TEMA 5: TABLAS

5.1 Introducción

5.2 Esquemas de recorrido en tablas

- **Secuencias marcadas**
- **Secuencias sin marca**

5.3 Búsqueda en tablas

5.4 Tablas multidimensionales

5.5 Algoritmos de ordenación

5.1 Introducción: Ejemplos de tablas

- Un tipo tabla que representa el número de apariciones de cada letra mayúscula en un texto:

  FrecuenciaLetras = **TIPO TABLA** ['A', 'Z'] **DE** Entero ≥ 0;

- Un tipo tabla que define una estructura para almacenar el gasto en educación en los veinticinco países de la Unión Europea

  GastosUE = **TIPO TABLA** [1, 25] **DE** Real;

- Un tipo tabla que define una estructura para almacenar las notas del examen de una asignatura que tendrá como máximo 200 matriculados.

  NotasAsignatura = **TIPO TABLA** [1, 200] **DE** Real;

Ejemplo: La tabla Curso representa un curso de estudiantes definidos mediante el tipo registro Estudiante. El valor -1 indica que no existe nota registrada para ese alumno

LÉXICO

Estudiante = **TIPO** < nombre : Secuencia de Carácter;
edad : Entero;
sexo : Booleano;
nota : Real > ;
MAX_ALUMNOS = 200 ;
Curso = **TIPO TABLA** [1, MAX_ALUMNOS] **DE** Estudiante;
notas : Curso;
cod : Entero [1, MAX_ALUMNOS];
// suponemos que la tabla 'notas' ya tiene valores

ALGORITMO

Leer (cod);
SI $notas_{cod}$.nota ≠ –1.0 **ENTONCES** Escribir ($notas_{cod}$.nota)
SI_NO Escribir ("No existe una calificación para ese alumno")
FIN_SI

FIN;

5.2 Esquemas de recorrido en tablas

Una secuencia S de longitud L de elementos de tipo Tb se puede representar en una tabla T cuyos elementos son del mismo tipo. El tamaño de la tabla (LMAX) debe ser al menos de tamaño L si la secuencia no está marcada y L+1 si la secuencia está marcada

Secuencia S

																L	L+1
E	L		S	O	L		S	E		E	S	C	O	N	D	E	#

Tabla T

LMAX

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
E	L		S	O	L		S	E		E	S	C	O	N	D	E	#							

5.2 Esquemas de recorrido en tablas: Representación de secuencias marcadas

SECUENCIAS	TABLAS
S : Secuencia de Tb	T : **TABLA** [1, LMAX] **DE** Tb
Posición de la ventana	i : Entero [1, LMAX]; // Primer Modelo
EA (S)	T_i
MarcaFin	Un elemento no significativo de Tb

Operaciones de consulta

Comenzar: implica situarse en el primer elemento de la tabla, por tanto la asignación i ←1 realiza esta acción.

Avanzar: para movernos al siguiente elemento hay que incrementar i en 1:
i ← i + 1

5.2 Esquemas de recorrido en tablas: Representación de secuencias marcadas

- **Operaciones de creación**
  - **Crear:** nos situamos en la primera posición con $i \leftarrow 1$.
  - **Registrar (x):** la asignación $t_i \leftarrow x$ registra x en la posición actual, y con $i \leftarrow i + 1$ avanzamos a la siguiente posición.
  - **Marcar:** en la posición i debemos grabar el elemento elegido como marca de fin, $t_i \leftarrow$ MarcaFin, donde $i = L + 1$.

- **Primer Esquema (Primer Modelo) adaptado a tablas**

LÉXICO

LMAX = "constante con la longitud máxima de la tabla";
MarcaFin = "constante de tipo Tb";
T :**TABLA** [1, LMAX] **DE** Tb; i : Entero [1, LMAX];

ALGORITMO

$i \leftarrow 1$; // Comenzar
{ Tratamiento inicial };
MIENTRAS $T_i \neq$ MarcaFin **HACER**
  { Tratamiento del elemento actual T_i };
  $i \leftarrow i + 1$ // Avanzar
FIN_MIENTRAS;
{ Tratamiento final }

FIN.

EJEMPLO : Sean los tipos Curso y Estudiante. El valor –2 de una nota es utilizado como marca de fin. Se desea escribir un algoritmo que obtenga la distribución de las notas: el número de suspensos, aprobados, notables, sobresalientes y matrículas de honor. Se considera suspenso una calificación < 5, aprobado >=5 y < 7, notable >= 7 y < 9, sobresaliente a partir de 9 y matrícula de honor solamente los alumnos que han obtenido un 10.

LÉXICO

NUM_NOTAS = 5;

MAX_ESTUDIANTES = 100;

MarcaFinNotas = –2.0;

Estudiante = **TIPO** < nombre : Secuencia de Carácter;

edad : Entero;

sexo : Booleano;

nota : Real >;

Curso = **TIPO TABLA** [1, MAX_ESTUDIANTES] **DE** Estudiante;

c : Curso;

fNotas : **TABLA** [1, NUM_NOTAS] **DE** Entero ≥ 0;

i : Entero [1, MAX_ESTUDIANTES];

j : Entero [1, NUM_NOTAS];

Ejemplo distribución de notas

```
ALGORITMO
      Leer (c);          // Se introducen los datos de los estudiantes en la tabla
      i ←1;              // Comenzar
      j RECORRIENDO [1, NUM_NOTAS] HACER          // Tratamiento inicial
            fNotas_j ← 0
      FIN_RECORRIENDO;
      MIENTRAS c_i.nota ≠ MarcaFinNotas HACER
        SEGÚN c_i.nota        // Tratamiento de cada elemento de la tabla
            c_i.nota < 5                          :        fNotas_1 ← fNotas_1 + 1
            (c_i.nota ≥ 5) Y (c_i.nota < 7)       :        fNotas_2 ← fNotas_2 + 1
            (c_i.nota ≥ 7) Y (c_i.nota < 9)       :        fNotas_3 ← fNotas_3 + 1
            (c_i.nota ≥ 9) Y (c_i.nota < 10)      :        fNotas_4 ← fNotas_4 + 1
            c_i.nota = 10                         :        fNotas_5 ← fNotas_5 + 1
        FIN_SEGÚN;
        i ← i + 1     //  Avanzar
      FIN_MIENTRAS;
      j RECORRIENDO [1, NUM_NOTAS] HACER          // Tratamiento final
            Escribir (fNotas_j)
      FIN_RECORRIENDO
FIN.
```

5.2 Esquemas de recorrido en tablas: Representación de secuencias no marcadas

SECUENCIAS	TABLAS
S : Secuencia de Tb	T : **TABLA** [1, LMAX] **DE** Tb

Posición de la ventana	i : Entero [0, LMAX]; // Segundo Modelo
	i : Entero [1, LMAX]; // Tercer Modelo

EA (S)	T_i
Longitud	long : Entero [0, LMAX]

Operaciones de consulta

Iniciar: se inicializa el índice, pero el elemento actual no es significativo. Se utiliza el valor 0 para señalar esta situación, por tanto se aplicará i ← 0.

Avanzar: igual que con secuencias marcadas, para obtener el siguiente elemento se aplicará la instrucción i ← i + 1.

EsÚltimo: la condición i = long indica que el índice está situado en el último elemento de la secuencia almacenada en la tabla (es preciso conocer la longitud de la tabla a priori, pues no hay marca de fin).

EsVacía: la tabla no contendrá ningún elemento cuando la longitud de la secuencia sea cero, long = 0.

5.2 Esquemas de recorrido en tablas: Representación de secuencias no marcadas

Operaciones de creación

- Crear: se aplicará i ← 1.
- Registrar (x): se aplicará t_i ← x; i ← i + 1.

- **Segundo Esquema (Segundo Modelo) adaptado a tablas**

```
i ← 0;
SEGÚN long
    long = 0:
        {Tratamiento sec. vacía }
    long ≠ 0:
        { Inic. del tratamiento }
        REPETIR
            i ← i +1;
            { Tratamiento de EA }
        HASTA i=long ;
        { Terminación del tratamiento }
FIN_SEGÚN;
```

5.2 Esquemas de recorrido en tablas: Representación de secuencias no marcadas

- **¿Qué esquema se adapta mejor al tratamiento de una tabla?**
- **Esquema utilizando la composición I-Recorriendo**

LÉXICO

LMAX = “constante con la longitud máxima de la tabla” ;

long : Entero [0, LMAX];

T : **TABLA** [1, LMAX] **DE** Tb;

i : Entero [1, LMAX];

ALGORITMO

SEGÚN long

long = 0: *{ Tratamiento final, caso de secuencia vacía }*

long ≠ 0: *{ Tratamiento inicial }*

i **RECORRIENDO** [1, long] **HACER**

{ Tratamiento del elemento actual T_i }

FIN_RECORRIENDO;

{ Tratamiento Final }

FIN_SEGÚN

FIN.

Ejemplo: Sea el tipo tabla Curso del problema anterior. Supuesta una secuencia no marcada de estudiantes almacenada en una tabla de dicho tipo, se desea escribir un algoritmo que muestre el porcentaje de hombres y de mujeres, y la nota media de cada sexo.

```
LÉXICO
   MAX_ESTUDIANTES = 100;
   Estudiante = TIPO < nombre: Secuencia de Carácter;
                       edad : Entero; sexo : Booleano;  // verdadero si es mujer,
                       nota : Real >;
   Curso = TIPO TABLA [1, MAX_ESTUDIANTES] DE Estudiante;
   c : Curso;
   contaM, contaH : Entero [0, MAX_ESTUDIANTES];  // contadores personas
   sumaM, sumaH : Real;                           // contadores notas
   mujeres, hombres : Real;                       // porcentajes de mujeres y hombres
   long : Entero [0, MAX_ESTUDIANTES];            // longitud secuencia almacenada
   i : Entero [1, MAX_ESTUDIANTES];               // índice para la tabla
```

Ejemplo porcentajes

```
ALGORITMO
Leer (c, long);    // Se introducen los datos estudiantes en la tabla c
SEGÚN long
    long = 0 : Escribir ("No hay alumnos");
    long ≠ 0 :
              sumaM ← 0.0; sumaH ← 0.0;
              contaM ← 0; contaH ← 0;
               i RECORRIENDO [1, long] HACER
                    SI ci.sexo ENTONCES
                              sumaM ←sumaM + ci.nota;
                              contaM ←contaM + 1
                    SI_NO
                              sumaH ←sumaH + ci.nota;
                              contaH ←contaH + 1
                    FIN_SI
              FIN_RECORRIENDO;
              mujeres ← contaM / long * 100; hombres ← contaH / long * 100;
              Escribir (mujeres, hombres, sumaM/contaM, sumaH/contaH)
   FIN_SEGÚN
FIN.
```

5.3 Búsqueda en tablas

- Se pueden realizar búsquedas con cualquiera de los modelos vistos, adaptados de forma adecuada.
- En el caso de las Tablas el modelo que más se ajusta es el tercer modelo, pues los valores de los índices del recorrido nunca se salen de los límites.

Esquema de búsqueda del primer modelo adaptado a tablas

ALGORITMO
    i $\leftarrow$ 1 ;
    MIENTRAS $T_i \neq$ MarcaFin **YDESPUES NO** Pro (T_i) **HACER**
        i $\leftarrow$ i + 1
    FIN_MIENTRAS;
    SI $T_i \neq$ MarcaFin **ENTONCES** *{ Tratamiento T_i cumple la propiedad }*
    SI_NO *{ Tratamiento ningún elemento cumple la propiedad }*
    FIN_SI
FIN.

5.3 Búsqueda en tablas

Esquema de búsqueda del segundo modelo adaptado a tablas

ALGORITMO
  SI long = 0 **ENTONCES**
      { Tratamiento final, caso de la secuencia vacía }
  SI_NO
    i ← 0 ;
    REPETIR
      i ← i + 1
    HASTA (i = long) **O** Pro (T_i) ;
    SI Pro (T_i) **ENTONCES** *{ Tratamiento T_i cumple la propiedad }*
    SI_NO *{ Tratamiento ningún elemento cumple la propiedad }*
    FIN_SI
  FIN_SI
FIN.

5.3 Búsqueda en tablas

Esquema de búsqueda del tercer modelo

ALGORITMO
    SEGÚN long
        long = 0 : *{ Tratamiento secuencia vacía (no encontrado) }*
        long > 0 :
                i ← 1;
                MIENTRAS (i ≠ long) **Y NO** Pro (T_i) **HACER**
                        i ← i + 1
                FIN_MIENTRAS;
                SI Pro (T_i) **ENTONCES**
                        { Tratamiento T_i cumple la propiedad }
                SI_NO
                        { Tratamiento ningún elemento cumple la propiedad }
                FIN_SI
    FIN_SEGÚN
FIN.

Ejemplo: Se desea escribir un algoritmo que dada la posición *P* de un día en el año calcule la fecha que le corresponde (mes y día), a partir de una tabla *tp* que almacena la posición del primer día de cada mes. La fecha se escribirá en el formato día/mes. Supóngase que el año no es bisiesto. Por ejemplo, a la posición 24 le corresponde la fecha 24/1 y a 144 la fecha 24/5.

LÉXICO

```
MesAño = TIPO Entero [1, 12];
DiaAño = TIPO Entero [1, 365];
tp : TABLA [MesAño] DE Entero = { 1, 32, 60, 91, 121, 152,
                    182, 213, 244, 274, 305, 335 };
p : DiaAño;                                  // dato de entrada
i : MesAño;                                  // índice para la búsqueda
mes : MesAño;                                // resultado: mes
dia : Entero [1, 31];                        // resultado: año
```

Ejemplo: cálculo de fecha

ALGORITMO (versión 1)

```
Leer (p);
i ← 2;
MIENTRAS (i ≠ 12) Y (tp_i ≤ p) HACER
    // INV { (∀j: 1 ≤ j < i : p ≥ tp_j) }
        i ← i + 1
FIN_MIENTRAS;
SI tp_i > p ENTONCES mes ← i - 1
SI_NO mes ← 12
FIN_SI;
dia ← p - tp_mes + 1;
Escribir (dia, "/", mes)
FIN.
```

ALGORITMO (versión 2)

```
Leer (p);
SI p ≥ tp_12 ENTONCES mes ← 12
SI_NO
    i ← 2;
    MIENTRAS tp_i ≤ p HACER
    // INV { (∀j: 1 ≤ j < i : p ≥ tp_j) }
            i ← i + 1
    FIN_MIENTRAS;
    mes ← i - 1
FIN_SI;
dia ← p - tp_mes + 1;
Escribir (dia, "/", mes)
FIN.
```

Ejemplo: Sea una secuencia de enteros representada mediante una tabla *t*, llamamos segmento con ceros en los extremos, *SegLimCeros*, al segmento <*iz*, *dr*> en el que $t_{iz} = 0$ y $t_{dr} = 0$. Sea n_j el número de enteros distintos de cero en un segmento *SegLimCeros*, $1 \leq j \leq$ *NumSegLimCeros*, donde *NumSegLimCeros* (*nslc*) es el número de segmentos *SegLimCeros* de la tabla *t*. Escriba un algoritmo que accediendo una única vez a cada elemento de la tabla *t* obtenga el siguiente valor:

$$\sum_{j=1}^{nslc} n_j$$

- Por ejemplo, si *t* almacena la secuencia {0, 9, 0, 0, 1, 2, 0, 0}, los segmentos con ceros en los extremos serían: <1, 3>, <1, 4>, <1, 7>, <1, 8>, <3, 4>, <3, 7>, <3, 8>, <4, 7>, <4, 8> y <7, 8>, y los valores de n_j serían, respectivamente, 1, 1, 3, 3, 0, 2, 2, 2, 2 y 0, por lo que el resultado sería 16.
- POST { noCerosSLC = "suma de enteros no ceros en los *SegLimCeros* de *t*" }
- INV { noCerosSLC = "suma de enteros no ceros en los *SegLimCeros* de P_{iz}" }

Ejemplo: *NumSegLimCeros*

- El elemento actual es cero ($t_i = 0$): encontramos nuevos segmentos *SegLimCeros* cuyo extremo derecho es *i*, tantos como ceros haya en P_{iz}. La cuestión que debemos responder es cómo actualizar noCerosSLC.
- El elemento actual es distinto de cero ($t_i \neq 0$): no encontramos nuevos segmentos *SegLimCeros*.
- INV { noCerosSLC = "suma de enteros distintos de cero en los *SegLimCeros* de la P_{iz}"; Y

  pesoNoCeros = "suma del peso de cada entero distinto de cero en la P_{iz}"; Y

  ceros = "número de ceros en la P_{iz}" }

Ejemplo: *NumSegLimCeros*

- Haremos una definición inductiva de las funciones que participan en el invariante ($t_{1..0}$ representa la *tabla vacía*, es decir la tabla donde no se ha recorrido ningún elemento, como [] en las definiciones inductivas sobre secuencias):

- ceros ($t_{1..0}$) = 0
  ceros ($t_{1..i}$) =
  ceros ($t_{1..i-1}$) si $t_i \neq 0$
  ceros ($t_{1..i-1}$) + 1 si $t_i = 0$
- pesoNoCeros ($t_{1..0}$) = 0
  pesoNoCeros ($t_{1..i}$) =
  pesoNoCeros ($t_{1..i-1}$) si $t_i = 0$
  pesoNoCeros ($t_{1..i-1}$) + ceros ($t_{1..i-1}$) si $t_i \neq 0$
- noCerosSLS ($t_{1..0}$) = 0
  noCerosSLS ($t_{1..i}$) =
  noCerosSLS ($t_{1..i-1}$) si $t_i \neq 0$
  noCerosSLS ($t_{1..i-1}$) + pesoNoCeros ($t_{1..i-1}$) si $t_i = 0$

Ejemplo: *NumSegLimCeros*

LÉXICO

LMAX = 100;
Rango = **TIPO** Entero [1, LMAX];
t : **TABLA** [Rango] **DE** Entero;
long : Rango;
i : Rango;
noCerosSLC : Entero;
pesoNoCeros : Entero;
ceros : Entero;

ALGORITMO

Leer (t, long);
noCerosSLC ← 0;
pesoNoCeros ← 0;
ceros ← 0;
i **RECORRIENDO** [1, long] **HACER**
  SI $t_i = 0$ **ENTONCES**
    noCerosSLC ← noCerosSLC + pesoNoCeros;
    ceros ← ceros + 1
  SI_NO pesoNoCeros ← pesoNoCeros + ceros
  FIN_SI
FIN_RECORRIENDO;
Escribir (noCerosSLC)
FIN.

5.4 Tablas Multidimensionales

- El concepto de tabla puede generalizarse para contemplar tablas de varias dimensiones en las que cada valor lleva asociado uno o más índices.

- Si una tabla representa una función de un argumento, podemos entender que una *tabla multidimensional* representa una función de *n* argumentos, uno por cada dimensión.

- Las tablas multidimensionales se utilizan para representar colecciones de objetos de la misma naturaleza, a los que se puede acceder mediante un conjunto de índices.

- La declaración de una tabla multidimensional de *n* dimensiones debe incluir *n* intervalos de ordinales, I_i, $1 \leq i \leq n$, y el tipo base *Tb*.

  Nombre_tabla = TIPO TABLA [$\mathbf{I_1}$; $\mathbf{I_2}$; ...; $\mathbf{I_n}$] DE Tb

5.4 Tablas Multidimensionales

- Representar matrices de puntos:

matrizPuntos = **TIPO TABLA** [1, NFilas; 1, NColumas] **DE** Punto

- Un texto se puede representar mediante una tabla de caracteres:

libro = **TIPO TABLA** [1, NPaginas; 1, NLineas; 1, NColumnas] **DE** Carácter

- Para el registro diario de las cantidades de lluvia caída a lo largo del S. XXI:

lluvias = **TIPO TABLA** [2001, 2100; 1, 12; 1, 31] **DE** Real $\geq$ 0

5.4 Tablas Bidimensionales: Esquema de recorrido

Esquema de Recorrido

{ Tratamiento inicial externo }
i **RECORRIENDO** [1, nf] **HACER**
// INV_{ext} = Verdadero
    { Tratamiento inicial interno }
    j **RECORRIENDO** [1, nc] **HACER**
    // INV_{int} = Verdadero
        { Tratamiento del elemento actual" (i, j) }
    FIN_RECORRIENDO;
    { Tratamiento final interno }
FIN_RECORRIENDO;
{ Tratamiento final }

5.4 Tablas Bidimensionales: Esquema de búsqueda

Esquema de Búsqueda (tercer modelo)

```
i ← 1;   j ← 1;  // Comenzar
// Supondremos que nf > 0 y nc > 0                    // EsÚltimo

MIENTRAS NO (i = nf Y j = nc) Y NO Pro (t_i,j) HACER
    // INV { se han visitado i -1 filas y los j - 1 elementos de la fila i
    //          y ningún elemento visitado cumple la propiedad }
    SI j < nc ENTONCES
        j ← j +1
    SI_NO                                   // Avanzar
        i ← i +1; j ← 1
    FIN_SI
FIN_MIENTRAS;
SI Pro (t_i,j) ENTONCES
    { Tratamiento t_i,j cumple la propiedad }
SI_NO
    { Tratamiento propiedad no se cumple }
FIN_SI;
```

Ejemplo: Dada una matriz de enteros *T* de *nxn*, escriba un algoritmo que obtenga la suma de los valores mayores de cada fila.

- POST = { sm = suma de los valores mayores de cada fila de *T* }
- INVext = { sm = suma de los mayores de las primeras i - 1 filas de T, $1 \leq i \leq nf$ }
- INVint = { mayor = mayor de los j - 1 elementos ya recorridos de la fila i, $1 \leq j \leq nc$ }

"Tratamiento inicial externo": sm $\leftarrow$ 0;

"Tratamiento inicial interno": mayor $\leftarrow$ $t_{i,1}$;

"Tratamiento elemento actual": **SI** mayor < $t_{i,j}$ **ENTONCES** mayor $\leftarrow$ $t_{i,j}$; **FIN_SI**

"Tratamiento final interno": sm $\leftarrow$ sm + mayor;

"Tratamiento final": Escribir (sm);

Ejemplo: suma de valores mayores

LÉXICO

NFilas = n; // constante con el número de filas

NColumnas = m; // constante con el número de columnas

RangoFilas = **TIPO** Entero [1, NFilas];

RangoColumnas = **TIPO** Entero [1, NColumnas];

t : **TABLA** [RangoFilas; RangoColumnas] **DE** Entero;

nf, i : RangoFilas;
nc, j : RangoColumnas;
sm, mayor: Entero;

Ejemplo suma de valores mayores

ALGORITMO
Leer (t, nf, nc);
sm $\leftarrow$ 0;
i **RECORRIENDO** [1, nf] **HACER**
    mayor $\leftarrow t_{i,1}$;   *// Recorrido interior ¬H2*
    j **RECORRIENDO** [2, nc] **HACER**
        SI mayor < $t_{i,j}$ **ENTONCES** mayor $\leftarrow t_{i,j}$ **FIN_SI**
        // INVint = { mayor = mayor de los j elementos ya
        // recorridos de la fila i, 1 $\leq$ j $\leq$ nc }
    FIN_RECORRIENDO;
    sm $\leftarrow$ sm + mayor
    // INVext { sm = suma de los mayores de las primeras i filas de T, 1 $\leq$ i $\leq$ nf }
FIN_RECORRIENDO;
// INVext y i = nf $\Rightarrow$ POST
Escribir (sm)
FIN.

Búsqueda de la posición de un carácter dado en una tabla

```
PosEnTabla : función (m: MatrizCar; car: Carácter) → Posición;
 PRE {  m tabla de car de dimensiones  N * M}
 POST { retorna la posición (i, j) si  car está la tabla m, (0,0) si no está}
LÉXICO
    i : Entero [1, N]; j : Entero [1, M];
    res : Posición;
ALGORITMO
    i ← 1; j ← 1;
    MIENTRAS  NO ( (i = N) Y (j = M) )  Y  (m_i,j ≠ car) HACER
        SI j < M ENTONCES
                j ← j + 1
        SI_NO
                i ← i + 1; j ← 1
        FIN_SI
    FIN_MIENTRAS;
    SI m_i,j =  car  ENTONCES res ← < i, j >
    SI_NO res ← < 0, 0 > FIN_SI;
    PosEnTabla ← res
FIN;
```

5.5 Algoritmos de ordenación

- Clasificación. Proceso dirigido a clasificar una colección de elementos atendiendo a cierto criterio
- Sea A = $\{a_1, a_2, a_3, ..., a_n\}$, ordenar A significa aplicarle una función f cuyo resultado es una permutación de A
- Ejemplo

  Dada la colección de elementos:

  {(Luis, 29), (Ana, 45), (Víctor, 18), (Fernando, 24)}

  es posible ordenarla según dos atributos:

  {(Ana, 45), (Fernando, 24), (Luis, 29), (Víctor, 18)}

  {(Víctor, 18), (Fernando, 24), (Luis, 29), (Ana, 45)}
- Clasificación interna vs. externa
- Nos centraremos en los algoritmos de clasificación interna. (Tablas)
  - Directos ($T(n) \in O(n^2)$)
    - Inserción, Selección, Intercambio

5.5 Algoritmos de ordenación

- Las colecciones de elementos a clasificar se almacenarán en tablas

```
// Tipo
  TipoDatos = TIPO <
                      clave : TipoClave;
                      (* otros campos *)
                    >;
// Variable
  a: TABLA [TipoIndice] DE TipoDatos;
Por simplicidad
  TipoClave = Entero;
  TipoIndice = Entero [1, n];
```

- El campo *clave* es el atributo que elegimos para hacer la ordenación
- En el proceso de ordenación aplicado por los algoritmos de clasificación directos que estudiaremos, los elementos de la tabla cumplirán:

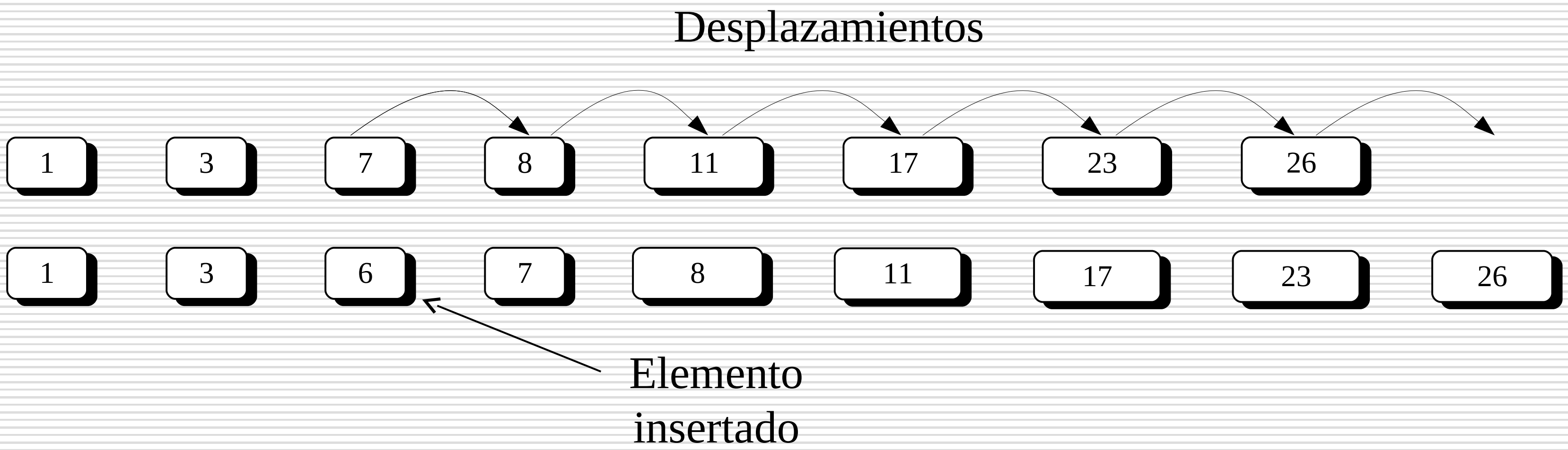
ORDENADA	DESORDENADA

5.5 Algoritmos de ordenación: inserción

- El método de ordenación por inserción consiste en **insertar** un elemento en la posición que le corresponde dentro de la parte ordenada
- Si la búsqueda de la posición de inserción se realiza mediante una **búsqueda lineal,** nos encontramos ante el algoritmo de **inserción directa**
- Si la búsqueda de la posición de inserción se efectúa mediante una **búsqueda binaria,** nos encontramos ante el algoritmo de **inserción binaria**

5.5 Algoritmos de ordenación: inserción

- Ejemplo, pretendemos insertar el 6 en la tabla (1, 3, 7, 8, 11, 17, 23, 26) y



- La operación de inserción se repite hasta llegar al último de la tabla.

5.5 Algoritmos de ordenación: inserción directa

- Se recorren los elementos de la tabla de tal forma que, en el paso q, los q **primeros elementos** de la tabla original están **ordenados**

Invariante de bucle:

$$\text{Ordenado}(q) \text{ y } \text{Permuta}(a, a', q)$$

donde a es el array inicial, a’ es el array en la iteración q, y

$\text{Ordenado}(q) = (\forall j, 1 <= j < q : a[j] <= a[j+1])$

$\text{Permuta}(a, a', q) = (\forall i, 1 <= i <= q :$

$$\text{Ocurr}(a, a[i], q) = \text{Ocurr}(a', a[i], q))$$

$\text{Ocurr}(t, e, q) = Nj: 1 <= j <= q: e=t[j]$

Postcondición:

$$\text{Ordenado}(n) \text{ y } \text{Permuta}(a, a', n)$$

5.5 Algoritmos de ordenación: inserción directa

- Ejemplo:

Sea a = (21, 15, 32, 8, 6, 30, 12, 25):

Array inicial	(21, 15, 32, 8, 6, 30, 12, 25)
	(***21***, 15, 32, 8, 6, 30, 12, 25)
q = 2	(***15***, **21**, 32, 8, 6, 30, 12, 25)
q = 3	(**15**, **21**, ***32***, 8, 6, 30, 12, 25)
q = 4	(***8***, **15**, **21**, **32**, 6, 30, 12, 25)
q = 5	(***6***, **8**, **15**, **21**, **32**, 30, 12, 25)
q = 6	(**6**, **8**, **15**, **21**, ***30***, **32**, 12, 25)
q = 7	(**6**, **8**, ***12***, **15**, **21**, **30**, **32**, 25)
q = 8	(**6**, **8**, **12**, **15**, **21**, ***25***, **30**, **32**)

5.5 Algoritmos de ordenación: inserción directa

- El algoritmo de inserción directa en notación SP:

LÉXICO
    TipoClave = ENTERO;
    TipoDatos = **TIPO** <clave : TipoClave; ...>
    TipoIndice = **TIPO** [**0**..n];
    q, j : TipoIndice;
    a : **TABLA** [TipoIndice] **DE** TipoDatos;
ALGORITMO
    q **RECORRIENDO** [2, n] **HACER**
        $a_0 \leftarrow a_q$; { Centinela }
        $j \leftarrow q - 1$;
        MIENTRAS a_0.clave < a_j.clave **HACER**
            $a_{j+1} \leftarrow a_j$;   { Desplazamiento }
            $j \leftarrow j - 1$;
        FIN_MIENTRAS;
        $a_{j+1} \leftarrow a_0$;
    FIN_RECORRIENDO
Fin.

5.5 Algoritmos de ordenación: inserción binaria

- Podemos mejorar el algoritmo de inserción empleando una **búsqueda binaria** para encontrar la posición de inserción
- El algoritmo de inserción binaria en notación SP:

LÉXICO

TipoClave = ENTERO;

TipoIndice = **TIPO** [1..n];

TipoDatos = **TIPO** <clave : TipoClave; ...>

i, j, inf, sup, med : TipoIndice;

x : TipoDatos;

5.5 Algoritmos de ordenación: inserción binaria

```
ALGORITMO
    i RECORRIENDO [2, n] HACER
        inf ← 1;
        sup ← i - 1;
        x ← a_i;
        MIENTRAS inf <= sup HACER
        (* Búsqueda *)
                med ← (inf + sup) DIV 2;
                SI x.clave < a_med.clave  ENTONCES sup ← med - 1
                SI_NO inf ← med + 1
                FIN_SI
        FIN_MIENTRAS;
        j RECORRIENDO [inf, i – 1] EN_SENTIDO_INVERSO HACER
                a_j+1 ← a_j; (* Desplazamiento *)
        FIN_RECORRIENDO;
        a_inf ← x;
    FIN_RECORRIENDO
```

5.5 Algoritmos de ordenación: selección directa

- Este método de ordenación consiste en seleccionar el menor elemento de la parte desordenada, e insertarlo en la cola de la parte ordenada
- Se recorren los elementos de la tabla de tal forma que, en el paso q, los q **primeros elementos** de la tabla están **ordenados** y son los q **menores** de toda la tabla

Invariante de bucle:

Ordenado(q) y Partición(q)

donde a es el array en la iteración q, y

Ordenado(q) = ($\forall$ j, 1 <= j < q : a[j] <= a[j+1])

Partición(q) = ($\forall$ k, j, (q < k <= n) y

(1 <= j <= q : a[k] >= a[j]))

Postcondición:

Ordenado(n) y Partición(n)

5.5 Algoritmos de ordenación: selección directa

- Ejemplo

Sea a = (21, 15, 32, 8, 6, 30, 12, 25):

Array inicial	(21, 15, 32, 8, <u>6</u>, 30, 12, 25)	Intercambio
q = 1	(**6**, 15, 32, <u>8</u>, 21, 30, 12, 25)	(21, 6)
q = 2	(**6**, **8**, 32, 15, 21, 30, <u>12</u>, 25)	(15, 8)
q = 3	(**6**, **8**, **12**, <u>15</u>, 21, 30, 32, 25)	(32, 12)
q = 4	(**6**, **8**, **12**, **15**, <u>21</u>, 30, 32, 25)	(15, 15)
q = 5	(**6**, **8**, **12**, **15**, **21**, 30, 32, <u>25</u>)	(21, 21)
q = 6	(**6**, **8**, **12**, **15**, **21**, **25**, 32, <u>30</u>)	(30, 25)
q = 7	(**6**, **8**, **12**, **15**, **21**, **25**, **30**, 32)	(32, 30)
	(**6**, **8**, **12**, **15**, **21**, ***25***, **30**, **32**)	

5.5 Algoritmos de ordenación: selección directa

- El algoritmo de selección directa en notación SP:

LÉXICO

pos, j, q : TipoIndice;
min : TipoDatos;

ALGORITMO

q **RECORRIENDO** [1, n – 1] **HACER**
pos ← q;
min ← a_q;
j **RECORRIENDO** [q+1, n] **HACER**
SI min.clave > a_j.clave **ENTONCES**
pos ← j;
min ← a_j;
FIN_SI
FIN_RECORRIENDO
a_{pos} ← a_q;
a_q ← min
FIN_RECORRIENDO