Árboles

Grafo:G(V,E) es un conjunto de vértices o nodos V = {v1,v2,...,vn} y de aristas E donde cada arista es una parejas de vértices (vi,vj)

Árbol: Grafo conexo y acíclico en donde un y sólo un nodo es designado como la raíz. La raíz es el punto de entrada a el árbol

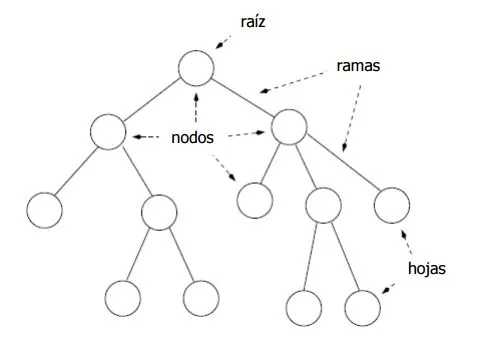
Camino de vi a vj (Camino(vi,vj)): Es un recorrido de vértices consecutivos a través de una secuencia de aristas o ligas. La longitud del camino es el nu ́mero de ligas que lo componen

Un grafo es conexo si existe un camino entre cualquier par de vértices

Algunas características:

* Existe un sólo un camino de la raíz a cualquier nodo del árbol (definición alterna)
* Ancestro: un nodo vi es ancestro de un nodo vj si vi aparece en un camino de la raiz a vj
* El nodo raiz es el u ́nico nodo que no tiene ancestros
* Un nodo vi es sucesor o hijo de un nodo vj si existe un liga (vj,vi) en E y vj es ancestro de vi
* Un nodo vi es descendiente de otro nodo vj si vj es ancestro de vi
* Un nodo hoja es aquel que no tiene sucesores
* Un nodo interior es aquel que no es ra ́ız ni hoja
* Estructura Jerárquica
* El nivel de un nodo vi es la longitud del Camino(raíz,vi)
* La altura del ́arbol es la longitud del camino m ́as largo en el ́arbol
* El orden de un un ́arbol es el m ́aximo nu ́mero de hijos que tiene un nodo. Un ́arbol general no tiene restricci ́on. Un ́arbol n-ario tiene un m ́aximo de n hijos por nodo
* Dadoun ́arbolG=(V,E),un subárboles un ́arbolG′ =(V′,E′) dondeV′ ⊆V yE′ ⊆E

**ÁRBOLES**

****

**Un árbol es:**

-Una estructura de datos no lineal.

-Un grafo (es una forma para modelar relaciones

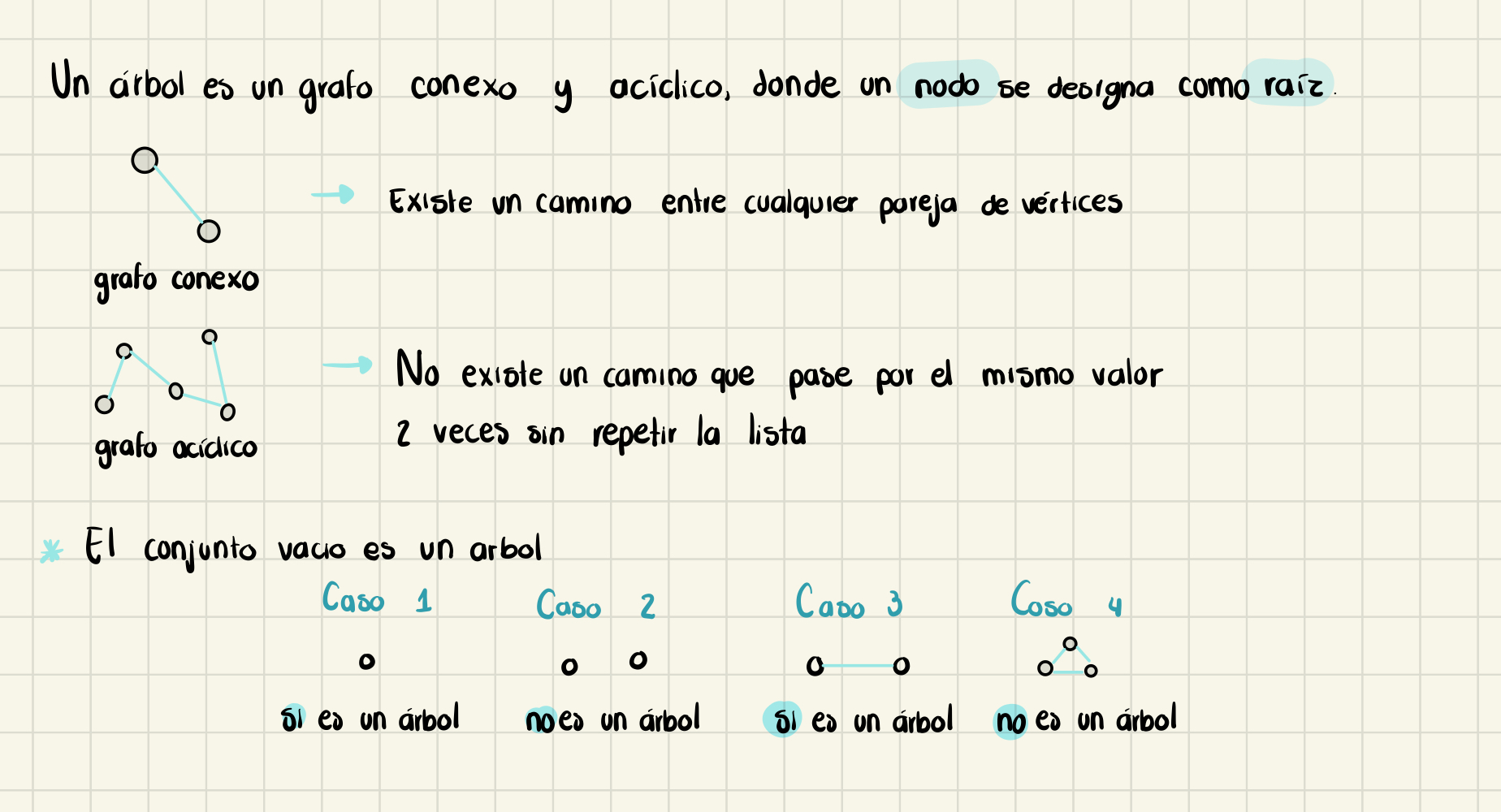
-Un grafo **conexo** y **acíclico**, donde un nodo (un nodo es un elemento de información que reside en el árbol) se designa como raíz.

-Es un objeto que comienza con una raíz y se extiende en varias ramificaciones o líneas, cada una de las cuales puede extenderse en ramificaciones hasta terminar, finalmente en una hoja.

Lo podemos ver matemáticamente como dos conjuntos:

(V,E) vértices V={v1,v2,v3,...,vn)

aristas E={(vi,vj),(vj+1,vi),...}



Todos los nodos, excepto la raíz, tienen una sola línea de entrada (el nodo raíz no tiene ninguna).

**Recorrer** un nodo es pasar por ese nodo sin procesarlo.

**Visitar** un nodo es procesarlo, procesar es hacer algo, como imprimir su contenido.

Características de los árboles:

\*Se dice que todos los nodos que son descendientes directos (**nodos hijo**) de un mismo nodo (**nodos padre**), son **hermanos**.

\* Todo nodo que no tiene ramificaciones (hijos), se conoce con el nombre de **terminal/hoja/externos**.

\* Los **antepasados** de un nodo son todos los nodos a través de los cuales hay que pasar para llegar a ese nodo desde el nodo raíz.

\* Todo nodo que no es raíz, ni terminal u hoja se conoce con el nombre de **interior/internos**.

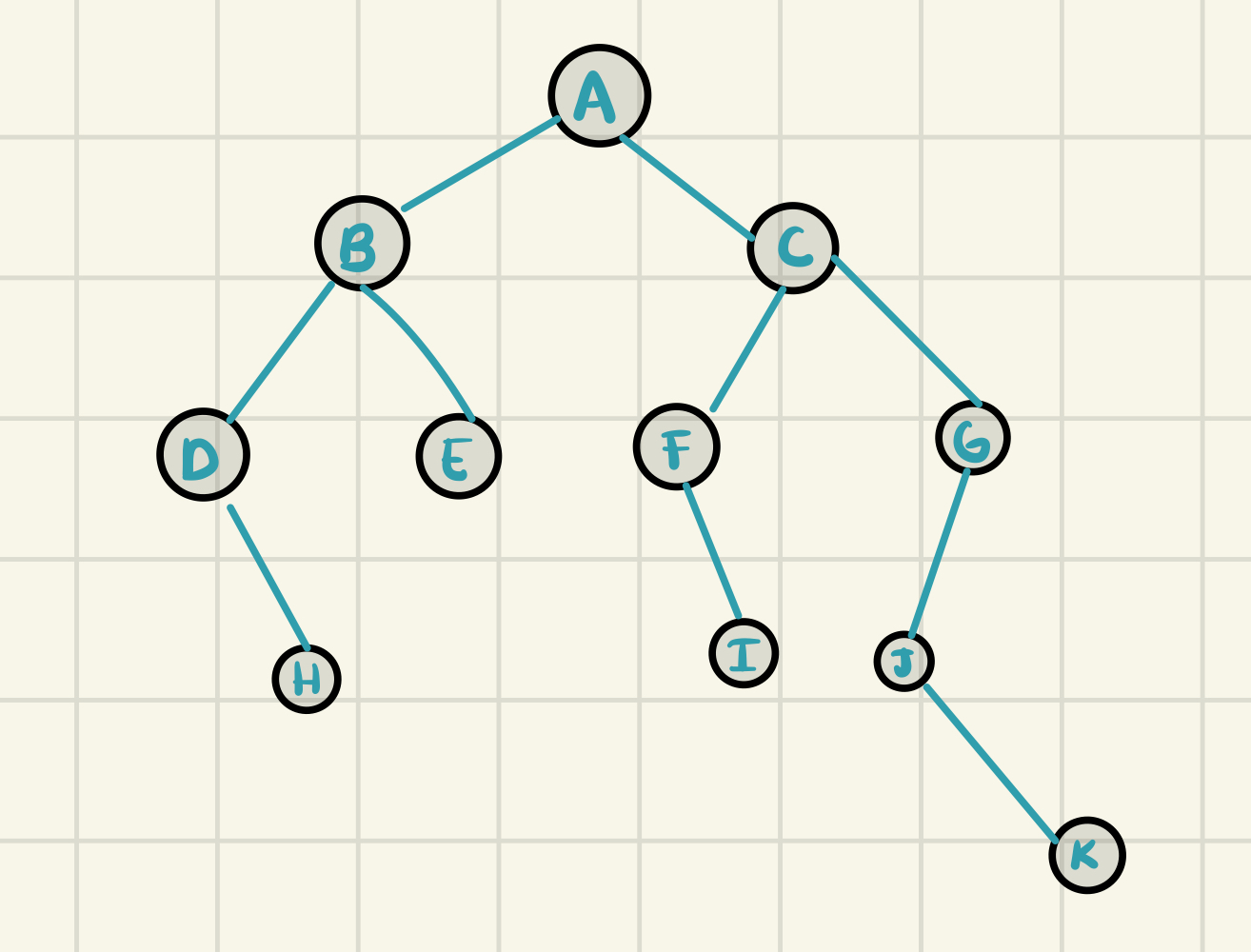
\* **Grado** es el número de descendientes directos de un determinado nodo. **Grado del árbol** es el máximo grado de todos los nodos del árbol.

\* **Nivel** es el número de arcos que deben ser recorridos para llegar a un determinado nodo.

\* **Altura** del árbol es el máximo número de niveles de todos los nodos del árbol.

\* El **orden de un árbol** es la cantidad máxima de hijos que puede tener cualquier nodo de ese árbol. Un **árbol n-ario** es un árbol en el que cada nodo puede tener un máximo de n hijos (es decir, usar ese término es equivalente a decir que es un árbol de orden n).

**Pre orden:**

1. visito nodo.
2. recorro hijo izquierdo.
3. recorro hijo derecho.

secuencia: A,B,D,H,E,C,F,I,G,J,K

**In orden:**

1. recorro hijo izquierdo.
2. visito nodo.
3. recorro hijo derecho.

secuencia: D,H,B,E,A,F,I,C,J,K,G

**Post orden:**

1. recorro hijo izquierdo.
2. recorro hijo derecho.
3. visito nodo.

secuencia: H,D,E,B,I,F,K,J,G,C,A

**Por nivel:**

1. primero visito el nivel 1.
2. luego el nivel 2…

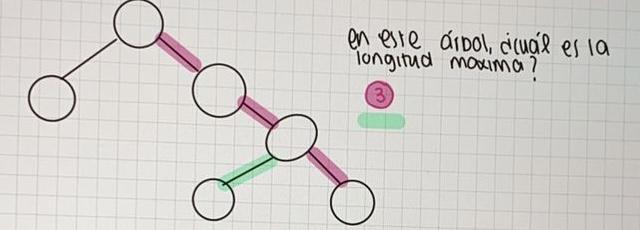
secuencia: A,B,C,D,E,F,G,H,I,J,K

**Árboles binarios:**

Los **árboles binarios** son árboles en los que cada nodo tiene un máximo de dos nodos hijo (se suele usar más este término que árbol de orden 2 o árbol 2-ario). Otras excepciones comunes son **árbol ternario** (en lugar de orden 3 o 3-ario) y **árbol cuaternario** (en lugar de orden 4 o 4-ario).

Los nodos hijo de cualquier nodo en un árbol binario ordenado se denominan el **hijo izquierdo** y el **hijo derecho**.

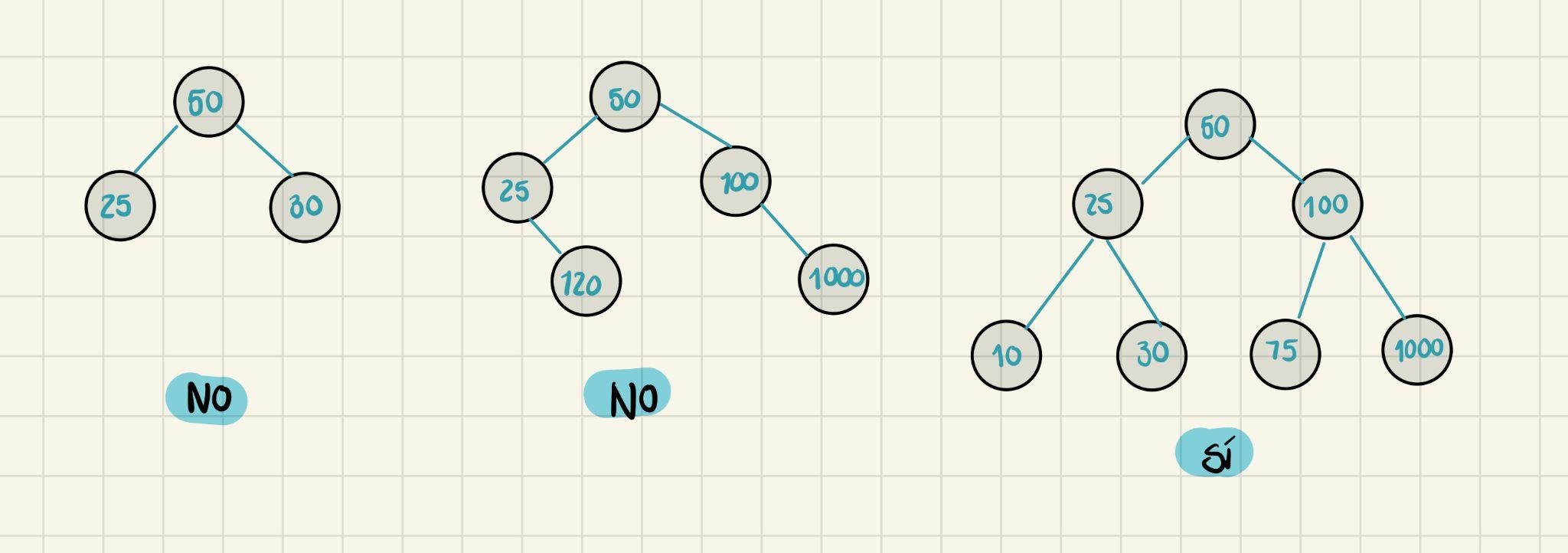
**Altura del árbol:**

****

Aunque en el penúltimo nodo tenga dos hijos, seguirá siendo de altura 3. Llamo a la izquierda y veo el resultado, después llamo a la derecha y comparo, el que sea mayor le sumo +1 y ese resultado es la longitud del papá (dónde se une el lado izquierdo y derecho).

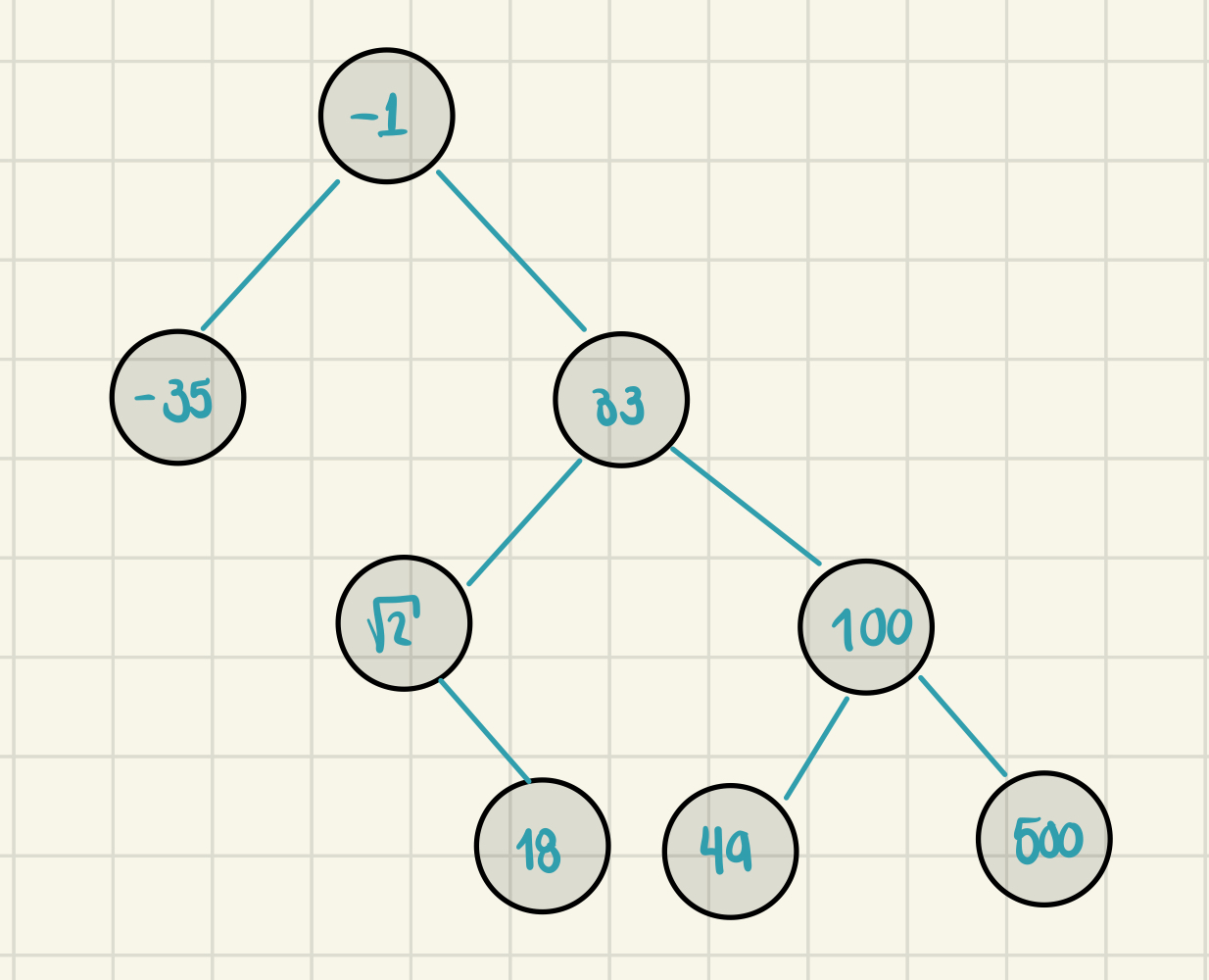
**Árboles binarios de búsqueda:**

* Almacenan por nodo un elemento comparable.
* Para todo nodo con elemento E todos los elementos del árbol menores o iguales al elemento E están en su subárbol izquierdo y los mayores en el derecho.



Ejemplo:

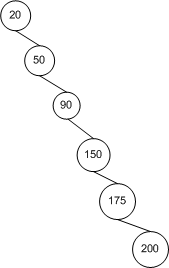
-1, 33, √**2**, 100, 500, -35, 49, 18



(Completar ejercicio)

Miércoles 15 Septiembre

**¿Cuánto me tardo (en notación asintótica) en insertar datos?**

El peor de los casos el árbol binario de búsqueda se construye de un solo lado por lo que tiene que recorrer un largo camino para insertar el valor. Ej:

**¿Cuánto tiempo toma eliminar un dato del árbol binario de búsqueda?**

**Código Borrar tiene que considerar 3 casos**

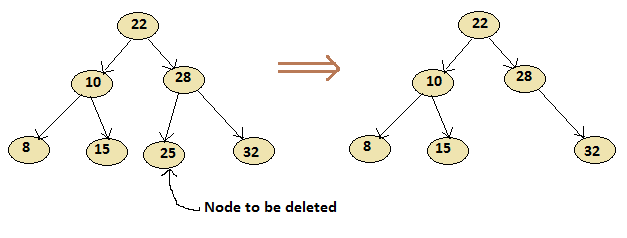
| **public void borra(T elem){  NodoBB<T> actual=busca(elem);  if(actual==null)  return;  if(actual.getDer()==null && actual.getIzq()==null )//caso hoja  actual=eliminaHoja(actual);  else if(actual.getDer()==null || actual.getIzq()==null)//un solo hijo  actual=eliminaUnHijo(actual);  else //dos hijos  actual=elimnaDosHijos(actual);   cont--;  }** |
| --- |

Explicación de los casos semánticamente y con código

**Caso 1 NINGÚN HIJO (HOJA):**

-raíz

-no raíz



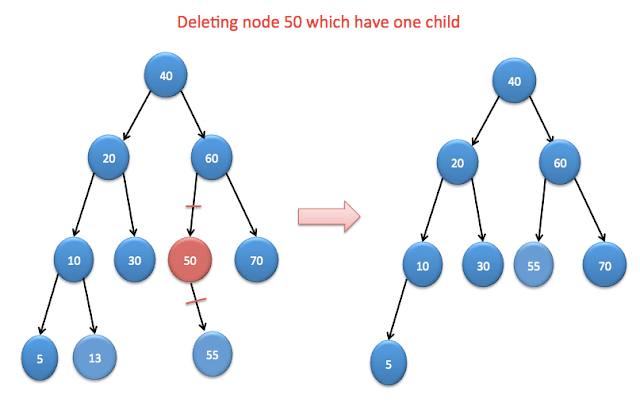
| private NodoBB<T> eliminaHoja(NodoBB<T> actual){   if(actual==raiz){  raiz=null;  return null;//o actual?  }  if(actual.getPapa().getIzq()==actual)  actual.getPapa().setIzq(null);  else  actual.getPapa().setDer(null);  return actual; |
| --- |

}

**Caso 2 SOLO UN HIJO:**

-raíz

-colgar hijo al abuelo

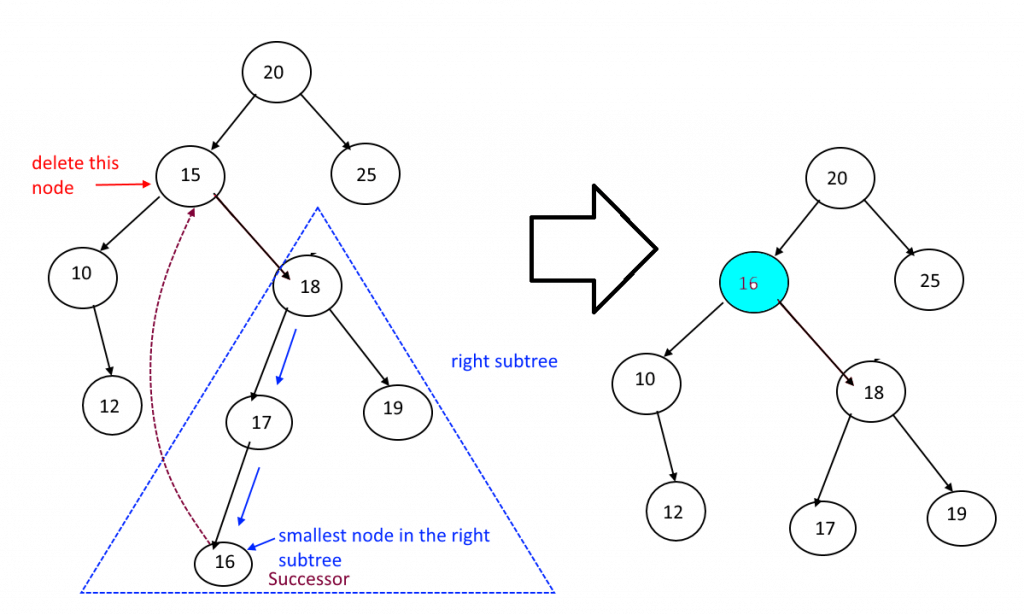


| private NodoBB<T> eliminaUnHijo(NodoBB<T> actual){  NodoBB<T> hijo;  if(actual.getDerecha()==null)  hijo =actual.getIzq();  else  hijo = actual.getDer();  if(actual==raiz)  raiz = hijo;   actual.getPapa().cuelga(hijo);   return hijo; } |
| --- |

**Caso 3 DOS HIJOS:**

-Encontrar el nodo en donde está

-Sustituir su valor por el valor del sucesor inorden



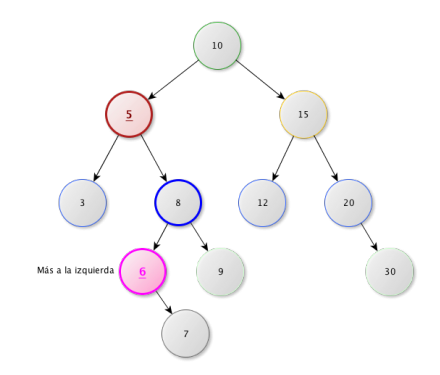
Para eliminar dos hijos también existen dos casos a considerar:

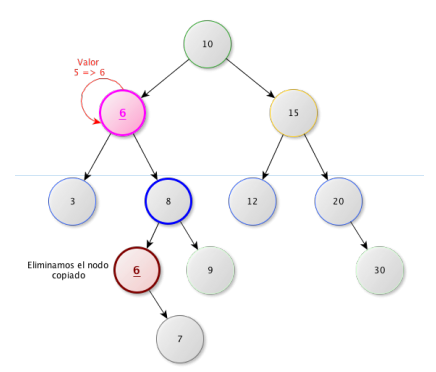
| private NodoBB<T> eliminaDosHijos2(NodoBB<T> actual){   NodoBB<T> sucInorden=actual.getDer();   while (sucInorden.getIzq()!=null)  sucInorden=sucInorden.getIzq();  actual.setDato(sucInorden.getDato);  sucInorden.papa.izq=sucInorden.der; |
| --- |

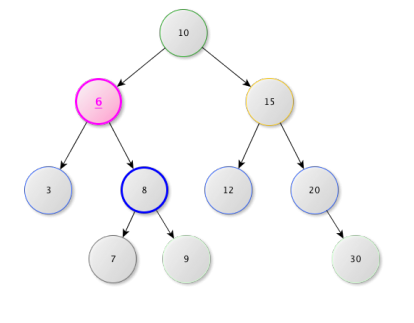
Cuando hay 2 hijos tenemos que considerar que hay que poner algo en la raíz, por lo que tenemos buscar que poner ahí que corresponda

* Tenemos que considerar que existen 2 casos en el caso de que sea la raiz y tenga dos hijos

1. Que al final quede parejo y no se tenga que poner algún null (todos tienen hijos)
2. Que al final un nodo sólo tiene un hijo, por lo tanto el padre tiene que apuntar a null







Algoritmos de búsqueda en árboles

Para que estos sean eficientes es importantes que la altura del árbol sea lo más chico (es decir, que los nodos del árbol tengan la mayor cantidad de hijos posible)

Árboles AVL (Adelson-Velskii y Landis)

* Es un árbol binario de búsqueda
* Factor de equilibrio de un nodo es la diferencia entre la altura de su sub-árbol derecho e izquierdo (puede ser: -1, 0, 1)

H(D) - H(I)

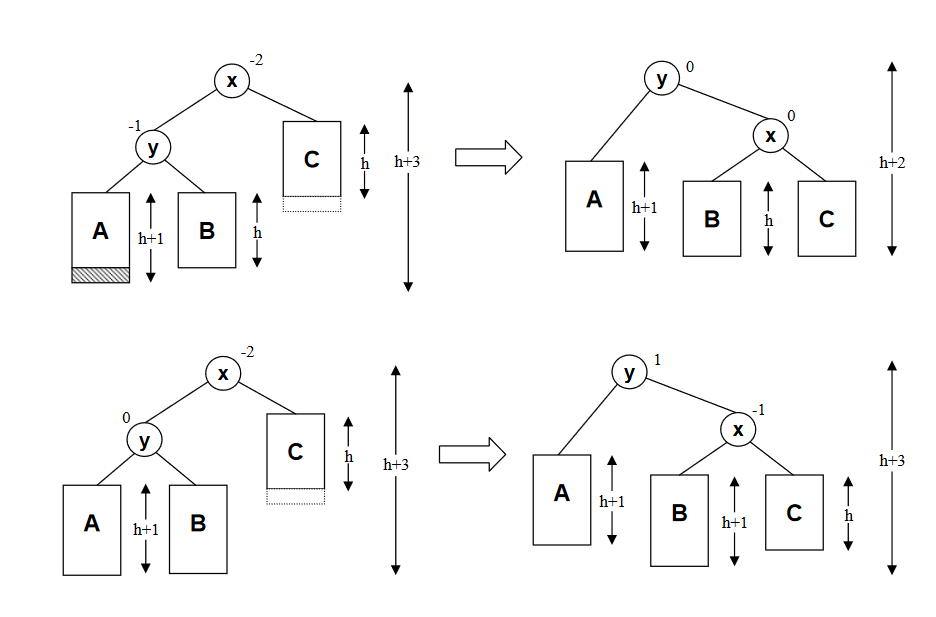
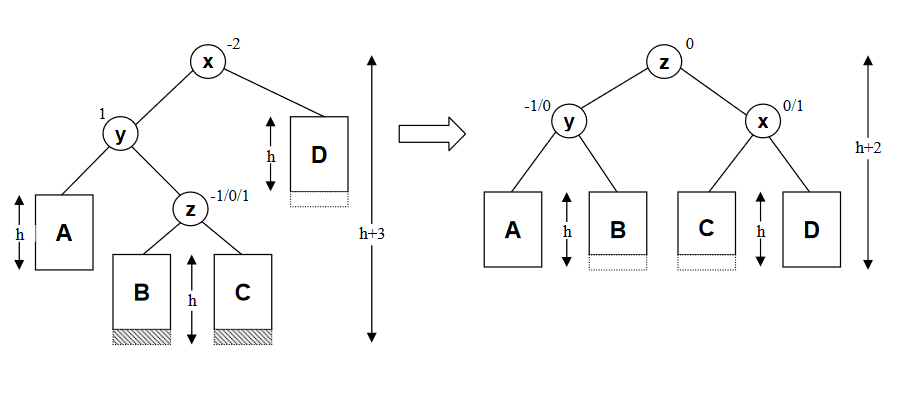
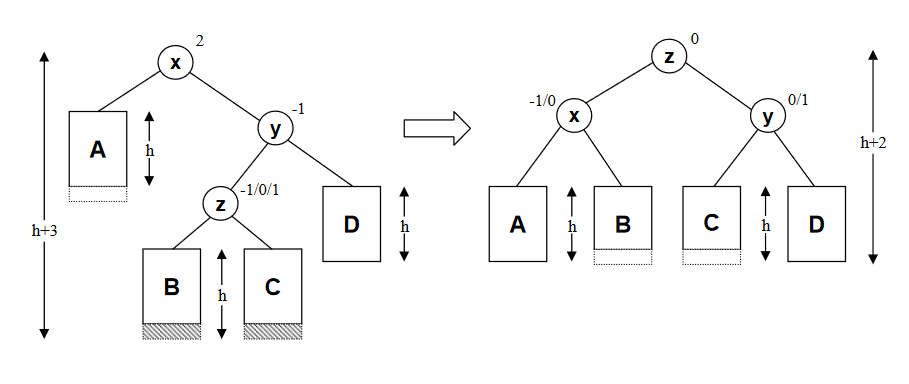
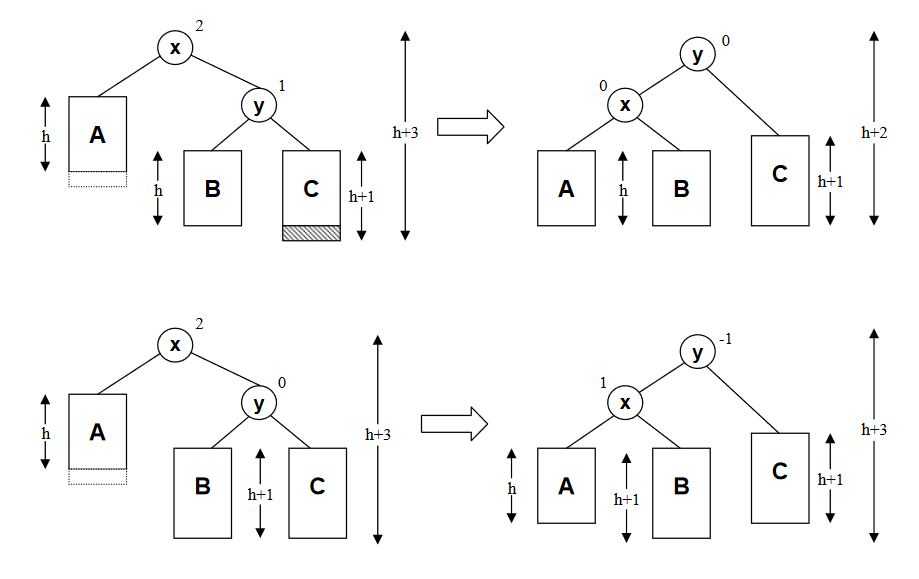
* Para un árbol AVL el factor de equilibrio es el fe de todos sus nodos es |fe| < 2
* Si hay 2 nodos consecutivos del mismo lado, ya no es un árbol AVL

¿Cómo se hace?

Después de cada operación de inserción o borrado voy a utilizar la fe de los nodos relevantes y si encuentro algún |fe| = 2 voy a **rotar**

**Rotaciones**

Sirven para poder balancear los árboles y hacerlo AVT

1. **Izquierda - Izquierda**
   1. Aplica si el padre tiene fe = -2 y el hijo izquierdo fe = -1 o 0
   2. 
2. **Izquierda - Derecha**
   1. Si el padre tiene fe = -2 y el hijo izquierdo fe = 1
   2. 
3. **Derecha - Izquierda**
   1. Padre tiene fe = 2 e hijo derecho tiene fe = -1
   2. 
4. **Derecha - Derecha**
   1. Padre fe = 2 e hijo derecho tiene fe = 1 o 0
   2. 

Insertar en un árbol AVL (explicación semántica)

1. Insertar el elemento como hoja, siguiendo estrategia de árboles binarios de busqueda (ABB)
2. Balancear

2.1) Recorrer a partir de este nuevo nodo hacia la raíz (apuntadores al papá) ajustando los factores de equilibrio.

-si vengo del lado derecho le sumo 1 al papá

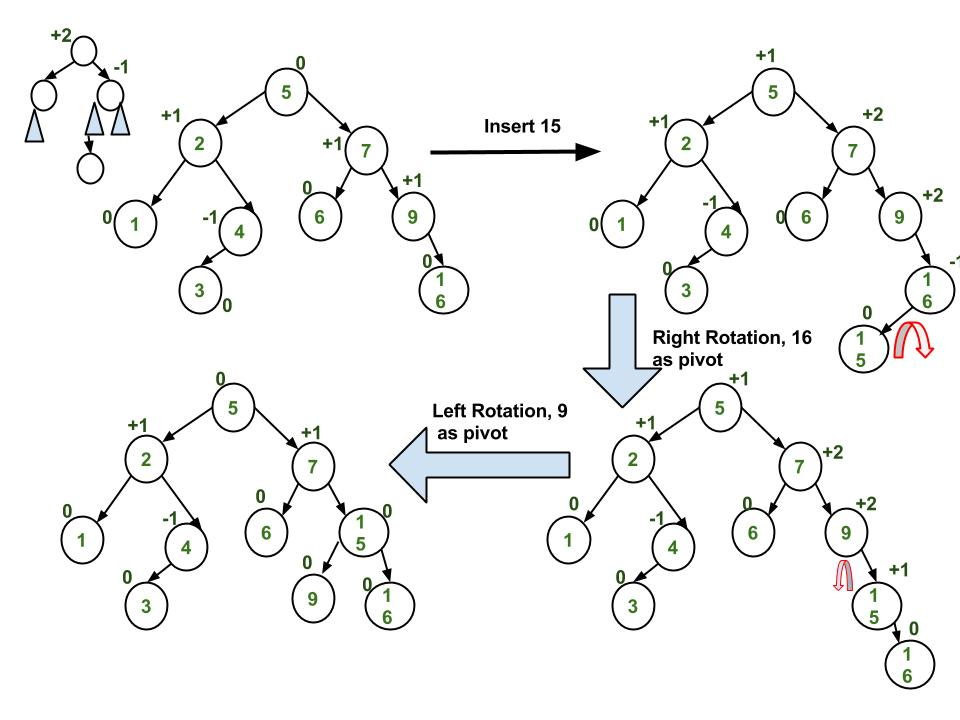
-si vengo del lado izquierdo le resto uno al papá.

2.2) Hasta que:

-LLegué a la raíz

-el Factor de equilibrio se volvió cero

-el Factor de equilibrio=2 y ejecuto una rotación.



Borrar en un árbol AVL (explicación semántica)

1. Busco el nodo
2. Eliminar de acuerdo al algoritmo de árboles binarios de búsqueda (ABB)

2.1) Viajo hacia la raíz actualizando los factores de equilibrio.

-si vengo de la izquierda le sumo 1 al papá

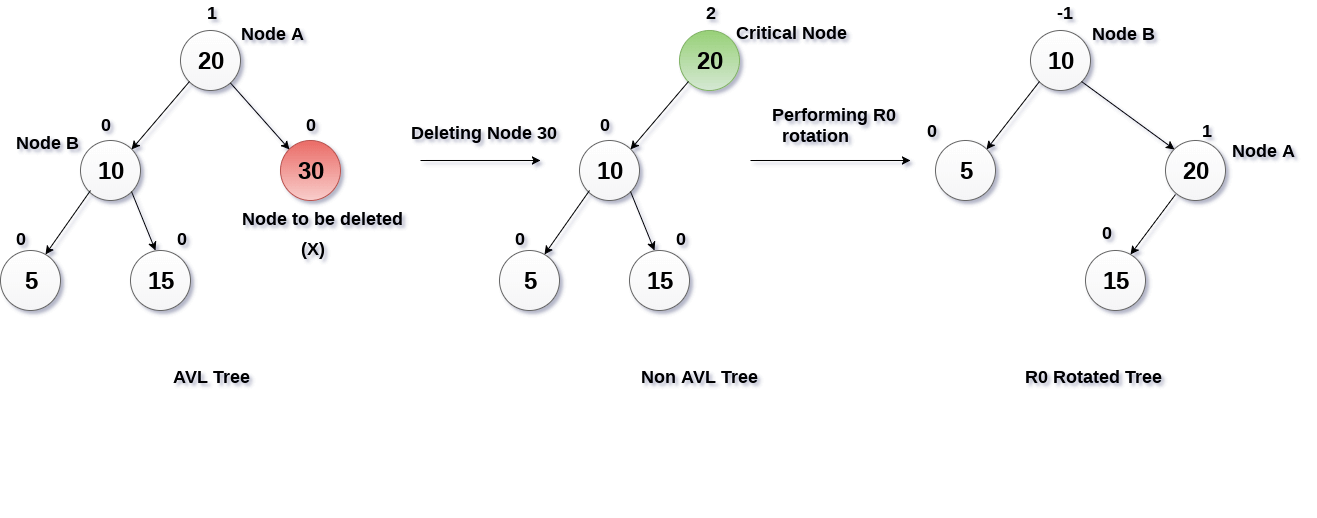
-si vengo de la derecha le resto 1 al papá

-si el factor de equilibrio = 2 efectuar una rotación y continuar el viaje a la raíz

2.2) Condición de parar

-llegué a la raíz

-el factor de equilibrio paso de 0(cero) a 1 ó -1



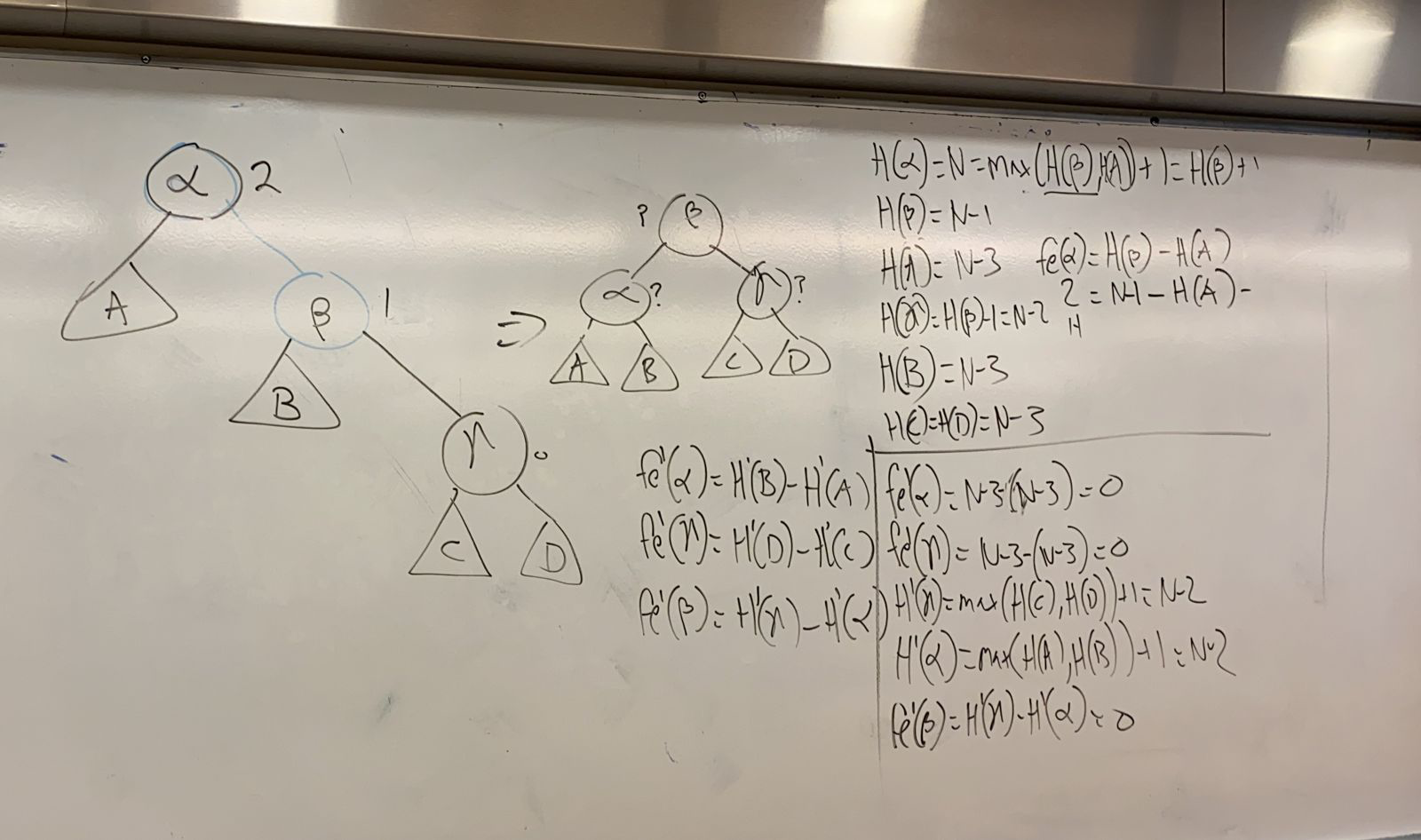
**ROTACIONES DE UN ÁRBOL AVL**

Nuestra función de rotar recibe el nodo cuyo (fe) factor de equilibrio==2 y regresa la nueva raíz del árbol.

1. Identificar (fe) y el tipo de rotación
2. Se recomienda nombrar nodos auxiliares con nombres familiares para hacer la rotacion usando el modelo que conocemos.

Se programará un cuelgaIzq y un cuelgaDer en los casos de que el nodo papá sea o no sea nulo, es decir, al hacer las rotaciones NO se debe olvidar actualizar los padres de los nodos rotados y por lo tanto si el último hijo es null entonces no se actualiza el papá.

Ejemplo



La diferencia entre un árbol binario de búsqueda y un AVL es que el AVL sí está balanceado, de este modo la búsqueda de un dato en AVL siempre es “log(n)” mientras que en los árboles bin de búsqueda en el peor de los casos se tarda ”n”.

Un árbol AVL garantiza, hasta cierto punto, que todos los niveles estén llenos antes de “iniciar” un nuevo nivel

----Se pasaron las notas de Heaps al documento de Heaps