Dibujando Rectángulos

NOTA: Si usted está leyendo este documento sin haber extraído el compactado que se le entregó, ciérrelo ahora, extraiga todos los archivos en el escritorio, y siga trabajando desde ahí. Es un error común trabajar en la solución dentro del compactado, lo cual provoca que los cambios no se guarden. Si usted comete este error y entrega una solución vacía, no tendrá oportunidad de reclamar.

En diversas técnicas y algoritmos de particionamiento espacial se utilizan comúnmente estructuras de datos jerárquicas que subdividen el espacio recursivamente. El **quad-tree** es una de estas estructuras de datos, pues se utiliza para subdividir el plano en 4 cuadrantes iguales.

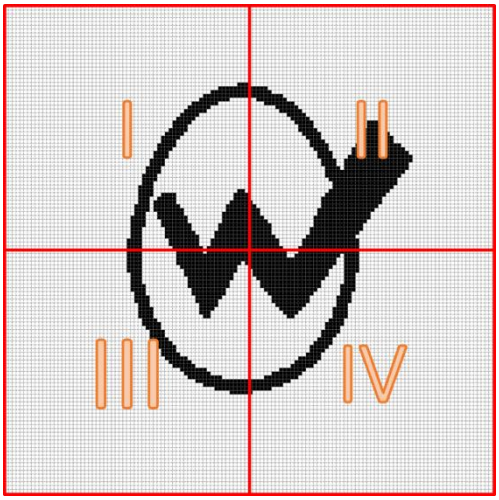


Figura 1. Subdivisión de una imagen en los 4 cuadrantes iguales

Un **quad-tree** es un árbol en el que cada nodo corresponde a un cuadrado. Cada nodo interno tiene exactamente 4 hijos, subdividiendo estos en 4 cuadrantes iguales el cuadrado que dicho nodo representa.

Entre los diferentes usos que tiene el **quad-tree** se encuentra la representación de imágenes binarias (imágenes en blanco y negro). En este caso los nodos tienen asociado un *color* que determina si la sección del espacio que cubre es totalmente blanca (le corresponde color ***blanco***), totalmente negra (le corresponde el color ***negro***) o tiene partes blancas y partes negras (le corresponde el color ***gris***).

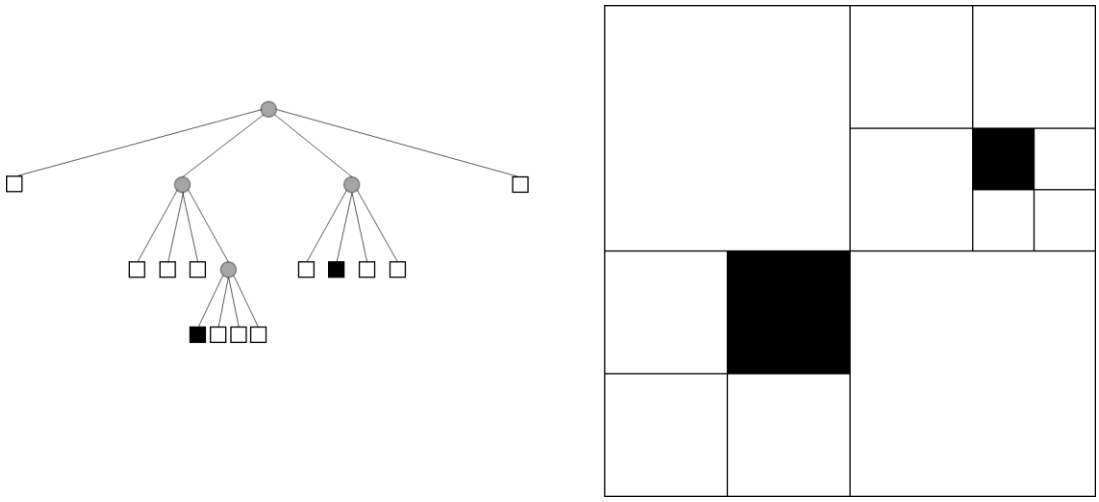


Figura 2: Quad-tree correspondiente a la imagen de la derecha

Los nodos de color *negro* o *blanco* son hojas, pues cubren una sección de un mismo color, mientras que los nodos grises son nodos internos. Esta estructura de datos ***describe en detalle*** la sección del espacio que representa y ***no posee redundancia*** alguna. Por tanto, cada nodo hoja es *blanco* o *negro*, y cada nodo interno es *gris*; además todo nodo *gris* tiene todos sus hijos de color *gris* o al menos dos hijos de diferente color.

Debido a la característica de dividir recursivamente en cuadrados la imagen, se usan imágenes con anchura y altura potencia de 2. Es decir, las imágenes serán de 2^N píxeles de ancho por 2^N píxeles de alto, para algún N.La interfaz `ILienzo` representa una imagen modelada mediante un **quad-tree** que permite dibujar rectángulos sobre ella.

```
public interface ILienzo
{
    /// <summary>
    /// Devuelve la cantidad de píxeles de la imagen.
    /// </summary>
    int Resolucion { get; }

    /// <summary>
    /// Devuelve la cantidad de nodos de color blancos que tiene el quad-tree.
    /// </summary>
    int CantidadDeNodosBlancos { get; }

    /// <summary>
    /// Devuelve la cantidad de nodos de color negro que tiene el quad-tree.
    /// </summary>
    int CantidadDeNodosNegros { get; }

    /// <summary>
    /// Devuelve la cantidad de nodos grises que tiene el quad-tree.
    /// </summary>
    int CantidadDeNodosGrises { get; }

    /// <summary>
    /// Rellena el rectángulo dado con color negro.
```

```
        /// </summary>
        /// <param name="fila">Fila donde comienza el rectángulo</param>
        /// <param name="columna">Columna donde comienza el rectángulo</param>
        /// <param name="ancho">Ancho del rectángulo</param>
        /// <param name="alto">Alto del rectángulo</param>
        void Dibuja(int fila, int columna, int ancho, int alto);

        /// <summary>
        /// Comprueba si un pixel ha sido pintado.
        /// </summary>
        /// <param name="fila">Fila del pixel</param>
        /// <param name="columna">Columna del pixel</param>
        /// <returns>Devuelve true en caso de que el pixel haya sido
        /// pintado, y false en caso contrario.</returns>
        bool EstaPintado(int fila, int columna);
    }
```

La propiedad `Resolucion` devuelve el tamaño de la imagen, es decir, cuántos píxeles tiene.

Dado que la imagen estará modelada mediante un **quad-tree**, la interfaz `ILienzo` contiene las propiedades `CantidadDeNodosBlancos`, `CantidadDeNodosNegros` y `CantidadDeNodosGrises` para consultar el estado de la estructura.

Inicialmente, la imagen inicialmente tiene ***todos sus píxeles blancos***. El método `Dibuja` modifica la imagen ***rellenando de color negro*** el rectángulo definido por sus parámetros. Este rectángulo puede ***cubrir o solaparse*** con píxeles que ya hayan sido rellenados; en este caso, esos píxeles no necesitan ser modificados, pero tenga en cuenta que puede que el **quad-tree** sufra cambios.

El método `EstaPintado` comprueba si un pixel tiene color negro.

Usted debe haber recibido junto a este documento una solución de Visual Studio con dos proyectos: una biblioteca de clases (*Class Library*) y una aplicación de consola (*Console Application*). Usted debe completar la implementación de la clase `Lienzo` en el namespace Weboo.Examen que ya implementa la interfaz `ILienzo` y cumpla con la funcionalidad descrita sobre la misma.

NOTA: Todo el código de la solución debe estar en este proyecto (biblioteca de clases), pues es el único código que será evaluado. Usted puede adicionar todo el código que considere necesario, pero no puede cambiar los nombres del namespace, clase o método mostrados. De lo contrario, el probador automático fallará y su prueba quedará invalidada. En particular, es imprescindible que usted no cambie los parámetros del constructor de la clase `Lienzo`. Por supuesto, usted puede (y debe) adicionar todo el código que necesite al cuerpo del constructor para inicializar sus estructuras de datos.

```
namespace Weboo.Examen
{
    public class Lienzo : ILienzo
    {
        public Lienzo(int N)
        {
            throw new NotImplementedException();
        }
```

```
        public int Resolucion
        {
            get { throw new NotImplementedException(); }
        }

        public int CantidadDeNodosBlancos
        {
            get { throw new NotImplementedException(); }
        }

        public int CantidadDeNodosNegros
        {
            get { throw new NotImplementedException(); }
        }

        public int CantidadDeNodosGrises
        {
            get { throw new NotImplementedException(); }
        }

        public void Dibuja(int fila, int columna, int ancho, int alto)
        {
            throw new NotImplementedException();
        }

        public bool EstaPintado(int fila, int columna)
        {
            throw new NotImplementedException();
        }
    }
}
```

Puede asumir que:

- Los parámetros del método `Dibuja` siempre especificarán un rectángulo dentro de la imagen.
- Los parámetros del método `EstaPintado` siempre especificarán una posición válida dentro de la imagen.

Ejemplo

```
ILienzo lienzo = new Lienzo(4);
Console.WriteLine("Cantidad de Píxeles: {0} ", lienzo.Resolucion);
// Cantidad de Píxeles: 256
Console.WriteLine("Nodos Blancos: {0}", lienzo.CantidadDeNodosBlancos);
// Nodos Blancos: 1
Console.WriteLine("Nodos Negros: {0}", lienzo.CantidadDeNodosNegros);
// Nodos Negros: 0
```

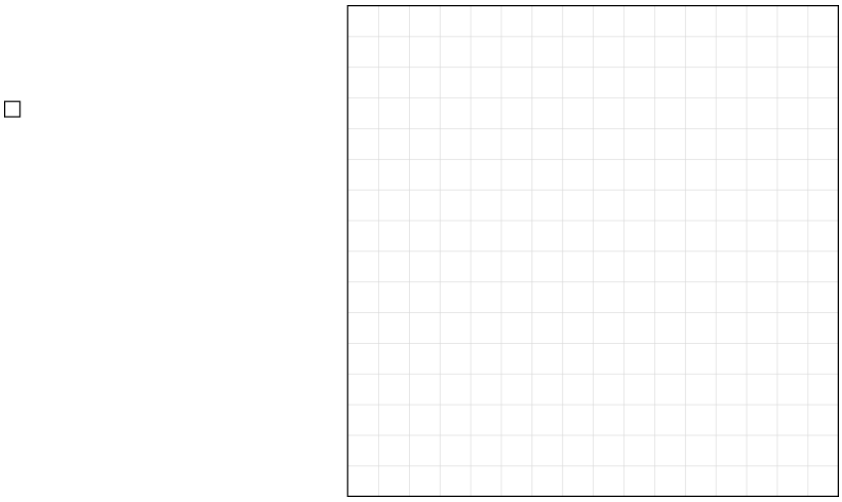
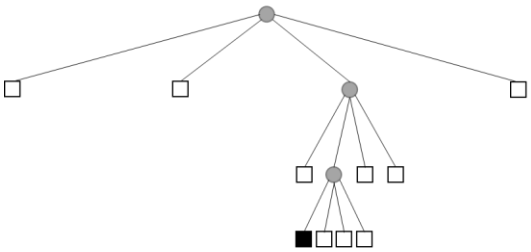


Figura 3. Imagen después de inicializar un lienzo de 16x16

```
lienzo.Dibuja(8, 4, 2, 2);
Console.WriteLine("Pixel ({0}, {1}): {2}", 8, 5, lienzo.EstaPintado(8, 5));
// Pixel (8, 5): true
Console.WriteLine("Nodos Blancos: {0}", lienzo.CantidadDeNodosBlancos);
// Nodos Blancos: 9
Console.WriteLine("Nodos Negros: {0}", lienzo.CantidadDeNodosNegros);
// Nodos Negros: 1
Console.WriteLine("Nodos Grises: {0}", lienzo.CantidadDeNodosGrises);
// Nodos Grises: 3
```



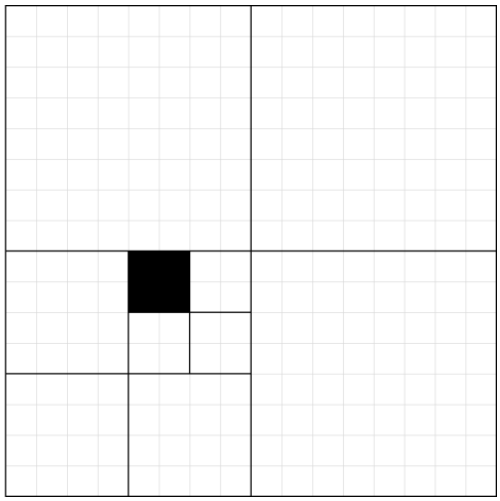


Figura 4. Después de dibujar el primer rectángulo

```
lienzo.Dibuja(8, 6, 2, 4);
Console.WriteLine("Pixel ({0}, {1}): {2}", 10, 7, lienzo.EstaPintado(10, 7));
// Pixel (10, 7): true
Console.WriteLine("Nodos Blancos: {0}", lienzo.CantidadDeNodosBlancos);
// Nodos Blancos: 7
Console.WriteLine("Nodos Negros: {0}", lienzo.CantidadDeNodosNegros);
// Nodos Negros: 3
```

```
Console.WriteLine("Nodos Grises: {0}", lienzo.CantidadDeNodosGrises);
// Nodos Grises: 3
```

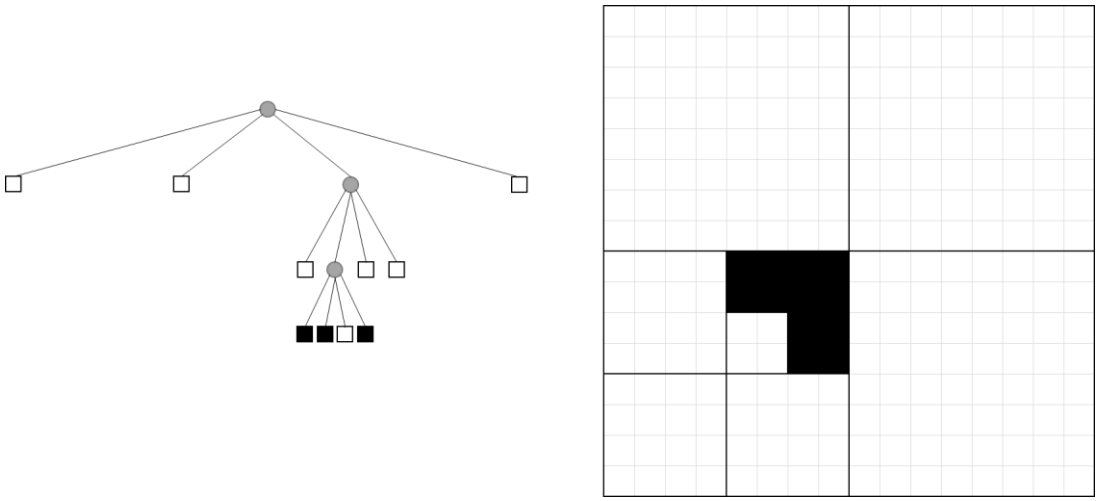


Figura 5. Después de dibujar el segundo rectángulo

Note que al quedar un nodo con todos sus hijos en negro este pasa entonces a ser negro y sus hijos desparecen (ejemplo a continuación).

```
lienzo.Dibuja(10, 4, 2, 2);
Console.WriteLine("Pixel ({0}, {1}): {2}", 8, 7, lienzo.EstaPintado(8, 7));
// Pixel (8, 7): true
Console.WriteLine("Nodos Blancos: {0}", lienzo.CantidadDeNodosBlancos);
// Nodos Blancos: 6
Console.WriteLine("Nodos Negros: {0}", lienzo.CantidadDeNodosNegros);
// Nodos Negros: 1
Console.WriteLine("Nodos Grises: {0}", lienzo.CantidadDeNodosGrises);
// Nodos Grises: 2
```

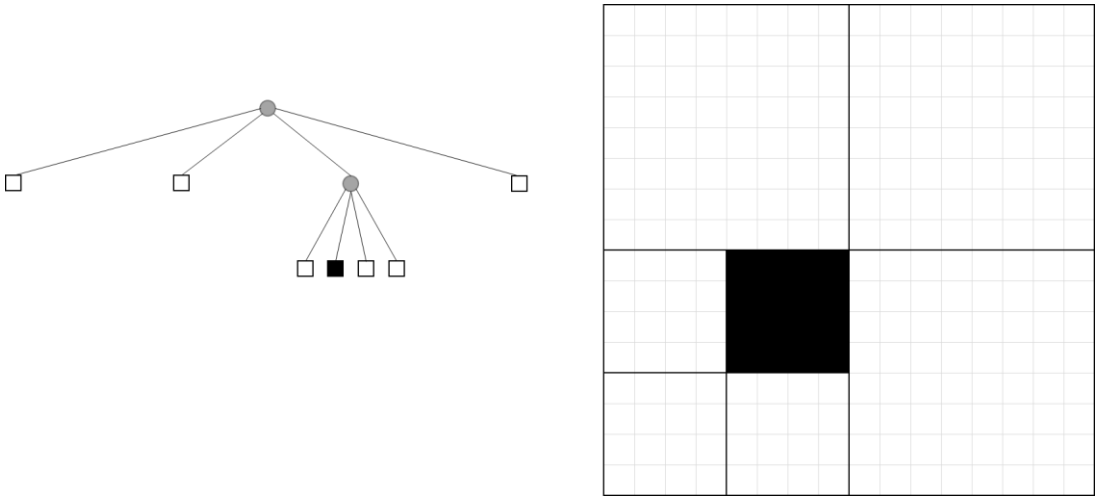
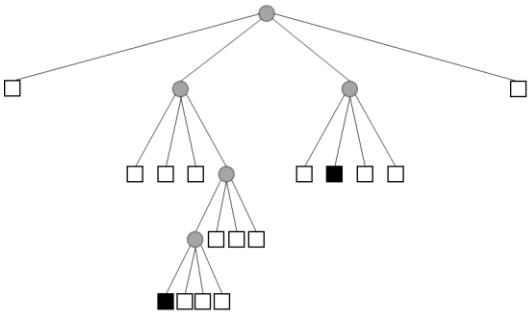


Figura 6. Después de dibujar el tercer rectángulo

```
lienzo.Dibuja(4, 12, 1, 1);
Console.WriteLine("Pixel ({0}, {1}): {2}", 4, 13, lienzo.EstaPintado(4, 13));
```

```
// Pixel (4, 13): false
Console.WriteLine("Nodos Blancos: {0}", lienzo.CantidadDeNodosBlancos);
// Nodos Blancos: 14
Console.WriteLine("Nodos Negros: {0}", lienzo.CantidadDeNodosNegros);
// Nodos Negros: 2
Console.WriteLine("Nodos Grises: {0}", lienzo.CantidadDeNodosGrises);
// Nodos Grises: 5
```



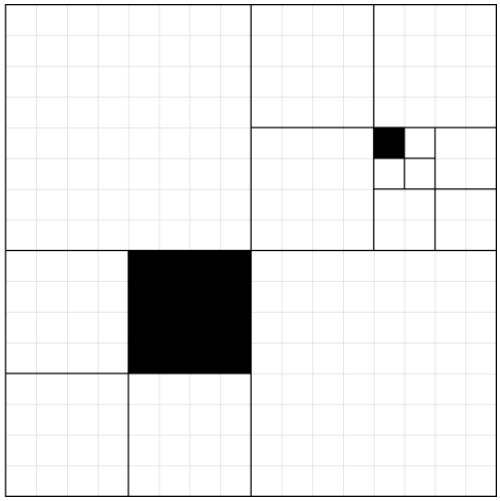
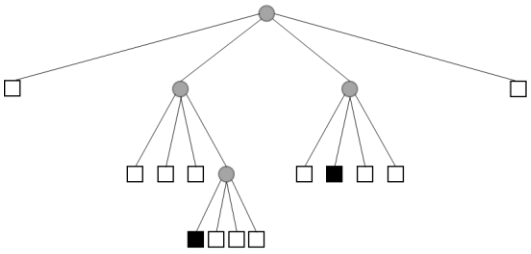


Figura 7. Después de dibujar el cuarto rectángulo

```
lienzo.Dibuja(4, 12, 2, 2);
Console.WriteLine("Pixel ({0}, {1}): {2}", 10, 10, lienzo.EstaPintado(10, 10));
// Pixel (10, 10): False
Console.WriteLine("Nodos Blancos: {0}", lienzo.CantidadDeNodosBlancos);
// Nodos Blancos: 11
Console.WriteLine("Nodos Negros: {0}", lienzo.CantidadDeNodosNegros);
// Nodos Negros: 2
Console.WriteLine("Nodos Grises: {0}", lienzo.CantidadDeNodosGrises);
// Nodos Grises: 4
```



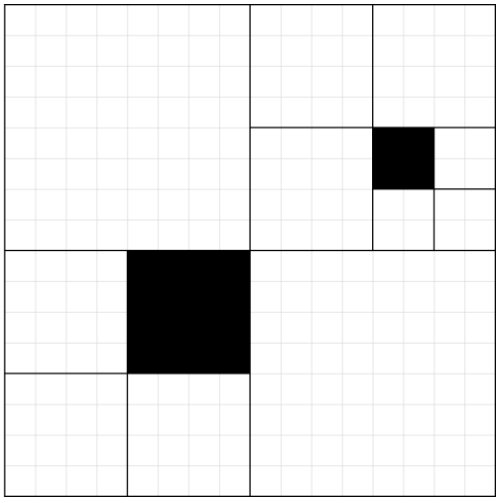


Figura 8. Después de dibujar el quinto rectángulo