Tomamos cualesquiera dos números x_k y x_m tales que $k < m$. Si $x_k < x_m$ entonces el índice i no puede estar en el rango $k < i \leq m$, ya que x_i es el mínimo en la secuencia. Nótese que no podemos excluir x_k. Por otro lado, si $x_k > x_m$ entonces el índice i debe estar en el rango $k < i \leq m$ ya que el orden fue cambiado en algún lugar de ese rango. De esta forma, con una comparación podemos eliminar varios elementos. Eligiendo k y m apropiadamente es posible encontrar al índice i en tiempo $O(\log n)$ comparaciones. El algoritmo está dado en el Listado 4.

Ejemplo



Consideremos la secuencia $S = \{14, 16, 19, 21, 23, 27, 1, 3, 4, 5, 7, 8, 9, 11, 12, 13\}$.
Aplicamos Encuentra_Indice(1,16). Tenemos que $mitad = 8$ y $X[8] = 3 < 13 = X[16]$.
Ejecutamos Encuentra_Indice(1,8). Ahora, $mitad = 4$ y $X[4] = 21 > 3 = X[8]$.
Llamamos a Encuentra_Indice(5,8). Tenemos $mitad = 6$ y $X[6] = 27 > 3 = X[8]$.
Aplicamos Encuentra_Indice(7,8). Tenemos que $mitad = 7$ y $X[7] = 1 < 3 = X[8]$.
Ejecutamos Encuentra_Indice(7,7). Ahora, i_izq= i_der y el algoritmo regresa al 7.

Listado 4 Búsqueda Binaria Cíclica

```
//PreC: X[1,n]  es una secuencia ciclica, no vacia y finita de enteros
//PostC: X no sufre cambios;
//       regresa la posicion p, indice del menor elemento

BBC (array X; int n){
   p = EncuentraIndice(1, n);
   return p
}

int EncuentraIndice(int i_izq, i_der ) {
  if  ( i_izq == i_der  )       return   i_izq;
     else{
        mitad = ( i_izq + i_der ) div 2;
        if   ( X[mitad] < X[i_der] )
           Encuentra_Indice ( i_izq, mitad );
        else
           Encuentra_Indice ( mitad+1, i_der )
     }
}//end Encuentra...
```

Índice Especial

En el siguiente problema de búsqueda la llave no está dada; en vez de ello buscamos un índice que satisfaga una propiedad especial.

Problema : Dada una secuencia ordenada de enteros distintos $a_1, a_2, ..., a_n$, determinar si existe un índice i tal que $a_i = i$.

Por ejemplo, para $S = \{-5, -4, -2, 0, 1, 3, 5, 7, 8, 9, 11, 13, 14, 15, 16\}$, $i = 11$.

Estrategia

La búsqueda binaria tradicional no es aplicable aquí, ya que el valor del elemento buscado no está dado. Sin embargo, la propiedad que buscamos es adaptable al principio de la búsqueda binaria. Para facilitar los cálculos, asumiremos que n, el tamaño del arreglo, es un número par.

Considere el valor de elemento en la posición $n/2, a_{n/2}$. Si este valor es exactamente $n/2$, hemos encontrado el índice deseado. En otro caso, si el valor es menor que $n/2$ entonces todos los números son distintos, el valor de $a_{n/2-1}$ es menor que $n/2 - 1$ y así. Ningún número en la mitad de la secuencia puede satisfacer la propiedad, pero podemos seguir buscando en la segunda mitad. Para la respuesta a "más grande que"se satisface un argumento similar. El algoritmo es dado en el Listado 5.

Listado 5 Búsqueda Binaria Índice Especial

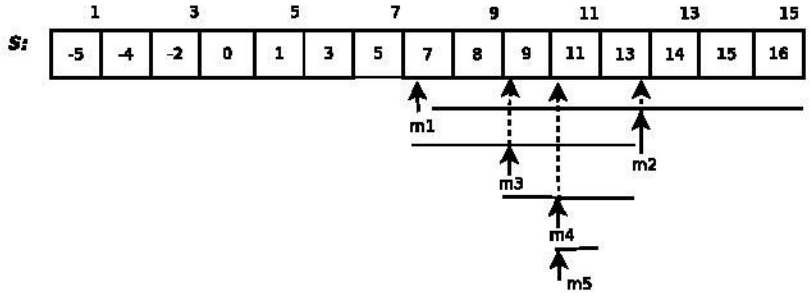
```
//PreC: A[1,n]  es un arreglo  ordenado, no vacio y finito de enteros.
//PostC: A no sufre cambios;
//       regresa la posicion p tal que A[p]=p


BB_IE(array X; int n){
   p = BusquedaEspecial(1, n);
   return p;
}

int Busqueda_Especial (int i_izq, i_der) {
  if   ( i_izq == i_der  )
     if   ( A[i_izq] = i_izq )  return  i_izq;
       else   return 0                          // busqueda  no exitosa
  else{                                         // continua la busqueda
        mitad = (i_izq + i_der) div 2;
        if  ( A [mitad] < mitad  )
           Busqueda_Especial ( mitad + 1, i_der )
        else
            Busqueda_Especial ( i_izq, mitad )
   } // end else
}// end Busqueda_E
```

Ejemplo

Sea secuencia $S = \{-5, -4, -2, 0, 1, 3, 5, 7, 8, 9, 11, 13, 14, 15, 16\}$. Aplicamos el proceso IndiceEspecial(1,15); es decir, nos quedamos con la parte izquierda, tenemos mitad=8, comparamos $A[8] = 7 < 8$ =mitad, entonces llamamos a IndiceEspecial(9,15); nótese que nos quedamos con la parte derecha. Ahora, mitad=12 y ($A[12] = 13 \not< 12$ =mitad), entonces ejecutamos IndiceEspecial(9,12). Ahora, mitad=10 y ($A[10] = 9 < 10$ =mitad), entonces aplicamos IndiceEspecial(11,12). Tenemos, mitad=11, comparamos $A[11] = 1 \not< 11$ =mitad, entonces llamamos a IndiceEspecial(11,11), como (i_izq = i_der) y (A[i_izq]=i_izq), el proceso regresa al índice i_izq= 11 y termina. La siguiente figura muestra esquemáticamente este ejemplo.



Secuencias de tamaño desconocido

Algunas veces usamos un procedimiento como la búsqueda binaria para duplicar el espacio de búsqueda en vez de partirlo a la mitad.

Considere un problema normal de búsqueda, pero suponga que el tamaño de la secuencia no es conocido. No podemos dividir a la mitad la secuencia, pues no conocemos sus cotas ni su tamaño. Buscaremos entonces un elemento x_i mayor o igual que z, el elemento buscado. Si encontramos tal elemento, podemos aplicar búsqueda binaria en el rango de 1 a i. Primero comparamos z con x_i. Si $z \leq x_i$ entonces z puede solo ser igual a x_i. Asumimos por inducción que sabemos que $z > x_j$ para alguna $j \geq 1$. Si comparamos a z con x_{2j}, entonces estamos duplicando el espacio de búsqueda con una sola comparación. Si $z \leq x_{2j}$, entonces sabemos que $x_j < z \leq x_{2j}$ y podemos encontrar z en $O(\log j)$ comparaciones adicionales. Además, si i es el menor índice tal que $z \leq x_i$ entonces toma $O(\log i)$ comparaciones encontrar un x_j tal que $z \leq x_j$ y realiza otras $O(\log i)$ comparaciones encontrar al índice i.

El mismo algoritmo puede ser usado cuando el tamaño de la secuencia es conocida, pero quizá i no sea suficientemente grande. Tendíamos, en tal caso, una versión mejorada de la búsqueda binaria con un desempeño computacional de $O(\log i)$ en vez de $O(\log n)$. Sin embargo, hay un factor de 2 en el desempeño computacional del algoritmo, pues ejecutamos dos búsquedas binarias. Este algoritmo es mejor sólo cuando i es $O(\sqrt{n})$.

1.4. Búsqueda Exponencial

Esta técnica es ideal para secuencias ordenadas de gran tamaño y es considerada como una variante de búsqueda binaria, por lo cual también emplea la estrategia divide y vencerás, lo que se hace evidente al observar cómo la técnica resuelve el problema.

Al igual que en la sección anterior se pide como condición inicial que la secuencia esté ordenada. Supondremos, sin pérdida de generalidad, que la secuencia está ordenada de forma ascendente. El problema se replantea como sigue:

Problema de Búsqueda.- Sea $S = \{s_1, s_2, \ldots, s_n\}$ una secuencia finita y no vacía, de tamaño n, ordenada ascendentemente, de números reales.
Determinar si existe z en el conjunto S con $z = s_i$, para $i, i = 1, 2, \ldots, n$.

Estrategia

Este método de búsqueda se basa en la idea de acotar y comparar, esto lo consigue dividiendo a S en intervalos de la forma $[2^i, 2^{i+1}]$. Una vez que se tiene el primer intervalo se compara a $s_{2^{i+1}}$ con z, y dependiendo de cómo sea la relación de orden se tienen tres posibles casos, los cuales surgen porque la secuencia está ordenada ascendentemente.

1. Si $s_{2^{i+1}} = z$, se considera una búsqueda exitosa y se da por finalizada.
2. Si $s_{2^{i+1}} > z$, se procede a realizar búsqueda binaria en el intervalo $[2^i, 2^{i+1}]$ para determinar si el elemento está en la secuencia.

3. Si $s_{2^{i+1}} < z$, se actualiza el valor de i como $i = i + 1$, se construye el nuevo intervalo con la forma $[2^i, 2^{i+1}]$, para realizar nuevamente la comparación. Cuando el tamaño de S sea menor a 2^{i+2} el último intervalo sería $[2^i, n]$.

Para el primer intervalo se fija la potencia de 2^k que se utilizará como límite superior, está potencia debe de ser relativamente grande; generalmente se usa $k = 8$ o $k = 10$. Sin embargo, por razones didácticas, para nuestro ejemplo, emplearemos como primera potencia a $k = 5$, es decir, la primera cota será 2^5.

Como ejemplo ilustrativo consideremos la Figura 1.4, en la que tenemos los primeros 500 números pares, es decir: $S = \{2, 4, 6, 8, \ldots, 998, 1000\}$ y buscamos el número $z = 608$.

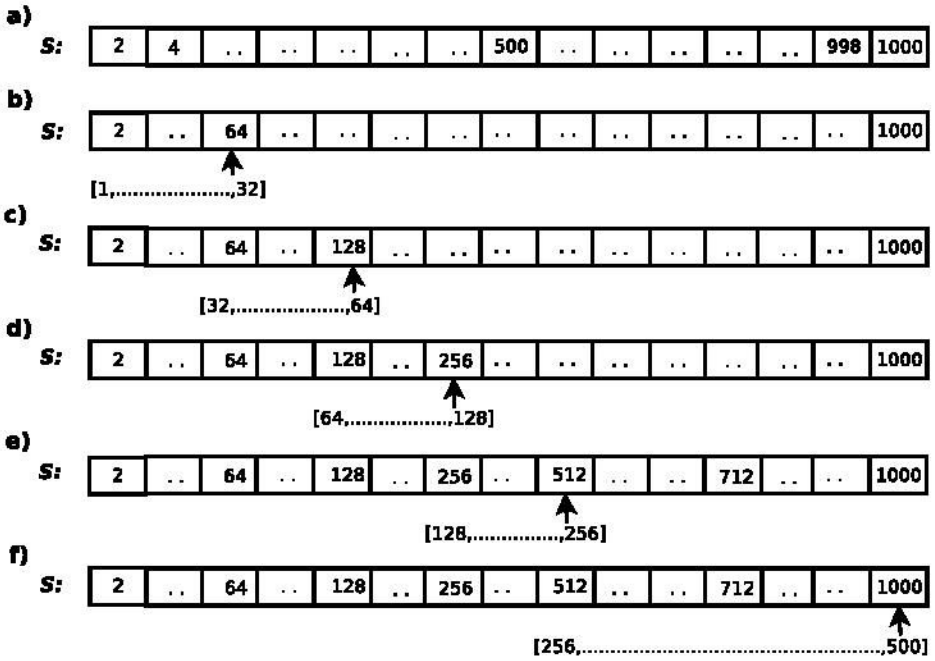


Figura 1.3: Ejemplo de búsqueda exponencial, I.

Se puede ver en los incisos del **(b)** al **(f)** que se realiza la partición en intervalos que son potencia de 2, así el primer intervalo va de la localidad 1 a la 32, es decir de 2^0 a 2^5, inciso **(b)**. Ahora, como $S[32] = 64$ se cumple con $64 < 608$, entonces se construye el siguiente intervalo $[2^5, 2^6] = [32, 64]$, inciso **(c)**. Nuevamente, el elemento en el límite superior del intervalo es menor al buscado por lo cual se construye el siguiente intervalo.

Dado que tenemos sólo 500 elementos en la secuencia el último intervalo construido contiene menos elementos que la potencia de 2 correspondiente, esté intervalo sería $[2^8, 2^9]$. Por tanto, el último intervalo debe ser $[2^8, n] = [256, 500]$.

Para nuestro ejemplo $z = 608$ es menor que 1000, entonces se inicia con búsqueda binaria, en el último intervalo, $[256, 500]$. La Figura 1.4, incisos del **(g)** al **(m)**, muestra el proceso.

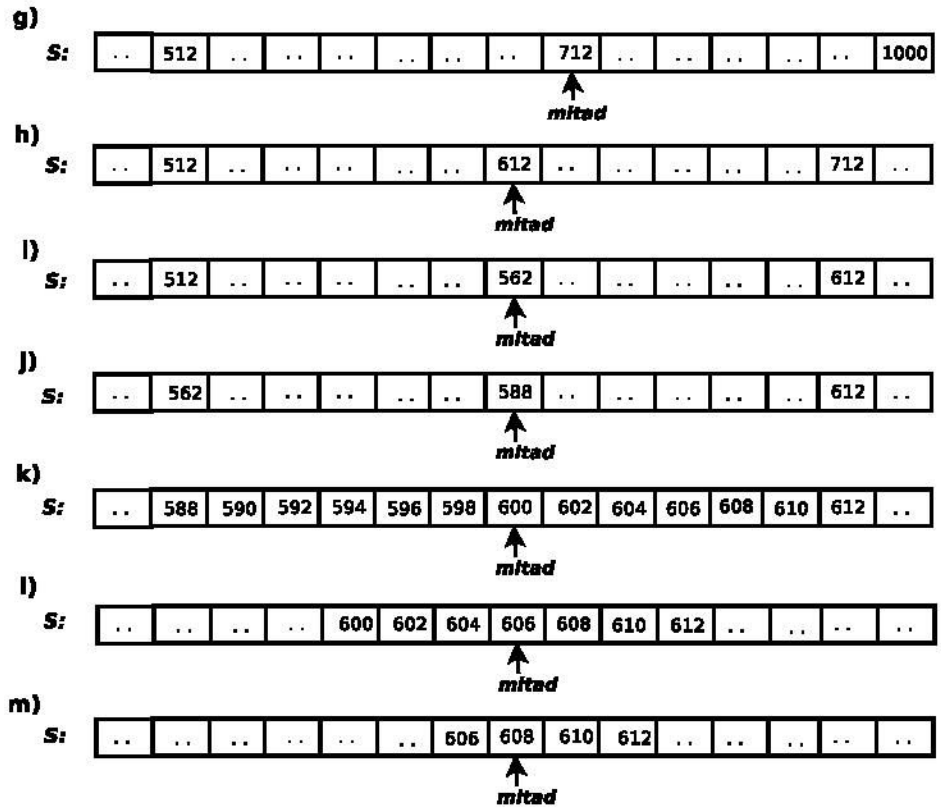


Figura 1.4: Ejemplo de búsqueda exponencial, II.

La búsqueda binaria en el intervalo ya no se ilustra detenidamente, sólo se hace énfasis en el elemento a la mitad debido a que esta técnica fue estudiada anteriormente.

Complejidad Computacional

Realizaremos el análisis del peor caso. Para facilitar los cálculos supondremos, sin pérdida de generalidad, que el tamaño de la secuencia es una potencia de dos, es decir $n = 2^k$, con $k \geq 8$ y la primera potencia es 2^8. El peor caso, para esta búsqueda, se da cuando el elemento buscado se encuentra en la último intervalo formado.

Nótese que para determinar que elemento buscado z se realizaron tantas comparaciones como intervalos, entonces es necesario determinar el número de intervalos creados. Para realizar esto, consideramos que cada intervalo es de la forma $[2^i, 2^{i+1}]$, como $n = 2^k$, entonces se forman $\log_2 n = k$ intervalos. Ahora bien, sobre el último intervalo $[2^{k-1}, 2^k]$, de tamaño 2^{k-1}, se realizaría una Búsqueda Binaria sobre un intervalo de de tamaño 2^{k-1}; por lo tanto, su tiempo de ejecución sería: $\log_2 2^{k-1} = (k-1) \approx \log n$. Por tanto, el desempeño computacional de la Búsqueda Exponencial, en el peor de los casos es de $O(\log_2 n)$.

Cuando el elemento z a buscar no está en la secuencia, este algoritmo acota el espacio de búsqueda y se queda con el intervalo donde estaría z. Para este escenario, en el peor caso, se llegaría al último intervalo y, por lo tanto, el desempeño computacional queda en $O(\log_2 n)$. Se puede demostrar que en caso promedio, la búsqueda exponencial requiere tiempo $O(\log \log n)$[7].

Algoritmo

Listado 6 Búsqueda Exponencial

```
//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 BExponencial(int z; array X) {

 int lim_inf, lim_sup, n;
   lim_inf = 1;                                  // limite inferior inicial
   lim_sup = pow(2,8);                           // limite superior inicial
   n = S.length;                                 // num. de elementos en S
   while (lim_sup <= n) do {
      if ((lim_sup = n) && ( z > X[lim_sup] ))
            return -1;                           // busqueda fallida
        else if ( X[lim_sup] = key )
            return lim_sup;                      // busqueda exitosa
         else if ( z < X[lim_sup] )
            return binary(z,lim_inf,lim_sup); // realiza busqueda binaria
          else {                                 // define el nuevo
            lim_inf := lim_sup;                  //   espacio de busqueda
            lim_sup := lim_sup*2;
          }
   }// end while

   if ( lim_sup > n )                            // aplica busqueda binaria
     return   binary(key,lim_inf,n);
}// end BExponencial
```

Para escribir el pseudocódigo de esta técnica se supone que la secuencia está contenida en un arreglo X y que no hay elementos repetidos.

Como se puede observar en el pseudocódigo del Listado 6, lo primero es determinar el probable espacio de búsqueda: [lim_inf , lim_sup]; hecho esto, se revisa la condición del ciclo while, el cual establecerá los siguientes intervalos.

En caso de entrar al ciclo, lo primero que se hace es revisar si lim_sup es igual a n, el tamaño de X, si es igual, se compara al elemento en la localidad lim_sup con el buscado, z, en caso de no ser el mismo se considera la búsqueda fallida y termina.

Listado 7 Proceso Binary

```
int procedure binary(int key; int min, max) {

   int mitad;
   while ( min < max ) do {
     mitad := (max+min)/2;
     if ( key=X[mitad] )  return mitad;
       else if ( X[mitad] > key ) min := mitad+1;
              else  max := mitad - 1;
   }//end while

   if ( min = max )
     if ( X[min] = key )   return min;
       else return -1;
}// end binary
```

En caso de que lim_sup $< n$, se tienen tres posibles casos:

1. Si X[lim_sup]$= z$, la búsqueda exitosa y se termina.
2. Si X[lim_sup]$< z$, entonces se inicia la búsqueda binaria en el intervalo [lim_inf , lim_sup].
3. Si X[lim_sup]$> z$, se establecen los nuevos valores para lim_inf y lim_sup.

Cuando se sale del ciclo y no se ha terminado la búsqueda o bien no se entró al ciclo, entonces se ejecuta la búsqueda binaria en el intervalo [lim_inf, n].

El código del procedimiento binary, presentado en el Listado 7, es esencialmente la búsqueda binaria. La única diferencia es que en binary se hace referencia al arreglo considerando los límites del arreglo.

Al observar de manera completa al pseudocódigo de búsqueda exponencial, se hace evidente que cuando se tiene una secuencia pequeña, éste se puede considerar como una búsqueda binaria, ya que solamente se emplearía el procedimiento binary. Para terminar, cabe mencionar que el valor de retorno -1, en caso de búsqueda fallida es válido para indicar que el elemento no está, pues estamos suponiendo que los índices del arreglo son positivos y el pseudocódigo presentado regresa la localidad donde se sí se encuentra el elemento z.

1.5. Búsqueda por Interpolación

Hasta ahora hemos estudiado técnicas que accesan linealmente a la secuencia, la dividen a la mitad o en intervalos. Sin embargo, si en el primer intento se quisiera iniciar la búsqueda a partir de una localidad cercana a la que podría contener al elemento deseado, se requiere de otro tipo de estrategia.

La idea de iniciar la búsqueda a partir de una posición que pensamos está cercana al elemento buscado, es la clave del método de búsqueda por interpolación y crea una posición candidata utilizando una función de interpolación lineal.

Esta técnica requiere que la secuencia de trabajo esté ordenada. El planteamiento del problema de búsqueda queda, ahora, como:

Problema de Búsqueda Sea $S = \{s_1, s_2, \ldots, s_n\}$ una secuencia de números, ordenada, finita y de tamaño n. Determinar, utilizando una función de interpolación, si existe el elemento z en S tal que $z = s_i$ con $i = 1, 2, \ldots, n$.

Estrategia

Para ver de forma intuitiva el funcionamiento de esta técnica consideremos el siguiente ejemplo: Cuando se quiere abrir un libro en cierta página, si conocemos el número total de páginas, abriremos éste dependiendo de qué número de página se quiera, así, si el libro tiene 100 páginas y se quiere la 50, se abrirá más o menos a la mitad; si se quiere la 25 se abrirá aproximadamente a un cuarto; y si se desea la 75 a tres cuartos. Es poco probable que al abrir el libro por primera vez se obtenga la página deseada, sin embargo es muy probable que se obtenga una cercana a ella y a partir de la cual se puede iniciar la búsqueda.

En el ejemplo se observa que dependiendo de qué página se quiera, es dónde se abre el libro; es decir, efectuamos una aproximación; en términos más formales diremos que se realiza una interpolación. La estrategia que emplea este método de búsqueda consiste en realizar interpolaciones sucesivas hasta que encontramos al elemento z o determinamos que éste no se encuentra. Nótese que al realizar cada interpolación también reducimos el espacio de búsqueda, ya que la siguiente búsqueda se realiza considerando el valor obtenido. De manera general, la búsqueda por interpolación consiste en:

1. Utilizar una función de interpolación lineal para determinar una posición p.
2. Comparar al elemento en la posición p con el dato buscado, z.
   a) Si son iguales, terminamos la búsqueda y la consideramos exitosa.
   b) En caso contrario, realizamos una nueva interpolación, en un espacio menor de búsqueda, y comparamos nuevamente.
3. Hacer el Paso 2, mientras se pueda seguir efectuando.
4. Si no es posible realizar el Paso 2 y el elemento buscado no está contenido en la posición p se termina la búsqueda y se considera fallida.

La función de interpolación lineal que se emplea es la siguiente:

$$pos = izq + \left\lfloor \frac{(z - S[izq])\ (der - izq)}{S[der] - S[izq]} \right\rfloor$$

Para que la interpolación sea efectiva los valores de las variables der e izq varían, dependiendo de la posición candidata obtenida, pos. Esto es, dependiendo de z y pos, se decide por el nuevo espacio de búsqueda, que puede ser el intervalo $[izq, pos]$ o $[pos, der]$. Consideremos un par de ejemplos para ilustrar el método.

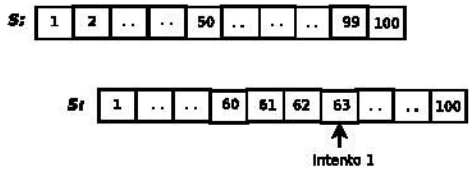


Figura 1.5: Ejemplo 1 de búsqueda por interpolación.

En la Figuras 1.5, la secuencia S está dada por los primeros cien enteros positivos: S={1,2,...,100} y se busca al 63. Aplicamos la búsqueda con los siguientes valores: $izq = 1, der = 100, z = 63, S[1] = 1$ y $S[100] = 100$, entonces,

$$pos = 1 + \left\lceil \frac{(63 - S[1])(100 - 1)}{S[100] - S[1]} \right\rceil = 1 + \left\lceil \frac{(63 - 1)(100 - 1)}{(100 - 1)} \right\rceil = 1 + 62 = 63.$$

Como se puede observar para este caso en la primera aplicación de la fórmula de interpolación da $p = 63$ que es donde se encuentra el elemento buscado y, por lo tanto, el proceso termina exitosamente.

En caso de buscarse un elemento que no se encuentre en la secuencia como sería el 101, la localidad por interpolación que se obtiene supera al número de elementos en la secuencia y, por lo tanto, se concluye que no está y se termina con la búsqueda.

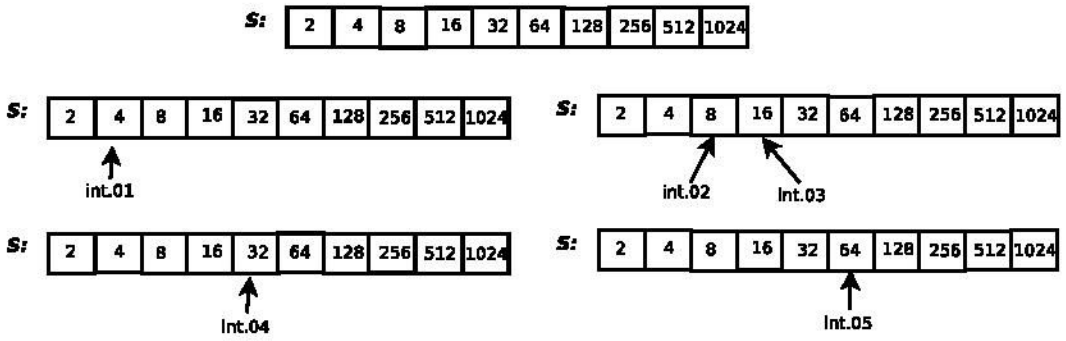


Figura 1.6: Ejemplo 2 de búsqueda por interpolación.

En la Figuras 1.6, ejemplo 2, la secuencia S está formada por las primeras diez potencias de 2: S={2, 4, 8, 16, 32, 64, 128, 256, 512, 1024} y se busca a $z = 64$. Aplicamos la búsqueda con los siguientes valores: $izq = 1, der = 10, z = 64$ y $S[1] = 2, S[10] = 1024$, entonces,

$$p_1 = 1 + \left\lceil \frac{(64 - 2)(10 - 1)}{(1024 - 2)} \right\rceil = 1 + \left\lceil \frac{(62)(9)}{1022} \right\rceil = 1 + \lceil 0{,}545 \rceil = 2.$$

Tenemos $S[2] = 4 \neq 64 = z$, entonces ahora $izq = 2, der = 10, z = 64, S[2] = 4$, así

$$p_2 = 2 + \left\lceil \frac{(64-4)(10-2)}{(1024-4)} \right\rceil = 2 + \left\lceil \frac{(60)(8)}{1020} \right\rceil = 2 + \lceil 0{,}470 \rceil = 3.$$

Revisamos: $S[3] = 8 \neq 64 = z$, el espacio de búsqueda queda $S[3.,10]$. Entonces los nuevos datos son: $izq = 3, der = 10, z = 64, S[3] = 8$, luego:

$$p_3 = 3 + \left\lceil \frac{(64-8)(10-3)}{(1024-8)} \right\rceil = 3 + \left\lceil \frac{(56)(7)}{1016} \right\rceil = 3 + \lceil 0{,}385 \rceil = 4.$$

Tenemos $S[4] = 16 \neq 64 = z$, entonces ahora $izq = 4, der = 10, z = 64, S[4] = 16$, así

$$p_4 = 4 + \left\lceil \frac{(64-16)(10-4)}{(1024-16)} \right\rceil = 4 + \left\lceil \frac{(46)(6)}{1008} \right\rceil = 4 + \lceil 0{,}285 \rceil = 5.$$

Revisamos: $S[5] = 32 \neq 64 = z$, el espacio de búsqueda queda $S[5.,10]$. Entonces los nuevos datos son: $izq = 5, der = 10, z = 64, S[5] = 32$, luego:

$$p_5 = 5 + \left\lceil \frac{(64-32)(10-5)}{(1024-32)} \right\rceil = 5 + \left\lceil \frac{(32)(5)}{992} \right\rceil = 5 + \lceil 0{,}161 \rceil = 6.$$

Finalmente, tenemos: $S[6] = 64 = z$ y la búsqueda ha terminado.

Para esta secuencia las posiciones obtenidas por interpolación avanzan de uno en uno, lo que se debe al hecho de que la distancia entre los datos no es constante y, por ello, la técnica supone que se está cerca del elemento buscado.

Los ejemplos mostrados dan una pista sobre el desempeño de este método y cómo influye la forma en la que están organizados los datos de entrada en él.

Análisis de Complejidad

El desempeño de este método depende tanto del tamaño de la secuencia como de la entrada, como se observó en los ejemplos. Por lo que, el método es muy eficiente cuando la secuencia de entrada consistente de elementos distribuidos en forma equidistante, como sería el caso de la numeración en las páginas de los libros.

En contraste cuando la secuencia está compuesta de elementos no distribuidos de manera equidistante, en el peor caso búsqueda por interpolación puede llegar a ser $O(n)$, lo cual indicaría que se comporta como una búsqueda lineal.

Manber [10] menciona lo siguiente en lo referente al caso promedio:

> "... se puede demostrar que el número promedio de comparaciones realizadas durante la búsqueda por interpolación, sobre todas las posibles secuencias, es $O(\log \log\ n)$; sin embargo, no se acostumbra hacer debido a que en la práctica no es mucho mejor que búsqueda binaria, por dos razones: la primera es, para secuencias de gran tamaño, es decir secuencias con n grande, $\log n$ es pequeño, pero $\log \log n$ no es mucho más pequeño y la segunda se refiere al hecho de que búsqueda por interpolación utiliza más operaciones ariméticas."

De lo anterior podemos concluir que en lo referente a complejidad en el peor caso búsqueda por interpolación tiene $O(n)$, mientras que para el caso promedio se tiene un $O(\log \log n)$, [7].

Algoritmo de Búsqueda por Interpolación

El pseudocódigo para búsqueda por interpolación es dado en el Listado 8, para implementarlo se supuso que la secuencia está contenida en un arreglo de n elementos, $X[1, n]$.

Listado 8 pseudocódigo Búsqueda por Interpolación

```
//  PreC: X[1,n] arreglo ordendo, finto y no vacio de enteros
// PostC: Regresa la posicion del elemento si esta en la secuencia,
//        Regresa -1 si no esta
//        X no sufre ningun cambio.
int BInterpolacion(int z; array X) {

   int izq, der, i;      // se definen las variables auxiliares.
   izq = 1;              // valor inicial del extremo izquierdo
   der = n;              // valor inicial del extremo derecho

   while (( X[der] >= z ) && ( z > X[izq] )) do {
                         // calcula la posicion a revisar
     i = izq+({(z-X[izq])(der-izq)} div {X[der]-X[izq]});

                        //  actualiza los valores para izq y der
     if ( z > A[i] )     izq  = i;
       else if ( z < X[i] )  der  = i-1;
              else  izq = i;
   }// end while

      if ( X[izq] = z ) return  izq      // exito
       else return -1                    // sin exito
}//end BInterpolacion
```

En el pseudocódigo, podemos observar que al inicio se asignan los valores iniciales para los índices extremos *izq* y *der* los cuales corresponden a la primera y última localidad en la secuencia respectivamente; la condición del ciclo while se cumple en el momento en que se llega a una localidad que cruce el valor del elemento buscado, con lo cual se asegura que éste termina, dentro del ciclo se calcula el valor de la posición a revisar i por medio de la función de interpolación, así mismo se actualizan los valores para los extremos *izq* o *der* según sea el caso considerando la relación de orden entre el elemento contenido en la localidad escogida por interpolación y el elemento buscado.

Finalmente, el valor que se regresa es la posición que contiene al elemento buscado, z, en caso de encontrarse en la secuencia, y en caso de no estar regresa -1, este valor fué escogido debido a que se considera que todos los índices del arreglo son positivos.