GUÍA #3.
ÁRBOLES BINARIOS DE BÚSQUEDA Y ÁRBOLES AVL.

Ciclo 01-2022

PROGRAMACIÓN CON ESTRUCTURAS DE DATOS

Facultad:	Ingeniería
Escuela:	Computación
Asignatura:	Programación con Estructuras de Datos

Tema: Árboles Binarios de Búsqueda (ABB).

Competencia

- Desarrolla sistemas de información informáticos mediante la integración de principios matemáticos, ciencia computacional y prácticas de ingeniería, considerando estándares de calidad y mejores prácticas validadas por la industria del software.

Materiales y Equipo

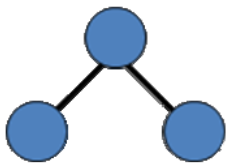
- Guía Número 3
- Computadora con programa Microsoft Visual C#.

PARTE I

Introducción Teórica

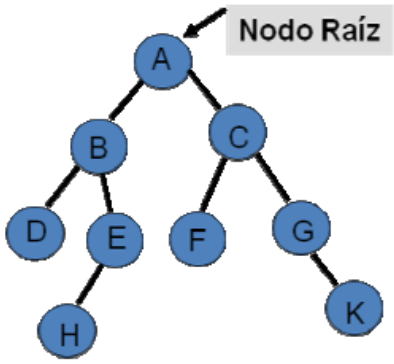
ESTRUCTURA DINÁMICA ÁRBOL BINARIO

En ciencias de la computación, un árbol binario es una estructura de datos en la cual cada nodo siempre tiene cero hijos (0), un hijo (1) o un hijo izquierdo y un hijo derecho (2). No pueden tener más de dos hijos (de ahí el nombre 'Binario'). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno

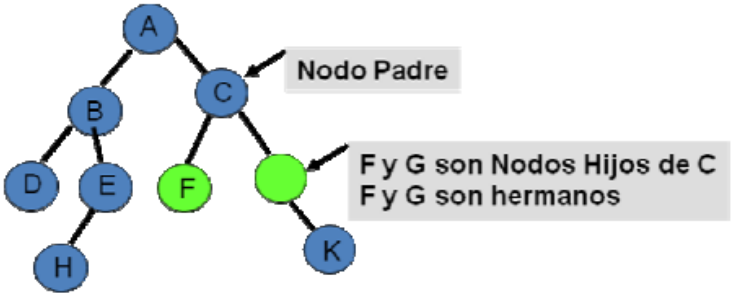


TERMINOLOGÍA:

Nodo: Cada elemento de un árbol
Nodo Raíz: Primer elemento agregado al árbol

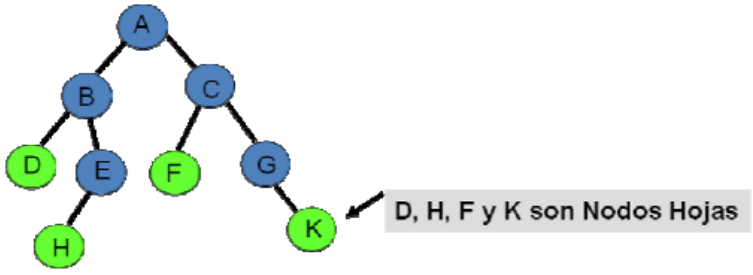


Nodo Padre: Se le llama así al nodo predecesor de un elemento.
Nodo Hijo: Es el nodo sucesor de un elemento.
Nodo Hermano: Nodos que tienen el mismo nodo padre



Subárbol:

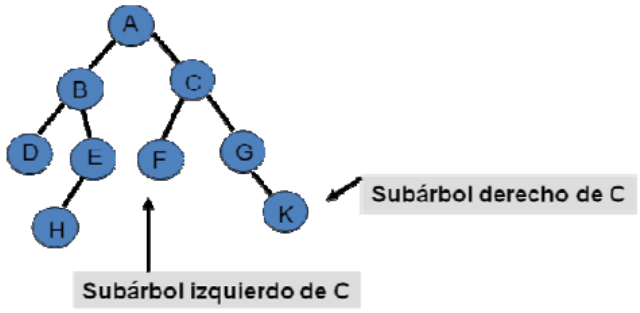
1. Si el nodo tiene 0 relaciones recibe el nombre de hoja.



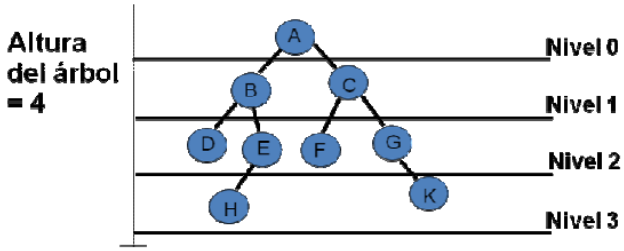
2. Si el nodo raíz tiene 1 relación a la izquierda, el segundo elemento de la relación es el subárbol izquierdo.

3. Si el nodo raíz tiene 1 relación a la derecha, el segundo elemento de la relación es el subárbol derecho.

Altura y Niveles:



La altura corresponde a la cantidad de niveles que existen, los niveles se inician desde 0.



Los árboles binarios poseen una especialización denominada **árboles binarios de búsqueda o ABB**, estos árboles se distinguen por tener un método particular para ingresar datos, respetan las siguientes reglas:

1. El primer elemento que se ingresa al árbol se convierte en la raíz
2. Todos los elementos que se ingresan posteriormente serán comparados con la raíz del árbol y con las raíces de los subárboles consecuentes.
    a. Todo valor mayor a la raíz será enviado al subárbol derecho para otra comparación o para su inserción inmediata.
    b. Todo valor menor a la raíz será enviado al subárbol izquierdo para otra comparación o para su inserción inmediata.

Procedimiento

Ejemplo 1. Implementación de un árbol binario de búsqueda en C#

1. Crear un proyecto de tipo Windows Form Application, se sugiere que le dé el nombre de **“Arbol Binario”.**
2. Agregar una **clase** al proyecto, para facilitar el entendimiento nómbrelo como **“Nodo Arbol”.** Esta clase la utilizaremos para definir el elemento “nodo” del árbol binario. (Para agregar una clase al proyecto, dar click en el Explorador de soluciones, agregar y dar agregar clase y nombrarla con la sugerencia dada).
3. Dentro de la clase creada, agregue el siguiente código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;          //librería para dibujar figuras geométricas
using System.Linq;
using System.Text;
using System.Threading;        //librería para manejo de hilos
using System.Windows.Forms;

namespace Arbol_Binario
{
    class Nodo_Arbol
    {
        public int info;                      //dato a almacenar en el nodo
        public Nodo_Arbol Izquierdo;          //nodo izquierdo del árbol
        public Nodo_Arbol Derecho;            //nodo derecho del árbol
        public Nodo_Arbol Padre;              //nodo raíz del árbol
        public int altura;
        public int nivel;
        public Rectangle nodo;            //para dibujar el nodo del árbol

//Variable que definen el tamaño de los círculos que representan los nodos del árbol
        private const int Radio = 30;
//Variable para el manejo de distancia horizontal
        private const int DistanciaH = 80;
//variable para el manejo de distancia vertical
        private const int DistanciaV = 10;
//variable para manejar posición eje X
        private int CoordenadaX;
//variable para manejar posición eje Y
        private int CoordenadaY;
        Graphics col;
        private Arbol_Binario arbol; //declarando un objeto de tipo árbol

        public Nodo_Arbol()           //constructor por defecto
        {
        }

//constructor por defecto para el objeto de tipo árbol
        public Arbol_Binario Arbol
{
            get
            { return arbol;}

            set
            { arbol = value;}


 }

//constructor con parámetros
public Nodo_Arbol (int nueva_info, Nodo_Arbol izquierdo, Nodo_Arbol  derecho, Nodo_Arbol padre)
        {
            info = nueva_info;
            Izquierdo = izquierdo;
            Derecho = derecho;
            Padre = padre;
            altura = 0;
        }

       //función para insertar un nodo en el árbol
        public Nodo_Arbol Insertar(int x, Nodo_Arbol t, int Level)
        {
```

```
            if (t == null)
            {
                t = new Nodo_Arbol(x, null, null, null);
                t.nivel = Level;
            }

            else if (x < t.info) //si el valor a insertar es menor que la raíz
            {
                Level++;
                t.Izquierdo = Insertar(x, t.Izquierdo, Level);
            }

            else if (x > t.info) //si el valor a insertar es mayor que la raíz
            {
                Level++;
                t.Derecho = Insertar(x, t.Derecho, Level);
            }
            else
            {
                MessageBox.Show("Dato existente en el Arbol", "Error de Ingreso");
            }
            return t;
        }

//Función para eliminar un nodo de un árbol binario
   public void Eliminar(int x, ref Nodo_Arbol t)
   {
        if (t != null)    //si la raíz es distinta de null
          {
            if (x < t.info) //si el valor a eliminar es menor que la raíz
             {
               Eliminar(x, ref t.Izquierdo);
              }
              else
              {
                  if (x > t.info) //si el valor a eliminar es mayor que la raíz
                  {
                      Eliminar(x, ref t.Derecho);
                  }

                  else
                  {
                      Nodo_Arbol NodoEliminar = t;   //se ubica el nodo a eliminar
                      //se verifica si tiene hijo derecho

                      if (NodoEliminar.Derecho == null){
                          t = NodoEliminar.Izquierdo;
                      }
                      else
                      {
                          //se verifica si tiene hijo izq
                          if (NodoEliminar.Izquierdo == null){
                              t = NodoEliminar.Derecho;
                          }

                          else
                          {
                              if (Alturas(t.Izquierdo) - Alturas(t.Derecho) > 0)
                      //Para verificar que hijo pasa a ser nueva raíz del subárbol
                              {
                                  Nodo_Arbol AuxiliarNodo = null;
                                  Nodo_Arbol Auxiliar = t.Izquierdo;
                                  bool bandera = false;
```

```
                while (Auxiliar.Derecho != null)
                {
                    AuxiliarNodo = Auxiliar;
                    Auxiliar = Auxiliar.Derecho;
                    bandera = true;
                }
                 // se crea nodo temporal
                     t.info = Auxiliar.info;
                     NodoEliminar = Auxiliar;

                if (bandera == true)
                {
                    AuxiliarNodo.Derecho = Auxiliar.Izquierdo;

                }

                else
                {
                    t.Izquierdo = Auxiliar.Izquierdo;
                }
            }

            else
            {
              if (Alturas(t.Derecho) - Alturas(t.Izquierdo) > 0)
                {
                    Nodo_Arbol AuxiliarNodo = null;
                    Nodo_Arbol Auxiliar = t.Derecho;
                    bool bandera = false;

                    while (Auxiliar.Izquierdo != null)
                    {
                        AuxiliarNodo = Auxiliar;
                        Auxiliar = Auxiliar.Izquierdo;
                        bandera = true;
                    }

                    t.info = Auxiliar.info;
                    NodoEliminar = Auxiliar;

                    if (bandera == true)
                    {
                     AuxiliarNodo.Izquierdo = Auxiliar.Derecho;
                    }

                    else
                    {
                        t.Derecho = Auxiliar.Derecho;
                    }
                }

            else
             {
              if (Alturas(t.Derecho) - Alturas(t.Izquierdo) == 0)
                    {
                        Nodo_Arbol AuxiliarNodo = null;
                        Nodo_Arbol Auxiliar = t.Izquierdo;
                        bool bandera = false;

                        while (Auxiliar.Derecho != null)
                        {
                            AuxiliarNodo = Auxiliar;
                            Auxiliar = Auxiliar.Derecho;
                            bandera = true;
```

```
                                        }

                                        t.info = Auxiliar.info;
                                        NodoEliminar = Auxiliar;

                                        if (bandera == true)
                                        {
                                    AuxiliarNodo.Derecho = Auxiliar.Izquierdo;

                                        }

                                        else
                                        {
                                            t.Izquierdo = Auxiliar.Izquierdo;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }

                else
                {
                 MessageBox.Show("Nodo NO existente el Arbol", "Error de eliminación");
                }
        }  //Final de la función eliminar

   //Función buscar un nodo
public void buscar(int x, Nodo_Arbol t)
        {
            if (t != null)
            {
                if (x == t.info)
                {
MessageBox.Show("Nodo encontrado en la posición X: "+ t.CoordenadaX +" Y:"+t.CoordenadaY);
                    encontrado(t);
                }
                else
                {
                    if (x < t.info)  //búsqueda en el subárbol izquierdo
                    {
                        buscar(x, t.Izquierdo);
                    }

                    else
                    {
                        if (x > t.info)  //búsqueda en el subárbol derecho
                        {
                            buscar(x, t.Derecho);
                        }
                    }
                }
            }

            else
            {
                MessageBox.Show("Nodo NO encontrado", "Error de búsqueda");
            }
        }
```

4. A continuación se agregan las funciones que permiten dibujar el Árbol Binario en el formulario. Siempre en la misma clase **"Nodo Arbol"**, agregue el siguiente código:

```
//Función posición nodo (donde se ha creado dibujo del nodo)
        public void PosicionNodo(ref int xmin, int ymin)
        {   int aux1, aux2;
            CoordenadaY = (int)(ymin + Radio / 2);

             //obtiene la posición del sub-árbol izquierdo
            if (Izquierdo != null)
            {
                Izquierdo.PosicionNodo(ref xmin, ymin + Radio + DistanciaV);
            }

            if ((Izquierdo != null) && (Derecho != null))
            {
                xmin += DistanciaH;
            }
        //si existe nodo derecho y el nodo izquierdo deja un espacio entre ellos
            if (Derecho != null)
            {
                Derecho.PosicionNodo(ref xmin, ymin + Radio + DistanciaV);
            }

            if (Izquierdo != null && Derecho != null)
          CoordenadaX = (int) ((Izquierdo.CoordenadaX + Derecho.CoordenadaX) / 2);
            else
                if (Izquierdo != null)
            {   aux1 = Izquierdo.CoordenadaX;
                Izquierdo.CoordenadaX = CoordenadaX - 80;
                CoordenadaX = aux1;
            }
            else
                    if (Derecho != null)
            {   aux2 = Derecho.CoordenadaX;
                  //no hay nodo izquierdo, centrar en nodo derecho
                Derecho.CoordenadaX = CoordenadaX + 80;
                CoordenadaX = aux2;
            }
                    else
                    {   CoordenadaX = (int) (xmin + Radio / 2);xmin += Radio;
                    }
        }

       //Función para dibujar las ramas de los nodos izquierdo y derecho
        public void DibujarRamas(Graphics grafo,Pen Lapiz)
        {
            if
             (Izquierdo != null)
            // Dibujará rama izquierda
            {
                grafo.DrawLine(Lapiz, CoordenadaX, CoordenadaY,
Izquierdo.CoordenadaX,Izquierdo.CoordenadaY);
                Izquierdo.DibujarRamas(grafo, Lapiz);
```

```
            }
            if
             (Derecho !=null)
            // Dibujará rama derecha
            {
                grafo.DrawLine(Lapiz, CoordenadaX, CoordenadaY,
Derecho.CoordenadaX,Derecho.CoordenadaY);
                Derecho.DibujarRamas(grafo, Lapiz);
            }
        }

       //Función para dibujar el nodo en la posición especificada
        public void DibujarNodo(Graphics grafo,Font fuente,Brush Relleno,Brush
RellenoFuente,Pen Lapiz,Brush encuentro)
        {
            col = grafo;
            // Dibuja el contorno del nodo
            Rectangle rect = new Rectangle ((int)(CoordenadaX - Radio / 2), (
int)(CoordenadaY - Radio / 2), Radio, Radio);
            Rectangle prueba = new Rectangle((int)(CoordenadaX - Radio / 2),
(int)(CoordenadaY - Radio/ 2), Radio, Radio);
            grafo.FillEllipse(encuentro, rect);
            grafo.FillEllipse(Relleno, rect);
            grafo.DrawEllipse(Lapiz, rect);
            // Para dibujar el nombre del nodo, es decir el contenido
            StringFormat formato = new StringFormat();
            formato.Alignment =StringAlignment.Center;
            formato.LineAlignment =StringAlignment.Center;
            grafo.DrawString(info.ToString(), fuente, RellenoFuente, CoordenadaX,CoordenadaY,
formato);
            //Dibuja los nodos hijos derecho e izquierdo.
            if(Izquierdo != null)
            {
                Izquierdo.DibujarNodo(grafo, fuente, Relleno, RellenoFuente, Lapiz,
encuentro);
            }
            if(Derecho !=null)
            {
                Derecho.DibujarNodo(grafo, fuente, Relleno, RellenoFuente, Lapiz, encuentro);
            }
        }

        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);
            Rectangle prueba =new Rectangle((int)(CoordenadaX - Radio / 2), (int)(CoordenadaY
- Radio/ 2), Radio, Radio);
            grafo.FillEllipse(Relleno, rect);
            grafo.DrawEllipse(Lapiz, rect);
            //Dibuja el nombre
            StringFormat formato = new StringFormat();
            formato.Alignment =StringAlignment.Center;
            formato.LineAlignment = StringAlignment.Center;
```

```
            grafo.DrawString(info.ToString(), fuente, RellenoFuente, CoordenadaX,CoordenadaY,
formato);
        }
        //Verificar altura del árbol
          private static int Alturas(Nodo_Arbol t)
        {
            return t == null ? -1 : t.altura;
        }

        public void encontrado(Nodo_Arbol t)
        {
            Rectangle rec = new Rectangle(t.CoordenadaX, t.CoordenadaY, 40, 40)
        }
      }//Clase
    }// Fin Namespace
```

5. Agregar una clase al proyecto, se sugiere darle el nombre de ***"Arbol_Binario".*** Esta clase se utiliza para definir la estructura "Arbol". Agregue el código

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

namespace Arbol_Binario
{
    class Arbol_Binario
    {
        public Nodo_Arbol Raiz;
        public Nodo_Arbol aux;

        public Arbol_Binario()
        {
            aux =new Nodo_Arbol();
        }

        public Arbol_Binario(Nodo_Arbol nueva_raiz)
        {
            Raiz = nueva_raiz;
        }
        // Función para agregar un nuevo nodo (valor) al Árbol Binario.
        public void Insertar(int x)
        {
            if (Raiz == null)
            {
                Raiz = new Nodo_Arbol(x, null, null, null);
                Raiz.nivel = 0;
            }
            else
                Raiz = Raiz.Insertar(x, Raiz, Raiz.nivel);
        }
        // Función para eliminar un nodo (valor) del Árbol Binario.
        public void Eliminar(int x)
        {
            if (Raiz ==  null)
                Raiz =new Nodo_Arbol(x,null,null,null);
            else
```

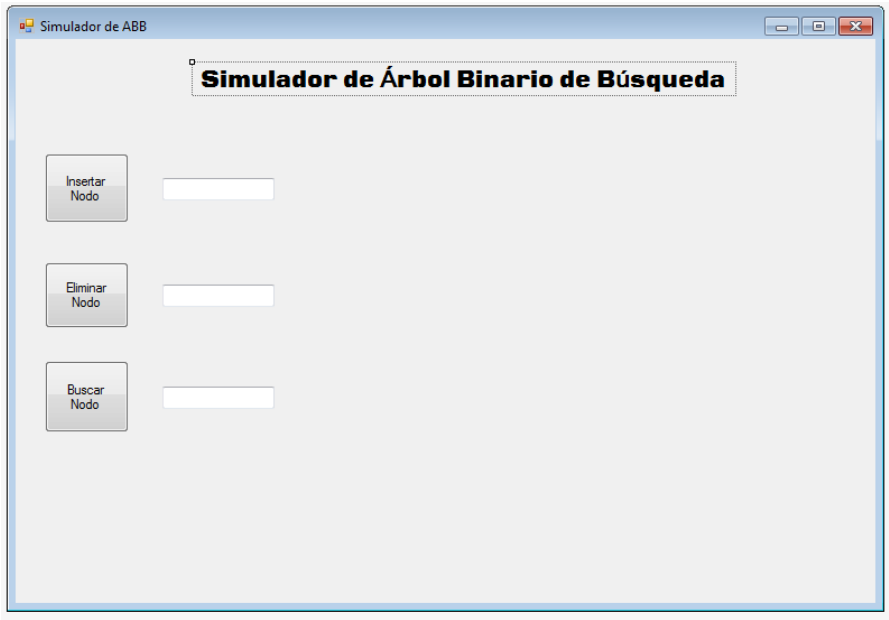
```
                Raiz.Eliminar(x,ref Raiz);
        }

        public void Buscar(int x)
        {
            if (Raiz != null)
            {
                Raiz.buscar(x, Raiz);
            }

        }
```

6. A continuación agregar funciones que servirán para dibujar el Árbol Binario en el formulario. Siempre dentro de la clase **"Arbol Binario",** agregue el siguiente código:

```
//*******Funciones para dibujar el árbol binario en el formulario **********
//Función para dibujar árbol binario
public void DibujarArbol(Graphics grafo,Font fuente,Brush Relleno,Brush RellenoFuente,Pen Lapiz,Brush
encuentro)
        {
            int x = 400; // Posiciones de la raíz del árbol
            int y = 75;
            if (Raiz == null )
                return;
            Raiz.PosicionNodo( ref x, y); //Posición de cada nodo
            Raiz.DibujarRamas(grafo, Lapiz); //Dibuja los Enlaces entre nodos
            //Dibuja todos los Nodos
            Raiz.DibujarNodo(grafo, fuente, Relleno, RellenoFuente, Lapiz, encuentro);
        }

        public int x1 = 400;
        // Posiciones iniciales de la raíz del árbol
        public int y2 = 75;
        // Función para Colorear los nodos
        public void colorear(Graphics grafo, Font fuente, Brush Relleno, Brush RellenoFuente, Pen
Lapiz, Nodo_Arbol Raiz, bool post, bool inor, bool preor)
        {
            Brush entorno = Brushes.Red;
            if(inor == true )
            {
                if (Raiz != null)
                {
           colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.Izquierdo, post, inor, preor);
                    Raiz.colorear(grafo, fuente, entorno, RellenoFuente, Lapiz);
                    Thread.Sleep(1000);

                    // pausar la ejecución 1000 milisegundos
                    Raiz.colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz);
             colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.Derecho, post, inor,preor);
                }
            }
            else
                if (preor == true)
                {
                    if
                     (Raiz != null)
                    {
                        Raiz.colorear(grafo, fuente, entorno, RellenoFuente, Lapiz);
                        Thread.Sleep(1000);
                        // pausar la ejecución 1000 milisegundos
                        Raiz.colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz);
                        colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.Izquierdo, post,
                                        inor, preor);
```

```
                        colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.Derecho, post,
                                        inor, preor);
                    }
                }
                else
                    if
                     (post ==true)
                    {
                        if
                         (Raiz !=null)
                        {
            colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.Izquierdo, post,inor, preor);
            colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.Derecho, post, inor, preor);
                  Raiz.colorear(grafo, fuente, entorno, RellenoFuente, Lapiz);
                  Thread.Sleep(1000); // pausar la ejecución 1000 milisegundos
                  Raiz.colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz);
                        }
                    }
        }
    }
}
```

7. Ahora se dibujará el formulario para implementar el simulador del Árbol Binario de Búsqueda (ABB) que queremos realizar. El diseño del mismo dependerá de cada estudiante y su distribución, sin embargo en la figura siguiente se muestra cómo podría lucir el simulador.



1 label

3 buttom

3 textbox

8. Una vez definido el diseño, se le proporciona el código que debe ir en el formulario

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

namespace Arbol_Binario
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        //Declaración de variables a utilizar
        int Dato = 0;
        int cont = 0;
        Arbol_Binario mi_Arbol = new Arbol_Binario(null);   //Creación del objeto Árbol
        Graphics g;                                         //Definición del objeto gráfico

         //Evento del formulario que permitirá dibujar el Árbol Binario
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            e.Graphics.Clear(this.BackColor);
            e.Graphics.TextRenderingHint = System.Drawing.Text.TextRenderingHint.AntiAliasGridFit;
            e.Graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
            g = e.Graphics;
       mi_Arbol.DibujarArbol(g, this.Font, Brushes.Blue, Brushes.White, Pens.Black, Brushes.White);

        }

        /*Evento que permitirá insertar un nodo al árbol (código de botón “Insertar Nodo” del
       formulario mostrado en la figura) */
        private void btnInsertar_Click(object sender, EventArgs e)
        {
            if (txtDato.Text == "")
            {
                MessageBox.Show("Debe Ingresar un Valor");
            }
            else
            {
                Dato = int.Parse(txtDato.Text);
                if (Dato <= 0 || Dato >= 100)
              MessageBox.Show("Solo Recibe Valores desde 1 hasta 99", "Error de Ingreso");
                else
                {
                    mi_Arbol.Insertar(Dato);
                    txtDato.Clear();
                    txtDato.Focus();
                    cont++;
                    Refresh();
                    Refresh();
                }
            }
        }

        /*Evento que permitirá eliminar un nodo del árbol (código de botón “Eliminar Nodo” del
       formulario mostrado en la figura) */
        private void btnEliminar_Click(object sender, EventArgs e)
```

```
        {
            if (txtEliminar.Text == "")
            {
                MessageBox.Show("Debe ingresar el valor a eliminar");
            }
            else
            {
                Dato = Convert.ToInt32(txtEliminar.Text);
                if (Dato <= 0 || Dato >= 100)
                {
            MessageBox.Show("Sólo se adminten valores entre 1 y 99", "Error de Ingreso");
                }
                else
                {
                    mi_Arbol.Eliminar(Dato);
                    txtEliminar.Clear();
                    txtEliminar.Focus();
                    cont--;
                    Refresh();
                    Refresh();
                }
            }
        }

       /*Evento que permitirá buscar un nodo en el árbol (código de botón “Buscar Nodo” del
       formulario mostrado en la figura) */
        private void btnBuscar_Click(object sender, EventArgs e)
        {
            if (txtBuscar.Text == "")
            {
                MessageBox.Show("Debe ingresar el valor a buscar");
            }
            else
            {
                Dato = Convert.ToInt32(txtBuscar.Text);
                if (Dato <= 0 || Dato >= 100)
                {
                    MessageBox.Show("Sólo se admiten valores entre 1 y 99", "Error de Ingreso");
                }
                else
                {
                    mi_Arbol.Buscar(Dato);
                    txtBuscar.Clear();
                    txtBuscar.Focus();
                    Refresh();
                    Refresh();
                }
            }
        }
      }
}
```

PARTE II

ÁRBOLES BALANCEADOS

Se considera que un árbol binario está balanceado cuando todos sus niveles, excepto el último, están integrados a la máxima capacidad de nodos.

Existen diferentes propuestas para balancear los árboles y cada una de ellas repercute en la eficiencia de las operaciones de inserción y eliminación de los nodos.

La más común es la técnica de árboles AVL.

ARBOLES AVL

Es un árbol de búsqueda binario que trata de mantenerlo lo más balanceado posible, conforme se realizan operaciones de inserción y eliminación. Fueron propuestos en 1962 por los matemáticos rusos Adelson, Velskii y Landis, de donde surge su nombre.

En los árboles AVL se debe cumplir el hecho de que para cualquier nodo del árbol, la diferencia entre las alturas de sus subárboles no exceda una unidad.

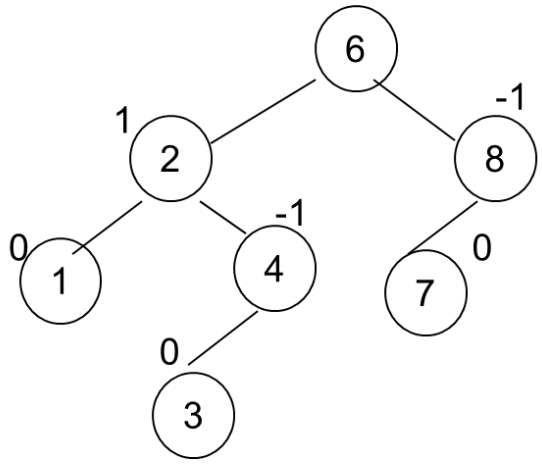
Factor de Balanceo

Los nodos de un árbol AVL guardan un valor -1, 0, 1 , que se conoce como Factor de Balanceo (FB) y representa la altura entre las alturas de sus subárboles.

Un FB igual a cero de un nodo significa que las alturas de sus subárboles son iguales.

Un FB positivo significa que la altura de su subárbol derecho es mayor al subárbol izquierdo. Un

FB negativo significa que la altura de su subárbol izquierdo es mayor al subárbol derecho.



Procedimiento

Ejemplo . Implementación de un árbol balanceado AVL en C#

- Crear un proyecto de tipo Windows Form Application, se sugiere que le dé el nombre de "Arbol_AVL".
- Agregar una clase al proyecto, para facilitar el entendimiento nómbrelo como "ALV".
- Dentro de la clase creada, agregue el siguiente código:

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

namespace Arbol_AVL
{
    class AVL
    {
        public int valor;
        public AVL NodoIzquierdo;
        public AVL NodoDerecho;
        public AVL NodoPadre;
        public int altura;
        public Rectangle prueba;
        private DibujaAVL arbol;
        public AVL()
        {
        }

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

```
        }
        // Constructor.
        public AVL(int valorNuevo, AVL izquierdo, AVL derecho, AVL padre)
        {
            valor = valorNuevo;
            NodoIzquierdo = izquierdo;
            NodoDerecho = derecho;
            NodoPadre = padre;
            altura = 0;
        }

        //Funcion para insertar un nuevo valor en el arbol AVL
        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);
                else
                    Raiz = RotacionIzquierdaDoble(Raiz);
            }

            if (Alturas(Raiz.NodoDerecho) - Alturas(Raiz.NodoIzquierdo) == 2)
            {
                if (valorNuevo > Raiz.NodoDerecho.valor)
                    Raiz = RotacionDerechaSimple(Raiz);
                else
                    Raiz = RotacionDerechaDoble(Raiz);
            }


            Raiz.altura = max(Alturas(Raiz.NodoIzquierdo), Alturas(Raiz.NodoDerecho)) + 1;
            return Raiz;
        }
        //FUNCION DE PRUEBA PARA REALIZAR LAS ROTACIONES

        //Función para obtener que rama es mayor
        private static int max(int lhs, int rhs)
        {
            return lhs > rhs ? lhs : rhs;
        }
        private static int Alturas(AVL Raiz)
        {
            return Raiz == null ? -1 : Raiz.altura;
        }

        AVL nodoE, nodoP;
        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.NodoDerecho.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 segun el caso
                            }
                            else
                            {
                                if (Alturas(Raiz.NodoDerecho) - Alturas(Raiz.NodoIzquierdo) > 0)
                                {

                                    AVL AuxiliarNodo = null;
                                    AVL Auxiliar = Raiz.NodoDerecho;
                                    bool Bandera = false;
                                    while (Auxiliar.NodoIzquierdo != null)
                                    {
                                        AuxiliarNodo = Auxiliar;
                                        Auxiliar = Auxiliar.NodoIzquierdo;
                                        Bandera = true;
                                    }
                                    Raiz.valor = Auxiliar.valor;
                                    NodoEliminar = Auxiliar;
                                    if (Bandera == true)
                                    {
                                        AuxiliarNodo.NodoIzquierdo = Auxiliar.NodoDerecho;
                                    }
                                    else
                                    {
                                        Raiz.NodoDerecho = Auxiliar.NodoDerecho;
                                    }
                                }

                                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.NodoIzquierdo = Auxiliar.NodoIzquierdo;
                                        }
                                    }
                                }

                            }
                        }
                    }
                }
            }
        }
        else
        {

            MessageBox.Show("Nodo inexistente en el arbol", "Error", MessageBoxButtons.OK);
```

```
        }

        return nodoP;

    }

    //Seccion de funciones de rotaciones

    //Rotacion Izquierda Simple
    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
    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
    private static AVL RotacionIzquierdaDoble(AVL k3)
    {
        k3.NodoIzquierdo = RotacionDerechaSimple(k3.NodoIzquierdo);
        return RotacionIzquierdaSimple(k3);
    }

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

    //Funcion para obtener la altura del arbol
    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
    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("Valor no encontrado", "Error", MessageBoxButtons.OK);
        }
/*++++++++++++FUNCIONES PARA DIBUJAR EL ÁRBOL ++++++++++++++*/

        private const int Radio = 30;
        private const int DistanciaH = 40;
        private const int DistanciaV = 10;

        private int CoordenadaX;
        private int CoordenadaY;

        //Encuentra la posición en donde debe crearse el nodo.
        public void PosicionNodo(ref int xmin, int ymin)
        {
            int aux1, aux2;

            CoordenadaY = (int)(ymin + Radio / 2);


            //obtiene la posición del Sub-Árbol izquierdo.
            if (NodoIzquierdo != null)
            {
                NodoIzquierdo.PosicionNodo(ref xmin, ymin + Radio + DistanciaV);
            }
            if ((NodoIzquierdo != null) && (NodoDerecho != null))
            {
                xmin += DistanciaH;
            }

            //Si existe el nodo derecho e izquierdo deja un espacio entre ellos.

            if (NodoDerecho != null)
            {
                NodoDerecho.PosicionNodo(ref xmin, ymin + Radio + DistanciaV);
            }

            // Posicion de nodos dercho e izquierdo.
            if (NodoIzquierdo != null)
            {
                if (NodoDerecho != null)
                {
                    //centro entre los nodos.
                    CoordenadaX = (int)((NodoIzquierdo.CoordenadaX + NodoDerecho.CoordenadaX) / 2);
                }
                else
                {
                    // no hay nodo derecho. centrar al nodo izquierdo.
                    aux1 = NodoIzquierdo.CoordenadaX;
                    NodoIzquierdo.CoordenadaX = CoordenadaX - 40;
                    CoordenadaX = aux1;
                }
            }
            else if (NodoDerecho != null)
            {
                aux2 = NodoDerecho.CoordenadaX;
                //no hay nodo izquierdo.centrar al nodo derecho.
                NodoDerecho.CoordenadaX = CoordenadaX + 40;
                CoordenadaX = aux2;
            }
            else
            {
                // Nodo hoja
                CoordenadaX = (int)(xmin + Radio / 2);
                xmin += Radio;
            }
```

```
        }

        // Dibuja las ramas de los nodos izquierdo y derecho
        public void DibujarRamas(Graphics grafo, Pen Lapiz)
        {
            if (NodoIzquierdo != null)
            {
                grafo.DrawLine(Lapiz, CoordenadaX, CoordenadaY, NodoIzquierdo.CoordenadaX,
NodoIzquierdo.CoordenadaY);
                NodoIzquierdo.DibujarRamas(grafo, Lapiz);
            }
            if (NodoDerecho != null)
            {
                grafo.DrawLine(Lapiz, CoordenadaX, CoordenadaY, NodoDerecho.CoordenadaX, NodoDerecho.CoordenadaY);
                NodoDerecho.DibujarRamas(grafo, Lapiz);
            }

        }

        //Dibuja el nodo en la posición especificada.
        public void DibujarNodo(Graphics grafo, Font fuente, Brush Relleno, Brush RellenoFuente, Pen Lapiz, int
dato, Brush encuentro)
        {
            //Dibuja el contorno del nodo.

            Rectangle rect = new Rectangle(
            (int)(CoordenadaX - Radio / 2),
            (int)(CoordenadaY - Radio / 2),
            Radio, Radio);

            if (valor == dato)
            {
                grafo.FillEllipse(encuentro, rect);
            }
            else
            {
                grafo.FillEllipse(encuentro, rect);
                grafo.FillEllipse(Relleno, rect);
            }
            grafo.DrawEllipse(Lapiz, rect);

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

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

        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 nombre.
            StringFormat formato = new StringFormat();

            formato.Alignment = StringAlignment.Center;
            formato.LineAlignment = StringAlignment.Center;

            grafo.DrawEllipse(Lapiz, rect);
            grafo.FillEllipse(Brushes.PaleVioletRed, rect);
            grafo.DrawString(valor.ToString(), fuente, Brushes.Black, CoordenadaX, CoordenadaY, formato);
        }
    }
}
```

- A continuación se agrega otra clase al proyecto, denominada DibujaAVL.cs

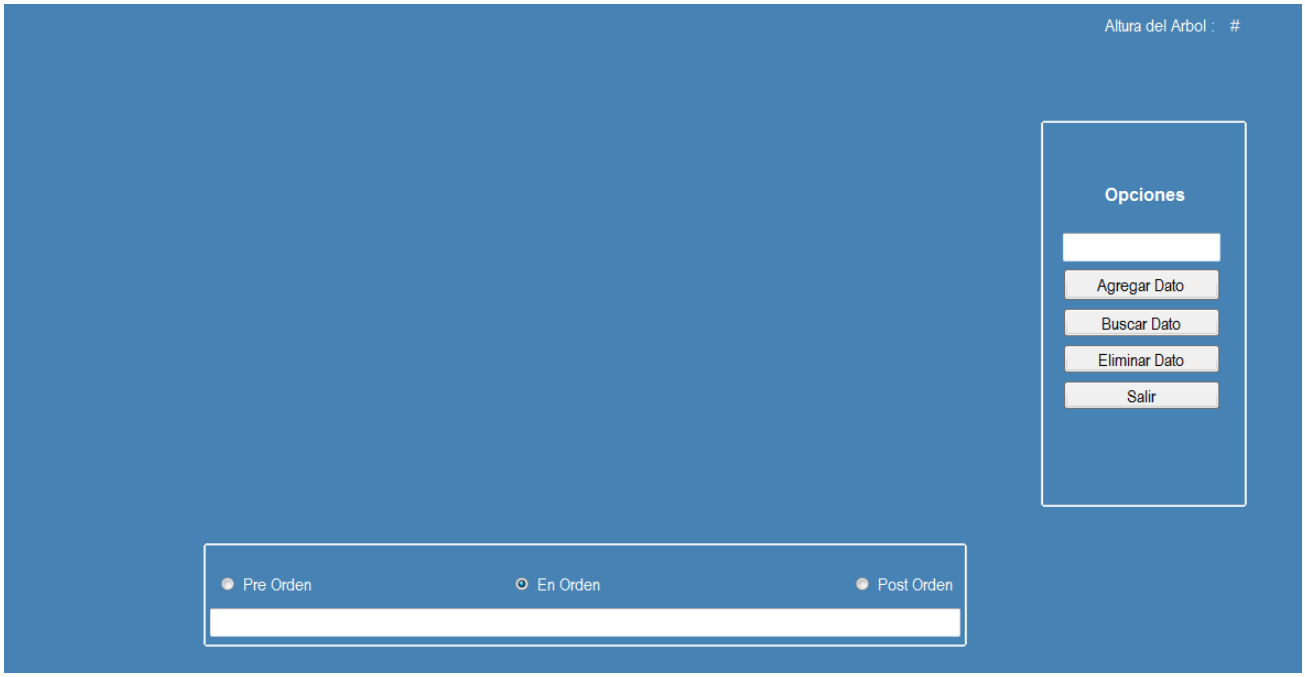
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;

namespace Arbol_AVL
{
    class DibujaAVL
    {
        public AVL Raiz;
        public AVL aux;

        // Constructor.
        public DibujaAVL()
        {
            aux = new AVL();
        }

        public DibujaAVL(AVL RaizNueva)
        {
            Raiz = RaizNueva;
        }

        // Agrega un nuevo valor al arbol.

        public void Insertar(int dato)
        {
            if (Raiz == null)
                Raiz = new AVL(dato, null, null, null);
            else
                Raiz = Raiz.Insertar(dato, Raiz);

        }
        //Eliminar un valor del arbol
        public void Eliminar(int dato)
        {
            if (Raiz == null)
                Raiz = new AVL(dato, null, null, null);
            else
                Raiz.Eliminar(dato, ref Raiz);

        }

        private const int Radio = 30;
        private const int DistanciaH = 40;
        private const int DistanciaV = 10;

        private int CoordenadaX;
```

```
        private int CoordenadaY;

        public void PosicionNodoreocrrido(ref int xmin, ref int ymin)
        {
            CoordenadaY = (int)(ymin + Radio / 2);
            CoordenadaX = (int)(xmin + Radio / 2);
            xmin += Radio;
        }

        public void colorear(Graphics grafo, Font fuente, Brush Relleno, Brush RellenoFuente, Pen Lapiz, AVL Raiz,
bool post, bool inor, bool preor)
        {
            Brush entorno = Brushes.Red;

            if (inor == true)
            {
                if (Raiz != null)
                {

                    colorear(grafo, fuente, Brushes.Blue, RellenoFuente, Lapiz, Raiz.NodoIzquierdo, post, inor,
preor);
                     Raiz.colorear(grafo, fuente, entorno, RellenoFuente, Lapiz);
                    Thread.Sleep(500);
                    Raiz.colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz);
                    colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.NodoDerecho, post, inor, preor);

                }
            }
            else if (preor == true)
            {
                if (Raiz != null)
                {

                    Raiz.colorear(grafo, fuente, Brushes.Yellow, Brushes.Blue, Pens.Black);
                    Thread.Sleep(500);
                    Raiz.colorear(grafo, fuente, Brushes.White, Brushes.Black, Pens.Black);
                    colorear(grafo, fuente, Brushes.Blue, RellenoFuente, Lapiz, Raiz.NodoIzquierdo, post, inor,
preor);
                    colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.NodoDerecho, post, inor, preor);

                }

            }
            else if (post == true)
            {
                if (Raiz != null)
                {
                    colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.NodoIzquierdo, post, inor, preor);
                    colorear(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.NodoDerecho, post, inor, preor);
                    Raiz.colorear(grafo, fuente, entorno, RellenoFuente, Lapiz);
                    Thread.Sleep(500);
                    Raiz.colorear(grafo, fuente, entorno, RellenoFuente, Lapiz);
                }
            }
        }

        public void colorearB(Graphics grafo, Font fuente, Brush Relleno, Brush RellenoFuente, Pen Lapiz, AVL Raiz,
int busqueda)
        {
            Brush entorno = Brushes.Red;
            if (Raiz != null)
            {
                Raiz.colorear(grafo, fuente, entorno, RellenoFuente, Lapiz);

                if (busqueda < Raiz.valor)
                {
                    Thread.Sleep(500);
                    Raiz.colorear(grafo, fuente, entorno, Brushes.Blue, Lapiz);
                    colorearB(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.NodoIzquierdo, busqueda);
```

```
                }
                else
                {
                    if (busqueda > Raiz.valor)
                    {
                        Thread.Sleep(500);
                        Raiz.colorear(grafo, fuente, entorno, RellenoFuente, Lapiz);
                        colorearB(grafo, fuente, Relleno, RellenoFuente, Lapiz, Raiz.NodoDerecho, busqueda);
                    }
                    else
                    {
                        Raiz.colorear(grafo, fuente, entorno, RellenoFuente, Lapiz);
                        Thread.Sleep(500);
                    }
                }
            }
        }

        //Dibuja el árbol
        public void DibujarArbol(Graphics grafo, Font fuente, Brush Relleno, Brush RellenoFuente, Pen Lapiz, int
dato, Brush encuentro)
        {
            int x = 100;
            int y = 75;
            if (Raiz == null) return;

            //Posicion de todos los Nodos.
            Raiz.PosicionNodo(ref x, y);

            //Dibuja los Enlaces entre nodos.
            Raiz.DibujarRamas(grafo, Lapiz);

            //Dibuja todos los Nodos.
            Raiz.DibujarNodo(grafo, fuente, Relleno, RellenoFuente, Lapiz, dato, encuentro);
        }

        public int x1 = 100;
        public int y2 = 75;
        public void restablecer_valores()
        {
            x1 = 100;
            y2 = 75;
        }

        public void buscar(int x)
        {
            if (Raiz == null)
                MessageBox.Show("Arbol AVL Vacío","Error",MessageBoxButtons.OK);
            else
                Raiz.buscar(x, Raiz);
        }

    }
}
```

- Ahora se dibujará el formulario para implementar el simulador del Árbol Balanceado AVL (AVL) que queremos realizar. El diseño del mismo dependerá de cada estudiante y su distribución, sin embargo en la figura siguiente se muestra cómo se sugiere que luzca el simulador. (Agregue también un error provider que se encuentra en el toolbox)

No necesariamente debe eliminar el borde de la ventana, eso es opcional.



- Una vez definido el diseño, se le proporciona el código que debe ir en el formulario

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace Arbol_AVL
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        int cont = 0;
        int dato = 0;
        int datb = 0;
        int cont2 = 0;

        DibujaAVL arbolAVL = new DibujaAVL(null);
        DibujaAVL arbolAVL_Letra = new DibujaAVL(null);
        Graphics g;

//No olvidar generar el evento Paint que es importante para dibujar en el form
        private void Form1_Paint(object sender, PaintEventArgs en)
        {
            en.Graphics.Clear(this.BackColor);
            en.Graphics.TextRenderingHint = System.Drawing.Text.TextRenderingHint.AntiAliasGridFit;
            en.Graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
            g = en.Graphics;

                arbolAVL.DibujarArbol(g, this.Font,
                   Brushes.White, Brushes.Black, Pens.White, datb, Brushes.Black);
            datb = 0;
```

```
            if (pintaR == 1)
            {
                arbolAVL.colorear(g, this.Font, Brushes.Black, Brushes.Yellow, Pens.Blue, arbolAVL.Raiz, post.Checked,
ino.Checked, pre.Checked);
                pintaR = 0;
            }
            if (pintaR == 2)
            {
                arbolAVL.colorearB(g, this.Font, Brushes.White, Brushes.Red, Pens.White, arbolAVL.Raiz,
int.Parse(valor.Text));
                pintaR = 0;
            }
        }

        private void btnIngresar_Click(object sender, EventArgs e)
        {
            errores.Clear();
            if (valor.Text == "")
            {
                errores.SetError(valor,"Valor obligatorio");
            }
            else
            {
                try{
                dato = int.Parse(valor.Text);

                    arbolAVL.Insertar(dato);
                    valor.Clear();
                    valor.Focus();
                    lblaltura.Text = arbolAVL.Raiz.getAltura(arbolAVL.Raiz).ToString();
                    cont++;
                    Refresh();
                    Refresh();
                }catch(Exception ex){
                    errores.SetError(valor,"Debe ser numérico");
                }
            }
        }

        int pintaR = 0;
        private void btnBuscar_Click(object sender, EventArgs e)
        {
            errores.Clear();
            if (valor.Text == "")
            {
                errores.SetError(valor,"Valor obligatorio");
            }
            else
            {
                try{
                datb = int.Parse(valor.Text);
                arbolAVL.buscar(datb);
                pintaR = 2;
                Refresh();
                valor.Clear();

                }catch(Exception ex){
                    errores.SetError(valor,"Debe ser numérico");
                }
            }
        }
        private void btneliminar_Click(object sender, EventArgs e)
        {
            errores.Clear();
            if (valor.Text == "")
            {
                errores.SetError(valor,"Valor obligatorio");
            }
```

```
            else
            {
                try{
                dato = int.Parse(valor.Text);
                valor.Clear();
                arbolAVL.Eliminar(dato);
                lblaltura.Text = arbolAVL.Raiz.getAltura(arbolAVL.Raiz).ToString();
                Refresh();
                Refresh();
                cont2++;
                }catch(Exception ex){

                    errores.SetError(valor,"Debe ser numérico");
                }
            }
            Refresh(); Refresh(); Refresh();
        }

        //Al presionar enter que realice las operaciones
        private void valor_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == Convert.ToChar(Keys.Enter))
            {
                errores.Clear();

                if (valor.Text == "")
                {
                    errores.SetError(valor, "Valor obligatorio");
                }
                else
                {
                    try
                    {

                        dato = int.Parse(valor.Text);
                        if (dato > 0)
                        {
                            arbolAVL.Insertar(dato);


                            valor.Clear();
                            valor.Focus();
                            lblaltura.Text = arbolAVL.Raiz.getAltura(arbolAVL.Raiz).ToString();
                            cont++;
                            Refresh();
                            Refresh();
                        }
                        else
                        {
                            errores.SetError(valor, "Debe ser un numero mayor que 0");
                        }
                    }
                    catch (Exception ex)
                    {
                        errores.SetError(valor, "Debe ser numérico");
                    }
                }
            }
        }

        private void bsalir_Click_1(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Desarrollo de habilidades

- Realizar las modificaciones necesarias, para que el simulador de AVL realice los siguientes recorridos:
  - ✓ Recorrido en orden
  - ✓ Recorrido Pre-orden
  - ✓ Recorrido Post-orden

- Tomando como base el ejercicio dado, modifique para:
  - ✓ Incluir en el formulario el valor del Factor de Balanceo por cada nodo, cada vez que se inserte o se elimine un dato
  - ✓ Muestre al usuario qué tipo de rotación se realizará (RSI-RSD-RDI-RDD)