# Tema 18: El TAD de las tablas

## Informática (2016–17)

José A. Alonso Jiménez

Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

# Tema 18: El TAD de las tablas

# Tema 18: El TAD de las tablas

# La clase de los índices de las tablas

- La clase de los índices de las tablas es `Ix`.
- `Ix` se encuentra en la librería `Data.Ix`
- Información de la clase `Ix`:

```
ghci> :info Ix
class (Ord a) => Ix a where
  range :: (a, a) -> [a]
  index :: (a, a) -> a -> Int
  inRange :: (a, a) -> a -> Bool
  rangeSize :: (a, a) -> Int
instance Ix Ordering -- Defined in GHC.Arr
instance Ix Integer -- Defined in GHC.Arr
instance Ix Int -- Defined in GHC.Arr
instance Ix Char -- Defined in GHC.Arr
instance Ix Bool -- Defined in GHC.Arr
instance (Ix a, Ix b) => Ix (a, b)
```

# La clase de los índices de las tablas

- `(range (m,n))` es la lista de los índices desde `m` hasta `n`, en el orden del índice. Por ejemplo,

```
range (0,4)           ⇝  [0,1,2,3,4]
range (3,9)           ⇝  [3,4,5,6,7,8,9]
range ('b','f')       ⇝  "bcdef"
range ((0,0),(1,2))   ⇝  [(0,0),(0,1),(0,2),
                              (1,0),(1,1),(1,2)]
```

- `(index (m,n) i)` es el ordinal del índice `i` dentro del rango `(m,n)`. Por ejemplo,

```
index (3,9) 5               ⇝  2
index ('b','f') 'e'         ⇝  3
index ((0,0),(1,2)) (1,1)   ⇝  4
```

# La clase de los índices de las tablas

- `(inRange (m,n) i)` se verifica si el índice `i` está dentro del rango limitado por `m` y `n`. Por ejemplo,

```
inRange (0,4) 3                 ⇝  True
inRange (0,4) 7                 ⇝  False
inRange ((0,0),(1,2)) (1,1)     ⇝  True
inRange ((0,0),(1,2)) (1,5)     ⇝  False
```

- `(rangeSize (m,n))` es el número de elementos en el rango limitado por `m` y `n`. Por ejemplo,

```
rangeSize (3,9)             ⇝  7
rangeSize ('b','f')         ⇝  5
rangeSize ((0,0),(1,2))     ⇝  6
```

# Tema 18: El TAD de las tablas

# El tipo predefinido de las tablas ("arrays")

- La librería de las tablas es `Data.Array`.
- Para usar las tablas hay que escribir al principio del fichero

```
import Data.Array
```

- Al importar `Data.Array` también se importa `Data.Ix`.
- `(Array i v)` es el tipo de las tablas con índice en `i` y valores en `v`.

# Creación de tablas

- `(array (m,n) ivs)` es la tabla de índices en el rango limitado por `m` y `n` definida por la lista de asociación `ivs` (cuyos elementos son pares de la forma (índice, valor)). Por ejemplo,

```
ghci> array (1,3) [(3,6),(1,2),(2,4)]
array (1,3) [(1,2),(2,4),(3,6)]
ghci> array (1,3) [(i,2*i) | i <- [1..3]]
array (1,3) [(1,2),(2,4),(3,6)]
```

# Ejemplos de definiciones de tablas

- `(cuadrados n)` es un vector de n+1 elementos tal que su elemento i–ésimo es $i^2$. Por ejemplo,

```
ghci> cuadrados 5
array (0,5) [(0,0),(1,1),(2,4),(3,9),(4,16),(5,25)]
```

```
cuadrados :: Int -> Array Int Int
cuadrados n = array (0,n) [(i,i^2) | i <- [0..n]]
```

- `v` es un vector con 4 elementos de tipo carácter. Por ejemplo,

```
v :: Array Integer Char
v = array (1,4) [(3,'c'),(2,'a'), (1,'f'), (4,'e')]
```

# Ejemplos de definiciones de tablas

- `m` es la matriz con 2 filas y 3 columnas tal que el elemento de la posición (i,j) es el producto de i por j.

```
m :: Array (Int, Int) Int
m = array ((1,1),(2,3)) [((i,j),i*j)) | i<-[1..2],j<-[1..3
```

- Una tabla está indefinida si algún índice está fuera de rango.

```
ghci> array (1,4) [(i , i*i) | i <- [1..4]]
array (1,4) [(1,1),(2,4),(3,9),(4,16)]
ghci> array (1,4) [(i , i*i) | i <- [1..5]]
array *** Exception: Error in array index
ghci> array (1,4) [(i , i*i) | i <- [1..3]]
array (1,4) [(1,1),(2,4),(3,9),(4,***
Exception: (Array.!): undefined array element
```

# Descomposición de tablas

- `(t ! i)` es el valor del índice `i` en la tabla `t`. Por ejemplo,

```
ghci> v
array (1,4) [(1,'f'),(2,'a'),(3,'c'),(4,'e')]
ghci> v!3
'c'
ghci> m
array ((1,1),(2,3)) [((1,1),1),((1,2),2),((1,3),3),
                     ((2,1),2),((2,2),4),((2,3),6)]
ghci> m!(2,3)
6
```

- `(bounds t)` es el rango de la tabla `t`. Por ejemplo,

```
bounds m  ⇝  ((1,1),(2,3))
```

- `(indices t)` es la lista de los índices de la tabla `t`. Por ejemplo,

```
indices m  ⇝  [(1,1),(1,2),(1,3),(2,1),(2,2),(2,3)]
```

# Descomposición de tablas

- `(elems t)` es la lista de los elementos de la tabla `t`. Por ejemplo,

```
elems m  ⇝  [1,2,3,2,4,6]
```

- `(assocs t)` es la lista de asociaciones de la tabla `t`. Por ejemplo,

```
ghci> assocs m
[((1,1),1),((1,2),2),((1,3),3),
 ((2,1),2),((2,2),4),((2,3),6)]
```

# Modificación de tablas

- `(t // ivs)` es la tabla `t` asignándole a los índices de la lista de asociación `ivs` sus correspondientes valores. Por ejemplo,

```
ghci> m // [((1,1),4), ((2,2),8)]
array ((1,1),(2,3))
      [((1,1),4),((1,2),2),((1,3),3),
       ((2,1),2),((2,2),8),((2,3),6)]
ghci> m
array ((1,1),(2,3))
      [((1,1),1),((1,2),2),((1,3),3),
       ((2,1),2),((2,2),4),((2,3),6)]
```

# Definición de tabla por recursión

- `(fibs n)` es el vector formado por los `n` primeros términos de la sucesión de Fibonacci. Por ejemplo,

```
ghci> fibs 7
array (0,7) [(0,1),(1,1),(2,2),(3,3),
             (4,5),(5,8),(6,13),(7,21)]
```

```
fibs :: Int -> Array Int Int
fibs n = a where
    a = array (0,n)
              ([(0,1),(1,1)] ++
               [(i,a!(i-1)+a!(i-2)) | i <- [2..n]])
```

# Otras funciones de creación de tablas

- `(listArray (m,n) vs)` es la tabla cuyo rango es `(m,n)` y cuya lista de valores es `vs`. Por ejemplo,

```
ghci> listArray (2,5) "Roma"
array (2,5) [(2,'R'),(3,'o'),(4,'m'),(5,'a')]
ghci> listArray ((1,2),(2,4)) [5..12]
array ((1,2),(2,4)) [((1,2),5),((1,3),6),((1,4),7),
                     ((2,2),8),((2,3),9),((2,4),10)]
```

# Construcción acumulativa de tablas

- `(accumArray f v (m,n) ivs)` es la tabla de rango `(m,n)` tal que el valor del índice `i` se obtiene acumulando la aplicación de la función `f` al valor inicial `v` y a los valores de la lista de asociación `ivs` cuyo índice es `i`. Por ejemplo,

```
ghci> accumArray (+) 0 (1,3) [(1,4),(2,5),(1,2)]
array (1,3) [(1,6),(2,5),(3,0)]
ghci> accumArray (*) 1 (1,3) [(1,4),(2,5),(1,2)]
array (1,3) [(1,8),(2,5),(3,1)]
```

# Construcción acumulativa de tablas

- `(histograma r is)` es el vector formado contando cuantas veces aparecen los elementos del rango r en la lista de índices is. Por ejemplo,

```
ghci> histograma (0,5) [3,1,4,1,5,4,2,7]
array (0,5) [(0,0),(1,2),(2,1),(3,1),(4,2),(5,1)]
```

```
histograma :: (Ix a, Num b) => (a,a) -> [a] -> Array a b
histograma r is =
    accumArray (+) 0 r [(i,1) | i <- is, inRange r i]
```

# Tema 18: El TAD de las tablas

# Signatura del TAD de las tablas

- Signatura:

```
tabla    :: Eq i => [(i,v)] -> Tabla i v
valor    :: Eq i => Tabla i v -> i -> v
modifica :: Eq i => (i,v) -> Tabla i v -> Tabla i v
```

- Descripción de las operaciones:
  - `(tabla ivs)` es la tabla correspondiente a la lista de asociación `ivs` (que es una lista de pares formados por los índices y los valores).
  - `(valor t i)` es el valor del índice `i` en la tabla `t`.
  - `(modifica (i,v) t)` es la tabla obtenida modificando en la tabla `t` el valor de `i` por `v`.

# Tema 18: El TAD de las tablas

# Propiedades del TAD de las tablas

1. `modifica (i,v') (modifica (i,v) t)`
   `= modifica (i,v') t`
2. Si `i /= i'`, entonces
   `modifica (i',v') (modifica (i,v) t)`
   `= modifica (i,v) (modifica (i',v') t)`
3. `valor (modifica (i,v) t) i = v`
4. Si `i /= i'`, entonces
   `valor (modifica (i,v) (modifica (k',v') t)) i'`
   `= valor (modifica (k',v') t) i'`

# Tema 18: El TAD de las tablas

# Las tablas como funciones

- Cabecera del módulo:

```
module TablaConFunciones
    (Tabla,
     tabla,    -- Eq i => [(i,v)] -> Tabla i v
     valor,    -- Eq i => Tabla i v -> i -> v
     modifica -- Eq i => (i,v) -> Tabla i v -> Tabla i v
    ) where
```

- Las tablas como funciones.

```
newtype Tabla i v = Tbl (i -> v)
```

# Las tablas como funciones

- Procedimiento de escritura.

```
instance Show (Tabla i v) where
    showsPrec _ _ cad = showString "<<Una tabla>>" cad
```

- Ejemplos de tablas:

```
t1 = tabla [(i,f i) | i <- [1..6] ]
     where f x | x < 3     = x
               | otherwise = 3-x

t2 = tabla [(4,89), (1,90), (2,67)]
```

# Las tablas como funciones

- `(valor t i)` es el valor del índice `i` en la tabla `t`. Por ejemplo,

```
valor t1 6  ⇝  -3
valor t2 2  ⇝  67
valor t2 5  ⇝  *** Exception: fuera de rango
```

```
valor :: Eq i => Tabla i v -> i -> v
valor (Tbl f) i = f i
```

- `(modifica (i,v) t)` es la tabla obtenida modificando en la tabla `t` el valor de `i` por `v`. Por ejemplo,

```
valor (modifica (6,9) t1) 6  ⇝  9
```

```
modifica :: Eq i => (i,v) -> Tabla i v -> Tabla i v
modifica (i,v) (Tbl f) = Tbl g
    where g j | j == i    = v
              | otherwise = f j
```

# Las tablas como funciones

- `(valor t i)` es el valor del índice `i` en la tabla `t`. Por ejemplo,

```
valor t1 6  ⇝  -3
valor t2 2  ⇝  67
valor t2 5  ⇝  *** Exception: fuera de rango
```

```
valor :: Eq i => Tabla i v -> i -> v
valor (Tbl f) i = f i
```

- `(modifica (i,v) t)` es la tabla obtenida modificando en la tabla `t` el valor de `i` por `v`. Por ejemplo,

```
valor (modifica (6,9) t1) 6  ⇝  9
```

```
modifica :: Eq i => (i,v) -> Tabla i v -> Tabla i v
modifica (i,v) (Tbl f) = Tbl g
    where g j | j == i    = v
              | otherwise = f j
```

# Las tablas como funciones

- `(valor t i)` es el valor del índice `i` en la tabla `t`. Por ejemplo,

```
valor t1 6  ⇝  -3
valor t2 2  ⇝  67
valor t2 5  ⇝  *** Exception: fuera de rango
```

```
valor :: Eq i => Tabla i v -> i -> v
valor (Tbl f) i = f i
```

- `(modifica (i,v) t)` es la tabla obtenida modificando en la tabla `t` el valor de `i` por `v`. Por ejemplo,

```
valor (modifica (6,9) t1) 6  ⇝  9
```

```
modifica :: Eq i => (i,v) -> Tabla i v -> Tabla i v
modifica (i,v) (Tbl f) = Tbl g
    where g j | j == i    = v
              | otherwise = f j
```

# Las tablas como funciones

- `(tabla ivs)` es la tabla correspondiente a la lista de asociación `ivs` (que es una lista de pares formados por los índices y los valores). Por ejemplo,

```
ghci> tabla [(4,89), (1,90), (2,67)]
<<Una tabla>>
```

```
tabla :: Eq i => [(i,v)] -> Tabla i v
tabla ivs =
    foldr modifica
          (Tbl (\_ -> error "fuera de rango"))
          ivs
```

# Las tablas como funciones

- `(tabla ivs)` es la tabla correspondiente a la lista de asociación `ivs` (que es una lista de pares formados por los índices y los valores). Por ejemplo,

```
ghci> tabla [(4,89), (1,90), (2,67)]
<<Una tabla>>
```

```
tabla :: Eq i => [(i,v)] -> Tabla i v
tabla ivs =
    foldr modifica
          (Tbl (\_ -> error "fuera de rango"))
          ivs
```

# Tema 18: El TAD de las tablas

# Las tablas como listas de asociación

- Cabecera del módulo

```
module TablaConListasDeAsociacion
    (Tabla,
     tabla,    -- Eq i => [(i,v)] -> Tabla i v
     valor,    -- Eq i => Tabla i v -> i -> v
     modifica -- Eq i => (i,v) -> Tabla i v -> Tabla i v
    ) where
```

- Las tablas como listas de asociación.

```
newtype Tabla i v = Tbl [(i,v)]
    deriving Show
```

# Las tablas como listas de asociación

- Ejemplos de tablas
  - Definición:

```
t1 = tabla [(i,f i) | i <- [1..6] ]
     where f x | x < 3     = x
               | otherwise = 3-x

t2 = tabla [(4,89), (1,90), (2,67)]
```

  - Evaluación:

```
ghci> t1
Tbl [(1,1),(2,2),(3,0),(4,-1),(5,-2),(6,-3)]

ghci> t2
Tbl [(4,89),(1,90),(2,67)]
```

# Las tablas como listas de asociación

- `(tabla ivs)` es la tabla correspondiente a la lista de asociación `ivs` (que es una lista de pares formados por los índices y los valores). Por ejemplo,

```
ghci> tabla [(4,89),(1,90),(2,67)]
Tbl [(4,89),(1,90),(2,67)]
```

```
tabla :: Eq i => [(i,v)] -> Tabla i v
tabla ivs = Tbl ivs
```

# Las tablas como listas de asociación

- `(tabla ivs)` es la tabla correspondiente a la lista de asociación `ivs` (que es una lista de pares formados por los índices y los valores). Por ejemplo,

```
ghci> tabla [(4,89),(1,90),(2,67)]
Tbl [(4,89),(1,90),(2,67)]
```

```
tabla :: Eq i => [(i,v)] -> Tabla i v
tabla ivs = Tbl ivs
```

# Las tablas como listas de asociación

- `(valor t i)` es el valor del índice `i` en la tabla `t`. Por ejemplo,

```
valor t1 6  ⇝  -3
valor t2 2  ⇝  67
valor t2 5  ⇝  *** Exception: fuera de rango
```

```
valor :: Eq i => Tabla i v -> i -> v
valor (Tbl []) i = error "fuera de rango"
valor (Tbl ((j,v):r)) i
    | i == j    = v
    | otherwise = valor (Tbl r) i
```

# Las tablas como listas de asociación

- `(valor t i)` es el valor del índice `i` en la tabla `t`. Por ejemplo,

```
valor t1 6  ⇝  -3
valor t2 2  ⇝  67
valor t2 5  ⇝  *** Exception: fuera de rango
```

```
valor :: Eq i => Tabla i v -> i -> v
valor (Tbl []) i = error "fuera de rango"
valor (Tbl ((j,v):r)) i
     | i == j    = v
     | otherwise = valor (Tbl r) i
```

# Las tablas como listas de asociación

- `(modifica (i,x) t)` es la tabla obtenida modificando en la tabla `t` el valor de `i` por `x`. Por ejemplo,

```
valor t1 6                          ⇝  -3
valor (modifica (6,9) t1) 6         ⇝  9
```

```
modifica :: Eq i => (i,v) -> Tabla i v -> Tabla i v
modifica p (Tbl []) = (Tbl [p])
modifica p'@(i,_) (Tbl (p@(j,_):r))
    | i == j     = Tbl (p':r)
    | otherwise  = Tbl (p:r')
    where Tbl r' = modifica p' (Tbl r)
```

# Las tablas como listas de asociación

- `(modifica (i,x) t)` es la tabla obtenida modificando en la tabla `t` el valor de `i` por `x`. Por ejemplo,

```
valor t1 6                          ⇝  -3
valor (modifica (6,9) t1) 6         ⇝  9
```

```
modifica :: Eq i => (i,v) -> Tabla i v -> Tabla i v
modifica p (Tbl []) = (Tbl [p])
modifica p'@(i,_) (Tbl (p@(j,_):r))
     | i == j      = Tbl (p':r)
     | otherwise   = Tbl (p:r')
     where Tbl r' = modifica p' (Tbl r)
```

# Tema 18: El TAD de las tablas

# Las tablas como matrices

- Cabecera del módulo:

```
module TablaConMatrices
    (Tabla,
     tabla,      -- Eq i => [(i,v)] -> Tabla i v
     valor,      -- Eq i => Tabla i v -> i -> v
     modifica,   -- Eq i => (i,v) -> Tabla i v -> Tabla i v
     tieneValor -- Ix i => Tabla i v -> i -> Bool
    ) where
```

- Importación de la librería auxiliar:

```
import Data.Array
```

- Las tablas como matrices.

```
newtype Tabla i v = Tbl (Array i v) deriving (Show, Eq)
```

# Las tablas como matrices

- Ejemplos de tablas:
  - Definición:

```
t1 = tabla [(i,f i) | i <- [1..6] ]
     where f x | x < 3      = x
               | otherwise = 3-x

t2 = tabla [(1,5),(2,4),(3,7)]
```

  - Evaluación:

```
ghci> t1
Tbl (array (1,6) [(1,1),(2,2),(3,0),
                  (4,-1),(5,-2),(6,-3)])

ghci> t2
Tbl (array (1,3) [(1,5),(2,4),(3,7)])
```

# Las tablas como matrices

- `(tabla ivs)` es la tabla correspondiente a la lista de asociación `ivs` (que es una lista de pares formados por los índices y los valores). Por ejemplo,

```
ghci> tabla [(1,5),(3,7),(2,4)]
Tbl (array (1,3) [(1,5),(2,4),(3,7)])
```

```
tabla :: Ix i => [(i,v)] -> Tabla i v
tabla ivs = Tbl (array (m,n) ivs)
    where indices = [i | (i,_) <- ivs]
          m       = minimum indices
          n       = maximum indices
```

# Las tablas como matrices

- `(tabla ivs)` es la tabla correspondiente a la lista de asociación `ivs` (que es una lista de pares formados por los índices y los valores). Por ejemplo,

```
ghci> tabla [(1,5),(3,7),(2,4)]
Tbl (array (1,3) [(1,5),(2,4),(3,7)])
```

```
tabla :: Ix i => [(i,v)] -> Tabla i v
tabla ivs = Tbl (array (m,n) ivs)
    where indices = [i | (i,_) <- ivs]
          m       = minimum indices
          n       = maximum indices
```

# Las tablas como matrices

- `(valor t i)` es el valor del índice `i` en la tabla `t`. Por ejemplo,

```
valor t1 6  ⇝  -3
valor t2 2  ⇝  67
valor t2 5  ⇝  *** Exception: fuera de rango
```

```
valor :: Ix i => Tabla i v -> i -> v
valor (Tbl t) i = t ! i
```

- `(modifica (i,x) t)` es la tabla obtenida modificando en la tabla `t` el valor de `i` por `x`. Por ejemplo,

```
valor t1 6                        ⇝  -3
valor (modifica (6,9) t1) 6  ⇝  9
```

```
modifica :: Ix i => (i,v) -> Tabla i v -> Tabla i v
modifica p (Tbl t) = Tbl (t // [p])
```

# Las tablas como matrices

- `(valor t i)` es el valor del índice `i` en la tabla `t`. Por ejemplo,

```
valor t1 6  ⇝  -3
valor t2 2  ⇝  67
valor t2 5  ⇝  *** Exception: fuera de rango
```

```
valor :: Ix i => Tabla i v -> i -> v
valor (Tbl t) i = t ! i
```

- `(modifica (i,x) t)` es la tabla obtenida modificando en la tabla `t` el valor de `i` por `x`. Por ejemplo,

```
valor t1 6                          ⇝  -3
valor (modifica (6,9) t1) 6  ⇝  9
```

```
modifica :: Ix i => (i,v) -> Tabla i v -> Tabla i v
modifica p (Tbl t) = Tbl (t // [p])
```

# Las tablas como matrices

- `(valor t i)` es el valor del índice `i` en la tabla `t`. Por ejemplo,

```
valor t1 6  ⇝  -3
valor t2 2  ⇝  67
valor t2 5  ⇝  *** Exception: fuera de rango
```

```
valor :: Ix i => Tabla i v -> i -> v
valor (Tbl t) i = t ! i
```

- `(modifica (i,x) t)` es la tabla obtenida modificando en la tabla `t` el valor de `i` por `x`. Por ejemplo,

```
valor t1 6                          ⇝  -3
valor (modifica (6,9) t1) 6  ⇝  9
```

```
modifica :: Ix i => (i,v) -> Tabla i v -> Tabla i v
modifica p (Tbl t) = Tbl (t // [p])
```

# Las tablas como matrices

- `(cotas t)` son las cotas de la tabla `t`. Por ejemplo,

```
t2          ⇝  Tbl (array (1,3) [(1,5),(2,4),(3,7)])
cotas t2    ⇝  (1,3)
```

```
cotas :: Ix i => Tabla i v -> (i,i)
cotas (Tbl t) = bounds t
```

- `(tieneValor t x)` se verifica si `x` es una clave de la tabla `t`.

Por ejemplo,

```
tieneValor t2 3   ⇝  True
tieneValor t2 4   ⇝  False
```

```
tieneValor :: Ix i => Tabla i v -> i -> Bool
tieneValor t = inRange (cotas t)
```

# Las tablas como matrices

- `(cotas t)` son las cotas de la tabla `t`. Por ejemplo,

```
t2          ⇝  Tbl (array (1,3) [(1,5),(2,4),(3,7)])
cotas t2    ⇝  (1,3)
```

```
cotas :: Ix i => Tabla i v -> (i,i)
cotas (Tbl t) = bounds t
```

- `(tieneValor t x)` se verifica si `x` es una clave de la tabla `t`. Por ejemplo,

```
tieneValor t2 3   ⇝  True
tieneValor t2 4   ⇝  False
```

```
tieneValor :: Ix i => Tabla i v -> i -> Bool
tieneValor t = inRange (cotas t)
```

# Las tablas como matrices

- `(cotas t)` son las cotas de la tabla `t`. Por ejemplo,

```
t2          ⇝  Tbl (array (1,3) [(1,5),(2,4),(3,7)])
cotas t2    ⇝  (1,3)
```

```
cotas :: Ix i => Tabla i v -> (i,i)
cotas (Tbl t) = bounds t
```

- `(tieneValor t x)` se verifica si `x` es una clave de la tabla `t`. Por ejemplo,

```
tieneValor t2 3   ⇝  True
tieneValor t2 4   ⇝  False
```

```
tieneValor :: Ix i => Tabla i v -> i -> Bool
tieneValor t = inRange (cotas t)
```

# Tema 18: El TAD de las tablas

# Comprobación de las propiedades de las tablas

- Importación de la implementación de las tablas que se desea verificar.

```
import TablaConListasDeAsociacion
```

- Importación de las librerías de comprobación.

```
import Test.QuickCheck
import Test.Framework
import Test.Framework.Providers.QuickCheck2
```

# Tema 18: El TAD de las tablas

# Generador de tablas

- `genTabla` es un generador de tablas. Por ejemplo,

```
ghci> sample genTabla
Tbl [(1,0)]
Tbl [(1,-1)]
Tbl [(1,0),(2,-1),(3,1),(4,1),(5,0)]
```

```
genTabla :: Gen (Tabla Int Int)
genTabla =
    do x <- arbitrary
       xs <- listOf arbitrary
       return (tabla (zip [1..] (x:xs)))

instance Arbitrary (Tabla Int Int) where
    arbitrary = genTabla
```

# Tema 18: El TAD de las tablas

# Especificación de las propiedades de las tablas

- Al modificar una tabla dos veces con la misma clave se obtiene el mismos resultado que modificarla una vez con el último valor.

```
prop_modifica_modifica_1 :: Int -> Int -> Int
                         -> Tabla Int Int -> Bool
prop_modifica_modifica_1 i v v' t =
    modifica (i,v') (modifica (i,v) t)
    == modifica (i,v') t
```

# Especificación de las propiedades de las tablas

- Al modificar una tabla dos veces con la misma clave se obtiene el mismos resultado que modificarla una vez con el último valor.

```
prop_modifica_modifica_1 :: Int -> Int -> Int
                         -> Tabla Int Int -> Bool
prop_modifica_modifica_1 i v v' t =
    modifica (i,v') (modifica (i,v) t)
    == modifica (i,v') t
```

# Especificación de las propiedades de las tablas

- Al modificar una tabla con dos pares con claves distintas no importa el orden en que se añadan los pares.

```
prop_modifica_modifica_2 :: Int -> Int -> Int -> Int
                         -> Tabla Int Int -> Property
prop_modifica_modifica_2 i i' v v' t =
    i /= i' ==>
    modifica (i',v') (modifica (i,v) t)
    == modifica (i,v) (modifica (i',v') t)
```

# Especificación de las propiedades de las tablas

- Al modificar una tabla con dos pares con claves distintas no importa el orden en que se añadan los pares.

```
prop_modifica_modifica_2 :: Int -> Int -> Int -> Int
                         -> Tabla Int Int -> Property
prop_modifica_modifica_2 i i' v v' t =
    i /= i' ==>
    modifica (i',v') (modifica (i,v) t)
    == modifica (i,v) (modifica (i',v') t)
```

# Especificación de las propiedades de las tablas

- El valor de la clave `i` en la tabla obtenida añadiéndole el par `(i,v)` a la tabla `t` es `v`.

```
prop_valor_modifica_1 :: Int -> Int
                      -> Tabla Int Int -> Bool
prop_valor_modifica_1 i v t =
    valor (modifica (i,v) t) i == v
```

# Especificación de las propiedades de las tablas

- El valor de la clave `i` en la tabla obtenida añadiéndole el par `(i,v)` a la tabla `t` es `v`.

```
prop_valor_modifica_1 :: Int -> Int
                      -> Tabla Int Int -> Bool
prop_valor_modifica_1 i v t =
    valor (modifica (i,v) t) i == v
```

# Especificación de las propiedades de las tablas

- Sean `i` e `j` dos claves distintas. El valor de la clave `j` en la tabla obtenida añadiéndole el par `(i,v)` a la tabla `t'` (que contiene la clave `j`) es el valor de `j` en `t'`.

```
prop_valor_modifica_2 :: Int -> Int -> Int
                      -> Tabla Int Int -> Property
prop_valor_modifica_2 i v j v' t =
    i /= j ==>
    valor (modifica (i,v) t') j == valor t' j
    where t' = modifica (j,v') t
```

# Especificación de las propiedades de las tablas

- Sean `i` e `j` dos claves distintas. El valor de la clave `j` en la tabla obtenida añadiéndole el par `(i,v)` a la tabla `t'` (que contiene la clave `j`) es el valor de `j` en `t'`.

```
prop_valor_modifica_2 :: Int -> Int -> Int -> Int
                      -> Tabla Int Int -> Property
prop_valor_modifica_2 i v j v' t =
    i /= j ==>
    valor (modifica (i,v) t') j == valor t' j
    where t' = modifica (j,v') t
```

# Tema 18: El TAD de las tablas

# Definición del procedimiento de comprobación

- `compruebaPropiedades` comprueba todas las propiedades con la plataforma de verificación. Por ejemplo,

```
compruebaPropiedades =
    defaultMain
        [testGroup "Propiedades del TAD tabla"
          [testProperty "P1" prop_modifica_modifica_1,
           testProperty "P2" prop_modifica_modifica_2,
           testProperty "P3" prop_valor_modifica_1,
           testProperty "P4" prop_valor_modifica_2]]
```

# Comprobación de las propiedades de las tablas

```
Propiedades del TAD tabla:
  P1: [OK, passed 100 tests]
  P2: [OK, passed 100 tests]
  P3: [OK, passed 100 tests]
  P4: [OK, passed 100 tests]

         Properties  Total
 Passed  4           4
 Failed  0           0
 Total   4           4
```