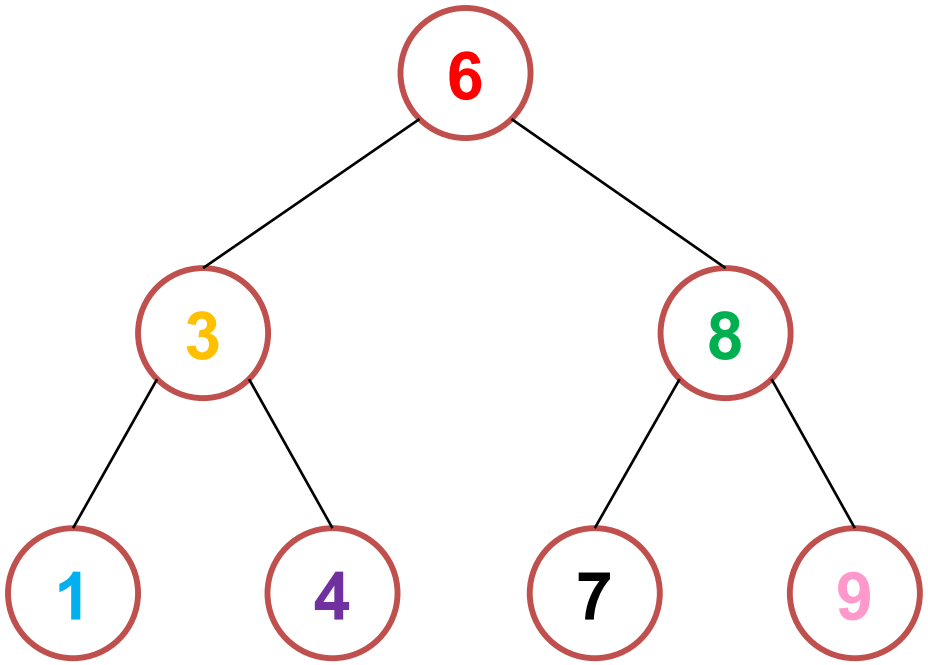
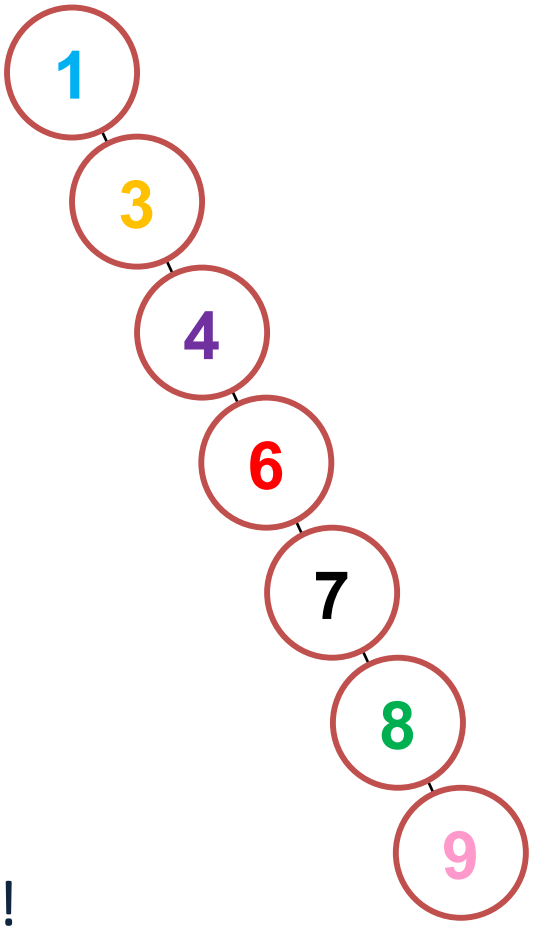
Mismos datos, distintos árboles





¡Ya no podemos garantizar altura $O(\log n)$!

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

Para esto, **es necesario mantenerlos balanceados**

La propiedad de balance debe cumplir dos condiciones

- debe asegurar que la profundidad de un árbol con *n* nodos sea O(log*n*)
- debe ser fácil de mantener —la complejidad de esta operación no puede ser mayor que O(log*n*)

P.ej., exigir que los subárboles izquierdo y derecho tengan la misma altura no es práctico

Es una idea simple

… pero la estructura recursiva de los árboles binarios implica que esta idea se aplique a todos los nodos —cada uno es raíz de un ABB

Asegura que la profundidad es logarítmica

… pero es demasiado restrictiva

Estudiaremos dos tipos de árboles de búsqueda —uno binario, el otro no— que mantienen el balance

… a un costo razonable:

- AVL, árbol binario en que el balance está definido como una diferencia entre las alturas de los subárboles izquierdo y derecho de un nodo que no puede superar un cierto valor máximo
- 2-3, árbol (no binario) en que el balance consiste en que todas las hojas están a la misma profundidad (desde la raíz), pero los nodos pueden almacenar una o dos claves y tener dos o tres hijos

Un **árbol AVL** es un ABB que cumple con la *propiedad de balance* AVL:

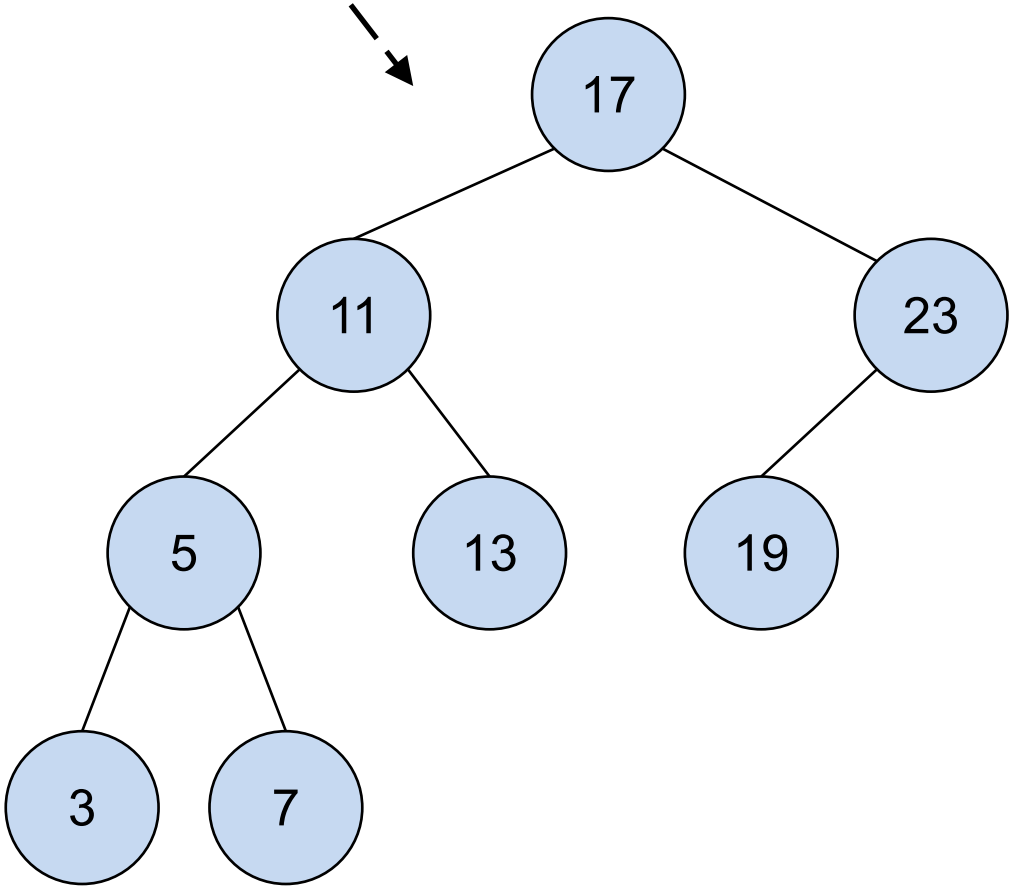
> *Para cualquier nodo del árbol, las alturas de los subárboles izquierdo y derecho pueden diferir a lo más en 1*

Fueron propuestos por G. Adel'son-Vel'skii y E. Landis en 1962

La propiedad de balance AVL implica que el árbol tiene altura logarítmica:

- ver diaps. #54 y 55

Árbol AVL



find (la búsqueda) no cambia

… y ahora es O(log*n*) en el peor caso

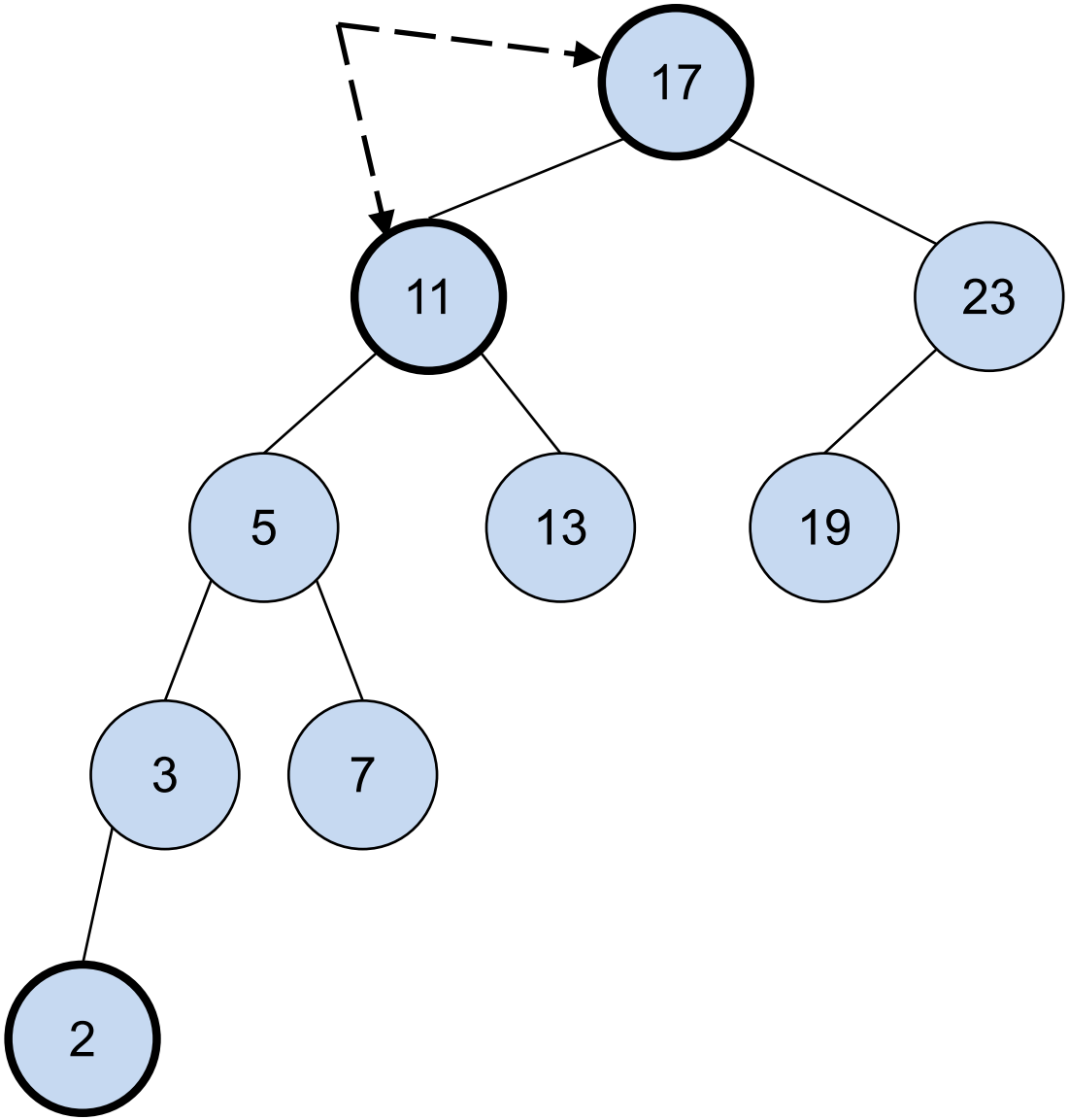
Las operaciones *insert* y *delete* no son tan simples como antes

Una inserción o una eliminación puede destruir el balance de varios nodos:

> … p.ej., en la diapositiva siguiente, la inserción del nodo (con clave) 2 produce desbalance en los nodos 11 y 17

El balance **debe ser restaurado** antes de que la operación pueda considerarse completa

Nodos desbalanceados luego de
la inserción del nodo con clave 2



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

Con esto, después de una inserción

… sólo los nodos que están en la ruta desde el punto de inserción hacia la raíz podrían haber quedado desbalanceados:

- sólo esos nodos tienen sus subárboles modificados

Siguiendo esa ruta, sea *x* el primer nodo (el más profundo) desbalanceado:

- *x.balance* < –1, o *x.balance* > +1
- si rebalanceamos *x*, habremos rebalanceado todo el árbol

Como *x* tiene a los más dos hijos y el desbalance significa que las alturas de los dos subárboles de *x* difieren en 2, hay cuatro casos de inserción posibles:

1) en el subárbol izquierdo del hijo izquierdo de *x*

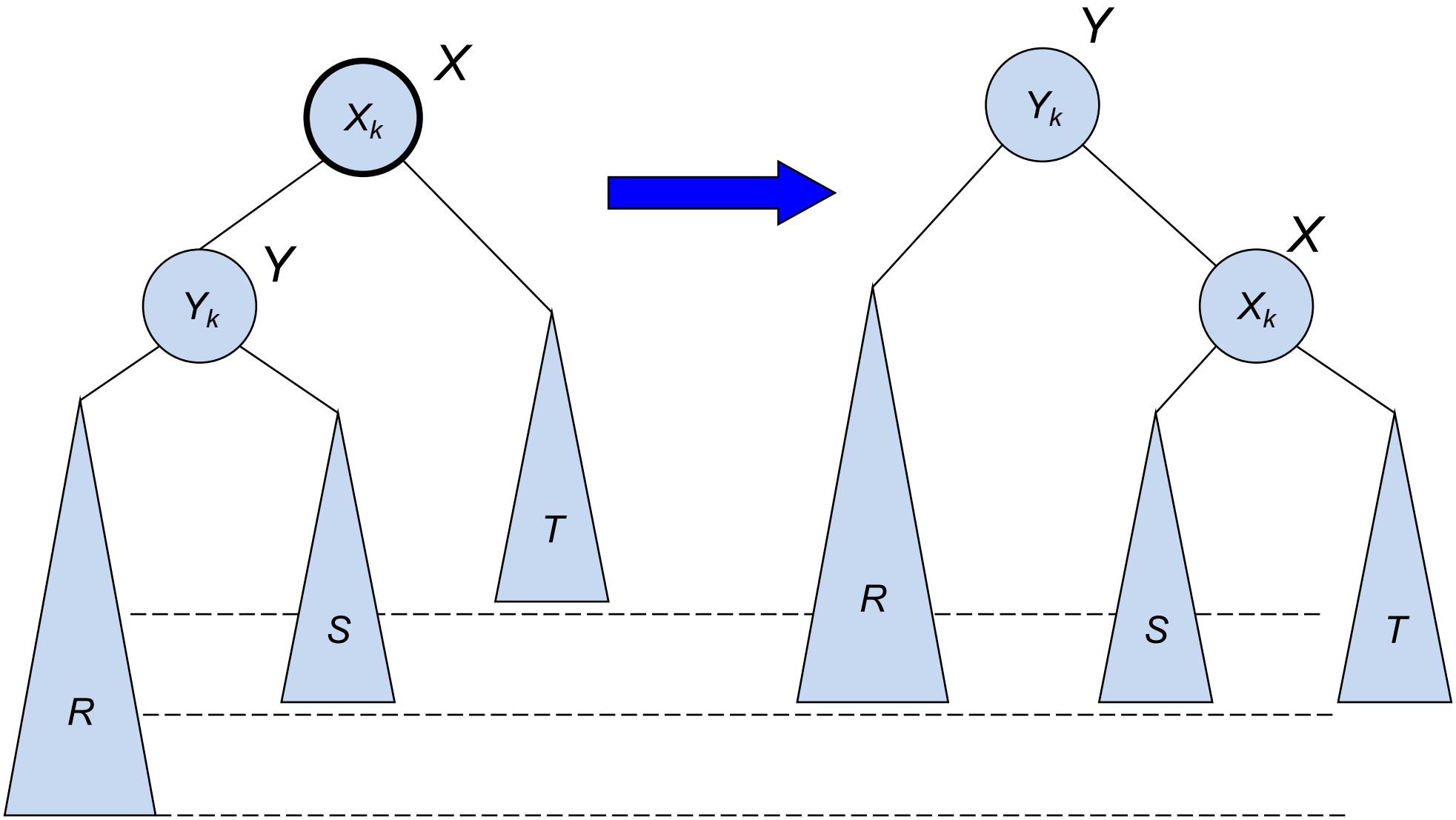
2) en el subárbol derecho del hijo izquierdo de *x*

3) en el subárbol izquierdo del hijo derecho de *x*

4) en el subárbol derecho del hijo derecho de *x*

Los casos de un mismo color (1 y 4, y 2 y 3) son simétricos entre ellos con respecto a *x*

Una **rotación simple** resuelve el caso básico A; p.ej., una *rotación simple a la derecha* en torno a *X*–*Y* para resolver el caso 1:



La **rotación** preserva la propiedad de ABB:

R sigue siendo el subárbol izquierdo del nodo *Y* (clave Y_k)

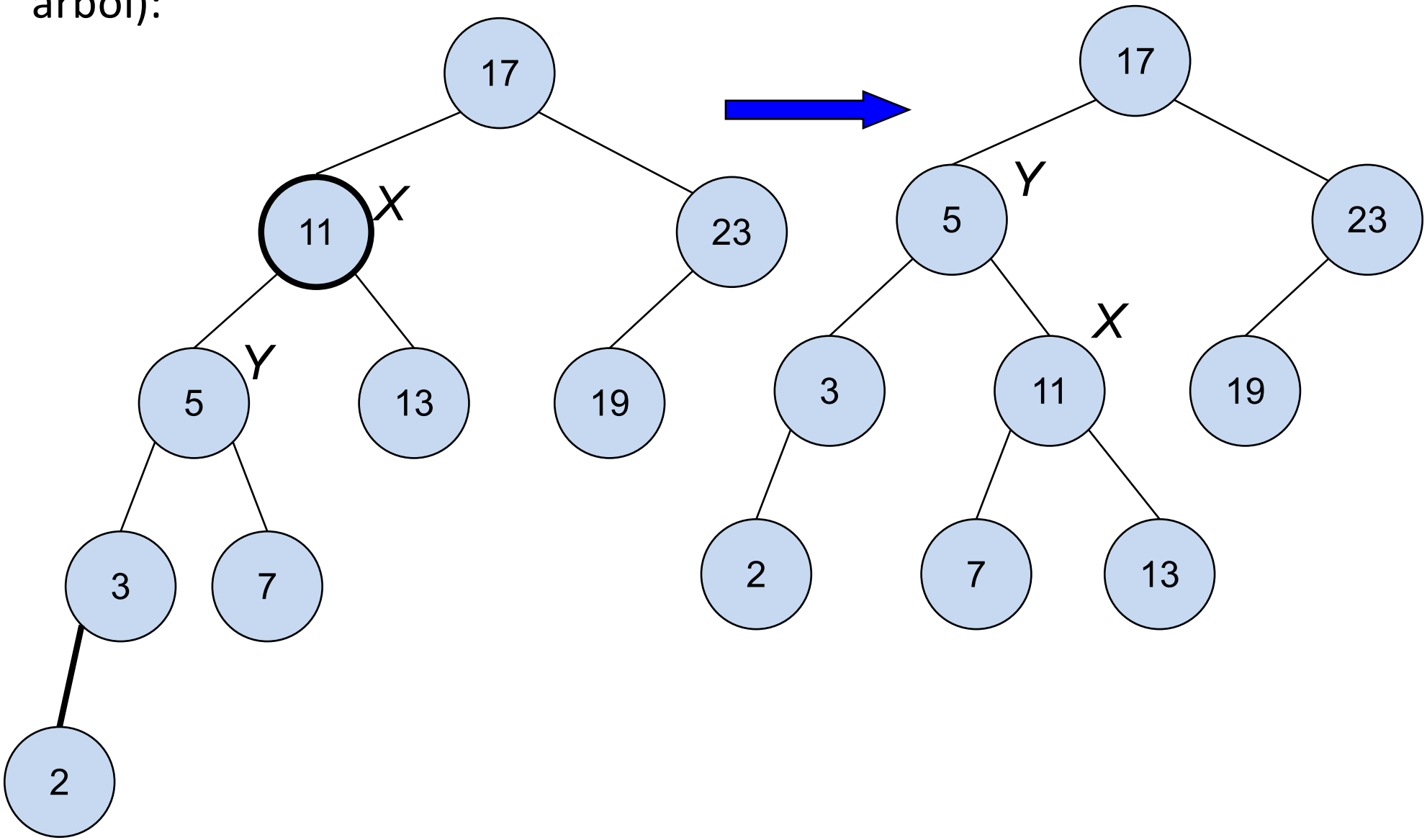
T sigue siendo el subárbol derecho del nodo *X* (clave X_k)

…:

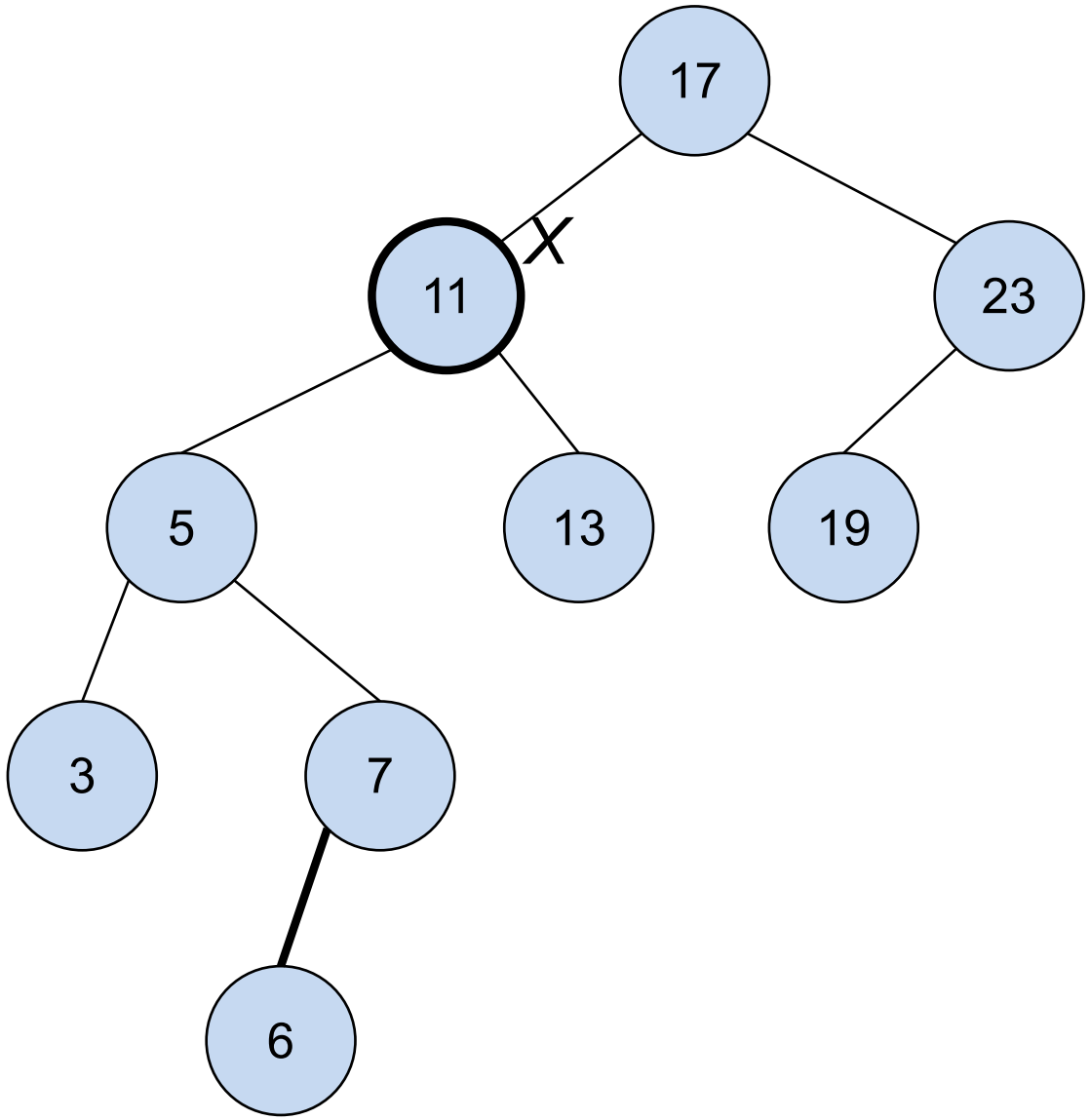
Como en el árbol original Y es el hijo izquierdo de $X \Rightarrow Y_k < X_k$ $\Rightarrow X$ puede ser hijo derecho de Y en el árbol rotado

En el árbol original las claves en S son mayores que Y_k (S es el subárbol derecho de Y) y son menores que X_k (están en el subárbol izquierdo de X) $\Rightarrow S$ puede ser el subárbol izquierdo de X en el árbol rotado

Ejemplo del caso A: El nodo con clave 2 fue insertado en el árbol de la izquierda, rompiendo el balance del nodo *X* (y la propiedad AVL del árbol):



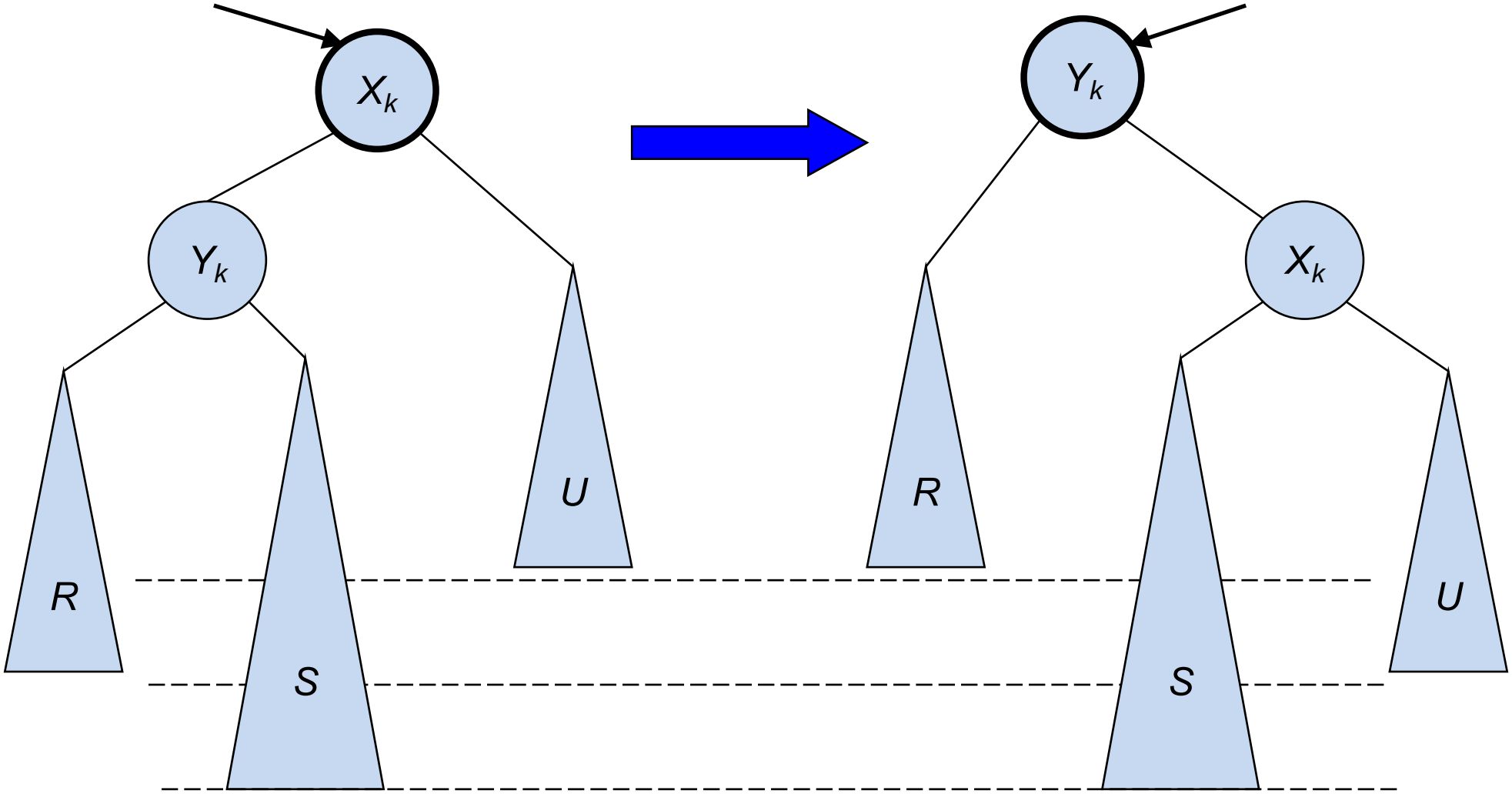
Ejemplo del caso B:



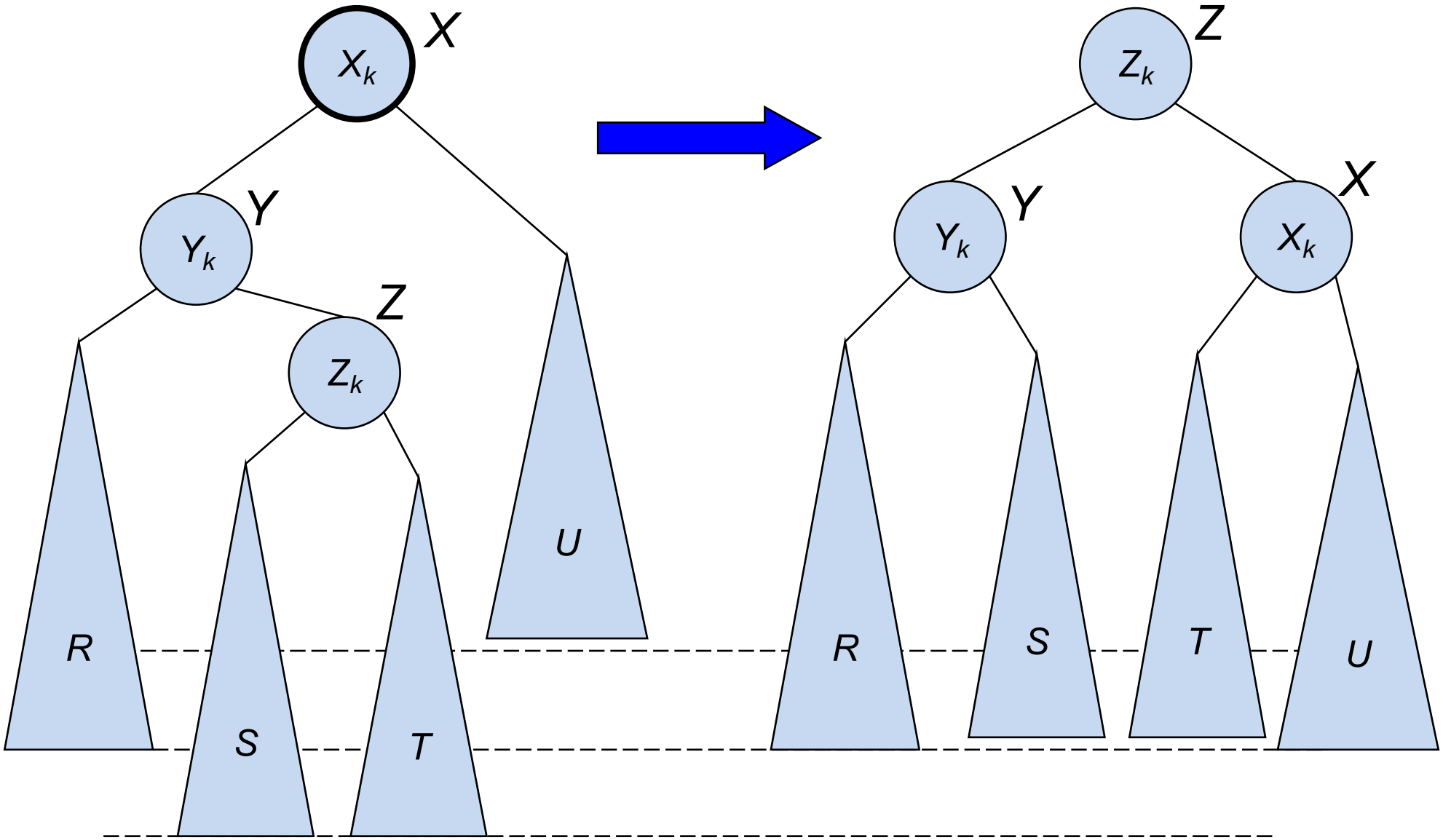
Una rotación simple *no* resuelve el caso básico B:

Nodo desbalanceado antes de la rotación

Nodo desbalanceado después de una rotación simple



Una **rotación doble** resuelve el caso básico B (el nodo insertado está en *S* o en *T*, pero no en ambos):



En el árbol original, *S* o *T* (pero no ambos), tiene una profundidad igual a la de *U* más dos

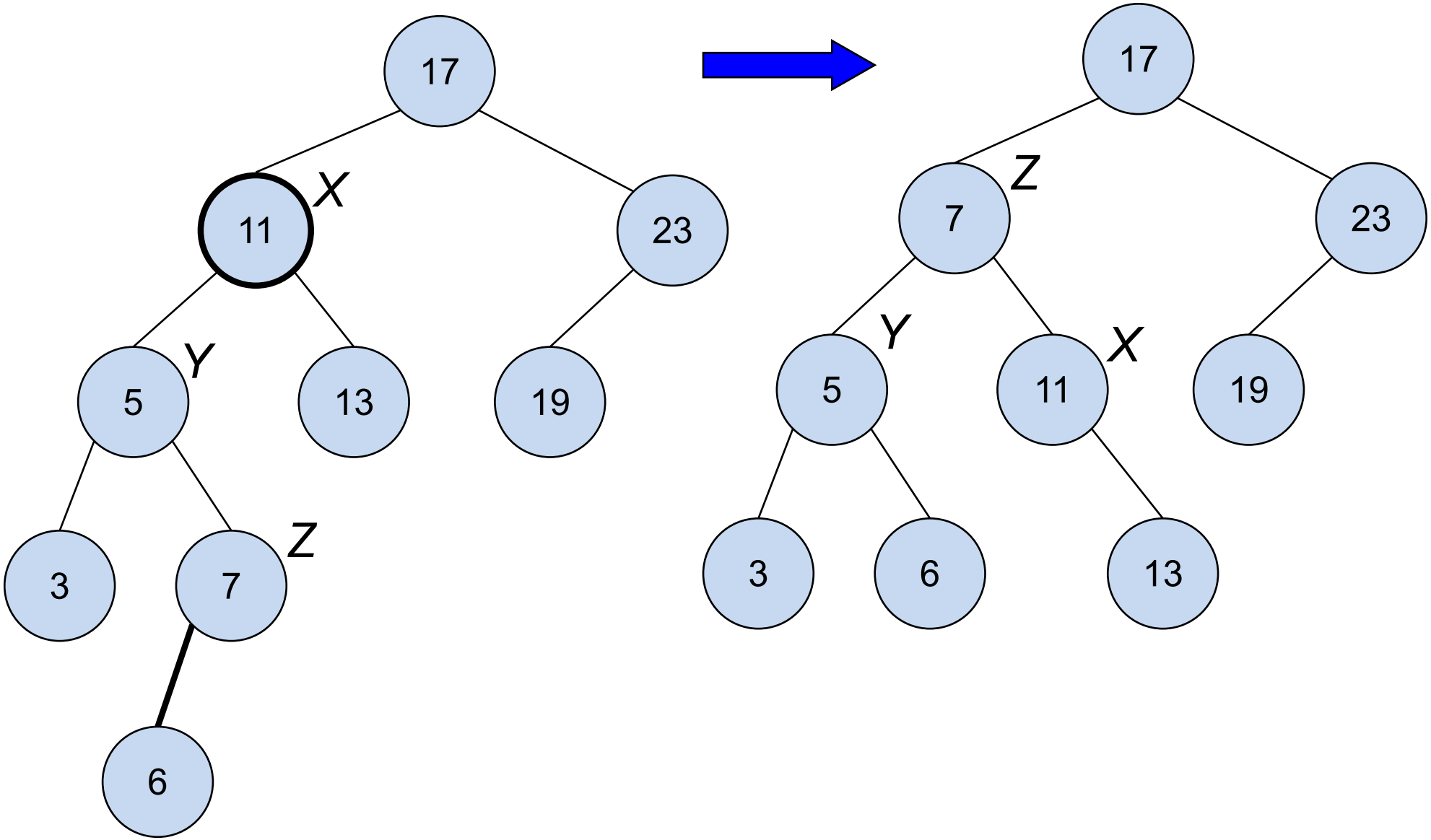
Primero, se hace una rotación simple a la izquierda en torno a *Y*–*Z*, sin involucrar a *X* (no se muestra):

- *S* se convierte en el subárbol derecho de *Y*
- *Y* se convierte en el hijo izquierdo de *Z*
- *Z* reemplaza a *Y* como hijo izquierdo de *X*

Luego, se hace una rotación simple a la derecha en torno a *X*–*Z*:

- *X* se convierte en el hijo derecho de *Z*
- *T* se convierte en el subárbol izquierdo de *X*
- *Z* ocupa el lugar que tenía *X* en el árbol original

Solución del ejemplo del caso B:



La altura de un árbol AVL es Θ(log*n*), como se demuestra a continuación

Un árbol AVL de altura h tiene al menos $F_{h+3} - 1$ nodos:

- F_i es el i-ésimo número de Fibonacci

Sea S_h el número de nodos del árbol AVL más pequeño de altura h:

$$S_0 = 1 \qquad S_1 = 2$$

Este árbol debe tener subárboles de alturas $h-1$ y $h-2$:

- al menos un subárbol tiene altura $h-1$
- la propiedad de balance implica que las alturas de los subárboles pueden diferir a lo más en 1

Estos subárboles deben tener el menor número de nodos para sus alturas:

$$S_h = S_{h-1} + S_{h-2} + 1$$

Por inducción, podemos probar la afirmación inicial

Sabemos que $F_k = \emptyset^k / \sqrt{5}$, en que $\emptyset = (1+\sqrt{5})/2 = 1.618\ldots$ (se puede demostrar por inducción)

Luego, un árbol AVL de altura h tiene al menos $\emptyset^{h+3} / \sqrt{5}$ nodos

Por lo tanto, la altura de un árbol AVL de n nodos es logarítmica con respecto a 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