PED941
Ciclo 01-2024
G01T

UDB VIRTUAL
Universidad Don Bosco, El Salvador

Guia 2

Docente: Carmen Celia Morales Samayoa

Apellidos	Nombres	Carné
González Romero	Irvin Eduardo	GR202825

REPOSITORIO: https://github.com/Irvin-g/guia2-PED

Solo mostrare todo el código funcional con el análisis completo de lo que se realizó.

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Arbo_AVL
{
    40 referencias
    class AVL
    {
        public int valor;
        public String tipoRotacion;
        public AVL NodoIzquierdo;
        public AVL NodoDerecho;
        public AVL NodoPadre;
        public int altura;
        public Rectangle prueba;
        private DibujaAVL arbol;
        1 referencia
        public AVL()
        {

        }

        0 referencias
        public DibujaAVL Arbol
        {
            get { return arbol;}
            set { arbol = value; }
        }

        //Constructor
        3 referencias
        public AVL(int valorNuevo, AVL izquierdo, AVL derecho, AVL padre)
        {
            valor = valorNuevo;
            NodoIzquierdo = izquierdo;
            NodoDerecho = derecho;
            NodoPadre = padre;
            altura = 0;
            tipoRotacion = ""; //Inicializa la variable del tipo de rotacion.
        }
```

```
        //Funcion para insertar un valor nuevo en el arbol AVL
        3 referencias
        public AVL Insertar(int valorNuevo, AVL raiz)
        {
            if (raiz == null)
                raiz = new AVL(valorNuevo, null, null, null);
            else if (valorNuevo < raiz.valor)
                raiz.NodoIzquierdo = Insertar(valorNuevo, raiz.NodoIzquierdo);
            else if (valorNuevo > raiz.valor)
                raiz.NodoDerecho = Insertar(valorNuevo, raiz.NodoDerecho);
            else
                MessageBox.Show("Valor Existente en el Arbol", "Error", MessageBoxButtons.OK);

            //Realiza las rotaciones simples o dobles segun el caso
            if(Alturas(raiz.NodoIzquierdo) - Alturas(raiz.NodoDerecho) == 2)
            {
                if (valorNuevo < raiz.NodoIzquierdo.valor)
                {
                    raiz = RotacionIzquierdaSimple(raiz);
                    raiz.tipoRotacion = "Rotación hacia izquierda simple";
                }
                else
                {
                    raiz = RotacionIzquierdaDoble(raiz);
                    raiz.tipoRotacion = "Rotación hacia izquierda doble";
                }

            }
            if (Alturas(raiz.NodoDerecho) - Alturas(raiz.NodoIzquierdo) == 2)
            {
                if (valorNuevo > raiz.NodoDerecho.valor)
                {
                    raiz = RotacionDerechaSimple(raiz);
                    raiz.tipoRotacion = "Rotación hacia derecha simple";
                }
                else
                {
                    raiz = RotacionDerechaDoble(raiz);
                    raiz.tipoRotacion = "Rotación hacia derecha doble";
                }
            }
            raiz.altura = max(Alturas(raiz.NodoIzquierdo), Alturas(raiz.NodoDerecho) + 1);
            return raiz;
        }
```

```
        //Funcion para obtener que rama es mayor
        5 referencias
        private static int max(int lhs, int rhs)
        {
            return lhs > rhs ? lhs : rhs;
        }
        //Obtiene la altura
        22 referencias
        private static int Alturas(AVL raiz)
        {
            return raiz == null ? -1 : raiz.altura;
        }


        AVL nodoE, nodoP;
        //Funcion eliminar
        3 referencias
        public AVL Eliminar(int valorEliminar, ref AVL raiz)
        {
            if(raiz != null)
            {
                if(valorEliminar < raiz.valor)
                {
                    nodoE = raiz;
                    Eliminar(valorEliminar, ref raiz.NodoIzquierdo);
                }
                else
                {
                    if(valorEliminar > raiz.valor)
                    {
                        nodoE = raiz;
                        Eliminar(valorEliminar, ref raiz.NodoDerecho);
                    }
                    else
                    {
                        //Posicionado sobre el elemento a eliminar
                        AVL NodoEliminar = raiz;
                        if(NodoEliminar.NodoDerecho == null)
                        {
                            raiz = NodoEliminar.NodoIzquierdo;
                            if(Alturas(nodoE.NodoIzquierdo) - Alturas(nodoE.NodoDerecho) == 2)
                            {
                                //MessageBox.Show("nodoE" + nodoE.valor.ToString());
                                if (valorEliminar < nodoE.valor)
                                    nodoP = RotacionIzquierdaSimple(nodoE);
                                else
                                    nodoE = RotacionDerechaSimple(nodoE);
```

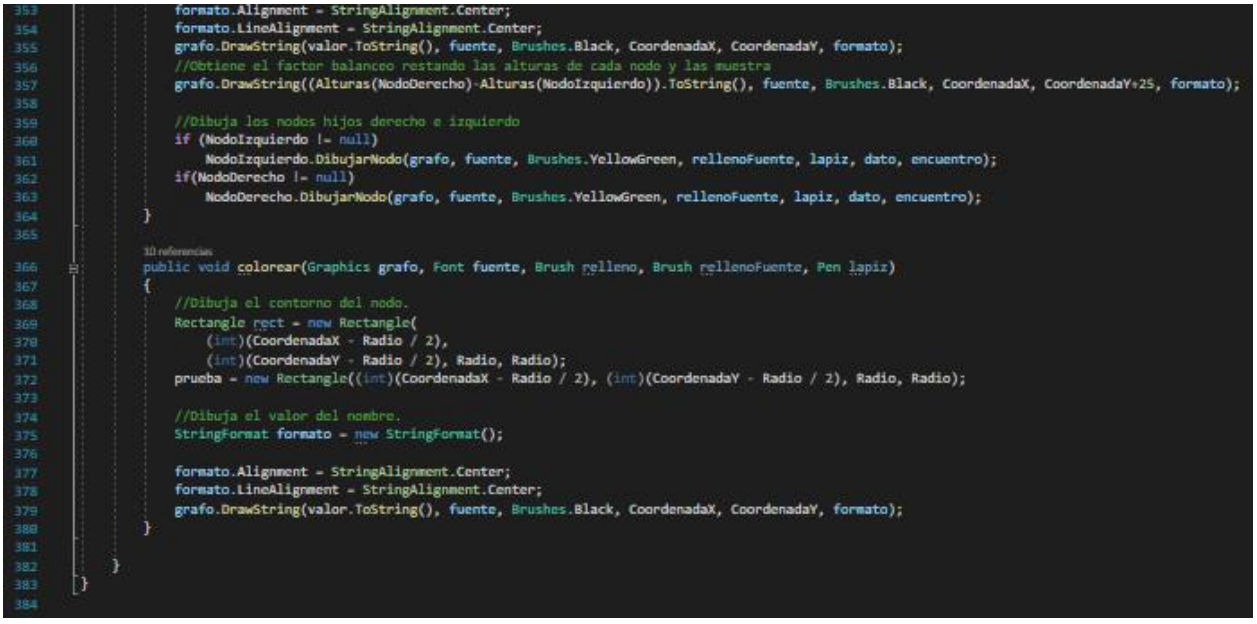
```
                            }
                            if(Alturas(nodoE.NodoDerecho) - Alturas(nodoE.NodoIzquierdo) == 2)
                            {
                                if (valorEliminar > nodoE.valor)
                                    nodoE = RotacionDerechaSimple(nodoE);
                                else
                                    nodoE = RotacionDerechaDoble(nodoE);
                                nodoP = RotacionDerechaSimple(nodoE);
                            }
                        }
                        else
                        {
                            if (NodoEliminar.NodoIzquierdo == null)
                                raiz = NodoEliminar.NodoDerecho;
                            else
                            {
                                if(Alturas(raiz.NodoIzquierdo) - Alturas(raiz.NodoDerecho) > 0)
                                {
                                    AVL AuxiliarNodo = null;
                                    AVL Auxiliar = raiz.NodoIzquierdo;
                                    bool Bandera = false;
                                    while(Auxiliar.NodoDerecho != null)
                                    {
                                        AuxiliarNodo = Auxiliar;
                                        Auxiliar = Auxiliar.NodoDerecho;
                                        Bandera = true;
                                    }
                                    raiz.valor = Auxiliar.valor;
                                    NodoEliminar = Auxiliar;
                                    if (Bandera == true)
                                        AuxiliarNodo.NodoDerecho = Auxiliar.NodoIzquierdo;
                                    else
                                        raiz.NodoIzquierdo = Auxiliar.NodoIzquierdo;

                                    //Realiza las rotaciones simples o dobles dependiendo del caso
                                }
                                else
                                {
                                    if (Alturas(raiz.NodoDerecho) - Alturas(raiz.NodoIzquierdo) > 0)
                                    {
                                        AVL AuxiliarNodo = null;
                                        AVL Auxiliar = raiz.NodoIzquierdo;
                                        bool Bandera = false;
                                        while(Auxiliar.NodoDerecho != null)
                                        {
                                            AuxiliarNodo = Auxiliar;
```

```
                                    Auxiliar = Auxiliar.NodoDerecho;
                                    Bandera = true;
                                }
                                raiz.valor = Auxiliar.valor;
                                NodoEliminar = Auxiliar;
                                if (Bandera == true)
                                    AuxiliarNodo.NodoDerecho = Auxiliar.NodoIzquierdo;
                                else
                                    raiz.NodoDerecho = Auxiliar.NodoIzquierdo;
                            }
                        }
                    }
                }
            }
        }
        else
            MessageBox.Show("No existe Nodo en el arbol", "Error", MessageBoxButtons.OK);
        return nodoP;
    }

    //Funciones de las rotaciones

    //Rotacion izquierda simple.
    4 referencias
    private static AVL RotacionIzquierdaSimple(AVL k2)
    {
        AVL k1 = k2.NodoIzquierdo;
        k2.NodoIzquierdo = k1.NodoDerecho;
        k1.NodoDerecho = k2;
        k2.altura = max(Alturas(k2.NodoIzquierdo), Alturas(k2.NodoDerecho)) + 1;
        k1.altura = max(Alturas(k1.NodoIzquierdo), k2.altura) + 1;
        return k1;
    }

    //Rotacion derecha simple.
    6 referencias
    private static AVL RotacionDerechaSimple(AVL k1)
    {
        AVL k2 = k1.NodoDerecho;
        k1.NodoDerecho = k2.NodoIzquierdo;
        k2.NodoIzquierdo = k1;
        k1.altura = max(Alturas(k1.NodoIzquierdo), Alturas(k1.NodoDerecho)) + 1;
        k2.altura = max(Alturas(k2.NodoDerecho), k1.altura) + 1;
        return k2;
    }
```

```
    //Doble rotacion izquierda.
    1 referencia
    private static AVL RotacionIzquierdaDoble(AVL k3)
    {
        k3.NodoIzquierdo = RotacionDerechaSimple(k3.NodoIzquierdo);
        return RotacionIzquierdaSimple(k3);
    }

    //Doble Rotacion Derecha.
    2 referencias
    private static AVL RotacionDerechaDoble(AVL k1)
    {
        k1.NodoDerecho = RotacionIzquierdaSimple(k1.NodoDerecho);
        return RotacionDerechaSimple(k1);
    }

    //Obtiene altura del arbol.
    5 referencias
    public int getAltura(AVL nodoActual)
    {
        if (nodoActual == null)
            return 0;
        else
            return 1 + Math.Max(getAltura(nodoActual.NodoIzquierdo), getAltura(nodoActual.NodoDerecho));

    }

    //Buscar un valor en el arbol.
    3 referencias
    public void buscar(int valorBuscar, AVL raiz)
    {
        if(raiz != null)
        {
            if(valorBuscar < raiz.valor)
                buscar(valorBuscar, raiz.NodoIzquierdo);
            else
            {
                if (valorBuscar > raiz.valor)
                    buscar(valorBuscar, raiz.NodoDerecho);
            }
        }
        else
            MessageBox.Show("No se encontro el valor", "Error", MessageBoxButtons.OK);
    }

    //Funciones para dibujar el arbol.
    private const int Radio = 30;
```

```
265         private const int DistanciaH = 40;
266         private const int DistanciaV = 10;
267
268         private int CoordenadaX;
269         private int CoordenadaY;
270
271         //Encuentra la posicion donde debe crearse el nodo.
            3 referencias
272         public void PosicionNodo(ref int xmin, int ymin)
273         {
274             int aux1, aux2;
275             CoordenadaY = (int)(ymin + Radio / 2);
276
277             //Obtiene la posicion del sub-arbol izquierdo.
278             if (NodoIzquierdo != null)
279                 NodoIzquierdo.PosicionNodo(ref xmin, ymin + Radio + DistanciaV);
280             if ((NodoIzquierdo != null) && (NodoDerecho != null))
281                 xmin += DistanciaH;
282
283             //Si existen ambos nodos dejara espacio entre ellos
284             if (NodoDerecho != null)
285                 NodoDerecho.PosicionNodo(ref xmin, ymin + Radio + DistanciaV);
286
287             // Posicion de ambos nodos
288             if(NodoIzquierdo != null)
289             {
290                 if (NodoDerecho != null)
291                     //Centro entre los nodos
292                     CoordenadaX = (int)((NodoIzquierdo.CoordenadaX + NodoDerecho.CoordenadaX) / 2);
293                 else
294                 {
295                     //No hay nodo derecho, centrar al izquierdo
296                     aux1 = NodoIzquierdo.CoordenadaX;
297                     NodoIzquierdo.CoordenadaX = CoordenadaX - 40;
298                     CoordenadaX = aux1;
299                 }
300             }
301             else if(NodoDerecho != null)
302             {
303                 aux2 = NodoDerecho.CoordenadaX;
304                 //No hay nodo izquierdo, centrar al derecho
305                 NodoDerecho.CoordenadaX = CoordenadaX - 40;
306                 CoordenadaX = aux2;
307             }
308             else
309             {
```

```
310                 //Nodo hoja
311                 CoordenadaX = (int)(xmin + Radio / 2);
312                 xmin += Radio;
313             }
314         }
315
316         //Dibuja las ramas de los nodos izquierdo y derecho
            3 referencias
317         public void DibujarRamas(Graphics grafo, Pen Lapiz)
318         {
319             if(NodoIzquierdo != null)
320             {
321                 grafo.DrawLine(Lapiz, CoordenadaX, CoordenadaY, NodoIzquierdo.CoordenadaX, NodoIzquierdo.CoordenadaY);
322                 NodoIzquierdo.DibujarRamas(grafo, Lapiz);
323             }
324             if(NodoDerecho != null)
325             {
326                 grafo.DrawLine(Lapiz, CoordenadaX, CoordenadaY, NodoDerecho.CoordenadaX, NodoDerecho.CoordenadaY);
327                 NodoDerecho.DibujarRamas(grafo, Lapiz);
328             }
329         }
330
331         //Dibuja el nodo en la posición especificada
            3 referencias
332         public void DibujarNodo(Graphics grafo, Font fuente, Brush relleno, Brush rellenoFuente, Pen lapiz, int dato, Brush encuentro)
333         {
334             //Dibuja el contorno del nodo.
335             Rectangle rect = new Rectangle(
336                 (int)(CoordenadaX - Radio/2),
337                 (int)(CoordenadaY - Radio / 2), Radio, Radio);
338
339             if (valor == dato)
340                 grafo.FillEllipse(encuentro, rect);
341             else
342             {
343                 grafo.FillEllipse(encuentro, rect);
344                 grafo.FillEllipse(relleno, rect);
345             }
346             grafo.DrawEllipse(lapiz, rect);
347
348             //Dibuja el valor del nodo.
349             StringFormat formato = new StringFormat();
350             StringFormat formatoAltura = new StringFormat();
351             formatoAltura.Alignment = StringAlignment.Center;
352             formatoAltura.LineAlignment = StringAlignment.Center;
```



```
            formato.Alignment = StringAlignment.Center;
            formato.LineAlignment = StringAlignment.Center;
            grafo.DrawString(valor.ToString(), fuente, Brushes.Black, CoordenadaX, CoordenadaY, formato);
            //Obtiene el factor balanceo restando las alturas de cada nodo y las muestra
            grafo.DrawString((Alturas(NodoDerecho)-Alturas(NodoIzquierdo)).ToString(), fuente, Brushes.Black, CoordenadaX, CoordenadaY+25, formato);

            //Dibuja los nodos hijos derecho e izquierdo
            if (NodoIzquierdo != null)
                NodoIzquierdo.DibujarNodo(grafo, fuente, Brushes.YellowGreen, rellenoFuente, lapiz, dato, encuentro);
            if(NodoDerecho != null)
                NodoDerecho.DibujarNodo(grafo, fuente, Brushes.YellowGreen, rellenoFuente, lapiz, dato, encuentro);
        }

        10 referencias
        public void colorear(Graphics grafo, Font fuente, Brush relleno, Brush rellenoFuente, Pen lapiz)
        {
            //Dibuja el contorno del nodo.
            Rectangle rect = new Rectangle(
                (int)(CoordenadaX - Radio / 2),
                (int)(CoordenadaY - Radio / 2), Radio, Radio);
            prueba = new Rectangle((int)(CoordenadaX - Radio / 2), (int)(CoordenadaY - Radio / 2), Radio, Radio);

            //Dibuja el valor del nombre.
            StringFormat formato = new StringFormat();

            formato.Alignment = StringAlignment.Center;
            formato.LineAlignment = StringAlignment.Center;
            grafo.DrawString(valor.ToString(), fuente, Brushes.Black, CoordenadaX, CoordenadaY, formato);
        }

    }
}
```

Análisis

Este código es una implementación de un Árbol AVL en C#

1. Espacio de nombres y directivas using: Define el espacio de nombres Arbo_AVL y agrega las directivas using necesarias para importar funcionalidades de otros espacios de nombres, como los controles de Windows Forms.

2. Clase AVL: Define la clase AVL, que representa un nodo en el árbol AVL. Contiene los siguientes campos:

- valor: El valor numérico almacenado en el nodo.
- tipoRotacion: Una cadena que indica el tipo de rotación realizada en el nodo.
- NodoIzquierdo, NodoDerecho, NodoPadre: Referencias a los nodos izquierdo, derecho y padre, respectivamente.
- altura: La altura del nodo en el árbol.
- prueba: Un rectángulo utilizado para dibujar el nodo en la interfaz gráfica.
- arbol: Un objeto de la clase DibujaAVL utilizado para dibujar el árbol AVL.

3. Constructores: Define varios constructores para la clase AVL.

4. Método Insertar: Inserta un nuevo valor en el árbol AVL y realiza rotaciones simples o dobles según sea necesario para mantener el balance del árbol.

5. Método Eliminar: Elimina un valor del árbol AVL y realiza las rotaciones necesarias para mantener el balance del árbol.

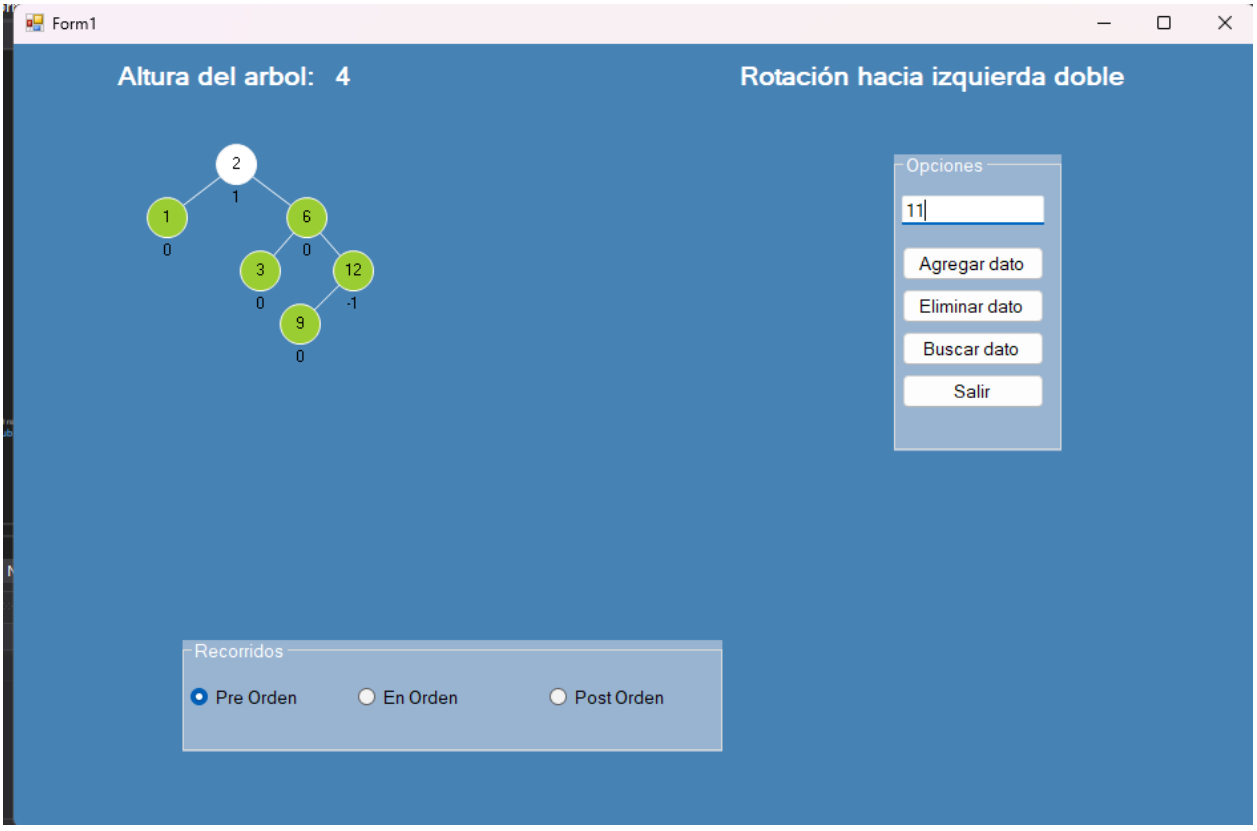
6. Métodos de Rotación: Implementan las rotaciones simples y dobles necesarias para mantener el balance del árbol AVL.

7. Métodos para dibujar el árbol: Contienen lógica para calcular las posiciones de los nodos y dibujar el árbol en una interfaz gráfica utilizando la clase Graphics de Windows Forms.

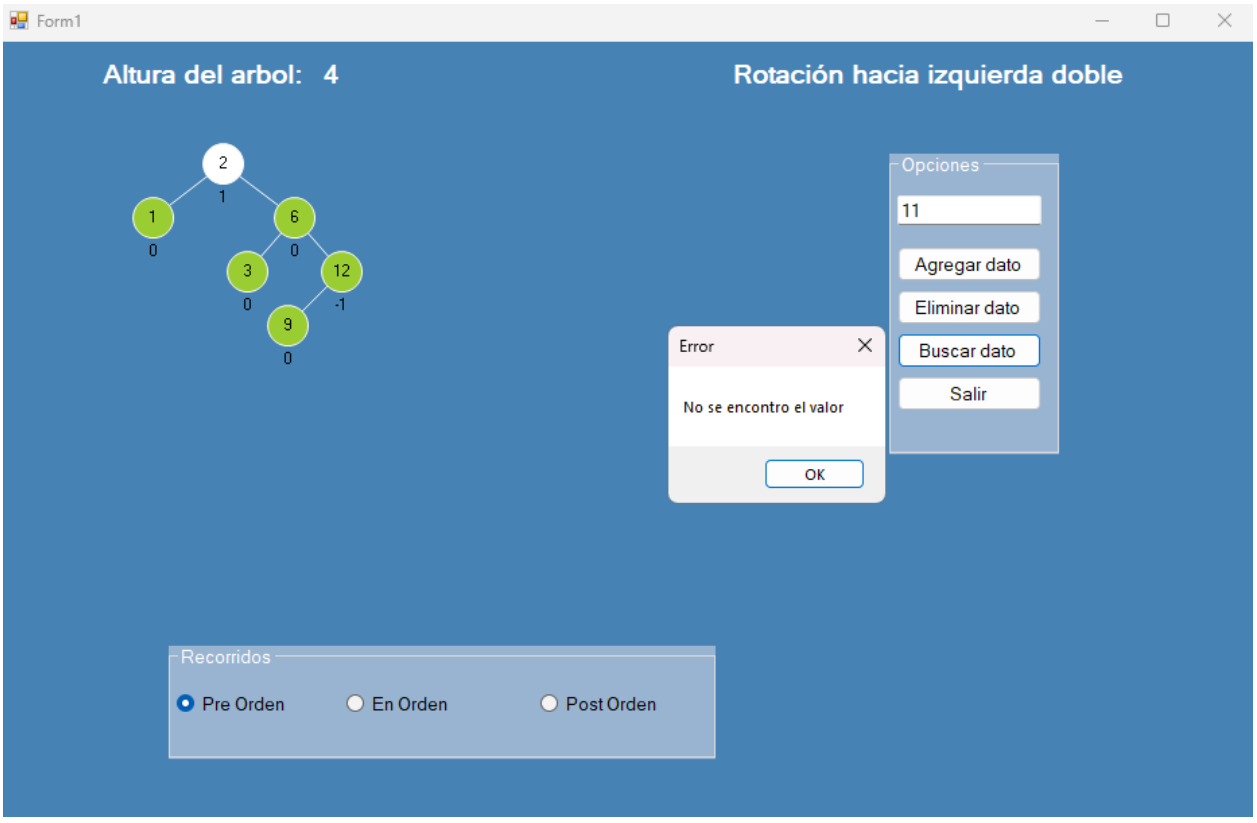
8. Método colorear: Similar a los métodos de dibujo, pero utilizado para colorear los nodos del árbol.

Resultado:

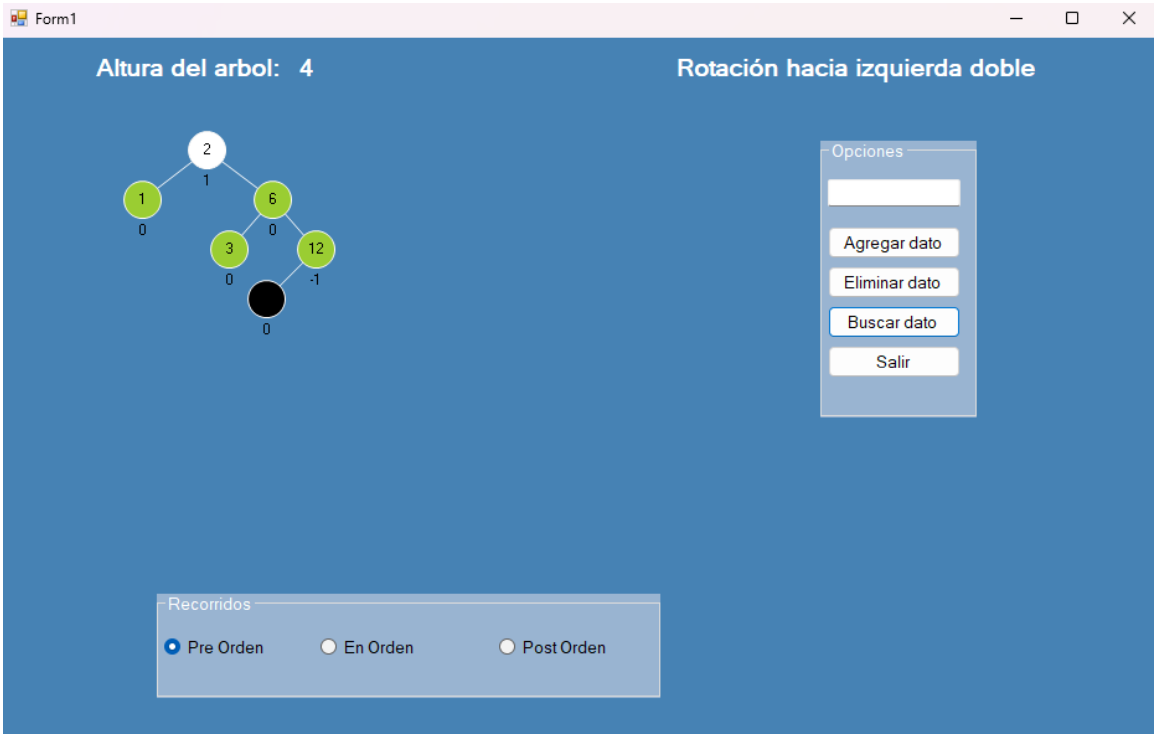
Agregamos valores en pre Orden



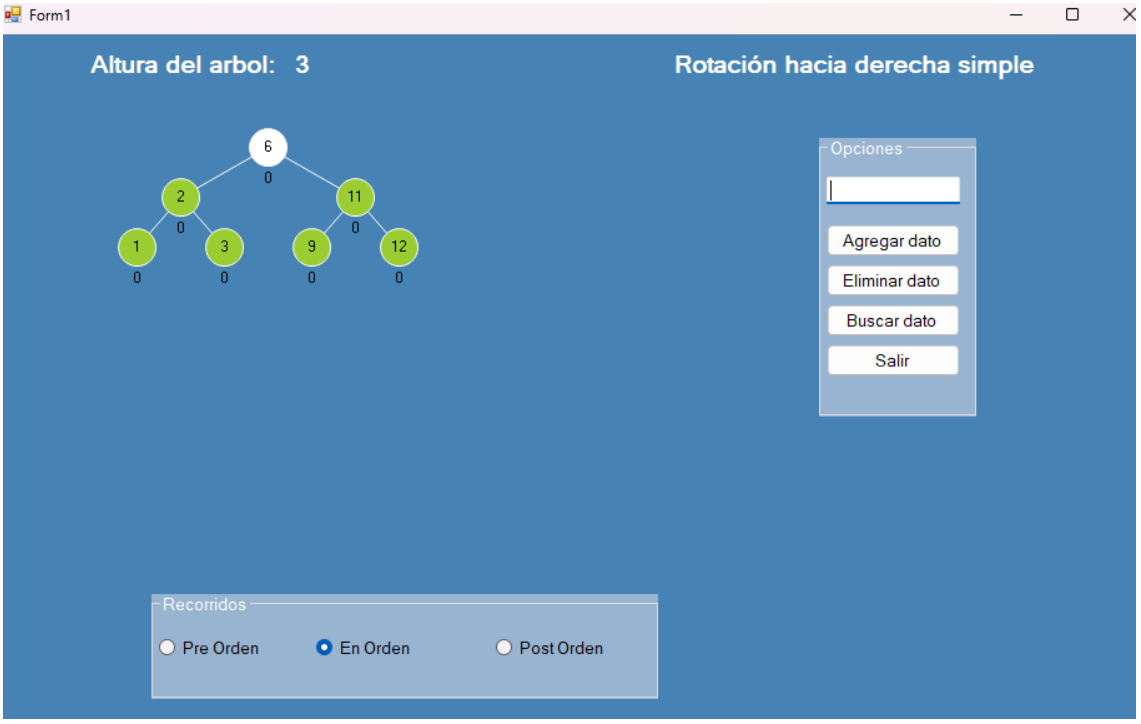
Al buscar dato inexistente se obtiene



Al buscar uno existente vemos que se colorea de negro



Al agregar nuevo dato y poner En Orden se muestra así:



Luego en Post Orden