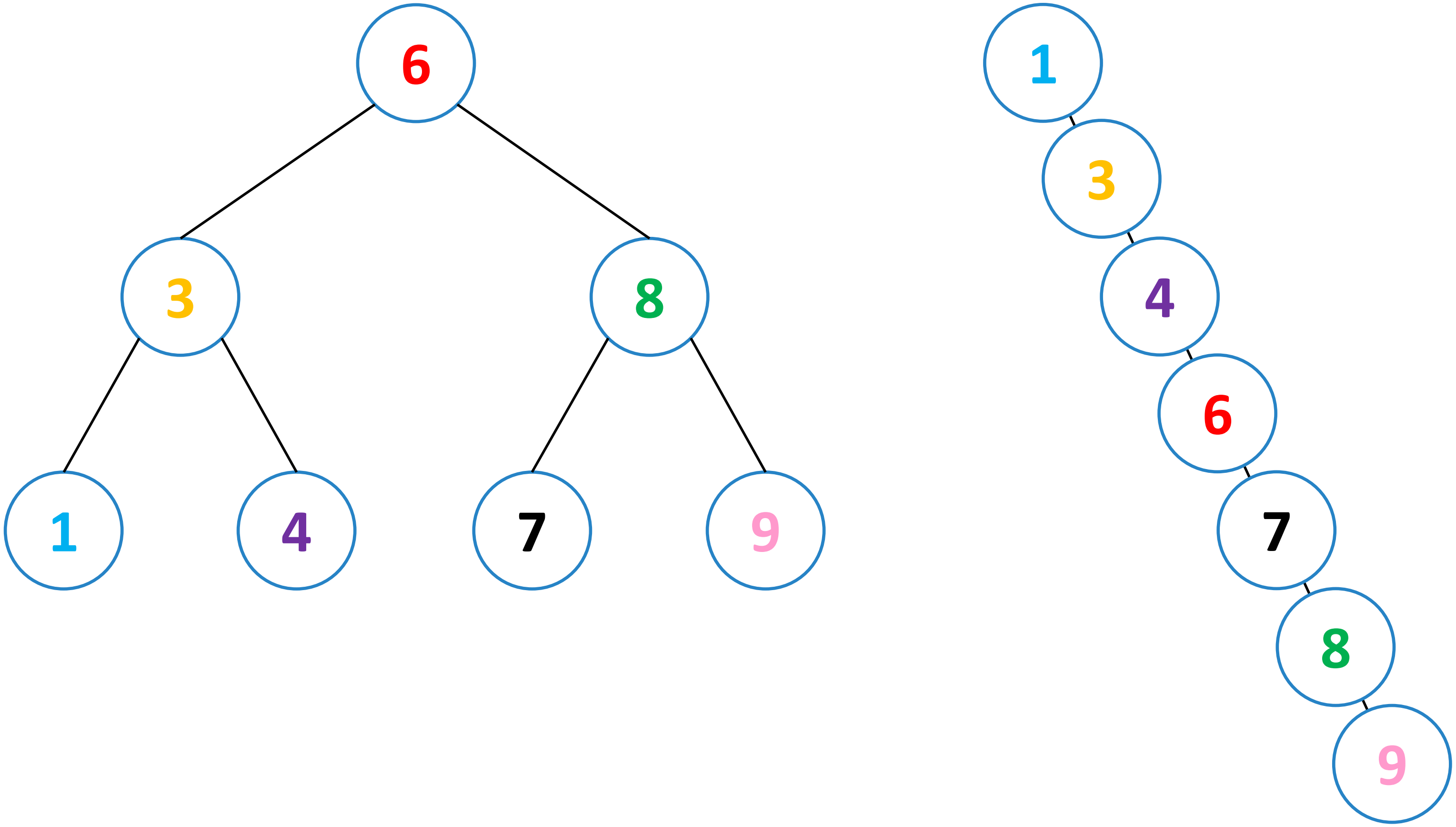
Raíces y raíces

¿Hay algunas raíces más convenientes que otras?

¿Qué pasa con el árbol si no queda un dato conveniente como raíz?

¿Cómo varía la complejidad de las operaciones?

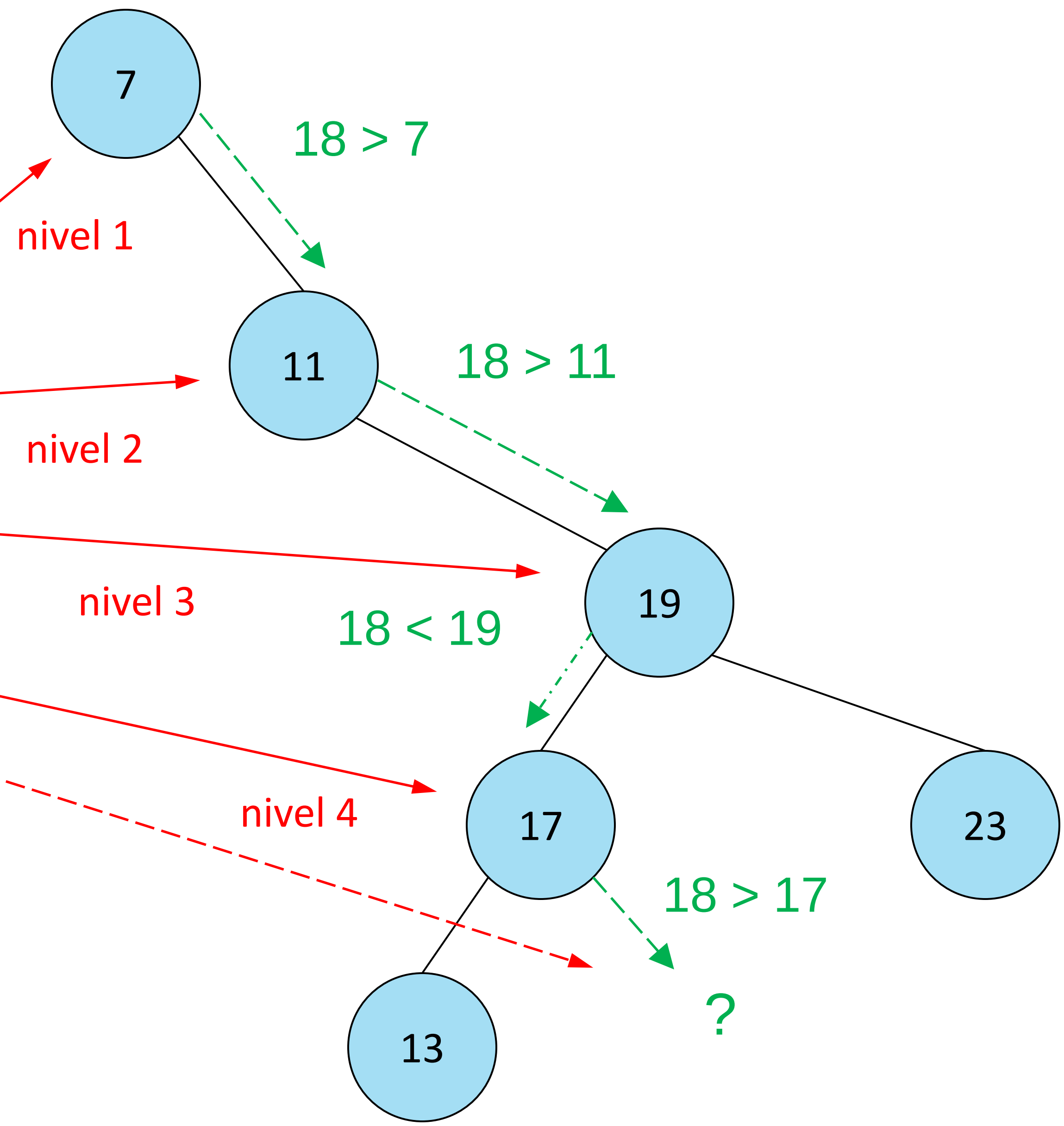
Mismos datos, distintos árboles



¿Cuáles son las consecuencias de estas diferencias?

Si en el árbol de la derecha buscamos la clave 18 a partir de la raíz:

- comparamos 18 con 7
- … luego con 11
- … de ahí con 19
- … con 17
- … y tratamos de ir al hijo derecho del nodo con clave 17
- **(** este último nodo no existe → la clave 18 **no está** almacenada en el árbol, y por lo tanto devolvemos Ø **)**



Complejidad de las operaciones

Todas las operaciones —*buscar*, *insertar* y *eliminar**— toman tiempo (o número de pasos) proporcional a la *altura del árbol* —el número máximo de niveles desde la raíz hasta la hoja "de más abajo"

… o la longitud de la rama más larga del árbol:

- la altura mínima de un ABB con n objetos es O(log$n$)
- … aunque en general podría ser O(n)

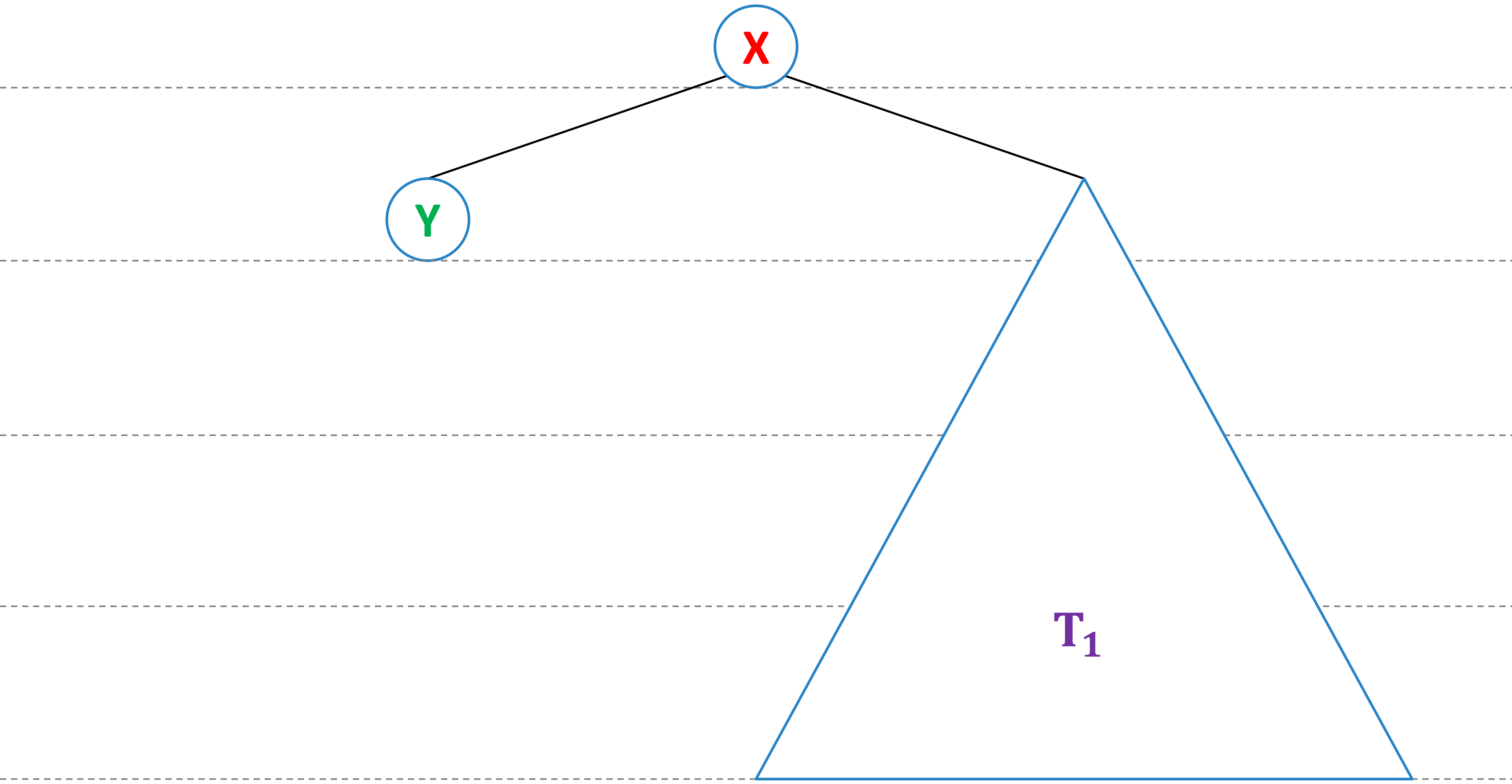
**insertar* y *eliminar* incluyen primero una búsqueda

Queremos asegurarnos de que el árbol esté **balanceado**

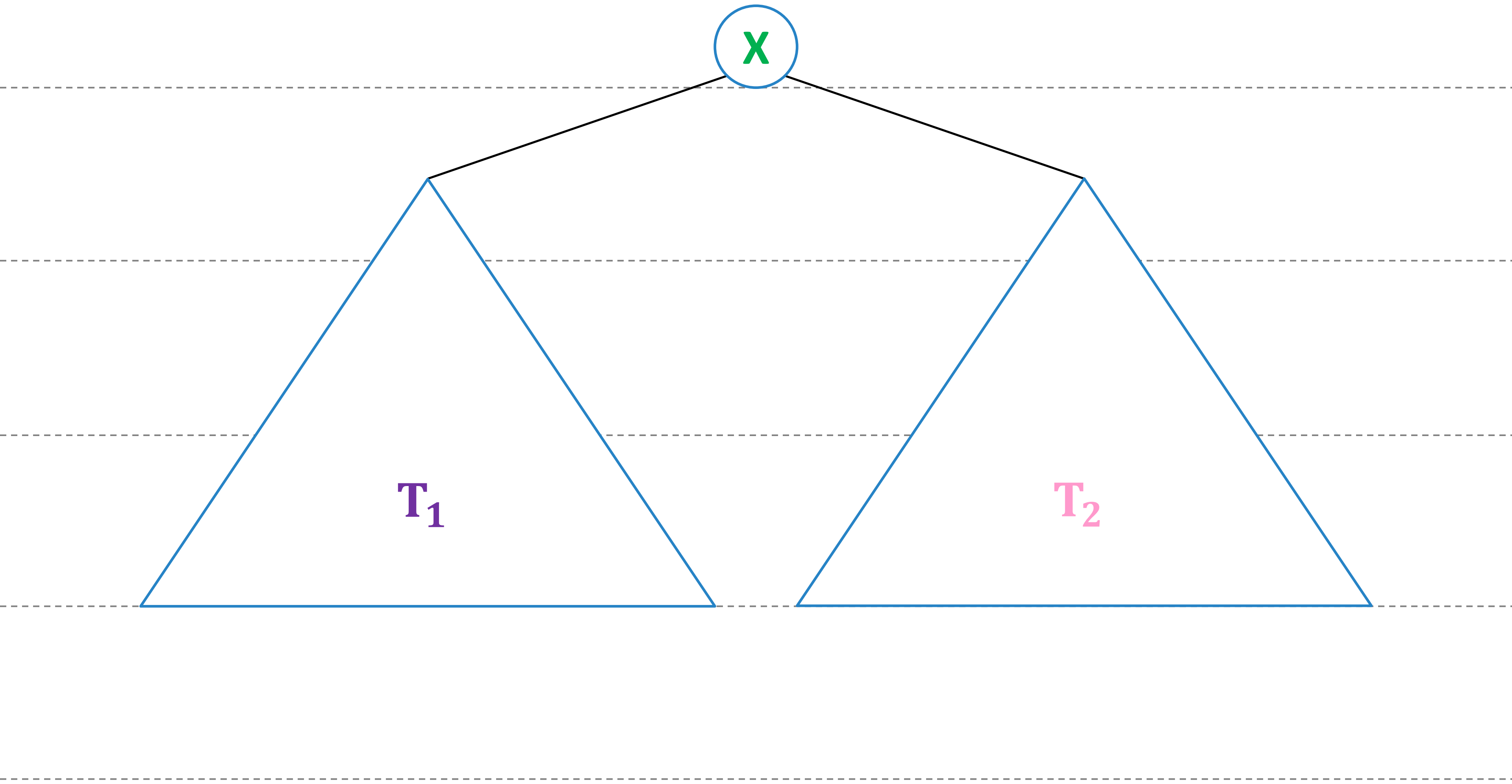
¿Cómo podríamos definir esta noción?

Nos interesa que se pueda cumplir recursivamente

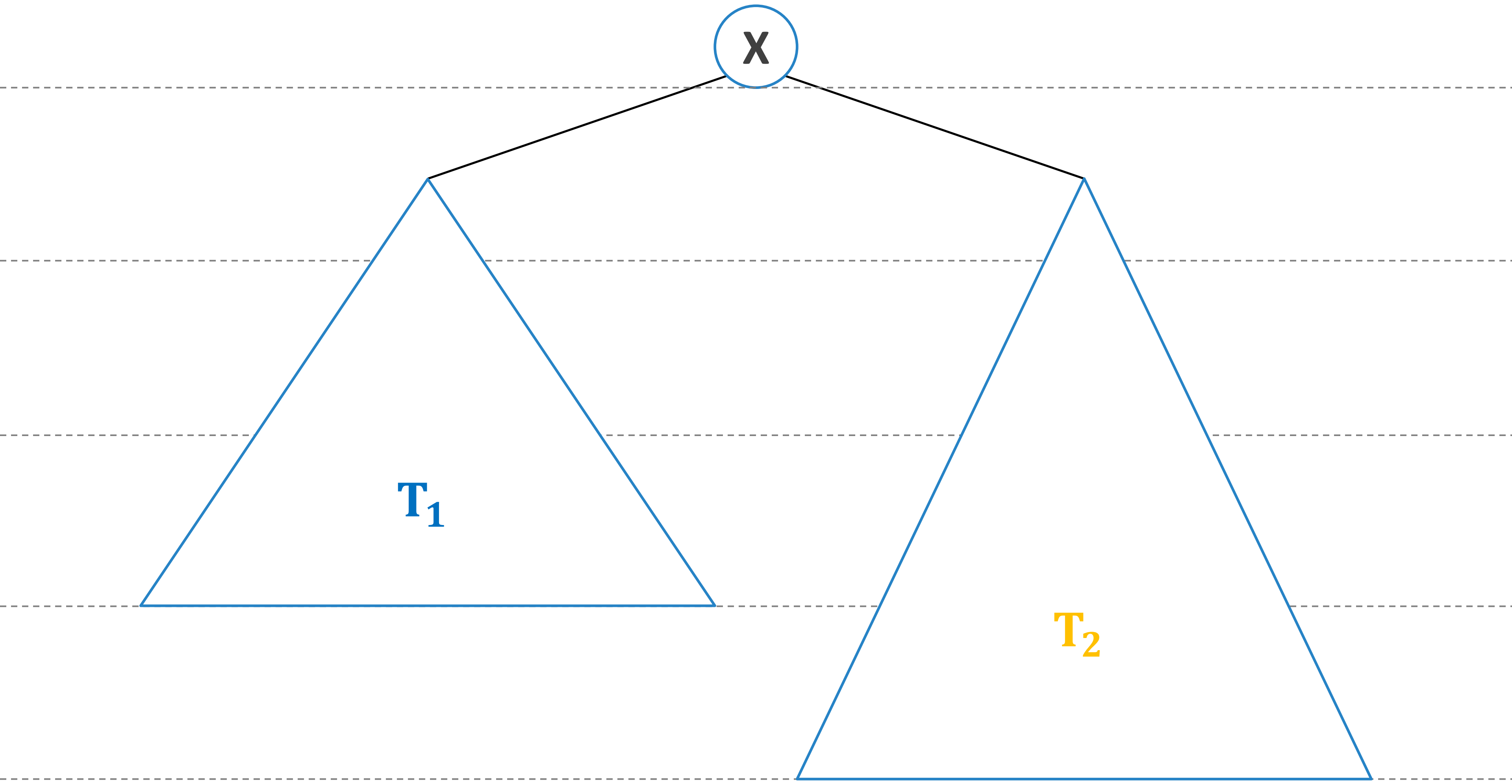
¿Está balanceado?



¿Está balanceado?



¿Está balanceado?



ABBs balanceados

Para ABBs, podemos garantizar que las operaciones de diccionario —*buscar*, *insertar* y *eliminar*— tomen tiempo O(log$n$) en el peor caso:

es necesario mantenerlos balanceados

La **propiedad de balance** debe cumplir dos condiciones:

- debe asegurar que la altura de un árbol con n nodos sea O(log$n$)
- debe ser fácil de mantener —la complejidad de la operación de (re)balancear el árbol no puede ser mayor que O(log$n$)

Árboles AVL

Diremos que un ABB está **AVL-balanceado** si:

- las alturas de los hijos de la raíz difieren a lo más en 1 entre ellas
- cada hijo a su vez está **AVL-balanceado**

Un ABB que cumple esta propiedad se llama **árbol AVL**

Operaciones en árboles AVL

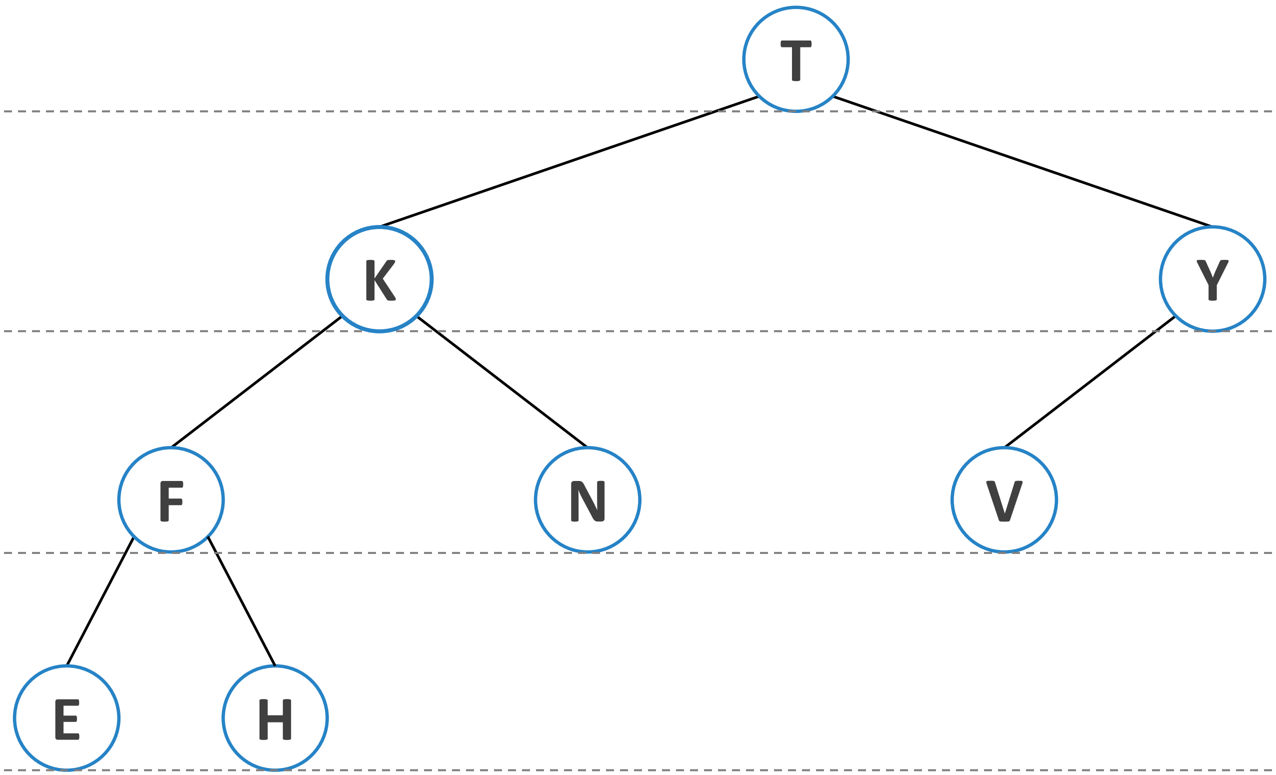
Al insertar o eliminar un nodo, es posible desbalancear el árbol

¿Cómo garantizamos el **balance** del árbol luego de cada operación?

Nos interesa conservar todas las propiedades de los ABB:

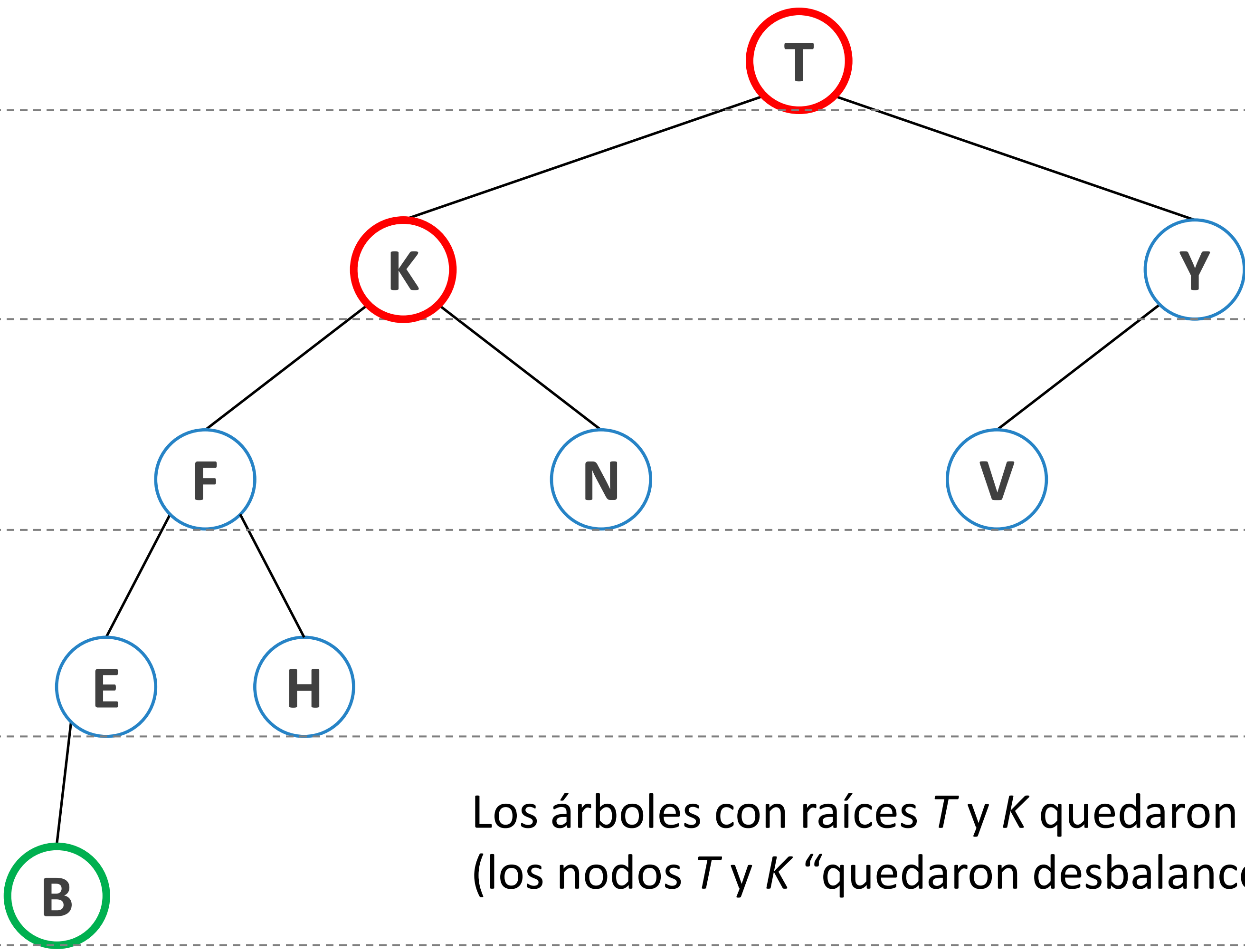
- en particular, el balance **debe ser restaurado** antes de que la operación —de inserción o elimionación— pueda considerarse completa

P.ej., árbol AVL inicial



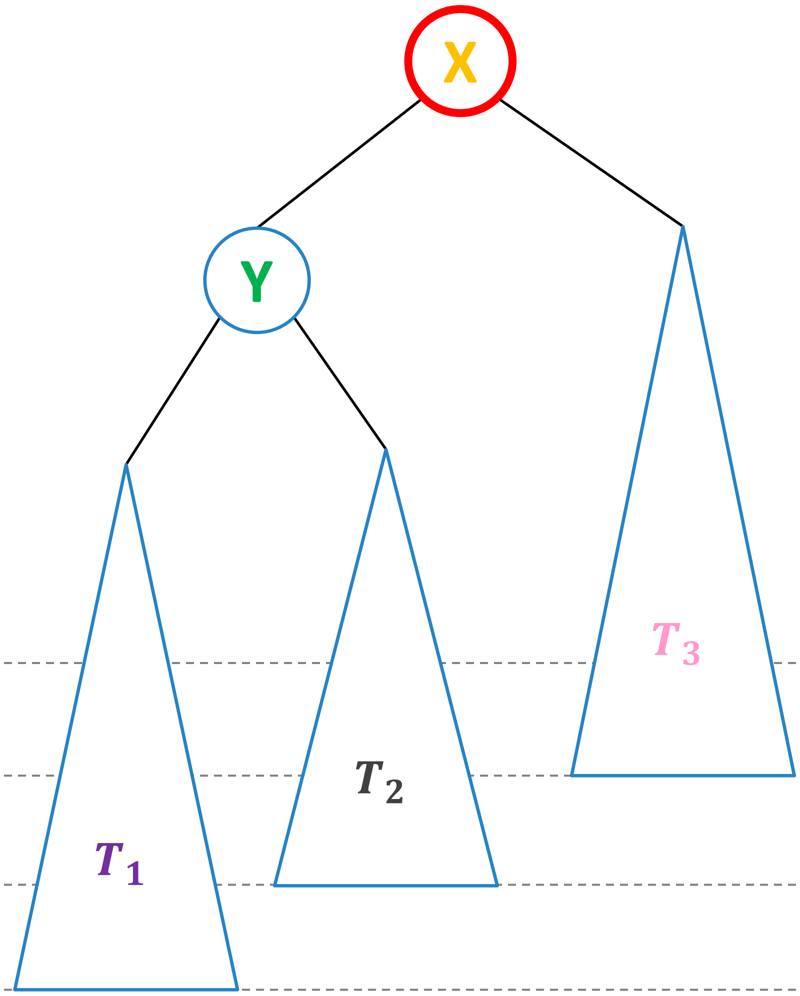
Para cualquier nodo, las alturas de sus hijos difieren a lo más en 1 entre ellas

… árbol luego de insertar *B*



Los árboles con raíces *T* y *K* quedaron desbalanceados
(los nodos *T* y *K* "quedaron desbalanceados")

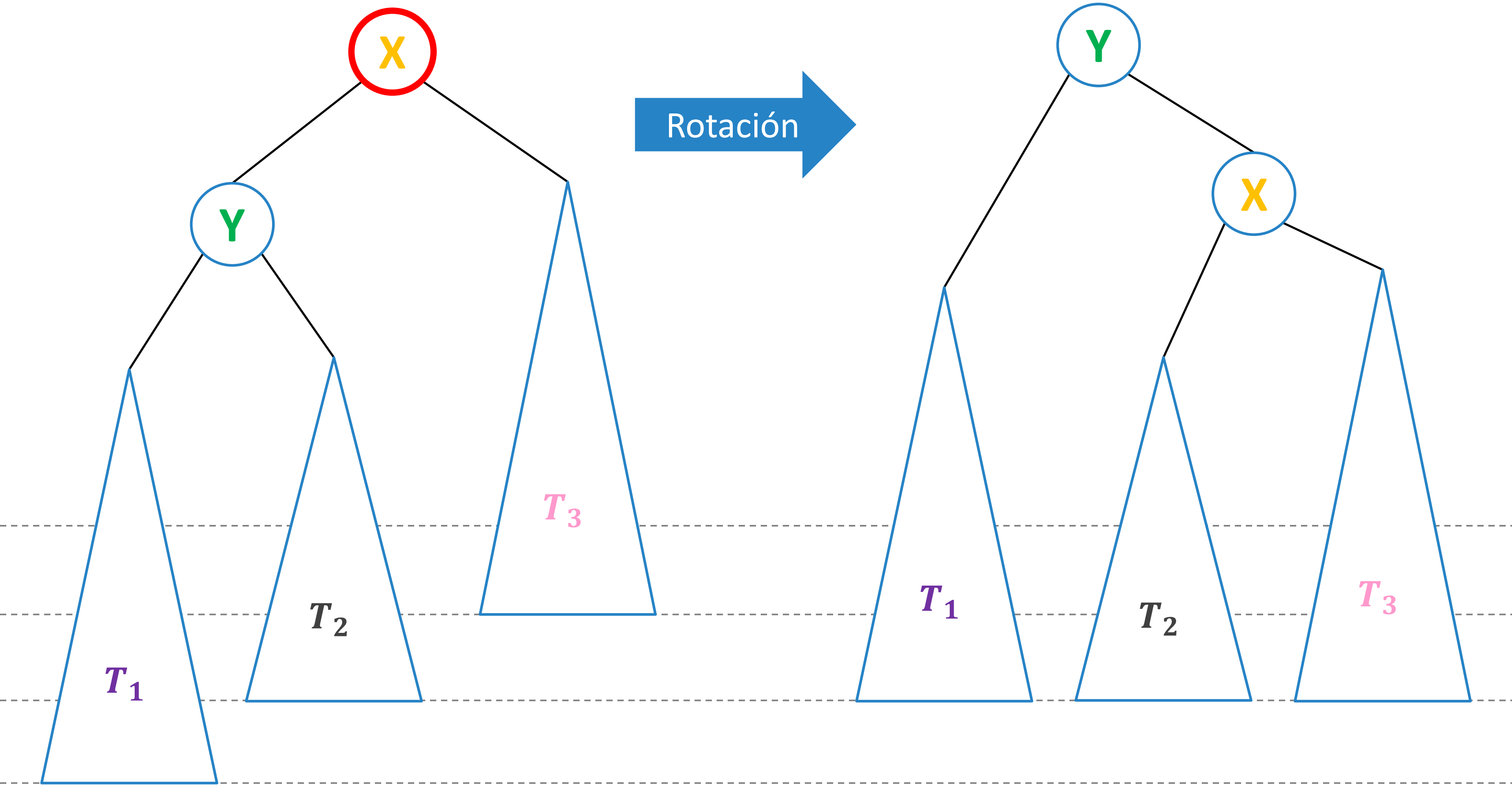
Más en general, luego de insertar en T_1



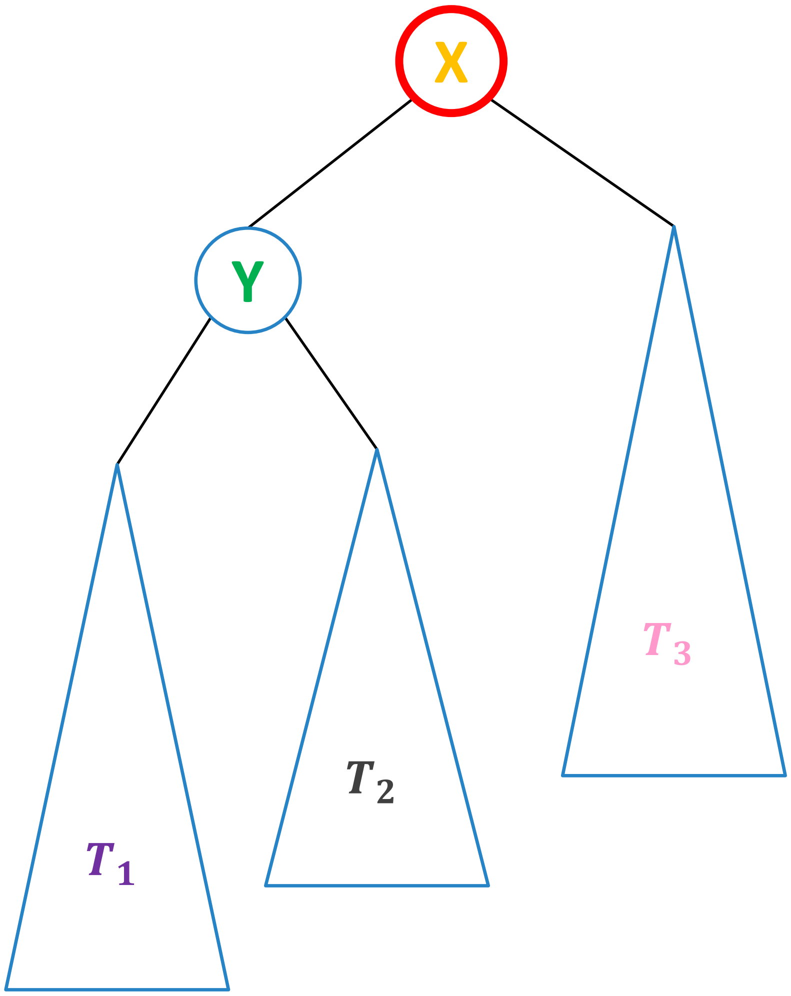
¿Cómo (re)balancear el árbol con raíz X?

(suponemos que T_1, T_2 y T_3 son AVLs)

Rotación a la derecha en torno a *X*-*Y*



Rotación *X*-*Y*



¿Cómo encontramos los nodos *X* y *Y* a rotar?

Agregamos a cada nodo *x* un **atributo de balance** adicional:

x.balance = –1 / 0 / +1 ,

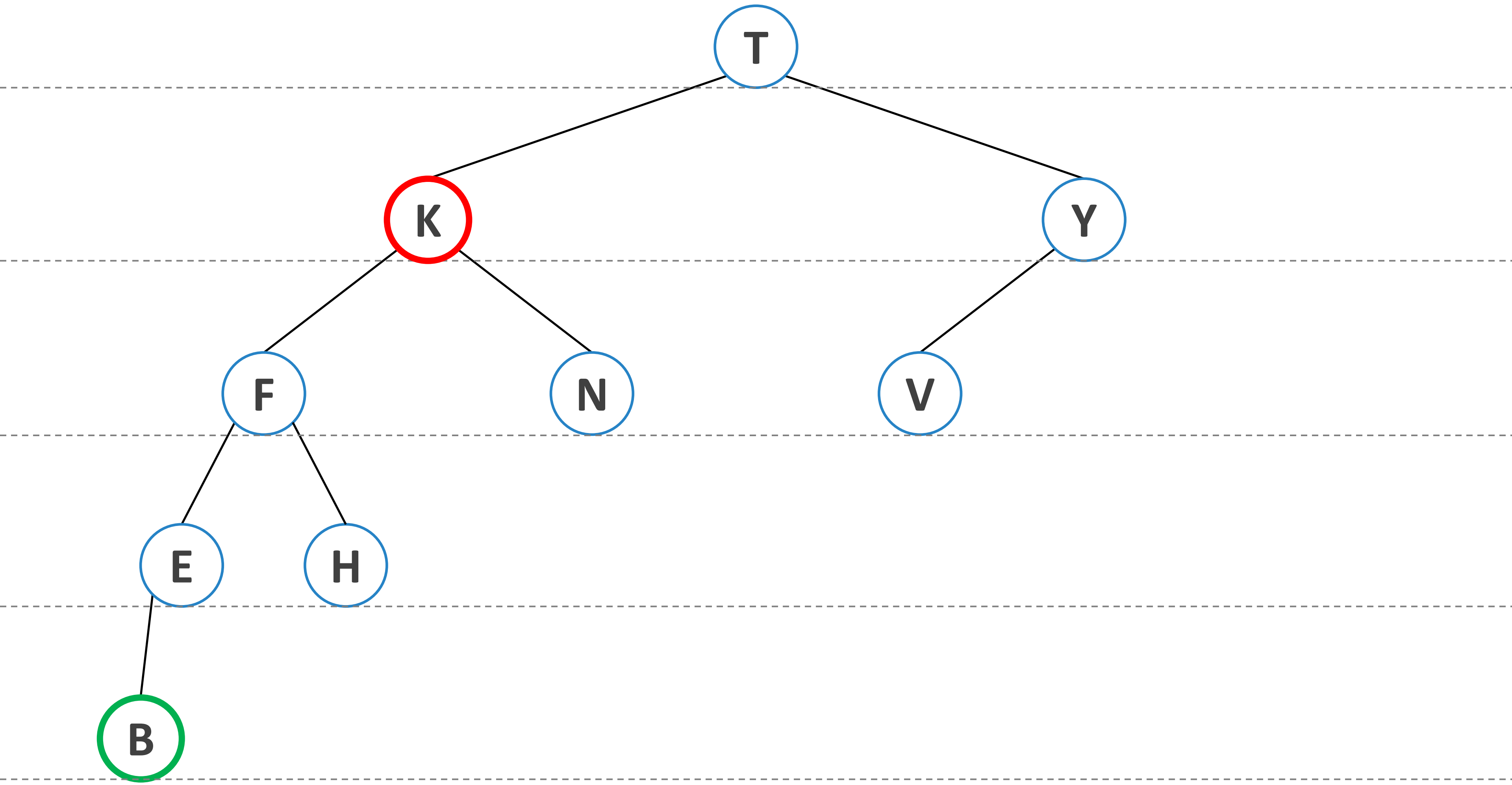
… dependiendo de si el subárbol izquierdo es más alto, ambos subárboles tienen la misma altura, o si el subárbol derecho es más alto, respectivamente

Luego de insertar, recorremos el árbol hacia arriba a lo largo de la **ruta de inserción**:

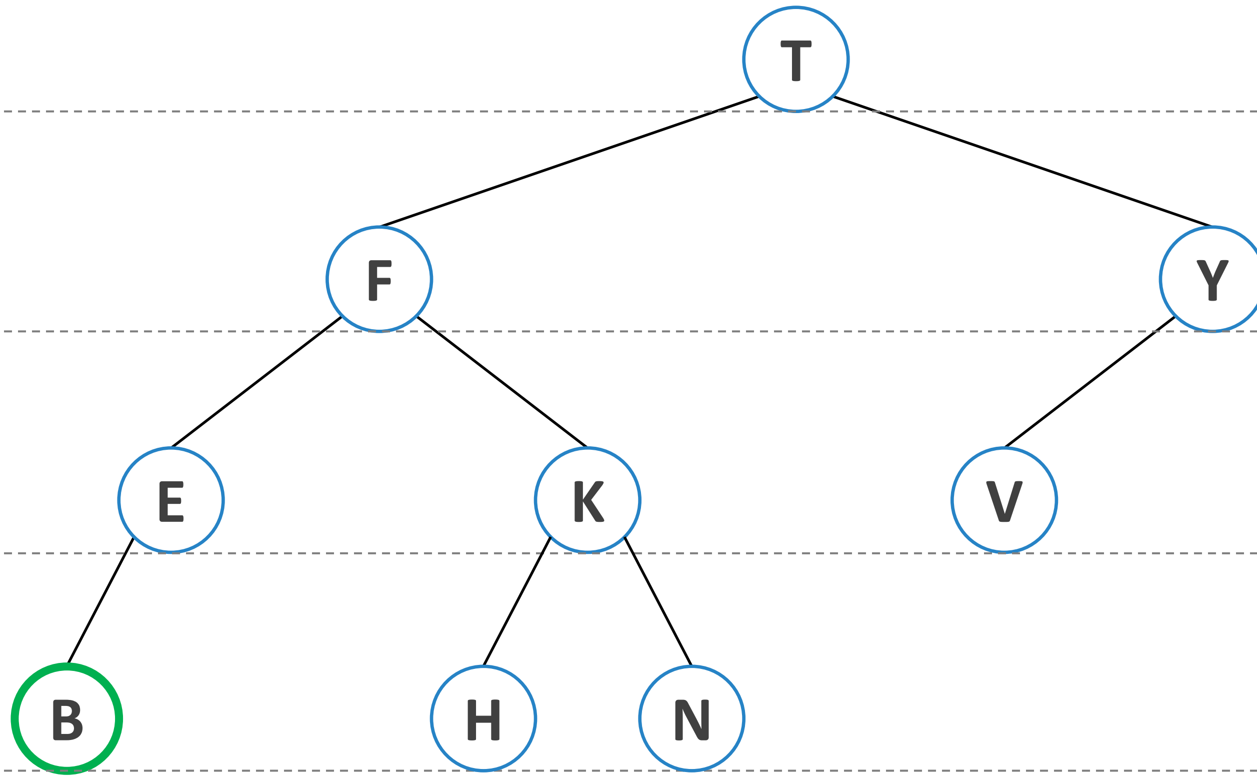
- definimos *X* como la raíz del **primer** árbol desbalanceado que encontremos (o como el primer nodo desbalanceado),

  … y *Y* como el siguiente nodo (hacia abajo) en la ruta de inserción

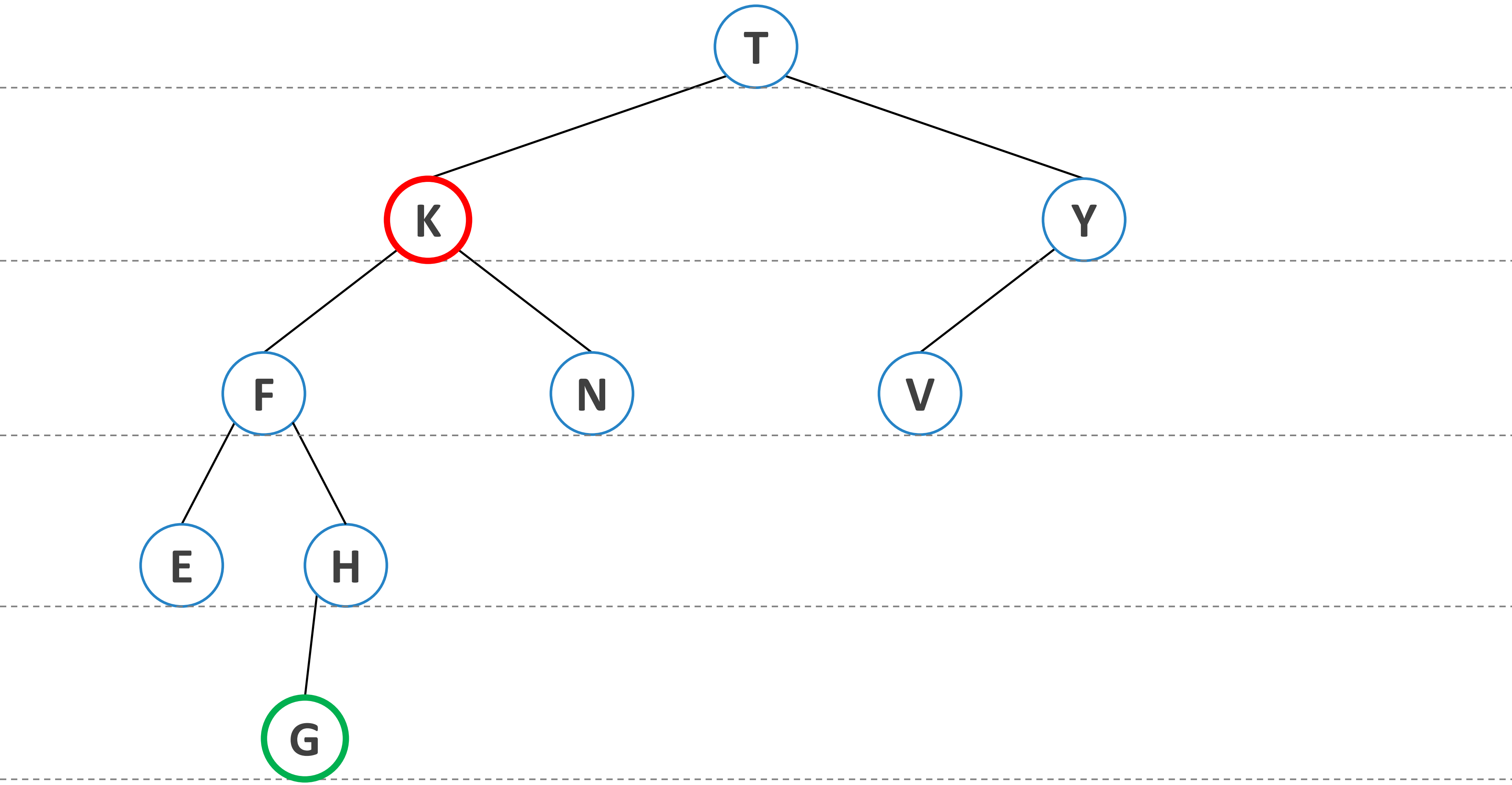
Luego de insertar *B*



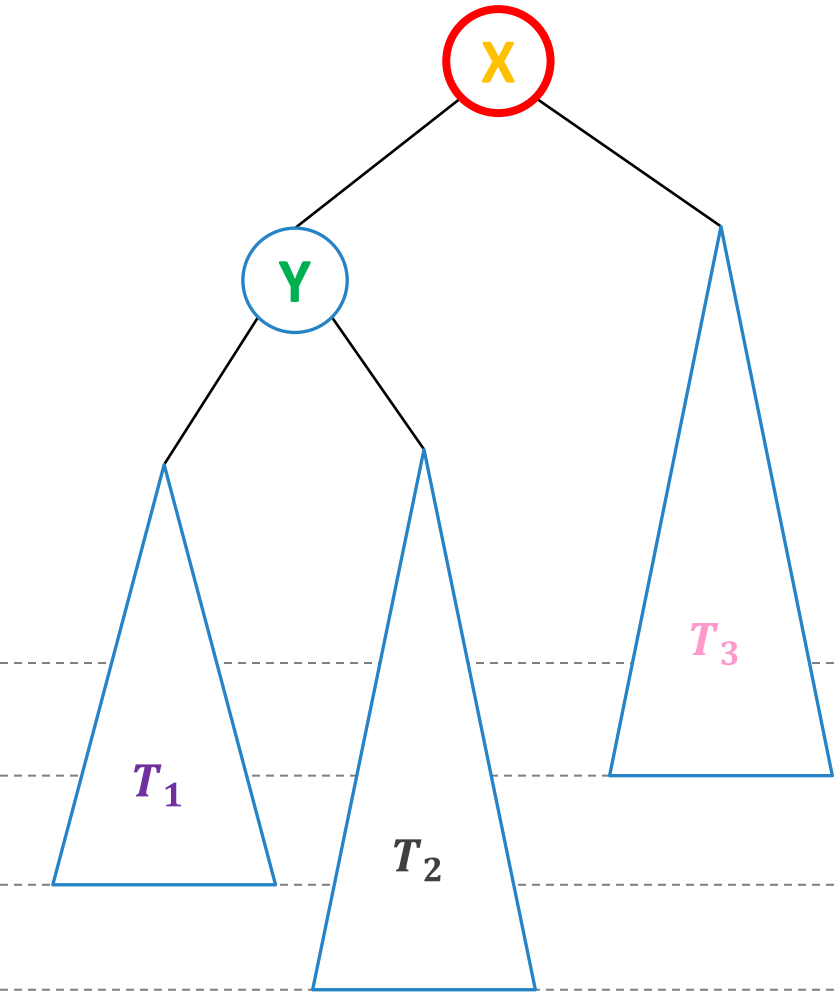
Rotación a la derecha en torno a *K-F*



Otro ej.: el árbol (inicial) luego de insertar *G*

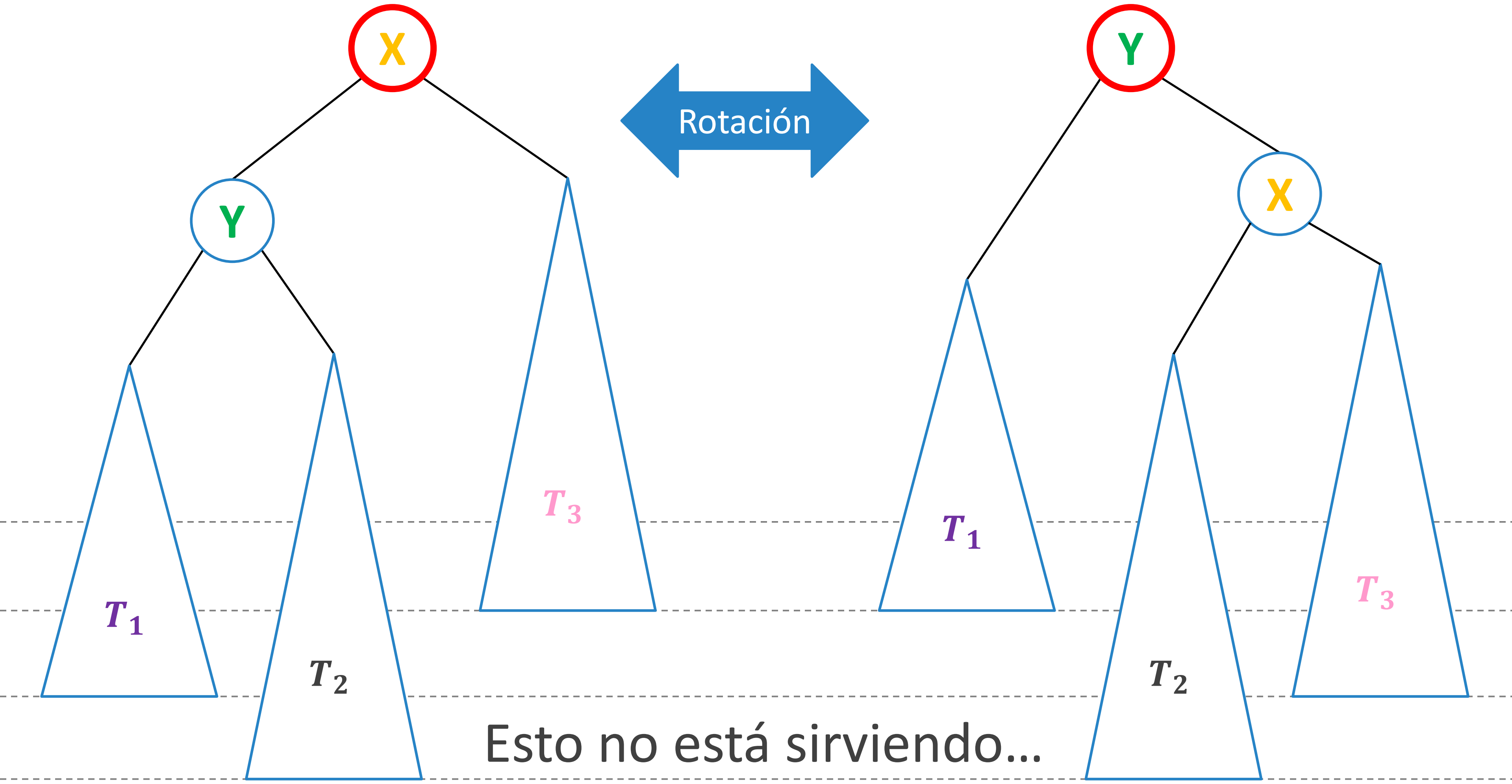


Más en general, el árbol luego de una inserción en T_2



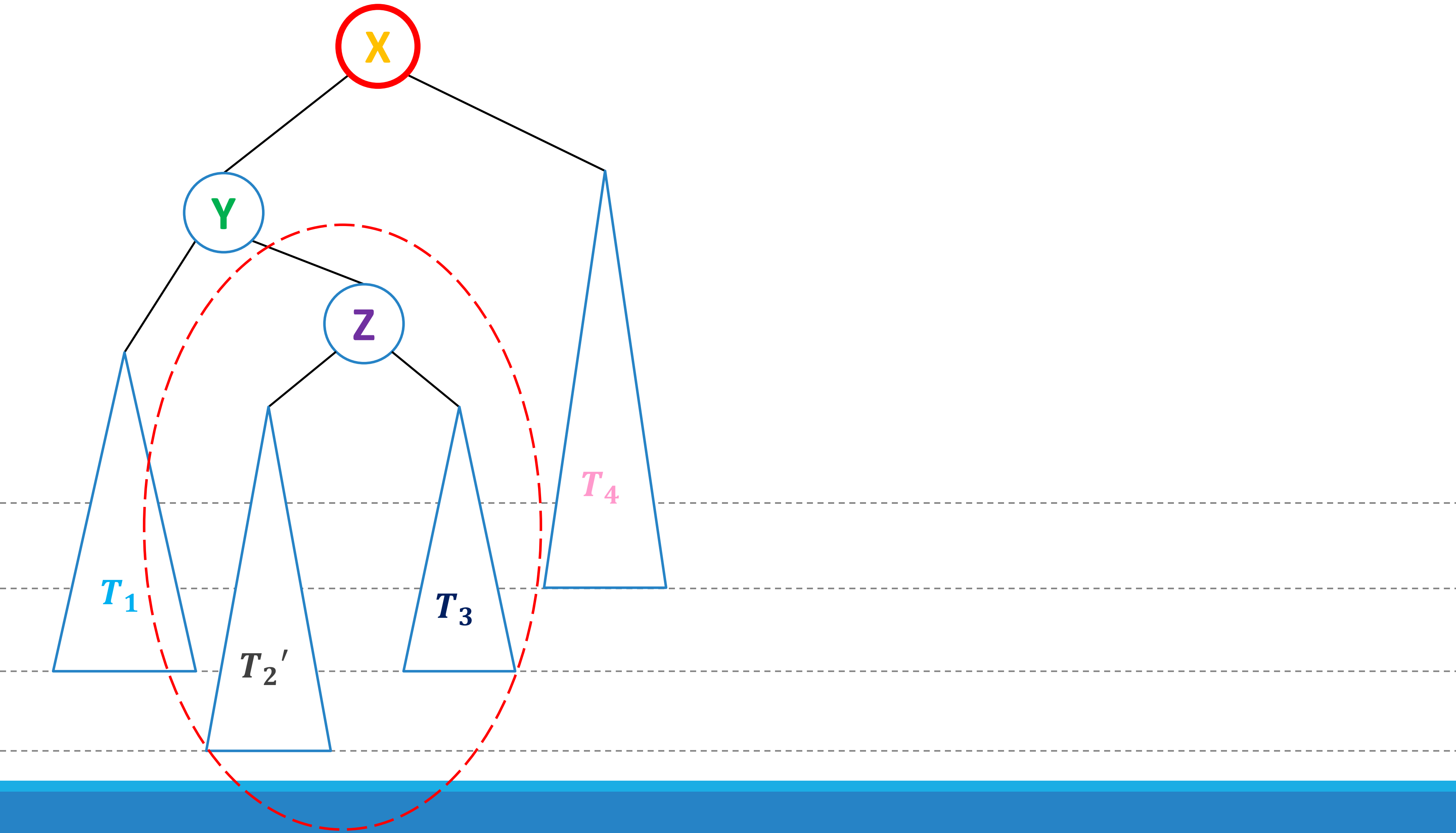
¿Cómo (re)balancear el árbol con raíz X?

¿Rotación a la derecha en torno a *X-Y* ?

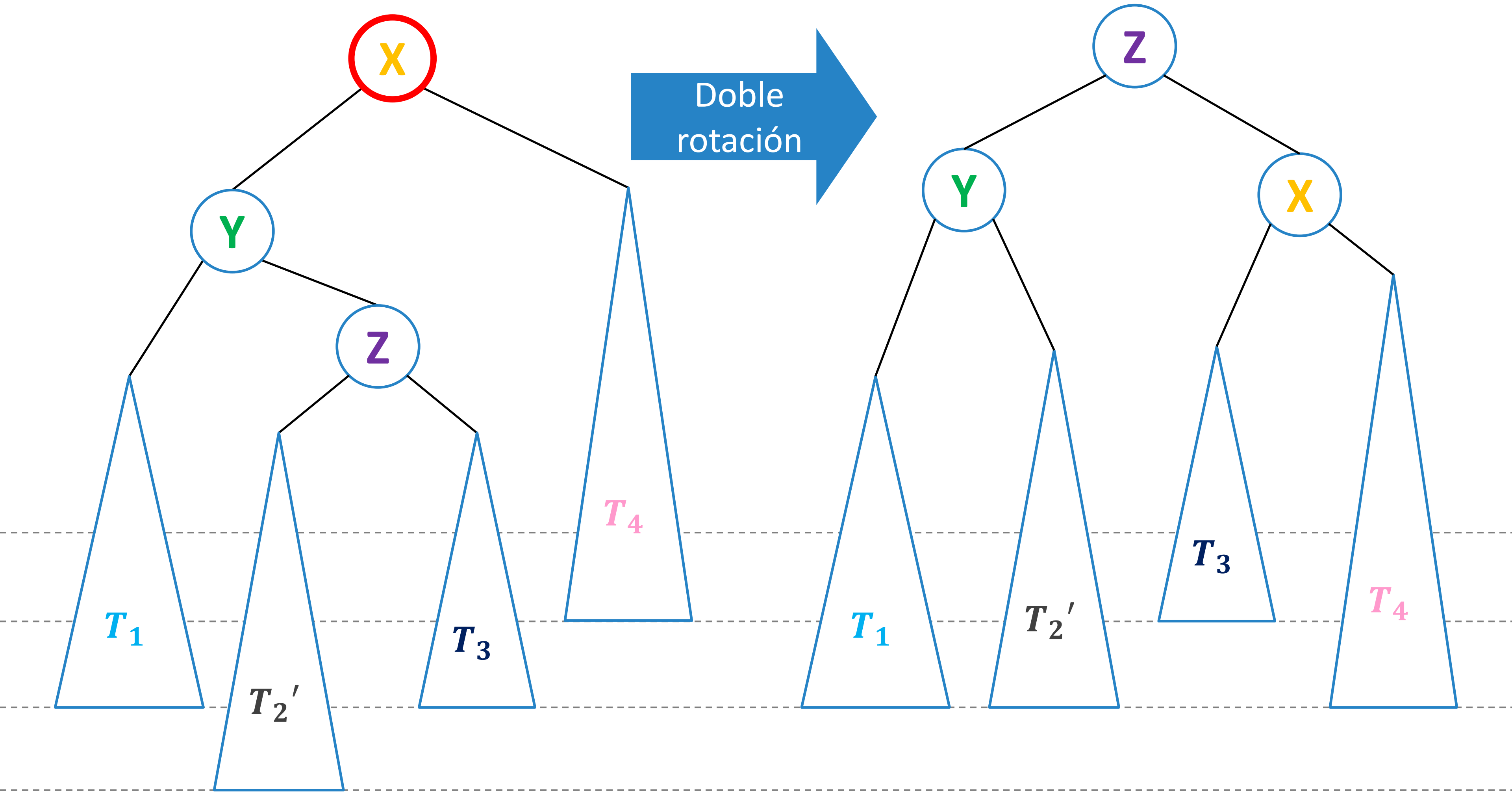


Esto no está sirviendo…

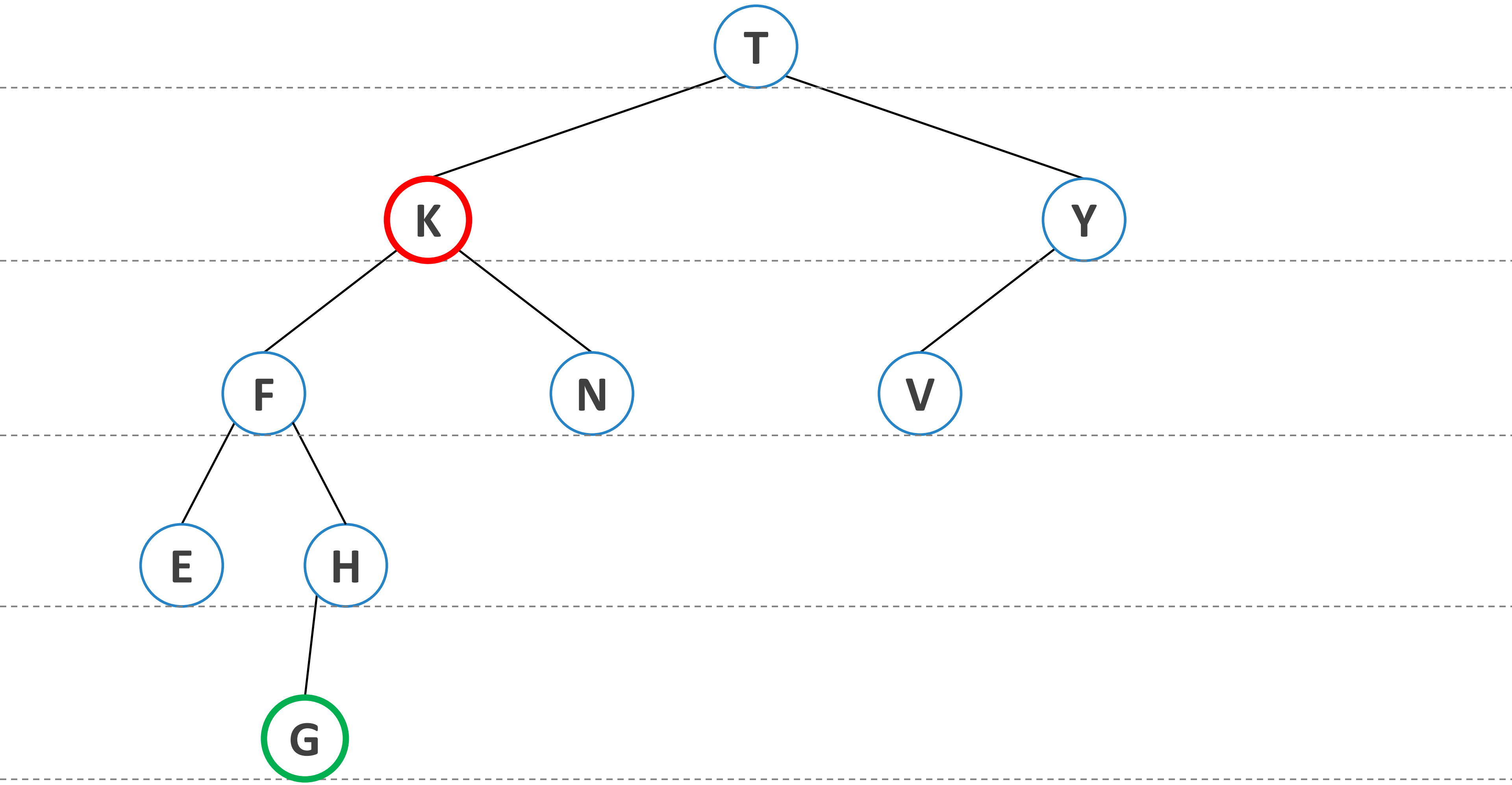
Hagamos doble click en T_2



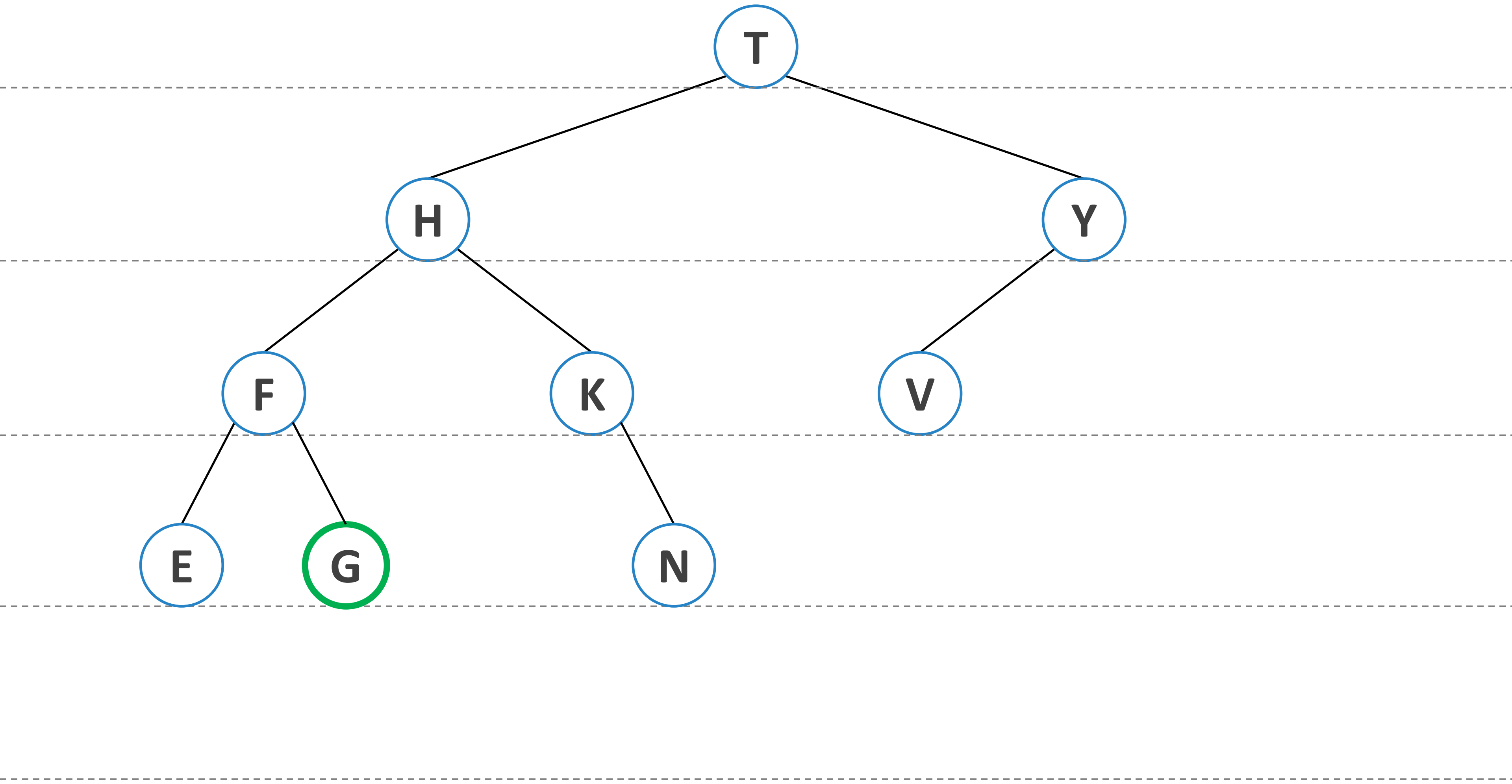
Doble rotación: primero a la izquierda en torno a *Y*-*Z*; luego a la derecha en torno a *X*-*Z*



Vovliendo al ejemplo, luego de insertar *G*



¡Doble rotación!



X y ***Y*** se definen igual que antes, y ***Z*** sería el siguiente nodo hacia abajo en la ruta de inserción

Tenemos definidos entonces ***X***, ***Y*** y ***Z*** en torno a los que hacemos las rotaciones

Resumen rotaciones

Tenemos entonces 4 casos de desbalance, que podemos definir según la ruta que toma la 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

Propiedades de las rotaciones

¿Qué tan costoso es rebalancear el árbol?

¿Cuántas rotaciones es necesario hacer en el peor caso?

Costo de rebalancear

Hacer una rotación tiene costo constante.

Al hacer estas rotaciones para el ***X*** que definimos, se soluciona el desbalance de ***X*** y **no es posible crear un nuevo desbalance,** por lo que siempre en el peor caso realizaremos una sola rotación (simple o doble)

Costo de la inserción

Luego de insertar entonces debemos revisar hacia arriba buscando el primer desbalance. El peor caso es que no haya un desbalance y lleguemos hasta la raíz buscando

Esto significa que toda la inserción sigue siendo $O(h)$

Altura de un árbol AVL

La complejidad sigue dependiendo de la altura del árbol

¿Pero cuál es la altura de un árbol AVL?

Altura de un árbol AVL

Podemos pensarlo al revés

¿Cuál es el máximo de nodos de un árbol de altura h?

¿Y el mínimo?

Altura de un árbol AVL: Mejor caso

El mejor caso sería un árbol lleno, es decir, $n = 2^h - 1$

$n \in \Theta\left(2^{\mathrm{h}}\right)$, por lo que $2^h \in \Theta(n)$

Como ambas funciones son crecientes, $h \in \Theta(\log n)$

Altura de un árbol AVL: Peor caso

Sea $m(h)$ la cantidad mínima de nodos que puede tener un árbol AVL de altura h

$$m(h) = \begin{cases} h, & h \leq 1 \\ 1 + m(h-1) + m(h-2), & h > 1 \end{cases}$$

Esta recurrencia se parece a la secuencia de Fibonacci

$$F(h) = \begin{cases} 1 & h \leq 1 \\ F(h-1) + F(h-2), & h > 1 \end{cases}$$

Es decir, para $h > 1,\ m(h) > F(h)$, por lo que $m(h) \in \Omega\big(F(h)\big)$

Altura de un árbol AVL: Peor caso

$F(h)$ tiene una expresión matemática:

$$F(h) = \left\lfloor \frac{\varphi^h}{\sqrt{5}} \right\rceil \leq \frac{\varphi^h}{\sqrt{5}}$$

Con $\varphi = \frac{1+\sqrt{5}}{2}$. Como $\varphi < 2, F(h) \in \Theta\left(2^h\right)$

Como para $h > 1, m(h) > F(h)$, tenemos que $m(h) \in \Omega\left(2^h\right)$

Altura de un árbol AVL: Peor caso

Para un h, definimos $n = m(h)$. Como dijimos, $n \in \Omega\left(2^h\right)$

Por definición, esto significa que existe un h_0 y un k tal que para todo $h > h_0$:

$$n > k \cdot 2^h$$

Desarrollando, tenemos que:

$$\frac{n}{k} > 2^h$$

Altura de un árbol AVL: Peor caso

Como ambas funciones son crecientes, desde $n > 2^{h_0}$,

$$h < \log_2\left(\frac{n}{k}\right) = \log_2 n - \log_2 k$$

Por lo tanto,

$$h \in O(\log n)$$

Altura de un árbol AVL

La altura h de un árbol AVL de n nodos es $O(\log n)$ en el mejor y el peor caso.

Por lo tanto:

$$h \in O(\log n)$$

Árbol binario:

- cada nodo *x* tiene a lo más dos hijos, uno izquierdo y otro derecho,
- … que, si están, son raíces de los subárboles izquierdo y derecho de *x*

Árbol binario de búsqueda (ABB):

- la clave almacenada en un nodo *x* es mayor (o igual) que cualquiera de las claves almacenadas en el subárbol izquierdo de *x*
- … y menor (o igual) que cualquiera de las claves almacenadas en el subárbol derecho de *x*

ABB balanceado:

- cumple una propiedad adicional de balance
- p.ej., en un árbol AVL, para cualquier nodo del árbol, las alturas de los subárboles izquierdo y derecho pueden diferir a lo más en 1