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

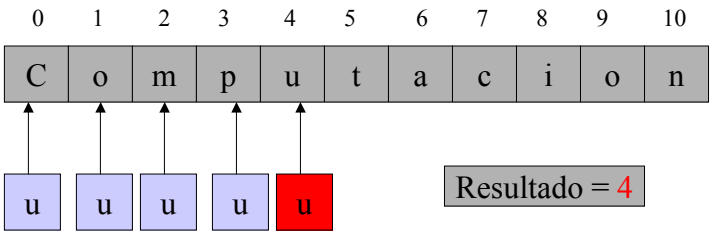
- Los procesos de búsqueda involucran recorrer un arreglo completo con el fin de encontrar algo. Lo más común es buscar el menor o mayor elemento (cuando es puede establecer un orden), o buscar el índice de un elemento determinado.
- Para buscar el menor o mayor elemento de un arreglo, podemos usar la 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

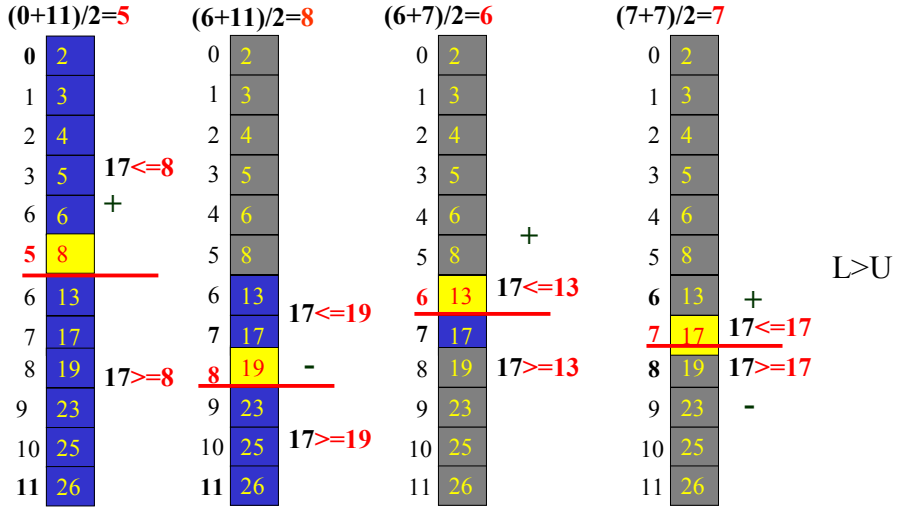
En el caso anterior de búsqueda se asume que los elementos están en cualquier orden. En el peor de los casos deben hacerse **n** operaciones de comparación.

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 → a[i]<a[k]
- Estamos buscando la posición del valor 17



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;
}
```

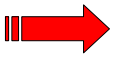
```
i= 0;
j= tamaño-1;
do {
      k= (i+j)/2;
      if (v[k]<=b )
            i=k+1;
      if (v[k]>=b )
            j= k-1;
 } while (i<=j);
```

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

Ordenamiento de Componentes

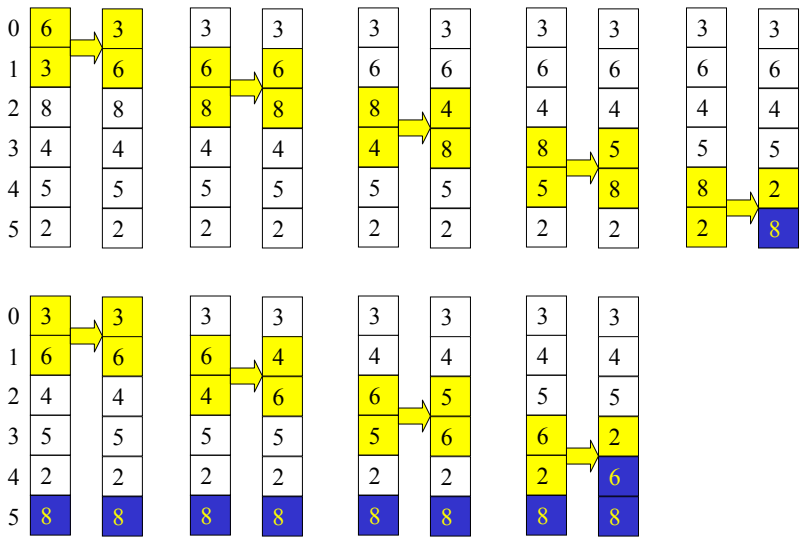
Ordenamiento Ascendente

- Existen numerosos algoritmos para ordenar. A continuación se verán algunos algoritmos de ordenamiento.
- Ordenamiento Burbuja (bublesort):

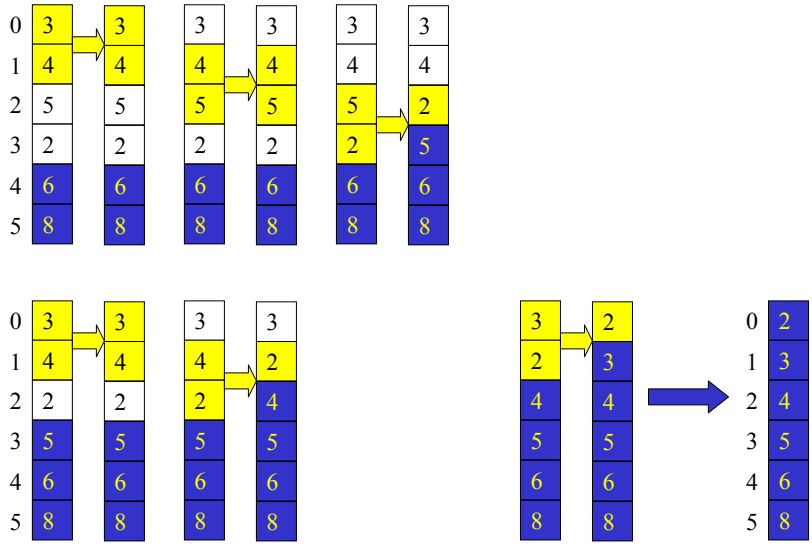
  Idea: vamos comparando elementos adyacentes y empujamos los valores más livianos hacia arriba (los más pesados van quedando abajo). Idea de la burbuja que asciende, por lo liviana que es.

0	3
1	6
2	8
3	4
4	5
5	2

Ordenamiento Burbuja



Ordenamiento Burbuja



Algoritmo Ordenamiento Burbuja

```
#include <stdio.h>
#define N 6
void intercambia(int *f,int *g) {
      int tmp;

      tmp = *f;
      *f = *g;
      *g = tmp;
 }

int main() {
      int i,j, v[N]={3,4,5,2,6,8};

      for (i=N-1;i>1;i--)
            for (j=0;j<i;j++)
                  if (v[j]>v[j+1])
            intercambia(&v[j],&v[j+1]);
      for (i=0;i<N;i++)
            printf("%d\n",v[i]);
      return 0;
}
```

```
for (i=N-1;i>0;i--)
    for (j=0;j<i;j++)
      if (V[j]>V[j+1])
          Intercambia(&A[j],&A[j+1]);
```

Complejidad y Eficiencia

- Cantidad de Comparaciones:
  - Constante: n*(n+1)/2
- Cantidad de Intercambios:
  - Mejor Caso:

0

    - Arreglo ordenado. Por lo tanto, no se hace ni un solo swap
  - Peor Caso:

n*(n+1)/2

    - Arreglo ordenado inversamente. Se necesitarán n*(n+1)/2 cantidad de swaps
  - En Promedio:

n*(n+1)/4

    - Serán algo como n*(n+1)/4 swaps
- Por lo tanto, la velocidad de ejecución depende cuadráticamente del tamaño del arreglo

Búsqueda en Arreglos Bidimensionales

Arreglos Bidimensionales.

- **Definición**:
  - Es un arreglo de dos dimensiones el cual está indexado por medio de 2 índices.

    A[i][j]
- **Declaración:**

```
tipo nombre_de_variable[tamaño_1][tamaño_2];
```

Arreglos Bidimensionales

- Una matriz bidimensional tiene la siguiente forma:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots\ldots & a_{1n} \\ a_{21} & a_{22} & \ldots\ldots & a_{2n} \\ \ldots\ldots & & & \\ a_{m1} & a_{m2} & \ldots\ldots & a_{mn} \end{bmatrix}_{mxn}$$

  - Para acceder al dato a_{ij} se hace de la siguiente manera:

    c=A[i][j];

Búsqueda

- Para buscar un elemento en un arreglo de dos dimensiones (el menor o el mayor), podemos suponer que uno de ellos es el menor (mayor), o mejor suponer un valor muy alto (o muy bajo), para luego contrastarlo uno a uno cada elemento, es decir una búsqueda secuencial.

Ejemplo de Búsqueda

```
#include <stdio.h>
#define N 3

int main() {
    int i,j,max,min, a[N][N];
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            a[i][j] = rand();
    max=-1000;
    min= 1000;
    for(i=0;i<N;i++)
        for(j=0;j<N;j++) {
            if (a[i][j]>max)
                max = a[i][j];
            if (a[i][j]<min)
                min = a[i][j];
        }
    printf("el maximo es %d y el minimo es %d\n",max,min);
    return 0;
}
```

Búsqueda por Filas

- Algoritmo: Búsqueda por filas

```
tipo A[filas][columnas]
for (i=0;i<filas;i++)
    for (j=0;j<columnas;j++)
       if (A[i][j]==elemento)
              printf("\nElemento encontrado!!!");
```

- Algoritmo: Búsqueda por columnas

```
tipo A[filas][columnas]
for (j=0;j<columnas;j++)
    for (i=0;i<filas;i++)
       if (A[i][j]==elemento)
              printf("\nElemento encontrado!!!");
```

Ejercicio 1

- El gerente de las salas de cine Cinemax desea conocer algunas estadísticas respecto de las películas más vistas. Las mismas 15 películas son exhibidas en cada una de las 7 salas de cine. Para cada sala se requiere almacenar el total de personas que han asistido a ver cada película.

  Se requiere saber cual es la mejor combinación sala-película, más vista

  Cual fue la película más vista.

Ejercicio 2

La gerencia de la empresa Machucambo, preocupada por las bajas remuneraciones de su personal, ha decidido entregar una bonificación ascendiente al 5% del sueldo a los 25 empleados con más baja remuneración. El gerente desea tener una lista con el RUT de los beneficiados y, además, desea saber a cuánto asciende el costo total de las bonificaciones. La empresa almacenará los datos del personal en dos arreglos paralelos: uno contendrá el RUT de los 121 empleados y otro estará en correspondencia con éste conteniendo el sueldo de cada uno. Los arreglos son:

```
#define N 121
int rut[n];
float sueldo[n];
```

Arreglos Bidimensionales

- Ejercicios:

  10.- Los conductores de un vehículo que llegan a un servicentro necesitan saber la distancia que recorreran desde una ciudad a otra, para así saber cuánto combustible necesitarán.

  La distancia entre las ciuidades viene dada en una matriz con la siguiente definición:

  ```
  int dist[N][N] ;
  ```

  en donde los índices representan ciudades y dist[i][j] representa la distancia en [Km] entre la ciudad i y la j.

  La ruta de ciudades que tiene que seguir un determinado conductor viene almacenada en un arreglo definido como:

  ```
  int ruta[M];
  ```

  Calcular la cantidad de combustible necesitado.