

Tipos de datos recursivos: Quadrees y Octrees

Curso 2004/05, Fecha:11/11/2004

Enunciado 1 (Quadtree) *El siguiente programa define un tipo de datos recursivo que representa quadrees.*

```
> module Quadrees where
> import LibXML(vacio , gen , genAs)

> data Color = RGB Int Int Int
>   deriving Show

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

> rojo , verde , ej :: QT
> rojo = B (RGB 255 0 0)
> verde = B (RGB 0 255 0)
> ej = D rojo verde verde rojo
```

Compilar el programa y definir algunos ejemplos de quadrees. Chequear el tipo de los ejemplos definidos.

Enunciado 2 (ver) *Añadir al programa anterior el siguiente fragmento que incluye la función verqt que permite visualizar una representación del quadtree en formato 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
```

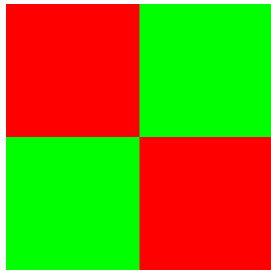
```

>   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

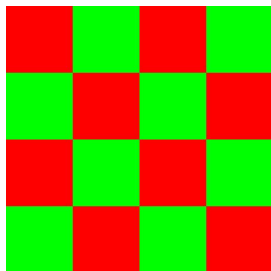
> rect :: Punto → Punto → Color → String
> rect (x,y) (x',y') (RGB r g b) =
>   vacio "rect" [( "x",show x),("y",show y),
>                 ("height",show (abs (x - x'))),
>                 ("width",show (abs (y - y'))),
>                 ("fill", "rgb ("++ show r ++", "++
>                         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

```

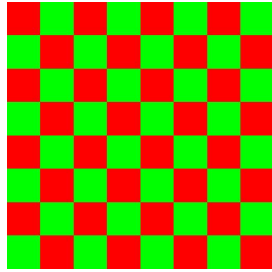
Enunciado 3 (diag) Construir un quadtree que represente una diagonal.



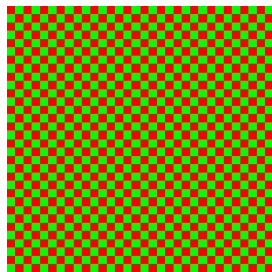
Enunciado 4 (repite) Construir una función *repite* que tome como argumento un quadtree y genere un nuevo quadtree repitiendo en cada cuadrante el quadtree original. Por ejemplo, al aplicar *repite* *diag* se obtendría:



Enunciado 5 (repiteN) Construir una función *repiteN* que tome como argumentos un número *n* y un quadtree *q* y genere un quadtree repitiendo *n* veces el quadtree *q*. Por ejemplo, para *n* = 3, se obtendrá la imagen:



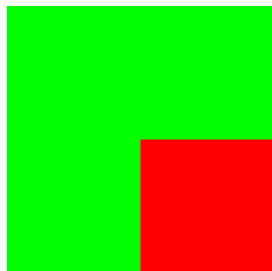
mientras que para $n = 5$, se obtendrá



Enunciado 6 (rota) Construir una función que tome un quadtree y lo rote 90 grados

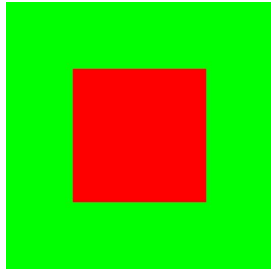
Enunciado 7 (esquina) Construir un programa que tome como parámetro un quadtree q y genere un nuevo quadtree con todos los cuadrantes de color verde salvo la esquina inferior derecha que tomará el valor q

Al dibujar, por ejemplo, esquina rojo se obtendría

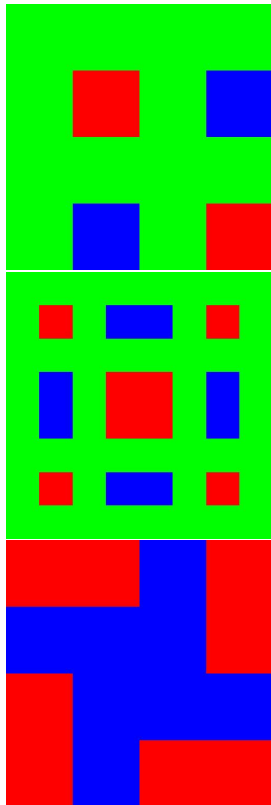


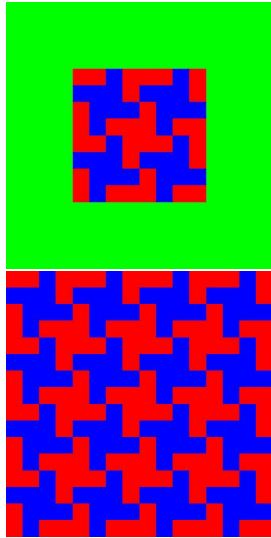
Enunciado 8 (rotaciones) Construir un programa que tome un quadtree y genere un nuevo quadtree a partir del anterior incluyendo en cada cuadrante una rotación de 90 grados del quadtree original

Al dibujar, por ejemplo, rotaciones (esquina rojo) se obtendría



Enunciado 9 (otros(Opcional)) *Construir otras figuras geométricas mediante combinaciones de rotaciones y esquinas*
Ejemplos:





Enunciado 10 (librería(Opcional)) Construir una librería de funciones que permitan manipular quadtrees para crear figuras geométricas diversas

Enunciado 11 (Octree) El siguiente programa define un tipo de datos recursivo que representa Octrees.

```
> module Octrees where
> data Color = RGB Float Float Float
>   deriving Show
> data OT = Vacio
>         | Cubo Color
>         | Esfera Color
>         | D OT OT OT OT OT OT OT OT
>   deriving Show
```

Algunos ejemplos:

```
> r = Cubo (RGB 1 0 0)
> g = Cubo (RGB 0 1 0)
> s = Esfera (RGB 0 0 1)
> v = Vacio
> ej = D r v v r v g s v
```

Compilar el programa y definir algunos ejemplos de octrees. Chequear el tipo de los ejemplos definidos.

Enunciado 12 (ver) Añadir al programa anterior el siguiente fragmento que incluye la función `verot` que permite visualizar una representación del quadtree en formato VRML

```

> w :: OT → IO ()
> w ot = writeFile "e.wrl"
>   (cabecera ++
>     viewPoint ++
>     background ++
>     verOT ((0,0,0),64) ot)

> type Point3D = (Float, Float, Float)
> type Dim = (Point3D, Float)

> verOT :: Dim → OT → String
> verOT _ Vacio = ""
> verOT (p,d) (Cubo c) = putBox p c (d,d,d)
> verOT (p,d) (Esfera c) = putSphere p c (d / 2)
> verOT ((x,y,z),d) (D c1 c2 c3 c4 c5 c6 c7 c8) =
>   let d2 = d / 2
>       d4 = d / 4
>   in if d <= 1 then ""
>      else verOT ((x-d4,y+d4,z+d4),d2) c1 ++
>            verOT ((x+d4,y+d4,z+d4),d2) c2 ++
>            verOT ((x-d4,y-d4,z+d4),d2) c3 ++
>            verOT ((x+d4,y-d4,z+d4),d2) c4 ++
>            verOT ((x-d4,y+d4,z-d4),d2) c5 ++
>            verOT ((x+d4,y+d4,z-d4),d2) c6 ++
>            verOT ((x-d4,y-d4,z-d4),d2) c7 ++
>            verOT ((x+d4,y-d4,z-d4),d2) c8

> cabecera =
>   "#VRML_V2.0_utf8\n\n"

> viewPoint = "Viewpoint_{_}++
>   ".....description_\\"Octrees\"_\n" ++
>   ".....orientation_0_0_1_0_\n" ++
>   ".....position_0_0.2_350_\n" ++ "}_\n"

> background =
>   "Background_{_skyColor_[_0_0.2_0.7,_0_0.5_1,_1_1_1_]_}"

> putBox p (RGB r g b) = translate p .
>                       color (r,g,b) .
>                       box

> putSphere :: Point3D → Color → Float → String
> putSphere p (RGB r g b) = translate p .
>                       color (r,g,b) .
>                       sphere

> box (sx,sy,sz) = "geometry_Box_" ++ llaves ("size_" ++ sh3 (sx,sy,sz))
> sphere r = "geometry_Sphere_" ++ llaves ("radius_" ++ show r)

```

```

> translate (x,y,z) s =
>   "Transform" ++ llaves ("translation_" ++ sh3 (x,y,z) ++ "\n" ++
>   "children_" ++ s ++ ".")
>
> color (r,g,b) s = "Shape" ++
>   llaves (s ++ "\nappearance_Appearance_" ++
>   llaves ("material_Material_" ++
>   llaves ("diffuseColor_" ++ sh3 (r,g,b))))
>
> llaves x = "{_" ++ x ++ "}"
>
> sh3 (x,y,z) = show x ++ "_" ++
>   show y ++ "_" ++
>   show z ++ "_"

```

Enunciado 13 (escalera) Construir un octree que represente una escalera.

En <http://www.di.uniovi.es/labra/PLF/prac/worlds/escal.wrl> puede observarse el resultado.

Enunciado 14 (repite) Construir una función repite que tome como argumento un octree y genere un nuevo octree repitiendo en cada cuadrante el octree original.

En <http://www.di.uniovi.es/labra/PLF/prac/worlds/rescal.wrl> puede observarse el resultado de visualizar repite escal.

Enunciado 15 (rota) Construir una función que tome un octree y lo rote 90 grados

En <http://www.di.uniovi.es/labra/PLF/prac/worlds/rotaes.wrl> puede observarse el resultado de visualizar rota escal.

Enunciado 16 (infinito) Construir un octree de resolución infinita

En <http://www.di.uniovi.es/labra/PLF/prac/worlds/inf.wrl> puede observarse un posible octree infinito

Enunciado 17 (cambia) Construir un programa que tome un octree y genere un nuevo octree intercambiando los cubos y las esferas

En <http://www.di.uniovi.es/labra/PLF/prac/worlds/cambaiinf.wrl> puede observarse el resultado de aplicar la función al octree anterior.

Enunciado 18 (esquinas) Construir un programa que tome un octree o y genere un nuevo octree con o en dos esquinas y todos los demás cuadrantes vacíos

En <http://www.di.uniovi.es/labra/PLF/prac/worlds/esquinainf.wrl> puede observarse el resultado de aplicar la función al octree infinito.

Enunciado 19 (otros(Opcional)) *Construir otras figuras geométricas mediante combinaciones de rotaciones y esquinas*

Construir una librería de combinadores y generadores de octrees

Enunciado 20 (galeria(Opcional)) *Construir una galería de figuras geométricas. Para ello, se generará un fichero HTML que contenga una tabla con las distintas figuras.*

En <http://www.di.uniovi.es/labra/PLF/prac/worlds/galeria.html> puede observarse una posible galería