PROGRAMACIÓN DE ÁRBOLES BINARIOS EN POO

J.B. Escobar Garcia
7690-25-1502 Universidad Mariano Gálvez
Matemática Discreta
jescobarg20@miumg.edu.com

2.Resúmen

Implementar programas en C++ que nos permitan crear, manipular y visualizar árboles binarios, aplicando los principios de POO, comprende estructuras internas de nodos y relaciones jerárquicas entre ellos mediante las inserciones, búsquedas y los recorridos. Se implementa un programa que crea árboles binarios, permitiendo al usuario establecer una raíz y agregar nuevos nodos en las posiciones de izquierda o derecha, siempre que estas estén disponibles.

El sistema incluye mecanismos de validación, como es la verificación de duplicidad y comprobación de nodos existentes antes de agregar nuevos elementos, lo que garantiza la integridad. Los recorridos preorden, inorden, postorden, son ejecutados correctamente, evidenciando la forma en que trabaja cada método y en el procesamiento de los datos.

3. Palabras clave POO, árboles binarios, preorden, inorden, postorden.

4.Desarrollo del tema

Un árbol es una estructura no lineal formada por un conjunto de nodos y ramas.

En los árboles binarios existen nodos especiales los cuales son llamados raíz. Así mismo, nodos que salen de la raíz llamados nodos rama (los hijos de las raíces), un nodo rama que no contiene hijos es llamado nodo hoja.

Los árboles binarios son conjuntos finitos de nodos los cuales constan de un nodo raíz la cual contiene dos subárboles binarios los cuales son, subárbol izquierdo y subárbol derecho.

Dependiendo de la raíz se realiza el análisis en que dirección comenzaran a descender los subárboles, teniendo en cuenta que al lado izquierdo irán los que son menores a la raíz, en el lado derecho se agregaran los datos mayores a la raíz, siguiendo esta lógica al ir descendiendo ya sea del lado izquierdo o derecho utilizaremos siempre la misma lógica, los mayores se dirigen al lado derecho y los menores al lado izquierdo.

Se pueden utilizar tres formas para realizar los recorridos de árboles binarios, preorden, inorden, postorden.

En programación son estructuras de datos fundamentales que desempeñan papeles en la organización y manipulación de datos. Una de las operaciones fundamentales son los recorridos.

Preorden: En este enfoque, se visita el nodo actual(raíz) antes de recorrer los subárboles izquierdo y derecho. Es favorable para copiar o clonar árboles.

Inorden: Se visita el primer subárbol izquierdo, luego el nodo actual y finalmente se recorre el subárbol derecho. Este método se usa frecuentemente para la obtención de listas ordenada de los elementos del árbol.

Postorden: Se visitan los subárboles izquierdo y derecho antes de llegar al nodo actual(raíz). Es útil para liberar memoria al eliminar un árbol.

Existen distintas variantes, o tipos, de árboles binarios que vale la pena analizar para comprender mejor cómo es su estructura. Un árbol binario balanceado tiene como máximo una diferencia entre alturas de sus subárboles izquierdo y derecho, para cada nodo del árbol. Un árbol binario completo contiene todos sus niveles llenos de nodos, excepto el último, que también puede estar lleno o llenarse de izquierda a derecha.
Un arbole binario completo es un tipo de árbol donde cada nodo tiene entre 0 y 2 nodos.
Un árbol binario perfecto tiene todos sus nodos hoja en un mismo nivel, significa que todos los niveles están llenos de nodos y todos los nodos internos tienen dos hijos.
Cuando decimos nivel nos referimos a cada generación dentro de los árboles. Por ejemplo, cuando a un nodo hoja se le agrega un hijo, ese nodo pasa a ser un nodo rama por consecuencia el árbol crece una generación por lo que ahora tiene un nivel más. Cada generación tiene un número de nivel distinto que las demás generaciones.
Un árbol vacío tiene 0 niveles
El nivel de la raíz es 1
El nivel de cada nodo se calcula contando la cantidad de nodos existentes desde la raíz hasta el nodo buscado.
La altura es el cálculo mediante la recursividad tomamos el nivel más grande de los dos subárboles de forma recursiva.
Conocemos como peso a el número de nodos que puede llegar a tener un árbol. Este factor es importante por que nos muestra una idea del tamaño del árbol y el tamaño de memoria que nos puede ocupar en tiempo d ejecución

5. Observaciones y comentarios

El programa implementa un árbol binario utilizando estructuras de C++ y los principios de la programación orientada a objetos. El código se dividido en secciones. Los nodos se crean con la función new, asegurando que cada inserción sea independiente y dinámica. El menú permite interactuar de forma sencilla con el usuario, facilitando su aprendizaje de estructura de árboles. Se incluye verificación para evitar errores de duplicidad.
El código esta bien organizado, presentando una estructura clara, separando las funciones por su propósito: Creación de nodos, inserción, búsqueda, recorridos y verificación del árbol. La clase Arbol cumple un rol importante al agrupar las operaciones en el árbol dentro de un solo objeto.

6. Conclusiones

1. La programación de árboles binarios representan herramientas fundamentales para la comprensión de organizaciones jerárquicas de datos y operaciones que se puedan realizar sobre ellos, como inserción, búsqueda, y recorrido.
2. La implementación orientada a objetos permite separar la lógica del manejo de nodos dentro de clases, promoviendo la reutilización del código y diseños más limpios.
3. Los recorridos demostraron ser un recurso esencial para recorrer y procesar datos de los árboles desde diferentes perspectivas.
4. El programa logra cumplir objetivos planteados, proporcionando interfaz clara, funcionalidad para la visualización del funcionamiento interno.

5. Usos de validaciones y estructuras bien definidas mejoran la robustez y confiabilidad del sistema, evitando la propagación de errores comunes como lo es la duplicidad de nodos o inserciones invalidas.

7. Bibliografía

Lopez, J. (27 de Octubre de 2023). *openwebinars.net*. Obtenido de OpenWebinars: https://openwebinars.net/blog/introduccion-a-poo-en-java-objetos-y-clases/

Manuel, J. (26 de Julio de 2021). *campusmvp.es*. Obtenido de Campusmvp: https://l1nq.com/4DP1R

webdesincusco. (29 de Octubre de 2025). Obtenido de Metodos y constructores en la programacion: https://webdesigncusco.com/metodos-y-constructores-en-la-programacion-orientada-a-objetos-poo/

CÓDIGO

J.B. Escobar Garcia
7690-25-1502 Universidad Mariano Gálvez
Matemática Discreta
jescobarg20@miumg.edu.com

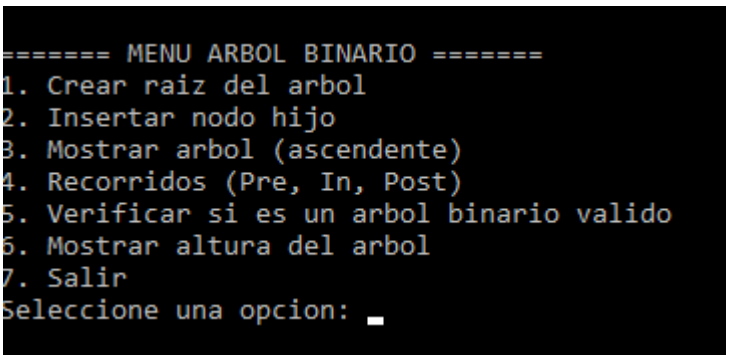
```
#include <iostream>
using namespace std;

// ---------- Estructura del Nodo ----------
struct Nodo {
   int dato;
   Nodo *izq;
   Nodo *der;
};

// ---------- Crear un nuevo nodo ----------
Nodo *crearNodoNuevo(int valor) {
   Nodo *n = new Nodo();
   n->dato = valor;
   n->izq = NULL;
   n->der = NULL;
   return n;
}

// ---------- Insertar nodo automáticamente ----------
void insertarAutomatico(Nodo *&raiz, int valor) {
   if (raiz == NULL) {
      raiz = crearNodoNuevo(valor);
      return;
   }

   if (valor == raiz->dato) {
      cout << "El valor ya existe en el arbol. No se agregara.\n";
      return;
   }

   if (valor < raiz->dato)
      insertarAutomatico(raiz->izq, valor);
   else
      insertarAutomatico(raiz->der, valor);
}

//Recorridos
```

```
void preorden(Nodo *nodo) {
    if (nodo == NULL) return;
    cout << nodo->dato << " ";
    preorden(nodo->izq);
    preorden(nodo->der);
}

void inorden(Nodo *nodo) {
    if (nodo == NULL) return;
    inorden(nodo->izq);
    cout << nodo->dato << " ";
    inorden(nodo->der);
}

void postorden(Nodo *nodo) {
    if (nodo == NULL) return;
    postorden(nodo->izq);
    postorden(nodo->der);
    cout << nodo->dato << " ";
}

//Calcular la altura
int altura(Nodo *raiz) {
    if (raiz == NULL)
        return 0;
    int izq = altura(raiz->izq);
    int der = altura(raiz->der);
    return (izq > der ? izq : der) + 1;
}

// Imprimir arbol de forma visual ascendente
void mostrarArbol(Nodo *raiz, int nivel = 0) {
    if (raiz == NULL)
        return;

    // Primero el subarbol derecho (valores mayores)
    mostrarArbol(raiz->der, nivel + 1);

    // Imprime el nodo con sangria para representar el nivel
    for (int i = 0; i < nivel; i++)
        cout << "    ";
    cout << raiz->dato << endl;
```

```
    // Luego el subarbol izquierdo (valores menores)
    mostrarArbol(raiz->izq, nivel + 1);
}

//Verificar si el árbol es binario válido
bool esBinarioValido(Nodo* raiz, int min, int max) {
    if (raiz == NULL)
        return true;
    if (raiz->dato <= min || raiz->dato >= max)
        return false;
    return esBinarioValido(raiz->izq, min, raiz->dato) &&
           esBinarioValido(raiz->der, raiz->dato, max);
}

//Menu principal
int main() {
    Nodo *raiz = NULL;
    int opcion, valor;

    do {
        cout << "\n======= MENU ARBOL BINARIO =======\n";
        cout << "1. Crear raiz del arbol\n";
        cout << "2. Insertar nodo hijo\n";
        cout << "3. Mostrar arbol (ascendente)\n";
        cout << "4. Recorridos (Pre, In, Post)\n";
        cout << "5. Verificar si es un arbol binario valido\n";
        cout << "6. Mostrar altura del arbol\n";
        cout << "7. Salir\n";
        cout << "Seleccione una opcion: ";
        cin >> opcion;

        switch (opcion) {
            case 1:
                if (raiz != NULL)
                    cout << "La raiz ya fue creada.\n";
                else {
                    cout << "Ingrese el valor de la raiz: ";
                    cin >> valor;
                    raiz = crearNodoNuevo(valor);
                    cout << "Raiz creada correctamente.\n";
                }
                break;
```

```
case 2:
   if (raiz == NULL)
      cout << "Debe crear la raiz primero.\n";
   else {
      cout << "Ingrese el valor del nuevo nodo: ";
      cin >> valor;
      insertarAutomatico(raiz, valor);
      cout << "Nodo insertado correctamente.\n";
   }
   break;

case 3:
   if (raiz == NULL)
      cout << "El arbol esta vacio.\n";
   else {
      cout << "\nEstructura del arbol (ascendente):\n";
      mostrarArbol(raiz);
   }
   break;

case 4:
   if (raiz == NULL)
      cout << "El árbol esta vacio.\n";
   else {
      cout << "\nPreorden: ";
      preorden(raiz);
      cout << "\nInorden: ";
      inorden(raiz);
      cout << "\nPostorden: ";
      postorden(raiz);
      cout << endl;
   }
   break;

case 5:
   if (raiz == NULL)
      cout << "El arbol esta vacio.\n";
   else {
      bool valido = esBinarioValido(raiz, -999999, 999999);
      if (valido)
         cout << "? El arbol es binario de busqueda valido.\n";
      else
         cout << "? El arbol NO cumple las propiedades de un ABB.\n";
```

```
            }
            break;

        case 6:
            cout << "La altura del arbol es: " << altura(raiz) << endl;
            break;

        case 7:
            cout << "Saliendo del programa...\n";
            break;

        default:
            cout << "Opcion invalida, intente de nuevo.\n";
      }

   } while (opcion != 7);

   return 0;
}
```



```
======= MENU ARBOL BINARIO =======
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 1
Ingrese el valor de la raiz: 15
Raiz creada correctamente.

======= MENU ARBOL BINARIO =======
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion:
```

```
======= MENU ARBOL BINARIO =======
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 2
Ingrese el valor del nuevo nodo: 20
Nodo insertado correctamente.
```

```
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 2
Ingrese el valor del nuevo nodo: 10
Nodo insertado correctamente.

======= MENU ARBOL BINARIO =======
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 2
Ingrese el valor del nuevo nodo: 25
Nodo insertado correctamente.

======= MENU ARBOL BINARIO =======
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 2
Ingrese el valor del nuevo nodo: 11
Nodo insertado correctamente.
```

```
======= MENU ARBOL BINARIO =======
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 3

Estructura del arbol (ascendente):
      25
   20
15
      11
   10
```

```
======= MENU ARBOL BINARIO =======
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 4

Preorden: 15 10 11 20 25
Inorden: 10 11 15 20 25
Postorden: 11 10 25 20 15
```

```
======= MENU ARBOL BINARIO =======
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 5
? El arbol es binario de busqueda valido.
```

```
======= MENU ARBOL BINARIO =======
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 6
La altura del arbol es: 3
```

```
======= MENU ARBOL BINARIO =======
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 7
Saliendo del programa...

--------------------------------
Process exited after 644.6 seconds with return value 0
Presione una tecla para continuar . . .
```