Splayträd
Weiss, avsnitt 4.5

Peter Ljunglöf

DAT036, Datastrukturer

3 dec 2012

Splayträd vs AVL

Både AVL-träd och splayträd är balanserade träd.

Men:

- AVL-träd balanserar endast vid insättning och borttagning
- Splayträd balanserar även vid uppslagning!
- Splayträd har ingen information om nodbalans!

Splayträd

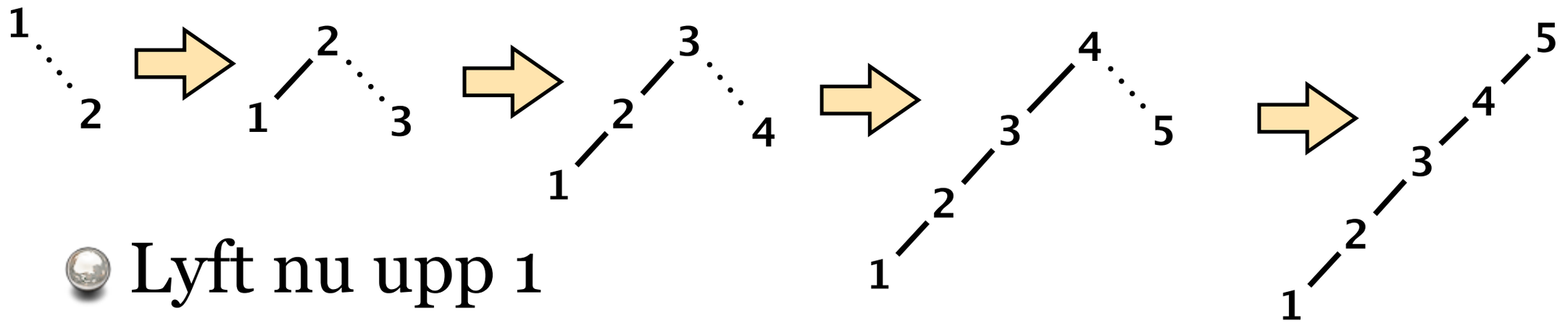
Grundidén är att när vi slår upp ett element *x*, så flyttar vi samtidigt upp *x* till rotnoden.

- Medan vi gör detta måste vi balansera om de interna noderna
- Förhoppningen är att med tiden så kommer trädet att bli någorlunda balanserat
- Insättning, uppslagning och borttagning blir O(log N), *i amorterad tid*

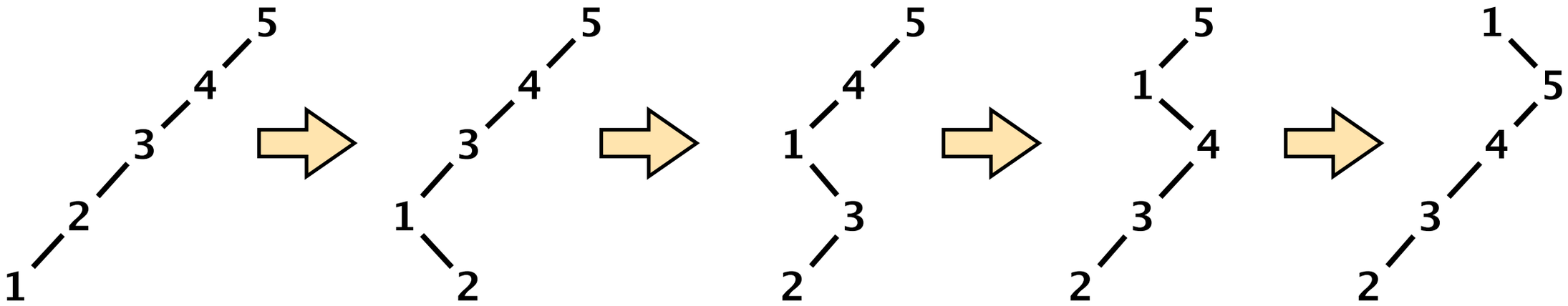
Enkelrotation går inte

Tyvärr går det inte med enkla rotationer. Exempel:

- Bygg först trädet från 1 2 3 4 5



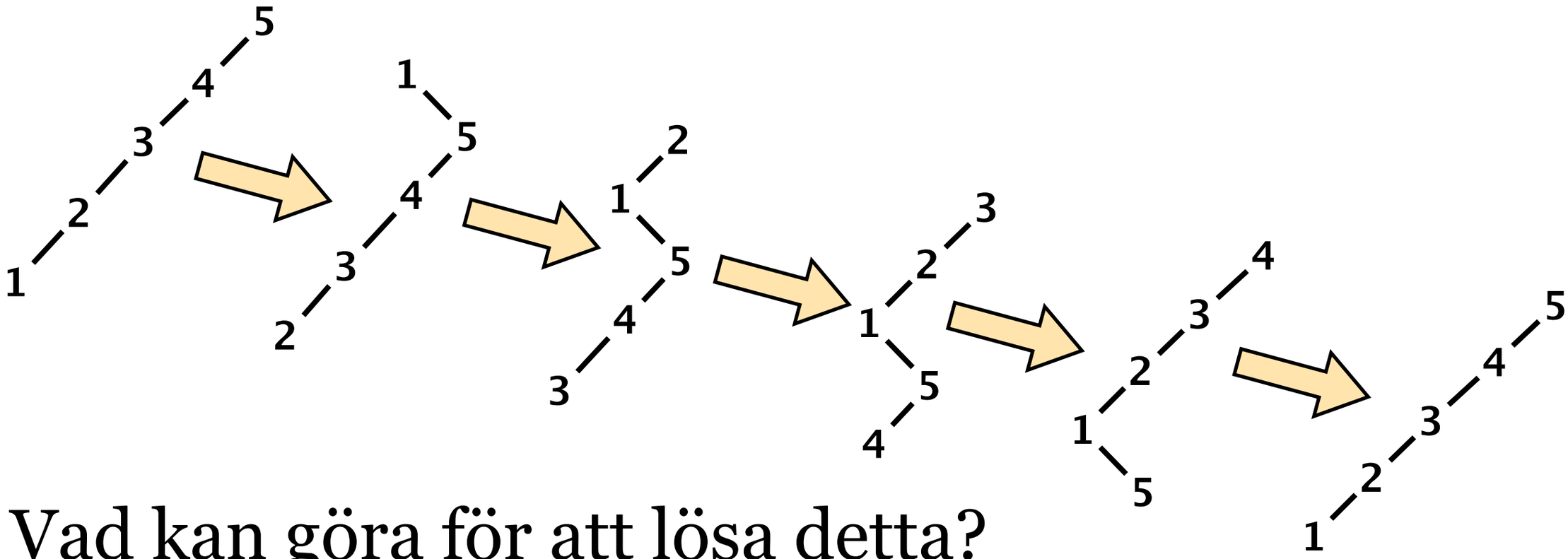
- Lyft nu upp 1



Enkelrotation går inte

Antag trädet byggt från 1, 2, 3, 4, 5

- lyft upp 1, sedan 2, sedan 3, 4 och sist 5



Vad kan göra för att lösa detta?

- **Svar**: splayning!

Splayning

Splayning motsvarar AVL-trädens dubbelrotation
Det finns fyra fall:

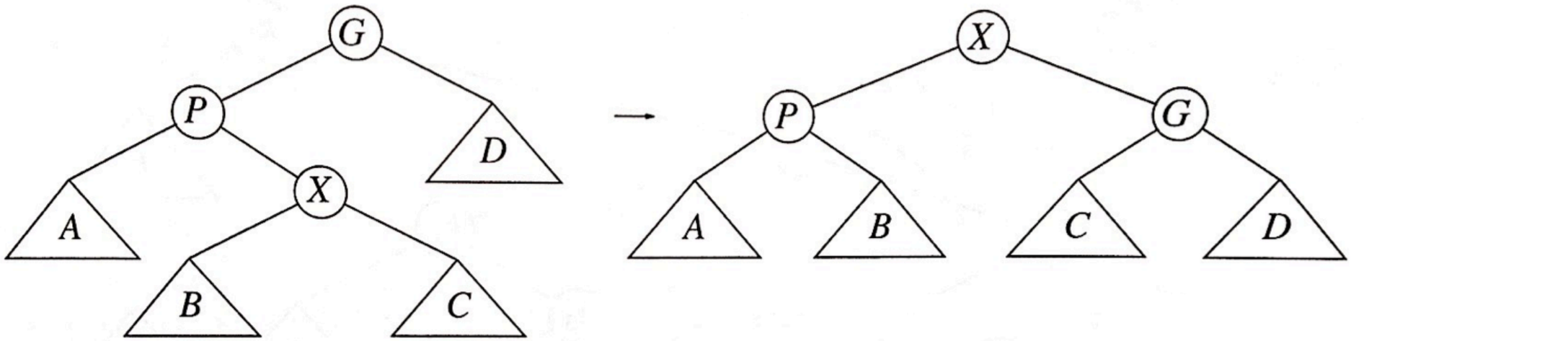


Figure 4.44 Zig-zag

Zag-zig är spegelvänt

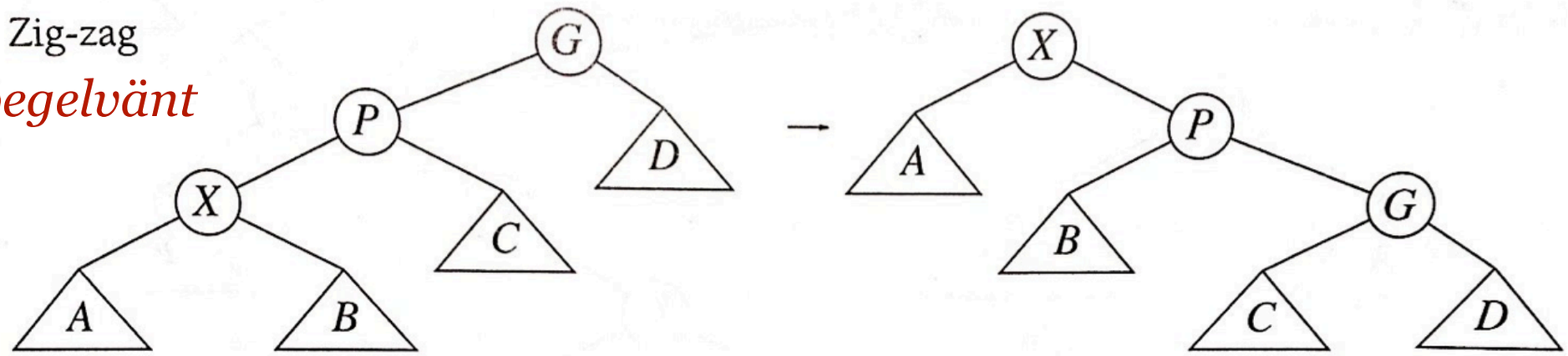


Figure 4.45 Zig-zig

Zag-zag är spegelvänt

Exempel

Nu ska vi bygga ett splayträd för exemplet i figur 4.46:

1 2 3 4 5 6 7

Vi gör det med samma Java-applet som förut:

http://people.ksp.sk/~kuko/bak/index.html

Det större exemplet i figur 4.47–4.55:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32

”The fundamental and crucial property of splay trees”

”When access paths are long (leading to longer-than-normal search time), the rotations tend to be good for future operations.”

- dvs, de leder till balanserade träd

”When accesses are cheap, the rotations can be bad.”

- dvs, de kan leda till obalanserade träd

”The main theorem, is that we never fall behind a pace of O(log N) per operation.”

- ”We are always on schedule, even though there are occasionally bad operations.”
- O(log N) i amorterad tid alltså
- beviset finns i avsnitt 11.5 i kursboken

Insättning och borttagning

För att stoppa in ett element x, så letar vi först upp den nod y där vi borde stoppa in x:

- splaya upp y till roten
- lägg till x som en ny rot och y som ett barn

För att ta bort ett element x, så splayar vi först upp det till roten, med delträd T_L och T_R:

- sedan hittar vi det största värdet m i det vänstra delträdet T_L och splayar upp det (till T_L')
- då kommer T_L' att inte ha något högerbarn, och vi kan låta T_R bli dess högerbarn
- den slutliga roten blir m med T_L' och T_R som barn