Tema 20: El TAD de los montículos

Informática (2016–17)

José A. Alonso Jiménez

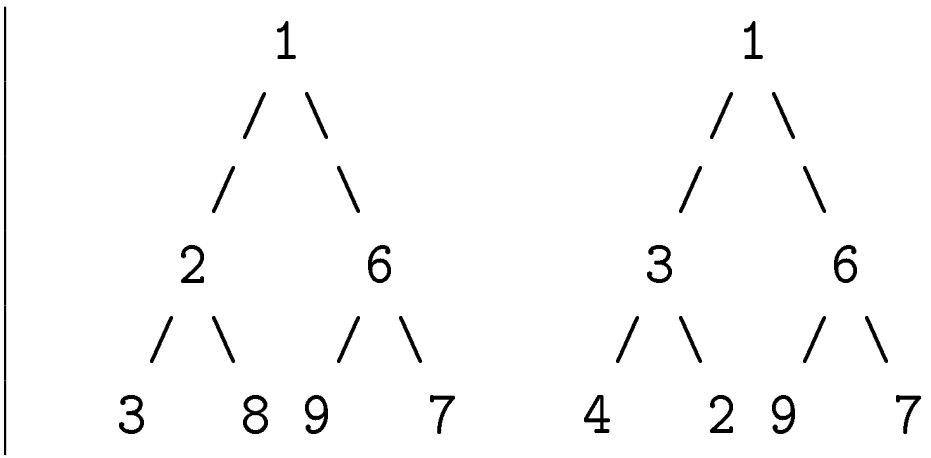
Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

Tema 20: El TAD de los montículos

Tema 20: El TAD de los montículos

Descripción de los montículos

Un montículo es un árbol binario en el que los valores de cada nodo es menor o igual que los valores de sus hijos. Por ejemplo,



el de la izquierda es un montículo, pero el de la derecha no lo es.

Signatura del 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
```

Signatura del TAD de los montículos

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.

Tema 20: El TAD de los montículos

Propiedades del TAD de los montículos

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)`

Tema 20: El TAD de los montículos

Los montículos como tipo de dato algebraico

- Cabecera del módulo:

```
module Monticulo
    (Monticulo,
     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
    ) where
```

- Librería auxiliar:

```
import Data.List (sort)
```

Los montículos como tipo de dato algebraico

- Los montículos como tipo de dato algebraico

```
data Ord a => Monticulo a
                  = Vacio
                  | M a Int (Monticulo a) (Monticulo a)
                  deriving Show
```

- La forma de los montículos no vacío es (M v r i d) donde
  - v es el valor de la raíz del montículo.
  - r es el rango del montículo; es decir, la menor distancia de la raíz a un montículo vacío.
  - i es el submontículo izquierdo y
  - f es el submontículo derecho.

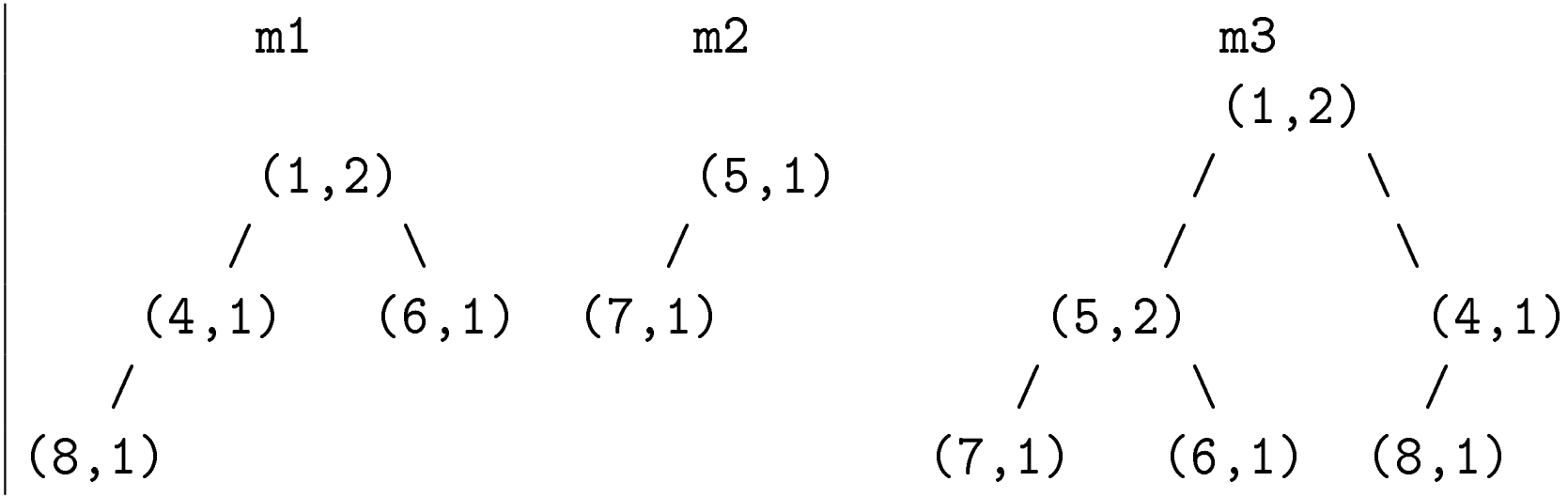
Los montículos como tipo de dato algebraico

Ejemplos de montículos

- Definición:

```
m1, m2, m3 :: Monticulo Int
m1 = foldr inserta vacio [6,1,4,8]
m2 = foldr inserta vacio [7,5]
m3 = mezcla m1 m2
```

- Representación:



Los montículos como tipo de dato algebraico

- `vacio` es el montículo vacío.

```
vacio :: Ord a => Monticulo a
vacio = Vacio
```

- `(rango m)` es el rango del montículo `m`; es decir, la menor distancia a un montículo vacío. Por ejemplo,

```
rango m1  ⇝  2
rango m2  ⇝  1
```

```
rango :: Ord a => Monticulo a -> Int
rango Vacio       = 0
rango (M _ r _ _) = r
```

Los montículos como tipo de dato algebraico

- `vacio` es el montículo vacío.

```
vacio :: Ord a => Monticulo a
vacio = Vacio
```

- `(rango m)` es el rango del montículo `m`; es decir, la menor distancia a un montículo vacío. Por ejemplo,

```
rango m1  ⇝  2
rango m2  ⇝  1
```

```
rango :: Ord a => Monticulo a -> Int
rango Vacio        = 0
rango (M _ r _ _) = r
```

Los montículos como tipo de dato algebraico

- `vacio` es el montículo vacío.

```
vacio :: Ord a => Monticulo a
vacio = Vacio
```

- `(rango m)` es el rango del montículo `m`; es decir, la menor distancia a un montículo vacío. Por ejemplo,

```
rango m1  ⇝  2
rango m2  ⇝  1
```

```
rango :: Ord a => Monticulo a -> Int
rango Vacio        = 0
rango (M _ r _ _) = r
```

Los montículos como tipo de dato algebraico

`(creaM x a b)` es el montículo creado a partir del elemento `x` y los montículos `a` y `b`. Se supone que `x` es menor o igual que el mínimo de `a` y de `b`. Por ejemplo,

```
ghci> m1
M 1 2 (M 4 1 (M 8 1 Vacio Vacio) Vacio) (M 6 1 Vacio Vacio)
ghci> m2
M 5 1 (M 7 1 Vacio Vacio) Vacio
ghci> creaM 0 m1 m2
M 0 2 (M 1 2 (M 4 1 (M 8 1 Vacio Vacio) Vacio) (M 6 1 Vacio V
      (M 5 1 (M 7 1 Vacio Vacio) Vacio)
```

```
creaM :: Ord a => a -> Monticulo a -> Monticulo a -> Monticulo
creaM x a b | rango a >= rango b = M x (rango b + 1) a b
            | otherwise          = M x (rango a + 1) b a
```

Los montículos como tipo de dato algebraico

`(creaM x a b)` es el montículo creado a partir del elemento `x` y los montículos `a` y `b`. Se supone que `x` es menor o igual que el mínimo de `a` y de `b`. Por ejemplo,

```
ghci> m1
M 1 2 (M 4 1 (M 8 1 Vacio Vacio) Vacio) (M 6 1 Vacio Vacio)
ghci> m2
M 5 1 (M 7 1 Vacio Vacio) Vacio
ghci> creaM 0 m1 m2
M 0 2 (M 1 2 (M 4 1 (M 8 1 Vacio Vacio) Vacio) (M 6 1 Vacio V
      (M 5 1 (M 7 1 Vacio Vacio) Vacio)
```

```
creaM :: Ord a => a -> Monticulo a -> Monticulo a -> Monticulo
creaM x a b | rango a >= rango b = M x (rango b + 1) a b
            | otherwise          = M x (rango a + 1) b a
```

Los montículos como tipo de dato algebraico

- `(mezcla m1 m2)` es el montículo obtenido mezclando los montículos `m1` y `m2`. Por ejemplo,

```
ghci> mezcla m1 m2
M 1 2 (M 5 2 (M 7 1 Vacio Vacio) (M 6 1 Vacio Vacio))
      (M 4 1 (M 8 1 Vacio Vacio) Vacio)
```

```
mezcla :: Ord a =>  Monticulo a -> Monticulo a
                    -> Monticulo a
mezcla m Vacio = m
mezcla Vacio m = m
mezcla m1@(M x _ a1 b1) m2@(M y _ a2 b2)
      | x <= y    = creaM x a1 (mezcla b1 m2)
      | otherwise = creaM y a2 (mezcla m1 b2)
```

Los montículos como tipo de dato algebraico

- `(mezcla m1 m2)` es el montículo obtenido mezclando los montículos `m1` y `m2`. Por ejemplo,

```
ghci> mezcla m1 m2
M 1 2 (M 5 2 (M 7 1 Vacio Vacio) (M 6 1 Vacio Vacio))
      (M 4 1 (M 8 1 Vacio Vacio) Vacio)
```

```
mezcla :: Ord a =>  Monticulo a -> Monticulo a
                    -> Monticulo a
mezcla m Vacio = m
mezcla Vacio m = m
mezcla m1@(M x _ a1 b1) m2@(M y _ a2 b2)
      | x <= y    = creaM x a1 (mezcla b1 m2)
      | otherwise = creaM y a2 (mezcla m1 b2)
```

Los montículos como tipo de dato algebraico

- `(inserta x m)` es el montículo obtenido añadiendo el elemento `x` al montículo `m`. Por ejemplo,

```
ghci> m1
M 1 2 (M 4 1 (M 8 1 Vacio Vacio) Vacio)
      (M 6 1 Vacio Vacio)
ghci> inserta 3 m1
M 1 2
  (M 4 1 (M 8 1 Vacio Vacio) Vacio)
  (M 3 1 (M 6 1 Vacio Vacio) Vacio)
```

```
inserta :: Ord a => a -> Monticulo a -> Monticulo a
inserta x m = mezcla (M x 1 Vacio Vacio) m
```

Los montículos como tipo de dato algebraico

- `(inserta x m)` es el montículo obtenido añadiendo el elemento `x` al montículo `m`. Por ejemplo,

```
ghci> m1
M 1 2 (M 4 1 (M 8 1 Vacio Vacio) Vacio)
      (M 6 1 Vacio Vacio)
ghci> inserta 3 m1
M 1 2
  (M 4 1 (M 8 1 Vacio Vacio) Vacio)
  (M 3 1 (M 6 1 Vacio Vacio) Vacio)
```

```
inserta :: Ord a => a -> Monticulo a -> Monticulo a
inserta x m = mezcla (M x 1 Vacio Vacio) m
```

Los montículos como tipo de dato algebraico

- `(menor m)` es el menor elemento del montículo `m`. Por ejemplo,

```
menor m1  ⇝  1
menor m2  ⇝  5
```

```
menor :: Ord a => Monticulo a -> a
menor (M x _ _ _) = x
menor Vacio       = error "menor: monticulo vacio"
```

- `(resto m)` es el montículo obtenido eliminando el menor elemento del montículo `m`. Por ejemplo,

```
ghci> resto m1
M 4 2 (M 8 1 Vacio Vacio) (M 6 1 Vacio Vacio)
```

```
resto :: Ord a => Monticulo a -> Monticulo a
resto Vacio       = error "resto: monticulo vacio"
resto (M x _ a b) = mezcla a b
```

Los montículos como tipo de dato algebraico

- `(menor m)` es el menor elemento del montículo `m`. Por ejemplo,

```
menor m1  ⇝  1
menor m2  ⇝  5
```

```
menor :: Ord a => Monticulo a -> a
menor (M x _ _ _) = x
menor Vacio       = error "menor: monticulo vacio"
```

- `(resto m)` es el montículo obtenido eliminando el menor elemento del montículo `m`. Por ejemplo,

```
ghci> resto m1
M 4 2 (M 8 1 Vacio Vacio) (M 6 1 Vacio Vacio)
```

```
resto :: Ord a => Monticulo a -> Monticulo a
resto Vacio       = error "resto: monticulo vacio"
resto (M x _ a b) = mezcla a b
```

Los montículos como tipo de dato algebraico

- `(menor m)` es el menor elemento del montículo `m`. Por ejemplo,

```
menor m1  ⇝  1
menor m2  ⇝  5
```

```
menor :: Ord a => Monticulo a -> a
menor (M x _ _ _) = x
menor Vacio       = error "menor: monticulo vacio"
```

- `(resto m)` es el montículo obtenido eliminando el menor elemento del montículo `m`. Por ejemplo,

```
ghci> resto m1
M 4 2 (M 8 1 Vacio Vacio) (M 6 1 Vacio Vacio)
```

```
resto :: Ord a => Monticulo a -> Monticulo a
resto Vacio       = error "resto: monticulo vacio"
resto (M x _ a b) = mezcla a b
```

Los montículos como tipo de dato algebraico

- `(esVacio m)` se verifica si m es el montículo vacío.

```
esVacio :: Ord a => Monticulo a -> Bool
esVacio Vacio = True
esVacio _     = False
```

Los montículos como tipo de dato algebraico

- `(esVacio m)` se verifica si m es el montículo vacío.

```
esVacio :: Ord a => Monticulo a -> Bool
esVacio Vacio = True
esVacio _     = False
```

Los montículos como tipo de dato algebraico

- `(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. Por ejemplo,

```
valido m1  ⇝  True
valido (M 3 5 (M 2 1 Vacio Vacio) Vacio)  ⇝  False
```

```
valido :: Ord a => Monticulo a -> Bool
valido Vacio = True
valido (M x _ Vacio Vacio) = True
valido (M x _ m1@(M x1 n1 a1 b1) Vacio) =
    x <= x1 && valido m1
valido (M x _ Vacio m2@(M x2 n2 a2 b2)) =
    x <= x2 && valido m2
valido (M x _ m1@(M x1 n1 a1 b1) m2@(M x2 n2 a2 b2)) =
    x <= x1 && valido m1 &&
    x <= x2 && valido m2
```

Los montículos como tipo de dato algebraico

- `(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. Por ejemplo,

```
valido m1  ⇝  True
valido (M 3 5 (M 2 1 Vacio Vacio) Vacio)  ⇝  False
```

```
valido :: Ord a => Monticulo a -> Bool
valido Vacio = True
valido (M x _ Vacio Vacio) = True
valido (M x _ m1@(M x1 n1 a1 b1) Vacio) =
    x <= x1 && valido m1
valido (M x _ Vacio m2@(M x2 n2 a2 b2)) =
    x <= x2 && valido m2
valido (M x _ m1@(M x1 n1 a1 b1) m2@(M x2 n2 a2 b2)) =
    x <= x1 && valido m1 &&
    x <= x2 && valido m2
```

Los montículos como tipo de dato algebraico

- `(elementos m)` es la lista de los elementos del montículo `m`. Por ejemplo,

  `elementos m1` ⇝ `[1,4,8,6]`

```
elementos :: Ord a => Monticulo a -> [a]
elementos Vacio        = []
elementos (M x _ a b) = x : elementos a ++ elementos b
```

Los montículos como tipo de dato algebraico

- `(elementos m)` es la lista de los elementos del montículo `m`. Por ejemplo,

  `elementos m1` $\rightsquigarrow$ `[1,4,8,6]`

```
elementos :: Ord a => Monticulo a -> [a]
elementos Vacio        = []
elementos (M x _ a b) = x : elementos a ++ elementos b
```

Los montículos como tipo de dato algebraico

- `(equivMonticulos m1 m2)` se verifica si los montículos `m1` y `m2` tienen los mismos elementos. Por ejemplo,

```
ghci> m1
M 1 2 (M 4 1 (M 8 1 Vacio Vacio) Vacio)
      (M 6 1 Vacio Vacio)
ghci> let m1' = foldr inserta vacio [6,8,4,1]
M 1 2 (M 4 1 Vacio Vacio)
      (M 6 1 (M 8 1 Vacio Vacio) Vacio)
ghci> equivMonticulos m1 m1'
True
```

```
equivMonticulos :: Ord a => Monticulo a -> Monticulo a
                   -> Bool
equivMonticulos m1 m2 =
    sort (elementos m1) == sort (elementos m2)
```

Los montículos como tipo de dato algebraico

- `(equivMonticulos m1 m2)` se verifica si los montículos `m1` y `m2` tienen los mismos elementos. Por ejemplo,

```
ghci> m1
M 1 2 (M 4 1 (M 8 1 Vacio Vacio) Vacio)
      (M 6 1 Vacio Vacio)
ghci> let m1' = foldr inserta vacio [6,8,4,1]
M 1 2 (M 4 1 Vacio Vacio)
      (M 6 1 (M 8 1 Vacio Vacio) Vacio)
ghci> equivMonticulos m1 m1'
True
```

```
equivMonticulos :: Ord a => Monticulo a -> Monticulo a
                   -> Bool
equivMonticulos m1 m2 =
    sort (elementos m1) == sort (elementos m2)
```

Los montículos como tipo de dato algebraico

- Los montículos son comparables por igualdad.

```
instance Ord a => Eq (Monticulo a) where
   (==) = equivMonticulos
```

Tema 20: El TAD de los montículos

Comprobación de las propiedades del TAD de los montículos

- Importación de la implementación a verificar.

```
import Monticulo
```

- Importación de librerías auxiliares.

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

Tema 20: El TAD de los montículos

Generador de montículos

- `(creaMonticulo xs)` es el montículo correspondiente a la lista `xs`. Por ejemplo,

```
ghci> creaMonticulo [6,1,4,8]
M 1 2 (M 4 1 (M 8 1 Vacio Vacio) Vacio)
      (M 6 1 Vacio Vacio)
ghci> creaMonticulo [6,8,4,1]
M 1 2 (M 4 1 Vacio Vacio)
      (M 6 1 (M 8 1 Vacio Vacio) Vacio)
```

```
creaMonticulo :: [Int] -> Monticulo Int
creaMonticulo = foldr inserta vacio
```

Generador de montículos

- `(creaMonticulo xs)` es el montículo correspondiente a la lista `xs`. Por ejemplo,

```
ghci> creaMonticulo [6,1,4,8]
M 1 2 (M 4 1 (M 8 1 Vacio Vacio) Vacio)
      (M 6 1 Vacio Vacio)
ghci> creaMonticulo [6,8,4,1]
M 1 2 (M 4 1 Vacio Vacio)
      (M 6 1 (M 8 1 Vacio Vacio) Vacio)
```

```
creaMonticulo :: [Int] -> Monticulo Int
creaMonticulo = foldr inserta vacio
```

Generador de montículos

- `genMonticulo` es un generador de montículos. Por ejemplo,

```
ghci> sample genMonticulo
VacioM
M (-1) 1 (M 1 1 VacioM VacioM) VacioM
...
```

```
genMonticulo :: Gen (Monticulo Int)
genMonticulo = do xs <- listOf arbitrary
                  return (creaMonticulo xs)

instance Arbitrary (Monticulo Int) where
    arbitrary = genMonticulo
```

Corrección del generador de montículos

- Prop.: `genMonticulo` genera montículos válidos.

```
prop_genMonticulo :: Monticulo Int -> Bool
prop_genMonticulo m = valido m
```

Comprobación:

```
ghci> quickCheck prop_genMonticulo
+++ OK, passed 100 tests.
```

Corrección del generador de montículos

- Prop.: `genMonticulo` genera montículos válidos.

```
prop_genMonticulo :: Monticulo Int -> Bool
prop_genMonticulo m = valido m
```

Comprobación:

```
ghci> quickCheck prop_genMonticulo
+++ OK, passed 100 tests.
```

Generador de montículos no vacíos

- `monticuloNV` es un generador de montículos no vacío. Por ejemplo,

```
ghci> sample monticuloNV
M 0 1 VacioM VacioM
M 1 1 (M 1 1 (M 1 1 VacioM VacioM) VacioM) VacioM
...
```

```
monticuloNV :: Gen (Monticulo Int)
monticuloNV = do xs <- listOf arbitrary
                 x <- arbitrary
                 return (creaMonticulo (x:xs))
```

Corrección del generador de montículos no vacíos

- Prop.: `monticuloNV` genera montículos no vacío.

```
prop_monticuloNV :: Monticulo Int -> Property
prop_monticuloNV m =
    forAll monticuloNV
           (\m -> (valido m) && not (esVacio m))
```

Comprobación:

```
ghci> quickCheck prop_monticuloNV
+++ OK, passed 100 tests.
```

Corrección del generador de montículos no vacíos

- Prop.: `monticuloNV` genera montículos no vacío.

```
prop_monticuloNV :: Monticulo Int -> Property
prop_monticuloNV m =
    forAll monticuloNV
           (\m -> (valido m) && not (esVacio m))
```

Comprobación:

```
ghci> quickCheck prop_monticuloNV
+++ OK, passed 100 tests.
```

Tema 20: El TAD de los montículos

Especificación de las propiedades de los montículos

- `vacio` es un montículo.

```
prop_vacio_es_monticulo :: Bool
prop_vacio_es_monticulo =
    esVacio (vacio :: Monticulo Int)
```

- `inserta` produce montículos válidos.

```
prop_inserta_es_valida :: Int -> Monticulo Int -> Bool
prop_inserta_es_valida x m =
    valido (inserta x m)
```

Especificación de las propiedades de los montículos

- `vacio` es un montículo.

```
prop_vacio_es_monticulo :: Bool
prop_vacio_es_monticulo =
    esVacio (vacio :: Monticulo Int)
```

- `inserta` produce montículos válidos.

```
prop_inserta_es_valida :: Int -> Monticulo Int -> Bool
prop_inserta_es_valida x m =
    valido (inserta x m)
```

Especificación de las propiedades de los montículos

- `vacio` es un montículo.

```
prop_vacio_es_monticulo :: Bool
prop_vacio_es_monticulo =
    esVacio (vacio :: Monticulo Int)
```

- `inserta` produce montículos válidos.

```
prop_inserta_es_valida :: Int -> Monticulo Int -> Bool
prop_inserta_es_valida x m =
    valido (inserta x m)
```

Especificación de las propiedades de los montículos

- Los montículos creados con `inserta` son no vacío.

```
prop_inserta_no_vacio :: Int -> Monticulo Int -> Bool
prop_inserta_no_vacio x m =
    not (esVacio (inserta x m))
```

- Al borrar el menor elemento de un montículo no vacío se obtiene un montículo válido.

```
prop_resto_es_valida :: Monticulo Int -> Property
prop_resto_es_valida m =
    forAll monticuloNV  (\m -> valido (resto m))
```

Especificación de las propiedades de los montículos

- Los montículos creados con `inserta` son no vacío.

```
prop_inserta_no_vacio :: Int -> Monticulo Int -> Bool
prop_inserta_no_vacio x m =
    not (esVacio (inserta x m))
```

- Al borrar el menor elemento de un montículo no vacío se obtiene un montículo válido.

```
prop_resto_es_valida :: Monticulo Int -> Property
prop_resto_es_valida m =
    forAll monticuloNV  (\m -> valido (resto m))
```

Especificación de las propiedades de los montículos

- Los montículos creados con `inserta` son no vacío.

```
prop_inserta_no_vacio :: Int -> Monticulo Int -> Bool
prop_inserta_no_vacio x m =
    not (esVacio (inserta x m))
```

- Al borrar el menor elemento de un montículo no vacío se obtiene un montículo válido.

```
prop_resto_es_valida :: Monticulo Int -> Property
prop_resto_es_valida m =
    forAll monticuloNV  (\m -> valido (resto m))
```

Especificación de las propiedades de los montículos

- El resto de `(inserta x m)` es `m` si `m` es el montículo vacío o `x` es menor o igual que el menor elemento de `m` y es `(inserta x (resto m))`, en caso contrario.

```
prop_resto_inserta :: Int -> Monticulo Int -> Bool
prop_resto_inserta x m =
    resto (inserta x m)
    == if esVacio m || x <= menor m then m
       else inserta x (resto m)
```

- `(menor m)` es el menor elemento del montículo `m`.

```
prop_menor_es_minimo :: Monticulo Int -> Bool
prop_menor_es_minimo m =
    esVacio m || esVacio (resto m) ||
    menor m <= menor (resto m)
```

Especificación de las propiedades de los montículos

- El resto de `(inserta x m)` es `m` si `m` es el montículo vacío o `x` es menor o igual que el menor elemento de `m` y es `(inserta x (resto m))`, en caso contrario.

```
prop_resto_inserta :: Int -> Monticulo Int -> Bool
prop_resto_inserta x m =
    resto (inserta x m)
    == if esVacio m || x <= menor m then m
       else inserta x (resto m)
```

- `(menor m)` es el menor elemento del montículo `m`.

```
prop_menor_es_minimo :: Monticulo Int -> Bool
prop_menor_es_minimo m =
    esVacio m || esVacio (resto m) ||
    menor m <= menor (resto m)
```

Especificación de las propiedades de los montículos

- El resto de `(inserta x m)` es `m` si `m` es el montículo vacío o `x` es menor o igual que el menor elemento de `m` y es `(inserta x (resto m))`, en caso contrario.

```
prop_resto_inserta :: Int -> Monticulo Int -> Bool
prop_resto_inserta x m =
    resto (inserta x m)
    == if esVacio m || x <= menor m then m
       else inserta x (resto m)
```

- `(menor m)` es el menor elemento del montículo `m`.

```
prop_menor_es_minimo :: Monticulo Int -> Bool
prop_menor_es_minimo m =
    esVacio m || esVacio (resto m) ||
    menor m <= menor (resto m)
```

Tema 20: El TAD de los montículos

Definición del procedimiento de comprobación

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

```
compruebaPropiedades =
    defaultMain
        [testGroup "Propiedades del TAD monticulo"
         [testProperty "P1" prop_genMonticulo,
          testProperty "P2" prop_monticuloNV,
          testProperty "P3" prop_vacio_es_monticulo,
          testProperty "P4" prop_inserta_es_valida,
          testProperty "P5" prop_inserta_no_vacio,
          testProperty "P6" prop_resto_es_valida,
          testProperty "P7" prop_resto_inserta,
          testProperty "P8" prop_menor_es_minimo]]
```

Comprobación de las propiedades de los montículos

```
ghci> compruebaPropiedades
Propiedades del TAD monticulo:
  P1: [OK, passed 100 tests]
  P2: [OK, passed 100 tests]
  P3: [OK, passed 100 tests]
  P4: [OK, passed 100 tests]
  P5: [OK, passed 100 tests]
  P6: [OK, passed 100 tests]
  P7: [OK, passed 100 tests]
  P8: [OK, passed 100 tests]

         Properties  Total
 Passed  8           8
 Failed  0           0
 Total   8           8
```

Tema 20: El TAD de los montículos

1. Especificación del TAD de los montículos
2. Implementación del TAD de los montículos
3. Comprobación de la implementación con QuickCheck
4. Implementación de las colas de prioridad mediante montículos
   - Las colas de prioridad como montículos

Las colas de prioridad como montículos

Cabecera del módulo:

```
module ColaDePrioridadConMonticulos
    (CPrioridad,
     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
    ) where
```

Importación cualificada:

```
import qualified Monticulo as M
```

Las colas de prioridad como montículos

- 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.
- Las colas de prioridad como montículos.

```
newtype CPrioridad a = CP (M.Monticulo a)
    deriving (Eq, Show)
```

Las colas de prioridad como montículos

- Ejemplo de cola de prioridad:

```
cp1 :: CPrioridad Int
cp1 = foldr inserta vacia [3,1,7,2,9]
```

- Evaluación:

```
ghci> cp1
CP (M 1 2
      (M 2 2
         (M 9 1 VacioM VacioM)
         (M 7 1 VacioM VacioM))
      (M 3 1 VacioM VacioM))
```

Las colas de prioridad como montículos

- `vacia` es la cola de prioridad vacía. Por ejemplo,

```
vacia  ⇝ CP Vacio
```

```
vacia :: Ord a => CPrioridad a
vacia = CP M.vacio
```

Las colas de prioridad como montículos

- `vacia` es la cola de prioridad vacía. Por ejemplo,

  `vacia` $\rightsquigarrow$ `CP Vacio`

```
vacia :: Ord a => CPrioridad a
vacia = CP M.vacio
```

Las colas de prioridad como montículos

- `(inserta x c)` añade el elemento `x` a la cola de prioridad `c`.

Por ejemplo,

```
ghci> inserta 5 cp1
CP (M 1 2
       (M 2 2
          (M 9 1 VacioM VacioM)
          (M 7 1 VacioM VacioM))
       (M 3 1
          (M 5 1 VacioM VacioM) VacioM))
```

```
inserta :: Ord a => a -> CPrioridad a -> CPrioridad a
inserta v (CP c) = CP (M.inserta v c)
```

Las colas de prioridad como montículos

- `(inserta x c)` añade el elemento `x` a la cola de prioridad `c`.

Por ejemplo,

```
ghci> inserta 5 cp1
CP (M 1 2
      (M 2 2
         (M 9 1 VacioM VacioM)
         (M 7 1 VacioM VacioM))
      (M 3 1
         (M 5 1 VacioM VacioM) VacioM))
```

```
inserta :: Ord a => a -> CPrioridad a -> CPrioridad a
inserta v (CP c) = CP (M.inserta v c)
```

Las colas de prioridad como montículos

- `(primero c)` es la cabeza de la cola de prioridad c. Por ejemplo,

```
primero cp1  ⇝  1
```

```
primero :: Ord a => CPrioridad a -> a
primero (CP c) = M.menor c
```

- `(resto c)` elimina la cabeza de la cola de prioridad c. Por ejemplo,

```
ghci> resto cp1
CP (M 2 2
      (M 9 1 VacioM VacioM)
      (M 3 1
         (M 7 1 VacioM VacioM) VacioM))
```

```
resto :: Ord a => CPrioridad a -> CPrioridad a
resto (CP c) = CP (M resto c)
```

Las colas de prioridad como montículos

- `(primero c)` es la cabeza de la cola de prioridad c. Por ejemplo,

```
primero cp1  ⇝  1
```

```
primero :: Ord a => CPrioridad a -> a
primero (CP c) = M.menor c
```

- `(resto c)` elimina la cabeza de la cola de prioridad c. Por ejemplo,

```
ghci> resto cp1
CP (M 2 2
      (M 9 1 VacioM VacioM)
      (M 3 1
         (M 7 1 VacioM VacioM) VacioM))
```

```
resto :: Ord a => CPrioridad a -> CPrioridad a
resto (CP c) = CP (M.resto c)
```

Las colas de prioridad como montículos

- `(primero c)` es la cabeza de la cola de prioridad c. Por ejemplo,

```
primero cp1  ⇝  1
```

```
primero :: Ord a => CPrioridad a -> a
primero (CP c) = M.menor c
```

- `(resto c)` elimina la cabeza de la cola de prioridad c. Por ejemplo,

```
ghci> resto cp1
CP (M 2 2
      (M 9 1 VacioM VacioM)
      (M 3 1
         (M 7 1 VacioM VacioM) VacioM))
```

```
resto :: Ord a => CPrioridad a -> CPrioridad a
resto (CP c) = CP (M.resto c)
```

Las colas de prioridad como montículos

- `(esVacia c)` se verifica si la cola de prioridad c es vacía. Por ejemplo,

```
esVacia cp1    ⇝  False
esVacia vacia  ⇝  True
```

```
esVacia :: Ord a => CPrioridad a -> Bool
esVacia (CP c) = M.esVacio c
```

- `(valida c)` se verifica si c es una cola de prioridad válida. En la representación mediante montículo todas las colas de prioridad son válidas.

```
valida :: Ord a => CPrioridad a -> Bool
valida _ = True
```

Las colas de prioridad como montículos

- `(esVacia c)` se verifica si la cola de prioridad c es vacía. Por ejemplo,

```
esVacia cp1    ⇝  False
esVacia vacia  ⇝  True
```

```
esVacia :: Ord a => CPrioridad a -> Bool
esVacia (CP c) = M.esVacio c
```

- `(valida c)` se verifica si c es una cola de prioridad válida. En la representación mediante montículo todas las colas de prioridad son válidas.

```
valida :: Ord a => CPrioridad a -> Bool
valida _ = True
```

Las colas de prioridad como montículos

- `(esVacia c)` se verifica si la cola de prioridad c es vacía. Por ejemplo,

```
esVacia cp1    ⇝  False
esVacia vacia  ⇝  True
```

```
esVacia :: Ord a => CPrioridad a -> Bool
esVacia (CP c) = M.esVacio c
```

- `(valida c)` se verifica si c es una cola de prioridad válida. En la representación mediante montículo todas las colas de prioridad son válidas.

```
valida :: Ord a => CPrioridad a -> Bool
valida _ = True
```

Bibliografía

1. Functional heap - Leftist tree