Resumen de los tipos abstractos de datos

José A. Alonso Jiménez
Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e I.A.
Universidad de Sevilla

11 de marzo de 2011

Resumen

Es este documento se resumen las especificaciones de los tipos abstractos de datos (TAD) estudiados en la asignatura de "Informática" de 1º del Grado en Matemáticas.

1. El TAD de las pilas

- Signatura:

```
vacia    :: Pila a
apila    :: a -> Pila a -> Pila a
cima     :: Pila a -> a
desapila :: Pila a -> Pila a
esVacia  :: Pila a -> Bool
```

- Descripción:
  - `vacia` es la pila vacía.
  - `(apila x p)` es la pila obtenida añadiendo `x` al principio de `p`.
  - `(cima p)` es la cima de la pila `p`.
  - `(desapila p)` es la pila obtenida suprimiendo la cima de `p`.
  - `(esVacia p)` se verifica si `p` es la pila vacía.
- Propiedades características:
  1. `cima (apila x p) = x`
  2. `desapila (apila x p) = p`
  3. `esVacia vacia`
  4. `not (esVacia (apila x p))`

2. El TAD de las colas

- Signatura:

```
vacia   :: Cola a
inserta :: a -> Cola a -> Cola a
primero :: Cola a -> a
resto   :: Cola a -> Cola a
esVacia :: Cola a -> Bool
valida  :: Cola a -> Bool
```

- Descripción de las operaciones:
  - `vacia` es la cola vacía.
  - `(inserta x c)` es la cola obtenida añadiendo `x` al final de `c`.
  - `(primero c)` es el primero de la cola `c`.
  - `(resto c)` es la cola obtenida eliminando el primero de `c`.
  - `(esVacia c)` se verifica si `c` es la cola vacía.
  - `(valida c)` se verifica si `c` representa una cola válida.
- Propiedades características:
  1. `primero (inserta x vacia) = x`
  2. Si `c` es una cola no vacía, entonces `primero (inserta x c) = primero c`,
  3. `resto (inserta x vacia) = vacia`
  4. Si `c` es una cola no vacía, entonces `resto (inserta x c) = inserta x (resto c)`
  5. `esVacia vacia`
  6. `not (esVacia (inserta x c))`

3. El TAD de las colas de prioridad

- Signatura:

```
vacia,   :: Ord a => CPrioridad a
inserta, :: Ord a => a -> CPrioridad a -> CPrioridad a
primero, :: Ord a => CPrioridad a -> a
resto,   :: Ord a => CPrioridad a -> CPrioridad a
esVacia, :: Ord a => CPrioridad a -> Bool
valida   :: Ord a => CPrioridad a -> Bool
```

- Descripción de las operaciones:
  - `vacia` es la cola de prioridad vacía.
  - `(inserta x c)` añade el elemento `x` a la cola de prioridad `c`.
  - `(primero c)` es el primer elemento de la cola de prioridad `c`.
  - `(resto c)` es el resto de la cola de prioridad `c`.
  - `(esVacia c)` se verifica si la cola de prioridad `c` es vacía.
  - `(valida c)` se verifica si `c` es una cola de prioridad válida.
- Propiedades características:
  1. `inserta x (inserta y c) = inserta y (inserta x c)`
  2. `primero (inserta x vacia) = x`
  3. Si $x \leq y$, entonces `primero (inserta y (inserta x c)) = primero (inserta x c)`
  4. `resto (inserta x vacia) = vacia`
  5. Si $x \leq y$, entonces `resto (inserta y (inserta x c)) = inserta y (resto (inserta x c))`
  6. `esVacia vacia`

7. `not (esVacia (inserta x c))`

4. El TAD de los conjuntos

- Signatura:

```
vacio,     :: Conj a
inserta    :: Eq a => a -> Conj a -> Conj a
elimina    :: Eq a => a -> Conj a -> Conj a
pertenece  :: Eq a => a -> Conj a -> Bool
esVacio    :: Conj a -> Bool
```

- Descripción de las operaciones:
  - `vacio` es el conjunto vacío.
  - `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`.
  - `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`.
  - `(pertenece x c)` se verifica si `x` pertenece al conjunto `c`.
  - `(esVacio c)` se verifica si `c` es el conjunto vacío.
- Propiedades características:
  1. `inserta x (inserta x c) = inserta x c`
  2. `inserta x (inserta y c) = inserta y (inserta x c)`
  3. `not (pertenece x vacio)`
  4. `pertenece y (inserta x c) = (x==y) || pertenece y c`
  5. `elimina x vacio = vacio`
  6. Si `x = y`, entonces `elimina x (inserta y c) = elimina x c`
  7. Si `x` $\neq$ `y`, entonces `elimina x (inserta y c) = inserta y (elimina x c)`
  8. `esVacio vacio`
  9. `not (esVacio (inserta x c))`

5. El 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`.
- Propiedades características:
  1. `modifica (i,v') (modifica (i,v) t) = modifica (i,v') t`
  2. Si `i` $\neq$ `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` $\neq$ `i'`, entonces
   `valor (modifica (i,v) (modifica (k',v') t)) i' = valor (modifica (k',v') t) i'`

6. El TAD de los árboles binarios de búsqueda

- Signatura:

```
vacio     :: ABB
inserta   :: (Ord a,Show a) => a -> ABB a -> ABB a
elimina   :: (Ord a,Show a) => a -> ABB a -> ABB a
crea      :: (Ord a,Show a) => [a] -> ABB a
menor     :: Ord a => ABB a -> a
elementos :: (Ord a,Show a) => ABB a -> [a]
pertenece :: (Ord a,Show a) => a -> ABB a -> Bool
valido    :: (Ord a,Show a) => ABB a -> Bool
```

- Descripción de las operaciones:
  - `vacio` es el ABB vacío.
  - `(pertenece v a)` se verifica si v es el valor de algún nodo del ABB a.
  - `(inserta v a)` es el árbol obtenido añadiendo el valor v al ABB a, si no es uno de sus valores.
  - `(crea vs)` es el ABB cuyos valores son vs.
  - `(elementos a)` es la lista de los valores de los nodos del ABB en el recorrido inorden.
  - `(elimina v a)` es el ABB obtenido eliminando el valor v del ABB a.
  - `(menor a)` es el mínimo valor del ABB a.
  - `(valido a)` se verifica si a es un ABB correcto.
- Propiedades características:
  1. `valido vacio`
  2. `valido (inserta v a)`
  3. `inserta x a /= vacio`
  4. `pertenece x (inserta x a)`
  5. `not (pertenece x vacio)`
  6. `pertenece y (inserta x a) = (x = y) || pertenece y a`
  7. `valido (elimina v a)`
  8. `elimina x (inserta x a) = elimina x a`
  9. `valido (crea xs)`
  10. `elementos (crea xs) = sort (nub xs)`
  11. `pertenece v a = elem v (elementos a)`
  12. $\forall$ `x` $\in$ `elementos a (menor a <= x)`

7. El TAD de los montículos

- Signatura:

```
vacio   :: Ord a => Monticulo a
inserta :: Ord a => a -> Monticulo a -> Monticulo a
menor   :: Ord a => Monticulo a -> a
```

```
resto   :: Ord a => Monticulo a -> Monticulo a
esVacio :: Ord a => Monticulo a -> Bool
valido  :: Ord a => Monticulo a -> Bool
```

- Descripción de las operaciones:
  - `vacio` es el montículo vacío.
  - `(inserta x m)` es el montículo obtenido añadiendo el elemento `x` al montículo `m`.
  - `(menor m)` es el menor elemento del montículo `m`.
  - `(resto m)` es el montículo obtenido eliminando el menor elemento del montículo `m`.
  - `(esVacio m)` se verifica si m es el montículo vacío.
  - `(valido m)` se verifica si `m` es un montículo; es decir, es un árbol binario en el que los valores de cada nodo es menor o igual que los valores de sus hijos.
- Propiedades características:
  1. `esVacio vacio`
  2. `valido (inserta x m)`
  3. `not (esVacio (inserta x m))`
  4. `not (esVacio m) ==> valido (resto m)`
  5. `resto (inserta x vacio) = vacio`
  6. `x <= menor m ==> resto (inserta x m) = m`
  7. Si `m` es no vacío y `x > menor m`, entonces `resto (inserta x m) = inserta x (resto m)`
  8. `esVacio m || esVacio (resto m) || menor m <= menor (resto m)`

8. El TAD de los polinomios

- Signatura:

```
polCero   :: Polinomio a
esPolCero :: Num a =>  Polinomio a -> Bool
consPol   :: Num a => Int -> a -> Polinomio a -> Polinomio a
grado     :: Polinomio a -> Int
coefLider :: Num a => Polinomio a -> a
restoPol  :: Polinomio a -> Polinomio a
```

- Descripción de las operaciones:
  - `polCero` es el polinomio cero.
  - `(esPolCero p)` se verifica si `p` es el polinomio cero.
  - `(consPol n b p)` es el polinomio $bx^n + p$.
  - `(grado p)` es el grado del polinomio `p`.
  - `(coefLider p)` es el coeficiente líder del polinomio `p`.
  - `(restoPol p)` es el resto del polinomio `p`.
- Propiedades características:
  1. `esPolCero polCero`
  2. `n > grado p && b /= 0 ==> not (esPolCero (consPol n b p))`
  3. `consPol (grado p) (coefLider p) (restoPol p) = p`
  4. `n > grado p && b /= 0 ==> grado (consPol n b p) = n`
  5. `n > grado p && b /= 0 ==> coefLider (consPol n b p) = b`
  6. `n > grado p && b /= 0 ==> restoPol (consPol n b p) = p`