**INDICE**

Introducción ………………………………………………………………………………………………….. 03

Objetivos ………………………………………………………………………………………………………. 04

Definición Simple …………………………………………………………………………………………… 05

Terminología Utilizada en Arboles …………………………………………………………………. 05

Profundidad de un Árbol ……………………………………………………………………………….. 05

Ordenes en árboles: ………………………………………………………………………………………. 06

Tipos de Arboles Según su Orden ………………………………………………………………….. 06

Tipos de Arboles Según su Contenido ……………………………………………………………. 06

Operaciones con Arboles ………………………………………………………………………………. 06

Recorrer un Árbol ………………………………………………………………………………………….. 06

Definición de un Árbol en C …………………………………………………………………………… 07

Código de un Árbol en C …………………………………………………………………………. 07 - 11

Conclusión …………………………………………………………………………………………………….. 12

Recomendaciones ………………………………………………………………………………………… 13

Bibliografía ……………………………………………………………………………………………………. 14

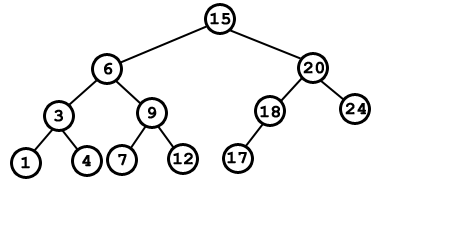
**Introducción**

**Objetivos**

**Estructura de datos Arboles**

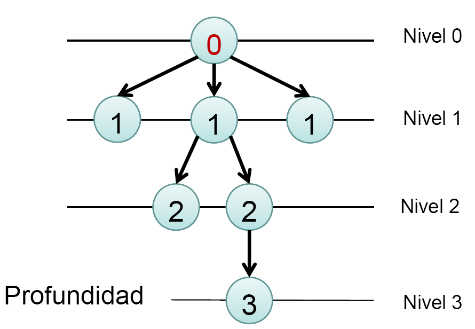
**Definición simple:**

La estructura de datos Árbol es una estructura de dato no lineal, compuesto por nodos los cuales hacen referencia de forma recursiva a otros nodos, formando una estructura gráfica, muy parecida a la de un árbol, de ahí su nombre.

La estructura árbol se puede definir de forma recursiva como una colección de nodos (a partir de un nodo base llamado raíz), donde cada nodo es una estructura de datos con un valor, junto con una lista de referencia a otros nodos (llamados nodos hijos), con la condición de que no pueden ser ciclicos.

**Terminología utilizada en árboles**

* **Raíz:** El nodo superior de un árbol.
* **Hijo:** Un nodo conectado directamente con otro cuando se aleja de la raíz.
* **Padre:** La noción inversa de *hijo*.
* **Hermanos:** Un conjunto de nodos con el mismo padre.
* **Descendiente:** Un nodo accesible por descenso repetido de padre a hijo.
* **Ancestro:** Un nodo accesible por ascenso repetido de hijo a padre.
* **Hoja:** Un nodo sin hijos.
* **Nodo interno:** Un nodo con al menos un hijo.
* **Grado:** Número de subárboles de un nodo.
* **Brazo:** La conexión entre un nodo y otro.
* **Camino:** Una secuencia de nodos y brazos conectados con un nodo descendiente.
* **Nivel:** El nivel de un nodo se define por 1 + (el número de conexiones entre el nodo y la raíz).
* **Altura de un nodo:** La altura de un nodo es el número de aristas en el camino más largo entre ese nodo y una hoja.
* **Altura de un árbol:** La altura de un árbol es la altura de su nodo raíz.
* **Profundidad:** La profundidad de un nodo es el número de aristas desde la raíz del árbol hasta un nodo.
* **Bosque:** Un bosque es un conjunto de árboles n ≥ 0 disjuntos.
* **Rama:** Una ruta del nodo raíz a cualquier otro nodo.

**Profundidad de un árbol:**

Se define la **profundidad de un nodo** en un arbol como la longitud del camino (único) que comienza en la raíz y termina en el nodo. También se denomina nivel.

**Órdenes en árboles:**

El orden se define como cuantos hijos puede tener un nodo, lo ideal es tener un número limitado de hijos para tener una estructura cómoda y de acceso eficiente.

* **Orden 2:** Un nodo puede tener 0, 1 o 2 hijos.
* **Orden 3:** Un nodo puede tener 0, 1, 2 o 3 hijos.
* **Orden 4:** Un nodo puede tener 0, 1, 2, 3 o 4 hijos.

**Tipos de árboles según su orden:**

* Árbol binario: Árbol de orden 2.
* Árbol ternario: Árbol de orden 3.
* Etc.

**Tipos de árboles según su contenido:**

* Árboles llenos: Es un árbol en el cual todos los nodos (a excepción de los nodos hoja) tienen todos sus hijos.
* Árboles completos: Es un árbol en el cual pueden faltar hijos (uno o varios nodos hoja), a condición de que el nodo hoja sea el hijo izquierdo.
* Árboles degenerados: Es un árbol en el cual cada nodo solo tiene un hijo, estos árboles se comportan como listas.
* Árboles balanceados: Es un árbol en el cual se intenta mantener la menor profundidad posible, haciendo algunas operaciones más eficientes.

**Operaciones con árboles:**

* Inserción
* Eliminación
* Localización
* Recorrido

**Recorrer un árbol:**

La operación de recorrer un árbol no es más que leer la información del árbol por medio de sus ramas, existen diferentes formas de leer la información las cuales son:

1. **En anchura**: la información se lee de forma horizontal (en niveles) tomando como inicio el nodo principal del árbol.
2. **En vertical**
   1. **En pre orden:** primero se procesa la raíz, luego se procesa el árbol izquierdo y por último se procesa el árbol derecho.
   2. **En in orden:** primero se procesa el árbol izquierdo, luego se procesa la raíz y por último se procesa el árbol derecho.
   3. **En post orden:** primero se procesa el árbol derecho, luego se procesa el árbol derecho y por último se procesa la raíz.

**Definición de un árbol en C**

Se define el árbol como un struct puesto que el árbol se convertirá en una estructura de dato, sus propiedades serán: un int el cual será el dato que se quiere guardar, tres punteros de los cuales dos serán referencias a sus dos ramas hijas y un puntero más la cual hace referencia a su padre, esto nos ayudará en la acción de eliminar un nodo, esta estructura se crea de esta manera porque será una estructura recursiva.

Cabe mencionar que un árbol debe de tener un elemento comparable en este caso será el mismo dato a guardar en el árbol.

typedef struct nodo {

int valor;

struct nodo \*izdo;

struct nodo \*drcho;

struct nodo \*padre;

} Nodo;

Las siguientes partes de la estructura árbol será explicado por medio del código completo para mayor comprensión

#include <stdio.h>

#include <stdlib.h>

//Definicon de la estructura arbol

typedef struct nodo{

int valor;

struct nodo \*izdo;

struct nodo \*drcho;

struct nodo \*padre;

} Nodo;

//Definicion de nombre alternativo a la estructura

typedef Nodo arbol;

//Funcion para crear un nodo, funcion creada de forma

//recursiva para poder usarse en cualquier parte de el arbol

//nodos hijos igualados a nul y preparados para generar una nueva rama

Nodo \*CrearNodo(int valor, Nodo \*padre){

Nodo \*nuevoNodo = (Nodo \*)malloc(sizeof(Nodo));

nuevoNodo->valor = valor;

nuevoNodo->izdo = nuevoNodo->drcho = NULL;

nuevoNodo->padre = padre;

return nuevoNodo;

}

//Funcion para destruir un nodo, primero vaciando sus hijos y luego

//liberando la memoria que usaba el nodo

void DestruirNodo(Nodo \*nodo){

nodo->izdo = nodo->drcho = NULL;

free(nodo);

}

//Funcion de apoyo creado de forma recursiva un nuevo nodo

//ademas que la funcion evalua en que lugar de el arbol

//puede ser insertado el nuevo nodo y si un nodo provehido esta vacio

static void InsertarConPadre(Nodo \*\*arbol, Nodo \*padre, int valor){

if (\*arbol == NULL){

\*arbol = CrearNodo(valor, padre);

}

else{

int valorRaiz = (\*arbol)->valor;

if (valor < valorRaiz){

InsertarConPadre(&(\*arbol)->izdo, \*arbol, valor);

}else{

InsertarConPadre(&(\*arbol)->drcho, \*arbol, valor);

}

}

}

//Funcion para agregar un nuevo nodo al arbol

void Insertar(Nodo \*\*arbol, int valor){

InsertarConPadre(arbol, NULL, valor);

}

//Funcion para recorrer un arbol de forma optima y recursiva

//y devolviendo una respuesta si existe el elemento solicitado

int Existe(Nodo \*arbol, int valor){

if (arbol == NULL){

return 0;

}else if (arbol->valor == valor){

return 1;

}else if (valor < arbol->valor){

return Existe(arbol->izdo, valor);

}else{

return Existe(arbol->drcho, valor);

}

}

//Funcion para obtener las ramas de un nodo de forma recursiva

Nodo \*Obtener(Nodo \*arbol, int valor){

if (arbol == NULL){

return NULL;

}else if (arbol->valor == valor){

return arbol;

}else if (valor < arbol->valor){

return Obtener(arbol->izdo, valor);

}else{

return Obtener(arbol->drcho, valor);

}

}

//Funcion para reemplazar un nodo en un arbol especificado

//la funcion verifica que exista un pdre para el arbol proporcionado

//y si existe padre procede a verificar en donde lo reemplaza y hace

//la accion de reemplazar

static void Reemplazar(Nodo \*arbol, Nodo \*nuevoNodo){

if (arbol->padre){

if (arbol->valor == arbol->padre->izdo->valor){

arbol->padre->izdo = nuevoNodo;

}else if (arbol->valor == arbol->padre->drcho->valor){

arbol->padre->drcho = nuevoNodo;

}

if (nuevoNodo){

nuevoNodo->padre = arbol->padre;

}

}

}

//Funcion de apoyo para la funcion de eliminarNodo

//Esta funcion devuelve el menor valor de el arbol

static Nodo \*Minimo(Nodo \*arbol){

if (arbol == NULL){

return NULL;

}if (arbol->izdo){

return Minimo(arbol->izdo);

}else{

return arbol;

}

}

//Funcion de apoyo para la funcion en el cual se le proporciona

//el nodo a eliminar y porcede a eliminarlo y dejar a sus hijos

//si es que existen

static void EliminarNodo(Nodo \*nodoEliminar){

if (nodoEliminar->izdo && nodoEliminar->drcho){

//Tratar elimiar con 2 hijos

Nodo \*minimo = Minimo(nodoEliminar->drcho);

nodoEliminar->valor = minimo->valor;

EliminarNodo(minimo);

}else if (nodoEliminar->izdo){

//Tratar eliminar con 1 hijo

Reemplazar(nodoEliminar, nodoEliminar->izdo);

DestruirNodo(nodoEliminar);

}else if (nodoEliminar->drcho){

//Tratar eliminar con 1 hijo

Reemplazar(nodoEliminar, nodoEliminar->drcho);

DestruirNodo(nodoEliminar);

}else{

//Tratar eliminar con 0 hijos

Reemplazar(nodoEliminar, NULL);

DestruirNodo(nodoEliminar);

}

}

//Funcion en el cual se elimina un nodo a partir de su funcion de apoyo

void Eliminar(Nodo \*arbol, int valor){

if (arbol == NULL){

return;

}else if (valor < arbol->valor){

Eliminar(arbol->izdo, valor);

}else if (valor > arbol->valor){

Eliminar(arbol->drcho, valor);

}else{

EliminarNodo(arbol);

}

}

//Funcion en la cual se imprime el arbol en forma de

//preorden

void Preorden(Nodo \*arbol){

if (arbol == NULL){

printf(" - ");

}else{

printf("( %d ", arbol->valor);

Preorden(arbol->izdo);

Preorden(arbol->drcho);

printf(" )");

}

}

//Funcion en la cual se imprime el arbol en forma de

//inorden

void Inorden(Nodo \*arbol){

if (arbol == NULL){

printf(" - ");

}else{

printf("( ");

Inorden(arbol->izdo);

printf(" %d ", arbol->valor);

Inorden(arbol->drcho);

printf(" )");

}

}

//Funcion en la cual se imprime el arbol en forma de

//postorden

void Postorden(Nodo \*arbol){

if (arbol == NULL){

printf(" - ");

}else{

printf("( ");

Postorden(arbol->izdo);

Postorden(arbol->drcho);

printf(" %d )", arbol->valor);

}

}

//Funcion main con ejemplos de insercion, eliminacion e impresion de datos

int main(void){

Nodo \*arbol = NULL;

Insertar(&arbol, 5); Inorden(arbol); printf("\n");

Insertar(&arbol, 10); Inorden(arbol); printf("\n");

Insertar(&arbol, 4); Inorden(arbol); printf("\n");

Insertar(&arbol, 9); Inorden(arbol); printf("\n");

Insertar(&arbol, 15); Inorden(arbol); printf("\n");

Insertar(&arbol, 20); Inorden(arbol); printf("\n");

Insertar(&arbol, 2); Inorden(arbol); printf("\n");

Eliminar(arbol,4); Inorden(arbol); printf("\n");

Eliminar(arbol,15); Inorden(arbol); printf("\n");

Eliminar(arbol,5); Inorden(arbol); printf("\n");

}

Codigo claro para descargar en el siguiente enlace:

<https://github.com/RafaelGarcia64/Estructura-de-datos-arboles-explicada>

**Conclusión**

**Recomendaciones**

Se recomienda tratar de que las líneas de código para cada función sean mínimas, ya que, si se quisiera implementar una aplicación con una cantidad significativa de funciones para listas simples, el código en general sería complejo y robusto.

Es recomendable, que, en nuestros árboles, todos los nodos contengan el mismo número de punteros, es decir, usar 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.

Es recomendable, aunque no es estrictamente necesario, para hacer más fácil moverse a través del árbol, añadir un puntero a cada nodo que apunte al nodo padre. De este modo podremos avanzar en dirección a la raíz, y no sólo hacia las hojas.

Se recomienda usar los arboles binarios, ya que los arboles tienden a hacer más óptimos por profundidad que por anchura, es decir, es más ágil en el recorrido

Por último, hay que tomar en cuenta que, si bien los arboles nos dan la posibilidad de registrar gran cantidad de datos de forma dinámica, hay que tener cuidado de no perder los enlaces de ella, ya que, si por error eliminamos un nodo que no deberíamos haber eliminado, no lo podremos recuperar; peor aún si es la cabecera (raíz) la que se pierde. Es importante conservar siempre el nodo raíz ya que es el nodo a partir del cual se desarrolla el árbol, si perdemos este nodo, perderemos el acceso a todo el árbol.

**Bibliografía**

* Playlist de YouTube canal Makigas : <https://www.youtube.com/playlist?list=PLTd5ehIj0goMTSK7RRAPBF4wP-Nj5DRvT>
* Artículo en Wikipedia: <https://es.wikipedia.org/wiki/%C3%81rbol_(inform%C3%A1tica)>
* Documento informativo de la universidad de Valladolid: <https://www.infor.uva.es/~cvaca/asigs/doceda/tema4.pdf>
* Blog sobre programación: <http://c.conclase.net/edd/?cap=006&fbclid=IwAR1A5kGjOX2lo_fzXvLYZQKSw6HC7Oy682wuilcmJ8unSxM4q_3e8IA3Dr8>