Contents

1 Búsqueda y Ordenación

1.1 Métodos de búsqueda

Existen aplicaciones en las cuales es necesario consultar si un elemento se encuentra dentro de una estructura.

A continuación veremos dos métodos de búsqueda:

- Búsqueda Lineal
- Búsqueda Binaria

1.2 Búsqueda Lineal

En la búsqueda lineal se recorre en forma secuencial la estructura hasta que

- ó bien se encuentra el elemento deseado. Por ejemplo en un arreglo sería:

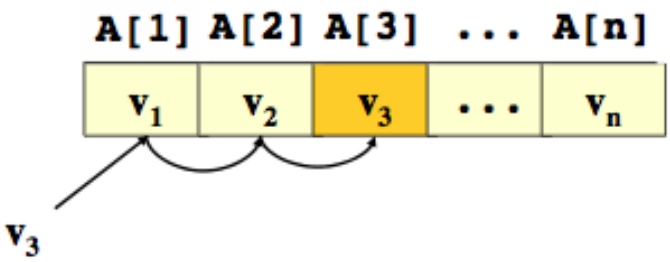

Figure 1: Búsqueda linea

- o se examina sin éxito todos los elementos de la estructura.

1.3 Ejemplo de búsqueda lineal en un arreglo

```
type
      arreglo = array[1..n] of integer;
Function BLineal(x: integer; A: arreglo): boolean;
var i : integer;
begin
  i := 1;
```

```
  while (i <= n) and (A[i] <> x) do
     i := i + 1;
  BLineal := i <= n
end;
```

1.4 Búsqueda Binaria

La búsqueda de un elemento en un arreglo puede acelerarse en forma considerable si los elementos del mismo están ordenados.

En tal caso, una forma eficiente de búsqueda es el de división sucesiva en partes iguales del intervalo donde debe buscarse el elemento.

Este método se conoce con el nombre de búsqueda binaria o bipartición.

1.5 Algoritmo de Búsqueda Binaria

Intuitivamente el algoritmo de búsqueda binaria procede de la siguiente forma:

1. Mirar si el elemento que se busca está en el punto medio del intervalo.
2. Si no está en esa posición, entonces repetir la búsqueda, pero concentrándose ahora en la primera o segunda mitad del intervalo, según sea el elemento menor o mayor que el valor en el punto medio.

1.6 Algoritmo de Búsqueda Binaria

Supongamos que estamos buscando un valor x en un intervalo inf..sup.

- Si x está en la posición
  medio = (inf + sup) DIV 2 entonces la búsqueda fue exitosa.
- Caso contrario,
  - si x < valor en medio, entonces buscar en el intervalo inf..medio -1.
  - sino buscar en el intervalo medio+1..sup.

1.7 Algoritmo de Búsqueda Binaria

- Existe otra condición de detención del algoritmo que corresponde al caso en que el valor x buscado no está en el arreglo.
- Dicha condición es verificable en cada iteración del algoritmo.

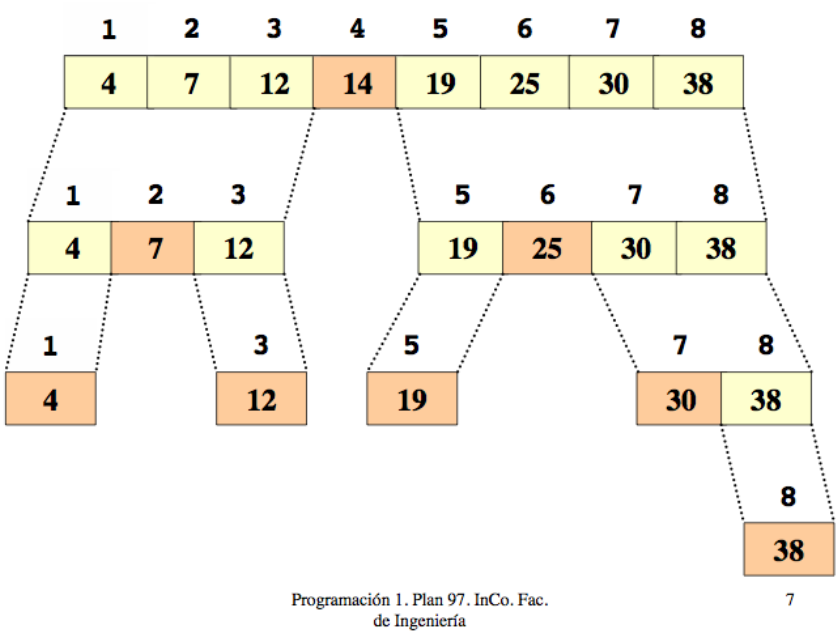

Figure 2: Búsqueda binaria

- Determina si el intervalo donde vamos a realizar la próxima búsqueda es vacío o no. Esto se verifica facilmente: inf..sup es vacío <=> inf > sup

```
Type
    arreglo = array [1 .. N] of Integer;
Function BBinaria(x: integer;A: arreglo): boolean;
{ precond: A ordenado}
var inf, sup, medio: 0..n+1;
begin
  inf := 1;
  sup := N;
  medio := (inf + sup) DIV 2;
  while (inf <= sup) and (A[medio] <> x) do begin
     if x < A[medio] then
        sup := medio - 1
     else
        inf := medio + 1;
     medio := (inf + sup) DIV 2
  end;
BBinaria := inf <= sup
end;
```

1.8 Ordenación

Por ordenar se entiende el proceso de reorganizar un conjunto de objetos en una cierta secuencia de acuerdo a un criterio especificado. En general, el objetivo de este proceso es facilitar la posterior búsqueda de elementos en el conjunto ordenado.Por ejemplo, el método de búsqueda binaria necesita que el arreglo esté ordenado para poder ser aplicado.

1.9 Ordenación

- Existen múltiples ejemplos reales de conjuntos ordenados: la guía telefónica, índices de libros, ficheros de bibliotecas, diccionarios, actas de exámenes, etc.
- Existe una gran variedad de métodos de ordenación para arrays. Veremos uno de esos métodos, llamado ordenación por inserción.

1.10 Ordenación

- Ordenar un conjunto de objetos a1,a2,. . . ,an consiste en encontrar una permutación ak1,ak2,. . . ,akn tal que dada una función de ordenación f se verifique: f(ak1) <= f(ak2) <= . . . <= f(akn)
- Normalmente, aplicar la función f a un objeto ai corresponde a seleccionar el valor de alguno de sus componentes (por ej., un campo en el caso de registros).
- A ese valor se le denomina la clave del objeto (puede ser el mismo objeto, por ej., cuando los ai son de un tipo simple).

1.11 Ordenación por Inserción

- Este método debe su nombre al hecho de que en la i- ésima pasada se inserta el i-ésimo elemento del arreglo en el lugar adecuado entre los i-1 elementos que lo preceden, los cuales ya fueron ordenados previamente.
- Como resultado de dicha inserción los i primeros elementos del arreglo quedan ordenados.

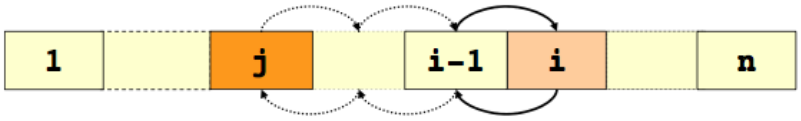

Figure 3: Ordenación por inserción

```
Type
    arreglo = array [1 .. N] of T;
Procedure OrdIns (var A: arreglo);
var
   i,j: 1..n;
begin
   for i := 2 to n do begin
      j := i;
      while (j >= 2) and (f(A[j]) < f(A[j-1])) do begin
         intercambio(A[j], A[j-1]);
         j := j - 1
      end
   end;
end;
```

1.12 Ordenación por Selección

- Se localiza el elemento de mayor clave del arreglo y se lo intercambia con el elemento que se encuentra en la última posición.
- Se repite este procedimiento en la porción del arreglo que no incluye la última posición.

Figure 4: Ordenación por selección

1.13 Ordenación por Selección

Supongamos nuevamente que el arreglo a ordenar es

```
type
    arreglo = array [1..N ] of T;
```

El método procede entonces de esta forma:

```
   for i := n downto 2 do
      encontrar el máximo entre 1 e i
      intercambiar el máximo con el elemento A[i]
```

1.14 Encontrar el máximo

```
Type
     rango = 2 .. N;
Function maximo(ultimo: rango; A: arreglo): rango;
var
    j, max : 1..n;
begin
    max := 1;
    for j := 2 to ultimo do
    if f(A[j]) > f(A[max]) then
        max := j;
    maximo := max
end;
```

```
Procedure OrdSel(var A: arreglo);
var
   i, mayor: 1..n;
   temp    : T;
begin
   for i := n downto 2 do  begin
      mayor := maximo(i,A);
      temp := A[mayor];
      A[mayor] := A[i];
      A[i] := temp
    end;
end;
```