Árboles AVL

Clase 09

IIC 2133 - Sección 4

Prof. Sebastián Bugedo

Sumario

Segundo Acto: Diccionarios

Árboles y tablas de hash

Playlist 2

Playlist: DatiWawos Segundo Acto

Además sigan en instagram:
@orquesta_tamen

Repaso: Diccionarios

Definición

Un **diccionario** es una estructura de datos con las siguientes operaciones

- **Asociar** un valor a una llave
- **Actualizar** el valor asociado a una llave
- **Obtener** el valor asociado a una llave
- En ciertos casos, **eliminar** de la estructura una asociación llave-valor

Estamos buscando cómo implementarlos de manera eficiente con una **nueva EDD**

Repaso: Árboles binarios de búsqueda

Definición

Un **árbol binario de búsqueda (ABB)** es una estructura de datos que almacena **pares (llave, valor)** asociándolos mediante punteros según una estrategia recursiva

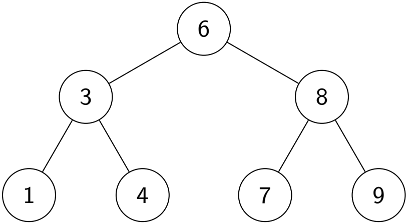
1. Un ABB tiene una **nodo** que contiene una tupla (llave, valor)
2. El nodo puede tener hasta dos ABB's asociados mediante punteros
   - Hijo izquierdo
   - Hijo derecho

y que además, satisface la **propiedad ABB**: las llaves menores que la llave del nodo están en el sub-árbol izquierdo, y las llaves mayores, en el sub-árbol derecho.

Nuestra primera EDD construída para resolver un problema

Repaso: Árboles binarios de búsqueda

Cada sub-árbol es un ABB: se cumple recursivamente la propiedad ABB



y la gracia es que si colapsamos los nodos...

La propiedad ABB garantiza **orden**

Repaso: Operaciones de un ABB

Además, definimos algoritmos para implementar las operaciones que nos interesan

- Búsqueda por valor de llave
- Inserción/actualización del valor asociado a una llave
- Eliminación de una llave y su valor

Todas estas operaciones tienen una complejidad que depende de la **altura** del árbol

Operación de eliminación

La eliminación es un poco más compleja

Si el nodo a eliminar es hoja o tiene solo un hijo

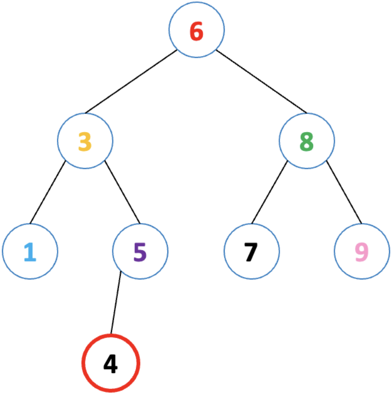
- Lo borramos
- Si tenía un hijo, el hijo lo reemplaza
- Es claro que se mantiene la propiedad ABB

En caso contrario...

¿Se puede reemplazar por otro árbol?

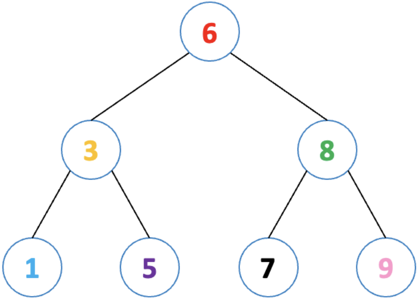
Ejemplo de eliminación

Si queremos eliminar el nodo con llave 4



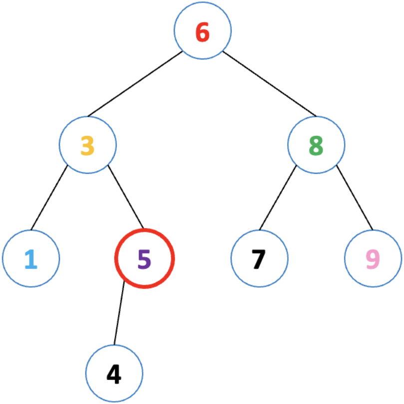
Ejemplo de eliminación

Simplemente se elimina y se preserva la propiedad ABB



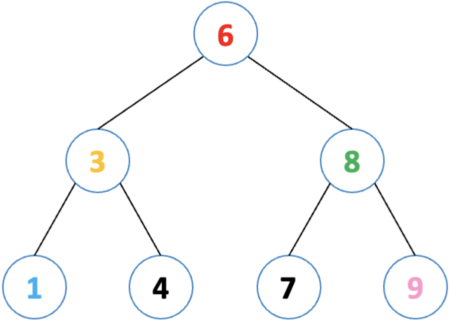
Ejemplo de eliminación

Si queremos eliminar el nodo con llave 5



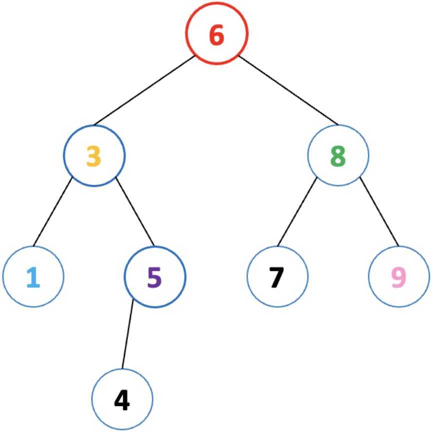
Ejemplo de eliminación

Se reemplaza por su único hijo y se preserva la propiedad ABB



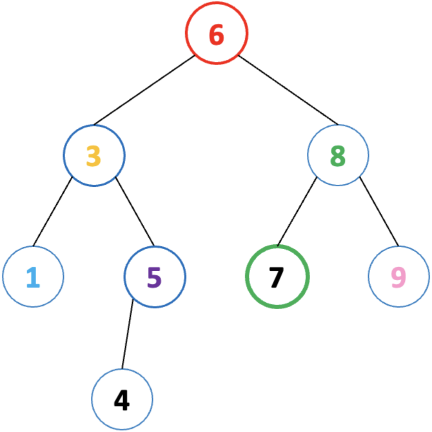
Ejemplo de eliminación

Si queremos eliminar el nodo con llave 6, estamos en problemas



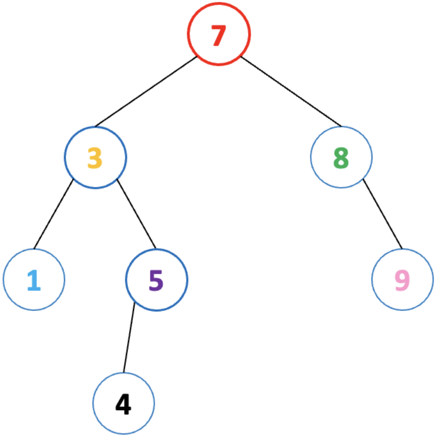
Ejemplo de eliminación

Podemos reemplazarlo por el nodo con llave 7 (**su sucesor**)



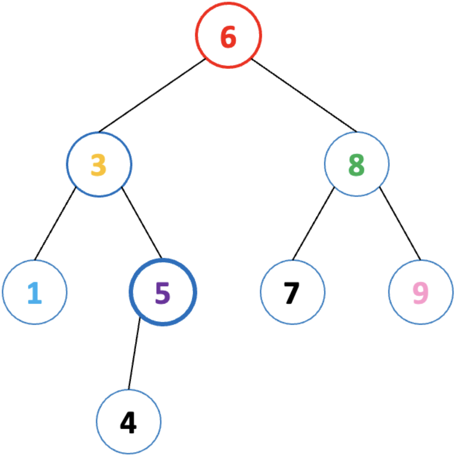
Ejemplo de eliminación

Y dado que no tenía hijos, no hay que hacer más modificaciones



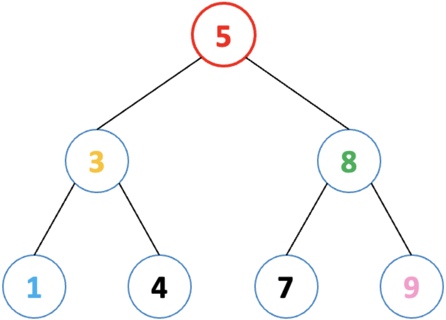
Ejemplo de eliminación

De forma alternativa, podemos reemplazarlo por el nodo con llave 5 (**su antecesor**)



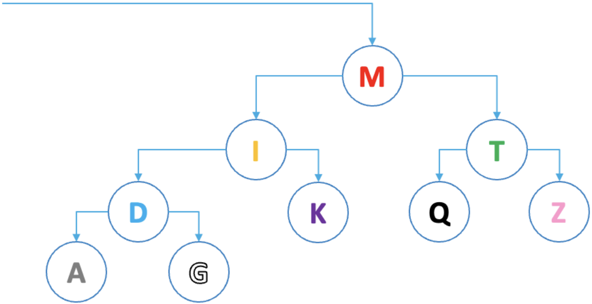
Ejemplo de eliminación

Y reubicamos su hijo con llave 4



Operación de eliminación

Nos interesa encontrar el sucesor/antecesor del nodo extraído




```
Min (A):
    if A.left = ∅ :
        return A
    return Min(A.left)
```

```
Max (A):
    if A.right = ∅ :
        return A
    return Max(A.right)
```

Proponemos el siguiente algoritmo que preserva la propiedad ABB

`Delete` (A, k):

  $(D, p) \leftarrow \texttt{Search}(A, \varnothing, k)$ ▷ `Permite saber el padre de` D

  if $D \neq \varnothing$ **:**

    if *D es hoja* **:** $D \leftarrow \varnothing$ y se elimina la referencia en p

    elif *D tiene un solo hijo H* **:** $D \leftarrow H$ y se actualiza p

    else:

      $R \leftarrow \texttt{Min}(D.right)$

      $t \leftarrow R.right$

      $D.key \leftarrow R.key$

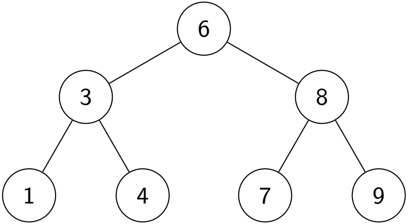
      $D.value \leftarrow R.value$

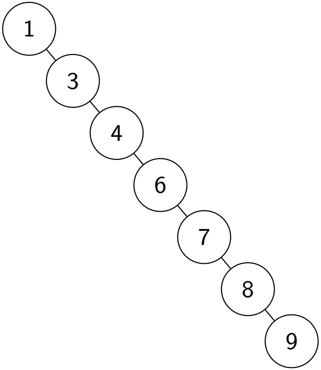
      $R \leftarrow t$

Notemos que al borrar un nodo, se debe eliminar la referencia desde su padre

El problema del balance

Dos ABB que contienen las llaves $\{1, 3, 4, 6, 7, 8, 9\}$





¿Cuáles son las consecuencias de esta diferencia de estructura?

Complejidad de las operaciones

Las operaciones de **búsqueda**, **inserción** y **eliminación** dependen de la estructura del ABB

- Mientras más demoremos en llegar a una hoja, más demorarán
- El tiempo de las operaciones es proporcional a la **altura** del árbol
- Contamos desde la raíz a la hoja más lejana

En un árbol con n nodos, la altura puede ser

- En el peor caso $\mathcal{O}(n)$
- Cuando el árbol es balanceado es $\mathcal{O}(\log(n))$

Hoy estudiaremos una forma de asegurar **balance**

Objetivos de la clase

- ☐ Comprender la importancia de acotar la altura de un ABB
- ☐ Conocer una posible definición de balance
- ☐ Aplicar las rotaciones para rebalancear ABBs luego de inserciones
- ☐ Demostrar que la altura de un árbol AVL es logarítmica en el número de nodos

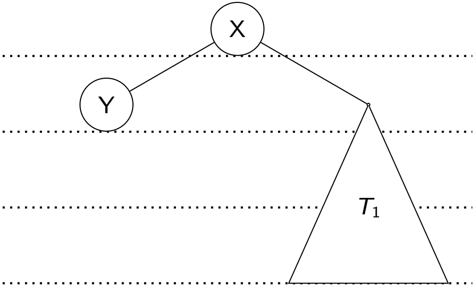
Sumario

Hacia una definición de *balance*

- Nos interesa dar una definición de *árbol balanceado*
- Debe ser recursiva (por la definición que dimos de ABB)
- Eventualmente podemos tener más de una definición

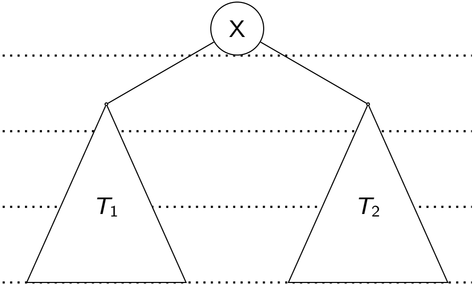
Hacia una definición de *balance*

¿Está balanceado el siguiente árbol? (Consideramos solo la altura de T_1)



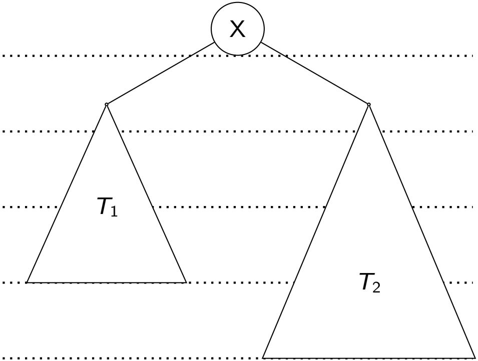
Hacia una definición de *balance*

¿Está balanceado el siguiente árbol?



Hacia una definición de *balance*

¿Está balanceado el siguiente árbol?



Hacia una definición de *balance*

La *propiedad de balance* debe cumplir dos condiciones

1. Asegurar que la altura de un árbol con n nodos sea $\mathcal{O}(log(n))$
2. Ser fácil de mantener

Este último punto es limitante

- Llamaremos al (re)balanceo después de las operaciones
- Queremos complejidad **no mayor** que $\mathcal{O}(\log(n))$
- De esa forma las operaciones seguirán siendo $\mathcal{O}(\log(n))$

Árboles AVL

Adelson-Velsky y Landis propusieron el siguiente modelo de balance

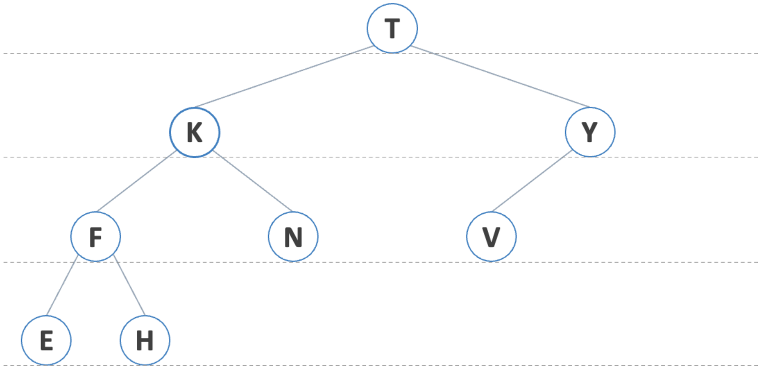
Definición

Un árbol binario de búsqueda está **AVL-balanceado** si

1. Las alturas de sus hijos difieren a lo más en 1 entre sí
2. Cada hijo está **AVL-balanceado**

Árboles AVL: ejemplo

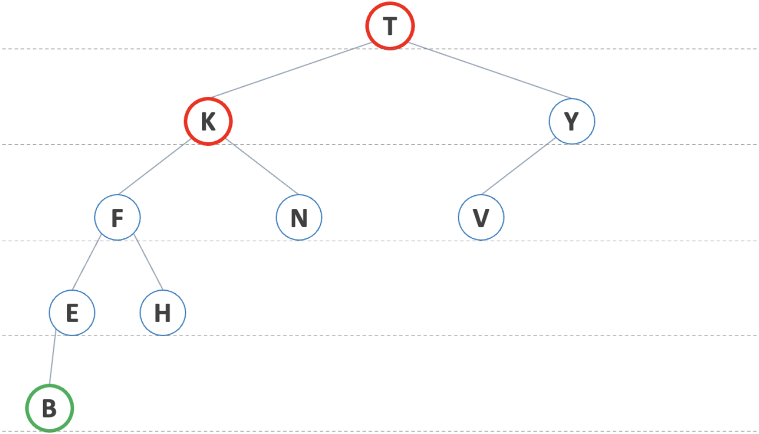
En el siguiente árbol hay una diferencia máxima de 1 entre hermanos



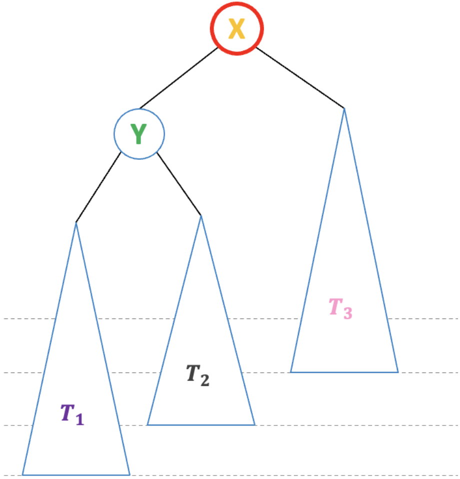
¿Qué pasa al agregar la llave B?

Árboles AVL: ejemplo

Ahora hay desbalance en los nodos T y K



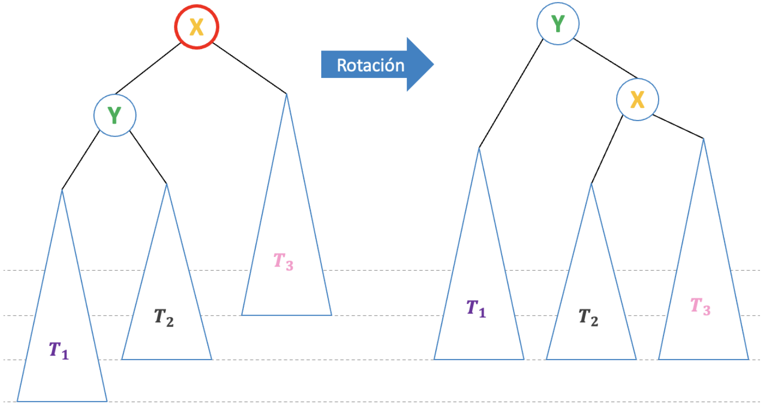
Árboles AVL: caso general post-inserción



- Supongamos que T_1, T_2 y T_3 son AVLs
- X e Y son nodos *en el camino hacia la inserción*
- Al insertar en T_1 se produjo desbalance para X
- ¿Podemos modificar *pocos* nodos para restaurar la propiedad AVL para X?

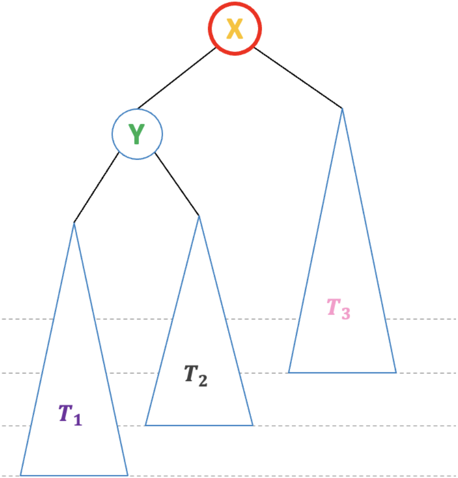
Árboles AVL: rotaciones

Realizamos un intercambio que llamaremos **rotación**



¿Qué subárbol no se mueve?

Árboles AVL: rotaciones



- ¿Cómo se determina qué nodos debemos rotar?
- ¿Cómo sabemos la dirección de la rotación?

Árboles AVL: rotaciones

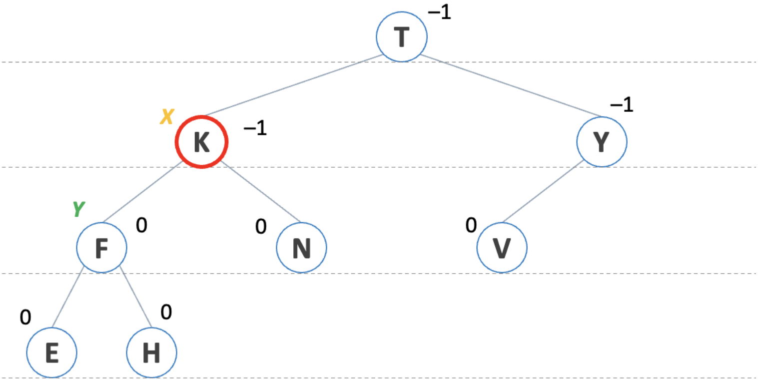
Para un nodo x del árbol

- Agregamos un **atributo de balance** con el cual decidiremos las rotaciones
- Lo denotamos por $x.balance$
- Tomará valor -1, 0 o 1 según sus hijos
  - Si el sub-árbol izquierdo es más alto: $x.balance = -1$
  - Si los hijos tienen la misma altura: $x.balance = 0$
  - Si el sub-árbol derecho es más alto: $x.balance = 1$

¿Cómo se debe actualizar este atributo al insertar un nodo?

Árboles AVL: rotaciones

Antes de agregar B en el árbol AVL que vimos, los atributos de balance son



Si B queda como hijo de E, ¿cómo actualizamos estos valores?

Árboles AVL: rotaciones

Planteamos la siguiente estrategia de actualización

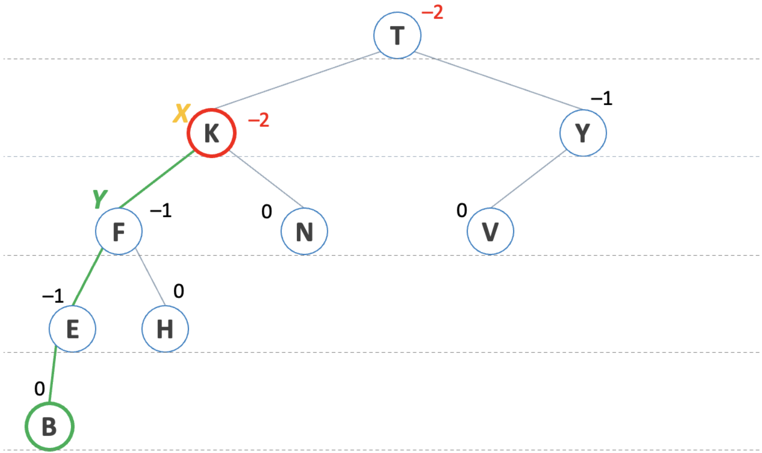
- Recorremos *hacia arriba* partiendo del nodo insertado
- Notemos que esto se puede hacer de manera eficiente si contamos con punteros al padre de cada nodo
- Vamos *revisando* la comparación de alturas de los hijos en el camino

Los nodos clave en la rotación se definen como sigue

- X es el **primer** nodo desbalanceado que encontramos cuando subimos
- Y es el hijo de X en la ruta de la inserción

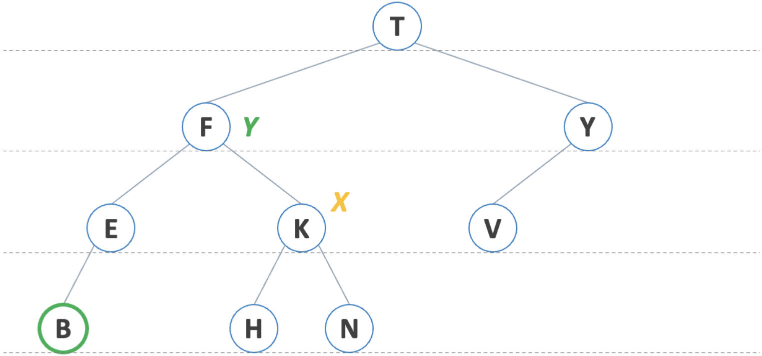
Árboles AVL: rotaciones

Luego de agregar B, los atributos de balance son



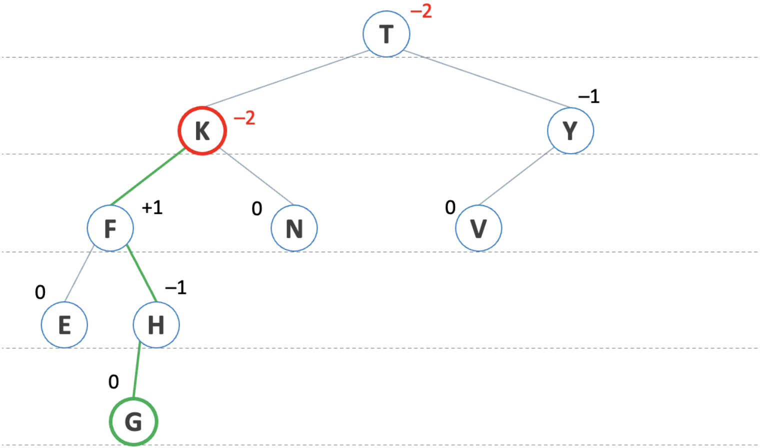
Árboles AVL: rotaciones

Efectuamos la rotación

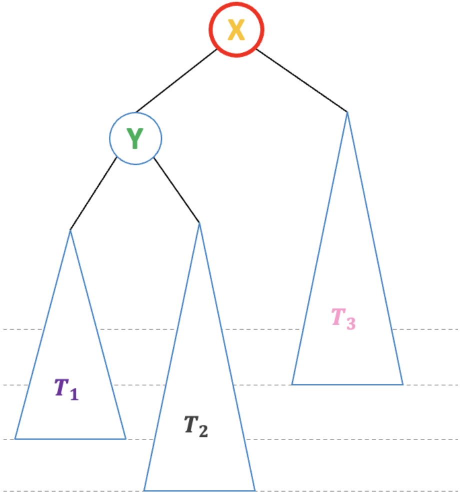


Árboles AVL: rotaciones

Si en lugar de B agregamos G al original, obtenemos



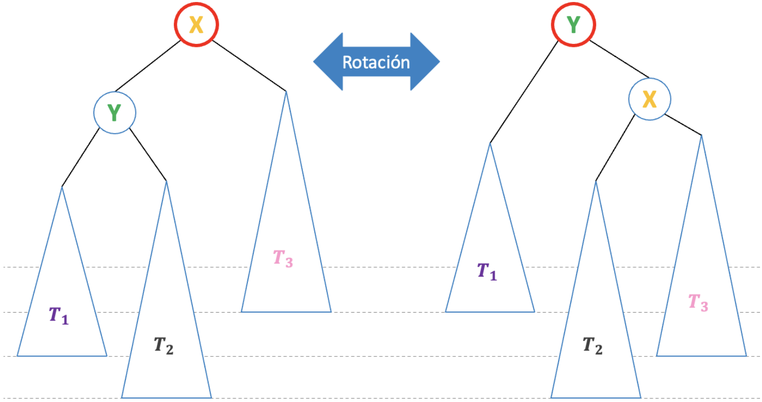
Árboles AVL: rotaciones



- Supongamos que luego de una inserción obtenemos este escenario
- ¿Cómo se rebalancea el nodo X?
- ¿Sirve hacer la misma rotación de antes?

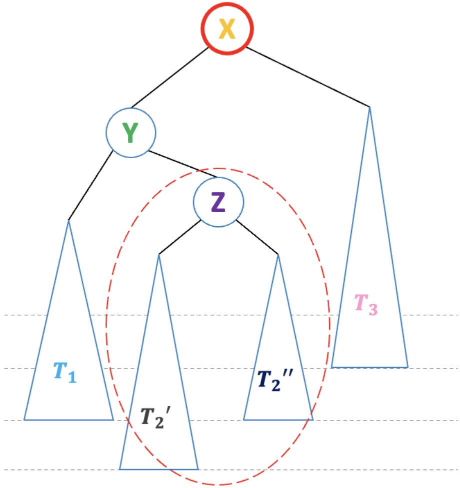
Árboles AVL: rotaciones

Aplicando la misma rotación no logramos resolver el problema



Revisemos el árbol central T_2 más en detalle

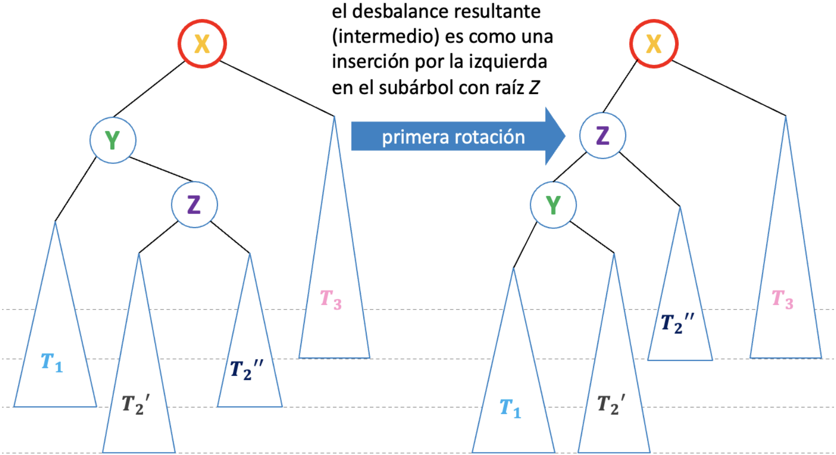
Árboles AVL: rotaciones



- Recordemos que X es el primero desbalanceado
- Por lo tanto, sabemos que los sub-árboles de T_2 son AVL

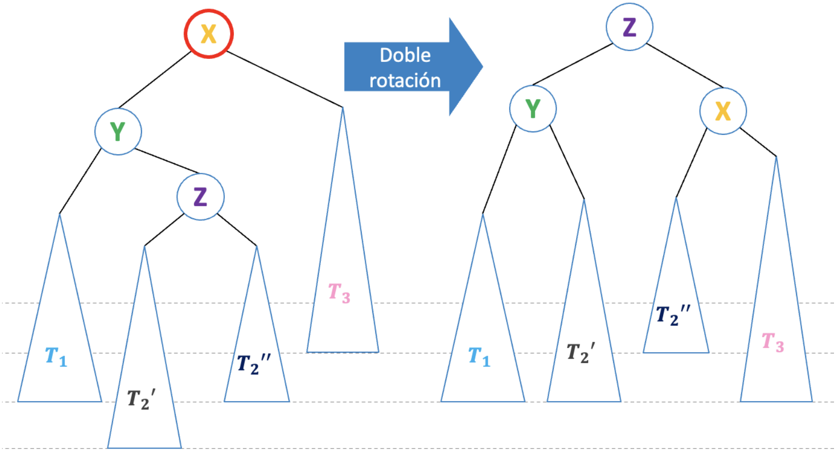
Árboles AVL: rotaciones dobles

Primero rotamos a la izquierda en torno a Y-Z



Árboles AVL: rotaciones dobles

Luego rotamos a la derecha en X-Z (como antes)



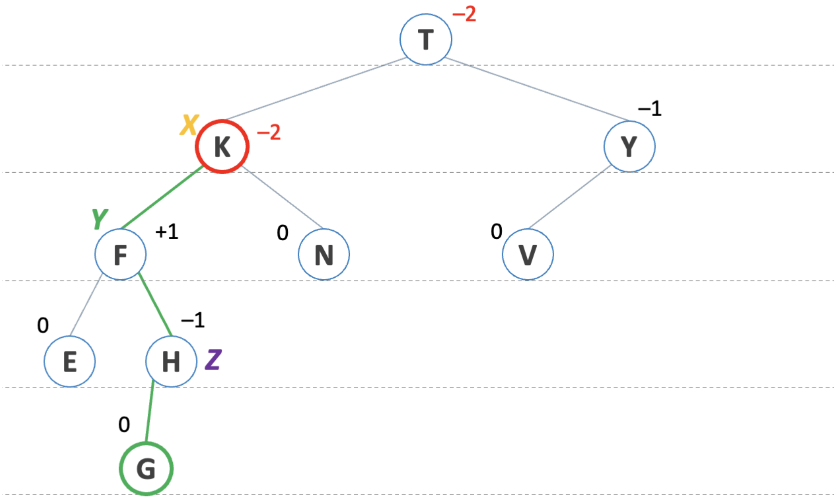
Árboles AVL: rotaciones

La estrategia en estos casos define los nodos X, Y, Z según

- X es el **primer** nodo desbalanceado que encontramos cuando subimos
- Y es el hijo de X en la ruta de la inserción
- Z es el hijo de Y en la ruta de la inserción

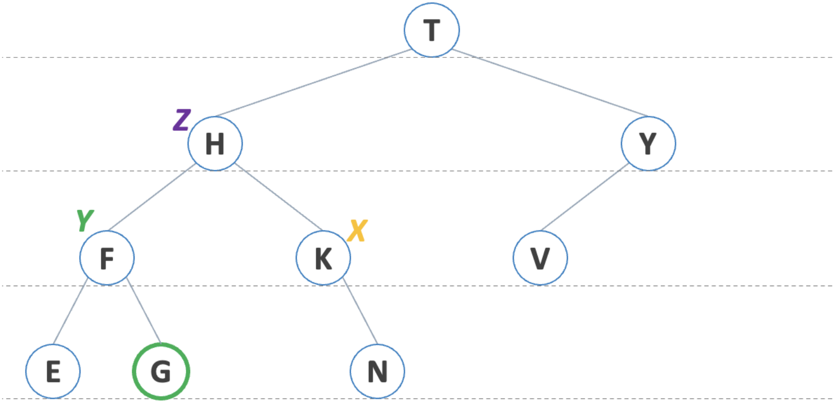
Árboles AVL: rotaciones dobles

Volvamos al ejemplo de agregar G, agregando la definición de Z



Árboles AVL: rotaciones dobles

Aplicamos la rotación dobles y resolvemos el desbalance



Árboles AVL: rotaciones

Hay 4 casos posibles de desbalance, según la ruta de inserción desde X

1. Izquierda + izquierda (LL): rotación simple
2. Izquierda + derecha (LR): rotación doble
3. Derecha + izquierda (RL): rotación doble
4. Derecha + derecha (RR): rotación simple

Los casos 1 y 4 son simétricos. Lo mismo aplica para 2 y 3

Árboles AVL: práctica

Ejercicio

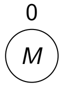
Construya un árbol AVL cuyas llaves se insertan en el siguiente orden:

$$M, D, H, B, A, C, S, K, I$$

Luego de cada inserción, rebalancee los subárboles en caso de ser necesario.

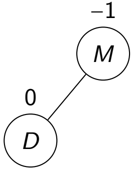
Ejercicio: inserción y rotaciones

Insertamos la llave M



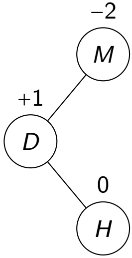
Ejercicio: inserción y rotaciones

Insertamos la llave D



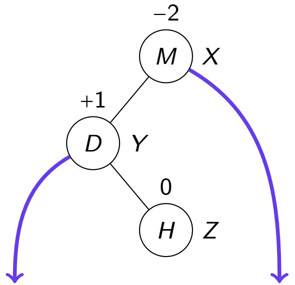
Ejercicio: inserción y rotaciones

Insertamos la llave M y se produce desbalance AVL



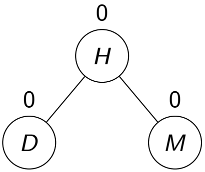
Ejercicio: inserción y rotaciones

Identificamos los nodos de la rotación (doble) y las direcciones para rotar



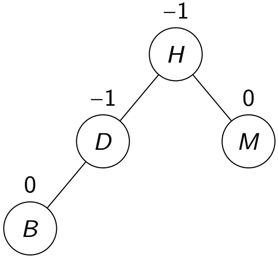
Ejercicio: inserción y rotaciones

Resultado de la rotación



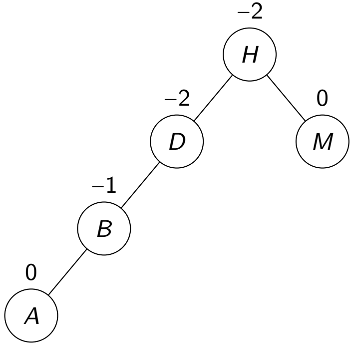
Ejercicio: inserción y rotaciones

Insertamos B y no se produce desbalance



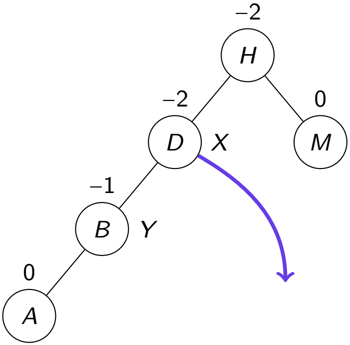
Ejercicio: inserción y rotaciones

Insertamos A y se produce desbalance



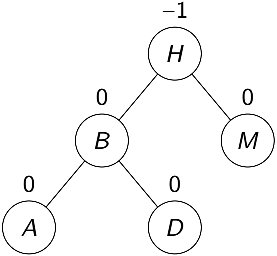
Ejercicio: inserción y rotaciones

Identificamos nodos para la rotación (simple) y la dirección



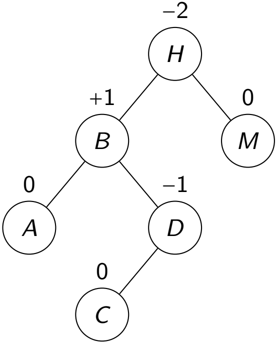
Ejercicio: inserción y rotaciones

Resultado de la rotación



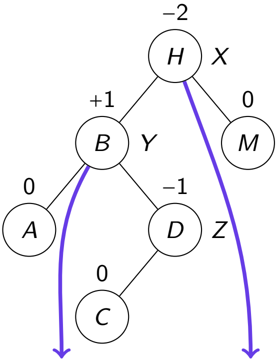
Ejercicio: inserción y rotaciones

Insertamos C y producimos desbalance



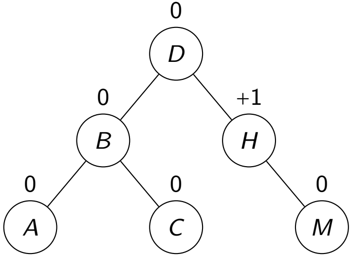
Ejercicio: inserción y rotaciones

Identificamos nodos de la rotación (doble) y las direcciones



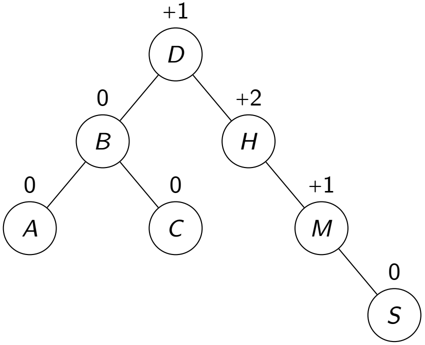
Ejercicio: inserción y rotaciones

Resultado de la rotación



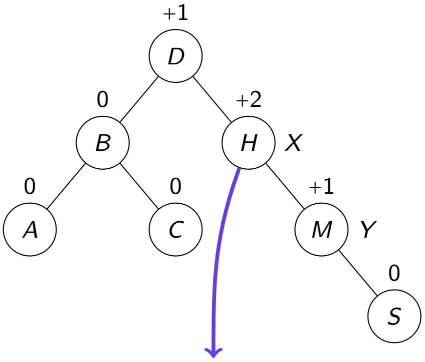
Ejercicio: inserción y rotaciones

Insertamos S y se produce desbalance



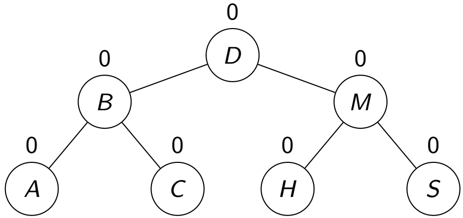
Ejercicio: inserción y rotaciones

Identificamos nodos para rotación (simple) y la dirección



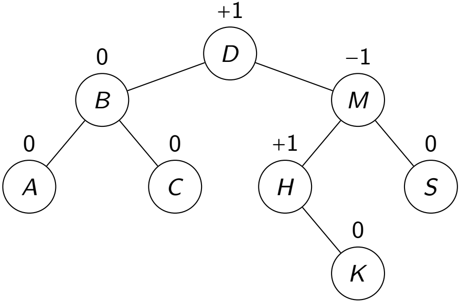
Ejercicio: inserción y rotaciones

Resultado de la rotación



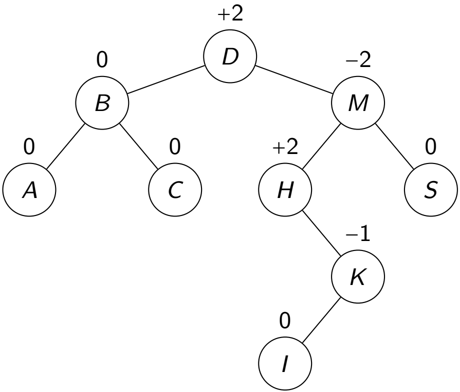
Ejercicio: inserción y rotaciones

Insertamos K y no hay desbalance



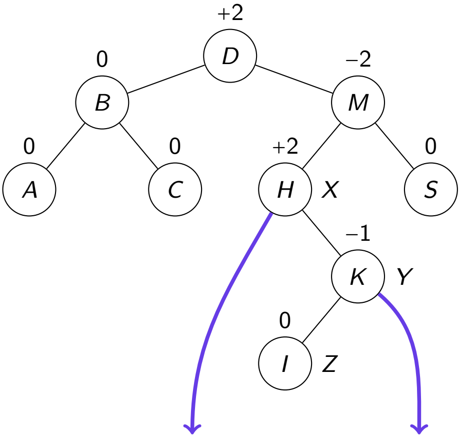
Ejercicio: inserción y rotaciones

Insertamos I y se produce desbalance



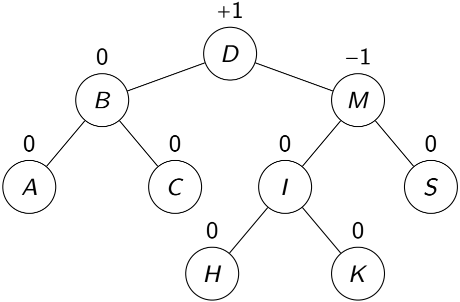
Ejercicio: inserción y rotaciones

Identificamos nodos de la rotación (doble) y las direcciones



Ejercicio: inserción y rotaciones

Resultado de la rotación



Sumario

Complejidad de las rotaciones

Una rotación tiene **costo constante**

- En una rotación simple se cambian 3 punteros: $\mathcal{O}(1)$
- En una rotación doble se cambian 5 punteros: $\mathcal{O}(1)$

Además, al rotar para restaurar balance de X, este se resuelve sin crear un nuevo balance

- No se aumentan las diferencias de altura respecto al resto del árbol
- En el peor caso, se realiza a lo más **una rotación** (simple o doble) **por inserción**

Complejidad de las rotaciones

Gracias a esto, para un árbol de altura h, una inserción contempla

- Inserción propiamente tal $\mathcal{O}(h)$
- Detectamos los nodos que participan de la rotación (si aplica) $\mathcal{O}(h)$
- En el peor caso, aplica rotar y se requiere una rotación $\mathcal{O}(1)$
- Estos pasos se realizan consecutivos Total $\mathcal{O}(h)$

¿Cuál es la altura de un árbol AVL en función de n?

Altura de un árbol AVL

Antes de atacar el problema, pensemos en la relación inversa: cómo depende n de h.

Para un árbol AVL T

- ¿Cuál es el máximo número de nodos de T si tiene altura h?
- ¿Y el mínimo?

Si probamos un crecimiento exponencial para n en función de h, probaremos que la altura está acotada por $\log(n)$

Altura de un árbol AVL: caso máximo

En primer lugar, consideremos el número máximo en un árbol AVL

Esto ocurre cuando el árbol está **lleno**, es decir, para cada nivel del árbol existe la máxima cantidad de nodos posible. Este caso claramente es un árbol AVL

Si k es el nivel de los nodos (recordando que la raíz está en el nivel 0 y que la altura es # niveles), la cantidad de nodos total es

$$n = \sum_{k=0}^{h-1} 2^k = 2^h - 1$$

con lo cual

$$n \in \Theta(2^h) \quad \Leftrightarrow \quad 2^h \in \Theta(n)$$

Como se trata de funciones crecientes deducimos que $h \in \Theta(\log(n))$

Altura de un árbol AVL: caso mínimo

En segundo lugar, consideramos el número mínimo $m(h)$ de nodos que puede tener un árbol AVL de altura h

Plantearemos una recurrencia que defina m

- Observamos que $m(1) = 1$ y $m(2) = 2$
- Para el caso general, nos interesa minimizar el número de nodos
- Nos concentramos en el caso en que los hijos difieren su altura en 1
- Además, dado que el árbol principal tiene altura h, uno de los hijos debe tener $h - 1$ y el otro $h - 2$
- Finalmente, exigimos que cada uno de estos hijos también tenga el mínimo posible. Con esto, planteamos una ecuación de recurrencia

$$m(h) = m(h-1) + m(h-2) + 1$$

Altura de un árbol AVL

Esta recurrencia es similar a la recurrencia de Fibonacci

$$F(0) = 0, F(1) = 1, \quad F(h) = F(h-1) + F(h-2), h \geq 2$$

cuyo término general satisface la aproximación $F(h) \approx \frac{\varphi^h}{\sqrt{5}}$. Con esto, se puede probar por inducción que

$$m(h) = F(h+3) - 1$$

y utilizando la aproximación se obtiene

$$m(h) = F(h+3) - 1 \approx \frac{\varphi^{h+3}}{\sqrt{5}} - 1$$

y deducimos que

$$h + 3 \approx \log_{\varphi}(\sqrt{5}(m(h) + 1))$$

Concluímos que $h \in \mathcal{O}(\log(n))$

Altura de un árbol AVL

Teorema

Todo árbol AVL con n nodos tiene altura h tal que

$$h \in \mathcal{O}(\log(n))$$

Sumario

Objetivos de la clase

- ☐ Comprender la importancia de acotar la altura de un ABB
- ☐ Conocer una posible definición de balance
- ☐ Aplicar las rotaciones para rebalancear ABBs luego de inserciones
- ☐ Demostrar que la altura de un árbol AVL es logarítmica en el número de nodos