UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
ESTRUCTURAS DE DATOS

ÁRBOLES BINARIOS

1. Árboles en General

Un **árbol** es una estructura que organiza sus elementos, formando jerarquías. Tiene como característica que es ramificada o no lineal porque a cada elemento le puede suceder uno o varios elementos, llamados **nodos**. Los árboles son estructuras de datos dinámicas porque pueden cambiar durante la ejecución de un programa.

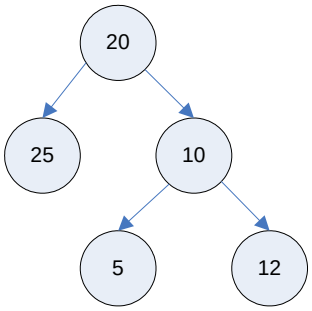
Un elemento n2 es **hijo** de un elemento n1, si n2 es la raíz de uno de los subárboles asociados con n1. En ese mismo caso, se dice que n1 es el **padre** de n2. Un elemento n2 es **hermano** de un elemento n3 si ambos tienen el mismo padre.

Un elemento de un árbol binario es una **hoja** si sus dos subárboles asociados son vacíos. En otras palabras, un nodo que no tiene hijos se denomina hoja. Los nodos con descendientes se denominan **nodos interiores**. Cualquier nodo sin sucesores se denomina un **nodo terminal**. En la figura anterior C, D, E y F, son hojas. Todo elemento de un árbol que no es una hoja se denomina un elemento **no terminal** o interior.

Para el árbol de la siguiente figura:

Se tiene que:

- La raíz es 20
- Los elementos 25, 5 y 12 son hojas
- Los nodos interiores son 20 y 10
- El padre de 5 es 10, de 25 es 20.
- Los hijos de 10 son 5 y 12.
- Los elementos 5 y 12 son hermanos.



Un elemento n1 es **ancestro** de un elemento n2, si existe un camino entre n1 y n2. En ese mismo caso, se dice que n2 es **descendiente** de n1. El **nivel** de un elemento dentro de un árbol binario se define como la longitud del camino que parte de la raíz y llega hasta él. De esta forma, el nivel de la raíz es 0 y el de cualquier elemento es uno más que el de su padre. El nivel determina qué tan lejos de la raíz se encuentra un elemento. El **ancestro común** más próximo de dos elementos n1 y n2 es un elemento n3, que cumple que es ancestro de ambos y se encuentra a mayor nivel que cualquier otro ancestro que compartan.

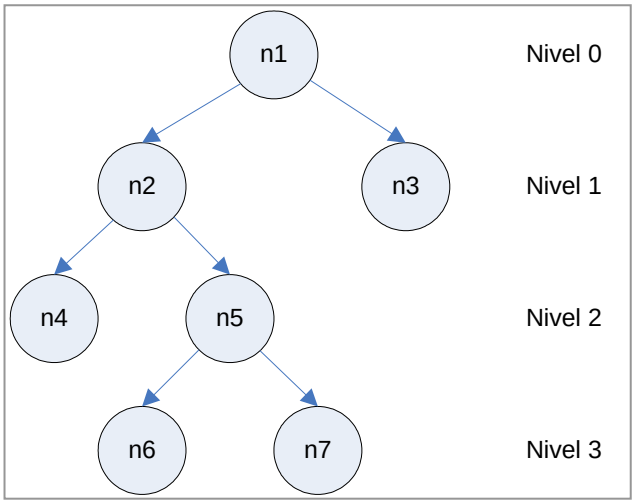
La **altura** de un árbol es la longitud del camino más largo que parte de la raíz más 1. Es el nivel más alto del árbol. La altura es igual a la longitud del camino desde el nodo raíz a la hoja más lejana

que sea alcanzable desde él. La altura de un árbol vacío se define como 0. Un árbol que contiene sólo raíz, tiene una altura de 1.

El **nivel** (profundidad) de un árbol, es el número de nodos que se encuentra entre él y la raíz. Por definición, el número de niveles de un árbol se define como el nivel de la hoja más profunda. El número de niveles de un árbol se llama **altura** (h).

El **grado** o aridad es el número de hijos del nodo. La aridad de un árbol se define como el máximo de la aridad de sus nodos. El **peso** de un árbol es el número de elementos que contiene. Recursivamente se puede definir como la suma de los pesos de sus elementos más 1. De acuerdo con la definición, el peso de un árbol vacío es 0.

Para el siguiente árbol:



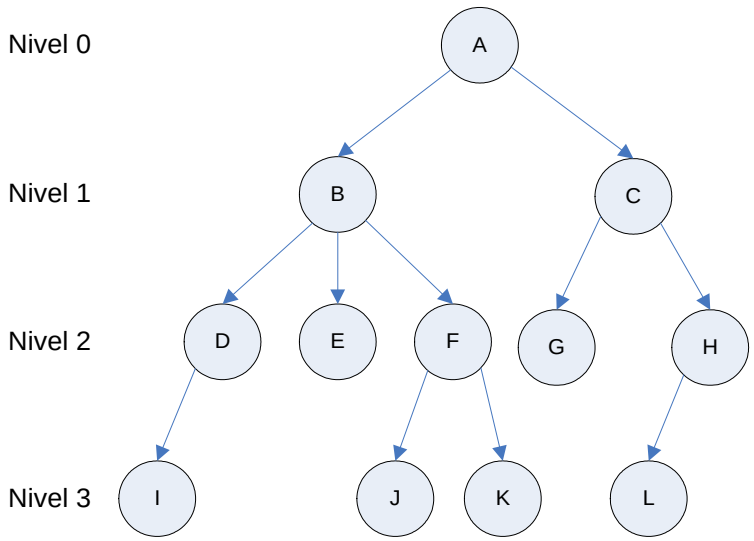
Se tiene que:

- La altura es 4.
- El peso es 7.
- El elemento n1 es ancestro de todos los elementos del árbol.
- El elemento n7 es descendiente de n2.
- El ancestro común más próximo de n4 y n7 es n2.
- El ancestro común más próximo de n6 y n1 es n1.

Los árboles pueden representarse a través de diagramas de Venn, anidación de paréntesis, notación decimal de Dewey, notación indentada o como grafos.

Por ejemplo, se puede encontrar la representación equivalente en grafos de la siguiente anidación de paréntesis:

A(B(D(I),E,F(J,K)),C(G,H(L)))



Grado = 3
G(raíz) = G(A)
G (A)= 2
G (B)= 3
G (C)= 2
G (D)= 1
G (E)= 0
G (F)= 2
G (G)= 0
G (H)= 1
G (I)= 0
G (J)= 0
G (K)= 0
G (L)= 0
n = 12
h = 4

La **longitud de camino** X es el número de arcos recorridos de la raíz al nodo X. Por definición la raíz tiene longitud de camino 1, sus descendientes directos longitud de camino 2 y así sucesivamente. Aplicando este concepto para el árbol anterior, se tiene:

Nodos	Raíz (Nodo A)	B y C	D, E, F, G y H	I, J, K y L
Longitud Camino	1	2	3	4

La **Longitud de Camino Interno** (LCI) es la suma de las longitudes de camino de todos los nodos y puede calcularse utilizando la siguiente fórmula:

$$LCI = \sum_{i=1}^{h} n_i * i$$

Donde: i: nivel del árbol
h: altura del árbol
n_i: nodos del nivel i

***Es de tomar en cuenta que la numeración de los niveles inicia en 1 y no en 0 como se estableció anteriormente, este cambio se aplica solamente para calcular la longitud del camino tanto interno como externo. Además la longitud de la raíz es uno por lo que el nivel de la raíz para el la longitud del camino debe ser uno y de esa forma cumplir con la formula.

Aplicando este concepto para el árbol anterior, se tiene:

$$LCI = 1*1 + 2*2 + 5*3 + 4*4 = 1 + 4 + 15 + 16 = 36$$

La Media de la Longitud de Camino Interno (LCIM) es el número de arcos a recorrer en promedio para llegar a cualquier nodo desde la raíz y se calcula así: **LCIM = LCI / n**

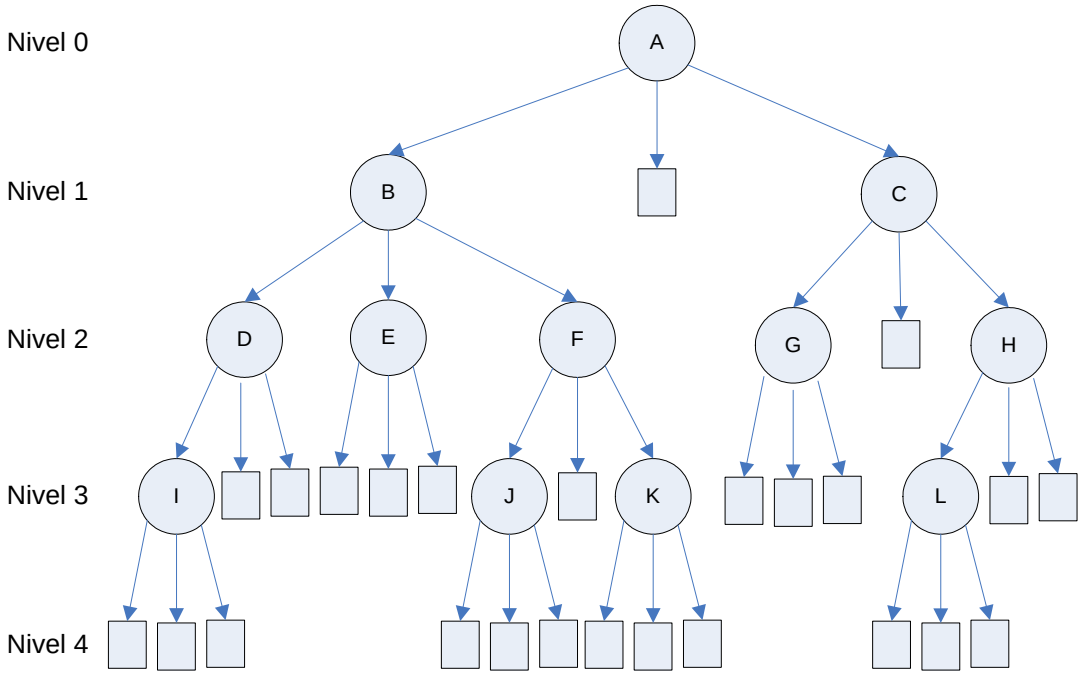
Aplicando este concepto para el árbol anterior, se tiene: LCIM = 36/12 = 3

En un **árbol extendido** el número de hijos de cada nodo es igual al grado del árbol. Se llaman **nodos especiales** a los nodos agregados para completar el árbol extendido.

Son características de los nodos especiales:

a) Reemplazan ramas vacías o nulas
b) No tienen hijos
c) Se representan en forma de cuadrado

El árbol extendido asociado al árbol anterior, con sus respectivos nodos especiales, se expresaría en forma de grafo así:



Grado = 3 **ne = 25** **h = 5**

G(raíz) = G(A) = G(B) = G(C) = G(D) = G(E) = G(F) = G(G) = G(H)= G(I) = G(J) = G(K) = G(L) = 3

La Longitud de Camino Externo (LCE) es la suma de las longitudes de camino de todos los nodos especiales de un árbol extendido y puede calcularse utilizando la siguiente fórmula:

$$LCE = \sum_{i=2}^{h} ne_i * i$$

Donde:
- i: nivel del árbol extendido
- h: altura del árbol extendido
- ne_i: nodos especiales del nivel i

Obsérvese que i comienza desde 2, puesto que la raíz se encuentra en el nivel 1 y no puede ser un nodo especial. Aplicando este concepto para el árbol extendido anterior, se tiene:

LCE = 1*2 + 1*3 + 11*4 + 12*5 = 2 + 3 + 44 + 60 = 109

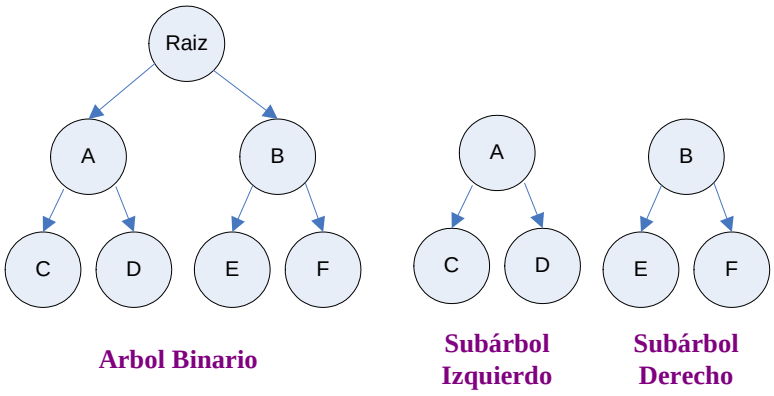
La Media de la Longitud de Camino Externo (LCEM) es el número de arcos a recorrer en promedio para llegar a cualquier nodo especial desde la raíz y se calcula así: **LCEM = LCE / ne**

Aplicando este concepto para el árbol extendido anterior, se tiene: LCEM = 109/25 = 4.36

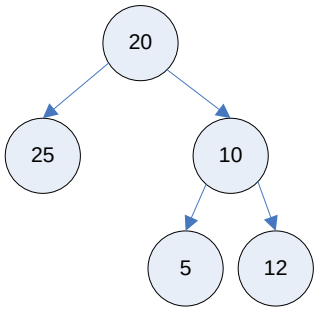
2. Árboles Binarios

Un **árbol binario** es un árbol en el que cada nodo no puede tener más de dos hijos o descendientes. El caso más sencillo de árbol binario es un árbol **vacío**, el cual no tiene elementos ni subárboles asociados.

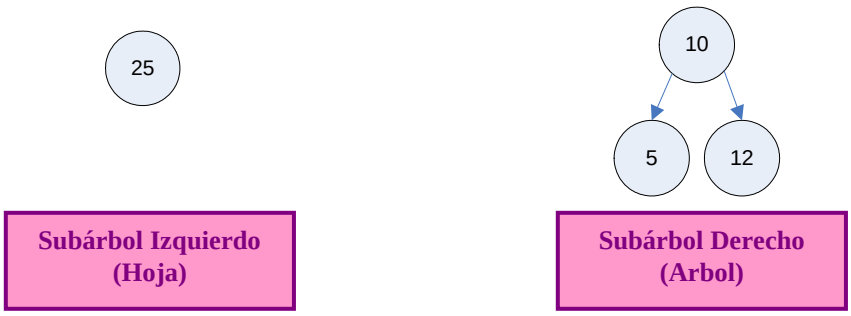
Un **árbol binario** es una estructura de datos **recursiva**, compuesta por un elemento, denominado **raíz**, y por dos árboles binarios asociados, denominados **Sub-Arbol Derecho (SAD)** y **Sub-Arbol Izquierdo (SAI)**. El hecho de definir la estructura de datos en términos de sí misma es lo que hace que se denomine recursiva. Es decir, cada subárbol se define como un árbol más simple. Los dos subárboles tienen la misma composición estructural del árbol completo.



Para el árbol de la siguiente figura:



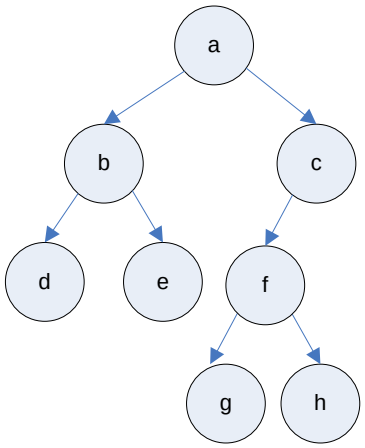
Se tiene que, los dos subárboles asociados son:



Un **camino** entre dos elementos n1 y n2 de un árbol binario es una secuencia que cumple que el primer elemento es n1, el último es n2, y cada elemento es padre de su sucesor. No siempre existe un camino entre dos elementos de un árbol, pero si existe, éste es único. La raíz de un árbol se caracteriza porque tiene un camino a cualquier elemento árbol. Un camino también puede definirse como una secuencia de nodos conectados dentro de un árbol.

La **longitud** de un camino es n-1, o sea, el número de veces que se debe aplicar la relación padre-hijo durante el recorrido. Siempre existe un camino de longitud 0 que lleva de un elemento r a sí mismo y corresponde a la secuencia <r>. Si r>0, se dice que el camino es propio. Por último, se tiene que un camino que parte de la raíz y termina en una hoja se conoce como una **rama**.

Para el árbol que se muestra en la siguiente figura:



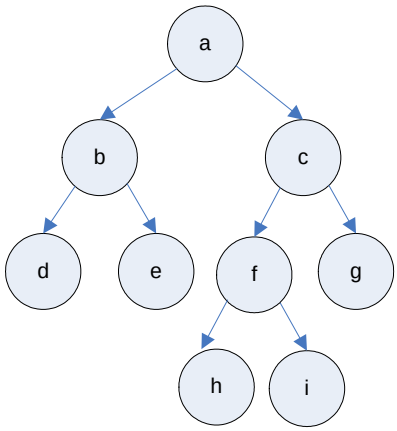
Se cumple que:

- La longitud del camino <a, b, e> es 2. La longitud del camino <a> es 0.
- No existe un camino entre d y c.
- El único camino que lleva de c a h es <c, f, h>
- El camino <a, c, f, g> es una rama.
- Desde la raíz existe un camino que lleva hasta cualquier otro elemento de la estructura.

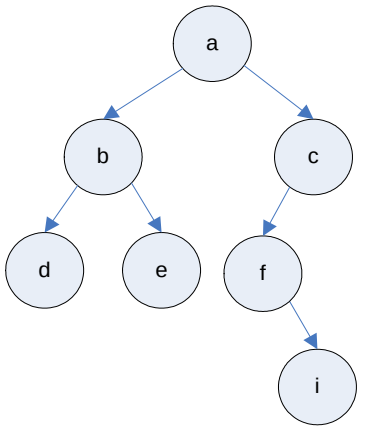
Un **Árbol Binario Completo (ABC)** es cuando todo elemento no terminal tiene asociados exactamente dos subárboles no vacíos. Eso equivale a decir que todo elemento de un árbol completo tiene los dos subárboles o no tiene ninguno.

Un árbol binario está **lleno** si es completo y, además, todas las hojas están en el mismo nivel. Un árbol binario está **casi lleno** si está lleno hasta el penúltimo nivel y todas las hojas del siguiente nivel están tan a la izquierda como es posible. De acuerdo con la definición dada anteriormente, un árbol lleno es un caso particular de un árbol casi lleno.

También puede decirse que un **árbol binario lleno** es aquel en el que cada nodo tiene o dos hijos o ninguno si es una hoja. Es decir, todos los nodos, excepto los del último nivel, tienen dos hijos.



Arbol Binario Completo



Arbol Binario No Completo

Un árbol binario lleno de altura h tiene todas sus hojas a nivel h y todos los nodos que están a nivel menor que h tiene cada uno dos hijos.

Un árbol binario completo de altura h es un árbol binario que está relleno a partir del nivel h-1, con el nivel h relleno de izquierda a derecha.

Si un árbol binario es lleno, es necesariamente completo

El número de nodos de un árbol lleno, puede calcularse con la fórmula siguiente:

$$n_{lleno} = 2^h - 1$$

Así: Para h=05 $n_{lleno} = 31$

h=09 $n_{lleno} = 511$

h=17 $n_{lleno} = 131, 071$

Un árbol binario es completamente (totalmente) equilibrado si los subárboles izquierdo y derecho de cada nodo tienen la misma altura.

Un árbol binario completo es equilibrado, mientras que un árbol binario lleno es totalmente equilibrado

Dos árboles binarios son **equivalentes** si ambos son vacíos, o si sus raíces son iguales, lo mismo que sus respectivos subárboles izquierdo y derecho. Dos árboles binarios son **isomorfos o similares** si tienen la misma estructura, pero no necesariamente los mismos elementos.

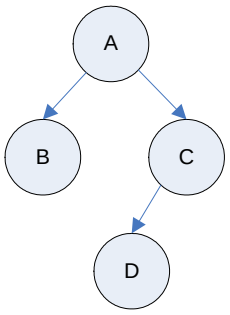
Dos árboles binarios son **distintos** cuando sus estructuras son diferentes.

Dos árboles binarios son **semejantes** si contienen los mismos elementos, aunque no sean isomorfos. En ese caso, se dice que tienen el mismo contenido, pero no la misma estructura.

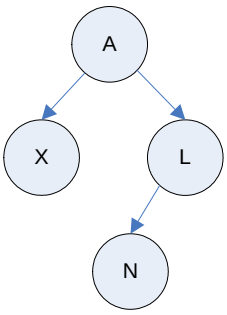
Dados los siguientes cuatro árboles, establecer las relaciones que guardan entre ellos:

SOLUCION: (3) es distinto a (1), (2) y (4) ; (1), (2) y (4) son similares o isomorfos ; (1) y (4) son equivalentes

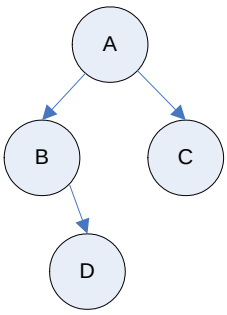
Un árbol binario a1 **ocurre** en otro árbol binario a2, si a1 y a2 son iguales, o si a1 ocurre en alguno de los subárboles a2.



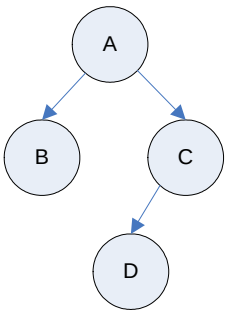
(1)



(2)



(3)



(4)

3. Árboles Binarios de Búsqueda.

En este tipo de árboles cada nodo tiene asociado un valor de clave y este valor de clave es mayor o igual que los valores de clave de los nodos de su subárbol izquierdo y menor o igual que todos los de su subárbol derecho. Según el tipo de aplicación puede no permitirse valores iguales en uno o ambos subárboles. Si se realiza un recorrido en inorden por el árbol accederemos a los valores de clave en orden ascendente.

Declaracion de la estructura.

Un árbol binario puede declararse de la siguiente manera:

```
typedef struct tArbol
{
  int clave;
  struct tArbol *hIzquierdo, *hDerecho;
} tArbol;
```

Operaciones

Búsqueda

El proceso de búsqueda en un **árbol binario de búsqueda** se realiza recursivamente. Primero se comprueba la raíz. Si el valor de clave es igual al elemento a buscar, determinamos que el valor existe. Si el elemento es menor realizamos una búsqueda recursiva en el subárbol izquierdo, si por el contrario es mayor realizamos la búsqueda en el subárbol derecho. Si alcanzamos un nodo hoja y no está el elemento, determinamos que éste no se encuentra en el árbol.

El siguiente código en lenguaje C muestra cómo realizar una búsqueda en un ABB:

```
int buscar(tArbol *a, int elem)
{
  if (a == NULL)
    return 0;
  else if (a->clave < elem)
    return buscar(a->hDerecho, elem);
  else if (a->clave > elem)
    return buscar(a->hIzquierdo, elem);
  else
    return 1;
}
```

Inserción

La inserción es similar a la búsqueda. Si el árbol pasado por parámetro está vacío se crea un nuevo nodo para él y se le su valor de clave correspondiente. Si no lo está, se comprueba si el elemento a insertar es menor con lo que insertamos el elemento en el subárbol izquierdo, o mayor, insertando el elemento en el subárbol derecho.

Este algoritmo en C muestra la inserción en un ABB. El árbol es pasado como puntero por referencia, para que los nuevos enlaces a los subárboles mantengan la coherencia.

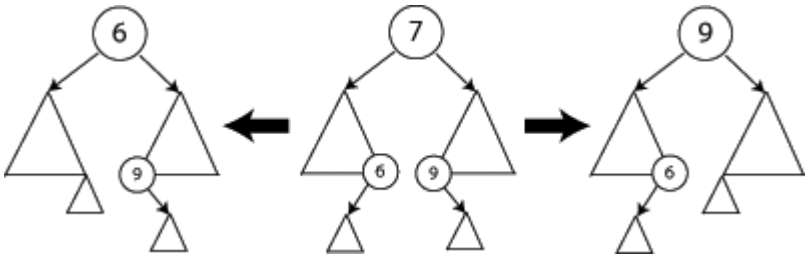
```
void insertar(tArbol **a, int elem)
{
  if (*a == NULL)
```

```
  {
    *a = (tArbol *) malloc(sizeof(tArbol));
    (*a)->clave = elem;
    (*a)->hIzquierdo =  NULL;
    (*a)->hDerecho = NULL;
  }
  else if ((*a)->clave < elem)
    insertar(&(*a)->hDerecho, elem);
  else if ((*a)->clave > elem)
    insertar(&(*a)->hIzquierdo, elem);
}
```

Borrado

La operación de borrado no es tan sencilla como las de búsqueda e inserción. Existen varios casos a tener en consideración:

- **Borrar un nodo sin hijos ó nodo hoja**: simplemente se borra y se establece a nulo el apuntador de su padre.
- **Borrar un nodo con un subárbol hijo**: se borra el nodo y se asigna su subárbol hijo como subárbol de su padre.
- **Borrar un nodo con dos subárboles hijo**: la solución está en reemplazar el valor del nodo por el de su predecesor o por el de su sucesor en inorden y posteriormente borrar este nodo. Su predecesor en inorden será el nodo más a la derecha de su subárbol izquierdo, y su sucesor el nodo más a la izquierda de su subárbol derecho. En la siguiente figura se muestra cómo existe la posibilidad de realizar cualquiera de ambos reemplazos:



El siguiente algoritmo en C realiza el borrado en un ABB. El procedimiento *reemplazar* busca la mayor clave del subárbol izquierdo y la asigna al nodo a eliminar.

```
void borrar(tArbol **a, int elem)
{
  void reemplazar(tArbol **a, tArbol **aux);
  tArbol *aux;

  if (*a == NULL)
    return;

  if ((*a)->clave < elem)
    borrar(&(*a)->hDerecho, elem);
  else if ((*a)->clave > elem)
    borrar(&(*a)->hIzquierdo, elem);
  else if ((*a)->clave == elem)
  {
    aux = *a;
    if ((*a)->hIzquierdo == NULL)
```

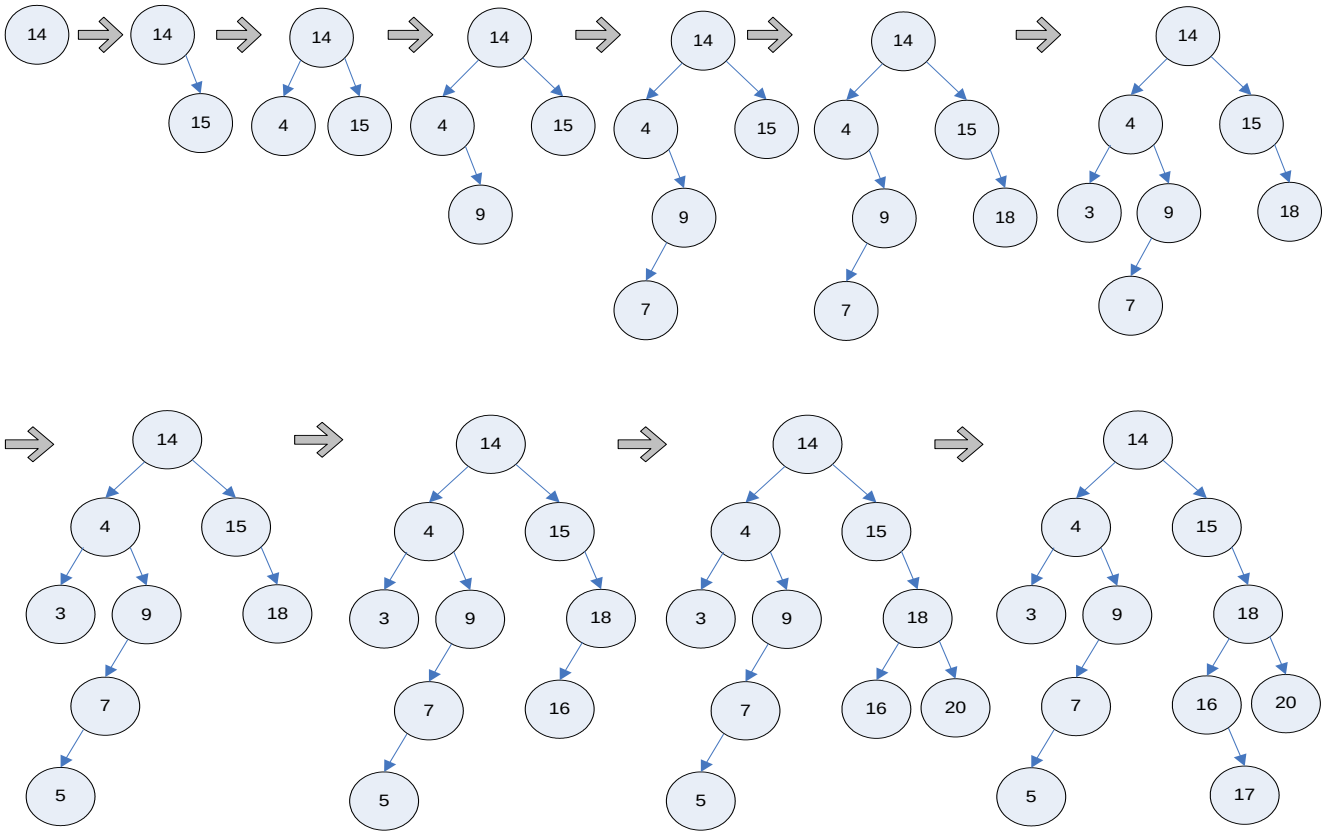
```
      *a = (*a)->hDerecho;
    else if ((*a)->hDerecho == NULL)
      *a = (*a)->hIzquierdo;
    else
      reemplazar(&(*a)->hIzquierdo, &aux);

    free(aux);
  }
}

void reemplazar(tArbol **a, tArbol **aux)
{
  if ((*a)->hDerecho == NULL)
  {
    (*aux)->clave = (*a)->clave;
    *aux = *a;
  }
  else
    reemplazar(&(*a)->hDerecho, aux);
}
```

4. Construcción de Árboles Binarios de Búsqueda.

Construir un árbol binario con la siguiente secuencia de claves, rechazando los valores duplicados: **14 15 4 9 7 18 3 5 16 4 20 17 9 14 5**



5. Ejercicios de Árboles Binarios

5.1. ¿Cuántos ancestros tiene un nodo de un árbol binario que esté en el nivel n?

5.2. Construir un árbol binario de ordenamiento con la siguiente secuencia de claves (ingresarlas todas, aún las duplicadas): **14 15 4 9 7 18 3 5 16 4 20 17 9 14 5**

5.3. Escribir programas en C para determinar:

a) Número de Nodos de un árbol binario

b) Suma de todas las claves de un árbol binario

c) Altura de un árbol binario

d) Determinar si un árbol binario es completo

e) Determinar si un árbol binario está lleno