Carne: 2025072099

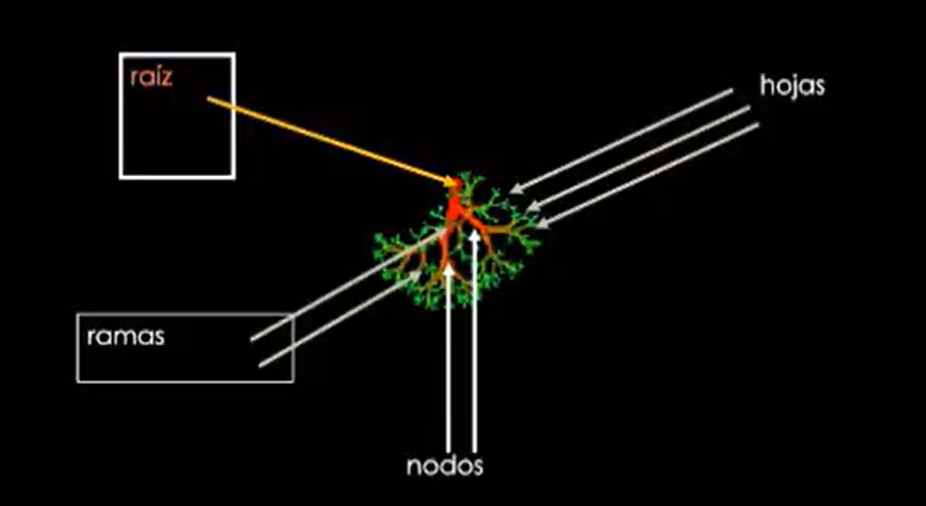
Grupo: 02

Estudiante: Ronny Espinoza Cordero

Aula invertida #7 Arboles Binarios

Arboles binarios

Viendo un árbol se pueden analizar sus elementos, la raíz, las ramas y las hojas, exactamente en ese orden. Aun así para un informático los árboles se deben analizar al revés, agregando un elemento, los nodos que son los elementos que son conectados con las ramas. Ósea se puede ver así:



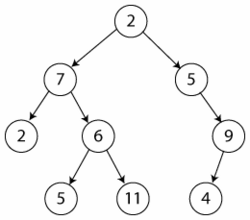
De esta forma se le pueden asignar a los elementos ciertas posiciones jerárquicas:  
El elemento superior es la raíz

Luego están los hijos de la raíz.

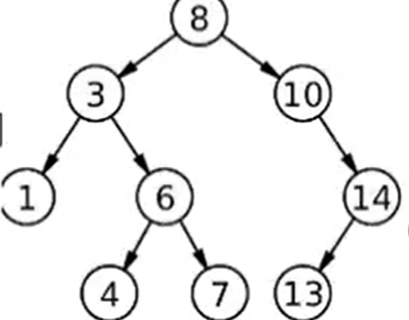
Seguido de esto los nietos de la raíz y así sucesivamente.

Y finalmente las hojas, que son elementos que no tienen hijos, ósea los elementos que siempre están al final.

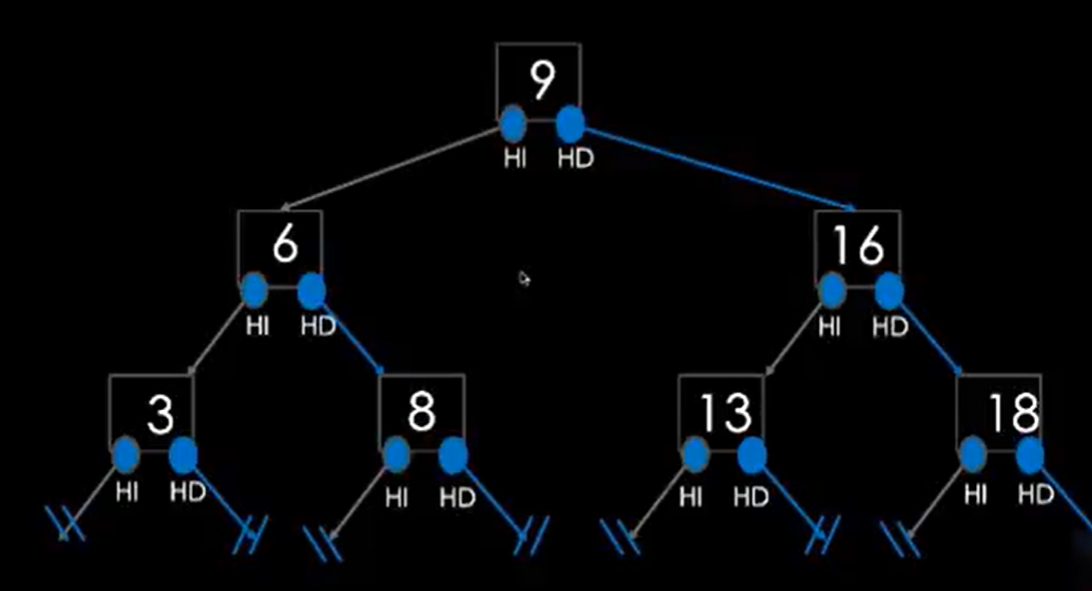
Un ejemplo de árbol binario y la vez su comparación con un árbol real es:

Al analizar un caso como el siguiente se puede notar que en cada parte del árbol hay una condición, por ejemplo, los números que están a la izquierda del siempre son menores que los números del medio y lo que están a la derecha siempre son mayores.



Analizando un árbol binario más a detalle se pueden entender mejor sus partes, analicemos este ejemplo:



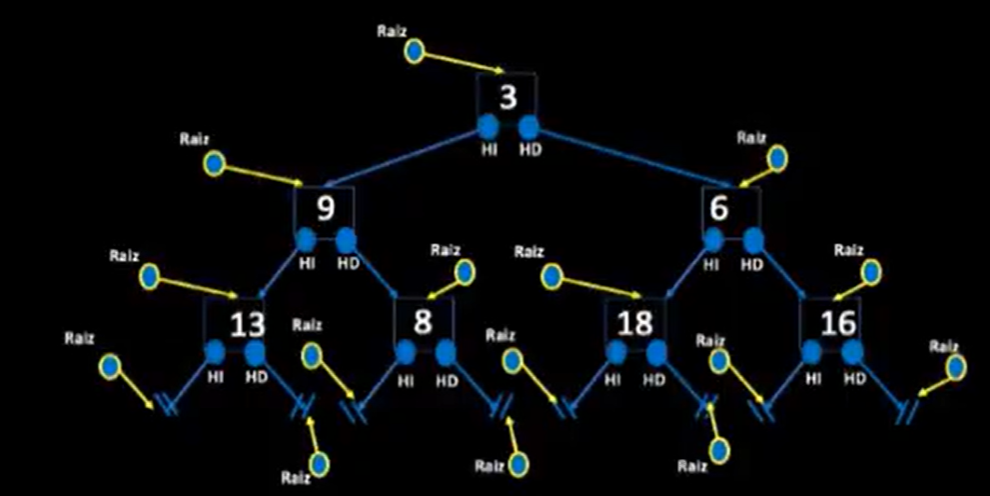
Cada elemento que este en un cuadrito representa un nodo, cada nodo tiene un Hijo a la Izquierda (HI)y un Hijo a la Derecha (HD).

Las flechas grises son las ramas izquierdas y las azules las ramas derechas.

El cuadrito con el número 9 es la raíz del árbol, este elemento es el único elemento conocido (algo parecido a lo que se vio en lista enlazada).

Los nodos intermedios (Nodos 6 y 16) son los nodos que no son la raíz y tampoco son los nodos finales.

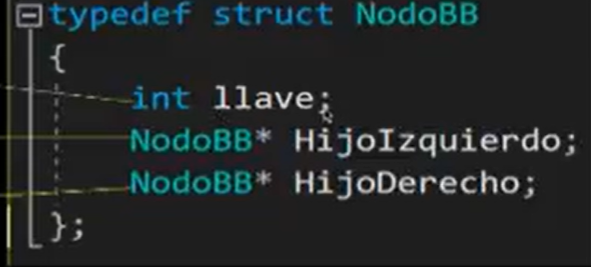
Finalmente se tiene la hoja (Nodos 3,8,13,18) una hoja son los nodos cuyas ramas apuntan a Nulo.

Se debe destacar que por definición recursiva cada **nodo** es una raíz tal y como se puede ver en el siguiente ejemplo:

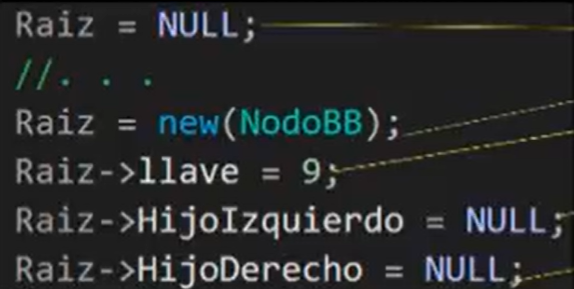
De esta forma una raíz siempre es un puntero que apunta a variables anónimas (los nodos) a medida que va avanzando el algoritmo. De esta forma se sigue avanzando hasta llegar a una raíz nula, cada vez se nota más el parecido con las listas simples.

**Nodo binario o nodobb**

Un nodo bb se refiere a un nodo binario de búsqueda. Este nodo es una estructura con una llave (forma de identificarlo), un puntero hacia la izquierda y un puntero hacia la derecha.



En el ejemplo anterior se tiene que el nodobb se genera y se ve así:



**Definición completa de árbol binario:**

De momento hemos hablado de representaciones, funciones y demás, pero no de una definición o un caso exacto.

Un árbol binario es una estructura o colección dinámica que permite almacenar 0 o más elementos del tipo Nobobb de forma jerárquica y que no puede contener llaves repetidas.

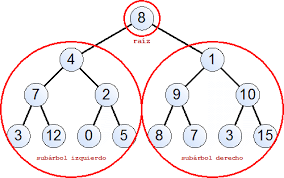
Dependiendo de si esta desordenado o no, un árbol binario se puede clasificar en Árbol Binario Normal y Árbol Binario de Búsqueda respectivamente.

Siempre se debe recordar que a nivel de representación un árbol binario es un árbol donde las raíces están al revés.

**Subarbol binario:**

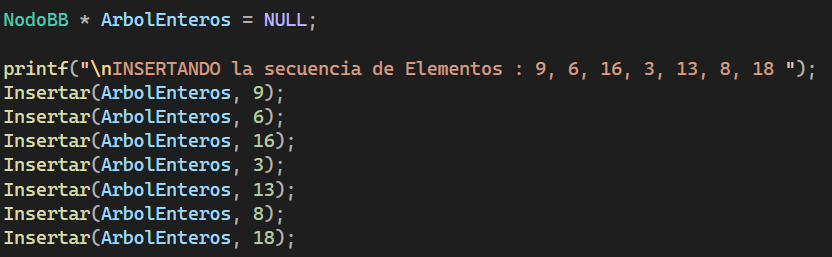
Es una porción de un árbol binario de búsqueda dividido en un subárbol izquierdo (lado izquierdo de la raíz) y un subárbol derecho (lado derecho de la raíz).

Cada subárbol tiene su propia raíz.



En síntesis se puede decir que un árbol binario de búsqueda tiene siempre un eje central, todo lo que esta a la izquierda es menor y todo lo que esta a la derecha es mayor recursivamente definido.

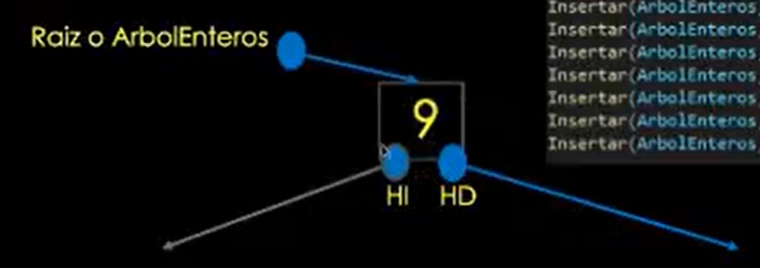
**Algoritmo de inserción en árboles binarios**

Estos algoritmos son muy parecidos por lo que se analizaran juntos. El llamado de la inserción en el main ocurre así. Se puede observar que antes de llamar la función se crea un puntero de tipo NodoBB para mandarlo en cada llamado de la función. Podemos notar que también se envía el elemento que queremos agregar.

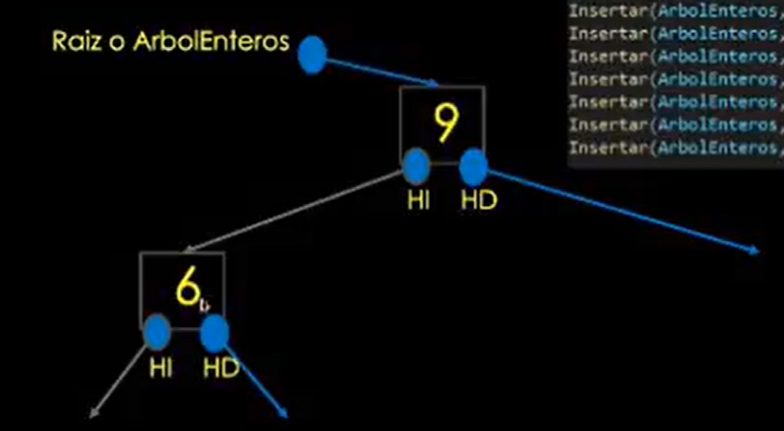
Gráficamente se puede ver así, primero se representa el hecho de que árbol enteros este apuntando a NULL.



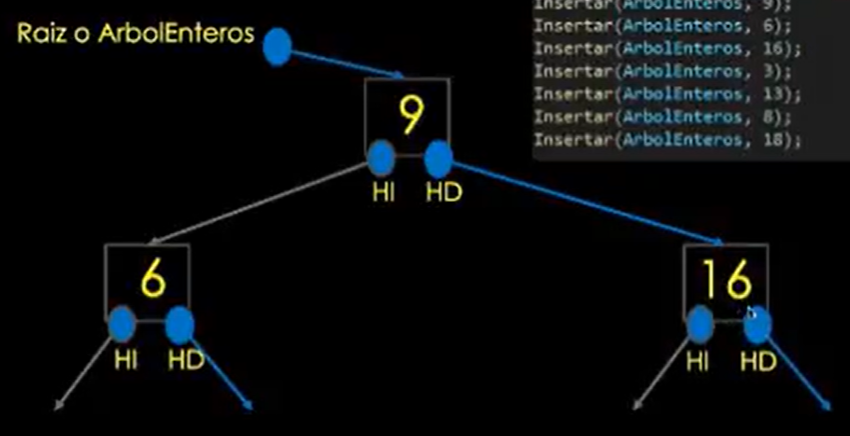
Luego se hace una comparación con el árbol, como el árbol está vacío en este momento nueve no es comparable con NULL por lo que el elemento solo se inserta:



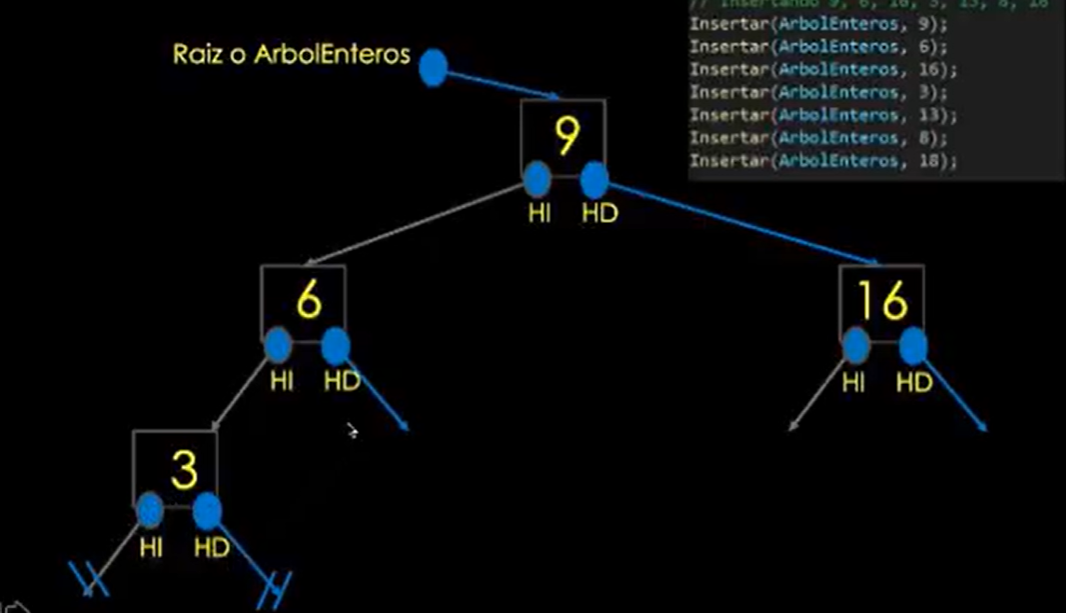
Y a partir de aquí se puede reconocer una secuencia, al agregar 6 este se compara con el árbol, como el 6 es menor que 9 se va por el lazo izquierdo (HI) al realizar la comparación con un valor NULL, ya que ese es el valor que posee el HI en este momento, el 6 se inserta y ya.



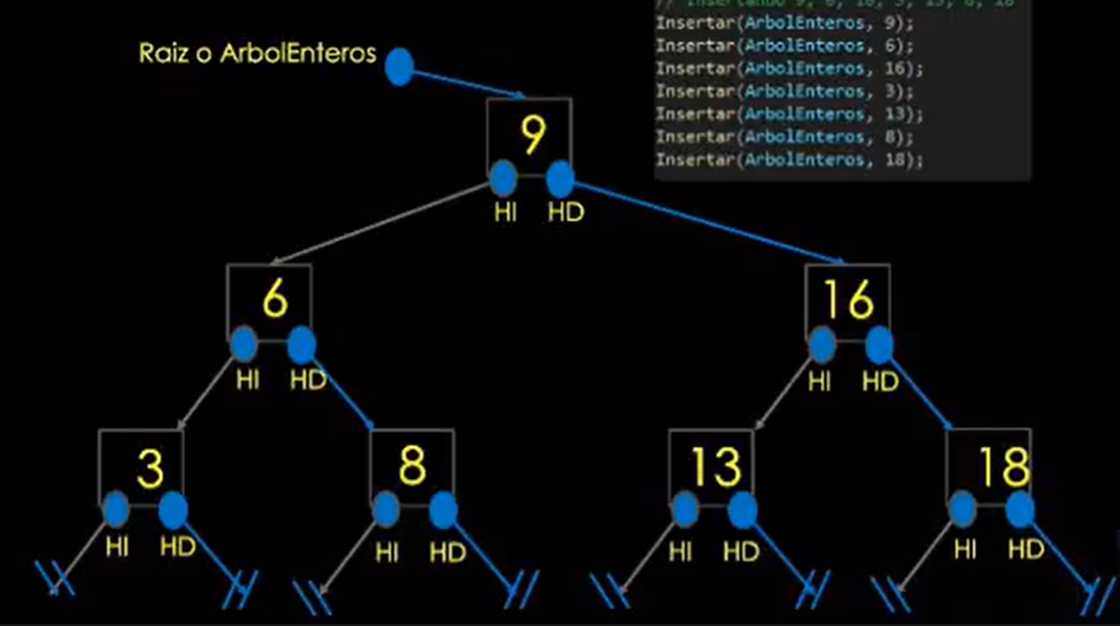
Esto ocurre igual con 16, solo que en este caso como el número es mayor a 9 el número se inserta al lado derecho.



Por último considerando otro caso esta el número 3, que es menor que 9 por lo que se va por el HI, al toparse con el número 6 este hace otra comparación, como 3 es menor que 6 este se va para el HI y ahí es insertado.

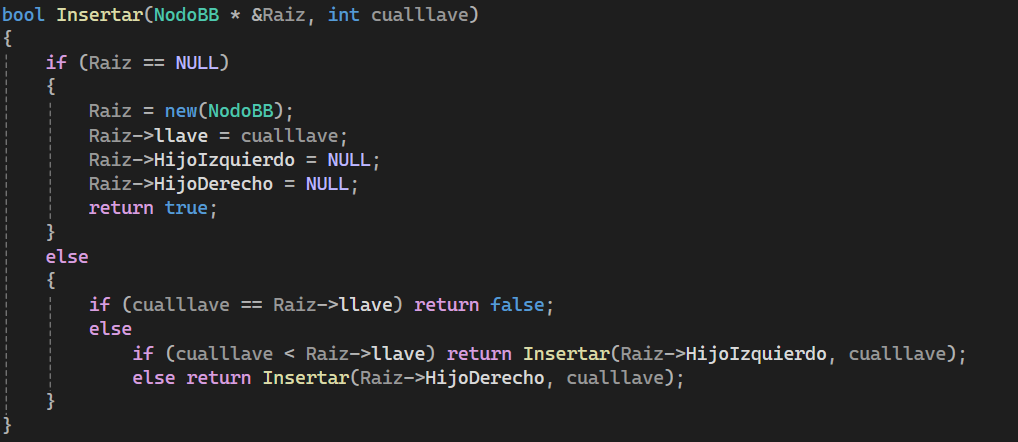


Finalmente el resultado de todas las inserciones es:

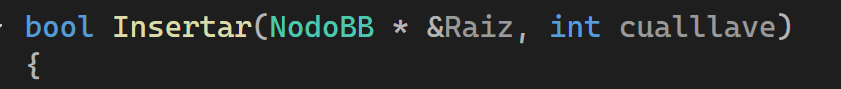


**Análisis de código de Insertar**

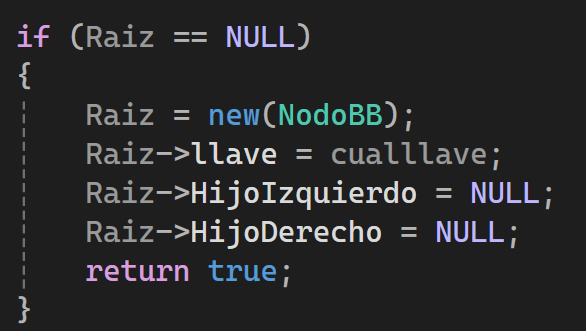
Para haber hecho todas las inserciones que se hicieron se debió de haber partido de una raíz que se usa para recorrer la secuencia. A medida que el código avanzaba la raíz iba cambiando sus valores, esto de forma que se pudiera recorrer el árbol, hubo momentos en los que el árbol adquiría valores de una variable anónima y otros donde obtenía el valor NULL. Cuando se llegaba a un valor Nulo se podía insertar un elemento sino se debía hacer una comparación. Tal y como se puede ver a continuación:



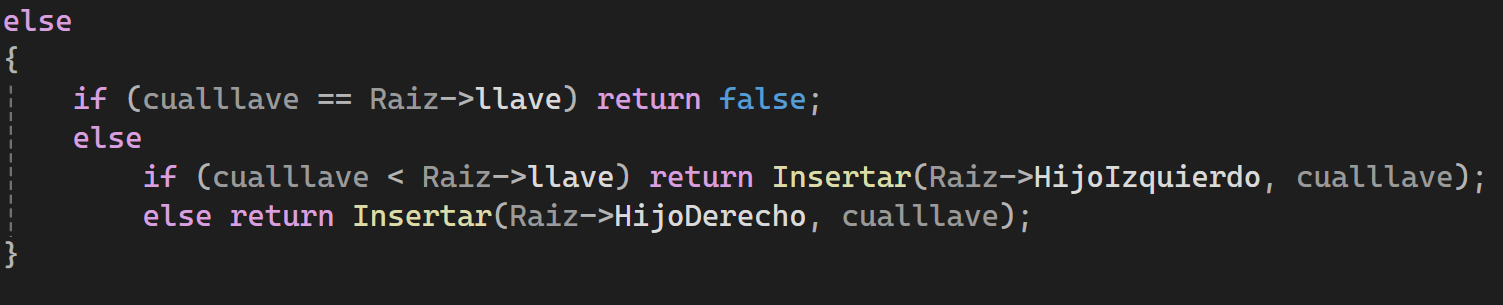
Analizando el código paso a por paso se puede ver que en un principio sus parámetros son la raíz que se tiene (parámetro por referencia ya que va a ser modificado) y lo que se va a insertar es la variable llamada cualllave. Otro punto importante es que se estaba función es una función booleana que retorna false cuando el elemento ya existe en el árbol y retorna true cuando el elemento es insertado de forma correcta.



Otro punto que no se explico pero que es muy importante analizar es que se está trabajando con una función recursiva. Entonces analizando el código se tiene el siguiente caso base que se cumple cuando la Raiz es NULL. Cuando esta condicion se cumple el código genera el nuevo nodo, le asigna el valor de cuallave y crea sus hijos, finalmente devuelve True ya que se cumplió con la inserción.

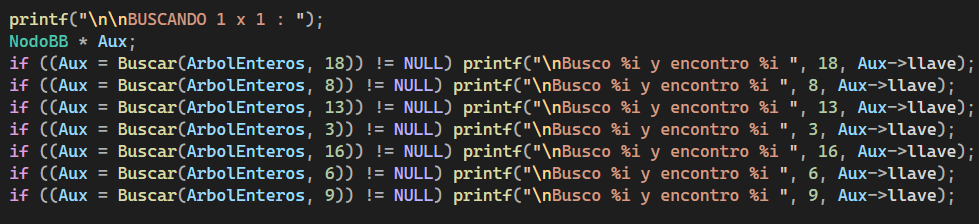


Luego el caso continuidad es el siguiente, esta condición primero verifica que el elemento no sea igual al que se esta analizando, si lo es retorna false y si no lo es empieza el ciclo recursivo, basado en si el elemento es menor o mayor que él. Si es menor lo manda por el HI si no lo es lo manda por el HD y así el ciclo sigue hasta que se llegue a NULL.

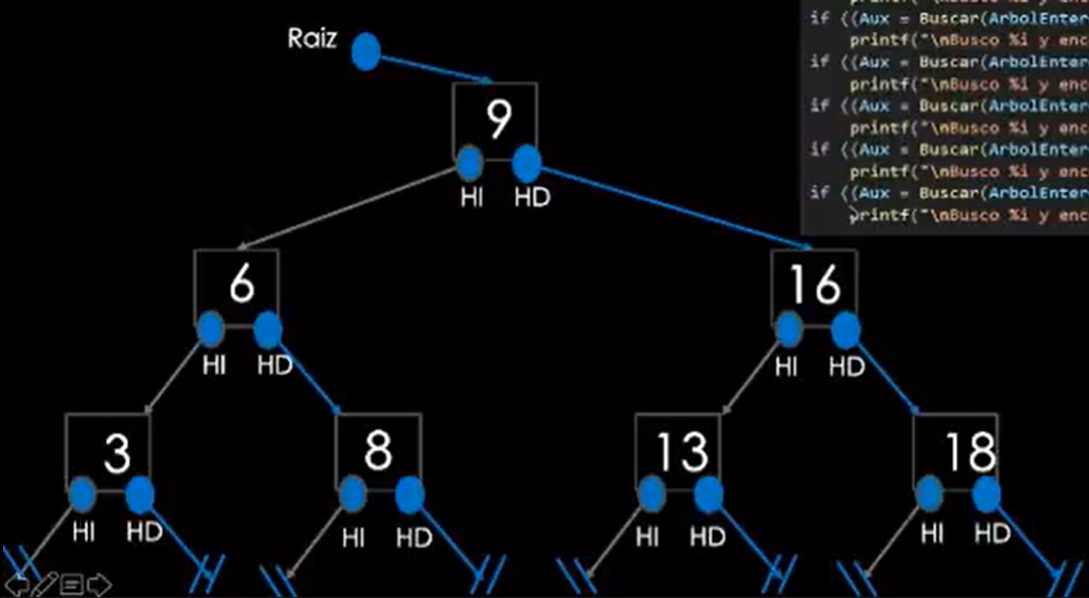


**Algoritmo de búsqueda en árboles binarios.**

En el main este algoritmo se plantea así:

****

Esto tiene una explicación muy simple y se puede ver con el ejemplo del árbol visto anteriormente:

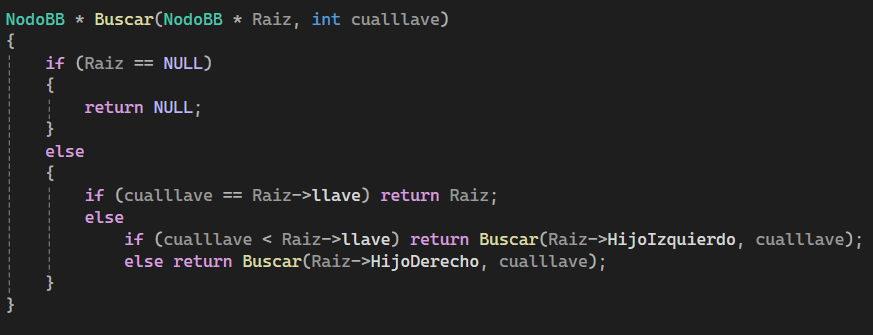


Aquí lo que ocurre es que se busca el elemento empezando desde la raíz y bajando según el valor del elemento a buscar (es menor o mayor), esto produce un ciclo hasta que se encuentre el elemento. Por ejemplo al tener un elemento como 3 este se compara con 9, como no es igual pero es menor entonces se va por el HI, como 3 tampoco es igual a 6 pero si es menor que este entonces se vuelve a pasar por el HI y finalmente los elementos coinciden.

Si por ejemplo hay un elemento como 20, que es mayor que 9,16 y 18 pero aun así no se encuentra en el árbol entonces se retorna Nulo, ósea que el elemento no existe en el árbol.

**Análisis de código de Buscar**

Este código es muy parecido a la función insertar aunque en realidad es un poco más simple.

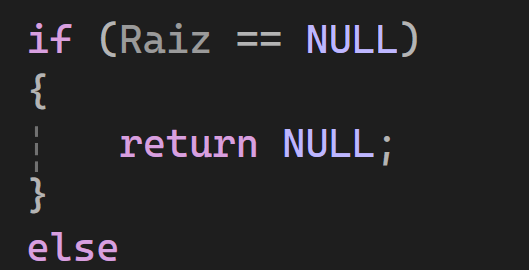


Analizando primero los parámetros se puede ver que son los mismos que en la función pasada, solo que aquí Raiz no es un parámetro por referencia debido a que el árbol no va a ser modificado. Aun así si fuera por referencia no habría ningún problema. Además el tipo de función también cambia ya que ahora es una función de tipo NodoBB dado que se va a devolver la dirección de un nodo.

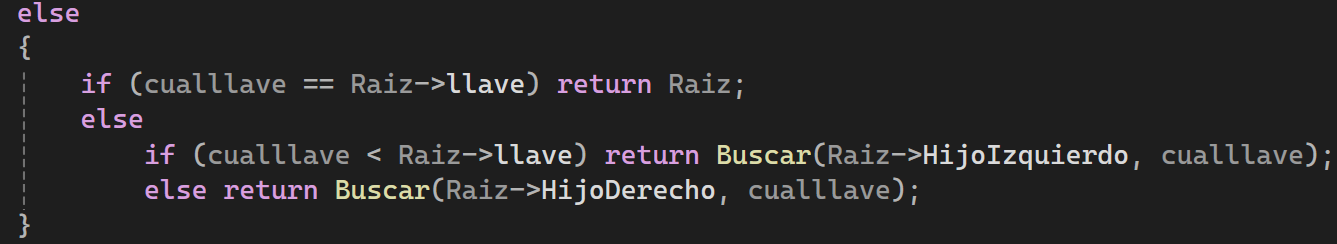


También se puede analizar que estamos con otra función recursiva por lo que podemos analizar su caso base. Este caso se puede dividir en 2 casos:

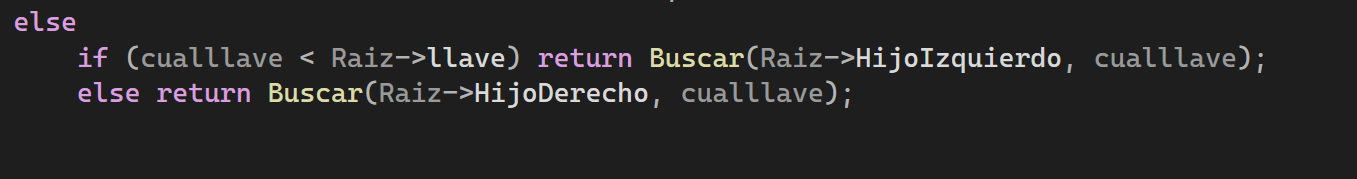
1. Si la dirección de la Raiz es Nula se retorna Nulo ya que no hay con quien comparar. Esto ocurre cuando el árbol está vacío o cuando se sale de rango.



2. Este caso base ocurre cuando se encuentra un elemento igual al que se está buscando



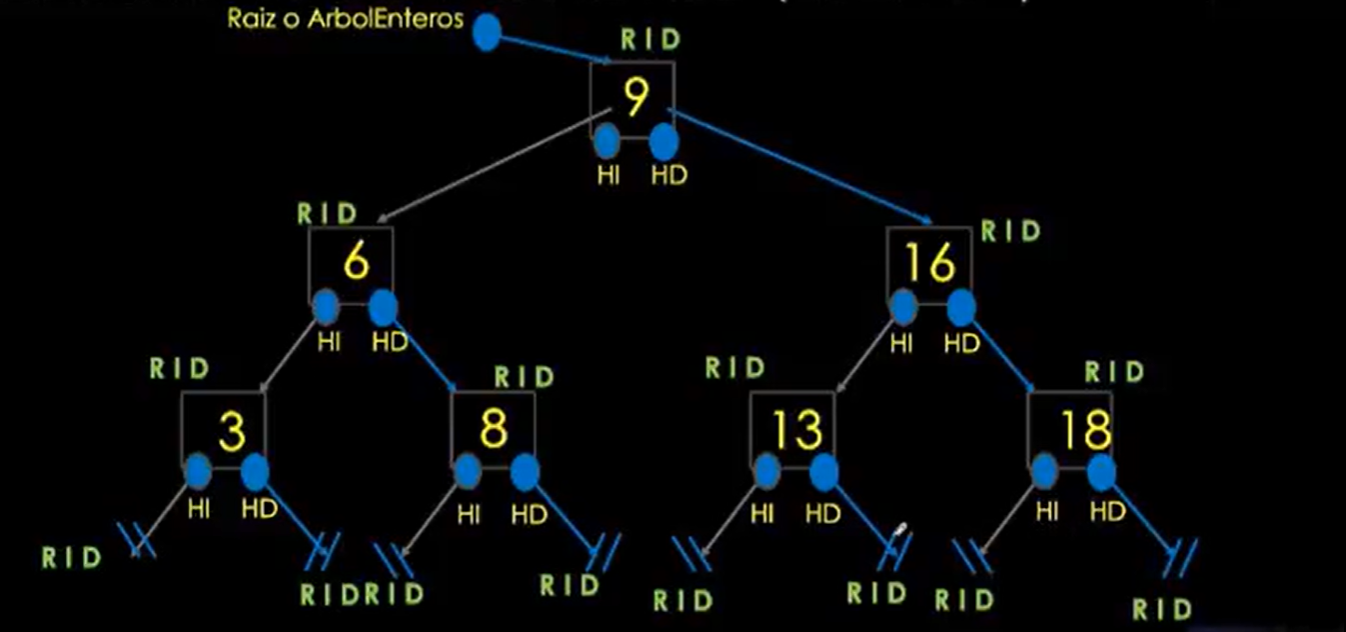
Por otro lado el caso recursivo que esta muy cerca del caso base 2 ocurre cuando la Raiz no es Nula pero tampoco se ha encontrado una raíz igual a esta esta, dependiendo del valor de cualllave y la Raiz actual se empieza a bajar por el HI o por el HD y empieza el ciclo recursivo.



**Algoritmo de recorrido RID-preorden**

El algoritmo más “egoísta” de todos. Las siglas de RID se refieran a Raiz-Izquierda-Derecha.

Cuando se habla de un recorrido se refiere al orden en el que se van a mostrar los elementos por ejemplo al tener la siguiente raíz:



Se puede decir que el orden de la raíz corresponde a RID-preorden de 9 6 3 8 16 13 18

¿Por qué?

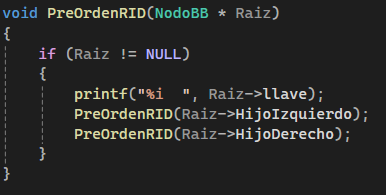
Empecemos con la raíz 9, se cumple la R del RID de 9, luego pasamos al lado izquierdo por lo que nos vamos a 6 aquí se cumple la R del RID de 6, luego pasamos al lado izquierdo y obtenemos 3 aquí se cumple la R del RID de 3 y los siguientes nodos son NULL.

Como ya se cumplió la I del RID de 6 entonces se pasa al lado derecho, que es 8 y así se cumple la de D del RID de 6.

Todo este proceso permite que se logre hacer la I del RID de 9 por lo que se pasa al lado D de ese mismo RID. El lado derecho del RID de 9 es 16, se cumple la R del RID de 16, se pasa al lado izquierdo que es 13, con esto se cumple la I del RID de 16 y finalmente para cumplir el RID completo de 16 se llega a 18.

**Análisis de código PreOrdenRID**

Este código no devuelve nada, sino que imprime valores y además es recursivo.

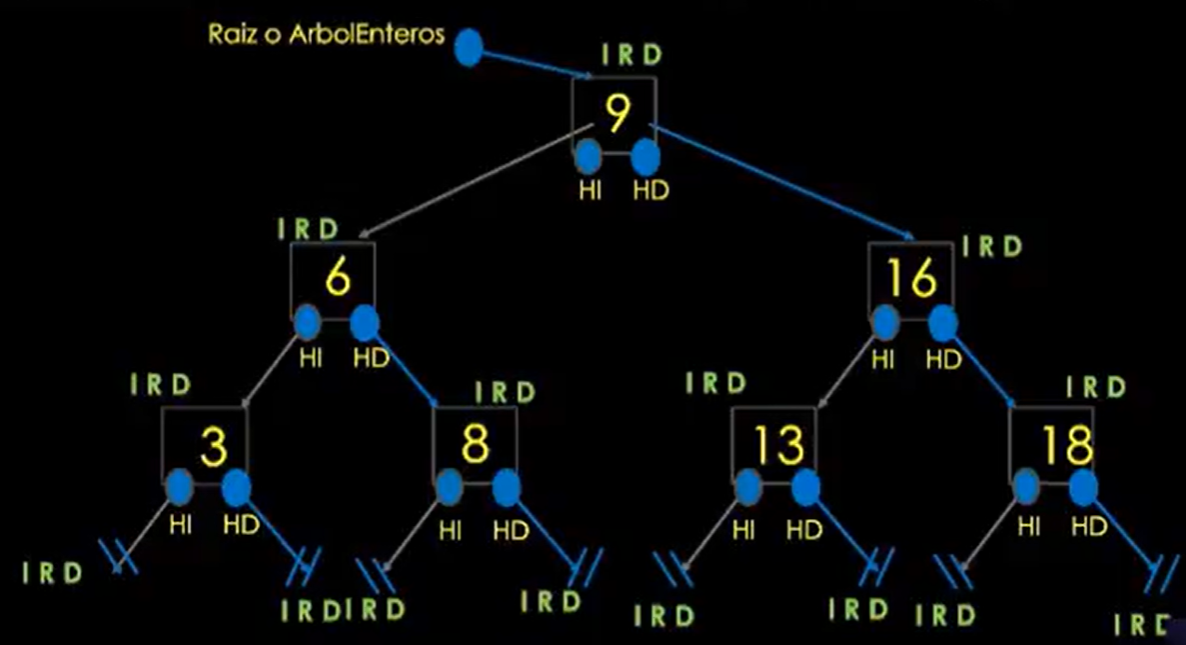


El código recibe la Raiz en un parámetro por valor, eso no importa ya que no se va a modificar nada. Si la Raiz no es NULL se ejecuta el código, es decir esta es el caso continuidad, esto también se puede interpretar como un caso donde no se ha llegado al fondo del árbol. Lo que se hace dentro de la raíz es que se hace el mismo proceso para el HijoIzquierdo y el HijoDerecho, funcionando igual a lo representado hace poco.

Si la raíz es NULL no se hace nada y empieza el backtraking.

**Algoritmo de recorrido IRD-enorden**

Aquí los elementos se muestran de forma ordenada. Las siglas de IRD representan lo mismo que en el algoritmo pasado Izquierda Raíz Derecha.

****

Analizando este mismo árbol se puede decir que el orden de la raíz corresponde a IRD =3 6 8 9 13 16 18

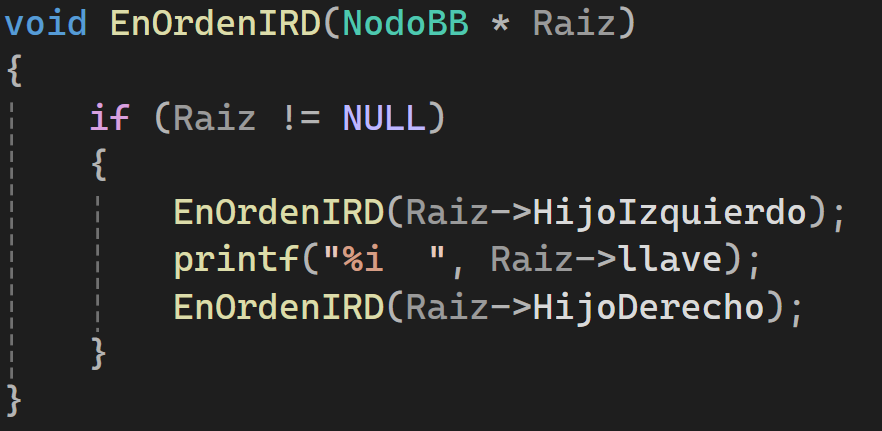
¿Por qué?

En este caso al empezar con la raíz 9 no se muestra esta raíz inmediatamente sino que por instrucción se pasa a 6, ósea la I de IRD de 9, en el 6 también se pasa a la izquierda cumpliendo la I de ese IRD, por lo que se muestra un 3, como 3 ya no tiene más Hijos entonces se puede decir que la I del IRD de 3 se cumplió, entonces se muestra 3 y como no tiene un HijoDerecho ya se cumplió el RID de 3. Esto también significa que se cumple la I del IRD de 6 por lo que se muestra el 6 y luego se pasa al 8 que es la D del IRD de 6, se muestra el 8 y con esto se cumple la D del IRD de 6.

Con todo esto se cumplió la I del IRD de 9, por lo que se puede mostrar 9 y luego pasar a la D del IRD de 9, aquí se hace el mismo proceso. Se muestra 16, 16 pasa a 13 que corresponde a su I, como 13 no tiene más hijos se muestra 16 y solo queda cumplir el D del IRD de 16 que corresponde a 18 y como este no tiene más hijos se terminó con el proceso.

**Análisis del código EnOrdenIRD**

Este código es muy parecido al analizado anteriormente, tiene la misma condicion, es recursivo y tampoco devuelve nada.

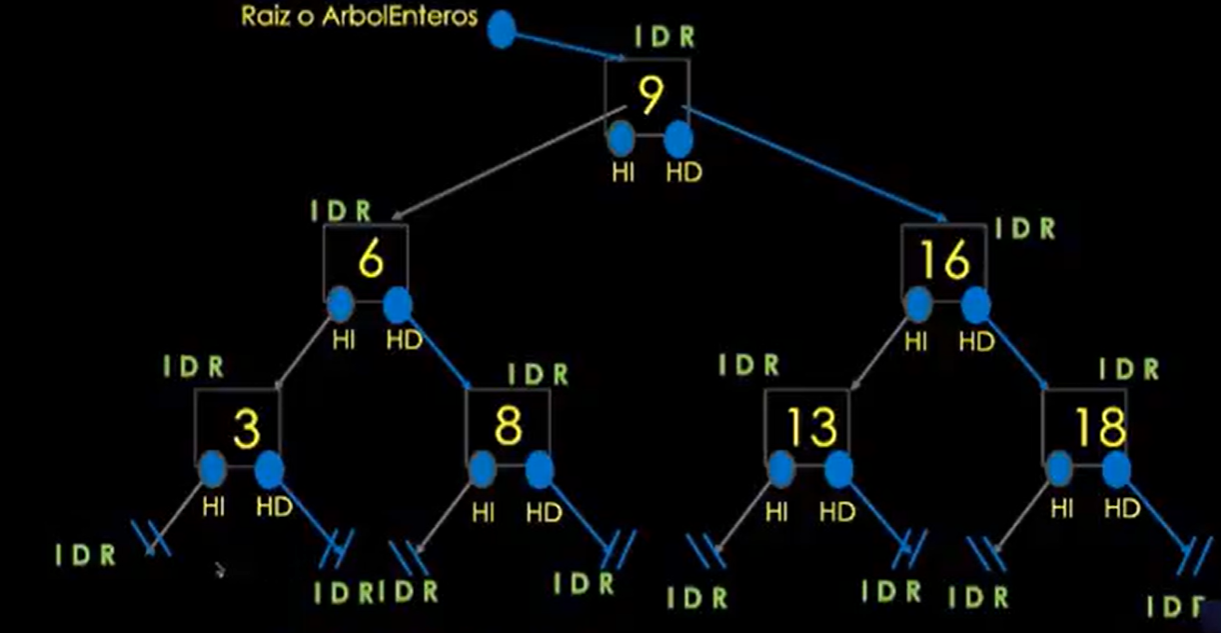


El else(invisible) y el if funcionan igual, lo único que cambia es el orden en el que se dan las cosas, cosa que resulta curiosa pero tiene sentido.

En el código pasado el orden era RID por lo que se imprimía la raíz, y se llamaba la función primero con el hijo Izquierdo y luego con el hijo Derecho. Aquí funciona parecido pero con el orden IRD.

**Algoritmo de recorrido IDR-posorden**

Se puede ver como un algoritmo en el que se muestran los elementos de abajo hacia arriba. Las letras de IDR representan lo mismo que en los ejemplos anteriores.

****

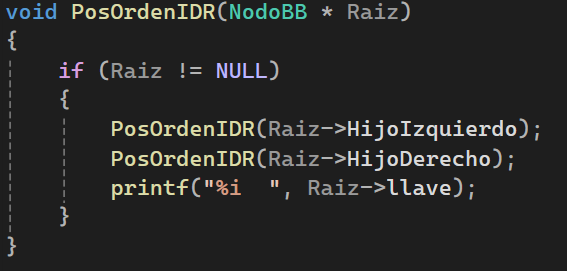
Analizando este mismo árbol se puede decir que el orden de la raíz corresponde a IDR =3 8 6 13 18 16 9

Al empezar en el 9 se pasa al 6(el lado I del 9), lo mismo para el 6, se pasa al 3, como este ya no tiene hijos entonces volvemos al 6 ya que se cumplió el I del 6, pero todavía no se muestra la raíz sino que pasamos al lado D ósea el 8, como el 8 no tiene hijos se muestra y se cumple el D del 6 por lo que se muestra la R y se cumple el I del 9.

Ahora pasamos al lado D del 9, pasamos al 16, el I del 16 es 13 y como este no tiene hijos se cumple el I del 16, ahora pasamos al D del 16, que corresponde a 18, como el 18 no tiene hijos se cumple la condición D del 16 y por lo tanto se cumple el D del 9 para finalmente mostrar el 9 para cumplir la condición R de 9.

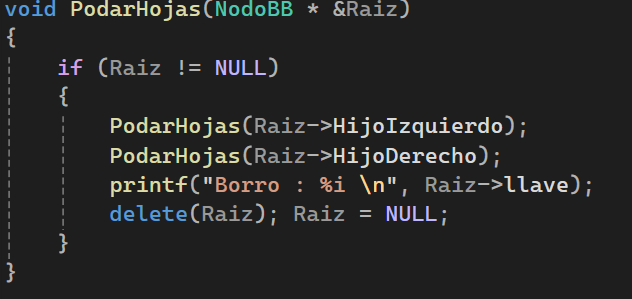
**Análisis del código PosOrdenIDR**

Esta función es muy parecida a las 2 anteriores.E igualmente, el único cambio es el orden de impresión de los elementos o el orden de llamado de cada función para cada elemento.



**Algoritmo y código de PodarHojas**

Este algoritmo como dice su nombre se encarga de eliminar los últimos elementos del árbol, es decir, elimina los elementos de abajo hacia arriba por lo que utiliza el orden PosOrdenIDR.

El algoritmo en el árbol ya usado elimina los elementos en este orden: 3 8 6 13 18 16 9. 

Como se puede ver el algoritmo es el mismo que en el de PosOrdenIDR con el único agregado de que aquí se eliminan los elementos en vez de imprimirlos. Si hay un print pero esto es para ver los elementos que se van borrando.

Recordemos que para eliminar una variable anónima se utiliza un delete(Raiz) y un Raiz= NULL.