Universidad Nacional
Jorge Basadre Grohmann

CURSO:

PROGRAMACÍÓN AVANZADA

TEMA:
Métodos de Búsqueda

SEMANA 05

2024-II

Msc. Ing. Milagros Gleny Cohaila Gonzales

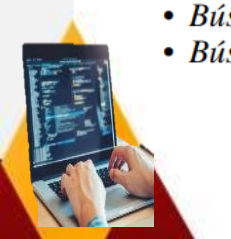
Método de Búsqueda

La recuperación de información, es una de las aplicaciones más importantes de las computadoras. La búsqueda (searching) de información está relacionada con las tablas para consultas (lookup). Estas tablas contienen una cantidad de información que se almacena en forma de listas de parejas de datos. En varios casos es necesario con frecuencia buscar un elemento en una lista. Una vez que se encuentra el elemento, la identificación de su información correspondiente es un problema menor. Por consiguiente, nos centraremos en el proceso de búsqueda. Supongamos que se desea buscar en el vector X[1].. X[n], que tiene componentes numéricos, para ver si contiene o no un número dado T. Si en vez de tratar sobre vectores se desea buscar información en un archivo, debe realizarse la búsqueda a partir de un determinado campo de información denominado campo clave. Así, en el caso de los archivos de empleados de una empresa, el campo clave puede ser el número de DNI o los apellidos. La búsqueda por claves para localizar registros es, con frecuencia, una de las acciones que mayor consumo de tiempo conlleva y, por consiguiente, el modo en que los registros están dispuestos y la elección del modo utilizado para la búsqueda pueden redundar en una diferencia sustancial en el rendimiento del programa. El problema de búsqueda cae naturalmente dentro de los dos casos típicos ya tratados. Si existen muchos registros, puede ser necesario almacenarlos en archivos de disco o cinta, externo a la memoria de la computadora. En este caso se llama búsqueda externa. En el otro caso, los registros que se buscan se almacenan por completo dentro de la memoria de la computadora. Este caso se denomina búsqueda interna. En la práctica, la búsqueda se refiere a la operación de encontrar la posición de un elemento entre un conjunto de elementos dados: lista, tabla o fichero. Ejemplos típicos de búsqueda son localizar nombre y apellidos de un alumno, localizar números de teléfono de una agenda, etc. Existen diferentes algoritmos de búsqueda. El algoritmo elegido depende de la forma en que se encuentren organizados los datos. 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.

Los métodos más usuales de búsqueda son:

- *Búsqueda secuencial o lineal.*
- *Búsqueda binaria.*
- *Búsqueda por transformación de claves (hash).*



Búsqueda Secuencial

El método más sencillo de buscar un elemento en un vector es explorar secuencialmente el vector o, dicho en otras palabras, recorrer el vector desde el primer elemento al último. Si se encuentra el elemento buscado, visualizar un mensaje similar a 'Fin de búsqueda'; en caso contrario, visualizar un mensaje similar a 'Elemento no existe en la lista'. En otras palabras, la búsqueda secuencial compara cada elemento del vector con el valor deseado, hasta que éste encuentra o termina de leer el vector completo.

La búsqueda secuencial no requiere ningún registro por parte del vector y, por consiguiente, no necesita estar ordenado. El recorrido del vector se realizará normalmente con estructuras repetitivas.

Se tiene un vector `A` *que contiene* `n` *elementos numéricos* `(n) >= 1(A[1], A[2], A[3], ..., A[n])` *y se desea buscar un elemento dado* `t`. *Si el elemento* `t` *se encuentra, visualizar un mensaje* `'Elemento encontrado'` *y otro que diga* `'posición = '`.

Si existen `n` elementos, se requerirán como media `n/2` comparaciones para encontrar un determinado elemento. En el caso más desfavorable se necesitarán `n` comparaciones.

Método 1

```
algoritmo busqueda_secuencial_1
  //declaraciones
inicio
  llenar (A,n)
  leer(t)
  //recorrido del vector
  desde i ← 1 hasta n hacer
    si A[i] = t entonces
      escribir('Elemento encontrado')
      escribir('en posicion', i)
    fin_si
  fin_desde
fin
```

Método 2

```
algoritmo busqueda_secuencial_2
  //...
inicio
  llenar (A,n)
  leer(t)
  i ← 1
  mientras (A[i] <> t) y (i =< n) hacer
    i ← i + 1
    //este bucle se detiene bien con A[i] = t o bien con i > n
  fin_mientras
  si A[i] = t entonces     //condición de parada
    escribir('El elemento se ha encontrado en la posición', i)
  si_no                    //recorrido del vector terminado
    escribir('El numero no se encuentra en el vector')
  fin_si
fin
```

Este método no es completamente satisfactorio, ya que si `t` no está en el vector A, `i` toma el valor `n + 1` y la comparación

```
A[i] <> t
```

producirá una referencia al elemento `A[n + 1]`, que presumiblemente no existe. Este problema se resuelve sustituyendo `i =< n` por `i < n` en la instrucción **mientras**, es decir, modificando la instrucción anterior **mientras** por

```
mientras (A[i] <> t) y (i < n) hacer
```

Método 3

```
algoritmo busqueda_secuencial_3
  //...
inicio
  llenar (A,n)
  leer(t)
  i ← 1
  mientras (A[i] <> t) y (i < n) hacer
    i ← i+1
    //este bucle se detiene cuando A[i] = t o i >= n
  fin_mientras
  si A[i] = t entonces
    escribir('El numero deseado esta presente y ocupa el lugar',i)
  si_no
    escribir(t, 'no existe en el vector')
  fin_si
fin
```

Método 4

```
algoritmo busqueda_secuencial_4
  //...
inicio
  llamar_a llenar(A,n)
  leer(t)
  i ← 1

  mientras i <= n hacer
    si t = A[i] entonces
      escribir('Se encontró el elemento buscado en la posicion',i)
      i ← n + 1
    si_no
      i ← i+1
    fin_si
  fin_mientras
fin
```

Búsqueda secuencial con centinela

Una manera muy eficaz de realizar una búsqueda secuencial consiste en modificar los algoritmos anteriores utilizando un elemento centinela. Este elemento se agrega al vector al final del mismo. El valor del elemento centinela es el del argumento. El propósito de este elemento centinela, A[n + 1], es significar que la búsqueda siempre tendrá éxito. El elemento A[n + 1] sirve como centinela y se le asigna el valor de t antes de iniciar la búsqueda. En cada paso se evita la comparación de i con N y, por consiguiente, este algoritmo será preferible a los métodos anteriores, concretamente el método 4. Si el índice alcanzase el valor n + 1, supondría que el argumento no pertenece al vector original y en consecuencia la búsqueda no tiene éxito.

Método 5

```
algoritmo busqueda_secuencial_5
  //declaraciones
inicio
  llenar(A,n)
  leer(t)
  i ← 1
  A[n + 1] ← t
  mientras A[i] <> t hacer
    i ← i + 1
  fin_mientras
  si i = n + 1 entonces
    escribir('No se ha encontrado elemento')
  si_no
    escribir('Se ha encontrado el elemento')
  fin_si
fin
```

Una variante del método 5 es utilizar una variable lógica (interruptor o *switch*), que represente la existencia o no del elemento buscado.

Localizar si el elemento `t` existe en una lista `A[i]`, donde `i` varía desde `1` a `n`.

En este ejemplo se trata de utilizar una variable lógica `ENCONTRADO` para indicar si existe o no el elemento de la lista.

Método 6

```
algoritmo busqueda_secuencia_6
  //declaraciones
inicio
  llenar (A,n)
  leer(t)
  i ← 1
  ENCONTRADO ← falso
  mientras (no ENCONTRADO) y (i =< n) hacer
    si A[i] = t entonces
      ENCONTRADO ← verdadero
    fin_si
    i ← i + 1
  fin_mientras
  si ENCONTRADO entonces
    escribir('El numero ocupa el lugar', i - 1)
  si_no
    escribir('El numero no esta en el vector')
  fin_si
fin
```

Nota

De todas las versiones anteriores, tal vez la más adecuada sea la incluida en el método 6. Entre otras razones, debido a que el bucle **mientras** engloba las acciones que permiten explorar el vector, bien hasta que *t* se encuentre o bien cuando se alcance el final del vector.

Método 7

```
algoritmo busqueda_secuencia_7
  //declaraciones
inicio
  llenar (A,n)
  leer(t)
  i ← 1
  ENCONTRADO ← falso
  mientras i =< n hacer
    si A[i] = t entonces
      ENCONTRADO ← verdad
      escribir ('El número ocupa el lugar'; i)
    fin_si
    i ← i + 1
  fin_mientras
  si_no (ENCONTRADO) entonces
    escribir('El numero no esta en el vector')
  fin_si
fin
```

Ejercicio 01 implementar el método 09 en C++ y Realizar el diagrama de Flujo.

Método 8

```
algoritmo busqueda_secuencial_8
  //declaraciones
inicio
  llenar (A,n)
  ENCONTRADO ← falso
  i ← 0
  leer(t)
  repetir
    i ← i+1
    si A[i] = t entonces
      ENCONTRADO ← verdad
    fin_si
  hasta_que ENCONTRADO o (i = n)
fin
```

Método 9

```
algoritmo busqueda_secuencial_9
  //declaraciones
inicio
  llenar (A,n)
  ENCONTRADO ← falso
  leer(t)
  desde i ← 1 hasta i ← n hacer
    si A[i] = t entonces
      ENCONTRADO ← verdad
    fin_si
  fin_desde
  si ENCONTRADO entonces
    escribir('Elemento encontrado')
  si_no
    escribir('Elemento no encontrado')
  fin_si
fin
```

El método de búsqueda lineal tiene el inconveniente del consumo excesivo de tiempo en la localización del elemento buscado. Cuando el elemento buscado no se encuentra en el vector, se verifican o comprueban sus n elementos.
En los casos en que el elemento se encuentra en la lista, el número podrá ser el primero, el último o alguno comprendido entre ambos. Se puede suponer que el número medio de comprobaciones o comparaciones a realizar es de (n+1)/2 (aproximadamente igual a la mitad de los elementos del vector). La búsqueda secuencial o lineal no es el método más eficiente para vectores con un gran número de elementos. En estos casos, el método más idóneo es el de búsqueda binaria, que presupone una ordenación previa en los elementos del vector. Este caso suele ser muy utilizado en numerosas facetas de la vida diaria. Un ejemplo de ello es la búsqueda del número de un abonado en una guía telefónica; normalmente no se busca el nombre en orden secuencial, sino que se busca en la primera o segunda mitad de la guía; una vez en esa mitad, se vuelve a tantear a una de sus dos submitades, y así sucesivamente se repite el proceso hasta que se localiza la página correcta.

Búsqueda binaria

- La búsqueda binaria utiliza un método de “divide y vencerás” para localizar el valor deseado. Con este método se examina primero el elemento central de la lista; si este es el elemento buscado, entonces la búsqueda ha terminado. En caso contrario se determina si el elemento buscado está en la primera o la segunda mitad de la lista y, a continuación, se repite este proceso, utilizando el elemento central de esa sublista. Supongamos la lista.

```
1231
1473
1545
1834
1892
1898        elemento central
1983
2005
2446
2685
3200
```

Si está buscando el elemento `1983`, se examina el número central, `1898`, en la sexta posición. Ya que `1983` es mayor que `1898`, se desprecia la primera sublista y nos centramos en la segunda

```
1983
2005
2446        elemento central
2685
3200
```

El número central de esta sublista es `2446` y el elemento buscado es `1983`, menor que `2446`; eliminamos la segunda sublista y nos queda

```
1983
2005
```

Como no hay término central, elegimos el término inmediatamente anterior al término central, `1983`, que es el buscado.

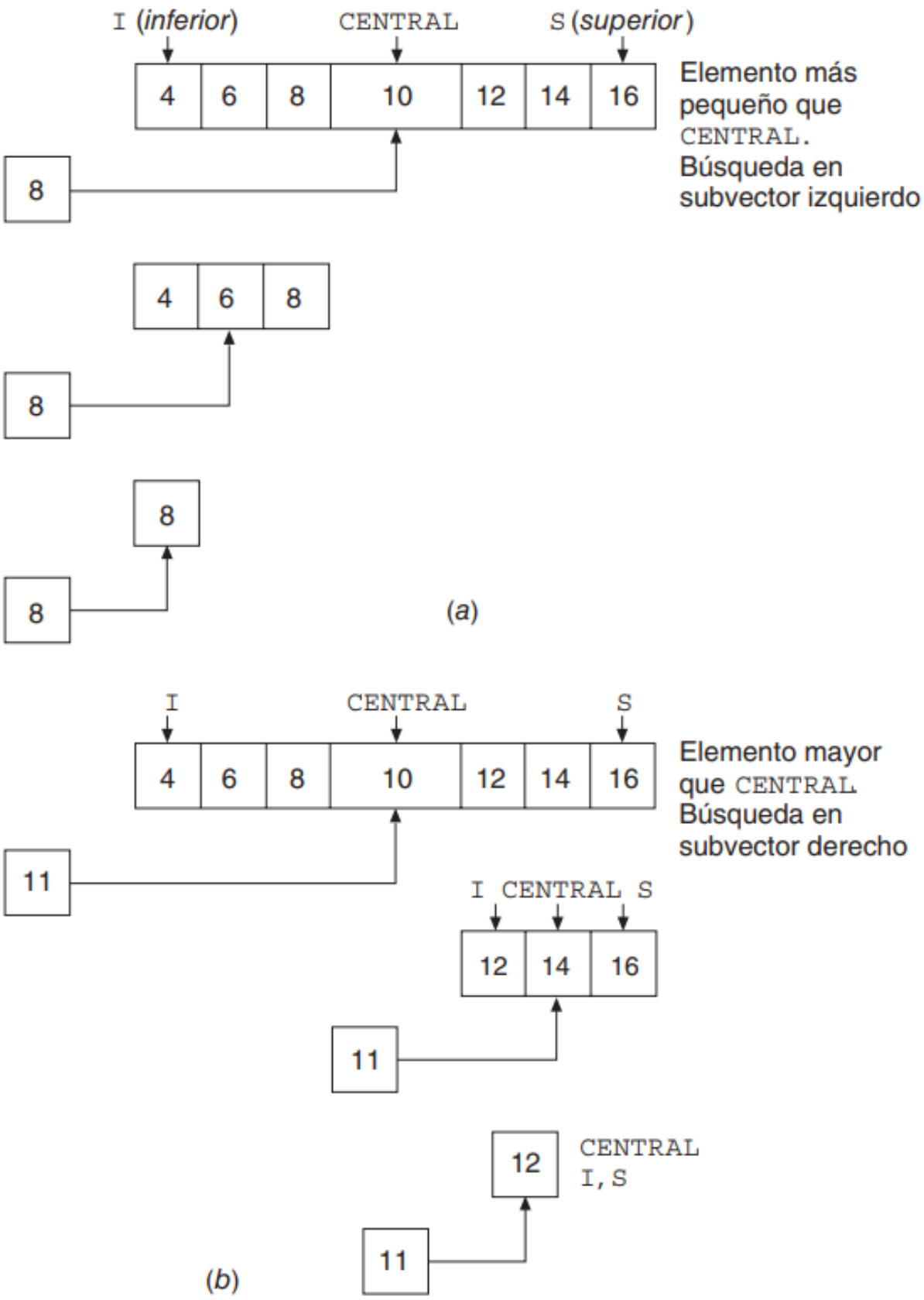
Se han necesitado tres comparaciones, mientras que la búsqueda secuencial hubiese necesitado siete.

La búsqueda binaria se utiliza en vectores ordenados y se basa en la constante división del espacio de búsqueda (recorrido del vector). Como se ha comentado, se comienza comparando el elemento que se busca, no con el primer elemento, sino con el elemento central. Si el elemento buscado —*t*— es menor que el elemento central, entonces *t* deberá estar en la mitad izquierda o inferior del vector; si es mayor que el valor central, deberá estar en la mitad derecha o superior, y si es igual al valor central, se habrá encontrado el elemento buscado.

El funcionamiento de la búsqueda binaria en un vector de enteros se ilustra en la Figura 10.3 para dos búsquedas: *con éxito* (localizado el elemento) y *sin éxito* (no encontrado el elemento).

El proceso de búsqueda debe terminar normalmente conociendo si la búsqueda *ha tenido éxito* (se ha encontrado el elemento) o bien *no ha tenido éxito* (no se ha encontrado el elemento) y normalmente se deberá devolver la posición del elemento buscado dentro del vector.

Ejemplo de búsqueda binaria: (a) con éxito, (b) sin éxito



Ejercicio 02. Implementar el siguiente pseudocódigo y Realizar el diagrama de Flujo.

Encontrar el algoritmo de búsqueda binaria para encontrar un elemento K *en una lista de elementos* X_1, X_2, . . . , X_n *previamente clasificados en orden ascendente.*

El array o vector *X* se supone ordenado en orden creciente si los datos son numéricos, o alfabéticamente si son caracteres. Las variables BAJO, CENTRAL, ALTO indican los límites inferior, central y superior del intervalo de búsqueda.

```
algoritmo busqueda_binaria
  //declaraciones
inicio
  //llenar (X,N)
  //ordenar (X,N)
  leer(K)
  //inicializar variables
  BAJO ← 1
  ALTO ← N
  CENTRAL ← ent ((BAJO + ALTO) / 2)
  mientras (BAJO =< ALTO) y (X[CENTRAL] <> K) hacer
    si K < X[CENTRAL] entonces
      ALTO ← CENTRAL - 1
    si_no
      BAJO ← CENTRAL + 1
    fin_si
    CENTRAL ← ent ((BAJO + ALTO) / 2)
  fin_mientras
  si K = X[CENTRAL] entonces
    escribir('Valor encontrado en', CENTRAL)
  si_no
    escribir('Valor no encontrado')
  fin_si
fin
```

Ejercicio 03. Implementar el siguiente pseudocódigo y Realizar el diagrama de Flujo.

Se dispone de un vector tipo carácter NOMBRE *clasificado en orden ascendente y de* N *elementos. Realizar el algoritmo que efectúe la búsqueda de un nombre introducido por el usuario.*

```
{inicializar todas las variables necesarias}
{NOMBRE       array de caracteres
N             numero de nombres del array NOMBRE
ALTO          puntero al extremo superior del intervalo
BAJO          puntero al extremo inferior del intervalo
CENTRAL       puntero al punto central del intervalo
X             nombre introducido por el usuario
ENCONTRADO    bandera o centinela}
```

```
algoritmo busqueda_nombre
inicio
  llenar (NOMBRE, N)
  leer (X)
  BAJO ← 1
  ALTO ← N
  ENCONTRADO ← falso
  mientras (no ENCONTRADO) y (BAJO =< ALTO) hacer
    CENTRAL ← ent (BAJO+ALTO) / 2
    //verificar nombre central en este intervalo
    si NOMBRE[CENTRAL] = X entonces
      ENCONTRADO ← verdad
    si_no
      si NOMBRE[CENTRAL] > X entonces
        ALTO ← CENTRAL - 1
      si_no
        BAJO ← CENTRAL + 1
      fin_si
    fin_si
  fin_mientras
  si ENCONTRADO entonces
    escribir('Nombre encontrado')
  si_no
    escribir('Nombre no encontrado')
  fin_si
fin
```

Código de referencia:

cadena.cpp busqueda_nombre.cpp

```cpp
// Algoritmo busqueda_nombre
#include<iostream>
#include<cmath>
using namespace std;
void llenar(string nombre[], float n);
void llenar(string nombre[], float n) {
    int i;
    for (i=1;i<=n;i++) {
        cout << "Ingrese el nombre["<<i<<"]: "<< endl;
        cin >> nombre[i];
    }
}
int main() {
    // Tipo
    // Array[1..100] de cadena: nombres
    string nombre[100];
    //var.
    int alto;
    int bajo;
    int central;
    bool encontrado;
    int n;
    string x;
    // Tipo
    // Inicio
    cout << "Ingrese la dimension:" << endl;
    cin >> n;
    llenar(nombre,n);
    cout << "Ingrese el valor de busqueda:" << endl;
    cin >> x;
    bajo = 1;
    alto = n;
    encontrado = false;
    while (!encontrado && bajo<=alto) {
        central = int((bajo+alto)/2);
        // verificar nombre central en este intervalo
        if (nombre[central]==x) {
            encontrado = true;
        } else {
            if (nombre[central]>x) {
                alto = central-1;
            } else {
                bajo = central+1;
            }
        }
    }
    if (encontrado==true) {
        cout << "Nombre encontrado" << endl;
    } else {
        cout << "Nombre no encontrado" << endl;
    }
    return 0;
}
```


GRACIAS!