Divide y Dibuja

> **!** Importante
>
> Antes de comenzar a programar asegúrese de haber descompactado toda la información, haber abierto en VS Code la solución dada, y ser capaz de compilarla. En caso de que su solución no compile por errores de configuración, llame immediatamente a uno de los profesores.

Un **quadtree** es una estructura de datos jerárquica que se utiliza para dividir un espacio bidimensional en cuatro cuadrantes o subregiones. Este tipo de árbol es especialmente útil para representar imágenes, ya que permite una representación eficiente de áreas donde los datos pueden ser homogéneos (como en imágenes en blanco y negro).

Cada nodo del quadtree puede tener hasta cuatro hijos, representando los cuadrantes superior izquierdo, superior derecho, inferior izquierdo e inferior derecho. La subdivisión continúa recursivamente hasta que se alcanza un nivel de detalle deseado o hasta que cada subregión contiene todos los píxeles del mismo color. Siempre se divide un nodo exactamente en 4 hijos, y exactamente a un cuarto del área para cada uno. O sea, todo nodo es hoja (negro o blanco) o es gris y tiene exactamente 4 hijos que dividen de forma uniforme el área del padre.

A modo de ejemplo, en la figura 1 se muestra una representación de una imagen de tamaño 8x8 con cierta región coloreada de negro, y el quadtree correspondiente.

Note que los hijos del primer nivel están nombrados como TL (top-left), TR (top-right), BR (bottom-right) y BL (bottom-left).

El primer hijo (TL) representa toda la región superior izquierda de la imagen, o sea, los 4x4 píxeles blancos arriba a la izquierda. Como todos los píxeles de esta región son blancos, basta un solo nodo para representar esta región completamente. De forma similar, el cuarto hijo (BL) representa la región completamente negra de abajo a la izquierda.

Por otro lado, el hijo TR (arriba a la derecha) contiene a su vez algunos píxeles negros y otros blancos, por lo que tiene que ser subdividido aún más. Tendrá entonces dos hijos completamente blancos y uno completamente negro. El cuarto hijo a su vez debe dividirse en tres

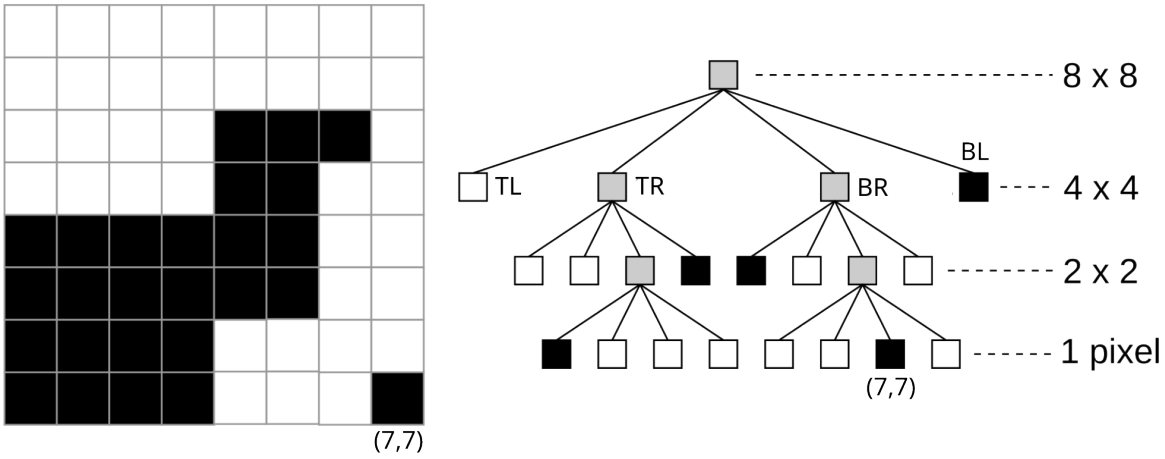

Figure 1

hijos blancos y uno negro. Todos estos nodos intermedios se clasifican entonces como de color gris.

Por último, el tercer hijo de la raíz (BR) tiene una estructura similar. Se ha señalado en la imagen el pixel de la esquina inferior derecha (7,7), que es un pixel de color negro aislado. En el árbol se puede apreciar cuál es el nodo que corresponde a este píxel.

Teniendo en cuenta la explicación anterior, usted debe implementar una clase que permita realizar las operaciones sobre un quadtree que represente una imagen cuadrada con tamaño potencia de 2. La operación fundamental que debe implementar consiste en dibujar un pixel de color negro o blanco, produciendo la subsecuente creación o eliminación de nodos según sea necesario para tener siempre un quadtree válido.

Además de esta operación, deberá implementar otras operaciones menores para garantizar que su quadtree funciona de manera correcta.

Interfaz

Los estudiantes deberán implementar la siguiente interfaz para representar su quadtree:

```
public enum QuadNodeColor
{
    White,
    Black,
    Gray,
}
```

```
public interface IQuadtree
{
    void DrawPixel(int x, int y, bool isBlack);
    int CountPixels();

    QuadNodeColor Color { get; }

    IQuadtree TopLeft { get; }
    IQuadtree TopRight { get; }
    IQuadtree BottomLeft { get; }
    IQuadtree BottomRight { get; }
}
```

Métodos a Implementar

1. **DrawPixel(int x, int y, bool isBlack)**: Este método debe permitir pintar un píxel en la posición (x, y) como negro o blanco. El método deberá subdividir y crear nuevos nodos o eliminarlos según sea necesario.
2. **CountPixels()**: Este método debe contar recursivamente cuántos píxeles negros hay en la imagen representada por el quadtree.
3. **Color**: Propiedad que devuelve el color del nodo, blanco o negro si es hoja, gris en caso contrario.
4. **TopLeft**: Propiedad que devuelve el hijo correspondiente al cuadrante superior izquierdo del quadtree.
5. **TopRight**: Propiedad que devuelve el hijo correspondiente al cuadrante superior derecho del quadtree.
6. **BottomLeft**: Propiedad que devuelve el hijo correspondiente al cuadrante inferior izquierdo del quadtree.
7. **BottomRight**: Propiedad que devuelve el hijo correspondiente al cuadrante inferior derecho del quadtree.

Además, deberán implementar el siguiente método en la clase estática **QuadTreeFactory**:

```
public static class QuadTreeFactory
{
    public IQuadTree Create(int size)
    {
        // Devuelva su instancia de IQuadTree aquí
    }
}
```

NOTA: El quadtree siempre será cuadrado y creado con un tamaño potencia de 2.