

Programación 2

**TAD Multiconjunto y
TAD Tabla**

(Función parcial, *Mapping*)

Multisets

Multisets

Lists

- Hay orden posicional de elementos
- Los elementos pueden repetirse

Sets

- No hay orden posicional de elementos
- Los elementos no se repiten

MultiSets

- No hay orden posicional de elementos
- Los elementos pueden repetirse

Ejemplo de Multiset: stock de productos

Relación entre Multisets y Permutaciones

Un TAD Multiset

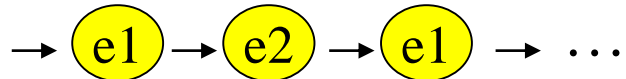
- **Vacio** m : construye el multiset m vacío;
- **Insertar x m : agrega x a m ;**
- **EsVacio** m : retorna true si y sólo si el multiset m está vacío;
- **Ocurrencias** x m : retorna la cantidad de veces que está x en m ;
- **Borrar** x m : elimina una ocurrencia de x en m , si x está en m ;
- **Destruir** m : destruye el multiset m , liberando su memoria.

Implementaciones

Multiset $m = \{(e_1, \#e_1), \dots, (e_i, \#e_i), \dots, (e_n, \#e_n)\}$

Adaptar y analizar para *multisets* las siguientes implementaciones vistas para conjuntos:

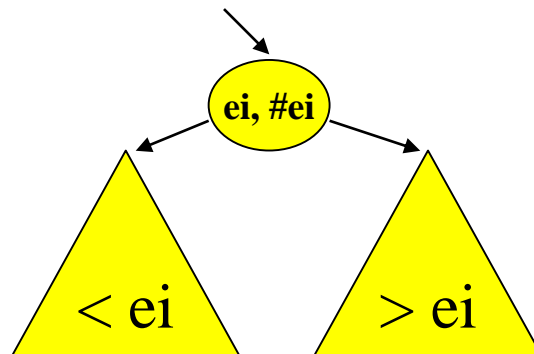
– Variantes de Listas



– Arreglos de Booleanos (ahora...)



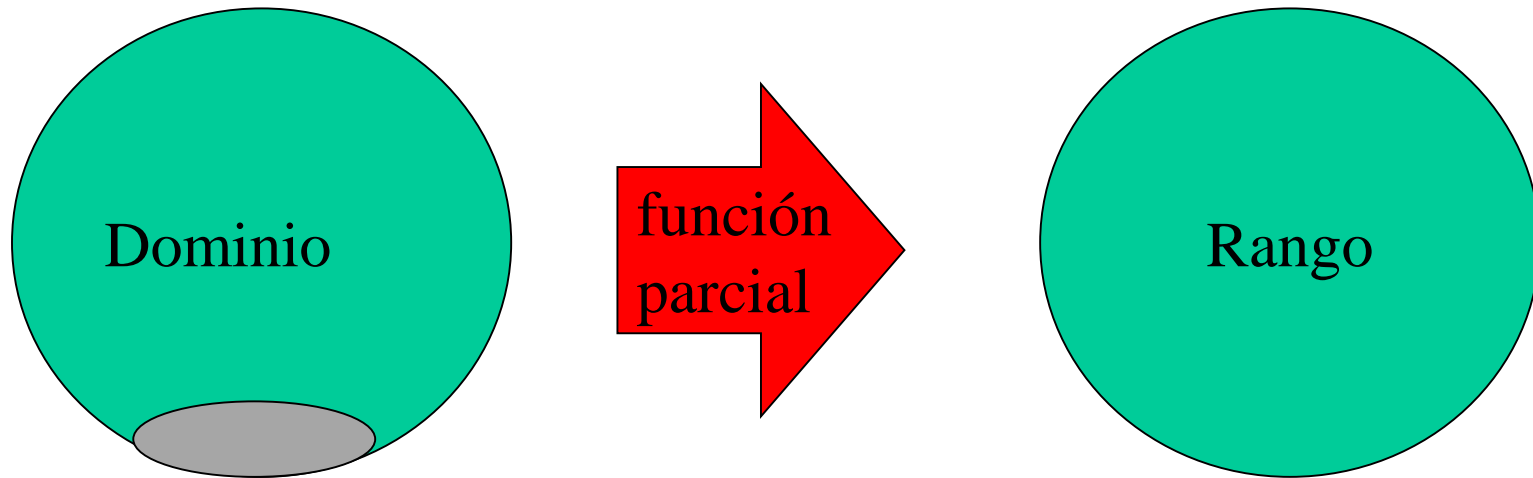
– ABBs



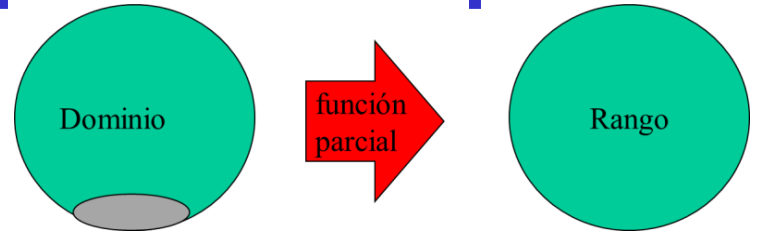
Tablas - Funciones Parciales (*Mappings*)

El TAD Tabla (Función parcial, *Mapping*)

Una **tabla** es una **función parcial** de elementos de un tipo, llamado el tipo dominio, a elementos de otro (posiblemente el mismo) tipo, llamado el tipo recorrido o rango o codominio.



TAD Tabla/Mapping. Operaciones para:



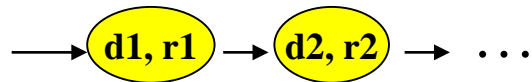
- construir una **tabla vacía**;
- **insertar** una correspondencia (d,r) en una tabla t . Si d está definida en t (tiene imagen), actualiza su correspondencia con r ;
- saber si una tabla **está vacía**;
- saber si un valor d **tiene imagen** en una tabla t ;
- **obtener la imagen** de un valor d (que existe) en una tabla t ;
- **eliminar** una correspondencia de una tabla, dado un valor del dominio.
- destruir una tabla.

Tabla $t \subseteq \text{Dominio} \times \text{Rango} = \{(d_1,r_1), \dots, (d_i,r_i), \dots, (d_n,r_n)\}$

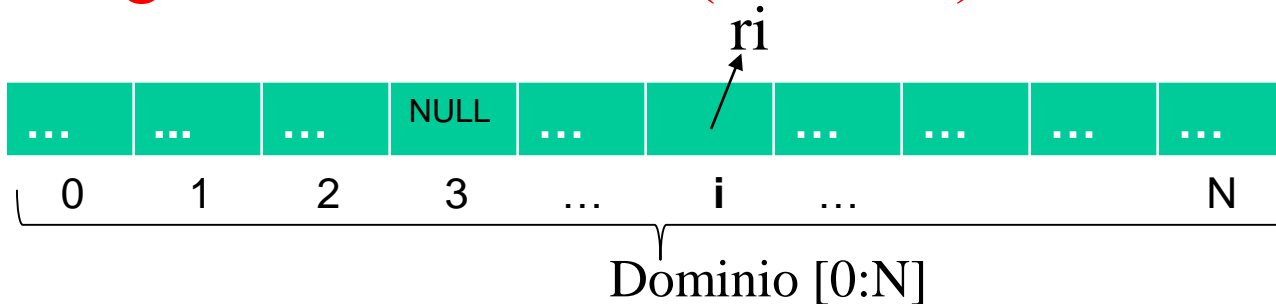
Implementaciones

- Adaptar y analizar para *mappings* las siguientes implementaciones vistas para conjuntos:

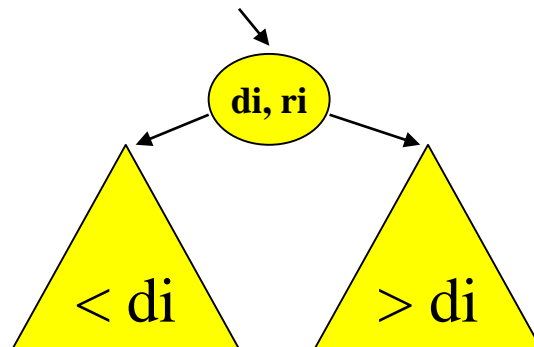
- Listas



- Arreglos de Booleanos (ahora...)



- ABBs



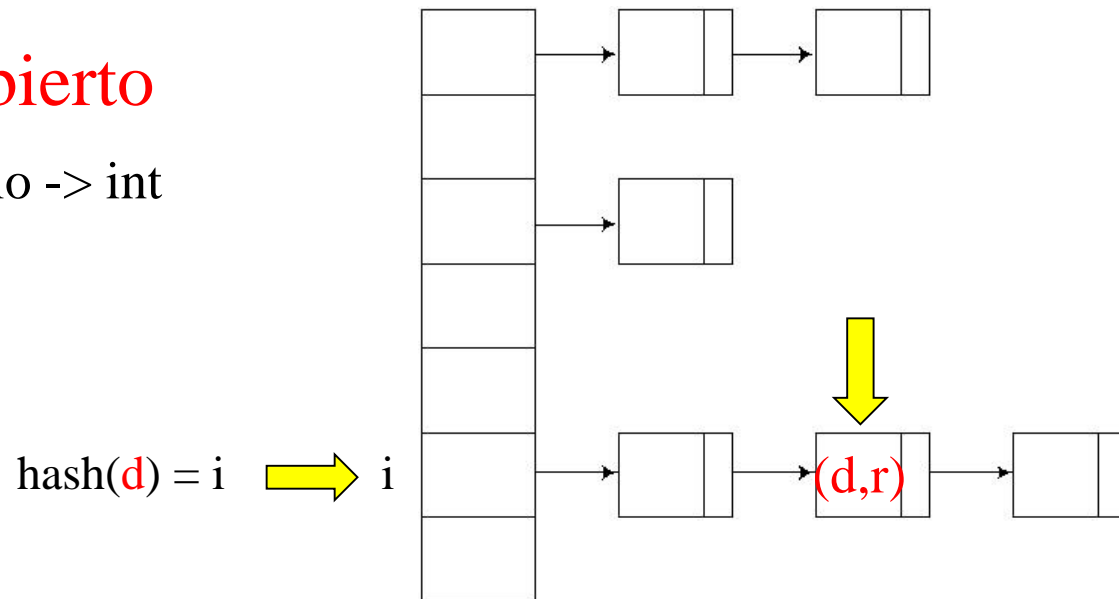
Implementaciones (cont)

- Adaptar y analizar para *mappings* las siguientes implementaciones vistas para conjuntos:
 - Arreglos con tope de pares (d,r) ordenado por dominio



- Hashing abierto

hash: Dominio \rightarrow int



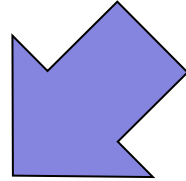
Insertar con Hashing

```
unsigned int hash (D d){return ...;}
```

```
\\ hash podría aplicarse a d y un entero M, realizando %M internamente
```

```
struct nodoHash{  
    D dom;  
    R ran;  
    nodoHash* sig;  
}
```

```
struct RepresentacionTabla{  
    nodoHash** tabla;  
    int cantidad;  
    int cota;  
}
```



insertar una correspondencia (d,r) en una tabla t. Si d está definida en t (tiene imagen), actualiza su correspondencia con r.

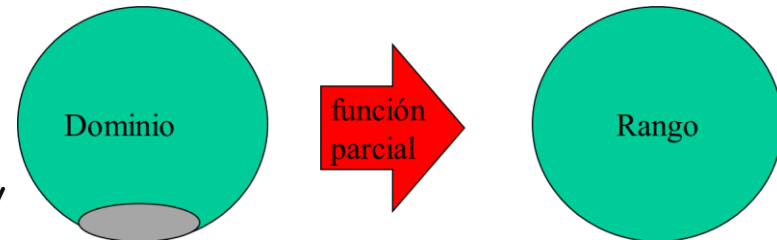
```
void insertarTabla (D d, R r, Tabla &t) {  
    int posicion = hash(d)%(t->cota); \\ hash podría aplicarse a d y t->cota...  
    nodoHash* lista = t->tabla[posicion];  
    while (lista!=NULL && lista->dom!=d)  
        lista = lista->sig;  
    if (lista==NULL){  
        nodoHash* nuevo = new nodoHash;  
        nuevo->dom = d;  
        nuevo->ran = r;  
        nuevo->sig = t->tabla[posicion];  
        t->tabla[posicion] = nuevo;  
        t->cantidad++;  
    }  
    else lista->ran = r;  
}
```

Especificación del TAD Tabla no acotada de D en R ($D \rightarrow R$)

```
#ifndef _TABLA_H
#define _TABLA_H
struct RepresentacionTabla;
typedef RepresentacionTabla * Tabla;

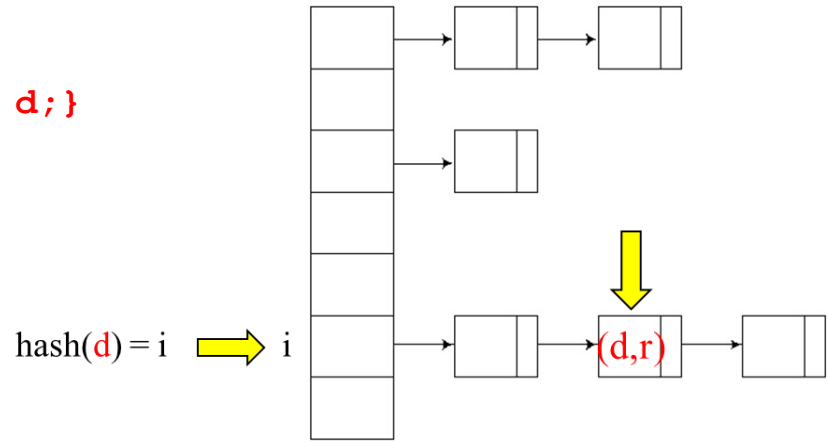
Tabla crearTabla (int cantidadEsperada);
// Devuelve la Tabla vacía no acotada, donde se estiman cantidadElementos.
void insertarTabla (D d, R r, Tabla &t);
/* Agrega la correspondencia (d,r) en t, si d no tenía imagen en t. En
   caso contrario actualiza la imagen de d con r. */
bool estaDefinidaTabla (D d, Tabla t);
// Devuelve true si y sólo si d tiene imagen en t.
bool esVaciaTabla (Tabla t);
// Devuelve true si y sólo si t es vacía.
R recuperarTabla (D d, Tabla t);
/* Retorna la imagen de d en t.
   Precondición: estaDefinidaTabla(d,t). */
void eliminarTabla (D d, Tabla &t);
/* Elimina de t la correspondencia que involucra a d, si d está definida
   en t. En caso contrario la operación no tiene efecto. */
int cantidadEnTabla (Tabla &t);
// retorna la cantidad de correspondencias (d,r) en t.
Tabla copiarTabla (Tabla t);
// retorna una copia de t sin compartir memoria.
void destruirTabla (Tabla &t);
// Libera toda la memoria ocupada por t.

#endif /* _Tabla_H */
```



Implementación de un Tabla no acotada de D en R (D→R) con hashing abierto

```
#include ...
#include "Tabla.h"
int hash (D d){ return ... }
\\unsigned int hash (unsigned int d){return d;}
struct nodoHash{
    D dom;
    R ran;
    nodoHash* sig;
}
struct RepresentacionTabla{
    nodoHash** tabla;
    int cantidad;
    int cota;
}
Tabla crearTabla (int cantidadEsperada) {
    Tabla t = new RepresentacionTabla();
    t->tabla = new nodoHash* [cantidadEsperada];
    for (int i=0; i<cantidadEsperada; i++) t->tabla[i]=NULL;
    t->cantidad = 0;
    t->cota = cantidadEsperada;
    return t;
}
```

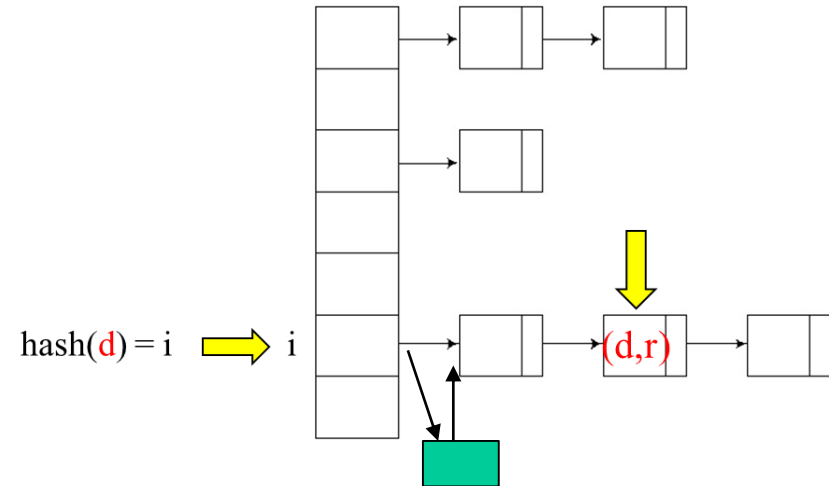


Implementación de Tabla acotado de D en R (D→R) con hashing abierto

```
void insertarTabla (D d, R r, Tabla &t) {  
    int posicion = hash(d) % (t->cota);  
    nodoHash* lista = t->tabla[posicion];  
    while (lista!=NULL && lista->dom!=d)  
        lista = lista->sig;  
    if (lista==NULL){  
        nodoHash* nuevo = new nodoHash;  
        nuevo->dom = d;  
        nuevo->ran = r;  
        nuevo->sig = t->tabla[posicion];  
        t->tabla[posicion] = nuevo;  
        t->cantidad++;  
    }  
    else lista->ran = r;  
}
```

```
bool estaDefinidaTabla (D d, Tabla t) {  
    int posicion = hash(d) % (t->cota);  
    nodoHash* lista = t->tabla[posicion];  
    while (lista!=NULL && lista->dom!=d)  
        lista = lista->sig;  
    return lista!=NULL;  
}
```

```
bool esVacíaTabla (Tabla t) { return t->cantidad==0; }  
...
```



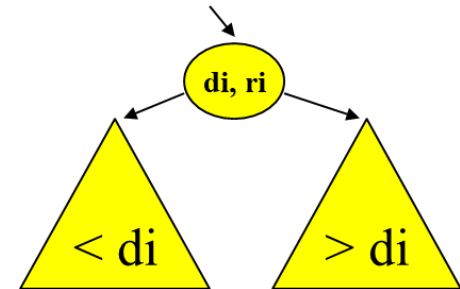
Implementación de un Tabla no acotada de D en R ($D \rightarrow R$) con un ABB

```
#include ...
#include "Tabla.h"

struct nodoABB{
    D dom;
    R ran;
    nodoABB* izq;
    nodoABB* der;
};

struct RepresentacionTabla{
    nodoABB* abb;
    int cantidad;
};

Tabla crearTabla (int cantidadEsperada) {
    Tabla t = new RepresentacionTabla();
    t->abb = NULL;
    t->cantidad = 0;
    return t;
}
```



Implementación de Tabla acotado de D en R ($D \rightarrow R$) con un ABB

/* Inserta en el ABB la pareja (d,r); si d está en a actualiza el r y devuelve true. En caso contrario retorna false. */

```
bool insertarABB (D d, R r, nodoABB* & a){
    if (a==NULL){
        a = new nodoABB;
        a->dom = d; a->ran = r;
        a->izq = a->der = NULL;
        return false;
    }
    else if (d == a->dom){
        a->ran = r;
        return true;
    }
    else if (d < a->dom) return insertarABB(d, r, a->izq);
    else return insertarABB(d, r, a->der);
}
```

```
void insertarTabla (D d, R r, Tabla &t) {
    if (!insertarABB(d, r, t->abb))
        t->cantidad++;
}
```

