

Algorithmen und Datenstrukturen

Sommersemester 2021

Stefan Näher

Universität Trier

naeher@uni-trier.de

Vorlesung 11

12. Mai 2021

Suche in einem binären Suchbaum

LOOKUP(x) liefert den Knoten der x enthält oder null, falls x nicht in T .

```
1. LOOKUP( $x$ )
2.  $p \leftarrow \text{root}$ ;
3. while  $p \neq \text{null} \wedge p.\text{key} \neq x$  do
4.   if  $x < p.\text{key}$ 
5.   then  $p \leftarrow p.\text{left}$ ;
6.   else  $p \leftarrow p.\text{right}$ ;
7. fi ;
8. od;
9. return  $p$ ;
```

Laufzeit: $\mathcal{O}(\text{Höhe}(T))$.

Einfügen in einen binären Suchbaum

INSERT(x): fügt einen Knoten (genauer Blatt) mit Schlüssel x hinzu (falls $x \notin T$).

1. INSERT(x)
2. // zunächst wie LOOKUP: merke den Parent von p in q
3. $p \leftarrow \text{root}; q \leftarrow \text{null};$
5. **while** $p \neq \text{null} \wedge p.\text{key} \neq x$ **do**
6. $q \leftarrow p;$
7. **if** $x < p.\text{key}$
8. **then** $p \leftarrow p.\text{left};$
9. **else** $p \leftarrow p.\text{right};$
10. **fi** ;
11. **od**;
12. **if** $p \neq \text{null}$ **then return** ;**fi** ; // x gefunden (nichts zu tun)

// füge neues Blatt u ein

13. $u \leftarrow \mathbf{new} \text{ bin_tree_node};$

14. $u.\text{key} \leftarrow x;$

15. $u.\text{left} \leftarrow \text{null};$

16. $u.\text{right} \leftarrow \text{null};$

17. **if** $x < q.\text{key}$

18. **then** $q.\text{left} \leftarrow u;$

19. **else** $q.\text{right} \leftarrow u;$

20. **fi**

Laufzeit: $O(\text{Höhe}(T)).$

Löschen in einem binären Suchbaum

DELETE(x) entfernt den Knoten v mit $v.\text{key} = x$ aus T

Sei v der Knoten der x enthält und p der Vater von v (null, falls $v = \text{root}$);

Wir unterscheiden 3 Fälle

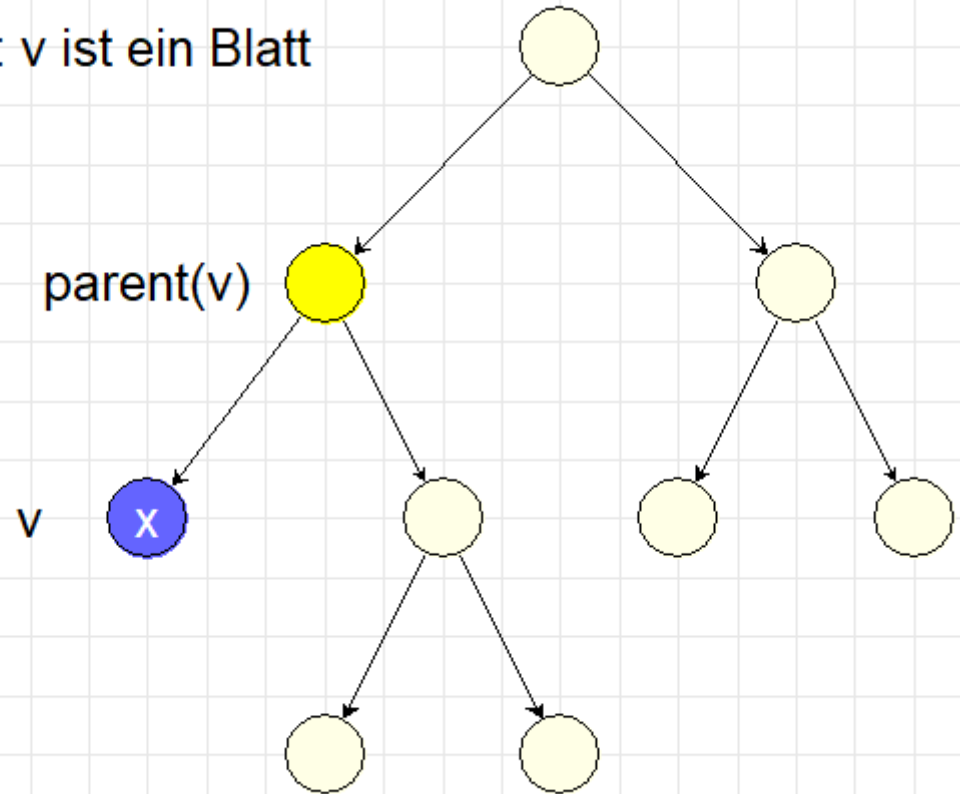
Fall 1: v ist ein **Blatt** ($v.\text{left} = v.\text{right} = \text{null}$)

Fall 2: v hat **ein Kind** u (entweder $v.\text{left} = \text{null}$ oder $v.\text{right} = \text{null}$)

Fall 3: v hat **zwei Kinder** ($v.\text{left} \neq \text{null}$ und $v.\text{right} \neq \text{null}$)

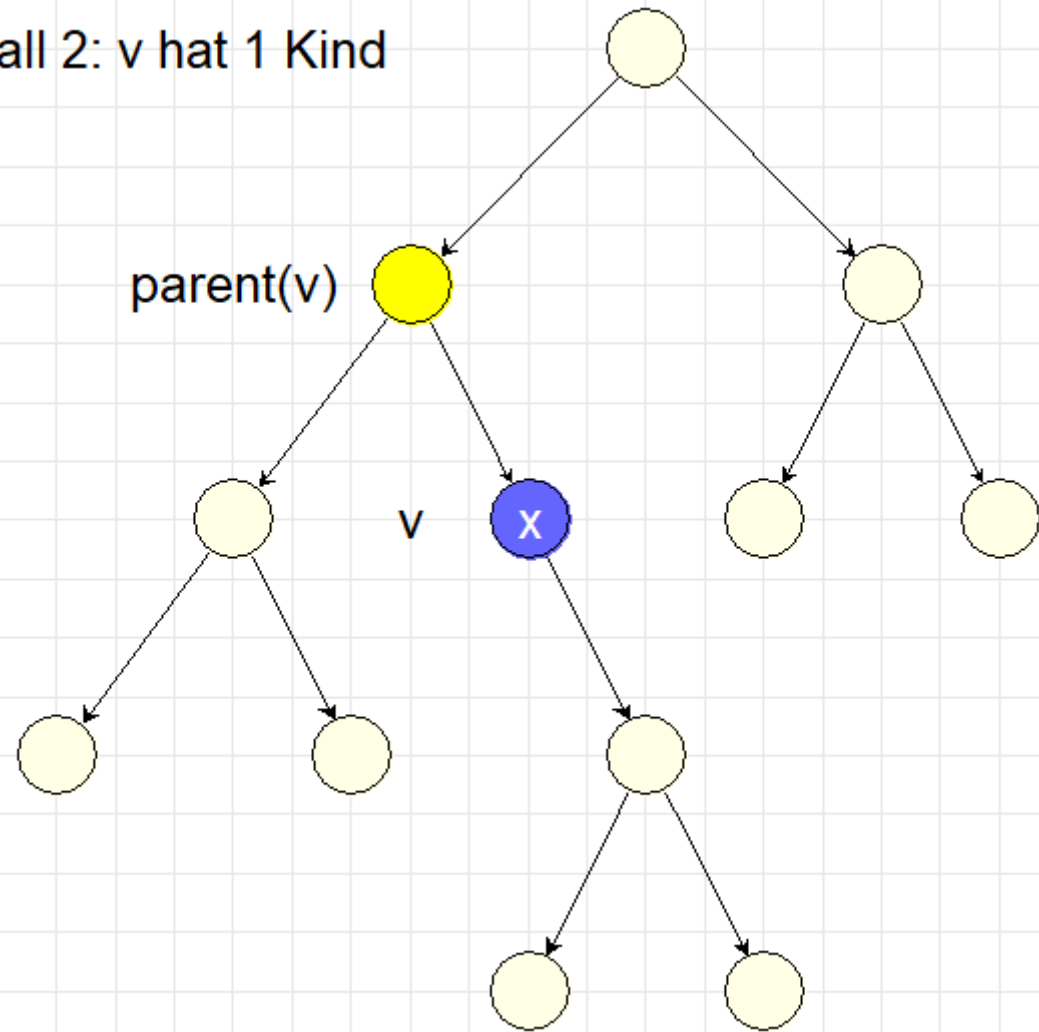
```
if  $v = p.\text{left}$   
then  $p.\text{left} \leftarrow \text{null};$   
else  $p.\text{right} \leftarrow \text{null};$   
fi
```

Fall 1: v ist ein Blatt

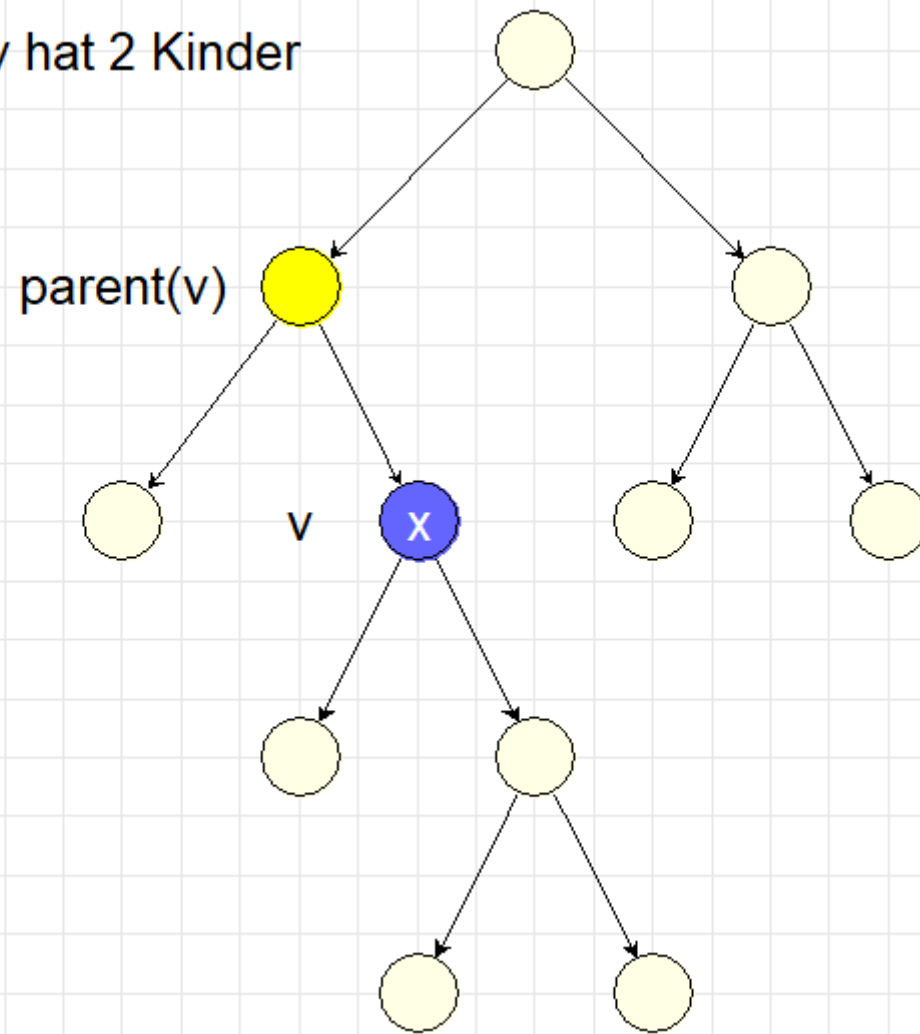


$u \leftarrow \text{Kind von } v;$
if $v = p.\text{left}$
then $p.\text{left} \leftarrow u;$
else $p.\text{right} \leftarrow u;$
fi

Fall 2: v hat 1 Kind



Fall 3: v hat 2 Kinder



Fall 3: v hat 2 Kinder

1. Finde Knoten u mit $u.key$ maximal in linkem Unterbaum

$u \leftarrow v.left;$

while $u.right \neq \text{null}$ **do**

$u \leftarrow u.right;$

od

// $u.key$ ist maximal mit $u.key < x$

// u hat maximal 1 Kind

2. Kopiere Inhalt von Knoten u in den Knoten v

$v.key \leftarrow u.key;$

3. Entferne u gemäß Fall 1 oder 2

 DELETE(u);

Laufzeiten

Alle Wörterbuchoperationen LOOKUP, INSERT, DELETE haben Laufzeit $\mathcal{O}(\text{Höhe}(T))$.

Wie hoch bzw. tief ist ein binärer Suchbaum mit n Knoten ?

Optimal: $\log n$

Worst-Case: $\mathcal{O}(n)$ (degeneriert)

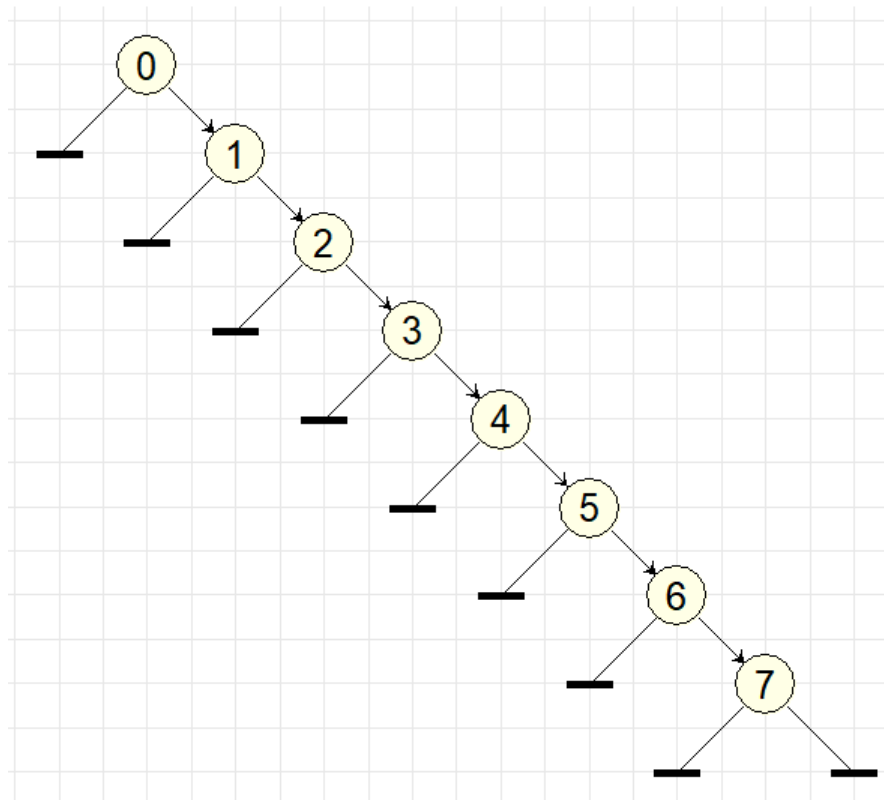
Balanciert: $\mathcal{O}(\log n)$

Weitere Operationen in der Übung:

Minimum, Maximum, Auflisten in aufsteigender Reihenfolge

Ein degenerierter Suchbaum

for $i = 0$ **to** $n - 1$ **do** INSERT(i); **od**



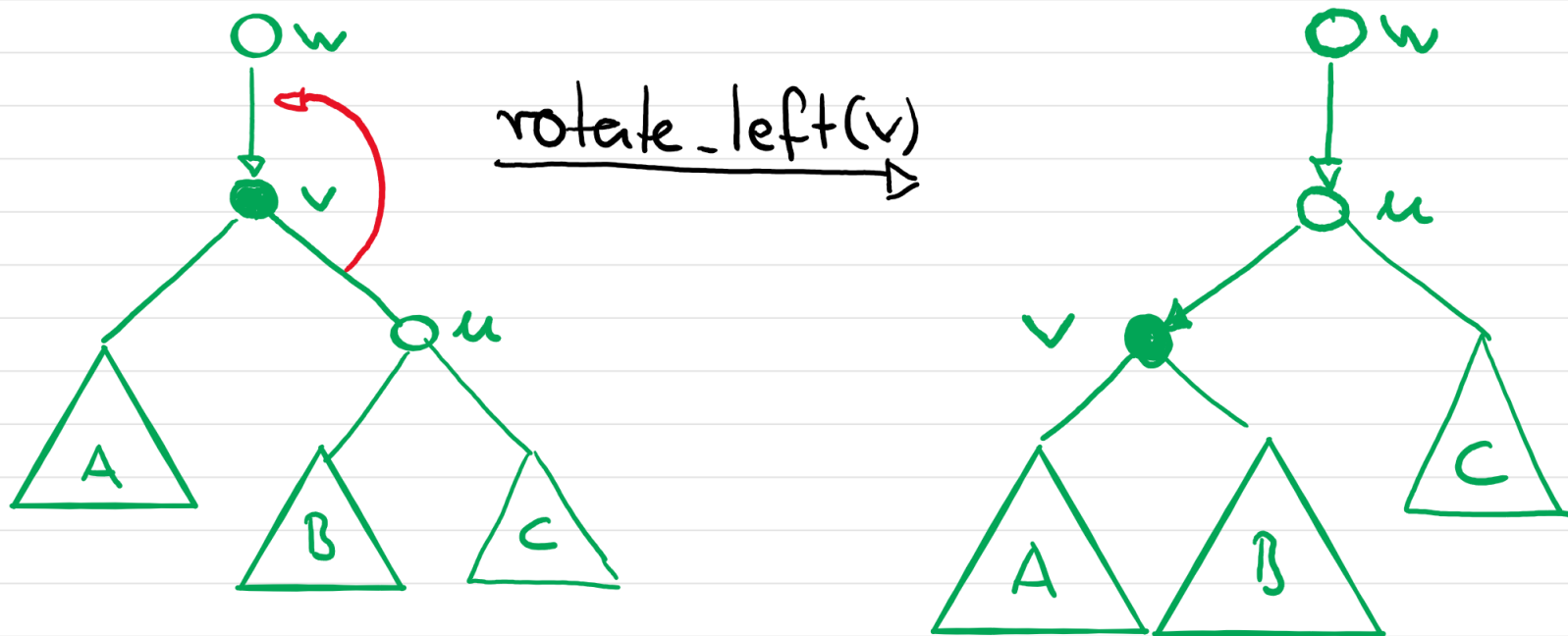
Rebalancierung

Garantiere eine Höhe von $\mathcal{O}(\log n)$ durch lokale Transformationen entlang des Suchpfades nach jeder Update-Operation (INSERT und DELETE).

Lokale Transformationen: Rotationen und Doppelrotationen

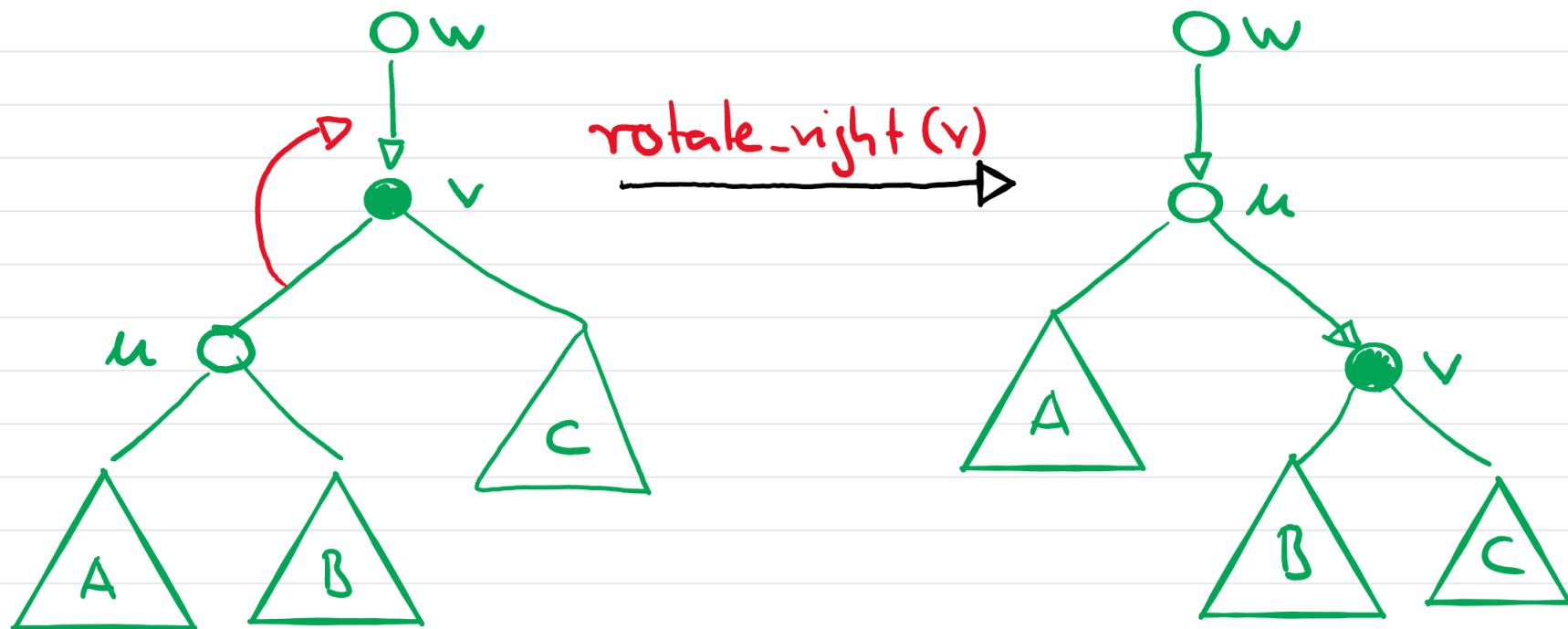
Rotation nach links am Knoten v

Sei w Parent und u rechtes Kind von v



Spezialfall: v ist Wurzel
d.h. w existiert nicht.

Rotation nach rechts am Knoten v

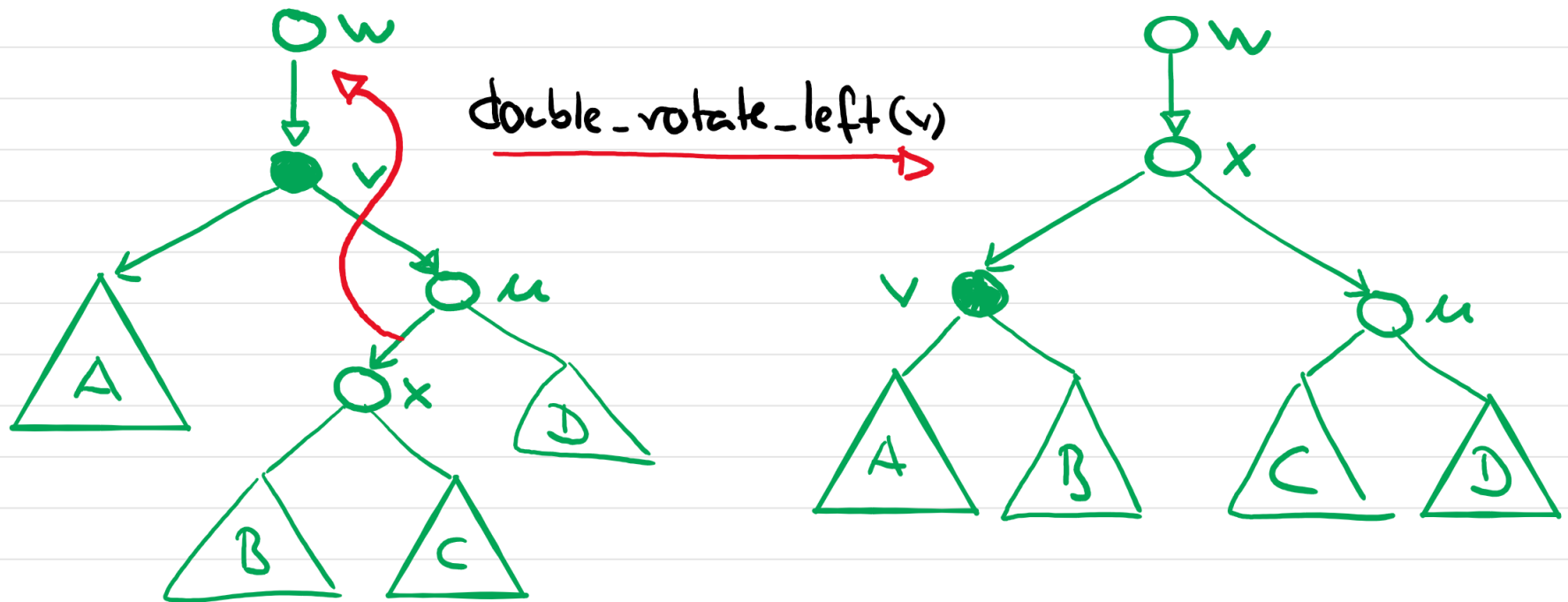


w : Parent(v)

u : linkes Kind(v)

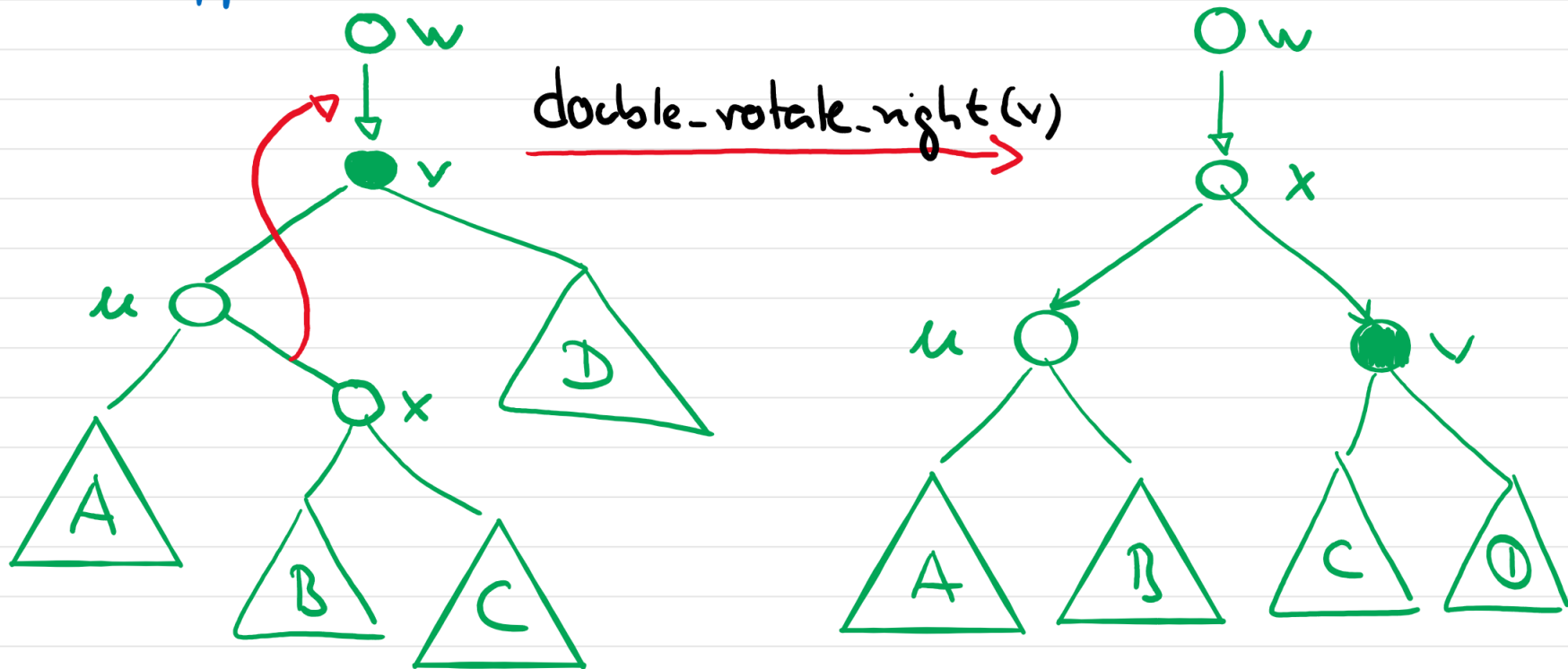
Doppelrotation nach links am Knoten v

Seien w Parent von v , u rechtes Kind von v
und x linkes Kind von u .



→ $\text{rotate_right}(u) + \text{rotate_left}(v)$

Doppelrotation nach rechts am Knoten v



16

→ rotate-left(u) + rotate-right(v)

AVL-Bäume Adelson-Velskij/Landis 1962

Ein **AVL-Baum** ist ein 1-ausgeglichener Suchbaum, d.h., es handelt sich um einen binären Suchbaum, in dem sich für jeden inneren Knoten die Höhen der beiden Teilbäume um **höchstens eins** unterscheiden.

Das Suchen nach einem Element funktioniert weiter wie bei “normalen” Suchbäumen (die Suchbaumeigenschaft ist ja erfüllt).

Beim Einfügen und Löschen muss evtl. durch nachträgliches **Rebalancieren** die 1-Ausgeglichenheit wiederhergestellt werden.

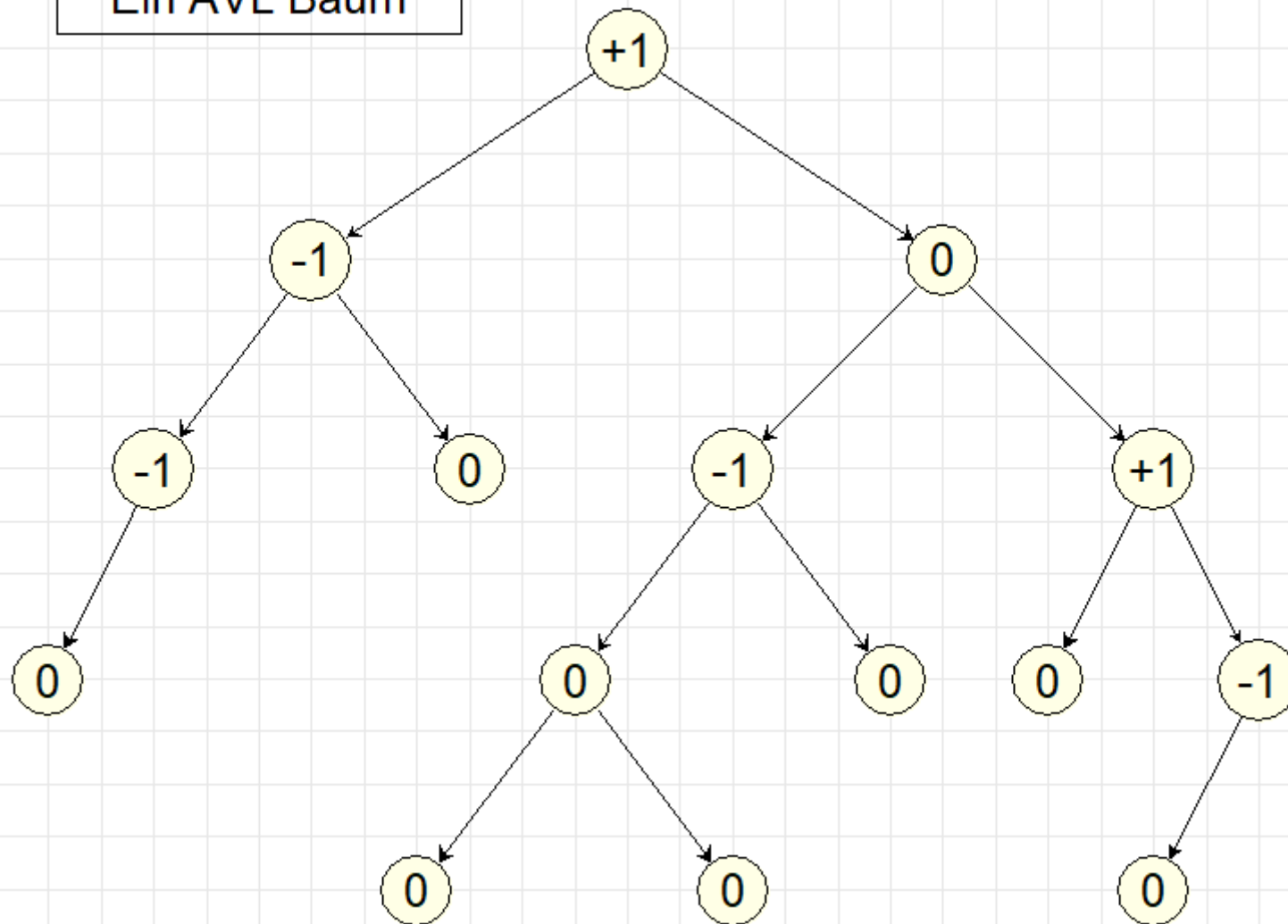
Das wollen wir zunächst beobachten in: <http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm>.

AVL-Bäume: Balance

Jedem Knoten wird als **Balancewert** die Differenz der Höhen des rechten und linken Teilbaumes zugeordnet.

$$\delta(v) = \text{höhe}(T_r(v)) - \text{höhe}(T_\ell(v))$$

Ein AVL Baum



AVL-Bäume: Einfügen

Wie üblich wird zunächst nach der Einfügestelle im Suchbaum gefahndet.

Der auf diese Weise beschriebene Suchpfad wird nach dem Einfügen rückwärts durchlaufen, um nach Knoten zu schauen, die aus der Balance geraten sind.

Wird solch ein Knoten x mit einem Balancewert von 2 oder (-2) gefunden, so muss rebalanciert werden.

Dies geschieht durch Umordnen des an x hängenden Teilbaums, was aber allenfalls die Balancewerte der Enkelknoten beeinflusst, sodass kein erneutes Absteigen nötig ist.

Dadurch kann sich die Höhe des an x hängenden Teilbaumes nicht verringern. Deshalb müssen wir beim Einfügen nicht weiter bis zur Wurzel hochsteigen, um möglicherweise weitere Rebalancierungen durchzuführen.

O.E. diskutieren wir ein x mit Balancewert 2 auf den nächsten Folien.