Arrays, Matrices y Vectores

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Tipo predefinido Data.Array

Importación e índices

Data.Array se instala por defecto, y ofrece una forma genérica de representar matrices y vectores mediante tablas. Estas tablas asocian índices a elementos.

- La forma de importarlo es:

```
import Data.Array
```

- Las tablas usan como índices la clase Ix (de `Data.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)
```

Tipo predefinido Data.Array

Índices

Ejemplos de funciones de índices definidas sobre enteros (también puede ser sobre Bool, Char...).

- (`range (m,n)`) es la lista de índices desde m hasta n:

```
range (0,4) == [0,1,2,3,4]
range (3,9) == [3,4,5,6,7,8,9]
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):

```
index (3,9) 5 == 2
```

- (`inRange (m,n) i`) verifica si el índice i está en el rango (m,n):

```
inRange (0,4) 3 == True
```

- (`rangeSize (m,n)`) es el número de índices en el rango (m,n):

```
rangeSize (3,9) == 7
```

Tipo predefinido Data.Array

Algunas funciones

El tipo es (`Array i v`): tablas con índice en i y valores en v.

- (`array (m,n) ivs`) es la creación de tablas asociando los índices en el rango (m,n) a los elementos de ivs:

```
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)]
```

También se pueden definir tablas con dos dimensiones si los índices son pares:

```
ghci> let m = array ((1,1),(2,3)) [((i,j),i*j)) | i<-[1..2],
                                                  j<-[1..3]]
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)]
```

Tipo predefinido Data.Array

Algunas funciones

Una forma genérica de crear una tabla es usando una función que indique el valor dependiendo del índice:

```
m nf nc =
  array ((0,0),(nf',nc')) [((i,j), f i j)) | i<-[1..nf], j<-[1..nc]]
    where f i j = i*nc+j
          nf' = nf-1
          nc' = nc-1
ghci> m 3 2
array ((0,0),(2,1)) [((0,0),0),((0,1),1),((1,0),2),((1,1),3),((2,0),4),
((2,1),5)]
```

Tipo predefinido Data.Array

Algunas funciones

- (`t ! i`) es el valor del índice i en la tabla t:

```
ghci> m!(2,3) == 6
```

- (`indices t`) son los índices de la tabla t:

```
ghci> indices m  == [(1,1),(1,2),(1,3),(2,1),(2,2),(2,3)]
```

- (`elems t`) son los elementos de la tabla t:

```
ghci> elems m  == [1,2,3,2,4,6]
```

- (`assocs t`) son las asociaciones índice - elemento de la tabla t:

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

Librería Data.Matrix

Ofrece una interfaz más "conveniente" para matrices de dos dimensiones que Data.Array. El tipo es (`Matrix a`).

- A la hora de importarlo es conveniente hacerlo de forma cualificada, para no colisionar con *Prelude*:

```
import Data.Matrix as M
```

- (`matrix m n f`) es el constructor principal, creando una matriz de m filas, n columnas y usando la función f para calcular el valor en cada posición i,j.

```
ghci> M.matrix 3 4 (\(i,j) -> i-j)
(  0 -1 -2 -3 )
(  1  0 -1 -2 )
(  2  1  0 -1 )
```

Librería Data.Matrix

Algunas funciones

- Se puede construir desde una lista con (`fromList m n xs`):

```
ghci> M.fromList 2 3 [1..20]
(  1  2  3 )
(  4  5  6 )
```

- Se puede construir desde una listas de listas (cada sublista es una fila) con (`fromLists xss`):

```
ghci> M.fromLists [[1,2,3],[4,5,6]]
( 1 2 3 )
( 4 5 6 )
```

- Acceso a elementos con (`p!(i,j)`):

```
ghci> (M.fromList 2 3 [1..6]) M.! (2,1)
4
```

Librería Data.Vector

Ofrece una interfaz más "conveniente" para vectores (1 dimensión) que Data.Array. El tipo es (`Vector a`).

- A la hora de importarlo es conveniente hacerlo de forma cualificada, para no colisionar con *Prelude*:

```
import Data.Vector as V
```

- Tiene asociada muchas funciones para construcción, acceso, etc.
- Conectada con Matrix, de tal forma que podemos extraer toda una fila o una columna como Vector. Por ejemplo:

```
ghci> M.getRow 2 (M.fromList 2 3 [1..6])
fromList [4,5,6]
```

1. El tipo predefinido Data.Array
2. Las librerías Data.Matrix y Data.Vector
3. Bibliografía

Bibliografía


J.A. Alonso, *El TAD de las tablas*. http://www.cs.us.es/~jalonso/cursos/i1m-19/temas/tema-18.pdf. Tema 18 I1M


J.A. Alonso, *Manual de la librería de vectores Data.Vector.*. http://www.cs.us.es/~jalonso/cursos/i1m-19/doc/manual-Data.Vector.html.


J.A. Alonso, *Manual de la librería de vectores Data.Matrix.*. http://www.cs.us.es/~jalonso/cursos/i1m-19/doc/manual-Data.Matrix.html.