Facultad: Ingeniería
Escuela: Computación
Asignatura: Programación IV

Tema: Arboles en C#.

Objetivos Específicos

- Definir el concepto de la estructura de datos Árbol.
- Implementar la estructura de datos Árbol en C #.

Materiales y Equipo

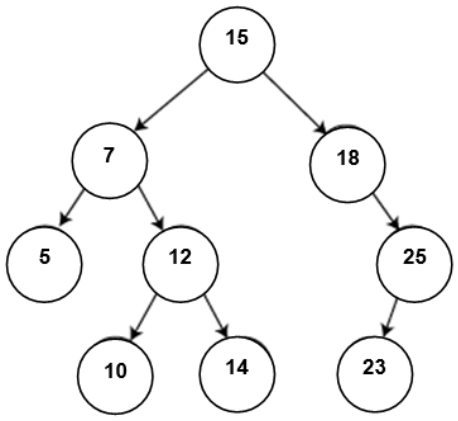
- Guía Número 7.
- Computadora con programa Microsoft Visual C#.

Introducción Teórica

Definición de Árbol Binario.

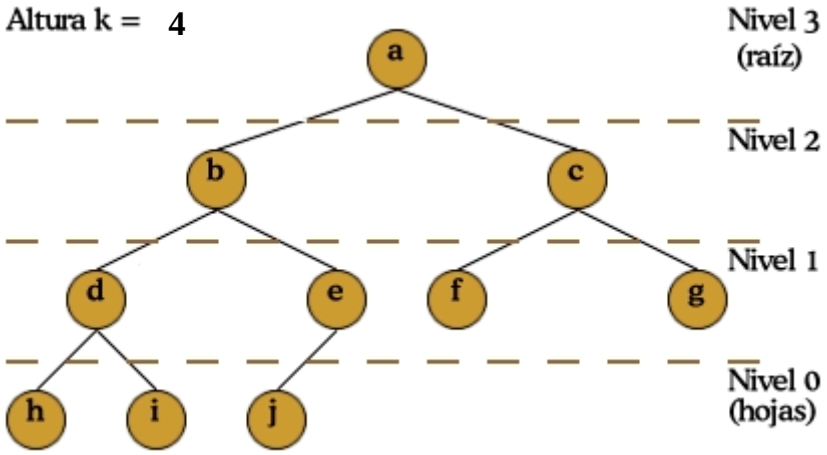
Un árbol binario es una estructura de datos de tipo árbol en donde cada uno de los nodos del árbol puede tener 0, 1, ó 2 sub árboles llamados de acuerdo a su caso como:

1. Si el nodo raíz tiene 0 relaciones, se llama 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.



La Figura anterior muestra un árbol binario sencillo de tamaño 9 y altura 4, con un nodo raíz cuyo valor es 15.

La Figura siguiente muestra un Árbol binario esencialmente completo.



Existen cuatro tipos de árbol binario:

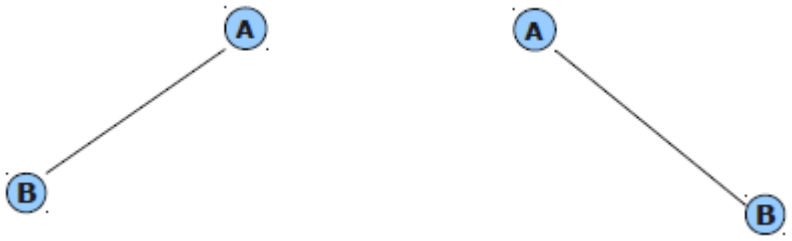
1. Arboles Binarios Distintos.
2. Arboles Binarios Similares.
3. Arboles Binarios Equivalentes.
4. Arboles Binarios Completos.

A continuación se hará una breve descripción de los diferentes tipos de árbol binario así como un ejemplo de cada uno de ellos.

Arboles Binarios Distintos.

Se dice que dos árboles binarios son distintos cuando sus estructuras son diferentes.

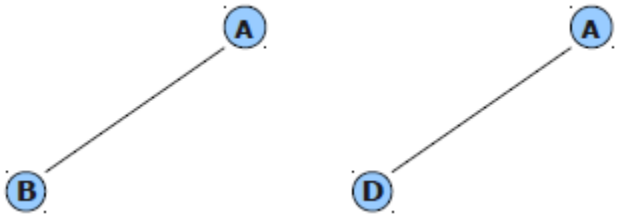
Ejemplo:



Arboles Binarios Similares.

Dos árboles binarios son similares cuando sus estructuras son idénticas, pero la información que contienen sus nodos es diferente.

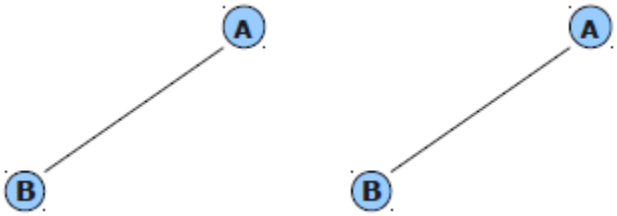
Ejemplo:



Arboles Binarios Equivalentes.

Son aquellos árboles que son similares y que además los nodos contienen la misma información.

Ejemplo:



Arboles Binarios Completos.

Son aquellos árboles en los que todos sus nodos excepto los del último nivel, tienen dos hijos: el subárbol izquierdo y el subárbol derecho.

Los nodos del árbol binario serán representados como registros que contendrán como mínimo tres campos. En un campo se almacenará la información del nodo. Los dos restantes se utilizarán para apuntar al subárbol izquierdo y derecho del subárbol en cuestión.

Cada nodo se representa gráficamente de la siguiente manera:

Izquierdo	Centro	Derecho

Procedimiento

Ejemplo 1. Implementaremos un árbol binario de búsqueda (ABB) en C#:

1. Crear un proyecto de tipo Windows Forms Application, se sugiere darle el nombre de “Arbol Binario”.

2. Agregar una clase al proyecto, se sugiere darle el nombre de “Nodo Arbol”. Esta clase la utilizaremos para definir el elemento “nodo” del árbol binario.

3. En esta clase, agregar el siguiente código:

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

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

      private Arbol_Binario arbol;        // declarando un objeto de tipo Árbol Binario

      public Nodo_Arbol ( )               // Constructor por defecto
      {
      }

      public Arbol_Binario Arbol       // Constructor por defecto para el objeto de tipo 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 Binario
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 Árbol", "Error de Ingreso");
   }

   return t;
}

      // Función para calcular la altura de un nodo en el Árbol Binario
private static int Alturas (Nodo_Arbol t)
{
   return t == null ? -1 : t.altura;
}

      // Función para eliminar un nodo del Árbol Binario
public void Eliminar (int x, ref Nodo_Arbol t)
{  if (t != null)    // Si 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;  // Ya se ubicó el nodo que se desea eliminar
            if (NodoEliminar.Derecho == null)      // Se verifica si tiene hijo derecho
            {
               t = NodoEliminar.Izquierdo;
            }
            else
            {  if (NodoEliminar.Izquierdo == null) // Se verifica si tiene hijo izquierdo
```

```
{
   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;
      }

      t.info = Auxiliar.info;              // Se crea un nodo temporal
      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 en el Árbol", "Error de Eliminación");
    }
}          // Final de la función Eliminar

    // Función para recorrer el Árbol Binario (búsqueda de un nodo)
public void buscar (int x, Nodo_Arbol t)
{
    if (t != null)
    {   if (x < t.info)                    // Búsqueda por el subárbol izquierdo
        {    buscar (x, t.Izquierdo);
        }
        else
        {   if (x > t.info)                // Búsqueda por el subárbol derecho
            {    buscar (x, t.Derecho);
            }
        }
    }
    else
        MessageBox.Show ("Nodo NO Encontrado en el Árbol", " Error de Búsqueda");
}
```

4. A continuación agregar las funciones que servirán para dibujar el Árbol Binario en el formulario. Siempre en la misma clase “Nodo Arbol”, agregar el siguiente código:

```
// ***** Funciones para el dibujo de los nodos del Árbol Binario en el Formulario *****

   // Variable que define el tamaño de los círculos que representarán los nodos del árbol
   private const int Radio = 30;
   private const int DistanciaH = 80;        // variable para manejar distancia horizontal
   private const int DistanciaV = 10;        // variable para manejar distancia vertical

   private int CoordenadaX;                  // variable para manejar posición Eje X
   private int CoordenadaY;                  // variable para manejar posición Eje Y

       //Función para encontrar la posición donde se creará (dibujará) el nodo
   public void PosicionNodo (ref int xmin, int ymin)
   {
      int aux1, aux2;
      CoordenadaY = (int) (ymin + Radio / 2);

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

      if ((Izquierdo != null) && (Derecho != null))
      {    xmin += DistanciaH;
      }

            //Si existe el 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)
{
         // 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);

   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);
   }
}
```

```
        // Función para dar color al nodo del Árbol Binario
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);

   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);
}
```

5. Agregar una clase al proyecto, se sugiere darle el nombre de “Arbol Binario”. Esta clase la utilizaremos para definir la estructura “Árbol”.

6. En esta clase, agregar el siguiente código:

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


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

            // Constructor por defecto
      public Arbol_Binario ( )
      {
         aux = new Nodo_Arbol ( );
      }
```

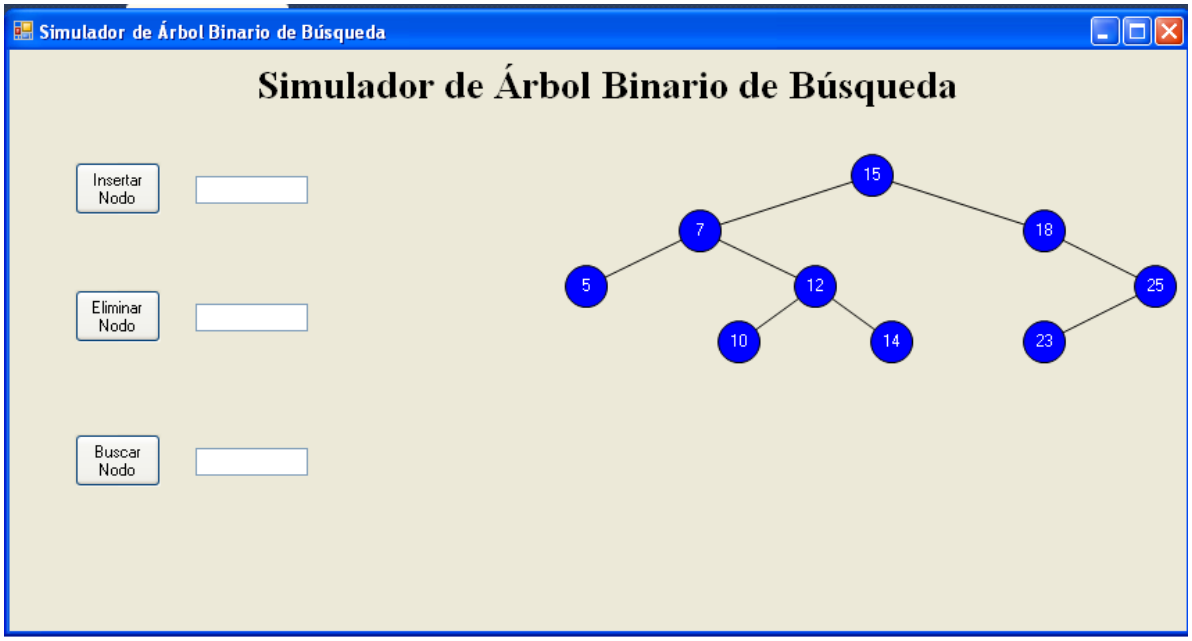
```
        // Constructor con parámetros
    public Arbol_Binario (Nodo_Arbol nueva_raiz)
    {
        Raiz = nueva_raiz;
    }

        // 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);
    }
```

7. A continuación agregar las funciones que servirán para dibujar el Árbol Binario en el formulario. Siempre en la misma clase “Arbol Binario”, agregar el siguiente código:

```
//  ******** Funciones para el dibujo del Árbol Binario en el Formulario *******

      // Función que dibuja el 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);
      }
    }
}
```

Ahora debemos diseñar el formulario que nos servirá para implementar el simulador del Árbol Binario de Búsqueda que queremos realizar.
Queda a creatividad de cada estudiante el diseño del mismo.
En la siguiente imagen se muestra cómo podría lucir el formulario:



A continuación, se les proporciona parte del código que debe ir en el formulario que deben diseñar a su gusto.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
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 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;

   mi_Arbol.DibujarArbol (g, this.Font, Brushes.Blue, Brushes.White, Pens.Black,
                              Brushes.White);
}

      /* Evento que permitirá insertar un nodo al árbol (código del 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 ( );
       }
    }
 }
```

Análisis de resultados

1. A partir de todo el código proporcionado, se les pide implementar las siguientes funciones:
   a. Eliminar Nodo.
   b. Buscar Nodo.

2. Realizar las modificaciones necesarias, para agregar la siguiente funcionalidad a la aplicación de Simulación del Árbol Binario de Búsqueda:

a. Desarrolle el método “Mostrar/Recorrer el Árbol” usando los siguientes recorridos:

- Recorrido en orden.
- Recorrido en Pre-orden.
- Recorrido en Post-orden.

Los algoritmos se muestran a continuación:

```
Recorrido enOrden (ArbolBinario raíz)
{   if (raiz)
    {       enOrden (raiz–>izq);
            visitar (raiz–>dato);
            enOrden (raiz–>der);
    }
}

Recorrido PreOrden (ArbolBinario raíz)
{   if (raiz)
    {       visitar (raiz–>dato);
            PreOrden (raiz–>izq);
            PreOrden (raiz–>der);
    }
}

Recorrido PostOrden (ArbolBinario raíz)
{   if (raiz)
    {       PostOrden (raiz–>izq);
            PostOrden (raiz–>der);
            visitar (raiz–>dato);
    }
}
```

3. Aplicar las modificaciones necesarias, para agregar mayor funcionalidad al programa simulador del Árbol ABB.

Deben implementarse las siguientes opciones:

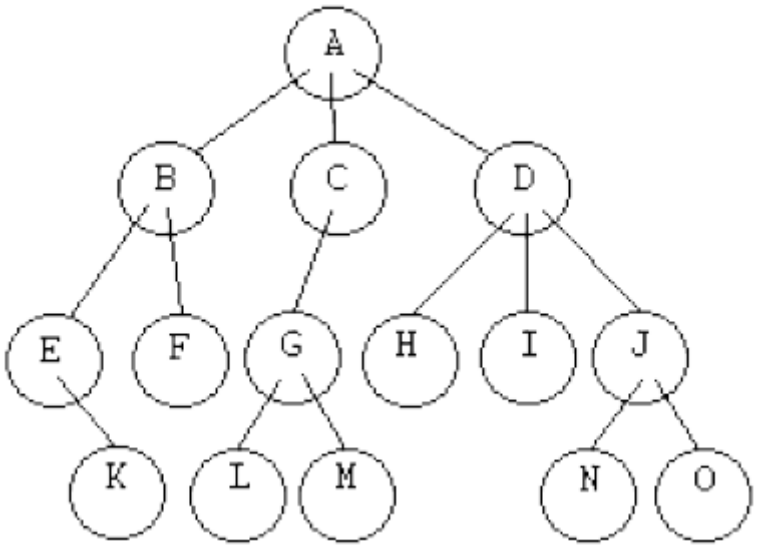
a. Mostrar la suma de los elementos que conforman el árbol (el valor de sus nodos).
b. Mostrar la suma de los elementos que son múltiplos de 2, de 3 y de 5 (el valor de sus nodos), identificarlos en el árbol (los nodos que están siendo operados).
c. Determinar el elemento máximo y el elemento mínimo en un ABB (identificarlos gráficamente en el árbol).
d. Determinar la altura del ABB.

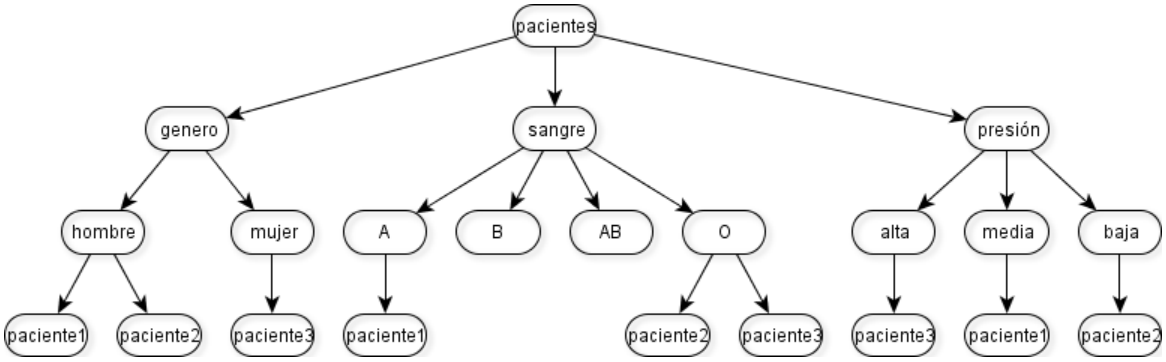
Investigación Complementaria

Para la siguiente semana:

Implementar en una interfaz gráfica de formulario (Windows Forms), un simulador de árboles binarios que permita realizar lo siguiente:

1. Permitir diseñar y manejar árboles genéricos, es decir, aquellos en que los nodos tengan la posibilidad de referenciar a más de dos nodos hijos. Se muestran unos ejemplos de este tipo de árboles.





2. Determinar si dos árboles binarios son similares o distintos. Debe mostrarse como salida los dos árboles comparados y la indicación si son similares o no y por qué; ó si son distintos o no y por qué.

Guía 7: Arboles en C#.

Hoja de cotejo: 7

Alumno:

Máquina No:

Docente:

GL:

Fecha:

EVALUACIÓN					
	%	**1-4**	**5-7**	**8-10**	**Nota**
CONOCIMIENTO	Del 20 al 30%	Conocimiento deficiente de los fundamentos teóricos	Conocimiento y explicación incompleta de los fundamentos teóricos	Conocimiento completo y explicación clara de los fundamentos teóricos	
APLICACIÓN DEL CONOCIMIENTO	Del 40% al 60%				
ACTITUD	Del 15% al 30%	No tiene actitud proactiva.	Actitud propositiva y con propuestas no aplicables al contenido de la guía.	Tiene actitud proactiva y sus propuestas son concretas.	
TOTAL	100%				