

PED941
Ciclo 01-2024
G01T



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.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Drawing;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7  using System.Windows.Forms;
8
9  namespace Arbo_AVL
10 {
11     40 referencias
12     class AVL
13     {
14         public int valor;
15         public String tipoRotacion;
16         public AVL NodoIzquierdo;
17         public AVL NodoDerecho;
18         public AVL NodoPadre;
19         public int altura;
20         public Rectangle prueba;
21         private DibujaAVL arbol;
22
23         1 referencia
24         public AVL()
25         {
26         }
27
28         0 referencias
29         public DibujaAVL Arbol
30         {
31             get { return arbol; }
32             set { arbol = value; }
33         }
34
35         //Constructor
36         3 referencias
37         public AVL(int valorNuevo, AVL izquierdo, AVL derecho, AVL padre)
38         {
39             valor = valorNuevo;
40             NodoIzquierdo = izquierdo;
41             NodoDerecho = derecho;
42             NodoPadre = padre;
43             altura = 0;
44             tipoRotacion = ""; //Inicializa la variable del tipo de rotacion.
45         }
46
47         //Funcion para insertar un valor nuevo en el arbol AVL
48         3 referencias
49         public AVL Insertar(int valorNuevo, AVL raiz)
50         {
51             if (raiz == null)
52             {
53                 raiz = new AVL(valorNuevo, null, null, null);
54             }
55             else if (valorNuevo < raiz.valor)
56             {
57                 raiz.NodoIzquierdo = Insertar(valorNuevo, raiz.NodoIzquierdo);
58             }
59             else if (valorNuevo > raiz.valor)
60             {
61                 raiz.NodoDerecho = Insertar(valorNuevo, raiz.NodoDerecho);
62             }
63             else
64             {
65                 MessageBox.Show("Valor Existente en el Arbol", "Error", MessageBoxButtons.OK);
66             }
67
68             //Realiza las rotaciones simples o dobles segun el caso
69             if (Alturas(raiz.NodoIzquierdo) - Alturas(raiz.NodoDerecho) == 2)
70             {
71                 if (valorNuevo < raiz.NodoIzquierdo.valor)
72                 {
73                     raiz = RotacionIzquierdaSimple(raiz);
74                     raiz.tipoRotacion = "Rotación hacia izquierda simple";
75                 }
76                 else
77                 {
78                     raiz = RotacionIzquierdaDoble(raiz);
79                     raiz.tipoRotacion = "Rotación hacia izquierda doble";
80                 }
81             }
82             if (Alturas(raiz.NodoDerecho) - Alturas(raiz.NodoIzquierdo) == 2)
83             {
84                 if (valorNuevo > raiz.NodoDerecho.valor)
85                 {
86                     raiz = RotacionDerechaSimple(raiz);
87                     raiz.tipoRotacion = "Rotación hacia derecha simple";
88                 }
89                 else
90                 {
91                     raiz = RotacionDerechaDoble(raiz);
92                     raiz.tipoRotacion = "Rotación hacia derecha doble";
93                 }
94             }
95             raiz.altura = max(Alturas(raiz.NodoIzquierdo), Alturas(raiz.NodoDerecho) + 1);
96             return raiz;
97         }
98     }
99 }
```

```

87 //Funcion para obtener que rama es mayor
88 //5 referencias
89 private static int max(int lhs, int rhs)
90 {
91     return lhs > rhs ? lhs : rhs;
92 }
93 //Obtiene la altura
94 //22 referencias
95 private static int Alturas(AVL raiz)
96 {
97     return raiz == null ? -1 : raiz.altura;
98 }
99
100 AVL nodoE, nodoP;
101 //Funcion eliminar
102 //3 referencias
103 public AVL Eliminar(int valorEliminar, ref AVL raiz)
104 {
105     if(raiz != null)
106     {
107         if(valorEliminar < raiz.valor)
108         {
109             nodoE = raiz;
110             Eliminar(valorEliminar, ref raiz.NodoIzquierdo);
111         }
112         else
113         {
114             if(valorEliminar > raiz.valor)
115             {
116                 nodoE = raiz;
117                 Eliminar(valorEliminar, ref raiz.NodoDerecho);
118             }
119             else
120             {
121                 //Posicionado sobre el elemento a eliminar
122                 AVL NodoEliminar = raiz;
123                 if(NodoEliminar.NodoDerecho == null)
124                 {
125                     raiz = NodoEliminar.NodoIzquierdo;
126                     if(Alturas(nodoE.NodoIzquierdo) - Alturas(nodoE.NodoDerecho) == 2)
127                     {
128                         //MessageBox.Show("nodoE" + nodoE.valor.ToString());
129                         if (valorEliminar < nodoE.valor)
130                             nodoP = RotacionIzquierdaSimple(nodoE);
131                         else
132                             nodoE = RotacionDerechaSimple(nodoE);
133                     }
134                 }
135                 else
136                 {
137                     if(Alturas(nodoE.NodoDerecho) - Alturas(nodoE.NodoIzquierdo) == 2)
138                     {
139                         if (valorEliminar > nodoE.valor)
140                             nodoE = RotacionDerechaSimple(nodoE);
141                         else
142                             nodoE = RotacionDerechaDoble(nodoE);
143                         nodoP = RotacionDerechaSimple(nodoE);
144                     }
145                 }
146             }
147         }
148     }
149     else
150     {
151         if (NodoEliminar.NodoIzquierdo == null)
152             raiz = NodoEliminar.NodoDerecho;
153         else
154         {
155             if(Alturas(raiz.NodoIzquierdo) - Alturas(raiz.NodoDerecho) > 0)
156             {
157                 AVL AuxiliarNodo = null;
158                 AVL Auxiliar = raiz.NodoIzquierdo;
159                 bool Bandera = false;
160                 while(Auxiliar.NodoDerecho != null)
161                 {
162                     AuxiliarNodo = Auxiliar;
163                     Auxiliar = Auxiliar.NodoDerecho;
164                     Bandera = true;
165                 }
166                 raiz.valor = AuxiliarNodo.valor;
167                 NodoEliminar = AuxiliarNodo;
168                 if (Bandera == true)
169                     AuxiliarNodo.NodoDerecho = Auxiliar.NodoIzquierdo;
170                 else
171                     raiz.NodoIzquierdo = Auxiliar.NodoIzquierdo;
172                 //Realiza las rotaciones simples o dobles dependiendo del caso
173             }
174             else
175             {
176                 if (Alturas(raiz.NodoDerecho) - Alturas(raiz.NodoIzquierdo) > 0)
177                 {
178                     AVL AuxiliarNodo = null;
179                     AVL Auxiliar = raiz.NodoIzquierdo;
180                     bool Bandera = false;
181                     while(Auxiliar.NodoDerecho != null)
182                     {
183                         AuxiliarNodo = Auxiliar;
184                     }
185                 }
186             }
187         }
188     }
189 }

```

```

131 }
132 if(Alturas(nodoE.NodoDerecho) - Alturas(nodoE.NodoIzquierdo) == 2)
133 {
134     if (valorEliminar > nodoE.valor)
135         nodoE = RotacionDerechaSimple(nodoE);
136     else
137         nodoE = RotacionDerechaDoble(nodoE);
138     nodoP = RotacionDerechaSimple(nodoE);
139 }
140 }
141 else
142 {
143     if (NodoEliminar.NodoIzquierdo == null)
144         raiz = NodoEliminar.NodoDerecho;
145     else
146     {
147         if(Alturas(raiz.NodoIzquierdo) - Alturas(raiz.NodoDerecho) > 0)
148         {
149             AVL AuxiliarNodo = null;
150             AVL Auxiliar = raiz.NodoIzquierdo;
151             bool Bandera = false;
152             while(Auxiliar.NodoDerecho != null)
153             {
154                 AuxiliarNodo = Auxiliar;
155                 Auxiliar = Auxiliar.NodoDerecho;
156                 Bandera = true;
157             }
158             raiz.valor = AuxiliarNodo.valor;
159             NodoEliminar = AuxiliarNodo;
160             if (Bandera == true)
161                 AuxiliarNodo.NodoDerecho = Auxiliar.NodoIzquierdo;
162             else
163                 raiz.NodoIzquierdo = Auxiliar.NodoIzquierdo;
164             //Realiza las rotaciones simples o dobles dependiendo del caso
165         }
166         else
167         {
168             if (Alturas(raiz.NodoDerecho) - Alturas(raiz.NodoIzquierdo) > 0)
169             {
170                 AVL AuxiliarNodo = null;
171                 AVL Auxiliar = raiz.NodoIzquierdo;
172                 bool Bandera = false;
173                 while(Auxiliar.NodoDerecho != null)
174                 {
175                     AuxiliarNodo = Auxiliar;
176                 }
177             }
178         }
179     }
180 }

```

```

177         Auxiliar = Auxiliar.NodoDerecho;
178         Bandera = true;
179     }
180     raiz.valor = Auxiliar.valor;
181     NodoEliminar = Auxiliar;
182     if (Bandera == true)
183         Auxiliar.NodoDerecho = Auxiliar.NodoIzquierdo;
184     else
185         raiz.NodoDerecho = Auxiliar.NodoIzquierdo;
186     }
187 }
188 }
189 }
190 }
191 }
192 }
193 else
194     MessageBox.Show("No existe Nodo en el arbol", "Error", MessageBoxButtons.OK);
195     return nodoP;
196 }
197
198 //Funciones de las rotaciones
199
200 //Rotacion izquierda simple.
201 4 referencias
202 private static AVL RotacionIzquierdaSimple(AVL k2)
203 {
204     AVL k1 = k2.NodoIzquierdo;
205     k2.NodoIzquierdo = k1.NodoDerecho;
206     k1.NodoDerecho = k2;
207     k2.altura = max(Alturas(k2.NodoIzquierdo), Alturas(k2.NodoDerecho)) + 1;
208     k1.altura = max(Alturas(k1.NodoIzquierdo), k2.altura) + 1;
209     return k1;
210 }
211
212 //Rotacion derecha simple.
213 6 referencias
214 private static AVL RotacionDerechaSimple(AVL k1)
215 {
216     AVL k2 = k1.NodoDerecho;
217     k1.NodoDerecho = k2.NodoIzquierdo;
218     k2.NodoIzquierdo = k1;
219     k1.altura = max(Alturas(k1.NodoIzquierdo), Alturas(k1.NodoDerecho)) + 1;
220     k2.altura = max(Alturas(k2.NodoDerecho), k1.altura) + 1;
221     return k2;
222 }

```

```

223 //Doble rotacion izquierda.
224 1 referencia
225 private static AVL RotacionIzquierdaDoble(AVL k3)
226 {
227     k3.NodoIzquierdo = RotacionDerechaSimple(k3.NodoIzquierdo);
228     return RotacionIzquierdaSimple(k3);
229 }
230
231 //Doble Rotacion Derecha.
232 2 referencias
233 private static AVL RotacionDerechaDoble(AVL k1)
234 {
235     k1.NodoDerecho = RotacionIzquierdaSimple(k1.NodoDerecho);
236     return RotacionDerechaSimple(k1);
237 }
238
239 //Obtiene altura del arbol.
240 5 referencias
241 public int getAltura(AVL nodoActual)
242 {
243     if (nodoActual == null)
244         return 0;
245     else
246         return 1 + Math.Max(getAltura(nodoActual.NodoIzquierdo), getAltura(nodoActual.NodoDerecho));
247 }
248
249 //Buscar un valor en el arbol.
250 3 referencias
251 public void buscar(int valorBuscar, AVL raiz)
252 {
253     if (raiz != null)
254     {
255         if (valorBuscar < raiz.valor)
256             buscar(valorBuscar, raiz.NodoIzquierdo);
257         else
258         {
259             if (valorBuscar > raiz.valor)
260                 buscar(valorBuscar, raiz.NodoDerecho);
261         }
262     }
263     else
264         MessageBox.Show("No se encontro el valor", "Error", MessageBoxButtons.OK);
265 }
266
267 //Funciones para dibujar el arbol.
268 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 posición donde debe crearse el nodo.
272 3 referencias
273 public void PosicionNodo(ref int xmin, int ymin)
274 {
275     int aux1, aux2;
276     CoordenadaY = (int)(ymin + Radio / 2);
277
278     //Obtiene la posición del sub-arbol izquierdo.
279     if (NodoIzquierdo != null)
280         NodoIzquierdo.PosicionNodo(ref xmin, ymin + Radio + DistanciaV);
281     if ((NodoIzquierdo != null) && (NodoDerecho != null))
282         xmin += DistanciaH;
283
284     //Si existen ambos nodos dejara espacio entre ellos
285     if (NodoDerecho != null)
286         NodoDerecho.PosicionNodo(ref xmin, ymin + Radio + DistanciaV);
287
288     // Posición de ambos nodos
289     if (NodoIzquierdo != null)
290     {
291         if (NodoDerecho != null)
292             //Centro entre los nodos
293             CoordenadaX = (int)((NodoIzquierdo.CoordenadaX + NodoDerecho.CoordenadaX) / 2);
294         else
295         {
296             //No hay nodo derecho, centrar al izquierdo
297             aux1 = NodoIzquierdo.CoordenadaX;
298             NodoIzquierdo.CoordenadaX = CoordenadaX - 40;
299             CoordenadaX = aux1;
300         }
301     }
302     else if (NodoDerecho != null)
303     {
304         aux2 = NodoDerecho.CoordenadaX;
305         //No hay nodo izquierdo, centrar al derecho
306         NodoDerecho.CoordenadaX = CoordenadaX - 40;
307         CoordenadaX = aux2;
308     }
309     else
310     {

```

```

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

```

```

353     formato.Alignment = StringAlignment.Center;
354     formato.LineAlignment = StringAlignment.Center;
355     grafo.DrawString(valor.ToString(), fuente, Brushes.Black, CoordenadaX, CoordenadaY, formato);
356     //Obtiene el factor balanceo restando las alturas de cada nodo y las muestra
357     grafo.DrawString((Alturas(NodoDerecho)-Alturas(NodoIzquierdo)).ToString(), fuente, Brushes.Black, CoordenadaX, CoordenadaY+25, formato);
358
359     //Dibuja los nodos hijos derecho e izquierdo
360     if (NodoIzquierdo != null)
361         NodoIzquierdo.DibujarNodo(grafo, fuente, Brushes.YellowGreen, rellenoFuente, lapiz, dato, encuentro);
362     if (NodoDerecho != null)
363         NodoDerecho.DibujarNodo(grafo, fuente, Brushes.YellowGreen, rellenoFuente, lapiz, dato, encuentro);
364 }
365
366 //referencias
367 public void colorear(Graphics grafo, Font fuente, Brush relleno, Brush rellenoFuente, Pen lapiz)
368 {
369     //Dibuja el contorno del nodo.
370     Rectangle rect = new Rectangle(
371         (int)(CoordenadaX - Radio / 2),
372         (int)(CoordenadaY - Radio / 2), Radio, Radio);
373     prueba = new Rectangle((int)(CoordenadaX - Radio / 2), (int)(CoordenadaY - Radio / 2), Radio, Radio);
374
375     //Dibuja el valor del nombre.
376     StringFormat formato = new StringFormat();
377     formato.Alignment = StringAlignment.Center;
378     formato.LineAlignment = StringAlignment.Center;
379     grafo.DrawString(valor.ToString(), fuente, Brushes.Black, CoordenadaX, CoordenadaY, formato);
380 }
381
382 }
383
384

```

Análisis

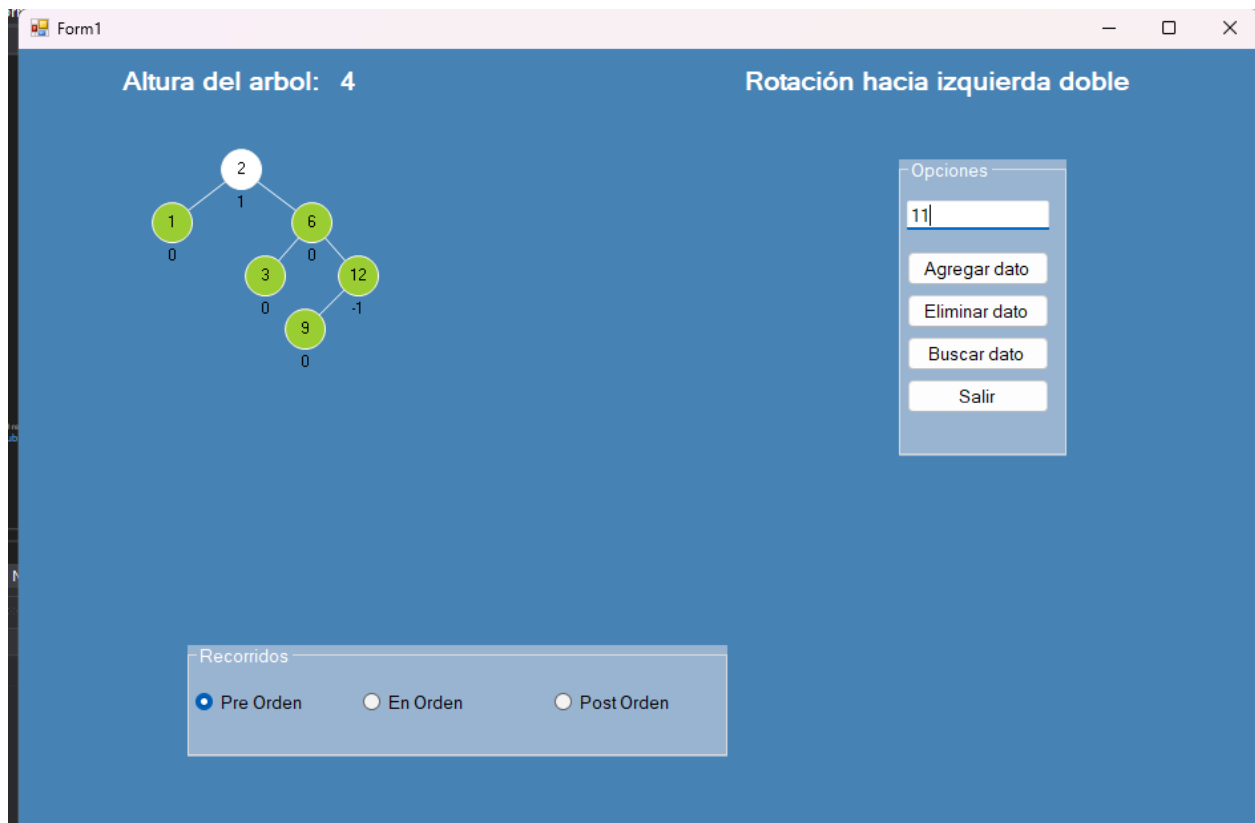
Este código es una implementación de un Árbol AVL en C#

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

Form1

Altura del arbol: 4

Rotación hacia izquierda doble

```
graph TD; 2((2)) --- 1((1)); 2 --- 6((6)); 6 --- 3((3)); 6 --- 12((12)); 12 --- 9((9));
```

Opciones

11

Agregar dato

Eliminar dato

Buscar dato

Salir

Error

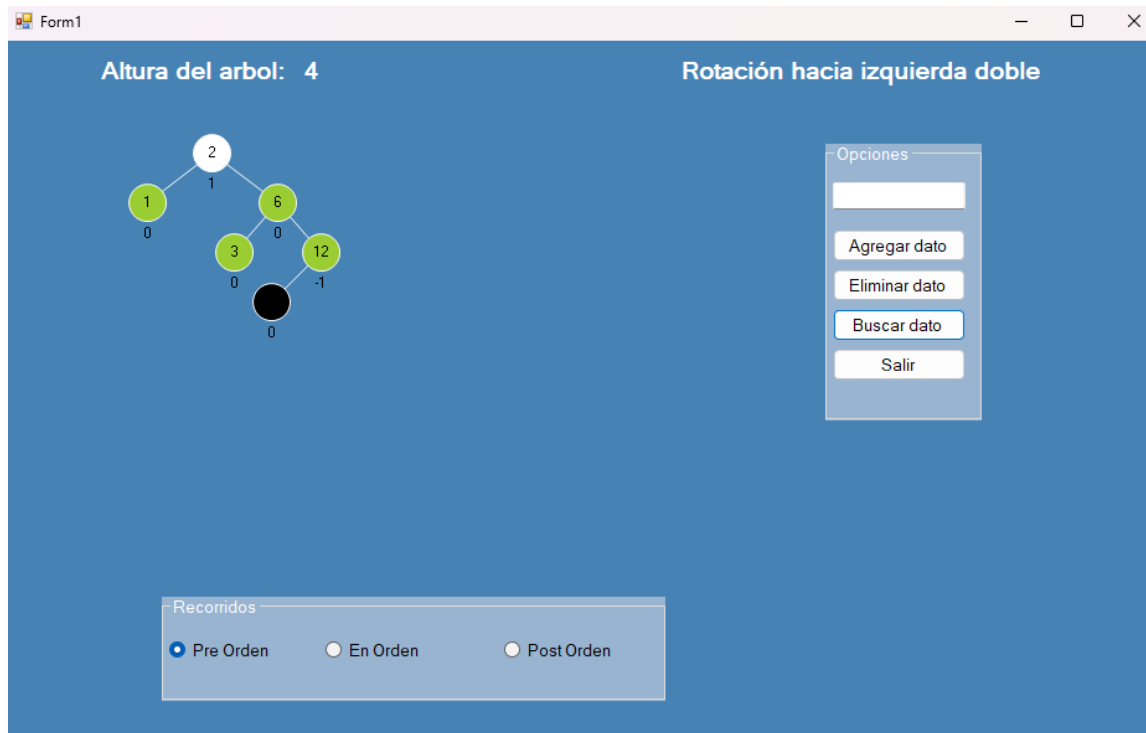
No se encontro el valor

OK

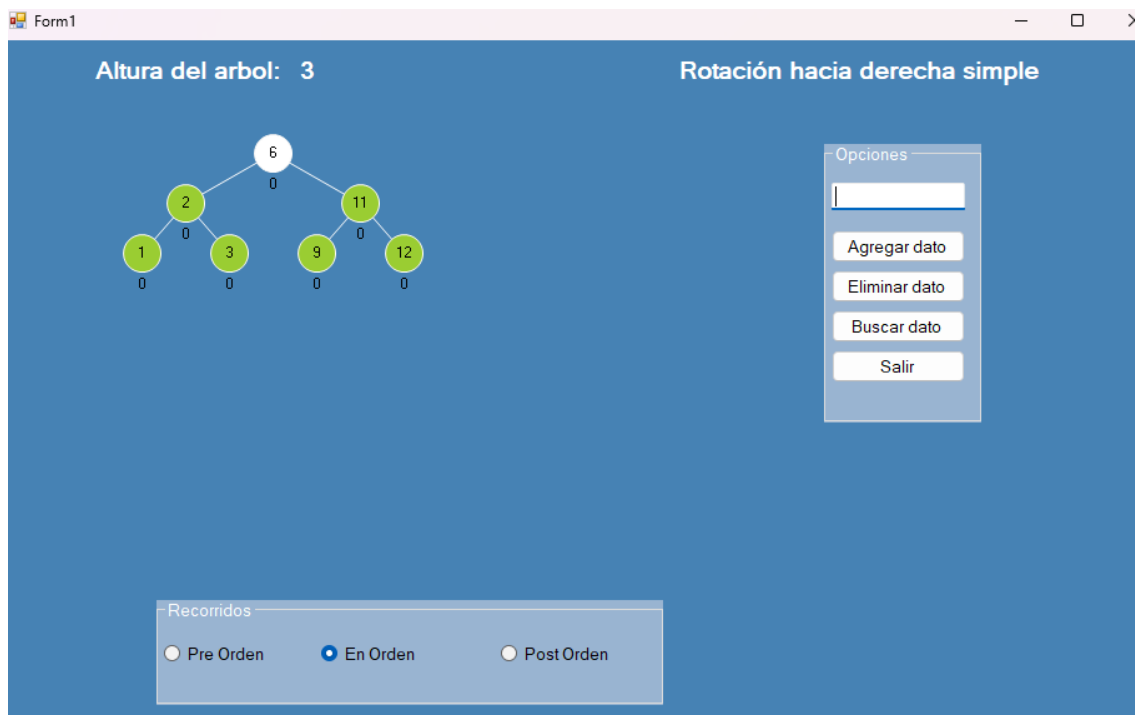
Recorridos

☒ Pre Orden ☐ En Orden ☐ Post Orden

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

Form1

Altura del arbol: 4

Rotación hacia derecha simple

```
graph TD; 6((6)) ---|1| 2((2)); 6 ---|1| 11((11)); 2 ---|0| 1((1)); 2 ---|0| 3((3)); 11 ---|1| 9((9)); 11 ---|1| 12((12)); 9 ---|0| 13((13));
```

Opciones

Agregar dato

Eliminar dato

Buscar dato

Salir

Recorridos

☐ Pre Orden ☐ En Orden ☒ Post Orden