



Departamento de Informática
Universidad Técnica Federico Santa María



Algoritmos de Búsqueda y Ordenamiento

Programación de Computadores

IWI-131-p1

Prof.: Teddy Alfaro Olave

Algoritmos de Búsqueda

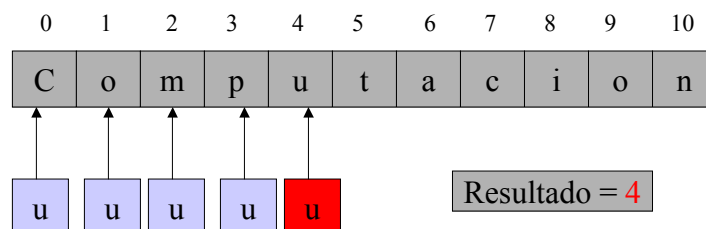
- Los procesos de búsqueda involucran recorrer un arreglo completo con el fin de encontrar algo. Lo más común es buscar el menor o mayor elemento (cuando es posible establecer un orden), o buscar el índice de un elemento determinado.
- Para buscar el menor o mayor elemento de un arreglo, podemos usar la estrategia, de suponer que el primero o el último es el menor (mayor), para luego ir **comparando con cada uno de los elementos**, e ir actualizando el menor (mayor). A esto se le llama Búsqueda Lineal.

Algoritmos de Búsqueda

- Definición:
 - Para encontrar un dato dentro de un arreglo, para ello existen diversos algoritmos que varían en complejidad, eficiencia, tamaño del dominio de búsqueda.
- Algoritmos de Búsqueda:
 - Búsqueda Secuencial
 - Búsqueda Binaria

Búsqueda Secuencial

- Consiste en ir comparando el elemento que se busca con cada elemento del arreglo hasta cuando se encuentra.
- Busquemos el elementos ‘u’



Búsqueda Secuencial

- Búsqueda del menor

```
menor = a[0];  
for (i=1;i<n;i++)  
    if ( a[i]<menor )  
        menor=a[i];
```

- Búsqueda del mayor

```
mayor= a[n-1];  
for (i=0;i<n-1;i++)  
    if ( a[i]>mayor )  
        mayor=a[i];
```

- Búsqueda de elemento

```
encontrado=-1;  
for (i=0;i<n;i++)  
    if ( a[i]==elemento_buscado )  
        encontrado=i;
```

Ejemplo

- Desarrollar un programa que posea una función que reciba como parámetro un arreglo de 10 enteros, y un entero, y retorne la posición del entero si es que se encuentra, de lo contrario devolver -1.

```
#include <stdio.h>  
  
int encuentra(int A[], int b) {  
    int k=1, result=-1;  
  
    do{  
        if (A[k]== b)  
            result =k;  
        else  
            k++;  
    }while ((result==-1)&&(k<10));  
    return result;  
}  
  
int main() {  
    int i, x[10];  
  
    for(i=0;i<10;i++)  
        scanf("%d",&x[i]);  
    i = encuentra( x, 10);  
    printf("resultado %d\n",i);  
  
    return 0;  
}
```

Eficiencia y Complejidad

- Considerando la Cantidad de Comparaciones
 - Mejor Caso:
 - 1** • El elemento buscado está en la primera posición. Es decir, se hace una sola comparación
 - Peor Caso:
 - n** • El elemento buscado está en la última posición. Necesitando igual cantidad de comparaciones que de elementos el arreglo
 - En Promedio:
 - n/2** • El elemento buscado estará cerca de la mitad. Necesitando en promedio, la mitad de comparaciones que de elementos
- Por lo tanto, la velocidad de ejecución depende linealmente del tamaño del arreglo

Búsqueda Binaria

En el caso anterior de búsqueda se asume que los elementos están en cualquier orden. En el peor de los casos deben hacerse **n** operaciones de comparación.

Una búsqueda más eficiente puede hacerse sobre un arreglo ordenado. Una de éstas es la Búsqueda Binaria.

La Búsqueda Binaria, compara si el valor buscado está en la mitad superior o inferior. En la que esté, subdivido nuevamente, y así sucesivamente hasta encontrar el valor.

Búsqueda Binaria

- Supuesto: Arreglo con datos ordenados en forma ascendente:
 $i < k \rightarrow a[i] < a[k]$
- Estamos buscando la posición del valor 17

$(0+11)/2=5$	$(6+11)/2=8$	$(6+7)/2=6$	$(7+7)/2=7$
0 2	0 2	0 2	0 2
1 3	1 3	1 3	1 3
2 4	2 4	2 4	2 4
3 5	3 5	3 5	3 5
4 6	4 6	4 6	4 6
5 8	5 8	5 8	5 8
6 13	6 13	6 13	6 13
7 17	7 17	7 17	7 17
8 19	8 19	8 19	8 19
9 23	9 23	9 23	9 23
10 25	10 25	10 25	10 25
11 26	11 26	11 26	11 26

$17 \leq 8$ +
 $17 \geq 8$ -
 $17 \leq 19$ -
 $17 \geq 19$ +
 $17 \leq 13$ -
 $17 \geq 13$ +
 $17 \leq 17$ +
 $17 \geq 17$ -
 $L > U$

Algoritmo de Búsqueda Binaria

```
#include <stdio.h>

int main() {
    int b,i,j,k, v[12];

    for(i=0;i<12;i++)
        scanf("%d",&v[i]);
    printf("fin del llenado\n");
    printf("ingrese numero a buscar ");
    scanf("%d",&b);
    i= 0;
    j= 12-1;
    do {
        k= (i+j)/2;
        if (v[k]<=b )
            i=k+1;
        if (v[k]>=b )
            j= k-1;
    } while (i<=j);
    printf("elemento %d esta en %d\n",v[k],k);
    return 0;
}
```



```
i= 0;
j= tamaño-1;
do {
    k= (i+j)/2;
    if (v[k]<=b )
        i=k+1;
    if (v[k]>=b )
        j= k-1;
} while (i<=j);
```

Algoritmo de Búsqueda Binaria

```
#include <stdio.h>

int main() {
    int b,i,j,k, v[12];

    for(i=0;i<12;i++)
        scanf("%d",&v[i]);
    printf("fin del llenado\n");
    printf("ingrese numero a buscar ");
    scanf("%d",&b);
    i= 0;
    j= 12-1;
    do {
        k= (i+j)/2;
        if (v[k]<=b )
            i=k+1;
        if (v[k]>=b )
            j= k-1;
    } while (i<=j);
    printf("elemento %d esta en %d\n",v[k],k);
    return 0;
}
```



```
i= 0;
j= tamaño-1;
do {
    k= (i+j)/2;
    if (v[k]<=b )
        i=k+1;
    if (v[k]>=b )
        j= k-1;
} while (i<=j);
```

Complejidad y Eficiencia

- Contando Comparaciones

- Mejor Caso:

1

- El elemento buscado está en el centro. Por lo tanto, se hace una sola comparación

- Peor Caso:

log(n)

- El elemento buscado está en una esquina. Necesitando $\log_2(n)$ cantidad de comparaciones

log(n/2)

- En Promedio:

- Serán algo como $\log_2(n/2)$

- Por lo tanto, la velocidad de ejecución depende logarítmicamente del tamaño del arreglo