



**Asignatura:** Algoritmia y Estructura de Datos

**Profesor:** D. Sc. Gerardo García Gil

2021-B

**Alumno:** José Rafael Ruiz Gudiño

**Ingeniería en Desarrollo de Software**

**Centro de Enseñanza Técnica Industrial (CETI)**

## **-Presentación**

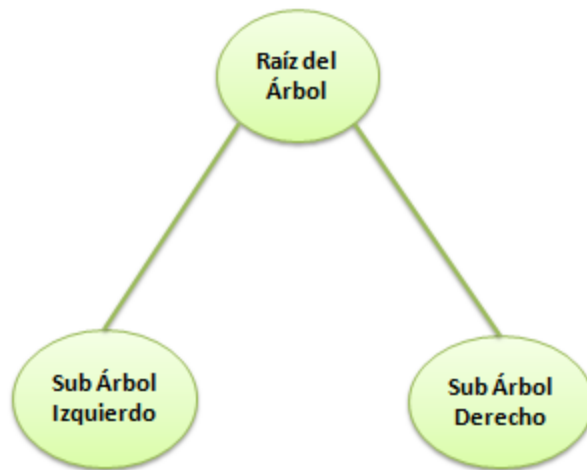
Los árboles son una de las estructuras de datos no lineales, empleadas en informática, tanto para resolver problemas de hardware como de software. Los árboles de directorios son organizaciones bastante empleadas por cualquier usuario o programador de una computadora. De igual manera cumplen un buen papel en la toma de decisiones, valido como árbol de decisiones.

Los árboles genealógicos y los organigramas son ejemplos comunes. Entre otras aplicaciones, los árboles se emplean para analizar circuitos eléctricos y para representar la estructura de fórmulas matemáticas, así como para organizar la información de bases de datos, para representar la estructura sintáctica de un programa fuente en compiladores y para la toma de decisiones.

## **-Introducción**

Los árboles binarios son estructuras de datos muy similares a las listas doblemente enlazadas, en el sentido que tienen dos punteros que apuntan a otros elementos, pero no tienen una estructura lógica de tipo lineal o secuencial como aquellas, sino ramificada. Tienen aspecto de árbol, de ahí su nombre.

Un árbol binario es una estructura de datos no lineal en la que cada nodo puede apuntar a uno o máximo a dos nodos. También se suele dar una definición recursiva que indica que es una estructura compuesta por un dato y dos árboles. Esto son definiciones simples. Este tipo de árbol se caracteriza porque tienen un vértice principal y de él se desprende dos ramas. La rama izquierda y la rama derecha a las que también se les conoce como subárboles.



## -Desarrollo

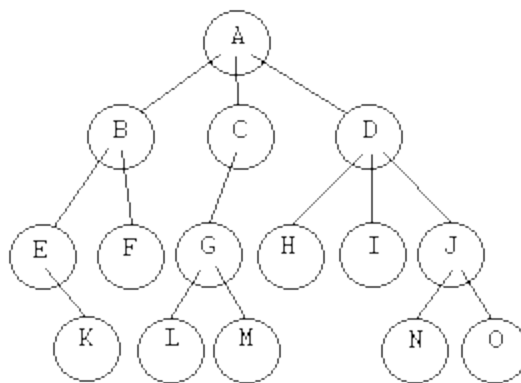
Un árbol es una estructura no lineal en la que cada nodo puede apuntar a uno o varios nodos.

También se suele dar una definición recursiva: un árbol es una estructura en compuesta por un dato y varios árboles.

Esto son definiciones simples. Pero las características que implican no lo son tanto.

Definiremos varios conceptos. En relación con otros nodos:

- **Nodo hijo:** cualquiera de los nodos apuntados por uno de los nodos del árbol. En el ejemplo, 'L' y 'M' son hijos de 'G'.
- **Nodo padre:** nodo que contiene un puntero al nodo actual. En el ejemplo, el nodo 'A' es padre de 'B', 'C' y 'D'.



Los árboles tienen otra característica importante: cada nodo sólo puede ser apuntado por otro nodo, es decir, cada nodo sólo tendrá un padre. Esto hace que estos árboles estén fuertemente jerarquizados, y es lo que en realidad les da la apariencia de árboles.

En cuanto a la posición dentro del árbol:

- **Nodo raíz:** nodo que no tiene padre. Este es el nodo que usaremos para referirnos al árbol. En el ejemplo, ese nodo es el 'A'.
- **Nodo hoja:** nodo que no tiene hijos. En el ejemplo hay varios: 'F', 'H', 'I', 'K', 'L', 'M', 'N' y 'O'.
- **Nodo rama:** aunque esta definición apenas la usaremos, estos son los nodos que no pertenecen a ninguna de las dos categorías anteriores. En el ejemplo: 'B', 'C', 'D', 'E', 'G' y 'J'.

Otra característica que normalmente tendrán los árboles es que todos los nodos contengan el mismo número de punteros, es decir, usaremos la misma estructura para todos los nodos del árbol. Esto hace que la estructura sea más sencilla, y por lo tanto también los programas para trabajar con ellos.

Tampoco es necesario que todos los nodos hijos de un nodo concreto existan. Es decir, que pueden usarse todos, algunos o ninguno de los punteros de cada nodo.

Un árbol en el que en cada nodo o bien todos o ninguno de los hijos existe, se llama **árbol completo**.

En una cosa, los árboles se parecen al resto de las estructuras que hemos visto: dado un nodo cualquiera de la estructura, podemos considerarlo como una estructura independiente. Es decir, un nodo cualquiera puede ser considerado como la raíz de un árbol completo.

Existen otros conceptos que definen las características del árbol, en relación a su tamaño:

- **Orden:** es el número potencial de hijos que puede tener cada elemento de árbol. De este modo, diremos que un árbol en el que cada nodo puede apuntar a otros dos es de *orden dos*, si puede apuntar a tres será de *orden tres*, etc.
- **Grado:** el número de hijos que tiene el elemento con más hijos dentro del árbol. En el árbol del ejemplo, el grado es tres, ya que tanto 'A' como 'D' tienen tres hijos, y no existen elementos con más de tres hijos.
- **Nivel:** se define para cada elemento del árbol como la distancia a la raíz, medida en nodos. El nivel de la raíz es cero y el de sus hijos uno. Así sucesivamente. En el ejemplo, el nodo 'D' tiene nivel 1, el nodo 'G' tiene nivel 2, y el nodo 'N', nivel 3.
- **Altura:** la altura de un árbol se define como el nivel del nodo de mayor nivel. Como cada nodo de un árbol puede considerarse a su vez como la raíz de un árbol, también podemos hablar de altura de ramas. El árbol del ejemplo tiene altura 3, la rama 'B' tiene altura 2, la rama 'G' tiene altura 1, la 'H' cero, etc.

Los árboles de orden dos son bastante especiales, de hecho les dedicaremos varios capítulos. Estos árboles se conocen también como **árboles binarios**.

## -Implementación

### Operaciones en Árboles binarios

El repertorio de operaciones que se pueden realizar sobre un ABB es parecido al que realizábamos sobre otras estructuras de datos, más alguna otra propia de árboles:

- Buscar un elemento.
- Insertar un elemento.
- Borrar un elemento.
- Movimientos a través del árbol:
  - Izquierda.
  - Derecha.
  - Raiz.
- Información:
  - Comprobar si un árbol está vacío.
  - Calcular el número de nodos.
  - Comprobar si el nodo es hoja.
  - Calcular la altura de un nodo.
  - Calcular la altura de un árbol.

### **Código de arbol con metodo borrar y buscar en C**

```
#include <stdio.h>
#include <stdlib.h>
struct Nodo{
    int dato;
    struct Nodo *izq;
    struct Nodo *der;
};
////////////////////////creacion de un nodo////////////////////////
//    size_t es un tipo de dato sin signo , que es nmayor a 0
struct Nodo *nuevonodo(int dato);
void insertar(struct Nodo *nodo,int dato);
void preorden(struct Nodo *nodo);
void inorden(struct Nodo *nodo);
void postorden(struct Nodo *nodo);
void Buscar(struct Nodo* a, int dat);
void Borrar(struct Nodo* arbol, int dat);
int Vacio(struct Nodo* r);
int EsHoja(struct Nodo* r);
int main(int argc, char *argv[]) {
    struct Nodo *raiz=nuevonodo(28);
    insertar(raiz, 11);
    insertar(raiz, 96);
    insertar(raiz, 21);
    insertar(raiz, 6);
```

```

        insertar(raiz, 97);
        insertar(raiz, 1);
        insertar(raiz, 30);
insertar(raiz, 10);
insertar(raiz, 2);

printf("\n Preorden\n");
preorden(raiz);

printf("\n Inorden\n");
inorden(raiz);

printf("\n Postorden\n");
postorden(raiz);
Borrar(raiz,11);
Borrar(raiz,96);
printf("\n Postorden\n");
postorden(raiz);
Buscar(raiz,30);
Buscar(raiz,96);
        return 0;
}
struct Nodo *nuevonodo(int dato){
    //se pide el espacio en memoria para el nuevo nodo
    size_t tamanodo=sizeof(struct Nodo);
    struct Nodo *nodo=(struct Nodo *)malloc(tamanodo);
        // Asignar el dato al iniciar
    nodo->dato=dato;
    nodo->izq=nodo->der=NULL;

    return nodo;
}
void insertar(struct Nodo *nodo,int dato){

    //si es mayor va a la derecha
    if(dato>nodo->dato){
        if(nodo->der==NULL){
            nodo->der=nuevonodo(dato);
        }
        else{
            insertar(nodo->der,dato);
        }
    }
    else{
        // sino a la izquierda
        if(nodo->izq==NULL){
            nodo->izq=nuevonodo(dato);
        }
        else{
            insertar(nodo->izq,dato);
        }
    }
}

```

```

void preorden(struct Nodo *nodo){
    if(nodo!=NULL){
        printf("%d ",nodo->dato);
        preorden(nodo->izq);
        preorden(nodo->der);
    }
}

void inorden(struct Nodo *nodo){
    if(nodo!=NULL){
        inorden(nodo->izq);
        printf("%d ",nodo->dato);
        inorden(nodo->der);
    }
}

void postorden(struct Nodo *nodo){
    if(nodo!=NULL){
        postorden(nodo->izq);
        postorden(nodo->der);
        printf("%d ",nodo->dato);
    }
}

void Borrar(struct Nodo *arbol, int dat) {
    struct Nodo* padre = NULL; /* (1) */
    struct Nodo* actual;
    struct Nodo* nodo;
    int aux;
    actual = arbol;
    while(!Vacio(actual)) { /* Búsqueda else implícito */
        if(dat == actual->dato) {
            if(EsHoja(actual)) {
                if(padre)
                    if(padre->der == actual) padre->der = NULL;
                    else if(padre->izq == actual) padre->izq = NULL;
                free(actual);
                actual = NULL;
                printf("\nNodo %d eliminado\n",dat);
                return;
            }
            else {
                /* Buscar nodo */
                padre = actual;
                if(actual->der) {
                    nodo = actual->der;
                    while(nodo->izq) {
                        padre = nodo;
                        nodo = nodo->izq;
                    }
                }
                else {
                    nodo = actual->izq;
                    while(nodo->der) {
                        padre = nodo;

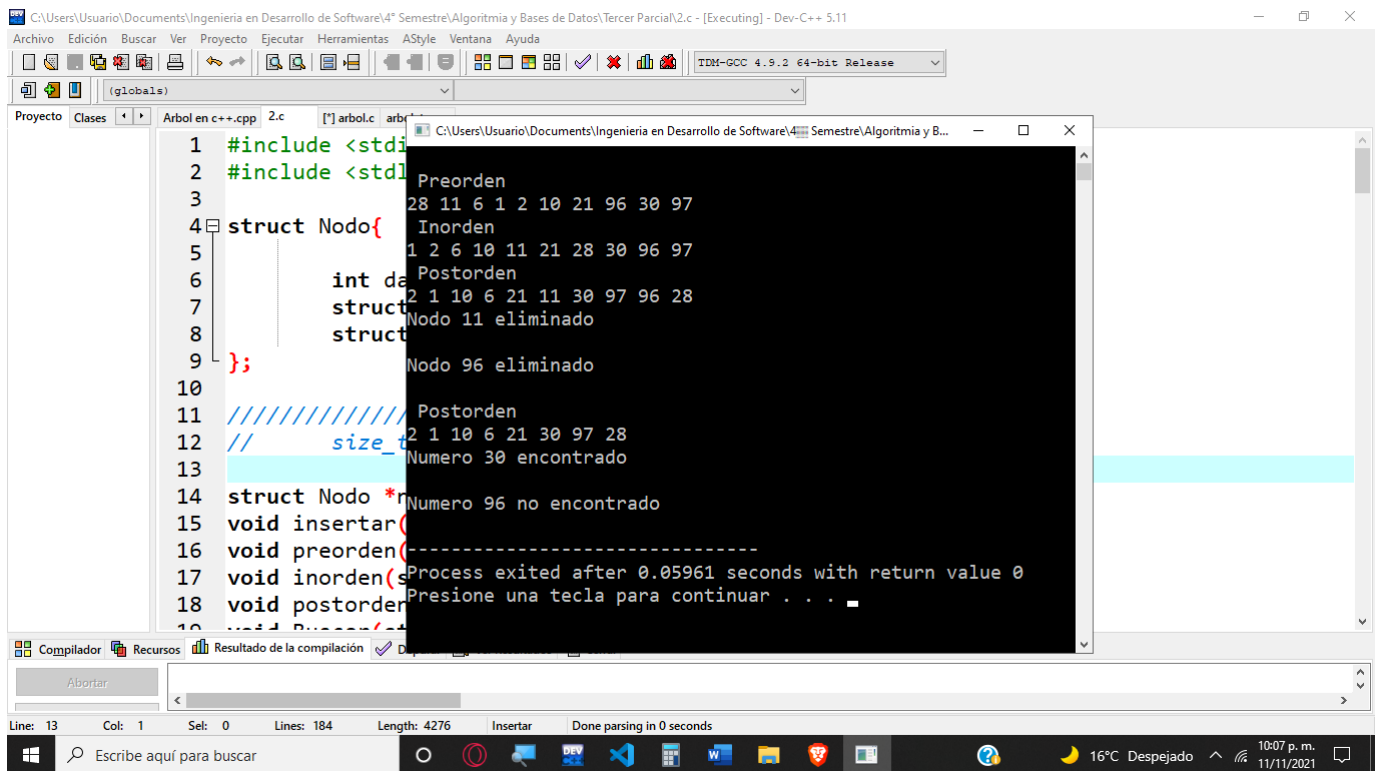
```

```

        nodo = nodo->der;
    }
}
/* Intercambio */
aux = actual->dato;
actual->dato = nodo->dato;
nodo->dato = aux;
actual = nodo;
}
}
else {
    padre = actual;
    if(dat > actual->dato) actual = actual->der;
    else if(dat < actual->dato) actual = actual->izq;
}
}}}
void Buscar(struct Nodo* a, int dat) {
    struct Nodo* actual = a;
    while(!Vacio(actual)) {
        if(dat == actual->dato){
            printf("\nNumero %d encontrado\n",dat);
            return; /* dato encontrado */
        }else if(dat < actual->dato) actual = actual->izq;
        else if(dat > actual->dato) actual = actual->der;
    }
    printf("\nNumero %d no encontrado\n",dat); /* No está en árbol*/
}
int Vacio(struct Nodo* r) {
    return r==NULL;
}
int EsHoja(struct Nodo* r) {
    return !r->der && !r->izq;
}

```

## **-Resultados**



```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Nodo{
5     int dato;
6     struct Nodo *hijoIzq;
7     struct Nodo *hijoDer;
8 };
9
10 // Preorden
11 void preorden(struct Nodo *p){
12     if(p != NULL){
13         printf("%d ", p->dato);
14         preorden(p->hijoIzq);
15         preorden(p->hijoDer);
16     }
17 }
18
19 // Inorden
20 void inorden(struct Nodo *p){
21     if(p != NULL){
22         inorden(p->hijoIzq);
23         printf("%d ", p->dato);
24         inorden(p->hijoDer);
25     }
26 }
27
28 // Postorden
29 void postorden(struct Nodo *p){
30     if(p != NULL){
31         postorden(p->hijoIzq);
32         postorden(p->hijoDer);
33         printf("%d ", p->dato);
34     }
35 }
36
37 // Borrar
38 void borrar(struct Nodo *p, int dato){
39     if(p == NULL) return;
40     if(p->dato == dato){
41         printf("Nodo %d eliminado\n", dato);
42         if(p->hijoIzq != NULL){
43             borrar(p->hijoIzq, dato);
44         }
45         if(p->hijoDer != NULL){
46             borrar(p->hijoDer, dato);
47         }
48         free(p);
49     }
50 }
51
52 // Buscar
53 struct Nodo *buscar(struct Nodo *p, int dato){
54     if(p == NULL) return NULL;
55     if(p->dato == dato) return p;
56     if(p->hijoIzq != NULL){
57         struct Nodo *resultado = buscar(p->hijoIzq, dato);
58         if(resultado != NULL) return resultado;
59     }
60     if(p->hijoDer != NULL){
61         struct Nodo *resultado = buscar(p->hijoDer, dato);
62         if(resultado != NULL) return resultado;
63     }
64     return NULL;
65 }
66
67 // Main
68 int main(){
69     struct Nodo *root = NULL;
70     // Insertar
71     insertar(&root, 28);
72     insertar(&root, 11);
73     insertar(&root, 6);
74     insertar(&root, 1);
75     insertar(&root, 10);
76     insertar(&root, 21);
77     insertar(&root, 96);
78     insertar(&root, 30);
79     insertar(&root, 97);
80
81     // Preorden
82     printf("Preorden\n");
83     preorden(root);
84
85     // Inorden
86     printf("Inorden\n");
87     inorden(root);
88
89     // Postorden
90     printf("Postorden\n");
91     postorden(root);
92
93     // Borrar
94     borrar(root, 11);
95     borrar(root, 96);
96
97     // Buscar
98     struct Nodo *nodo30 = buscar(root, 30);
99     if(nodo30 != NULL){
100         printf("Numero 30 encontrado\n");
101     }
102     struct Nodo *nodo96 = buscar(root, 96);
103     if(nodo96 == NULL){
104         printf("Numero 96 no encontrado\n");
105     }
106
107     return 0;
108 }
```

Preorden  
28 11 6 1 2 10 21 96 30 97  
Inorden  
1 2 6 10 11 21 28 30 96 97  
Postorden  
2 1 10 6 21 11 30 97 96 28  
Nodo 11 eliminado  
Nodo 96 eliminado  
Postorden  
2 1 10 6 21 30 97 28  
Numero 30 encontrado  
Numero 96 no encontrado  
Process exited after 0.05961 seconds with return value 0  
Presione una tecla para continuar . . .

En la captura en la primera parte se ordenan los nodos del arbol que se ingresaron en el mismo código de modo preorden, después en inorden y al último en postorden, enseguida se realiza la eliminación de dos nodos con el método borrar que en este caso fue el nodo 11 y 96. Posteriormente se despliega el árbol nuevamente de modo postorden para así verificar que se eliminaron los nodos exitosamente. Luego se realiza el método de buscar con dos nodos: 30 y 96. Como podemos observar el nodo 30 lo encuentra satisfactoriamente, sin embargo, el nodo 96 fue imposible de buscar debido a que no existe más, a causa del uso del método borrar.

## -Conclusión

Esta práctica fue verdaderamente un aumento de dificultad notorio debido a que se debe comprender primeramente como es la estructura de un árbol, los tipos que hay y las maneras de ordenarlo ya sea preorden, postorden o inorden, según sea el caso. Me fue algo sencillo comprender el como se insertan los nodos en un árbol debido a que me pareció un poco similar a las listas enlazadas. Pero lo que fue realmente complicado de entender fue el método de borrar. Esto primeramente porque son varios casos que se presentan como lo es cuando se tiene un hijo, dos hijos o ningún hijo que sería un nodo hoja. Aunque debo mencionar que a veces me sentía un poco perdido debido a que se hace mucho uso de la recursividad en estos métodos. Sin embargo después de estar un rato leyendo código lo pude entender bien y pude comprender el funcionamiento del programa.

## -Referencias

- Árboles Binarios. (2014, 10 junio). Estructuras de datos. Recuperado 11 de noviembre de 2021, de <https://hhmosquera.wordpress.com/arbolesbinarios/>
- Pozo, S. (2002a, abril 1). *Estructuras de datos (cap6)*. C con Clase. Recuperado 11 de noviembre de 2021, de <http://conclase.net/c/edd/cap6>



Pozo, S. (2002b, abril 1). *Estructuras de datos (cap7)*. C con Clase. Recuperado 11 de noviembre de 2021, de <http://conclase.net/c/edd/cap7>