

El quadtree de una imagen binaria

Una especificación algebraica en Nereus — material para alumnos

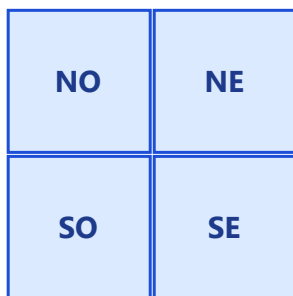
1. ¿Qué es un quadtree?

Una **imagen binaria** es una grilla cuadrada en la que cada celda es **negra** o **blanca**. Guardar la grilla celda por celda es caro cuando hay grandes zonas de un mismo color. El **region quadtree** aprovecha esas zonas uniformes.

La idea es una subdivisión recursiva en cuatro:

- si una región cuadrada es **de un solo color**, se guarda como una **hoja** (`black()` o `white()`);
- si la región es **mixta**, se parte en sus **cuatro cuadrantes** y se guarda como un nodo `quad(NO, NE, SO, SE)`.

El orden de los cuadrantes es siempre el mismo: Noroeste, Noreste, Suroeste, Sureste.



Los cuatro cuadrantes de una región y su orden fijo.

2. Los tres constructores

En Nereus el tipo `QuadTree` es una clase autónoma con **tres constructores básicos**:

```
BASIC CONSTRUCTORS black, white, quad
```

```
black: -> QuadTree;      -- hoja: región uniformemente negra
white: -> QuadTree;      -- hoja: región uniformemente blanca
quad:  QuadTree * QuadTree * QuadTree * QuadTree -> QuadTree;  -- NO,NE,SO,SE
```

Cualquier imagen binaria se escribe combinando estos tres. Por ejemplo, una imagen 2×2 con la fila de arriba negra y la de abajo blanca es:

```
quad(black(), black(), white(), white())
```

3. Las operaciones

Sobre esos constructores se definen, por recursión, las operaciones que observan y transforman la imagen:

Operación	Qué hace
<code>isUniform(t)</code>	¿es <code>t</code> una hoja, o un nodo mixto?
<code>isBlack(t)</code>	color de una hoja (operación <i>parcial</i> : pide <code>isUniform(t)</code>)
<code>numLeaves(t)</code>	cuántas hojas (regiones uniformes) tiene
<code>depth(t)</code>	profundidad del árbol
<code>blackArea(t, n)</code>	celdas negras, siendo <code>n</code> el lado de la región
<code>invert(t)</code>	la imagen con negro y blanco intercambiados
<code>isNormalized(t)</code> / <code>normalize(t)</code>	forma canónica: fusiona cuadrantes redundantes

Cada operación tiene un axioma para `black`, otro para `white` y otro para `quad`. El de `quad` es siempre el caso recursivo:

```

numLeaves(black())      = 1;
numLeaves(white())     = 1;
numLeaves(quad(a, b, c, d)) =
    numLeaves(a) + numLeaves(b) + numLeaves(c) + numLeaves(d);

```

4. Un ejemplo: una imagen de 4×4

Tomemos esta imagen de 4×4 celdas. Las líneas gruesas marcan la división en los cuatro cuadrantes de 2×2:

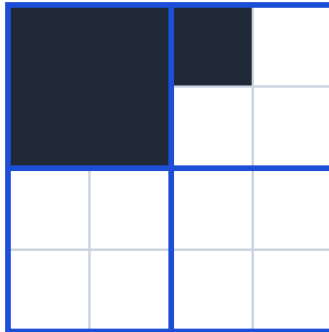


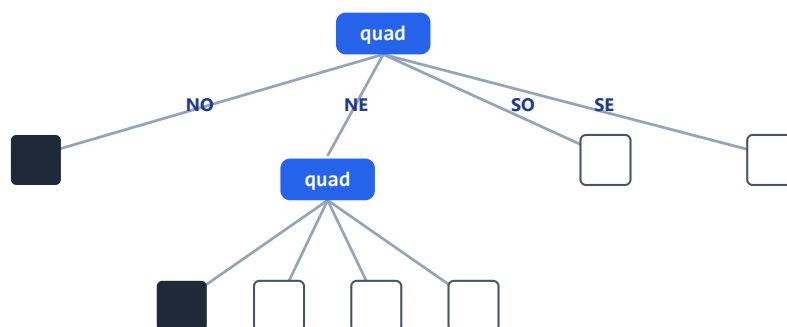
Imagen 4×4: 5 celdas negras. NO es todo negro; NE es mixto; SO y SE son todo blanco.

Mirando cuadrante por cuadrante:

- **NO** — las 4 celdas negras $\Rightarrow$ una hoja `black()` ;
- **NE** — mixto (su celda NO es negra, las otras tres blancas) $\Rightarrow$ `quad(black(), white(), white(), white())` ;
- **SO** — todo blanco $\Rightarrow$ `white()` ;
- **SE** — todo blanco $\Rightarrow$ `white()` .

El árbol completo es entonces:

```
quad(black(),
      quad(black(), white(), white(), white()),
      white(),
      white())
```



El quadtree de la imagen. En azul los nodos `quad` ; las hojas son cuadrados negros (`black()`) o blancos (`white()`).

5. Medir la imagen

Aplicando las operaciones recursivas sobre el árbol del ejemplo:

Operación	Cálculo	Resultado
numLeaves	1 (NO) + 4 (NE) + 1 (SO) + 1 (SE)	7 hojas
depth	1 + $\max(0, 1, 0, 0)$	2
blackArea(., 4)	4 (NO) + 1 (NE) + 0 (SO) + 0 (SE)	5 celdas

`blackArea` recibe el lado de la región (4) y lo va dividiendo por 2 al bajar: el cuadrante NO es una hoja negra de lado 2, así que aporta $2 \times 2 = 4$; dentro de NE, la única hoja negra es de lado 1 y aporta 1. Total: **5 celdas negras**, que es exactamente lo que se ve en la grilla.

6. Invertir la imagen

`invert` intercambia negro y blanco. Sobre una hoja cambia el color; sobre un nodo, invierte recursivamente sus cuatro cuadrantes:

```
invert(black())      = white();
invert(white())      = black();
invert(quad(a,b,c,d)) = quad(invert(a), invert(b), invert(c), invert(d));
```

Aplicado al ejemplo, produce el negativo de la imagen:

```
quad(white(),
     quad(white(), black(), black(), black()),
     black(),
     black())
```

7. `normalize`: la forma canónica

Una misma imagen puede escribirse de varias maneras. Si un nodo `quad` termina con sus **cuatro hijos siendo la misma hoja**, ese nodo es redundante: representa una región uniforme y debería ser una sola hoja.

`isNormalized` detecta si quedan nodos así, y `normalize` los **fusiona de abajo hacia arriba**. Por ejemplo:

```
normalize(quad(quad(white,white,white,white), white, white, white))
1) se normaliza cada cuadrante; el cuadrante NO es
   quad(white,white,white,white) → sus 4 hijos son la misma
   hoja ⇒ se fusiona en white()
2) ahora el nodo raíz es quad(white,white,white,white)
   ⇒ se fusiona otra vez en white()
```

El resultado, `white()`, es la **forma canónica**: la representación más compacta de esa imagen (una región toda blanca). Dos quadrees normalizados representan la misma imagen si y solo si son idénticos.

Idea clave. Las operaciones que solo *observan* (`numLeaves` , `depth` , `blackArea`) devuelven un número o un booleano. Las que *transforman* (`invert` , `normalize`) devuelven **otro QuadTree**: la recursión reconstruye el árbol nodo por nodo. En Nereus todo es inmutable — nunca se modifica un árbol, siempre se construye uno nuevo.

8. Para practicar

1. Escribí el quadtree de una imagen 4×4 totalmente negra. ¿Cuánto valen `numLeaves` y `depth` ? ¿Está normalizada?
2. Dibujá la imagen que representa `quad(white(), black(), black(), white())` y calculá `blackArea(·, 2)`.
3. Comprobá, sobre el ejemplo de la sección 4, que `invert(invert(t))` devuelve el árbol original.
4. ¿Por qué `isBlack` tiene una precondición? ¿Qué pregunta no tiene sentido hacerle a un nodo `quad` ?
5. Inventá un quadtree que *no* esté normalizado y aplicá `normalize` paso a paso.