

# Listas y Quadtrees

Curso 2004/05, Fecha:20/01/2005

**Enunciado 1 (inserta)** Definir una función *inserta*  $x$   $xs$  que devuelva la lista resultante de insertar  $x$  en cualquier posición de  $xs$

?- *inserta* 1 [2,3]

[1,2,3] | [2,1,3] | [2,3,1]

**Enunciado 2 (permutaciones)** Definir una función *perm*  $xs$  que devuelva una lista que es una permutación de  $xs$ .

?- *perm* [1,2,3]

[1, 2, 3] | [2, 1, 3] | [2, 3, 1] | [1, 3, 2] | [3, 1, 2] | [3, 2, 1]

**Enunciado 3 (ordenada)** Definir una función *ordenada* que devuelva *True* si los elementos de una lista están en orden creciente

?- *ordenada* [1,3,2]

**False**

?- *ordenada* [1,2,5]

**True**

**Enunciado 4 (ordena)** Definir una función *ordena* que tome una lista de elementos y devuelva una lista con dichos elementos ordenados

?- *ordena* [1,3,2]

[1,2,3]

**Enunciado 5 (Quicksort)** Implementar el algoritmo *quicksort* en Curry

**Enunciado 6 (mergesort(Opcional))** Implementar el algoritmo *mergesort* de ordenación por mezcla en Curry

**Enunciado 7 (nReinas)** El problema de las  $n$  reinas consiste en disponer  $n$  reinas en un tablero de ajedrez de forma que no se ataquen entre sí.

Si se representan las reinas mediante una lista que indica la altura de cada reina, la siguiente función chequea que las reinas no se atacan entre sí:

```

> noSeAtacan [] = success
> noSeAtacan (y:ys) = noAtaca (1,y) ys & noSeAtacan ys

> noAtaca (x,y) [] = success
> noAtaca (x,y) (y':ys) = y /= y'
>                               & abs (y' - y) /= x
>                               & noAtaca (x+1,y) ys

> abs x | x < 0 = -x
>           | otherwise = x

```

Definir una función  $nReinas :: Int \rightarrow [Int]$  que genere una lista con las posiciones que deben tomar  $n$  reinas en un tablero de ajedrez sin que se ataquen entre sí.

```

?- nReinas 8
[5,2,6,1,7,4,8,3]

```

**Enunciado 8 (QuadTrees)** Una posible representación de quadrees en Curry podría ser:

```

> data Color = RGB Int Int Int
>   deriving (Eq, Show)

> rojo = RGB 255 0 0
> verde = RGB 0 255 0
> azul = RGB 0 0 255

> data QT = B Color
>         | D QT QT QT QT
>   deriving Show

> q :: QT
> q = D (B rojo) (B verde) (B verde) (B rojo)

```

La siguiente función puede utilizarse para guardar una representación de un quadtree como un fichero SVG

```

> type Punto = (Int, Int)
> type Dim = (Punto, Int)

> verqt :: QT -> IO ()
> verqt q = writeFile "qtree.svg"
>           (gen "svg" (ver ((0,0),512) q))

> ver :: Dim -> QT -> String
> ver ((x,y),d) (B c) =
>   rect (x,y) (x+d+1,y+d+1) c

```

```

> ver ((x,y),d) (D ul ur dl dr) =
>   let d2 = d `div` 2
>   in
>     if d <= 0 then ""
>     else ver ((x,y),d2)      ul ++
>           ver ((x+d2,y),d2)  ur ++
>           ver ((x,y+d2),d2)  dl ++
>           ver ((x+d2,y+d2),d2) dr

> rect :: Punto → Punto → Color → String
> rect (x,y) (x',y') (RGB r g b) =
>   vacio "rect" [("x",show x),("y",show y),
>                ("height",show (abs (x - x'))),
>                ("width",show (abs (y - y'))),
>                ("fill", "rgb(" ++ show r ++ "," ++
>                               show g ++ "," ++
>                               show b ++ ")"),
>                ("stroke-width", "0")]

> vacio :: String → [(String, String)] → String
> vacio e as = "<" ++ e ++ ponAtts as ++ ">"

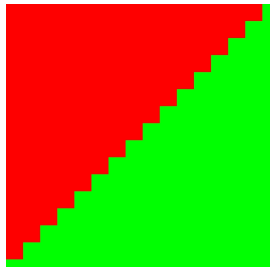
> ponAtts :: [(String, String)] → String
> ponAtts = concat . map f
>   where f (v,n) = "_" ++ v ++ "=" ++ n ++ "\" \" \"

> genAs e as c = "<" ++ e ++ ponAtts as ++ ">" ++ c ++ "</" ++ e ++
">"
> gen e c = genAs e [] c

```

Nota: El código anterior ha sido tomado de las prácticas anteriores realizadas en Haskell

- Definir algunas de las funciones que trabajaban en Haskell con quadrees en Curry
- Definir funciones que generen quadrees infinitos en Curry
- Definir una función que genere un quadtree en forma de escalera



**Enunciado 9 (Colorea)** La siguiente función se cumple si en un quadtree no existen cuadrantes adyacentes del mismo color

```

> noColor :: QT → Success
> noColor (B _) = success
> noColor (D a b c d) =
>   noColor a & noColor b & noColor c & noColor d &
>   diff ar bl & diff cr dl & diff ad cu & diff bd du
>   where ar = right a
>         bl = left b
>         cr = right c
>         dl = left d
>         ad = down a
>         cu = up c
>         bd = down b
>         du = up d

> — Auxiliary data structure to check if the borders
> — of a quadtree have the same color
> data Tree a = L a | F (Tree a) (Tree a)
>
> up (B x)           = L x
> up (D a b _ _) = F x y
>   where { x = up a; y = up b }
>
> down (B x)         = L x
> down (D _ _ c d) = F x y
>   where { x = down c; y = down d }
>
> left (B x)         = L x
> left (D a _ c _) = F x y
>   where { x = left a; y = left c }
>
> right (B x)        = L x
> right (D _ b _ d) = F x y
>   where { x = right b; y = right d }
>
>
> diff (L x) (L y)   = x /= y
> diff (L x) (F a b) = notInTree x a & notInTree x b
> diff (F a b) (L x) = notInTree x a & notInTree x b
> diff (F a b) (F c d) = diff a c & diff b d

> notInTree x (L y)   = x /= y
> notInTree x (F a b) = notInTree x a & notInTree x b

```

- Construir una función  $col :: QT \rightarrow [Color] \rightarrow QT$  que toma un quadtree  $q$  y una lista de colores y devuelve el quadtree resultante de colorear  $q$  con los colores de la lista

- Construir una función  $\text{colorea}::QT \rightarrow [\text{Color}] \rightarrow QT$  que tome un quadtree  $q$  y una lista de colores  $cs$ , y devuelva un quadtree coloreado con los colores  $cs$  y con la misma forma que  $q$ , pero que no contenga 2 cuadrados adyacentes del mismo color
- Una vez construidas las funciones anteriores, es posible almacenar el resultado en un fichero SVG mediante la siguiente función:

```

> verColorea qt = verqt
>               ( head
>                 ( findall
>                   (\x → x :=: colorea qt)))

```

---

**Enunciado 10 (Algoritmo(Opcional))** Implementar un algoritmo para colorear quadtrees más eficiente que el anterior. ¿Cuál es el número mínimo de colores que se necesitan para garantizar que no haya 2 casillas adyacentes del mismo color?. En *Algorithms for coloring quadtrees* se pueden encontrar diversos algoritmos para colorear quadtrees. ¿Serías capaz de implementarlos en Curry o en Haskell?