

Estructuras de Datos

Grado en Informática, Ingeniería del Software y Computadores

ETSI Informática

Universidad de Málaga

El Tipo Abstracto de Datos Bolsa

@ Pablo López

Dpto. Lenguajes y Ciencias de la Computación

Universidad de Málaga

Metodología de Trabajo

1. Especificar informalmente el TAD
2. Especificar formalmente el TAD
 1. Interfaz
 2. Especificación algebraica
3. Implementar el TAD
4. Analizar la eficiencia
5. Usar el TAD para resolver problemas

Especificación Informal

- Una bolsa es similar a un conjunto, pero los elementos pueden aparecer **repetidos**:

{ 'a', 'b', 'c', 'f', 'a', 'a', 't', 'c', 'a', 'c' }

- Como en cualquier colección, podremos:
 - **Insertar** un dato
 - **Eliminar** un dato
 - **Consultar** un dato (cuántas veces aparece)
 - **Comprobar** si la colección está **vacía**

Especificación formal: interfaz

El TAD **Bag** *a* tiene las siguientes operaciones:

-- constructores

empty :: Bag *a*

insert :: Ord *a* => *a* -> Bag *a* -> Bag *a*

El tipo base debe ser Ord o Eq

-- selectores

isEmpty :: Bag *a* -> Bool

occurrences :: Ord *a* => *a* -> Bag *a* -> Int

-- transformadores

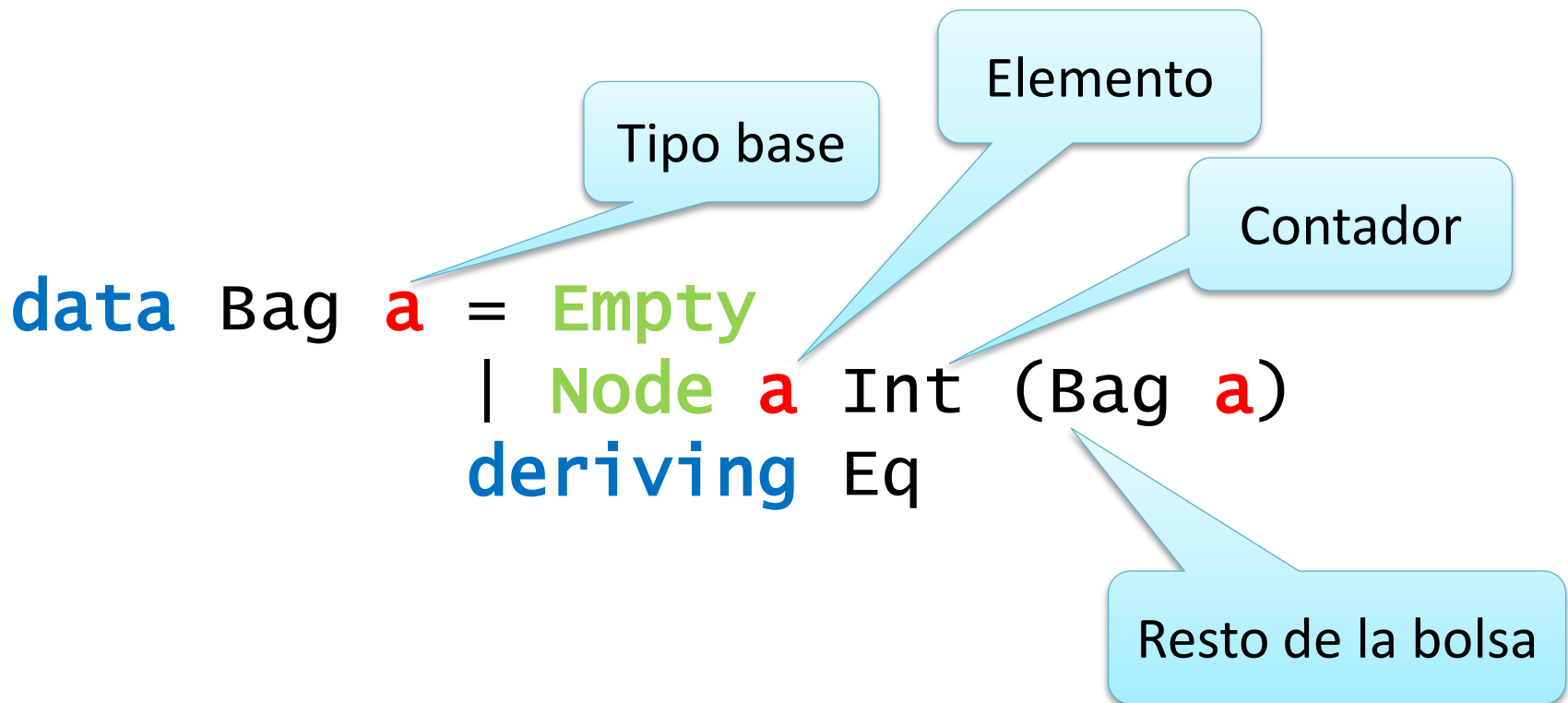
delete :: Ord *a* => *a* -> Bag *a* -> Bag *a*

Especificación formal: axiomas

- Basta especificar el resultado que deben devolver los **selectores** y **transformadores** cuando se aplican a una bolsa obtenida con los constructores **empty** e **insert**
- Las especificaciones de **occurrences** y **delete** deben distinguir dos casos para el constructor **insert**, según el dato esté o no presente en la bolsa

Implementación del TAD Bolsa (I)

- Utilizaremos el siguiente tipo algebraico recursivo parametrizado:



Implementación de la Bolsa (II)

■ La bolsa:

{ 'a', 'b', 'c', 'a', 'a', 't', 'c', 'a', 'c' }

se presenta en Haskell por:

Invariante: contadores positivos

Node 'a' 4 (Node 'b' 1 (Node 'c' 3 (Node 't' 1 Empty)))

Invariante: ordenado por elemento, sin repetidos

Implementación de la Bolsa: insert

insert 'a' Empty ->
Node 'a' 1 Empty

insert 'a' (Node 'a' 5 (Node 'c' 3 Empty)) ->
Node 'a' 6 (Node 'c' 3 Empty)

insert 'b' (Node 'a' 5 (Node 'c' 3 Empty)) ->
Node 'a' 5 (Node 'b' 1 (Node 'c' 3 Empty))

insert 'w' (Node 'a' 5 (Node 'c' 3 Empty)) ->
Node 'a' 5 (Node 'c' 3 (Node 'w' 1 Empty))

Operaciones auxiliares sobre bolsas

b1 = Node 'a' 2 (Node 'b' 3 Empty)

b2 = Node 'a' 5 (Node 'c' 1 Empty)

union b1 b2 =
Node 'a' 7 (Node 'b' 3 (Node 'c' 1 Empty))

intersection b1 b2 =
Node 'a' 2 Empty

difference b1 b2 =
Node 'b' 3 Empty

difference b2 b1
Node 'a' 3 (Node 'c' 1 Empty)

Eficiencia de la implementación

- Para cada operación, determinar el **número de pasos** que deben realizarse para llevarla a cabo:
 - **$O(1)$** – número de pasos constante, independiente del tamaño de la bolsa
 - **$O(n)$** – el número de pasos es proporcional al tamaño de la bolsa
- ¿Es ventajoso mantener los elementos ordenados? **¿Por qué?**

Uso del TAD Bolsa

Implementar en `BagClient.hs` la función:

```
cardinal :: Ord a => Bag a -> Int
```

que devuelve el número de elementos de una bolsa:

```
cardinal (mkBag "Haskell") = 7
```

Con las operaciones facilitadas en la interfaz es **imposible**; necesitamos un plegado.

Plegado de Bolsa: tipo

Para manejar bolsas como cliente es necesario utilizar el siguiente plegado:

foldBag :: Ord a =>
 (a -> Int -> b -> b) ->
 b ->
 Bag a ->
 b

Solución del
caso base

f x ox solRestoBolsa

Plegado de Bolsa: implementación

foldBag f solBase bolsa = plegar bolsa
where

plegar Empty = solBase

plegar (Node x ox s) = f x ox (plegar s)

un dato y su contador

solución del resto

El cardinal con un plegado

- Caso Base = 0
- Caso Recursivo:

$$\begin{array}{lcl} f & x & ox \\ f & 'a' & 5 \end{array} \quad \begin{array}{lcl} solResto & = & \\ 21 & & \end{array} = 21 + 5 = 26$$

La 'a' aparece 5 veces

El resto de la bolsa
tiene 21 elementos

cardinal xs = **foldBag** f 0 xs

where

f x ox solResto = ox + solResto

Uso del plegado

- Obtener una lista (sin repeticiones) con los elementos de una bolsa:

```
keys :: Ord a => Bag a -> [a]
```

```
keys xs = foldBag f [] xs
```

```
where
```

```
  f x ox solRestoBolsa = x : solRestoBolsa
```

- Ejemplo de uso:

```
keys (mkBag "abracadabra")  
"abcdr"
```