

Instructor: Ernesto Enrique García Ramos

Contacto: egarcia97.r@gmail.com

gr15i04001@usonsonate.edu.sv

## Guía 8: Implementación de código Árbol Binario

### Objetivos:

- Implementar conocimientos de árboles binarios

En la siguiente guía aplicaremos los conocimientos teóricos de árboles, codificándolos e implementándolos de manera dinámica en la inserción de variables primitivas (int).

### Clases

#### Nodo.h

```
class Nodo{
    private:
        int dato;
        Nodo *izquierdo;
        Nodo *derecho;
    public:
        Nodo(int v, Nodo *izq=NULL, Nodo *der=NULL) {
            this->dato=v;
            this->izquierdo=izq;
            this->derecho=der;
        }
        friend class ArbolBinario;
};typedef Nodo *pnodo;
```

#### ArbolBinario.h

```
#include "Nodo.h"
class ArbolBinario{
    private:
        pnodo raiz;
        pnodo actual;
        int contador;
        int altura;
        void Podar(pnodo nodo){
            if(nodo) {
                Podar(nodo->izquierdo); // Podar izquierdo
                Podar(nodo->derecho);    // Podar derecho
                delete nodo;              // Eliminar nodo
                nodo = NULL;
            }
        }
        void auxContador(pnodo nodo){
            contador++; // Otro nodo
            // Continuar recorrido
            if(nodo->izquierdo) auxContador(nodo->izquierdo);
            if(nodo->derecho)   auxContador(nodo->derecho);
        }
    public:
        void crearArbolBinario();
        void imprimirArbolBinario();
        void eliminarArbolBinario();
        void alturaArbolBinario();
        void contarArbolBinario();
};
```

# Programación 3

Facultad de Ingeniería y Ciencias Naturales

```
    }
    void auxAltura(pnodo nodo, int a){
        // Recorrido postorden
        if(nodo->izquierdo){ auxAltura(nodo->izquierdo, a+1); }
        if(nodo->derecho){ auxAltura(nodo->derecho, a+1); }
        // Proceso, si es un nodo hoja, y su altura es mayor que
la actual del
        // árbol, actualizamos la altura actual del árbol
        if(EsHoja(nodo) && a > this->altura)
            {this-> altura = a;}
    }
public:
    ArbolBinario(){
        this->raiz=NULL;
        this->actual=NULL;
        this->contador=0;
    }
    ~ArbolBinario(){
        Podar(this->raiz);
        this->raiz=NULL;
    }
    pnodo getRaiz(){
        return this->raiz;
    }
    void Insertar(int dat){
        pnodo padre = NULL;

        this->actual =this-> raiz;
        // Buscar el int en el árbol, manteniendo un puntero al
nodo padre
        while(!Vacio(this->actual) && dat != this->actual->dato)
        {
            padre = this->actual;
            if(dat > this->actual->dato){
                this-> actual =this-> actual->derecho;
            }
            else{
                if(dat <this->actual->dato){
                    actual = actual->izquierdo;
                }
            }
        }

        // Si se ha encontrado el elemento, regresar sin insertar
        if(!Vacio(this->actual)){
            return;
        }
        // Si padre es NULL, entonces el árbol estaba vacío, el
nuevo nodo será
        // el nodo raiz
        if(Vacio(padre)){
            this->raiz = new Nodo(dat);}
        // Si el int es menor que el que contiene el nodo padre,
lo insertamos
        // en la rama izquierda
        else{
            if(dat < padre->dato){
                padre->izquierdo = new Nodo(dat); }
            else{
```

# Programación 3

Facultad de Ingeniería y Ciencias Naturales

```
        if(dat > padre->dato){
            padre->derecho = new Nodo(dat);
        }
    }

    // Si el int es mayor que el que contiene el nodo padre,
    lo insertamos
    // en la rama derecha

}

void Borrar(int dat){
    pnode padre = NULL;
    pnode nodo;
    int aux;

    this->actual = this->raiz;
    // Mientras sea posible que el valor esté en el árbol
    while(!Vacio(this->actual)) {
        if(dat ==this->actual->dato) { // Si el valor está en
el nodo actual
            if(EsHoja(this->actual)) { // Y si además es un
nodo hoja: lo borramos
                if(padre) // Si tiene padre (no es el nodo raiz)
                // Anulamos el puntero que le hace referencia
                if(padre->derecho ==this-> actual) padre-
>derecho = NULL;
                else if(padre->izquierdo == actual) padre-
>izquierdo = NULL;
                delete actual; // Borrar el nodo
                actual = NULL;
                return;
            }
            else { // Si el valor está en el nodo actual, pero
no es hoja
                // Buscar nodo
                padre = this->actual;
                // Buscar nodo más izquierdo de rama derecha
                if(this->actual->derecho) {
                    nodo = this->actual->derecho;
                    while(nodo->izquierdo) {
                        padre = nodo;
                        nodo = nodo->izquierdo;
                    }
                }
                // O buscar nodo más derecho de rama izquierda
                else {
                    nodo =this-> actual->izquierdo;
                    while(nodo->derecho) {
                        padre = nodo;
                        nodo = nodo->derecho;
                    }
                }
                // Intercambiar valores de no a borrar u nodo
encontrado
                // y continuar, cerrando el bucle. El nodo
encontrado no tiene
                // por qué ser un nodo hoja, cerrando el bucle
nos aseguramos
                // de que sólo se eliminan nodos hoja.
            }
        }
    }
}
```

# Programación 3

Facultad de Ingeniería y Ciencias Naturales

```
        aux = this->actual->dato;
        this->actual->dato = nodo->dato;
        nodo->dato = aux;
        this->actual = nodo;
    }
}
else { // Todavía no hemos encontrado el valor, seguir
buscándolo
    padre =this-> actual;
    if(dat >this-> actual->dato) this-> actual =this->
actual->derecho;
    else if(dat < this->actual->dato) this->actual =
this->actual->izquierdo;
    }
}
}
bool Buscar(int dat){
    this->actual = raiz;

    // Todavía puede aparecer, ya que quedan nodos por mirar
    while(!Vacio(this->actual)) {
        if(dat ==this-> actual->dato) return true; // int
encontrado
        else if(dat >this-> actual->dato) this->actual = this-
>actual->derecho; // Seguir
        else if(dat < this->actual->dato) this->actual = this-
>actual->izquierdo;
    }
    return false; // No está en árbol
}
bool Vacio(pnodo r) {
    return r==NULL;
}
bool EsHoja(pnodo r) {
    return !r->derecho && !r->izquierdo;
}
int NumeroNodos(){
    this->contador = 0;
    auxContador(raiz); // FUnción auxiliar
    return contador;
}
int AlturaArbol(){
    this->altura = 0;
    auxAltura(raiz, 0); // Función auxiliar
    return altura;
}
int Altura( int dat){
    int altura = 0;
    this->actual = this->raiz;

    // Todavía puede aparecer, ya que quedan nodos por mirar
    while(!Vacio(this->actual)) {
        if(dat == this->actual->dato) return altura; // int encontrado
        else {
            altura++; // Incrementamos la altura, seguimos buscando
            if(dat >this-> actual->dato) this-> actual =this-> actual-
>derecho;
            else if(dat < actual->dato) this->actual =this-> actual-
>izquierdo;
```

```
    }  
}  
return -1; // No está en árbol  
}  
int &ValorActual() {  
    return actual->dato;  
}  
void Raiz() {  
    this->actual = this->raiz;  
}  
  
void Mostrar(int &d)  
{  
    cout << d << ", ";  
}  
};
```

## Ejercicio

1. Crear métodos posorden, preorden e inorden
2. Hacer dinámica la inserción, eliminación, ordenamiento de datos

Nota: Si considera hacer una modificación al código proporcionado es libre de hacerlo, anexar el motivo y comentar adecuadamente.