REPORTE DE PRÁCTICA

| Nombre Completo | Machado Sanchez Javier |
| --- | --- |
| Número de la Unidad a Evaluar | 4 |
| Tema a Evaluar | Manejando Árboles con Recursividad |
| Fecha de Entrega | 26 de Octubre del 2023 |

**Programa Fuente**

| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace P30\_Manejando\_Arboles\_Recursividad  {  internal class Program  {  //Declaración de clase árbol  public class clsArbolBinario  { //Elemento  public string nombre;  //Apuntadores  public clsArbolBinario Derecha;  public clsArbolBinario Izquierda;  }  //Fin clase ÁrbolBinario  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Inicio clase Métodos Árbol  public class clsMetodosArbol  {  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Inicio métodos  //Inicio procedimiento insertar  public void InsertarElemento()  {  if (Raiz == null)  { //Creación de raíz en caso de estar vacío el árbol  Raiz = new clsArbolBinario  {  nombre = elemento,  Derecha = null,  Izquierda = null  };  Console.WriteLine($"\n\n\tSe ha ingresado el número {elemento} con éxito.");  }  else  { //Cuándo ya hay raíz  Program.flag = false;  Temp1 = Raiz;  //Ciclo para encontrar el lugar del valor dentro del árbol  do  { //Condición para encontrar el subárbol al que pertenece el número  if (elemento.CompareTo(Temp1.nombre) > 0)  { //Comprobando si existe elemento en la rama derecha  if (Temp1.Derecha == null)  { //Insertando número  Temp2 = new clsArbolBinario  {  nombre = elemento,  Derecha = null,  Izquierda = null  };  Temp1.Derecha = Temp2;  flag = true; //Bandera para saber que ya se insertó  }  else Temp1 = Temp1.Derecha;  }  else  { //Comprobando si existe elemento en la rama izquierda  if (Temp1.Izquierda == null)  { //Insertando número  Temp2 = new clsArbolBinario  {  nombre = elemento,  Derecha = null,  Izquierda = null  };  Temp1.Izquierda = Temp2;  flag = true; //Bandera para saber que ya se insertó  }  else Temp1 = Temp1.Izquierda;  }  } while (flag != true);  Console.WriteLine($"\n\n\tSe ha ingresado el número {elemento} con éxito.");  }  }  //Fin procedimiento insertar  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Inicio procedimiento PreOrden  public void PreOrden(clsArbolBinario Resultado)  {  if (Resultado != null) //Condición para caso base  {  Console.Write($"[ {Resultado.nombre} ] "); //Raíz  PreOrden(Resultado.Izquierda); //Llamador más pequeño para subárbol izquierdo  PreOrden(Resultado.Derecha); //Llamador más pequeño para subárbol derecho  }  }  //Fin procedimiento PreOrden  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Inicio procedimiento InOrden  public void InOrden(clsArbolBinario Resultado)  {  if (Resultado != null) //Condición para caso base  {  InOrden(Resultado.Izquierda); //Llamador más pequeño para subárbol izquierdo  Console.Write($"[ {Resultado.nombre} ] "); //Raíz  InOrden(Resultado.Derecha); //Llamador más pequeño para subárbol derecho  }  }  //Fin procedimiento InOrden  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Inicio procedimiento PostOrden  public void PostOrden(clsArbolBinario Resultado)  {  if (Resultado != null) //Condición para caso base  {  PostOrden(Resultado.Izquierda); //Llamador más pequeño para subárbol izquierdo  PostOrden(Resultado.Derecha); //Llamador más pequeño para subárbol derecho  Console.Write($"[ {Resultado.nombre} ] "); //Raíz  }  }  //Fin procedimiento PostOrden  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Inicio procedimiento Amplitud  public void Amplitud(Queue<clsArbolBinario> ArbolAmplitud)  {  if (ArbolAmplitud.Count > 0) //Condición para caso base  {  clsArbolBinario Auxiliar = ArbolAmplitud.Dequeue(); //Pasando un apuntador hacia otro temporal y eliminando de la cola  Console.Write($"[ {Auxiliar.nombre} ] "); //Raíz  if (Auxiliar.Izquierda != null)  {  ArbolAmplitud.Enqueue(Auxiliar.Izquierda);  }  if (Auxiliar.Derecha != null)  {  ArbolAmplitud.Enqueue(Auxiliar.Derecha);  }  Amplitud(ArbolAmplitud); //Llamador más pequeño  }  }  //Fin procedimiento Amplitud  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Inicio procedimiento Eliminar  public clsArbolBinario Eliminar(clsArbolBinario Resultado, string elemento)  {  if (elemento.CompareTo(Resultado.nombre) < 0) Resultado.Izquierda = Eliminar(Resultado.Izquierda, elemento);  else if (elemento.CompareTo(Resultado.nombre) > 0) Resultado.Derecha = Eliminar(Resultado.Derecha, elemento);  else  {  //Nodo hoja  if (Resultado.Izquierda == null && Resultado.Derecha == null) Resultado = null;  //Nodo con un hijo  else if (Resultado.Izquierda == null) Resultado = Resultado.Derecha;  else if (Resultado.Derecha == null) Resultado = Resultado.Izquierda;  //Nodo con dos hijos  else  {  string valor\_minimo = SmallestNode(Resultado.Izquierda);  Resultado.nombre = valor\_minimo;  Resultado.Derecha = Eliminar(Resultado.Derecha, valor\_minimo);  }  }  return Resultado;  }  //Fin procedimiento Eliminar  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Inicio función SmallestNode  public string SmallestNode(clsArbolBinario Resultado)  {  string valor\_minimo = Resultado.nombre;  while (Resultado.Izquierda != null)  {  valor\_minimo = Resultado.Izquierda.nombre;  Resultado = Resultado.Izquierda;  }  return valor\_minimo;  }  //Fin función SmallestNode  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Inicio procedimiento Mostrar Hijos  public void MostrarHijos(clsArbolBinario Resultado)  {  if (Resultado != null)  {  if(elemento.CompareTo(Resultado.nombre) == 0)  {  if (Resultado.Izquierda != null) Console.Write($"[{Resultado.Izquierda.nombre}]");  if (Resultado.Derecha != null) Console.Write($"[{Resultado.Derecha.nombre}]");  }  MostrarHijos(Resultado.Izquierda);  MostrarHijos(Resultado.Derecha);  }  }  //Fin procedimiento Mostrar Hijos  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Fin métodos  }  //Fin clase clsMetodosArbol  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // VARIABLES GLOBALES  public static string elemento;  public static bool flag;  public static Queue<clsArbolBinario> ArbolAmplitud = new Queue<clsArbolBinario>();  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // DECLARACIÓN DE ARBOLES  public static clsArbolBinario Raiz, Resultado, Temp1, Temp2 = new clsArbolBinario();    //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  static void Main(string[] args)  {  Console.Title = "P29 Árboles Binarios Inserción y Recorridos";  //Declaración de variables auxiliares para control de menú  char opcion; bool aux\_repetir;  //Declaración de objeto para árbol binario  clsMetodosArbol MetodosArbol = new clsMetodosArbol();  //Declaración de menú  do  {  Console.Write("\n\t\t.: MENÚ OPCIONES :." +  "\n\n\t[1] - Insertar" +  "\n\t[2] - PreOrden" +  "\n\t[3] - InOrden" +  "\n\t[4] - PostOrden" +  "\n\t[5] - Amplitud" +  "\n\t[6] - Eliminar" +  "\n\t[7] - Mostrar Hijos" +  "\n\t[8] - Salir" +  "\n\n\tIngrese el número de la opción deseada: ");  if (char.TryParse(Console.ReadLine(), out opcion))  {  Console.Write($"\n\n\tSe ha capturado la opción \"{opcion}\" correctamente" +  $"\n\n\tPresione la tecla <INTRO> para continuar...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  //Switch case para controlar las opciones  switch (char.ToUpper(opcion))  {  case '1':  {  do  { //Captura de datos  Console.Write("\n\t\t.: INSERCCIÓN :." +  "\n\n\tIngrese el nombre a insertar: ");  Program.elemento = Console.ReadLine();  //Llamada a método de insertar  MetodosArbol.InsertarElemento();  Resultado= Raiz;  do  {  Console.Write("\n\t¿Desea insertar otro elemento [S=Sí][N=No] ?" +  "\n\tIngrese la letra de la respuesta: ");  if (char.TryParse(Console.ReadLine(), out opcion)) aux\_repetir = false;  else  {  aux\_repetir = true;  Console.Write("\n\n\tLa opción no se ha podido capturar" +  "\n\n\tPresione la tecla <INTRO> para intentarlo de nuevo...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  }  } while (aux\_repetir);  Console.Write("\n\n\tPresione la tecla <INTRO> para continuar...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  } while (char.ToUpper(opcion) != 'N');  }  break;  case '2':  {  Console.Write("\n\t\t.: RECORRIDO PREORDEN :." +  "\n\n\t| Este es el recorrido en PreOrden del árbol |\n\n\t");  //Llamada a método para recorrer en PreOrden  MetodosArbol.PreOrden(Program.Resultado);  Console.Write("\n\n\tPresione la tecla <INTRO> para continuar...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  }  break;  case '3':  {  Console.Write("\n\t\t.: RECORRIDO INORDEN :." +  "\n\n\t| Este es el recorrido en InOrden del árbol |\n\n\t");  //Llamada a método para recorrer en InOrden  MetodosArbol.InOrden(Program.Resultado);  Console.Write("\n\n\tPresione la tecla <INTRO> para continuar...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  }  break;  case '4':  {  Console.Write("\n\t\t.: RECORRIDO POSTORDEN :." +  "\n\n\t| Este es el recorrido en PostOrden del árbol |\n\n\t");  //Llamada a método para recorrer en PostOrden  MetodosArbol.PostOrden(Program.Resultado);  Console.Write("\n\n\tPresione la tecla <INTRO> para continuar...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  }  break;  case '5':  {  Console.Write("\n\t\t.: RECORRIDO AMPLITUD :." +  "\n\n\t| Este es el recorrido en Amplitud del árbol |\n\n\t");  //Condición para validar si esta vacío el árbol  if (Raiz == null) Console.WriteLine("\n\tÁrbol vacío");  else  {  ArbolAmplitud.Clear();  ArbolAmplitud.Enqueue(Raiz);  //Llamada a método para recorrer en Amplitud  MetodosArbol.Amplitud(Program.ArbolAmplitud);  }  Console.Write("\n\n\tPresione la tecla <INTRO> para continuar...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  }  break;  case '6':  {  do  { //Captura de datos  Console.Write("\n\t\t.: ELIMINACIÓN :." +  "\n\n\tIngrese el nombre a eliminar: ");  Program.elemento = Console.ReadLine();  if (Raiz == null) Console.Write("\n\tÁrbol vacío");  else  {  if (Resultado == null) Console.WriteLine($"\n\tEl nombre \"{elemento}\" no existe.");  else  {  //Llamada a método de eliminar  Raiz = MetodosArbol.Eliminar(Resultado, Program.elemento);  Resultado = Raiz;  Console.Write($"\n\n\tEl nombre \"{elemento}\" se ha eliminado correctamente.");  do  {  Console.Write("\n\n\t¿Desea eliminar otro elemento [S=Sí][N=No] ?" +  "\n\tIngrese la letra de la respuesta: ");  if (char.TryParse(Console.ReadLine(), out opcion)) aux\_repetir = false;  else  {  aux\_repetir = true;  Console.Write("\n\n\tLa opción no se ha podido capturar" +  "\n\n\tPresione la tecla <INTRO> para intentarlo de nuevo...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  }  } while (aux\_repetir);  }  }  Console.Write("\n\n\tPresione la tecla <INTRO> para continuar...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  } while (char.ToUpper(opcion) != 'N');  }  break;  case '7':  {  do  { //Captura de datos  Console.Write("\n\t\t.: MOSTRAR HIJOS :." +  "\n\n\tIngrese el nombre del padre: ");  Program.elemento = Console.ReadLine();  if (Raiz == null) Console.Write("\n\tÁrbol vacío");  else  {  //Llamada a método de mostrar hijos  Console.Write($"\n\n\t| Estos son los nodos del padre \"{elemento}\" |\n\n\t");  MetodosArbol.MostrarHijos(Resultado);  do  {  Console.Write("\n\n\t¿Desea mostrar los hijos de otro elemento [S=Sí][N=No] ?" +  "\n\tIngrese la letra de la respuesta: ");  if (char.TryParse(Console.ReadLine(), out opcion)) aux\_repetir = false;  else  {  aux\_repetir = true;  Console.Write("\n\n\tLa opción no se ha podido capturar" +  "\n\n\tPresione la tecla <INTRO> para intentarlo de nuevo...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  }  } while (aux\_repetir);  }  Console.Write("\n\n\tPresione la tecla <INTRO> para continuar...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  } while (char.ToUpper(opcion) != 'N');  }  break;  case '8':  {  //Caso 4. Salida del programa  Console.Write("\n\t\t:. SALIDA :." +  "\n\n\tGracias por utilizar nuestro programa" +  "\n\n\tPresione la tecla <Esc> para salir...");  while (Console.ReadKey().Key != ConsoleKey.Escape) ;  }  break;  default:  {  //En caso de ingresar una opción inválida  Console.Write("\n\t\t.: OPCIÓN INVALIDA :." +  "\n\n\tPor favor ingrese una opción existente" +  "\n\n\tPresione la tecla <INTRO> para continuar...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  }  break;  }  }  else  {  Console.Write("\n\t\t.: OPCIÓN INVALIDA :." +  "\n\n\tPor favor ingrese una opción existente" +  "\n\n\tPresione la tecla <INTRO> para continuar...");  while (Console.ReadKey().Key != ConsoleKey.Enter) ; Console.Clear();  }  } while (opcion != '8');  }  }  } |
| --- |

**Ventana de Entrada de datos**

| Recorrido PreOrden:  Javier, Alejandro, Beto, Sergio, Karla, Victor  Recorrido InOrden:  Alejandro, Beto, Javier, Karla, Victor, Sergio  Recorrido PostOrden:  Beto, Alejandro, Karla, Victor, Sergio, Javier  Recorrido Amplitud:  Javier, Alejandro, Sergio, Beto, Karla, Victor  Hijos de Alejandro: Beto  Hijos de Sergio: Karla, Victor |
| --- |

**Ventana de Resultados**

|  |
| --- |

**EXPLICA BREVEMENTE QUÉ TE PARECIÓ EL PROBLEMA, QUE SE TE DIFICULTO Y COMO LO SOLUCIONASTE.**

| Un programa divertido de realizar y de analizar. Tuve algunas complicaciones con el método de eliminar, puesto que el código siempre al eliminar una raíz con dos hijos, sube siempre el valor más pequeño del subárbol derecho. En un proyecto de prueba traté de solucionarlo, al final entendí que el programa debía encontrar el valor más pequeño del subárbol derecho y el valor más grande del subárbol izquierdo para así compararlos respecto a la raíz y conocer el valor más cercano para subirlo como reemplazo. Lo pude lograr con datos numéricos me fue imposible replicarlo, debido a que trabajar con cadenas es muy distinto. Todo ese código lo descarté para este programa y dejé el original, que tiene un problema de excepción que se da en casos específicos. Este problema lo arreglé en el código de prueba pero como lo descarté decidí subir este con el problema.  Se me ha ocurrido pasar esas dos cadenas y pasarlas a un arreglo de char, pienso que de esa forma se podrá comprar los dos valores con la raíz y así poder ver cual es el más cercano. Esto no lo desarrollo aún debido a que no tengo tiempo de sobra por el momento. |
| --- |