BÚSQUEDA

Algoritmos y Estructuras de Datos II

Lic. Ana María Company

INTRODUCCIÓN

- La recuperación de la información es una de las aplicaciones más importantes de las computadoras.
- Ejemplos de búsqueda:
  - Localizar nombre y apellido de un alumno
  - Localizar un número de teléfono en una agenda

- La búsqueda puede ser:
  - INTERNA → se realiza sobre datos que se encuentran en la memoria principal, por ejemplo en un arreglo.
  - EXTERNA → es cuando todos sus elementos se encuentran en memoria secundaria (archivos almacenados en dispositivos de cinta, disco, etc.)

BÚSQUEDA

- La operación de **búsqueda** de un elemento N en un conjunto de elementos consiste en :
  - Determinar si N pertenece al conjunto y, en ese caso indicar su posición en él.
  - Determinar si N no pertenece al conjunto.
- Existen diferentes algoritmos de búsqueda.
- El algoritmo elegido depende de la forma en que se encuentren organizados los datos:

ordenados/desordenados

BÚSQUEDA - MÉTODOS

- Los métodos más conocidos de búsqueda son:
  - Búsqueda secuencial o lineal
  - Búsqueda binaria
  - Búsqueda por transformación de claves (hash)

BÚSQUEDA SECUENCIAL

- Supongamos una lista de elementos almacenados en un vector.
- El método más sencillo de buscar un elemento en un vector es recorrer el vector desde el primer elemento al último.
  - Si se encuentra el elemento buscado, visualizar un mensaje similar a *"Elemento encontrado"*, en caso contrario, visualizar un mensaje similar a *"El elemento buscado no se encuentra en la lista"*.
- Es decir, la búsqueda secuencial compara cada elemento del vector con el valor buscado, hasta que éste se encuentra o se termina de leer el vector(o lista) completo.

BÚSQUEDA SECUENCIAL

- El recorrido del vector se realizará normalmente con estructuras repetitivas.
- La búsqueda secuencial es un algoritmo válido para un vector cualquiera sin necesidad de que esté ordenado.
- También se puede aplicar con muy pocas variaciones a otras estructuras secuenciales, como, por ejemplo, a los archivos.

EJEMPLO
BÚSQUEDA SECUENCIAL

```
70  int busquedaSecuencial( tVectorInt pVector, int elem ) {
71  /*  result se usará para guardar la posición del
72      valor encontrado ó -1 en caso contrario */
73      int result = -1; /
74      int i = 0;
75      while( ( pVector[i] != elem ) && ( i != MAX ) ){
76          i = i + 1;
77      }
78      if( pVector[i] == elem ) {
79      /*  significa que se ha encontrado el elemento elem,
80          entonces se devuelve la posición del elemento
81          en el vector */
82          result = i;
83      }
84      return result;
85  }
```

BÚSQUEDA SECUENCIAL ORDENADA

- El algoritmo de **búsqueda secuencial** puede ser optimizado si el vector V está ordenado (supongamos que de forma creciente).

- En este caso, la búsqueda secuencial desarrollada anteriormente es ineficiente, ya que, si el elemento buscado *elem* no se encuentra en el vector, se tendrá que recorrer todo el vector, cuando se sabe que si se llega a una componente con valor mayor que *elem*, ya no se encontrará el valor buscado.

- Una primera solución a este nuevo problema sería modificar la condición de salida del bucle cambiando

v[i] = elem por **v[i] >= elem**

EJEMPLO
BÚSQUEDA SECUENCIAL ORDENADA

```
100 int busquedaSecuencialOrdenada( tVectorInt pVector, int elem ) {
101 /*  Pre-Condicion: pVector ordenado crecientemente */
102 /*  result se usará para guardar la posición del
103     valor encontrado ó -1 en caso contrario */
104     int result = -1;
105     int i = 0;
106     while ( ( pVector[i] < elem ) && ( i <= MAX ) ){
107         i = i + 1;
108     }
109     if ( pVector[i] == elem ) {
110         /*  significa que se ha encontrado el elemento elem,
111         entonces se devuelve la posición del elemento
112         en el vector */
113         result = i;
114     }
115     return result;
116 }
```

BÚSQUEDA BINARIA

- Este algoritmo de búsqueda recibe el nombre de **búsqueda binaria**, ya que va dividiendo el vector en dos sub-vectores de igual tamaño.
- El hecho de que el vector esté ordenado se puede aprovechar para conseguir una mayor eficiencia en la búsqueda.
- El algoritmo sería ….

BÚSQUEDA BINARIA

- Comparar *elem* con el elemento central; si *elem* es ese elemento ya hemos terminado ☺

- en otro caso buscamos en la mitad del vector que nos interese (según sea *elem* menor o mayor que el elemento en el centro, buscaremos en la primera o segunda mitad del vector, respectivamente).

- Posteriormente, si no se ha encontrado el elemento repetiremos este proceso comparando *elem* con el elemento central del sub-vector seleccionado, y así sucesivamente hasta que o bien encontremos el valor *elem* o bien podamos concluir que *elem* no está (porque el sub-vector de búsqueda está vacío).

EJEMPLO
BÚSQUEDA BINARIA

```
135 int busquedaBinaria( tVectorInt pVector, int elem ) {
136     /*  Pre-Condicion: pVector ordenado crecientemente
137         Devuelve -1 (si elem no esta en pVector) ó la posición i (si pVector[i] = elem) */
138     int extInf, extSup; /* extremos del intervalo */
139     int posMed;         /* posicion central del intervalo */
140     bool encontrado;
141 
142     /*  result se usará para guardar la posición del valor encontrado ó -1 en caso contrario */
143     int result = -1;
144     extInf = 0;
145     extSup = MAX;
146     encontrado = false;
147     while ( ( !encontrado ) && ( extSup >= extInf ) ) {
148         posMed = ( extSup + extInf ) / 2;
149         if ( elem == pVector[posMed] ) {
150             encontrado = true;
151         }
152         else {
153             /* se actualizan los extremos del intervalo */
154             if ( elem > pVector[posMed] ) {
155                 /* se actualiza el extremo inferior del intervalo */
156                 extInf = posMed + 1;
157             } else {
158                 /* se actualiza el extremo superior del intervalo */
159                 extSup= posMed - 1;
160             }
161         }
162     }
163 
164     if ( encontrado ) {
165         result = posMed;
166     }
167 
168     return result;
169 }
```

BÚSQUEDA DE MÁXIMOS Y MÍNIMOS

- En muchos casos, es necesario determinar el **mayor** o el **menor** valor de un conjunto de datos.

- Existen tres métodos para la resolución de este problema:
  - Ramificación del árbol
  - Campeonato
  - Supuesto o prepo

BÚSQUEDA DE MÁXIMOS Y MÍNIMOS

- **Ramificación del árbol:** Consiste en las combinaciones de comparaciones de todas las variables que intervienen. Este método se realiza teniendo en cuenta que todos los campos deben estar simultáneamente en memoria (es del tipo de búsqueda interna).

- **Campeonato:** Consiste en la comparación de a pares de todas las variables que intervienen. En este método los campos también deben estar simultáneamente en memoria.

- **Supuesto o Prepo:** Consiste en suponer que una de las variables que existe en memoria, en el mismo momento, es mayor o menor de todas, y luego se realiza las comparaciones sucesivas con las restantes. Este método se adapta para los algoritmos de búsqueda externa (los campos no están simultáneamente en memoria, sino que ingresan registro a registro).

EJEMPLO
BÚSQUEDA DEL MÁXIMO

```
140  /*  Se busca el mayor valor y se devuelve la posición
141      en la cual se encuentra - Supuesto o prepo */
142  int buscarMayor( tVectorInt pVector )  {
143      int i, mayor, posMayor;
144      mayor = 0;
145      posMayor = -1;
146      /*  también se puede suponer que el mayor es el
147          primer elemento del arreglo:
148          mayor = pVector[0];
149          posMayor = 0; */
150      for ( i = 0; i<MAX; i++ ) {
151          if ( pVector[i] > mayor ) {
152              mayor = pVector[i];
153              posMayor = i;
154          }
155      }
156      return posMayor;
157  }
```

EJEMPLO
BÚSQUEDA DEL MÍNIMO

```
162 int buscarMenor( tVectorInt pVector )  {
163     int i, menor, posMenor;
164     menor = 99999;
165     posMenor = -1;
166     /*  también se puede suponer que el menor es el
167         primer elemento del arreglo:
168         menor = pVector[0];
169         posMenor = 0; */
170     for ( i = 0; i<MAX; i++) {
171         if ( pVector[i] < menor ) {
172             menor = pVector[i];
173             posMenor = i;
174         }
175     }
176     return posMenor;
177 }
```

BIBLIOGRAFÍA

- Mark Allen Weiss - Estructuras de Datos y Algoritmos - Florida International University - Año: 1995 - Editorial: Addison-Wesley Iberoamericana .
- Joyanes Aguilar, Luis - Programación en Pascal - 4ª Edición - Año: 2006 - Editorial: McGraw-Hill/Interamericana de España, S.A.U.
- Cristóbal Pareja Flores, Manuel Ojeda Aciego, Ángel Andeyro Quesada, Carlos Rossi Jiménez - Algoritmos y Programación en Pascal.
- Joyanes Aguilar, Luis - Fundamentos de la Programación. Algoritmos, Estructuras de Datos y Objetos - 3ª Edición - Editorial: McGraw-Hill