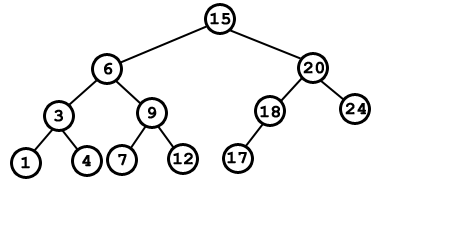
**Estructura de datos Arboles**

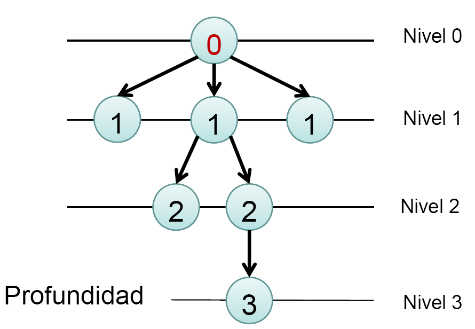
**Definición simple:**

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:**

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

**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>