AuD: große Übung
18.12.2014

Christian Scheffer

- Bäume
- Bin. Baume
- Bin. Suchbäume
- AVL-Bäume

Baum:

Baum:

- gerichteter Graph $G = (V, E)$

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- mit $\forall v \in V \setminus \{w\} : \exists!(v', v) \in E$

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- mit $\forall v \in V \setminus \{w\} : \exists!(v', v) \in E$

Jeder Knoten (außer) Wurzel
ist Kind (eind. Vater)

Baum:

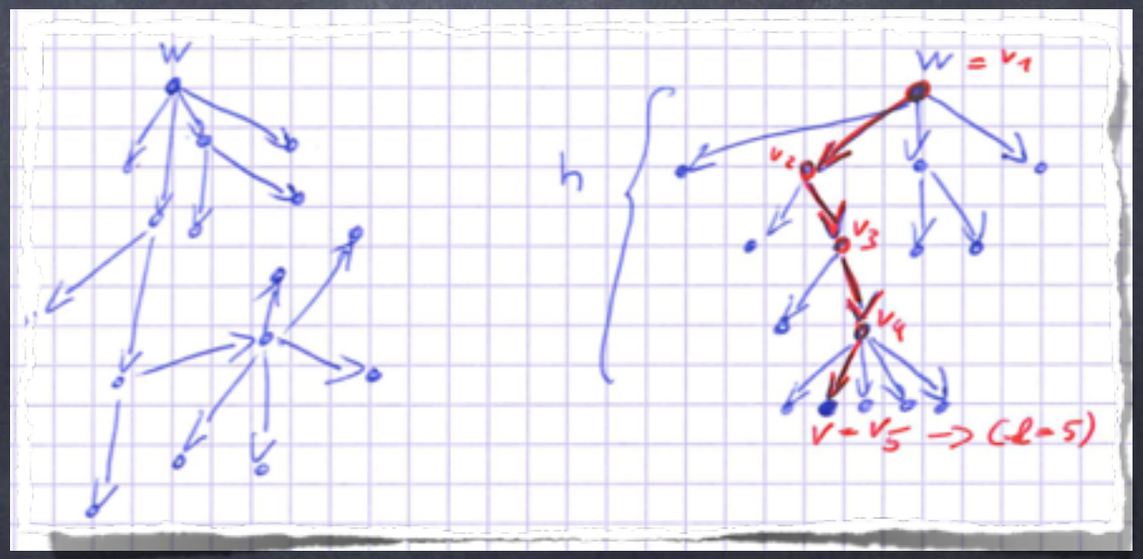
- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- mit $\forall v \in V \setminus \{w\} : \exists! (v', v) \in E$

  Jeder Knoten (außer) Wurzel ist Kind (eind. Vater)
- $\forall v \in V \setminus \{w\} : \exists$ Folge von Kanten $(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$ mit $v_1 = w, v_k = v$

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- mit $\forall v \in V \setminus \{w\} : \exists! (v', v) \in E$

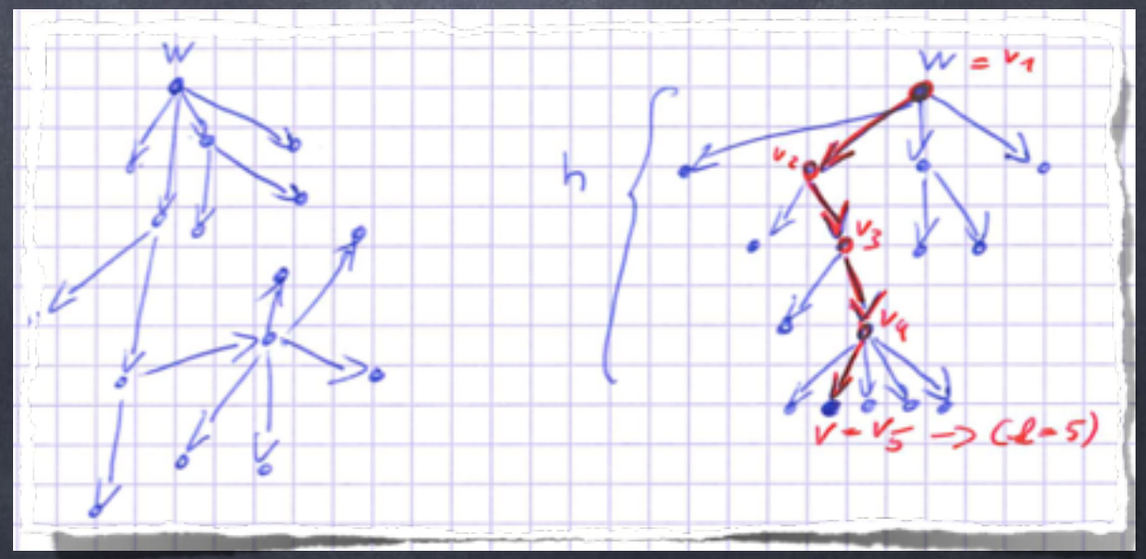
  Jeder Knoten (außer) Wurzel ist Kind (eind. Vater)
- $\forall v \in V \setminus \{w\} : \exists$ Folge von Kanten $(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$ mit $v_1 = w, v_k = v$



Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- mit $\forall v \in V \setminus \{w\} : \exists! (v', v) \in E$

  Jeder Knoten (außer) Wurzel ist Kind (eind. Vater)
- $\forall v \in V \setminus \{w\} : \exists$ Folge von Kanten $(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$ mit $v_1 = w, v_k = v$
- Folge eindeutig.

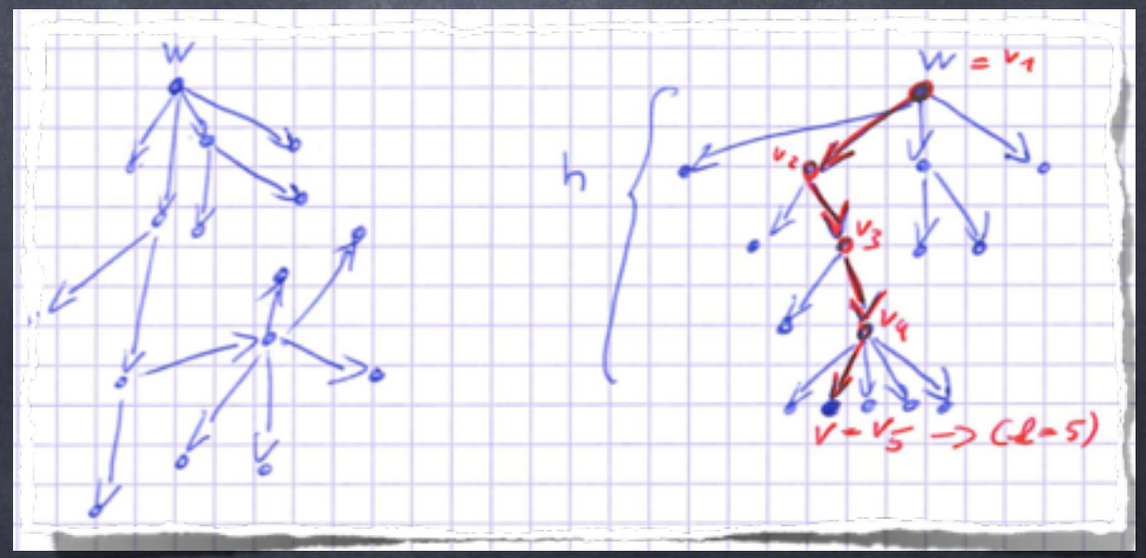


Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- mit $\forall v \in V \setminus \{w\} : \exists! (v', v) \in E$

  Jeder Knoten (außer) Wurzel ist Kind (eind. Vater)
- $\forall v \in V \setminus \{w\} : \exists$ Folge von Kanten $(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$ mit $v_1 = w, v_k = v$
- Folge eindeutig.

  Warum?

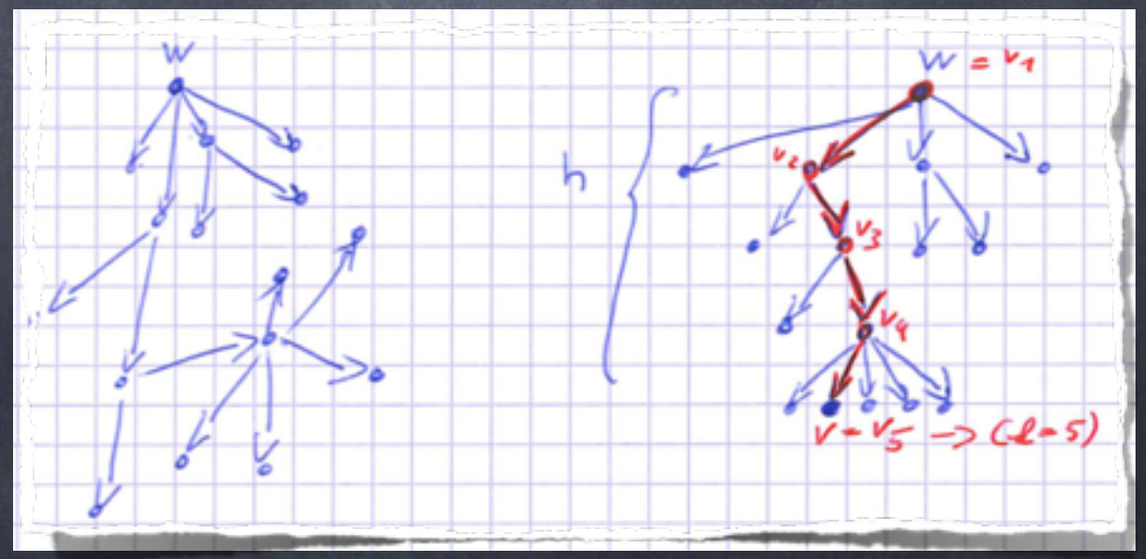


Baum:

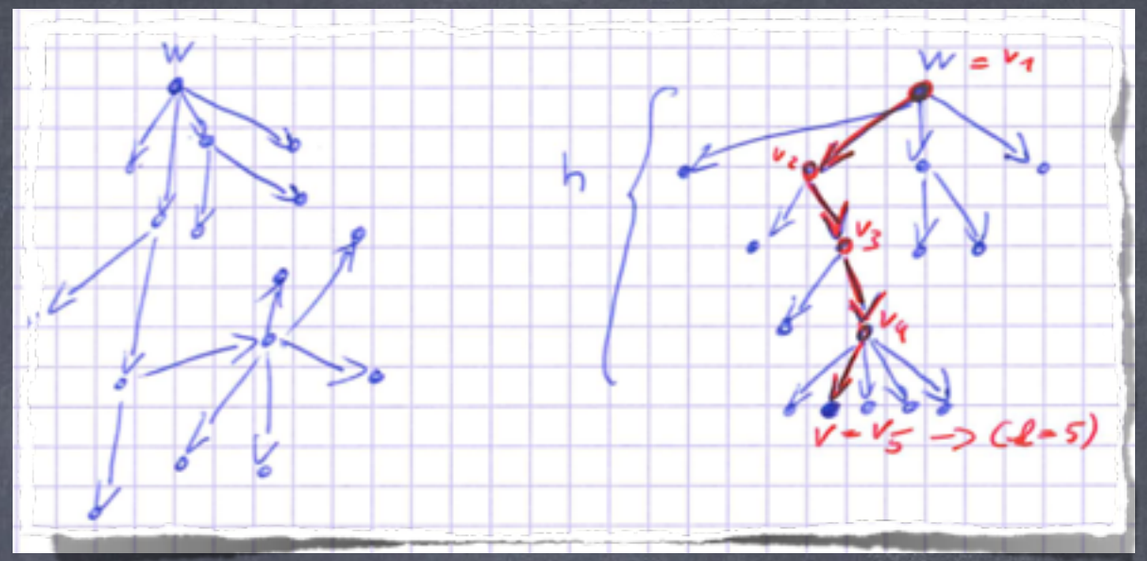
- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- mit $\forall v \in V \setminus \{w\} : \exists! (v', v) \in E$

  Jeder Knoten (außer) Wurzel ist Kind (eind. Vater)
- $\forall v \in V \setminus \{w\} : \exists$ Folge von Kanten $(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$ mit $v_1 = w, v_k = v$
- Folge eindeutig. Warum?
- Knoten ohne Kinder: Blätter

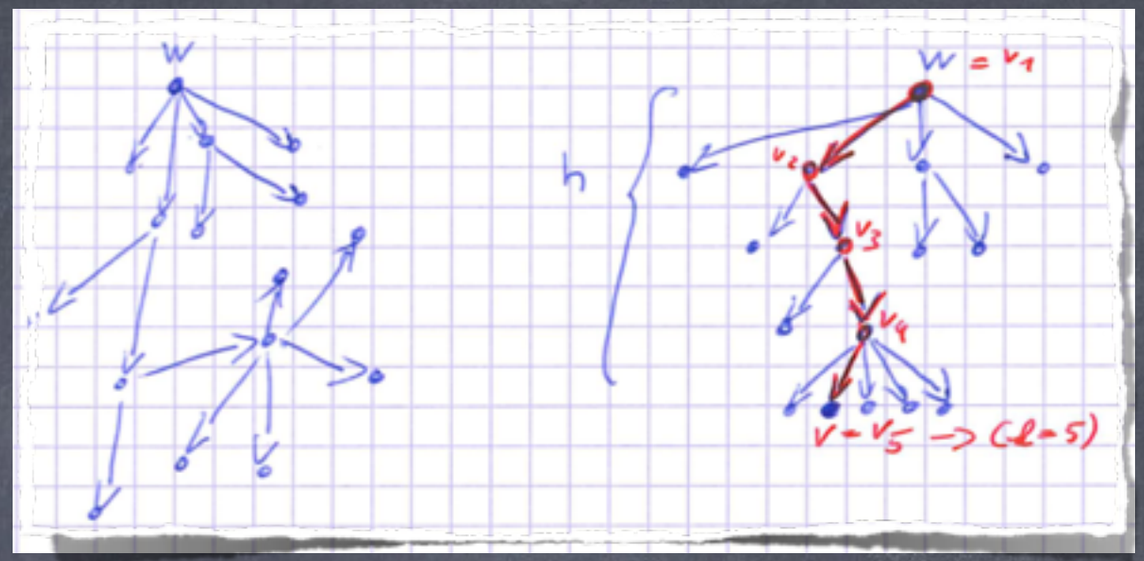
  Welchen Grad hat Blatt?



-Folge eindeutig.
Warum?

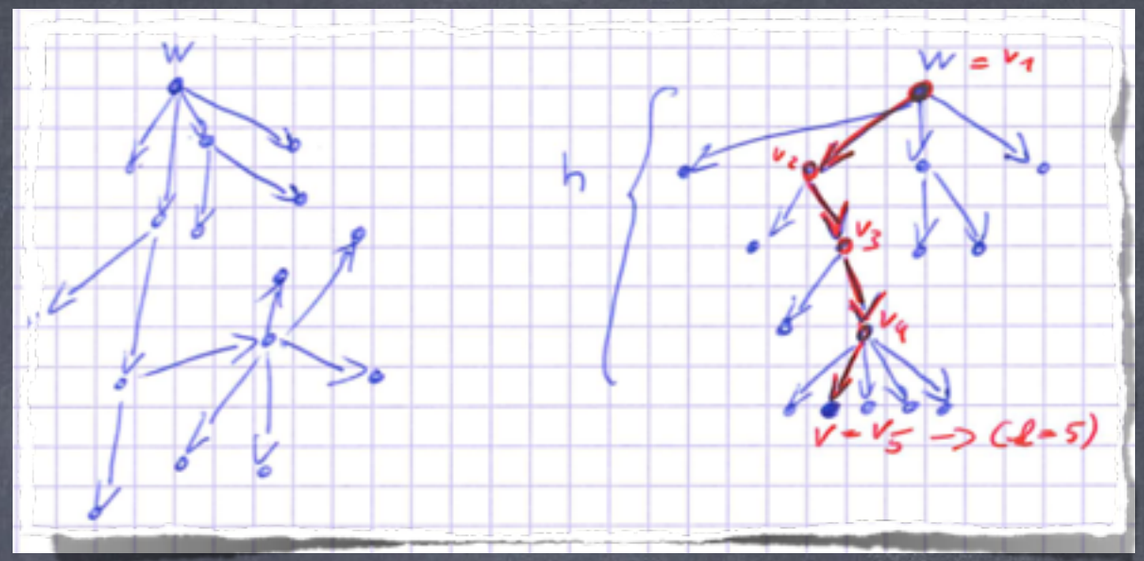


- Folge eindeutig. Warum?
- Beweis per Widerspruch

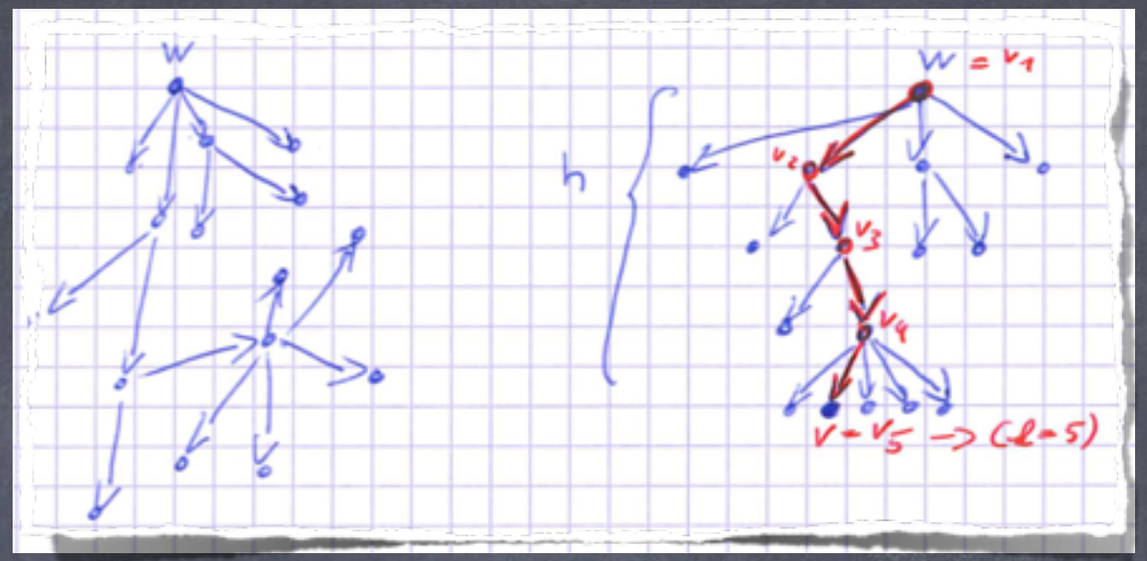


- Folge eindeutig. Warum?
- Beweis per Widerspruch

  Allg. Struktur:



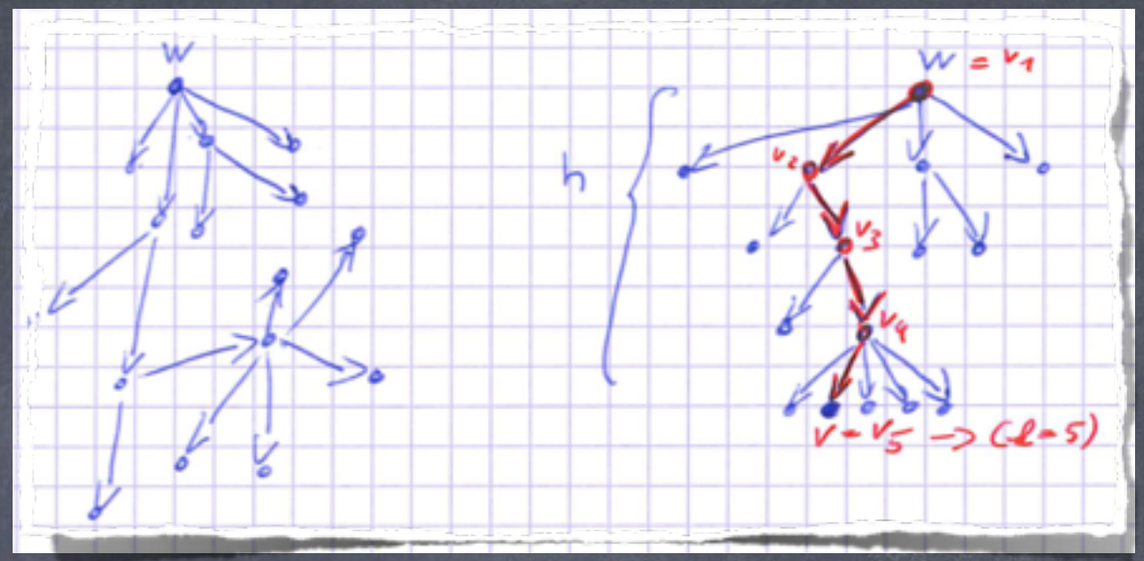
- Folge eindeutig. Warum?
- Beweis per Widerspruch



Allg. Struktur:

Nimm Gegenteil an und führe zum Widerspruch

- Folge eindeutig. Warum?
- Beweis per Widerspruch

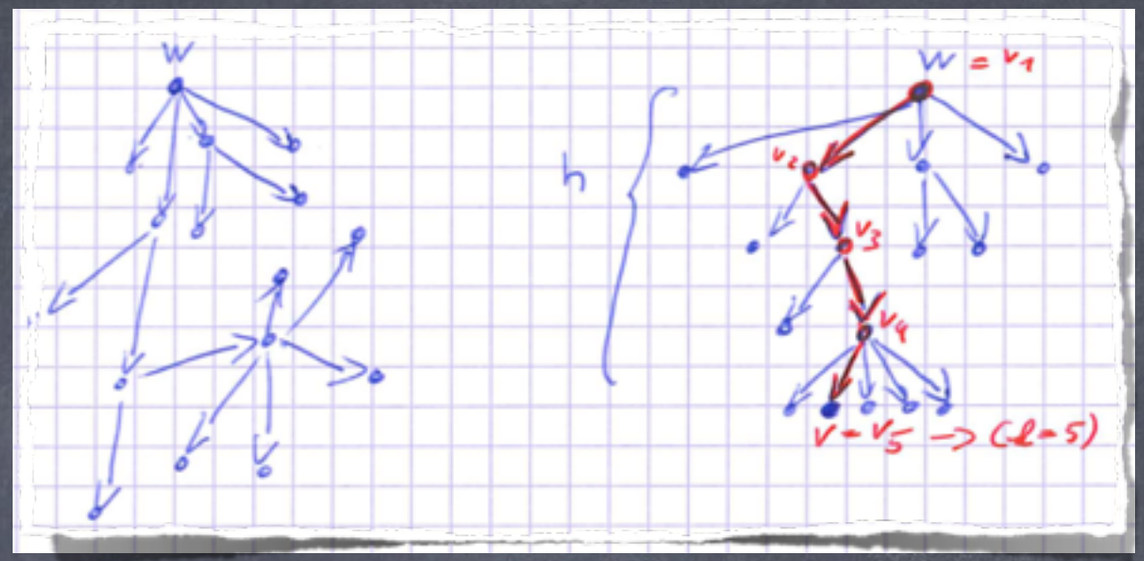


Allg. Struktur:

Nimm Gegenteil an und führe zum Widerspruch

=>Annahme falsch

- Folge eindeutig. Warum?
- Beweis per Widerspruch



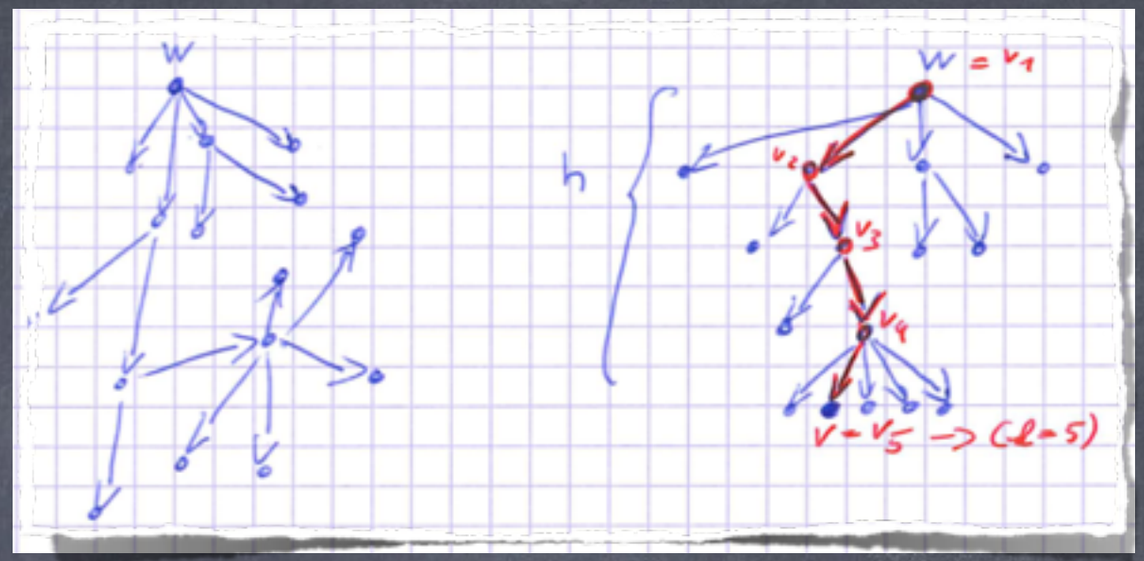
Allg. Struktur:

Nimm Gegenteil an und führe zum Widerspruch

=>Annahme falsch

Konkret:

- Folge eindeutig. Warum?
- Beweis per Widerspruch



Allg. Struktur:

Nimm Gegenteil an und führe zum Widerspruch

=>Annahme falsch

Konkret:

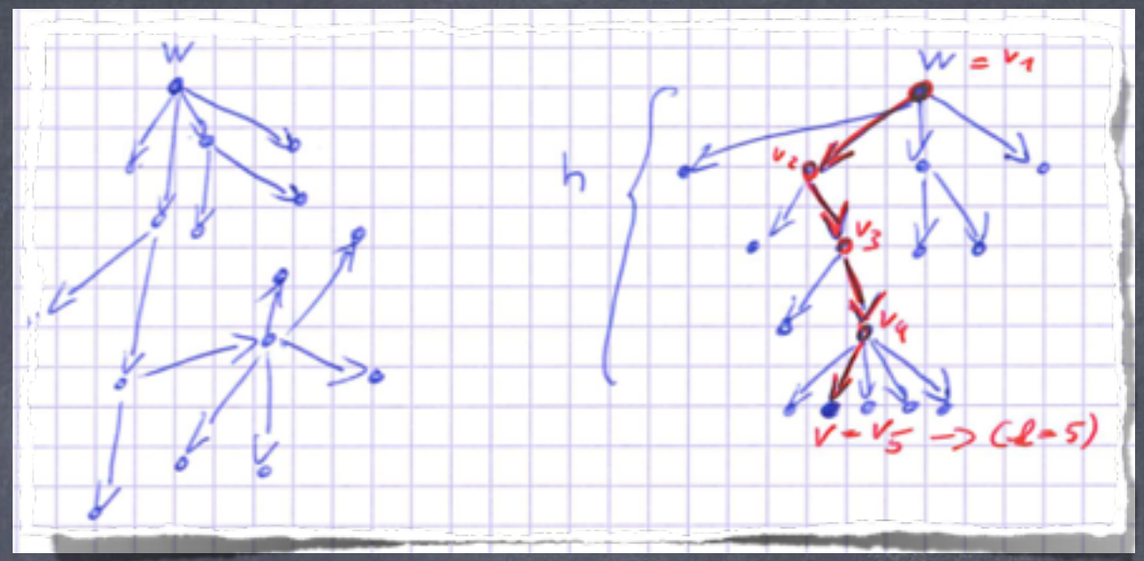
es. ex. mindestens zwei

$(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$

$(v'_1, v'_2), \ldots, (v'_{k'-1}, v'_{k'}) \in E$

(unterschiedlich!)

- Folge eindeutig. Warum?
- Beweis per Widerspruch



Allg. Struktur:

Nimm Gegenteil an und führe zum Widerspruch

=>Annahme falsch

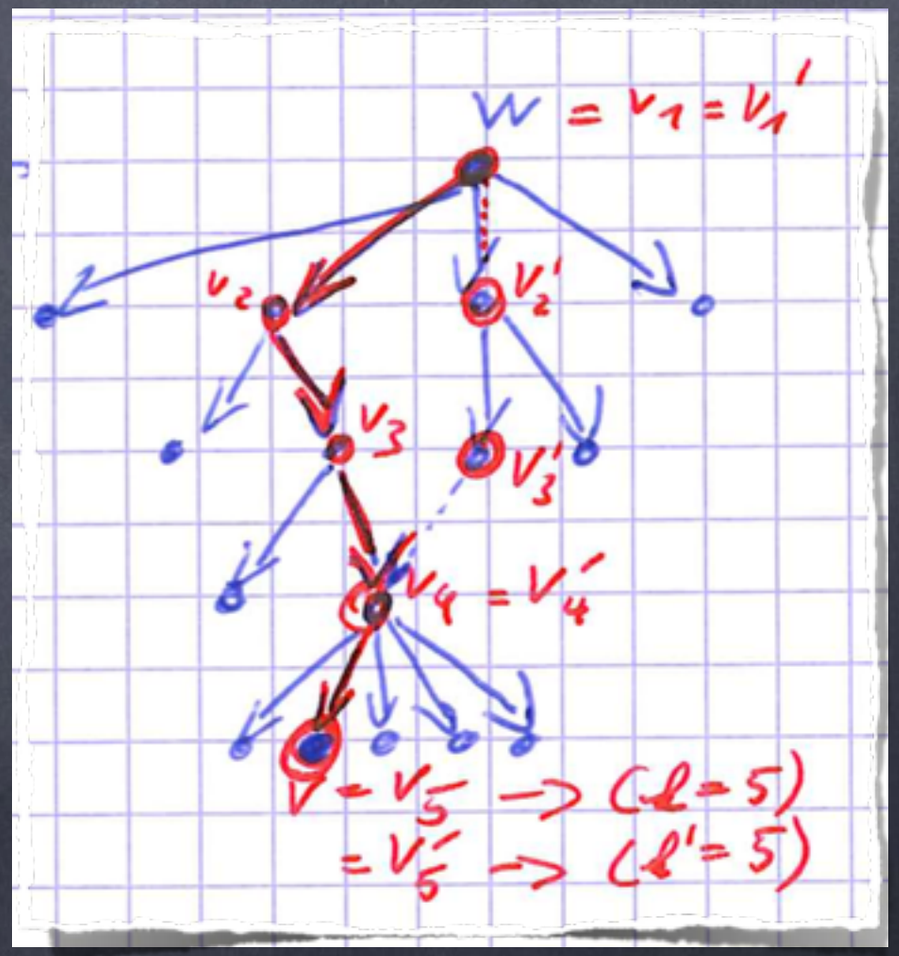
Konkret:

es. ex. mindestens zwei

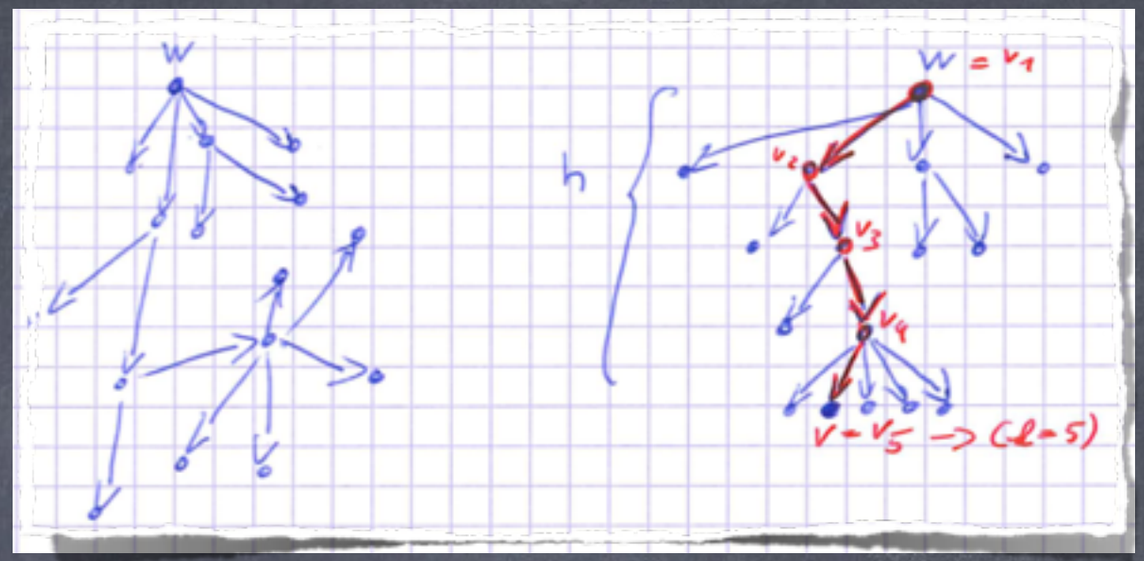
$(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$

$(v'_1, v'_2), \ldots, (v'_{k'-1}, v'_{k'}) \in E$

(unterschiedlich!)



- Folge eindeutig. Warum?
- Beweis per Widerspruch



Allg. Struktur:

Nimm Gegenteil an und führe zum Widerspruch

=>Annahme falsch

Konkret:

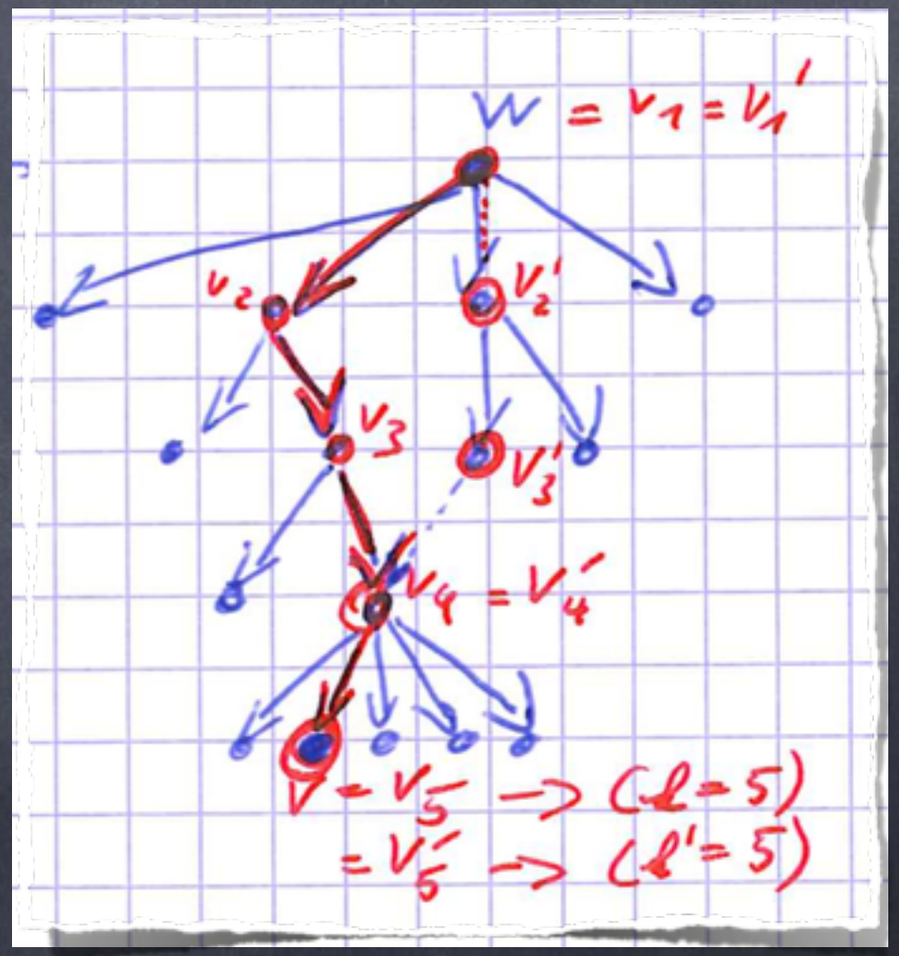
es. ex. mindestens zwei

$(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$

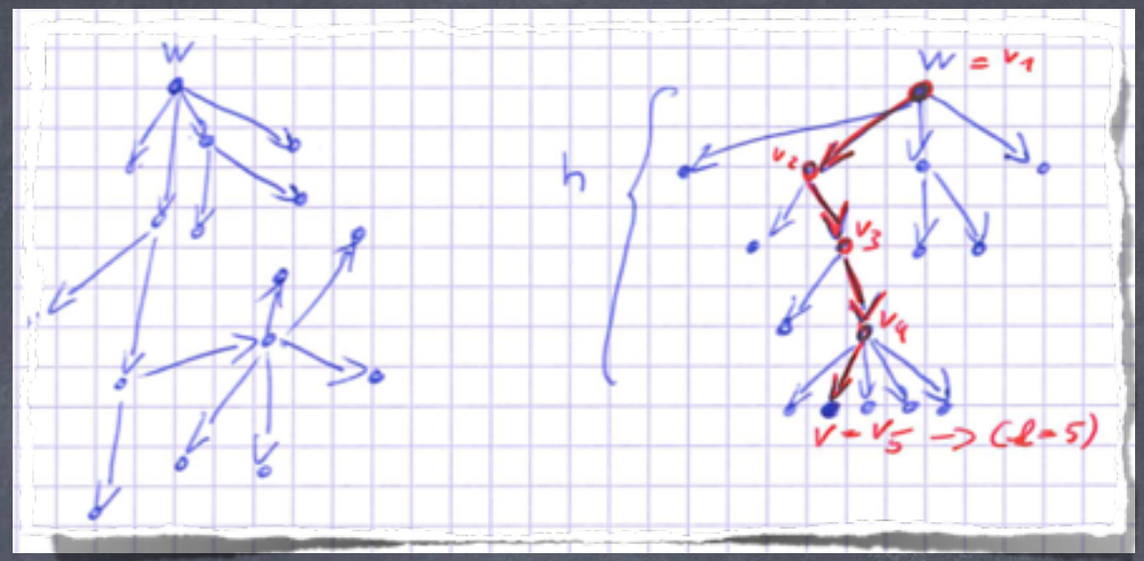
$(v'_1, v'_2), \ldots, (v'_{k'-1}, v'_{k'}) \in E$

(unterschiedlich!)

=> es ex. (mind) ein Knoten mit zwei Vätern



- Folge eindeutig. Warum?
- Beweis per Widerspruch



Allg. Struktur:

Nimm Gegenteil an und führe zum Widerspruch

=>Annahme falsch

Konkret:

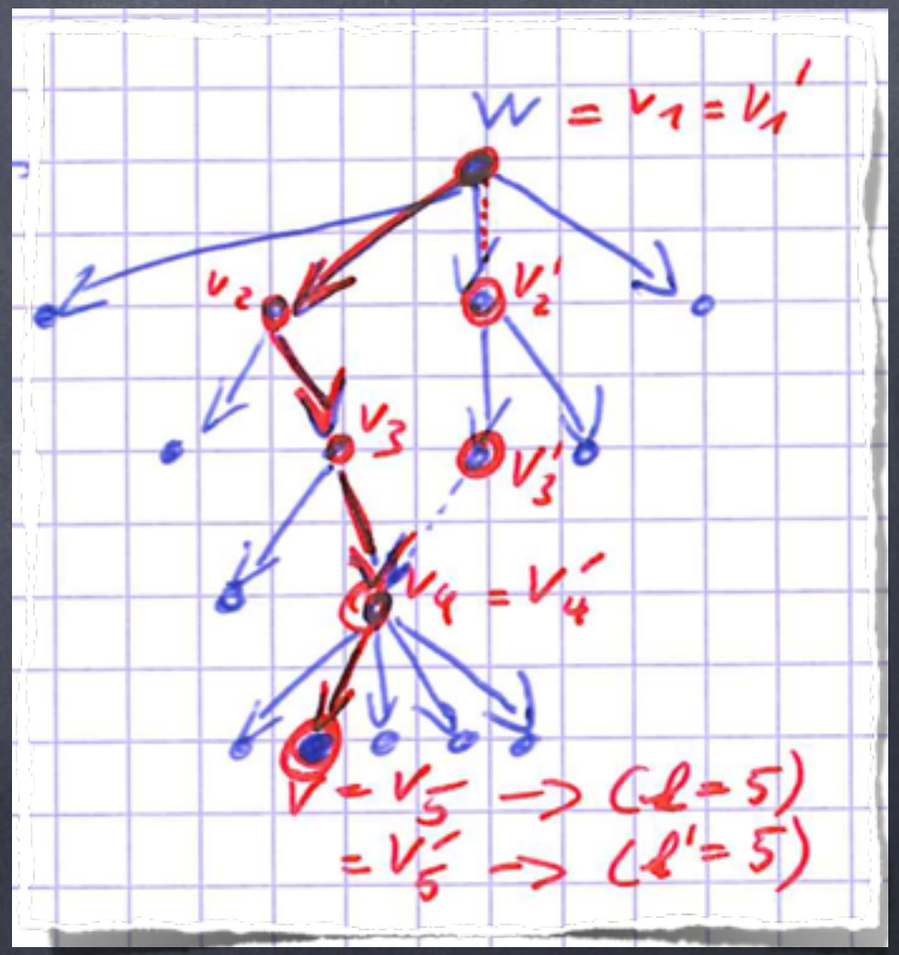
es. ex. mindestens zwei

$(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$

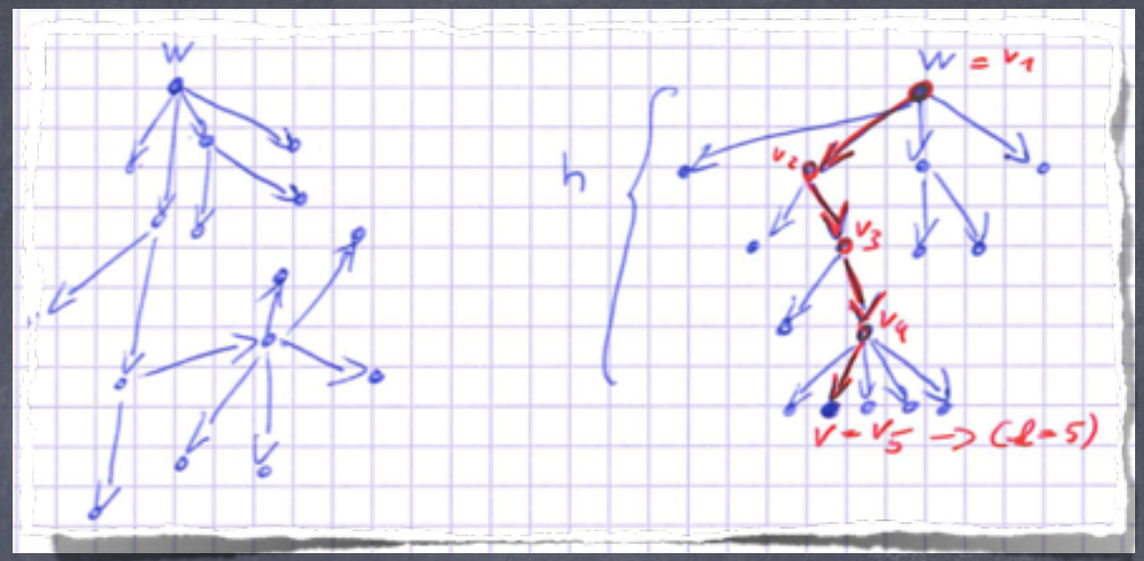
$(v'_1, v'_2), \ldots, (v'_{k'-1}, v'_{k'}) \in E$

(unterschiedlich!)

=> es ex. (mind) ein Knoten mit zwei Vätern



- Folge eindeutig. Warum?
- Beweis per Widerspruch



Allg. Struktur:

Nimm Gegenteil an und führe zum Widerspruch

=>Annahme falsch

Konkret:

es. ex. mindestens zwei

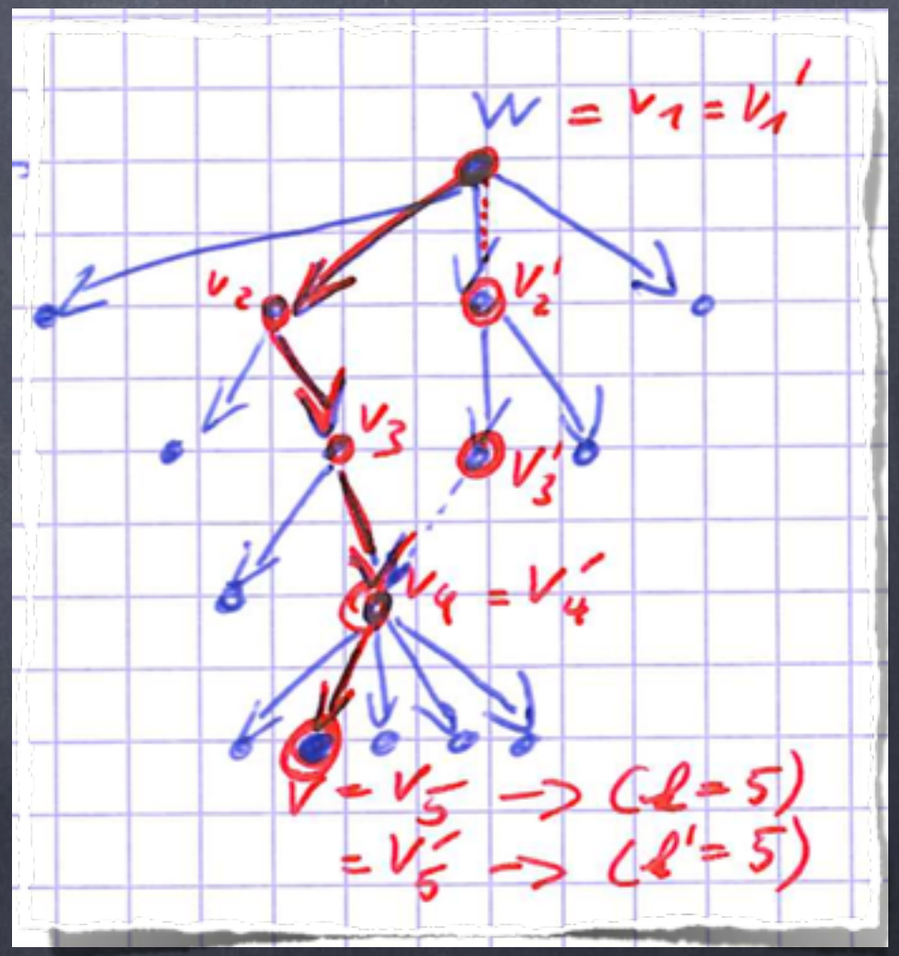
$(v_1, v_2), \ldots, (v_{k-1}, v_k) \in E$

$(v'_1, v'_2), \ldots, (v'_{k'-1}, v'_{k'}) \in E$

(unterschiedlich!)

=> es ex. (mind) ein Knoten mit zwei Vätern

=>Pfad eindeutig



Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition
  - Rekursiv

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition
  - Rekursiv ... Funktion, die sich selbst aufruft.

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition
  - Rekursiv ... Funktion, die sich selbst aufruft.
  - Baum ist:

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition
  - Rekursiv
  - Baum ist:
    - Blatt oder

... Funktion, die sich selbst aufruft.

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition
  - Rekursiv ... Funktion, die sich selbst aufruft.
  - Baum ist:
    - Blatt oder
    - Vater mit Kindern, die jeweils Wurzel eines Baumes sind

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition
  - Rekursiv ... Funktion, die sich selbst aufruft.
  - Baum ist:
    - Blatt oder Anker
    - Vater mit Kindern, die jeweils Wurzel eines Baumes sind

Baum:

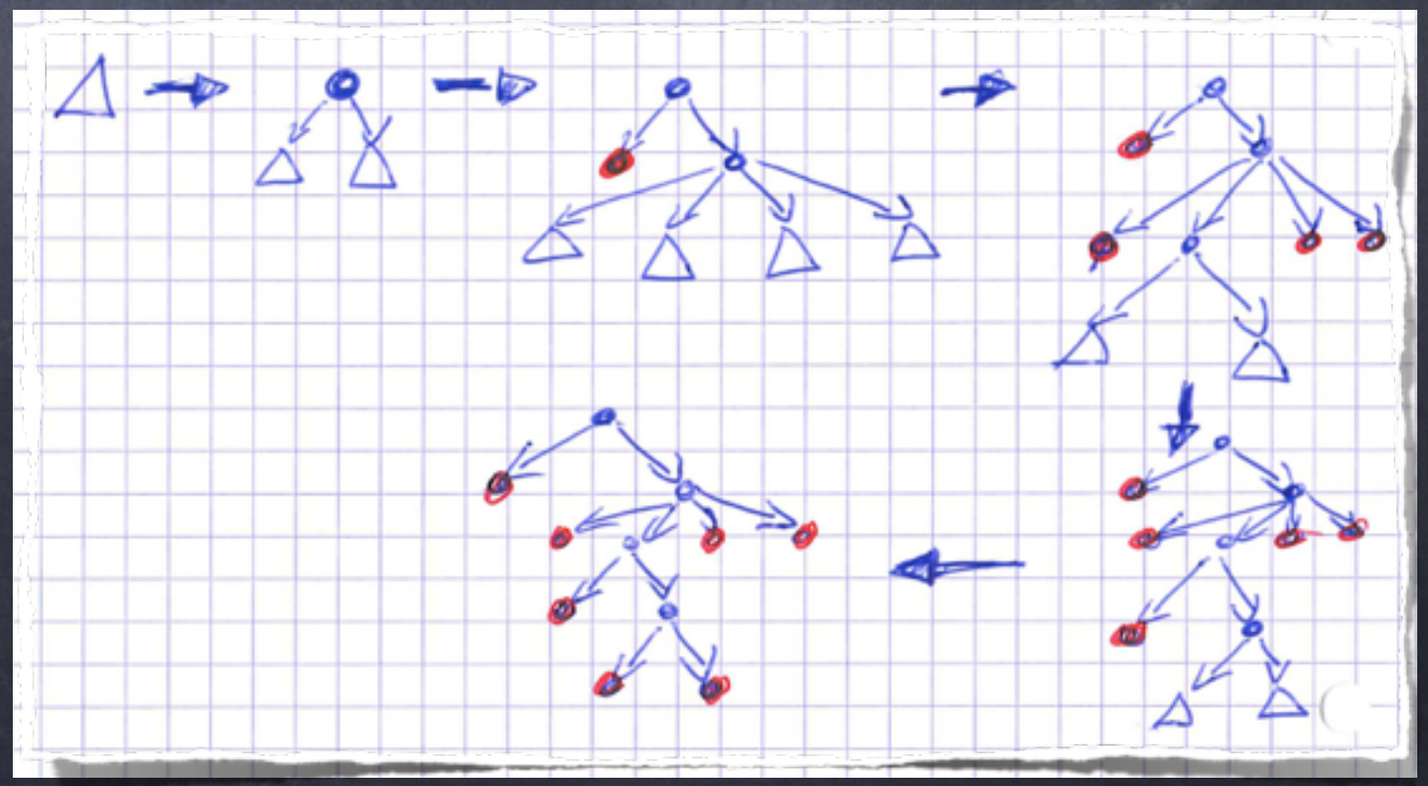
- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition
  - Rekursiv ... Funktion, die sich selbst aufruft.
  - Baum ist:
    - Blatt oder Anker
    - Vater mit Kindern, die jeweils Wurzel eines Baumes sind

rek. Aufruf

Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition
  - Rekursiv ... Funktion, die sich selbst aufruft.
  - Baum ist:
    - Blatt oder Anker
    - Vater mit Kindern, die jeweils Wurzel eines Baumes sind

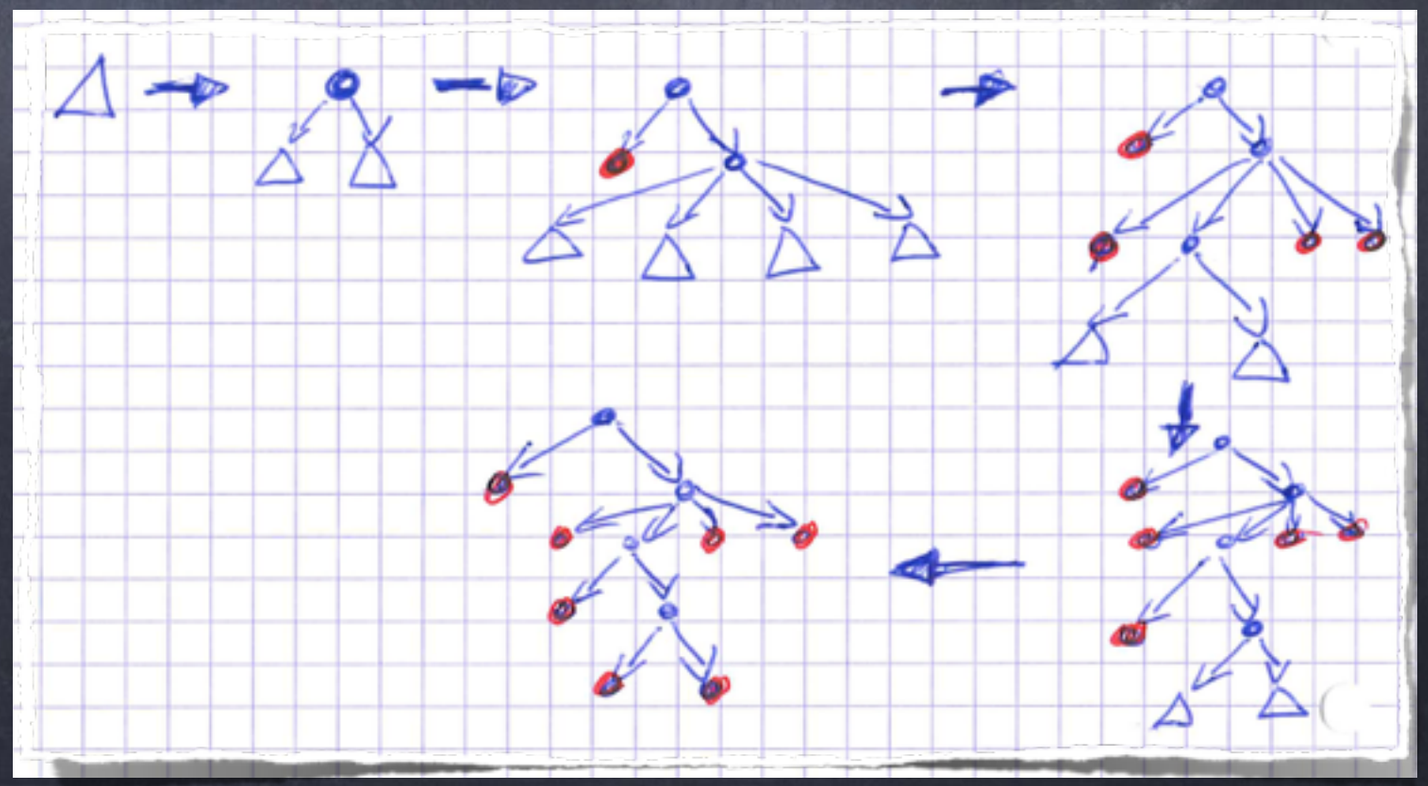
rek. Aufruf



Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition
  - Rekursiv ... Funktion, die sich selbst aufruft.
  - Baum ist:
    - Blatt oder Anker
    - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Rekursion nicht nur Funktionen

rek. Aufruf

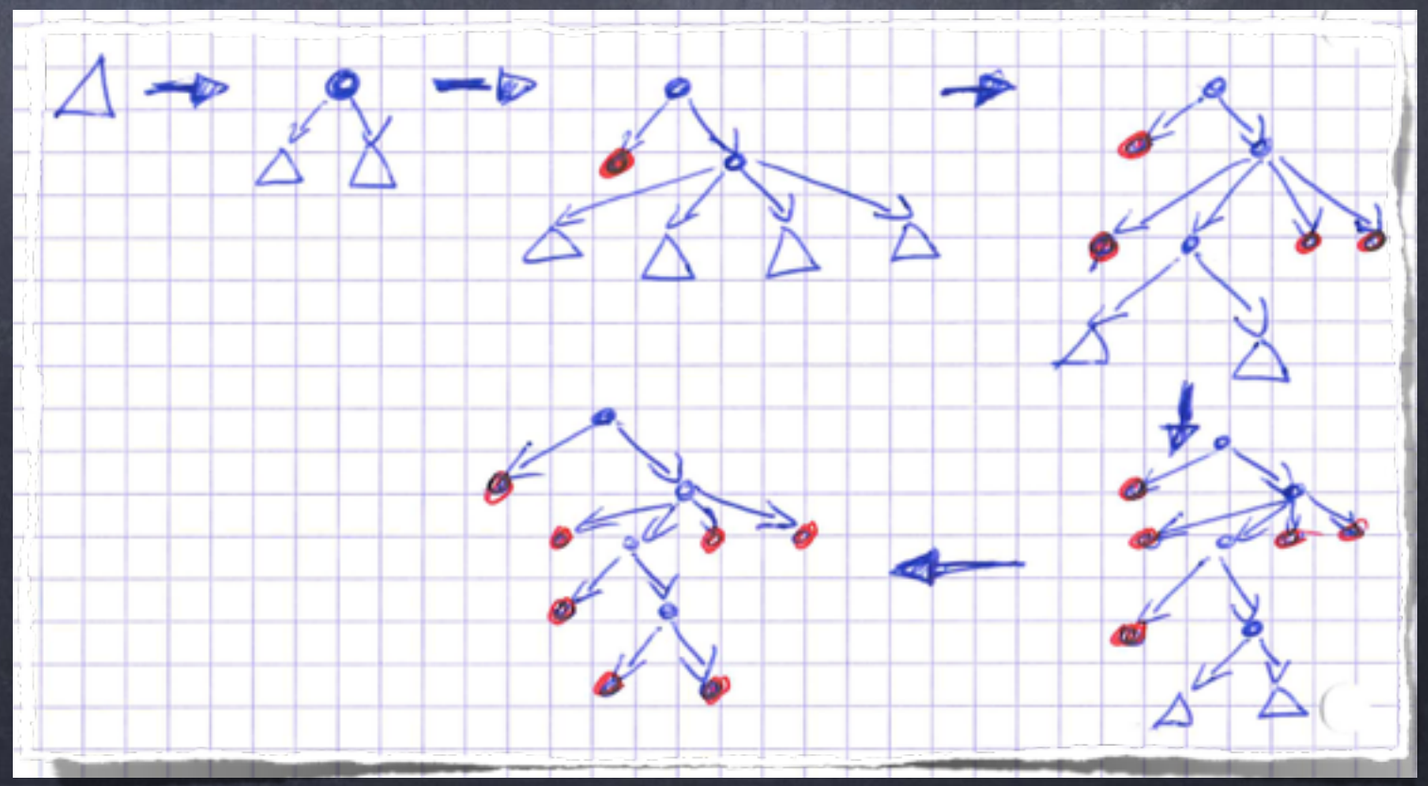


Baum:

- gerichteter Graph $G = (V, E)$
- ausgez. Wurzel $w \in V$
- Alternativ Definition
  - Rekursiv ... Funktion, die sich selbst aufruft.
  - Baum ist:
    - Blatt oder Anker
    - Vater mit Kindern, die jeweils Wurzel eines Baumes sind

rek. Aufruf

- Rekursion nicht nur Funktionen
- Rekursive Strukturen
  - ->Beweis Strukturen



- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind

- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig

- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion

- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes

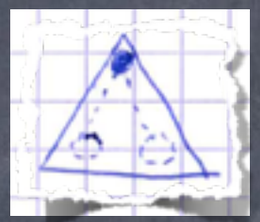
- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

      Was heißt das?

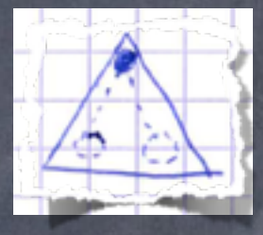
- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

      Was heißt das?



- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

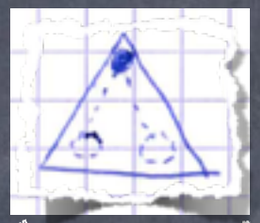
      Was heißt das?



    - IV: lautet...

- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

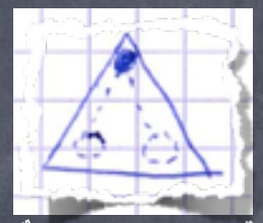
      Was heißt das?



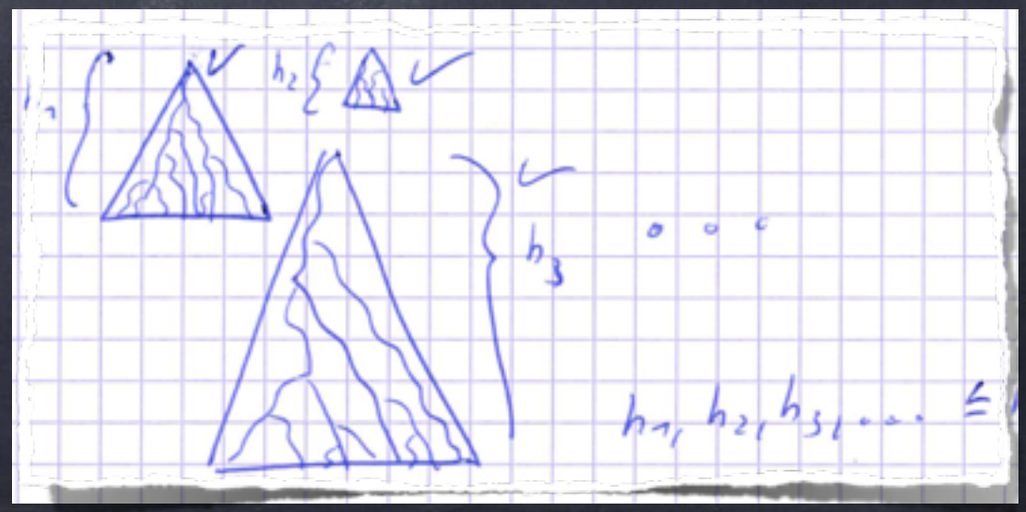
    - IV: lautet... In allen Bäumen der Höhe h gilt: Pfade von Wurzel zu Blatt sind eindeutig

- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

      Was heißt das?

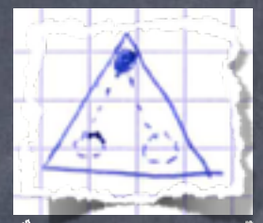


    - IV: lautet... In allen Bäumen der Höhe h gilt: Pfade von Wurzel zu Blatt sind eindeutig

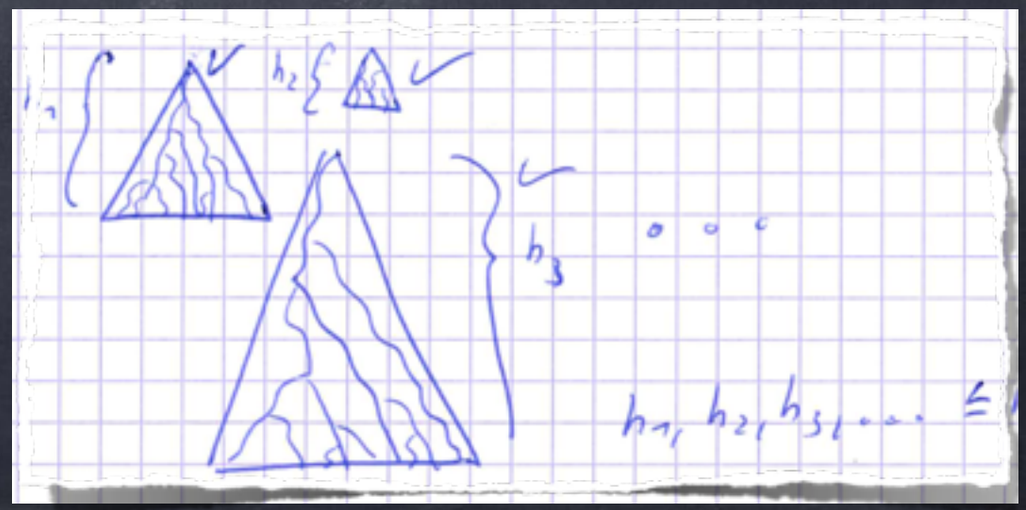


- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

      Was heißt das?

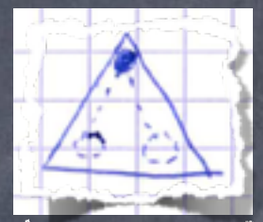


    - IV: lautet... In allen Bäumen der Höhe h gilt: Pfade von Wurzel zu Blatt sind eindeutig
    - IS:

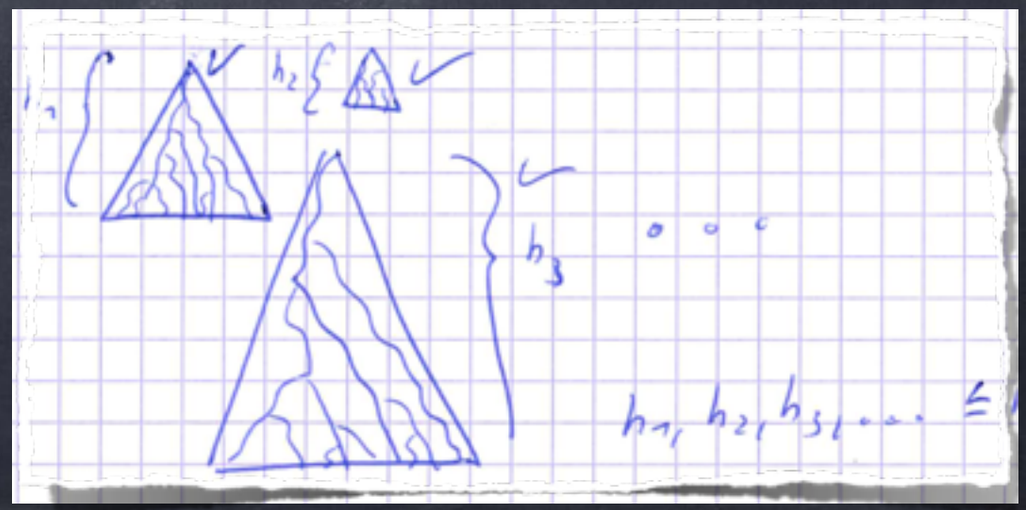


- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

      Was heißt das?

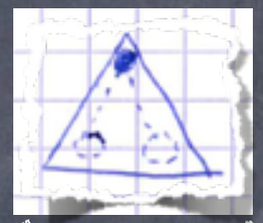


    - IV: lautet... In allen Bäumen der Höhe h gilt: Pfade von Wurzel zu Blatt sind eindeutig
    - IS: $1, ..., h \rightarrow h + 1$

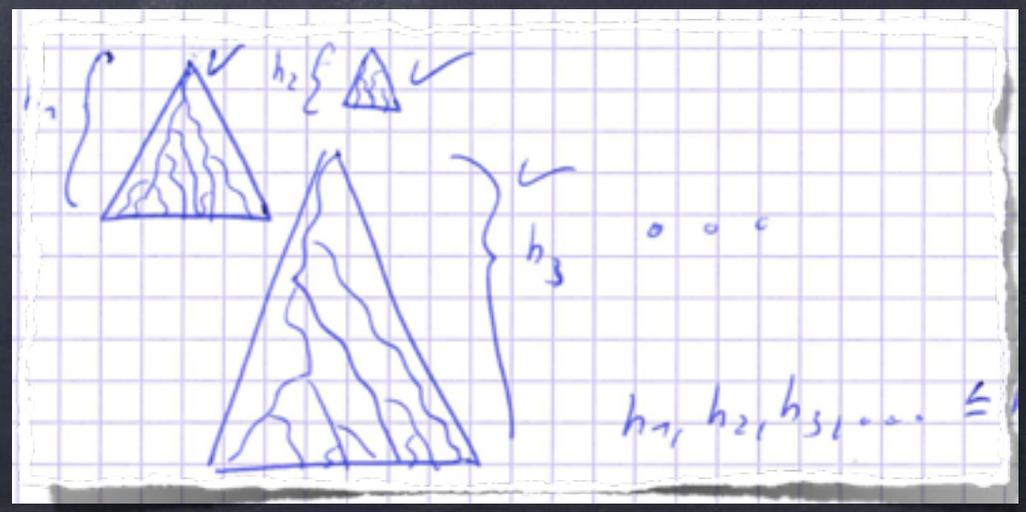


- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

      Was heißt das?

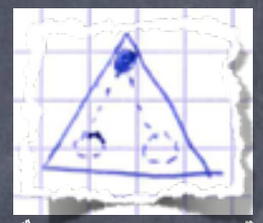


    - IV: lautet... In allen Bäumen der Höhe h gilt: Pfade von Wurzel zu Blatt sind eindeutig
    - IS: $1, ..., h \rightarrow h + 1$



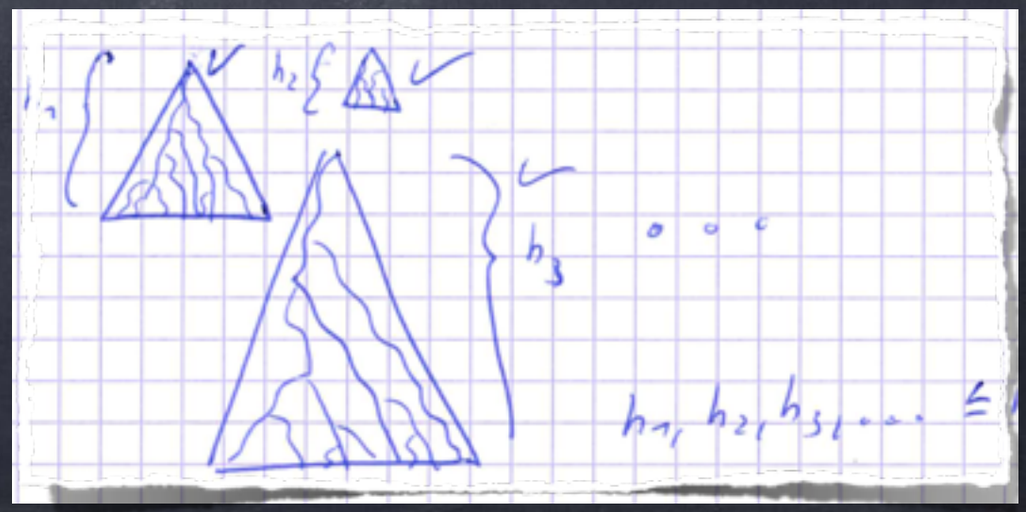
- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

      Was heißt das?



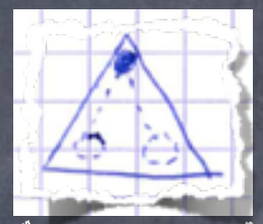
    - IV: lautet... In allen Bäumen der Höhe h gilt: Pfade von Wurzel zu Blatt sind eindeutig
    - IS: $1, ..., h \rightarrow h + 1$

      IV=>Weg von Kind der Wurzel zu Blatt: eindeutig

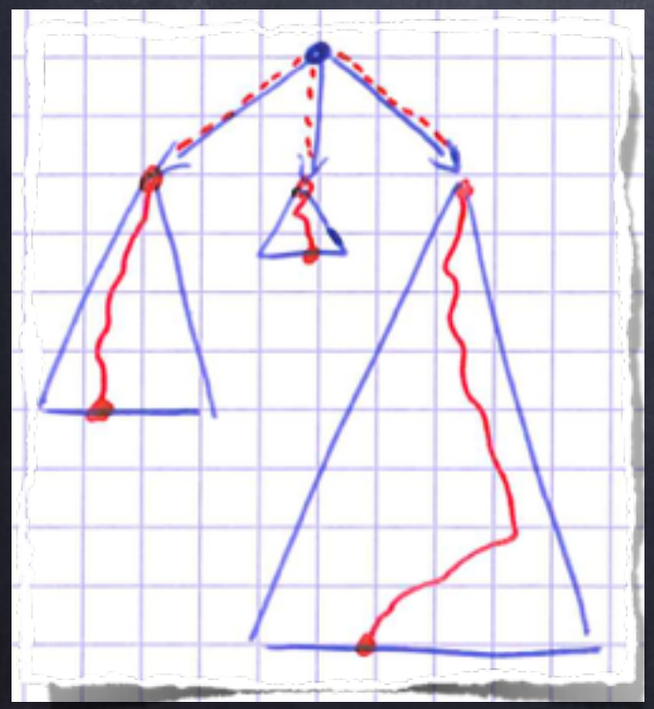


- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

      Was heißt das?

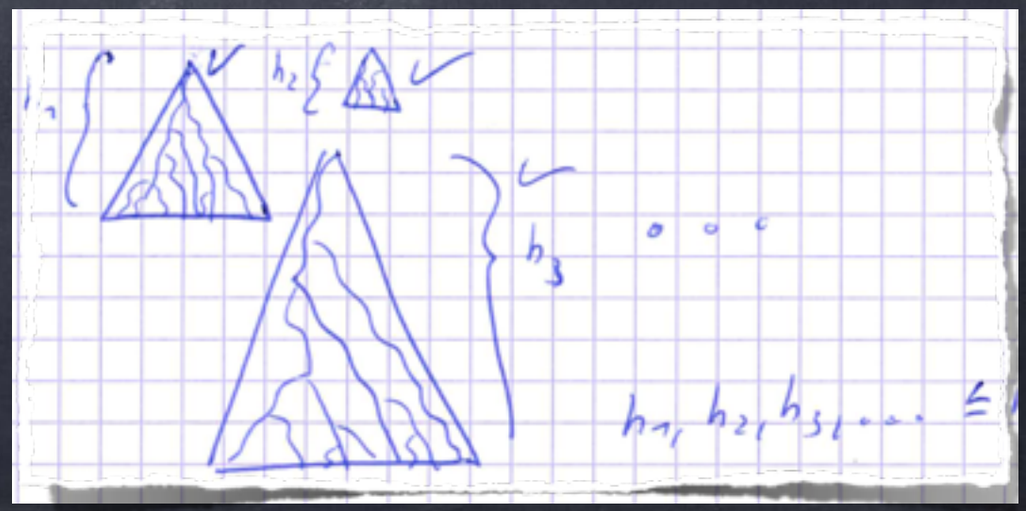


    - IV: lautet... In allen Bäumen der Höhe h gilt: Pfade von Wurzel zu Blatt sind eindeutig



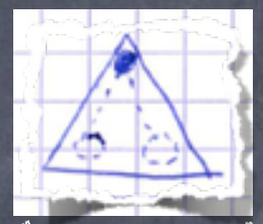
    - IS: $1, \ldots, h \rightarrow h + 1$

      IV=>Weg von Kind der Wurzel zu Blatt: eindeutig

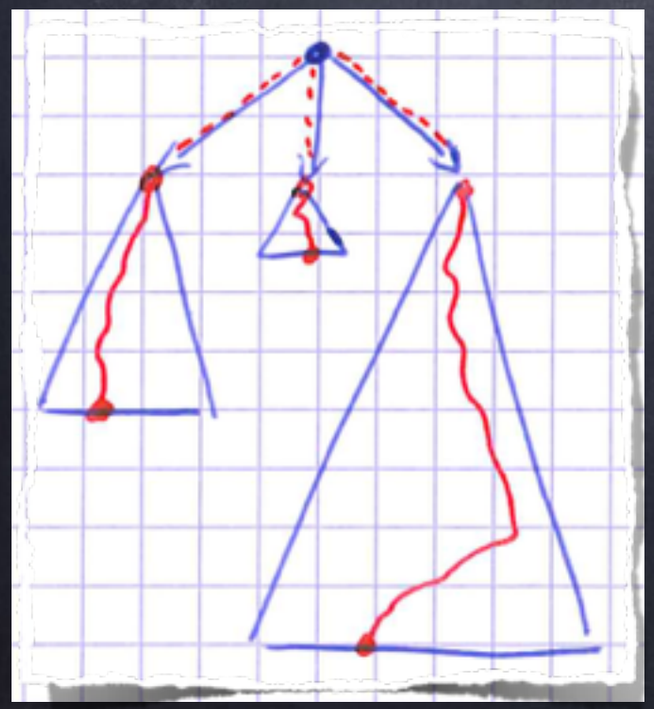


- Baum ist:
  - Blatt oder
  - Vater mit Kindern, die jeweils Wurzel eines Baumes sind
  - Weg von Wurzel zu Blatt eindeutig
  - Beweis per Induktion
    - Leiterstufen <-> Höhe des Baumes
    - IA: $h = 1$

      Was heißt das?

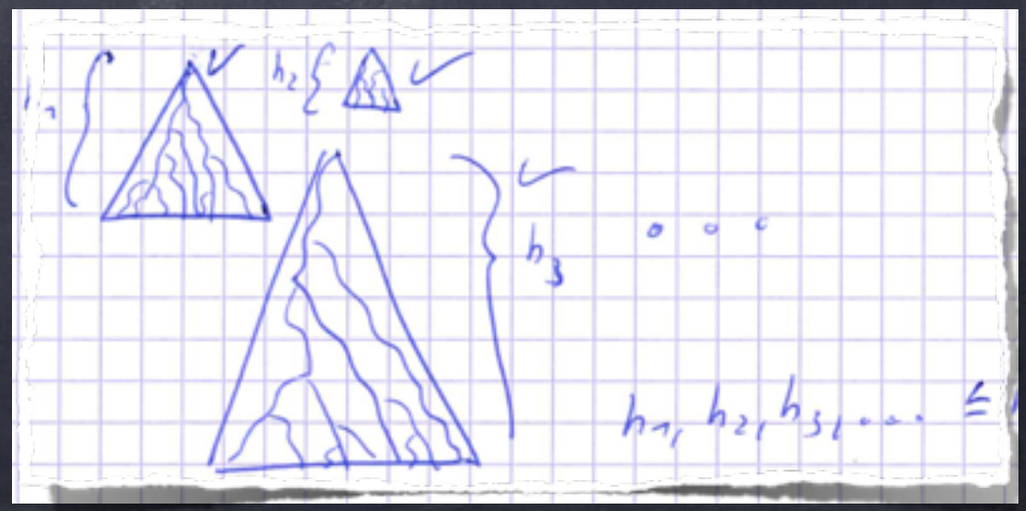


    - IV: lautet... In allen Bäumen der Höhe h gilt: Pfade von Wurzel zu Blatt sind eindeutig



    - IS: $1, \ldots, h \rightarrow h + 1$

      IV=>Weg von Kind der Wurzel zu Blatt: eindeutig
      - Kinder eind. mit Wurzel verb.



- Was ist Binärbaum?

-Was ist Binärbaum? zwei Kinder

-Was ist Binärbaum? zwei Kinder
-Was ist voller Baum?

-Was ist Binärbaum? zwei Kinder
-Was ist voller Baum? 0 oder 2 Kinder

- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)

- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)
    - Alle Blätter gleiche Tiefe und voll

- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)
    - Alle Blätter gleiche Tiefe und voll
    - Behauptung: Vollständiger bin. Baum hat 2^h Blätter

- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)
    - Alle Blätter gleiche Tiefe und voll
    - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
    - Beweis per Induktion

- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)
    - Alle Blätter gleiche Tiefe und voll
    - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
    - Beweis per Induktion ... über h

- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)
    - Alle Blätter gleiche Tiefe und voll
    - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
    - Beweis per Induktion ... über h
      IA:

- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)
    - Alle Blätter gleiche Tiefe und voll
    - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
    - Beweis per Induktion ... über h
      IA: $h = 1$

- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)

  - Alle Blätter gleiche Tiefe und voll
  - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
  - Beweis per Induktion ... über h
    IA: $h = 1$ => 2^1 Blätter

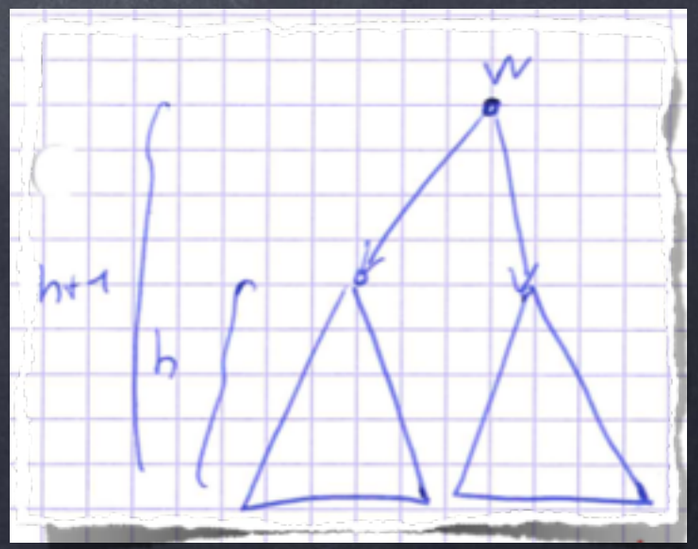
- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)

  - Alle Blätter gleiche Tiefe und voll
  - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
  - Beweis per Induktion ... über h

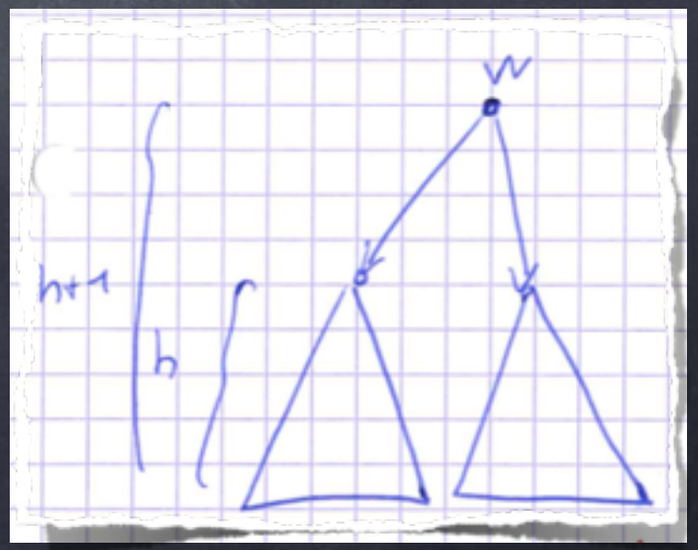
    IA: $h = 1$ => 2^1 Blätter

    IV: Aussage gelte für alle Bäume der Höhe h

- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)
    - Alle Blätter gleiche Tiefe und voll
    - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
    - Beweis per Induktion ... über h
      IA: $h = 1$ => 2^1 Blätter
      IV: Aussage gelte für alle Bäume der Höhe h



- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)
    - Alle Blätter gleiche Tiefe und voll
    - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
    - Beweis per Induktion ... über h
      IA: $h = 1$ => 2^1 Blätter
      IV: Aussage gelte für alle Bäume der Höhe h
      IS: $\leq h \rightarrow h + 1$



- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)

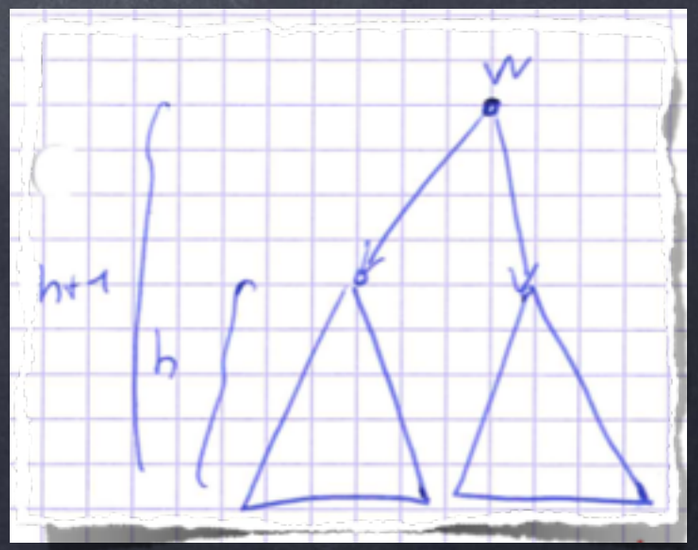
  - Alle Blätter gleiche Tiefe und voll
  - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
  - Beweis per Induktion ... über h

    IA: $h = 1$ => 2^1 Blätter

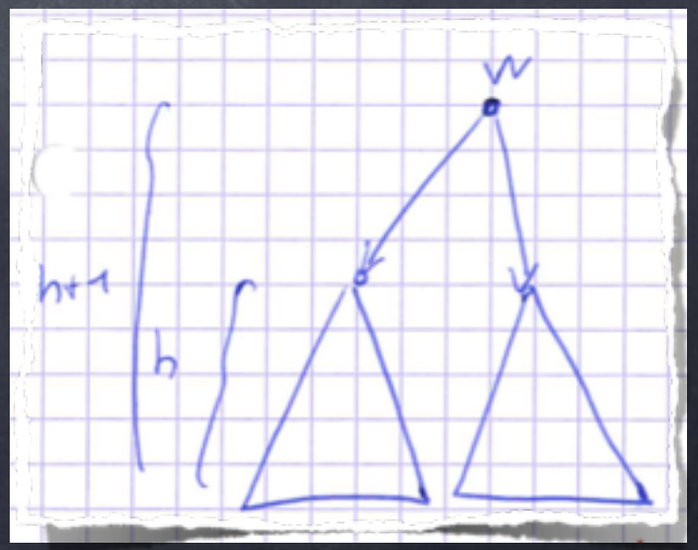
    IV: Aussage gelte für alle Bäume der Höhe h

    IS: $\leq h \rightarrow h + 1$

    Wurzel hat zwei Teilbäume der Höhe h



- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)
    - Alle Blätter gleiche Tiefe und voll
    - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
    - Beweis per Induktion ... über h
      IA: $h = 1$ => 2^1 Blätter
      IV: Aussage gelte für alle Bäume der Höhe h
      IS: $\leq h \rightarrow h + 1$
      Wurzel hat zwei Teilbäume der Höhe h
      =>Jeweils 2^h Blätter



- Was ist Binärbaum? zwei Kinder
- Was ist voller Baum? 0 oder 2 Kinder
- Vollständiger Baum:
  (nicht gleich voll)
    - Alle Blätter gleiche Tiefe und voll
    - Behauptung: Vollständiger bin. Baum hat 2^h Blätter
    - Beweis per Induktion ... über h

      IA: $h = 1$ => 2^1 Blätter

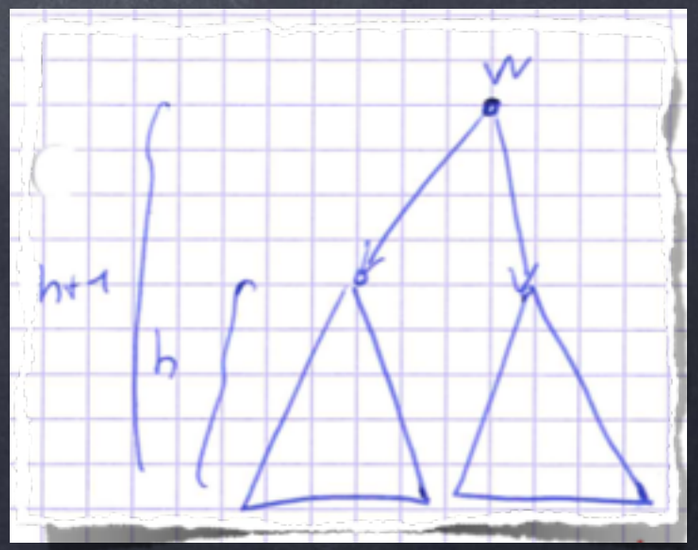
      IV: Aussage gelte für alle Bäume der Höhe h

      IS: $\leq h \rightarrow h + 1$

      Wurzel hat zwei Teilbäume der Höhe h

      =>Jeweils 2^h Blätter

      =>$2^h + 2^h = 2^{h+1}$ Blätter



- Was ist ein Suchbaum

- Was ist ein Suchbaum
  - Bin. Baum

- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel

- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel
  - Totale Ordnung: bsp. ganze Zahlen

- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel
  - Totale Ordnung: bsp. ganze Zahlen
  - linkes Kind $<$, rechtes Kind $\geq$

- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel
  - Totale Ordnung: bsp. ganze Zahlen
  - linkes Kind $<$, rechtes Kind $\geq$
  - bin. Suche $\mathcal{O}(n)$

- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel
  - Totale Ordnung: bsp. ganze Zahlen
  - linkes Kind $<$, rechtes Kind $\geq$
  - bin. Suche $\mathcal{O}(n)$
  - dyn. Verwaltung: Hinzufügen, Entfernen

- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel
  - Totale Ordnung: bsp. ganze Zahlen
  - linkes Kind $<$, rechtes Kind $\geq$
  - bin. Suche $\mathcal{O}(n)$
  - dyn. Verwaltung: Hinzufügen, Entfernen

-Was ist ein Suchbaum

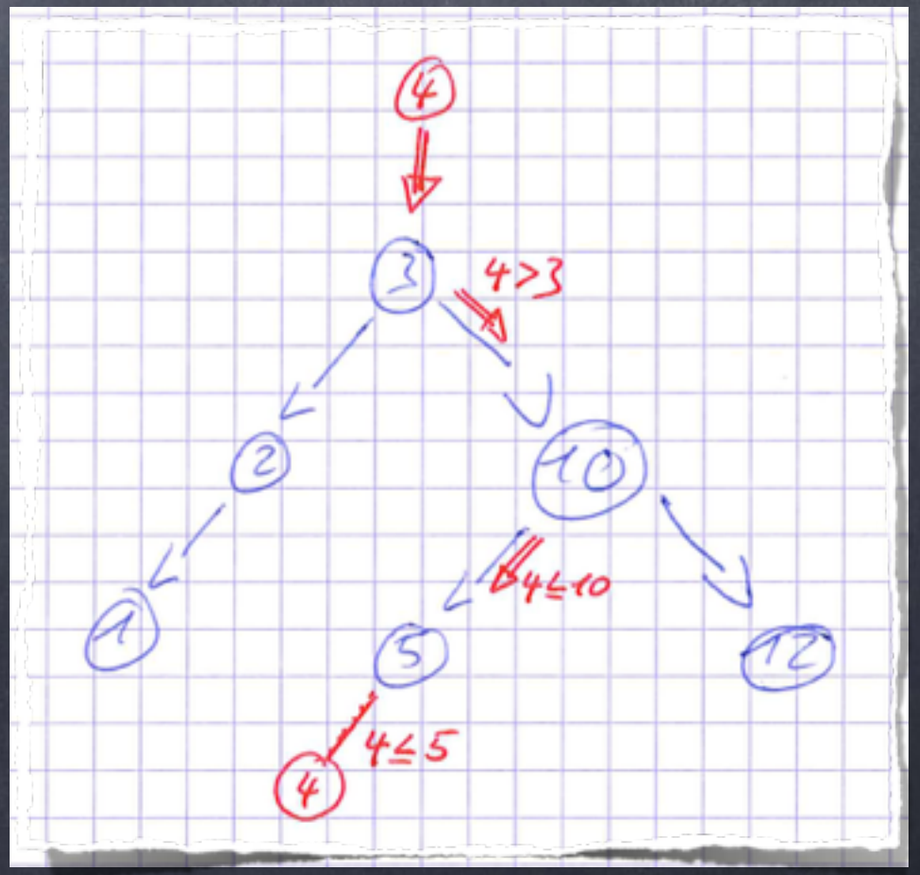
- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganze Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-INSERT(T, z)
y ← NIL
x ← wurzel[T]
while x ≠ NIL
      do y ← x
         if schlüssel[z] < schlüssel[x]
            then x ← links[x]
            else x ← rechts[x]
p[z] ← y
if y = NIL
   then wurzel[T] ← z
   else if schlüssel[z] < schlüssel[y]
           then links[y] ← z
           else rechts[y] ← z
```

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganze Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-INSERT(T, z)
y ← NIL
x ← wurzel[T]
while x ≠ NIL
      do y ← x
         if schlüssel[z] < schlüssel[x]
            then x ← links[x]
            else x ← rechts[x]
p[z] ← y
if y = NIL
   then wurzel[T] ← z
   else if schlüssel[z] < schlüssel[y]
            then links[y] ← z
            else rechts[y] ← z
```



- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel
  - Totale Ordnung: bsp. ganzel Zahlen
  - linkes Kind $<$, rechtes Kind $\geq$
  - bin. Suche $\mathcal{O}(n)$
  - dyn. Verwaltung: Hinzufügen, Entfernen

-Was ist ein Suchbaum

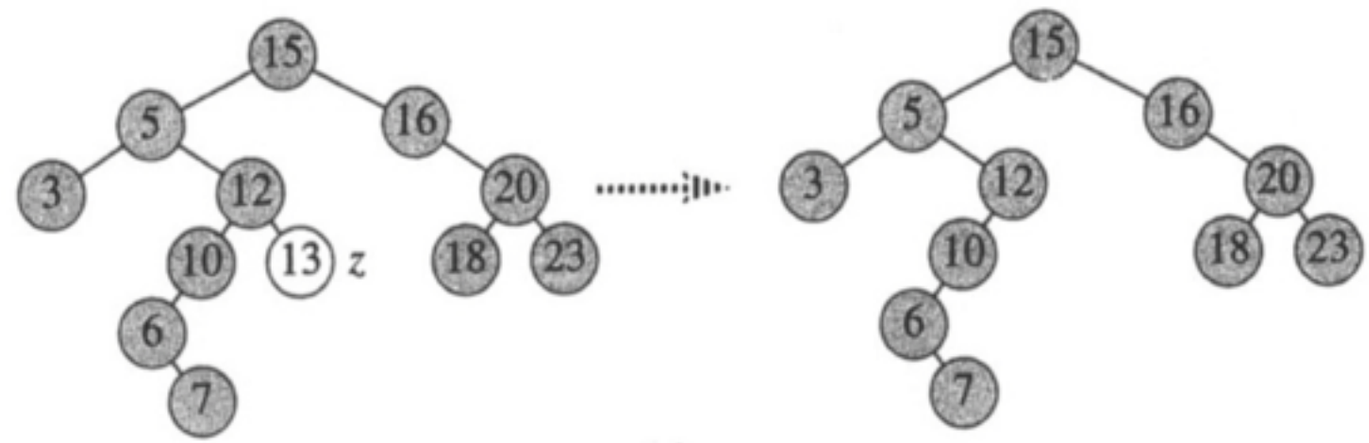
- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

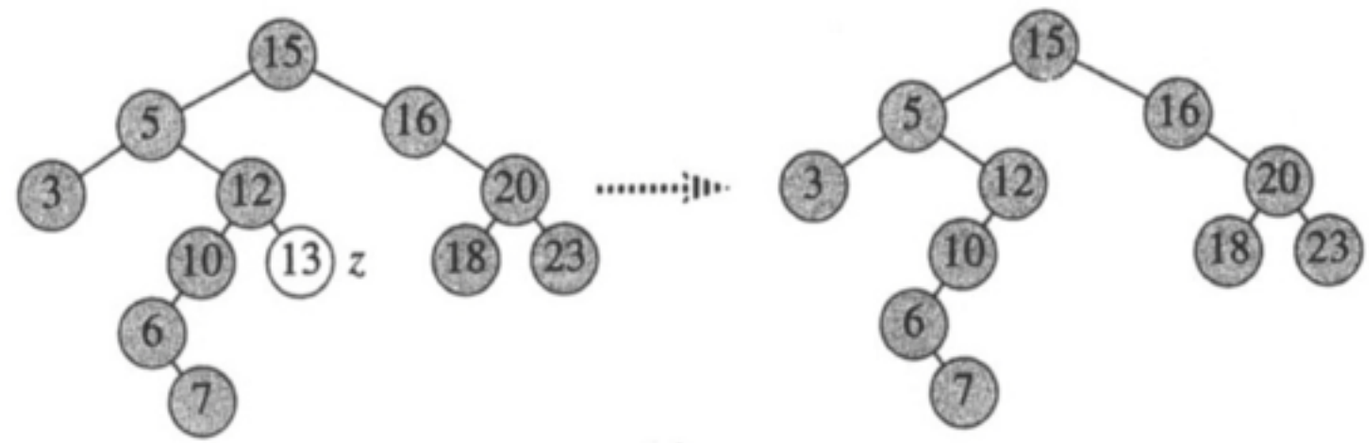


(a)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

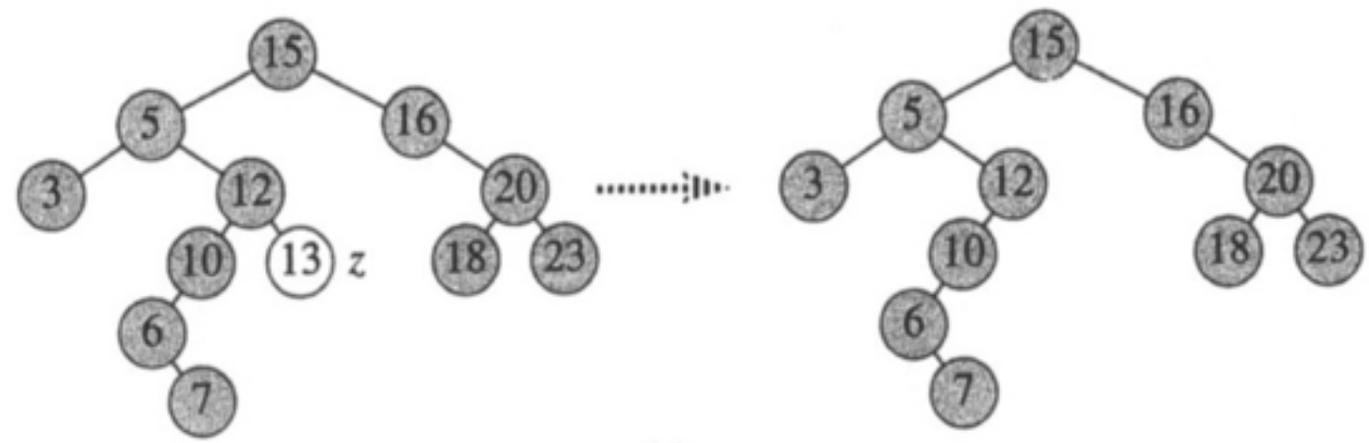


(a)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

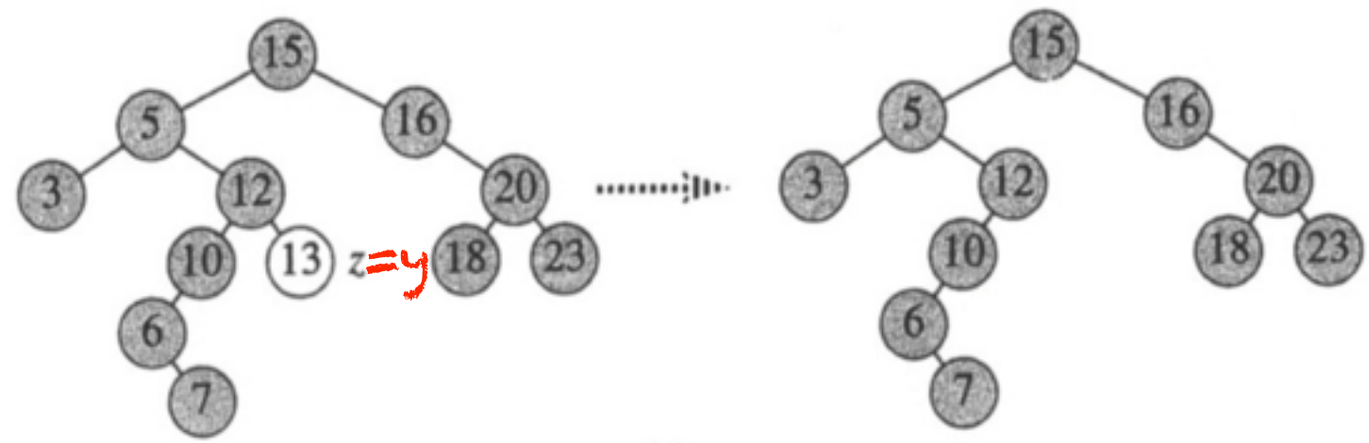


(a)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

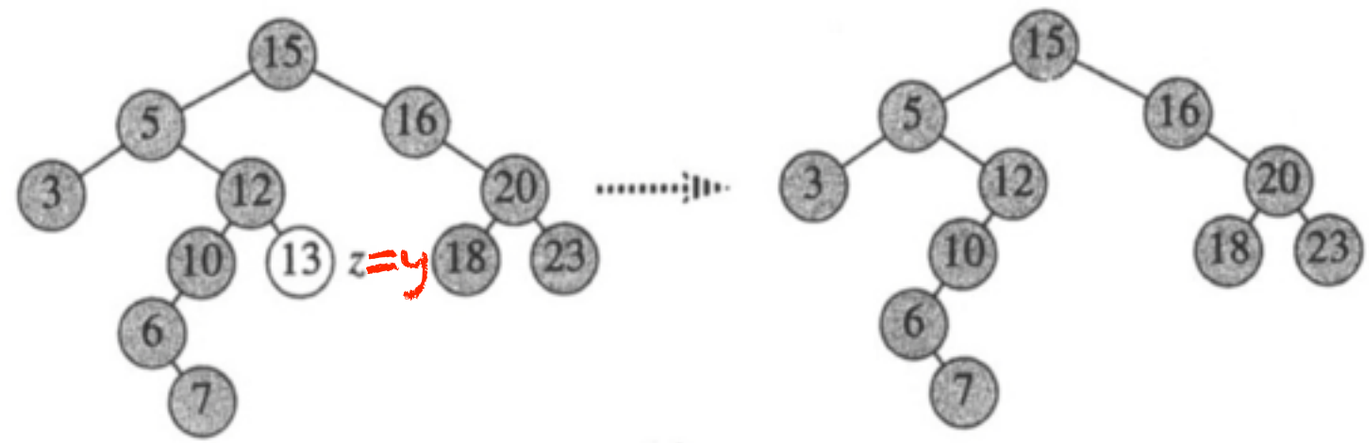


(a)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

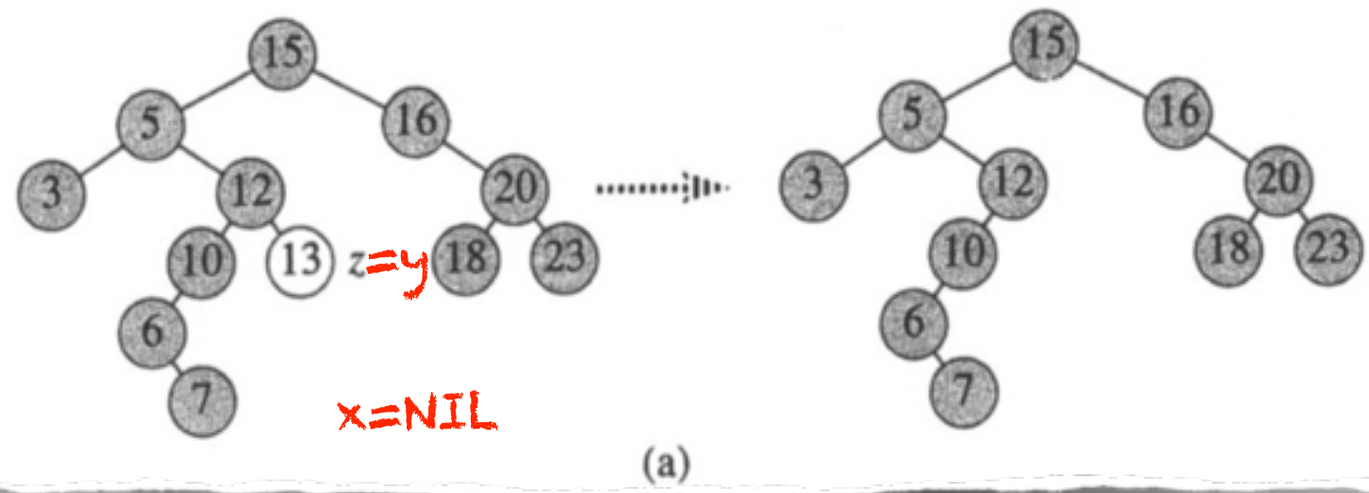


(a)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

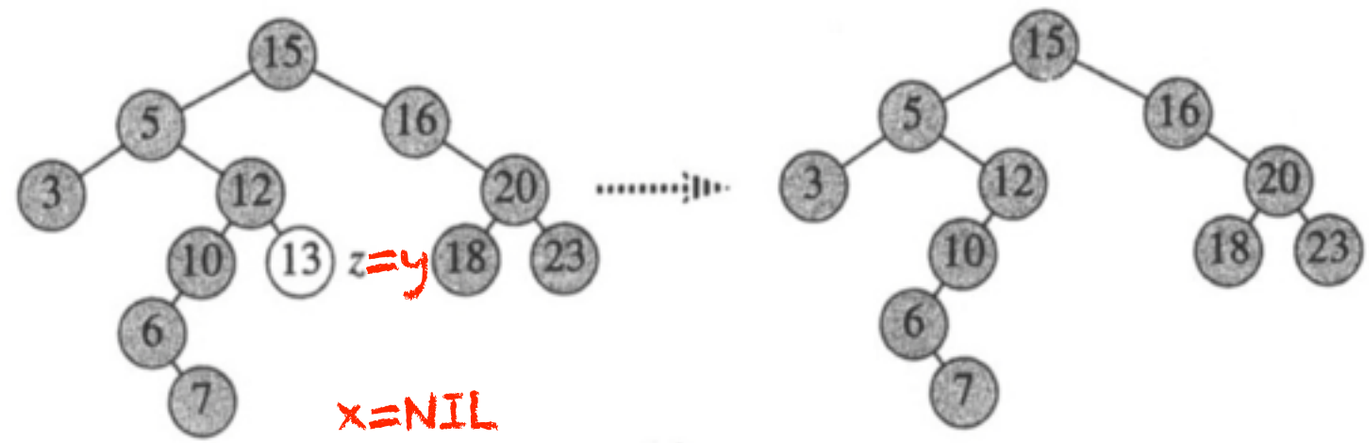


(a)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

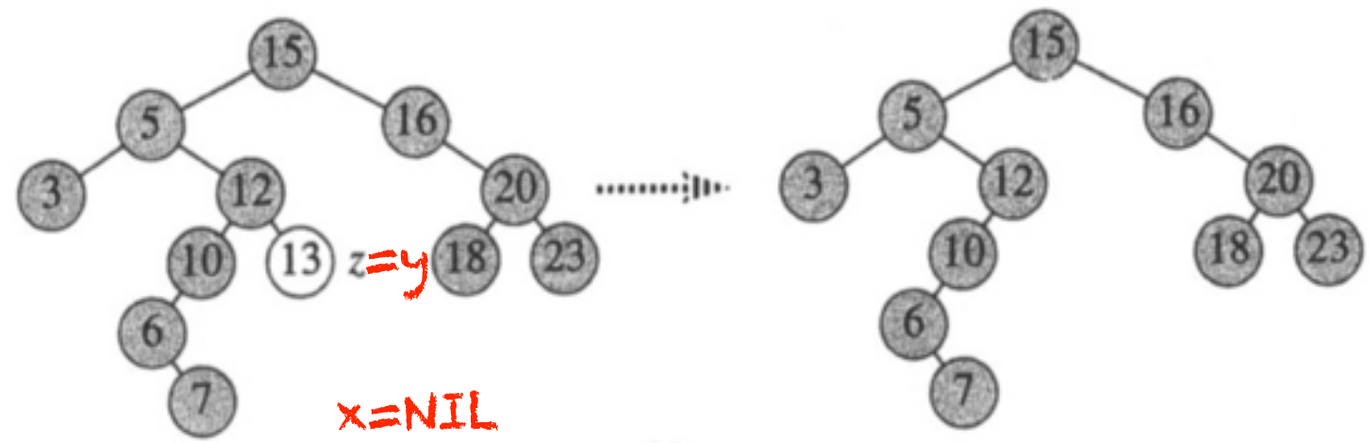


(a)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



(a)

-Was ist ein Suchbaum

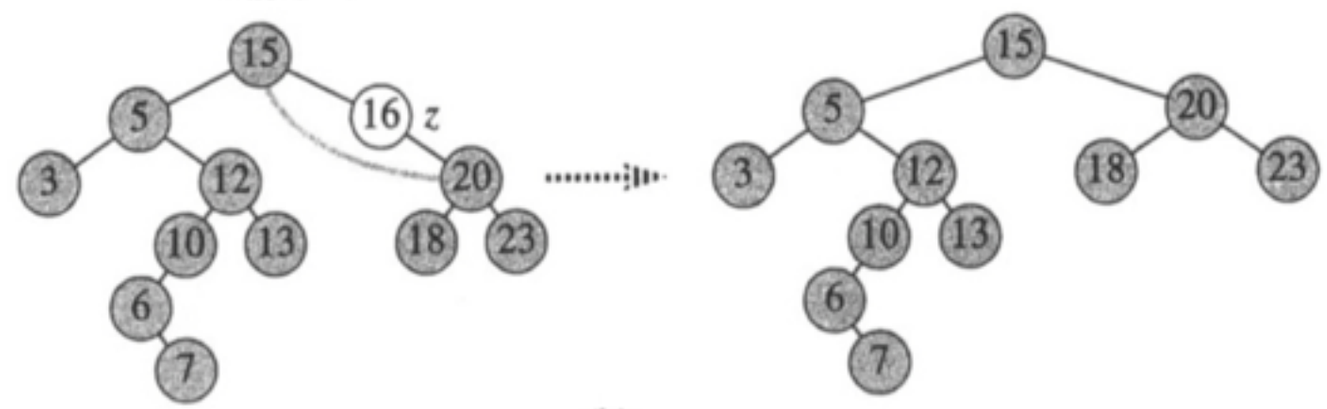
- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

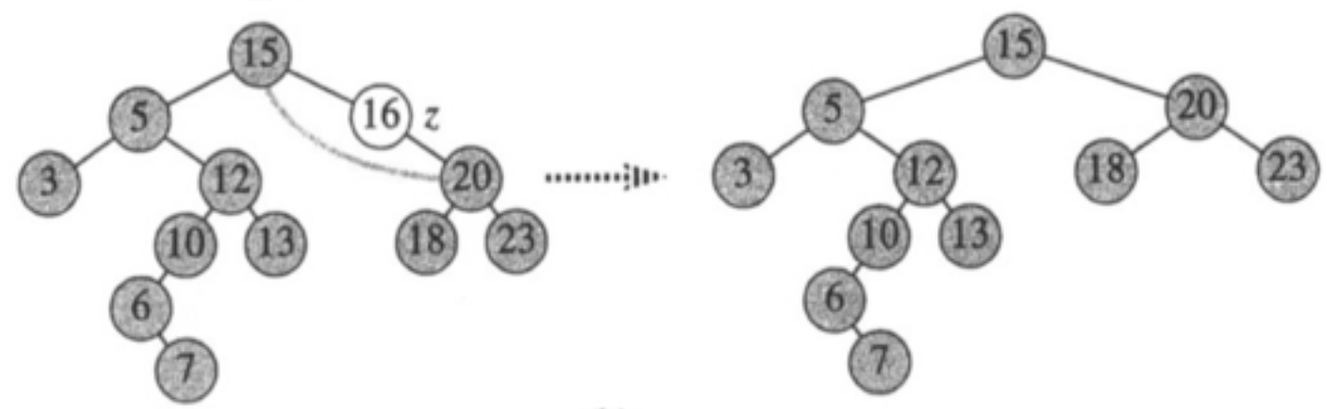


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

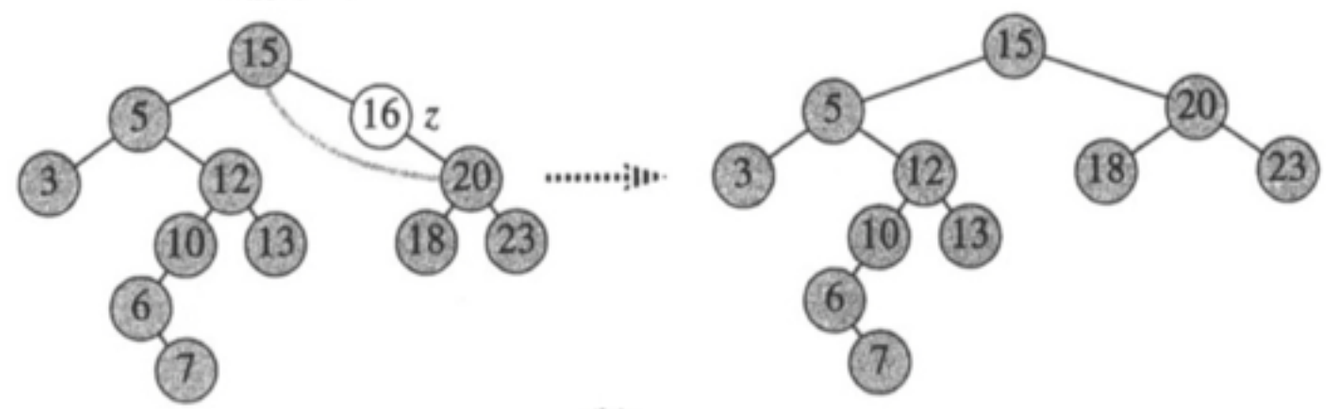


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

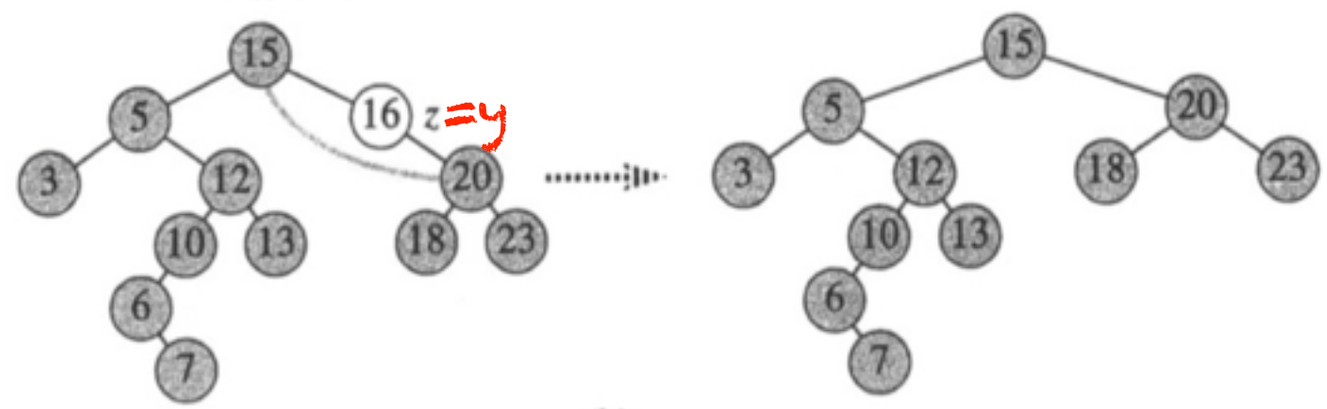


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

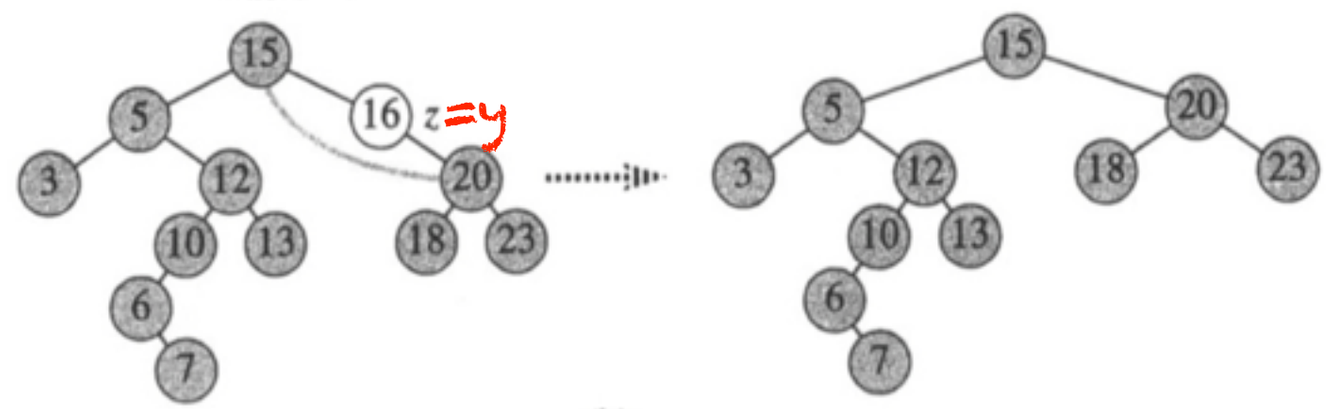


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

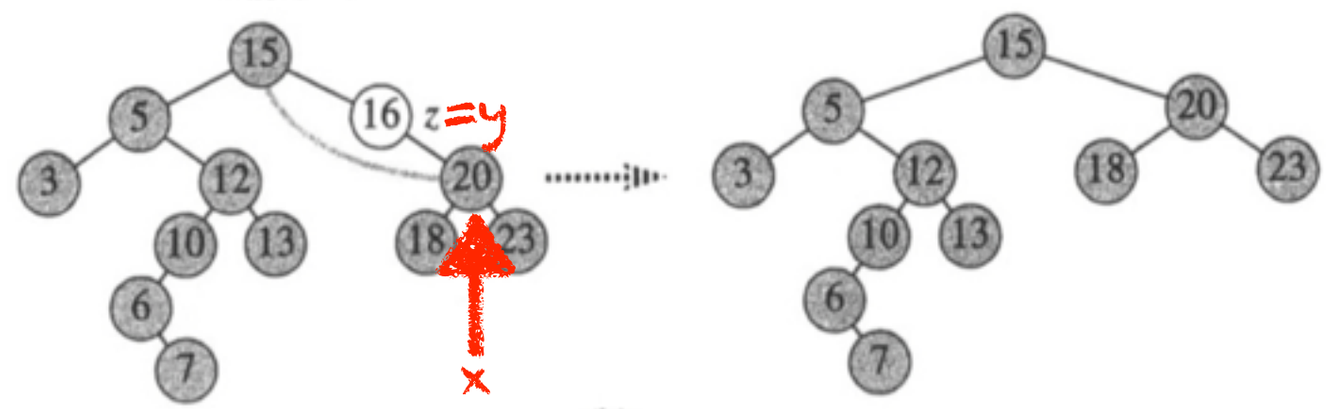


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

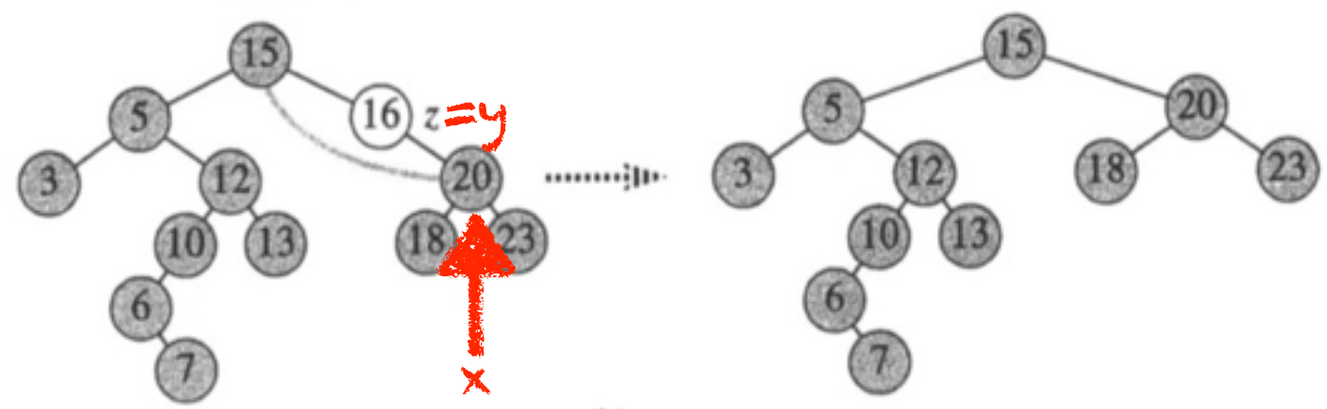


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

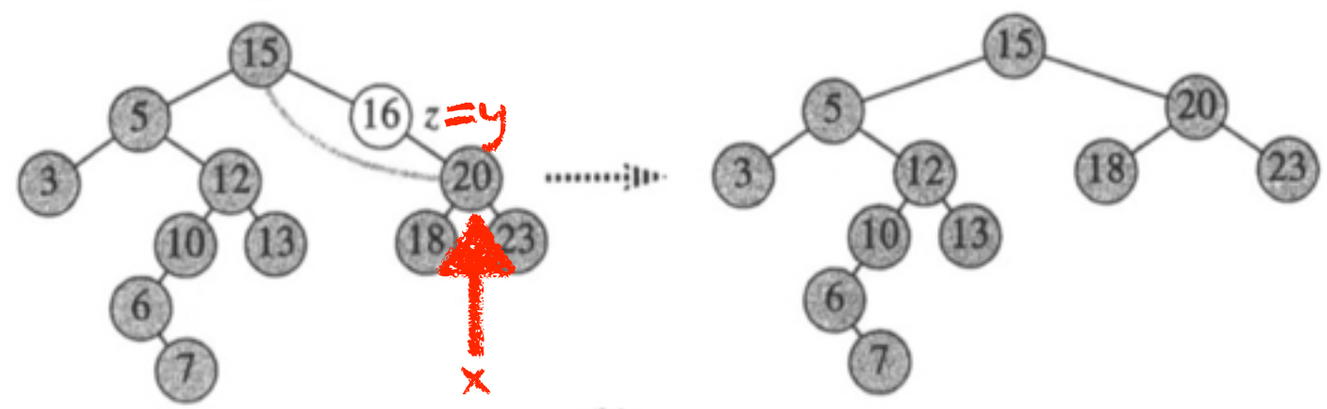


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

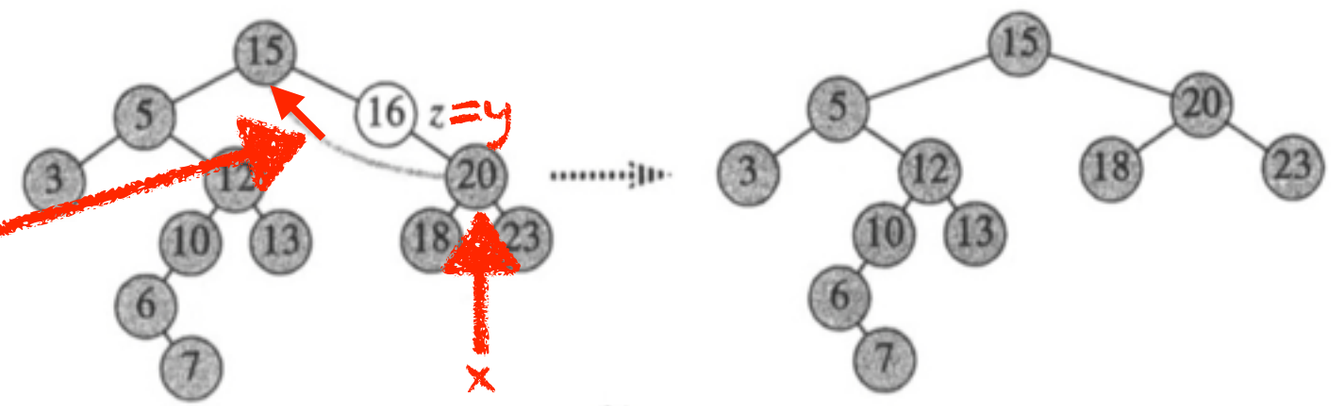


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

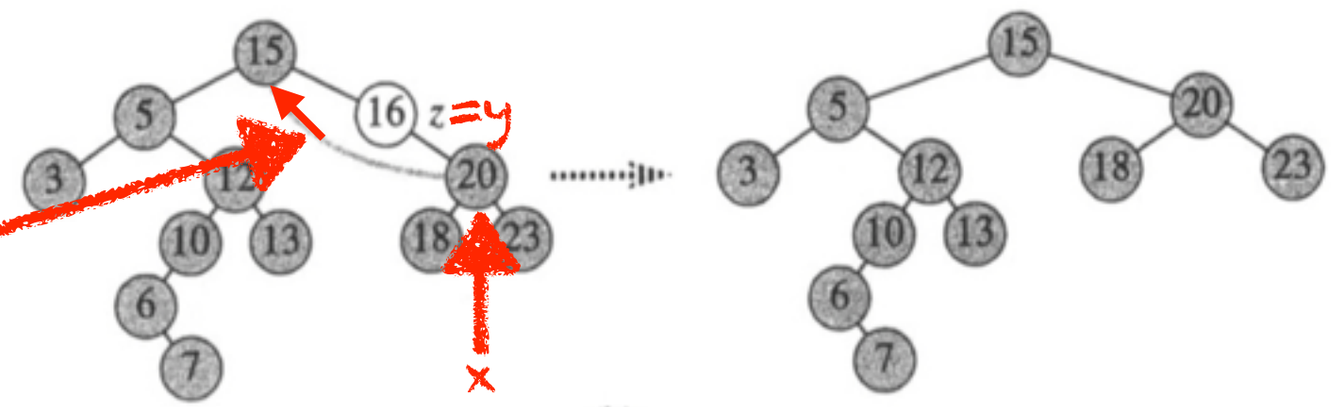


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

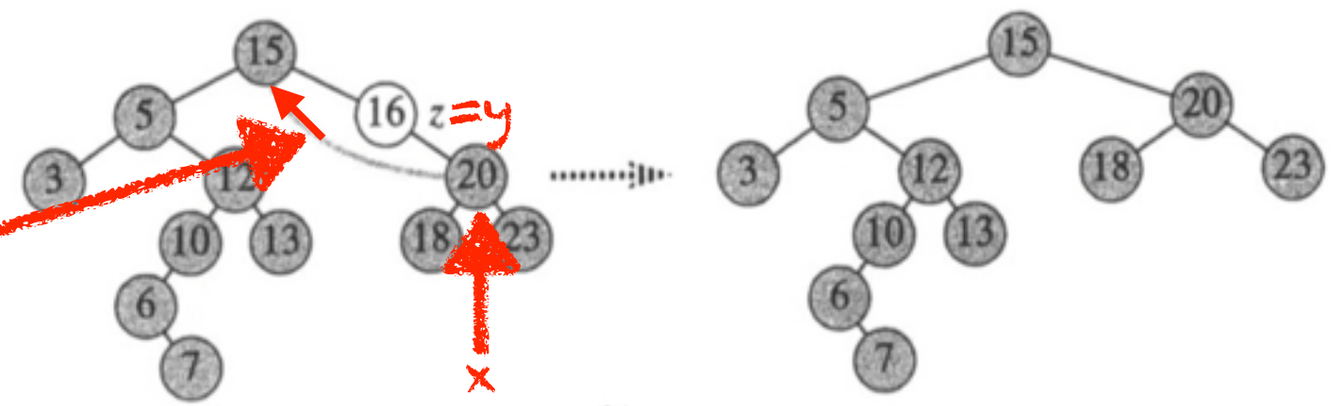


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

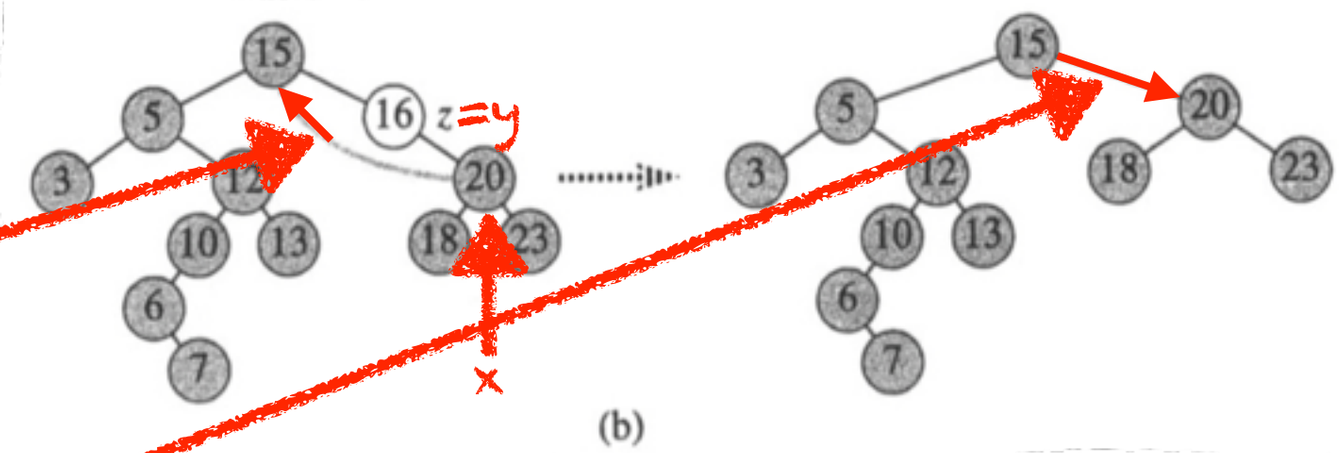


(b)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



(b)

-Was ist ein Suchbaum

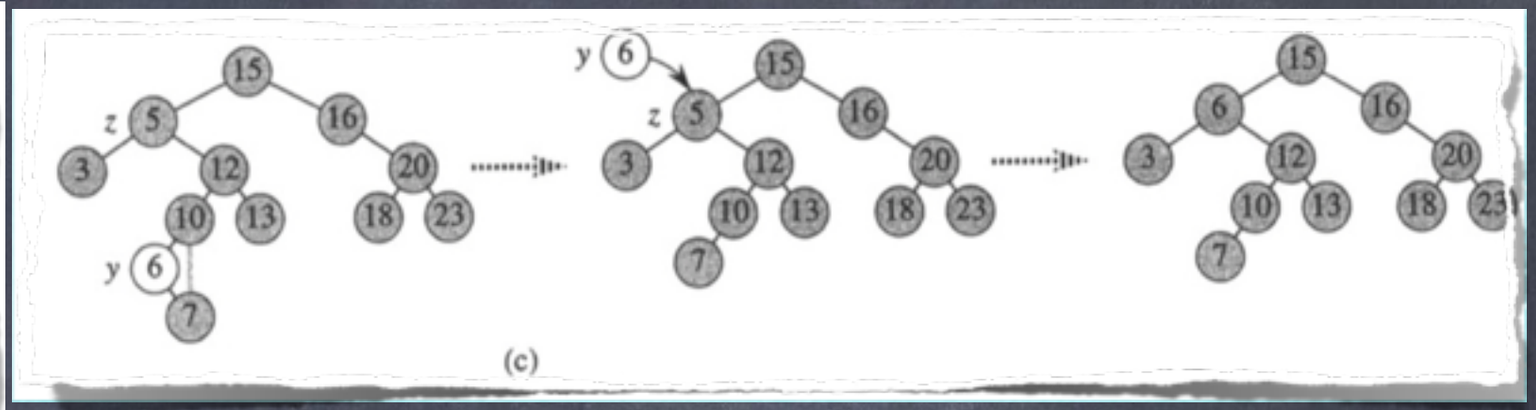
- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

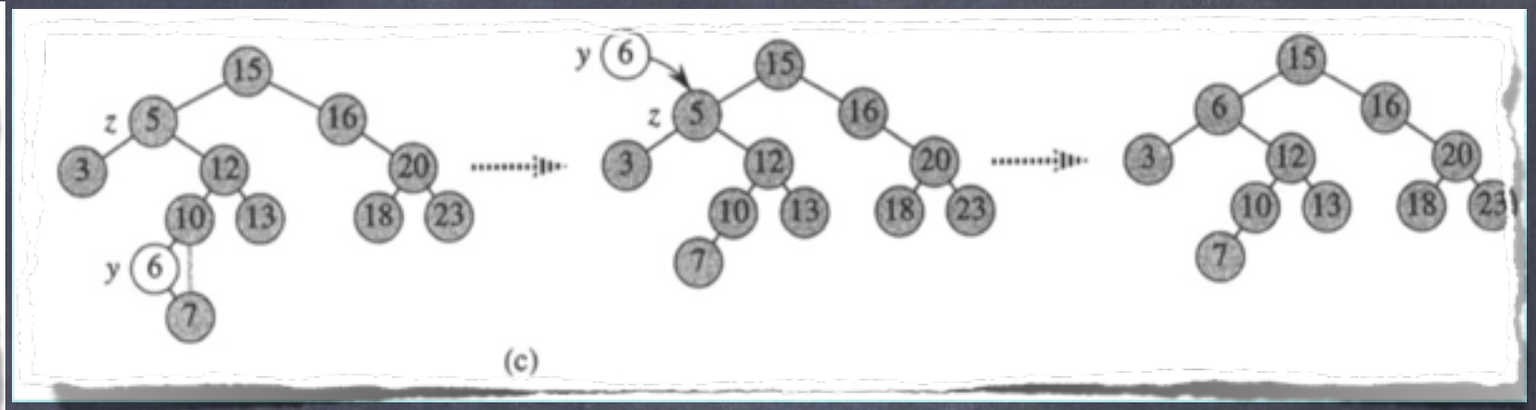


(c)

-Was ist ein Suchbaum

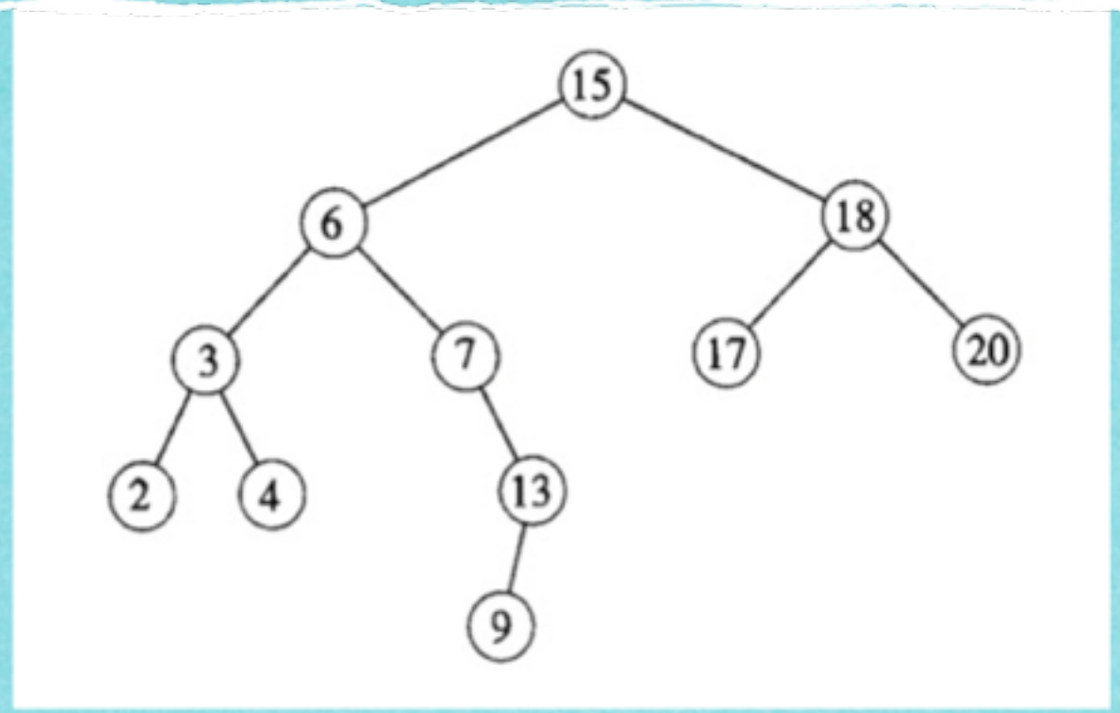
- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



(c)

- Was ich mir Suchbaum
- hlen
- ntfernen

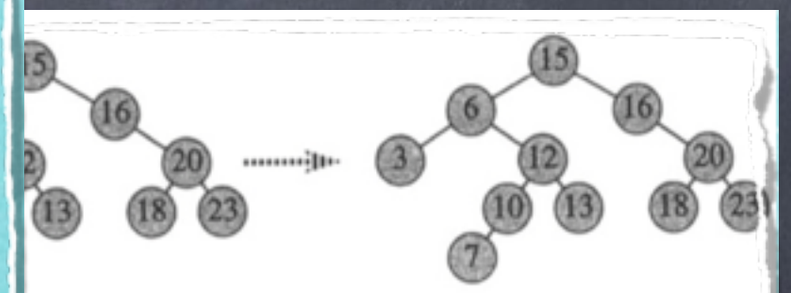


TREE-SUCCESSOR(x)

```
if rechts[x] ≠ NIL
    then return TREE-MINIMUM(rechts[x])
y ← p[x]
while y ≠ NIL und x = rechts[y]
        do x ← y
           y ← p[y]
return y
```

TREE-DELETE(T, z

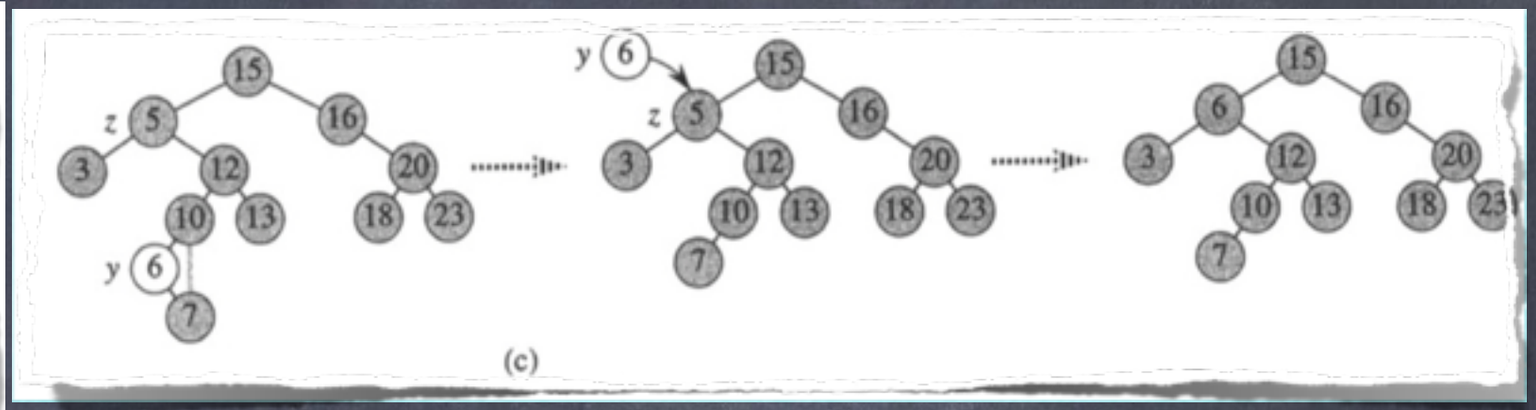
```
if links[z] = NI
    then y ← z
    else y ← 
if links[y] ≠ NI
    then x ← l
    else x ← r
if x ≠ NIL
    then p[x] ←
if p[y] = NIL
    then wurze
    else if y =
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

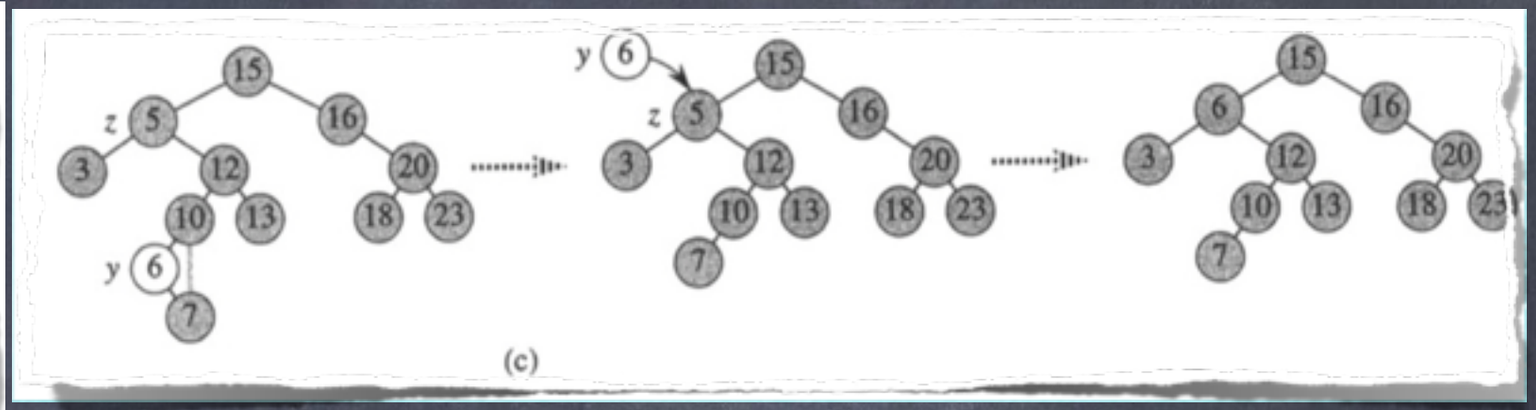


(c)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

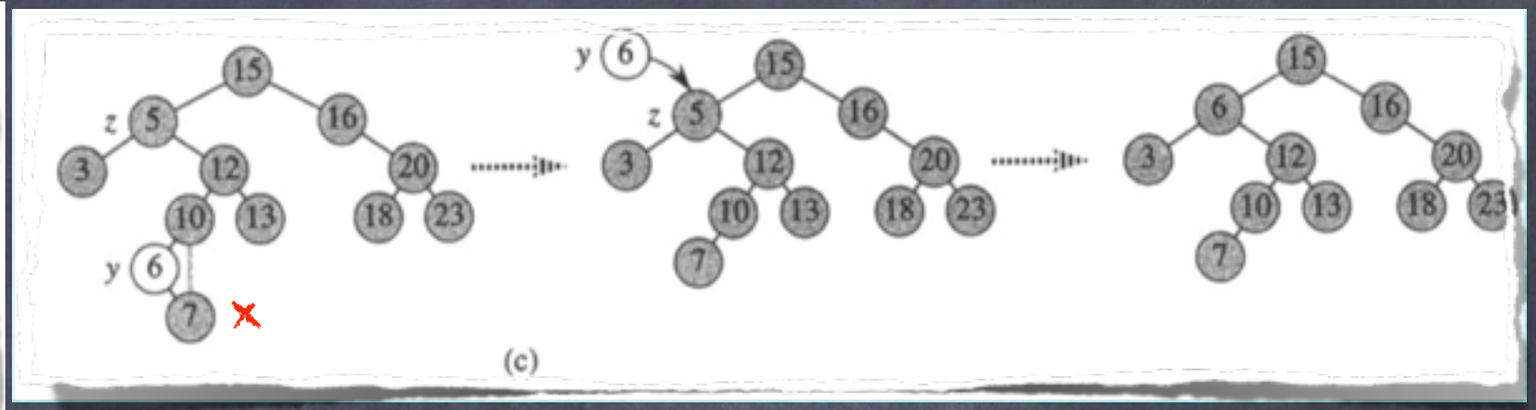


(c)

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

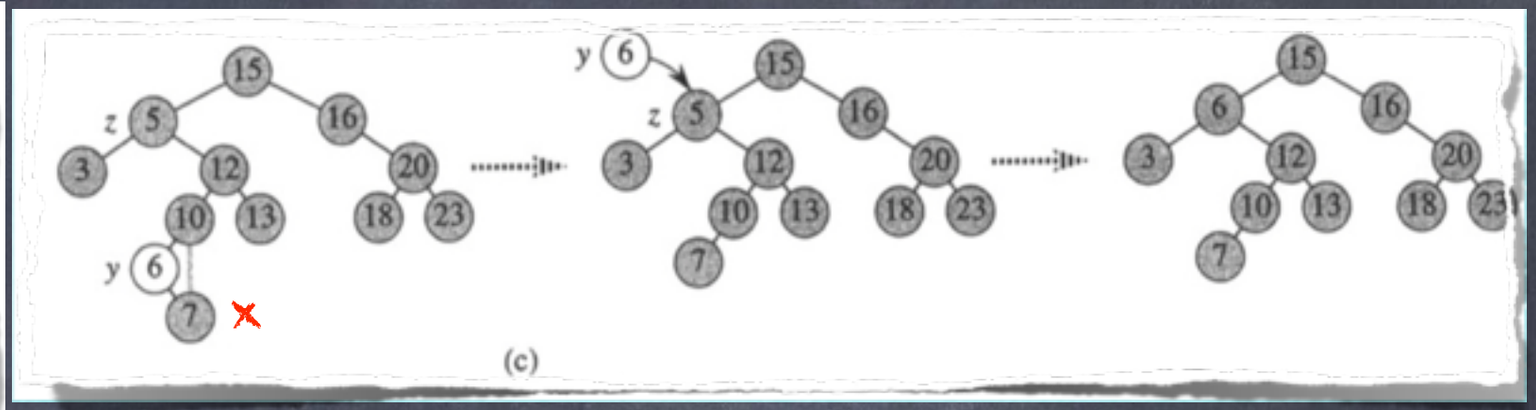
```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

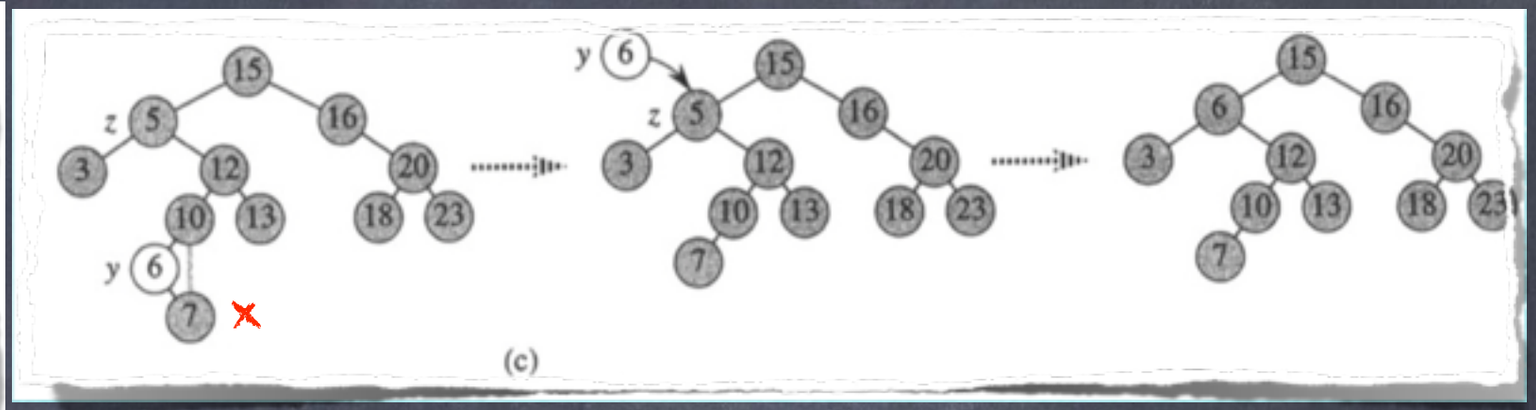
```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

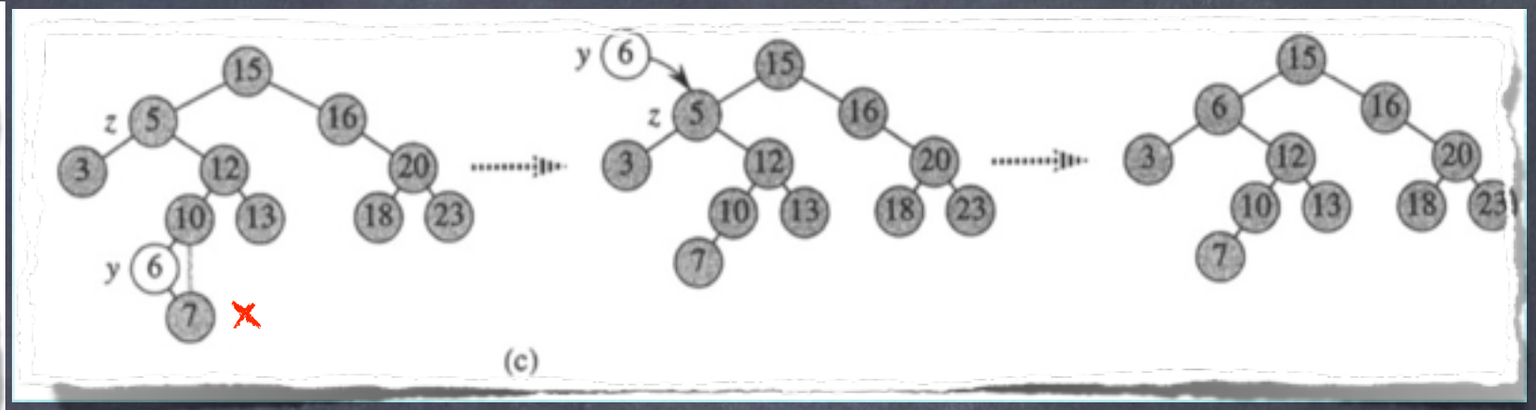
```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

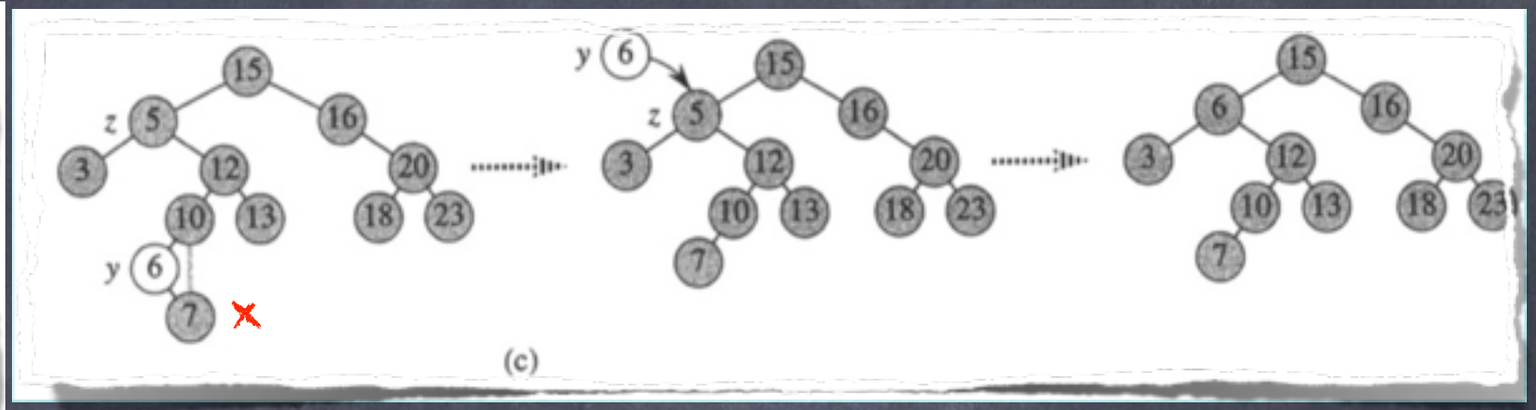
```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

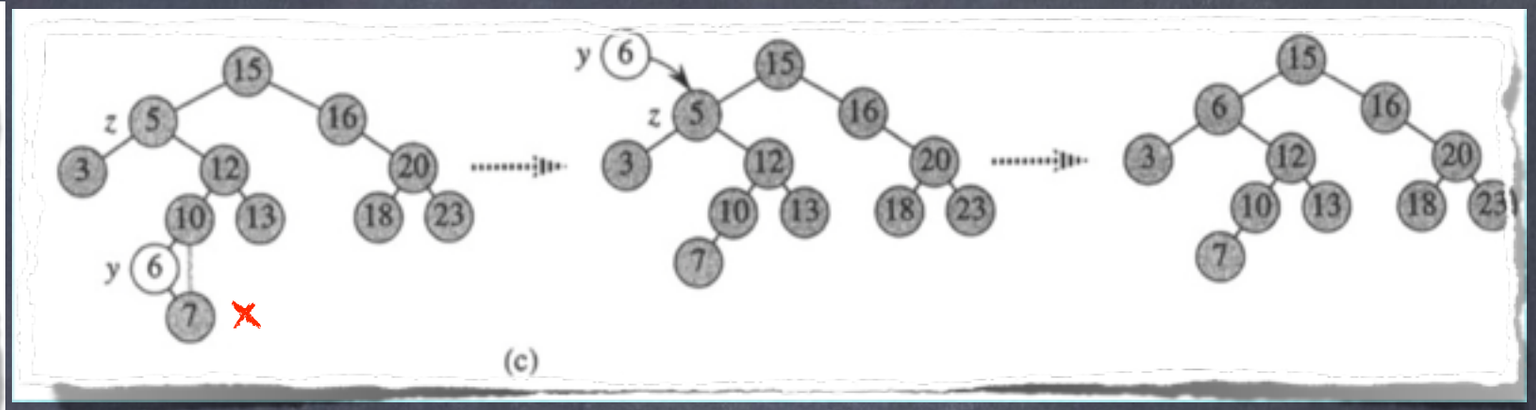
```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

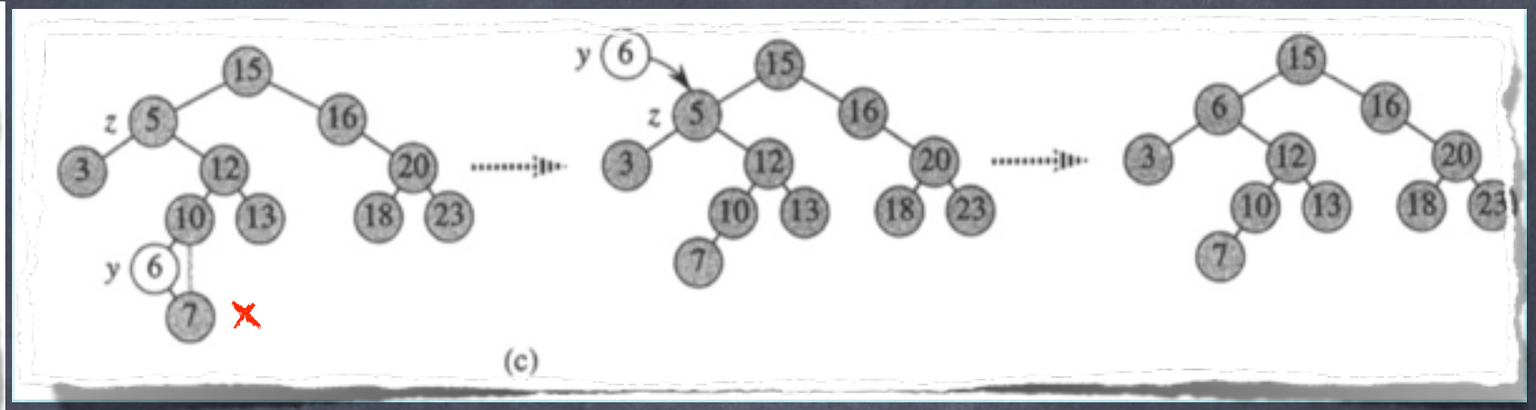
```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```

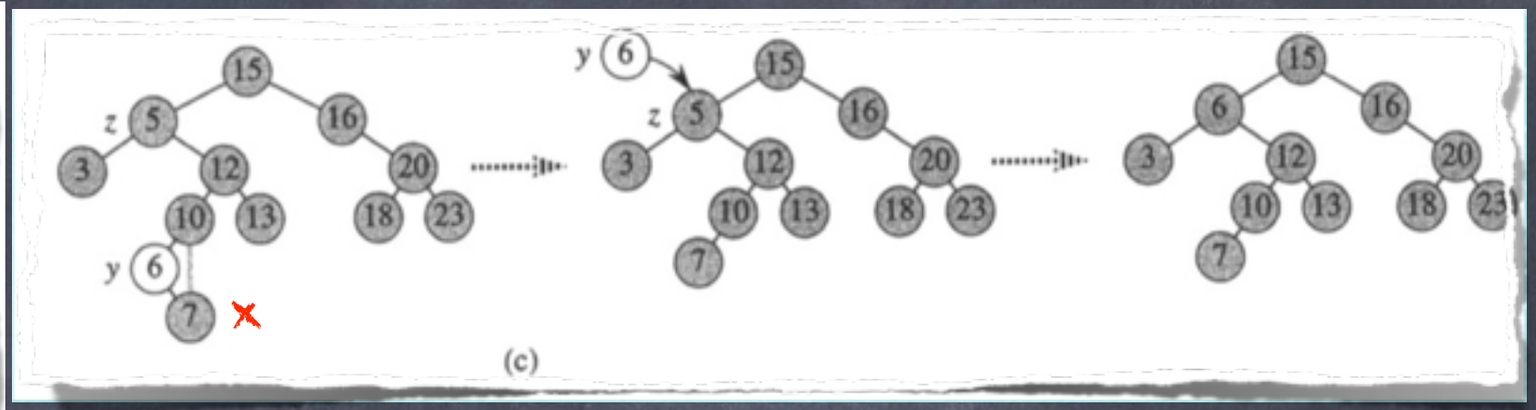


y kann kein linkes Kind haben... warum?

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



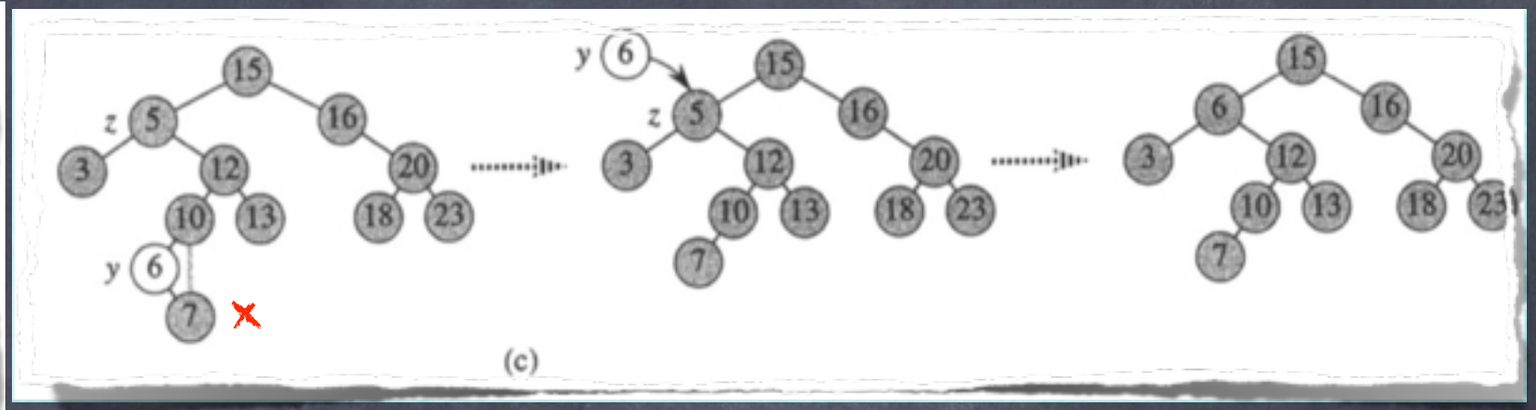
y kann kein linkes Kind haben... warum?

sonst wäre y nicht der Nachfolger von z!

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$
- -dyn. Verwaltung: Hinzufügen, Entfernen

```
TREE-DELETE(T, z)
if links[z] = NIL oder rechts[z] = NIL
    then y ← z
    else y ← TREE-SUCCESSOR(z)
if links[y] ≠ NIL
    then x ← links[y]
    else x ← rechts[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then wurzel[T] ← x
    else if y = links[p[y]]
            then links[p[y]] ← x
            else rechts[p[y]] ← x
if y ≠ z
    then schlüssel[z] ← schlüssel[y]
         kopiere die Satellitendaten von y in z
return y
```



y kann kein linkes Kind haben... warum?

sonst wäre y nicht der Nachfolger von z!

- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel
  - Totale Ordnung: bsp. ganzel Zahlen
  - linkes Kind $<$, rechtes Kind $\geq$
  - bin. Suche $\mathcal{O}(n)$
  - dyn. Verwaltung: Hinzufügen, Entfernen

- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel
  - Totale Ordnung: bsp. ganzel Zahlen
  - linkes Kind $<$, rechtes Kind $\geq$
  - bin. Suche $\mathcal{O}(n)$ zu langsam
  - dyn. Verwaltung: Hinzufügen, Entfernen

- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel
  - Totale Ordnung: bsp. ganzel Zahlen
  - linkes Kind $<$, rechtes Kind $\geq$
  - bin. Suche $\mathcal{O}(n)$ zu langsam
  - dyn. Verwaltung: Hinzufügen, Entfernen

- Was ist ein AVL-Baum:

- Was ist ein Suchbaum
  - Bin. Baum
  - Schlüssel
  - Totale Ordnung: bsp. ganzel Zahlen
  - linkes Kind $<$, rechtes Kind $\geq$
  - bin. Suche $\mathcal{O}(n)$ zu langsam
  - dyn. Verwaltung: Hinzufügen, Entfernen

- Was ist ein AVL-Baum:
  - bin. Suchbaum mit:
    Höhen der Kinder unterscheiden sich max. um 1

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$ zu langsam
- -dyn. Verwaltung: Hinzufügen, Entfernen

-Was ist ein AVL-Baum:

- -bin. Suchbaum mit:
  Höhen der Kinder unterscheiden sich max. um 1
- -bin. Suche $\mathcal{O}(\log(n))$ warum?

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$ zu langsam
- -dyn. Verwaltung: Hinzufügen, Entfernen

-Was ist ein AVL-Baum:

- -bin. Suchbaum mit:
  Höhen der Kinder unterscheiden sich max. um 1
- -bin. Suche $\mathcal{O}(\log(n))$ warum?
  $\mathcal{O}(h)$ und $h \in \mathcal{O}(\log(n))$

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$ zu langsam
- -dyn. Verwaltung: Hinzufügen, Entfernen

-Was ist ein AVL-Baum:

- -bin. Suchbaum mit:
  Höhen der Kinder unterscheiden sich max. um 1
- -bin. Suche $\mathcal{O}(\log(n))$ warum?

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

Beweis, siehe VL

-Was ist ein Suchbaum

- -Bin. Baum
- -Schlüssel
- -Totale Ordnung: bsp. ganzel Zahlen
- -linkes Kind $<$, rechtes Kind $\geq$
- -bin. Suche $\mathcal{O}(n)$ zu langsam
- -dyn. Verwaltung: Hinzufügen, Entfernen

-Was ist ein AVL-Baum:

- -bin. Suchbaum mit:
  Höhen der Kinder unterscheiden sich max. um 1
- -bin. Suche $\mathcal{O}(\log(n))$ warum?
  $\mathcal{O}(h)$ und $h \in \mathcal{O}(\log(n))$

  Beweis, siehe VL
- -Aufrechterhaltung von $h \in \mathcal{O}(\log(n))$

Aufrechterhaltung: bal. Höhe

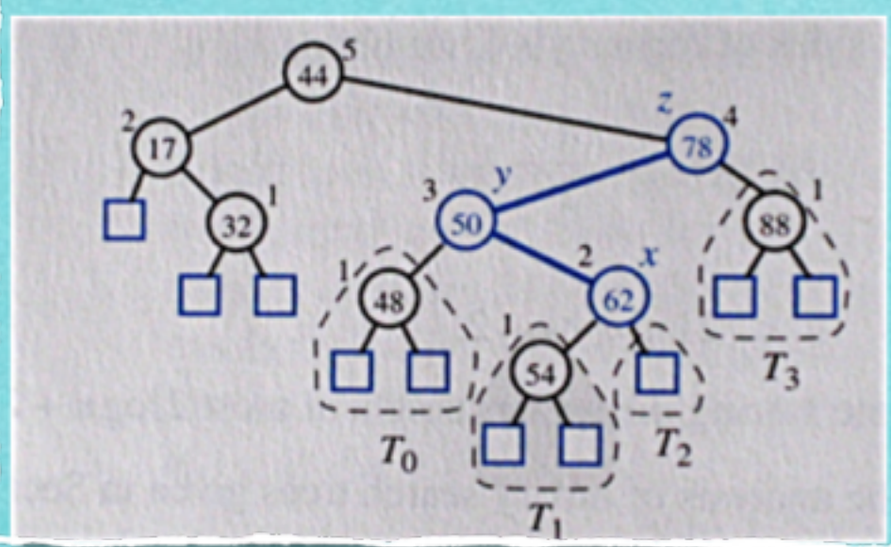
RESTRUCTURE(x)

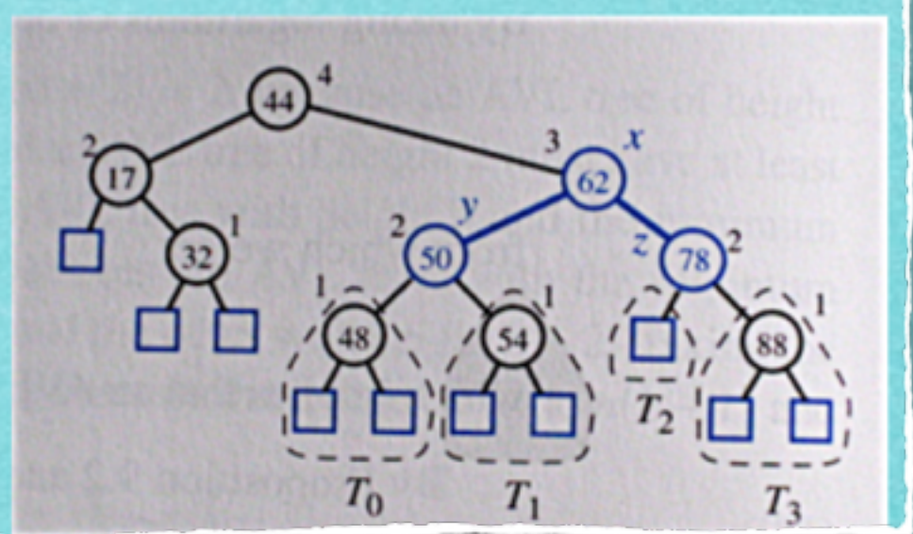
1. Sei (a, b, c) die Größensortierung der Knoten x, y, z;
   seien (T_0, T_1, T_2, T_3) die Größensortierung der vier Teilbäume unter x, y, z, die nicht Wurzeln x, y, z haben

2. Ersetze den Teilbaum mit Wurzel z durch einen neuen Teilbaum mit Wurzel b.

3. Setze a als linkes Kind von b, mit T_0 und T_1 als linken und rechten Teilbaum unter a;
   setze c als rechtes Kind von b, mit T_2 und T_3 als linken und rechten Teilbaum unter c.

4. RETURN





Aufrechterhaltung: bal. Höhe

Betrachte 1.+2. direkten Vorfahr
->Sortiere

Schneide Teilbäume ab
->Sortiere

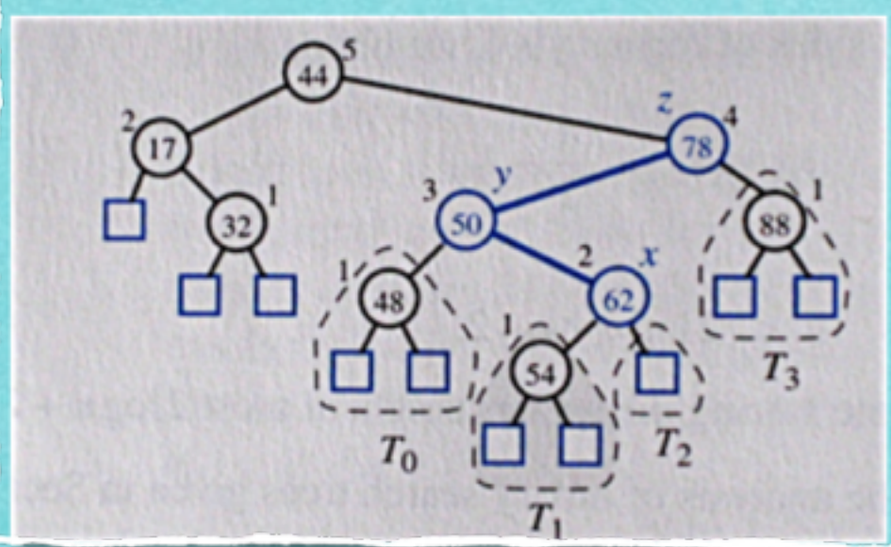
RESTRUCTURE(x)

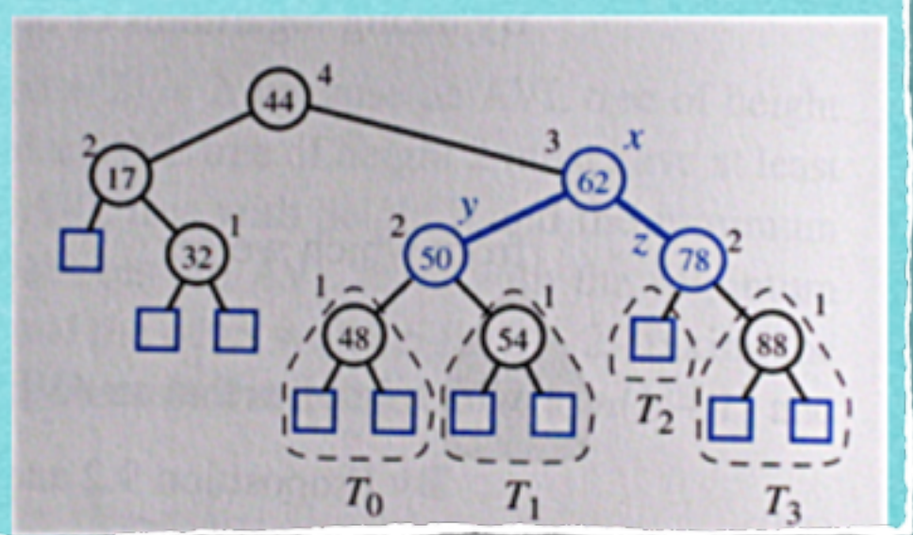
1. Sei (a, b, c) die Größensortierung der Knoten x, y, z;
   seien (T_0, T_1, T_2, T_3) die Größensortierung der vier Teilbäume unter x, y, z, die nicht Wurzeln x, y, z haben

2. Ersetze den Teilbaum mit Wurzel z durch einen neuen Teilbaum mit Wurzel b.

3. Setze a als linkes Kind von b, mit T_0 und T_1 als linken und rechten Teilbaum unter a;
   setze c als rechtes Kind von b, mit T_2 und T_3 als linken und rechten Teilbaum unter c.

4. RETURN





Aufrechterhaltung: bal. Höhe - Einfügen

Anwenden von RESTRUCTURE:

Angenommen, durch Hinzufügen eines Knotens v ist der Baum unbalanciert geworden.
Sei z der nach dem Einfügen niedrigste unbalancierte Vorfahre von v.
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Aufrechterhaltung: bal. Höhe - Einfügen

Anwenden von RESTRUCTURE:

Angenommen, durch Hinzufügen eines Knotens v ist der Baum unbalanciert geworden.
Sei z der nach dem Einfügen niedrigste unbalancierte Vorfahre von v.
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Warum?

Aufrechterhaltung: bal. Höhe - Einfügen

Anwenden von RESTRUCTURE:

Angenommen, durch Hinzufügen eines Knotens v ist der Baum unbalanciert geworden.
Sei z der nach dem Einfügen niedrigste unbalancierte Vorfahre von v.
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Warum?
Da sonst nicht unbalanciert

Aufrechterhaltung: bal. Höhe - Einfügen

Anwenden von RESTRUCTURE:

Angenommen, durch Hinzufügen eines Knotens v ist der Baum unbalanciert geworden.
Sei z der nach dem Einfügen niedrigste unbalancierte Vorfahre von v.
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

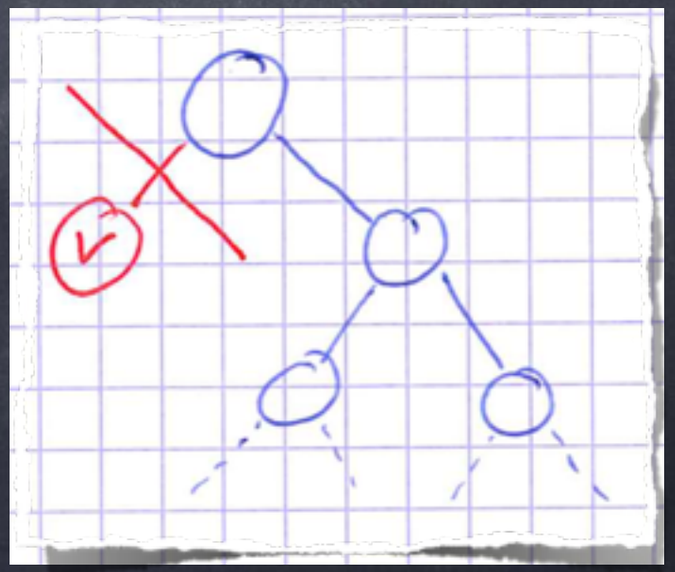
Warum?
Da sonst nicht unbalanciert

Beispiel an der Tafel

Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

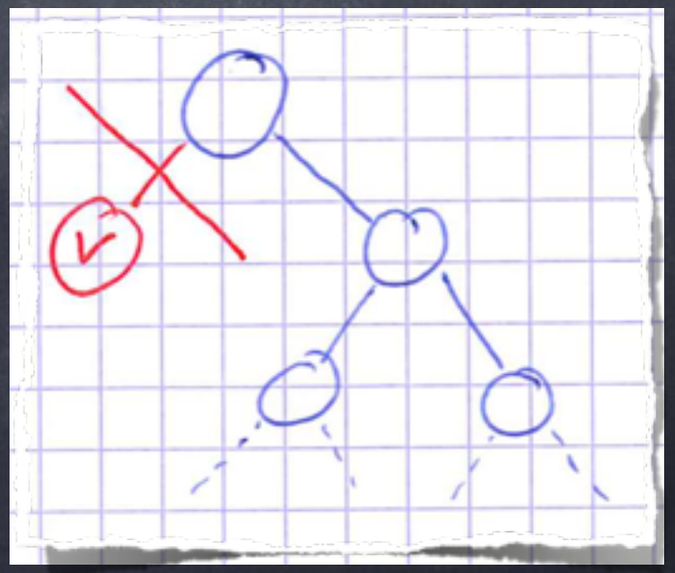
Angenommen, durch Hinzufügen eines Knotens v ist der Baum unbalanciert geworden.
Sei z der nach dem Einfügen niedrigste unbalancierte Vorfahre von v.
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.



Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

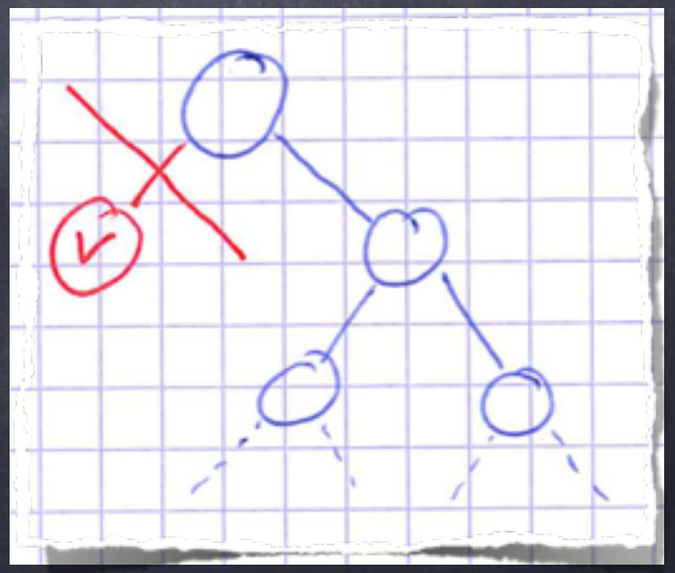
Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Einfügen niedrigste unbalancierte Vorfahre von v.
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.



Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

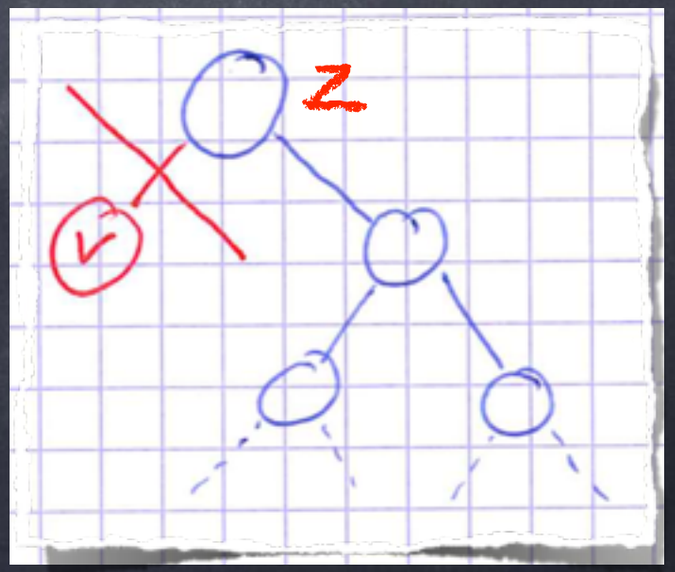
Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.



Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

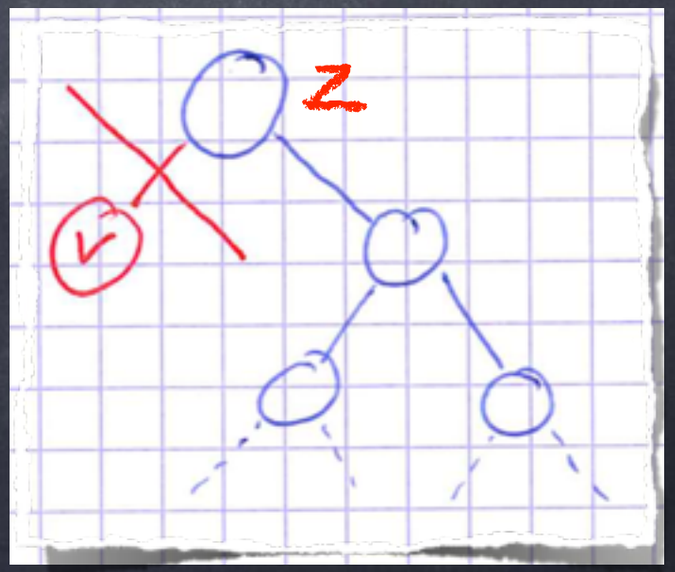
Angenommen, durch Löschen eines Knotens v ist der Baum unbalanciert geworden.
Sei z der nach dem Löschen niedrigste unbalancierte Vorfahre von v.
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.



Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

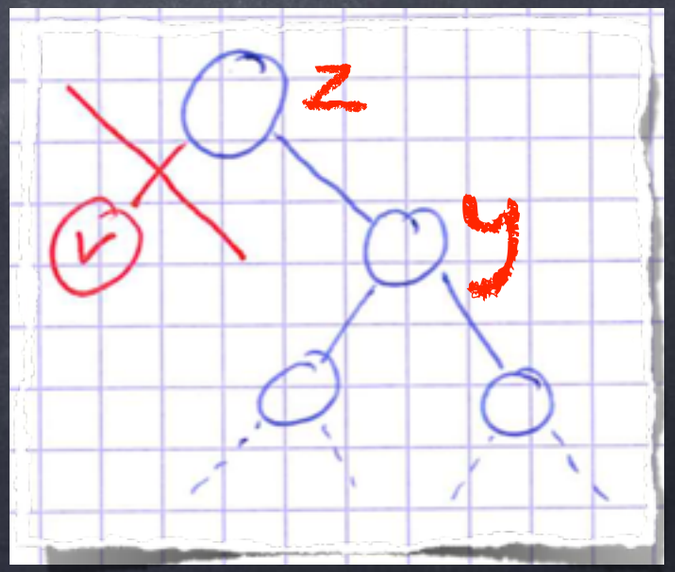
Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist; y muss zwei höher sein als das andere Kind von z.**
Sei x das Kind von y, das im selben Teilbaum wie v liegt.



Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

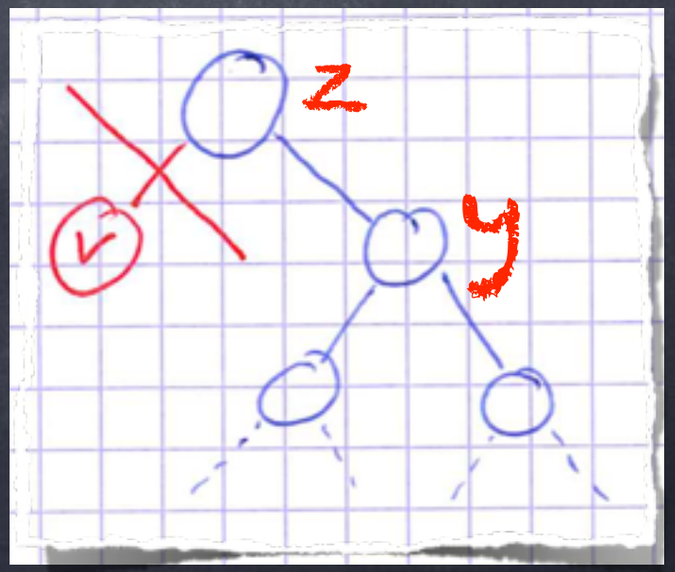
Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist; y muss zwei höher sein als das andere Kind von z.**
Sei x das Kind von y, das im selben Teilbaum wie v liegt.



Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist;** der Teilbaum von z mit y muss 2 höher sein als der andere
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

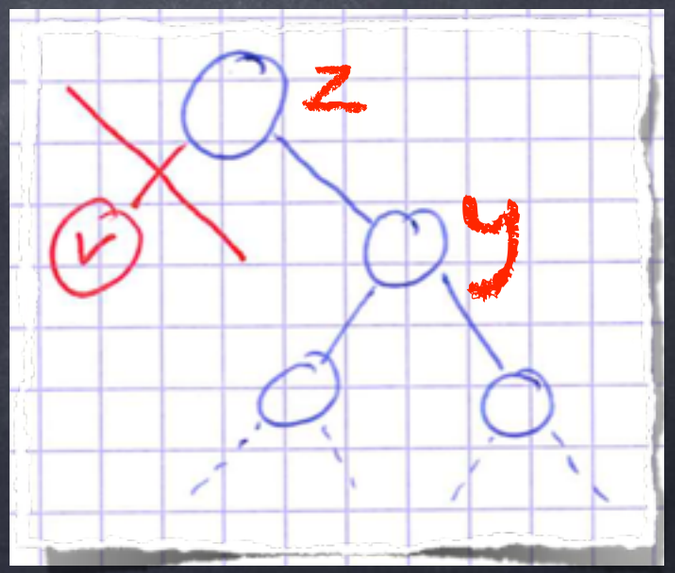


Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist;** der Teilbaum von z mit y muss 2 höher sein als der andere
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Warum?



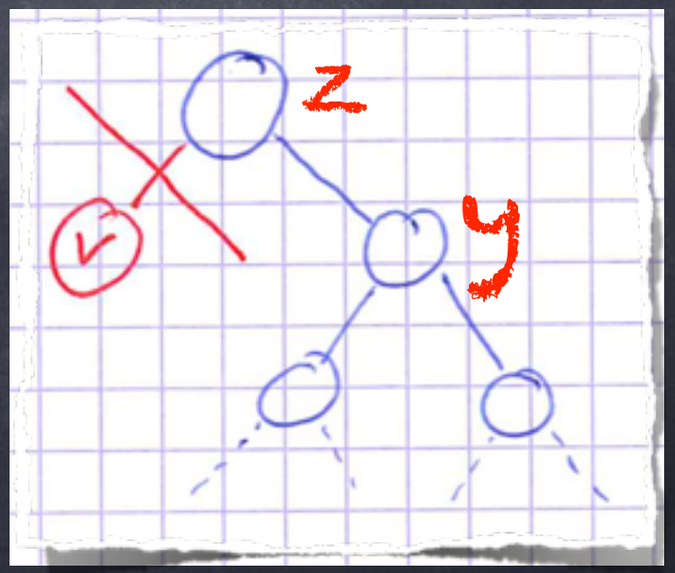
Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist;** der Teilbaum von z mit y muss 2 höher sein als der andere
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Warum?

Weil z unbalanciert



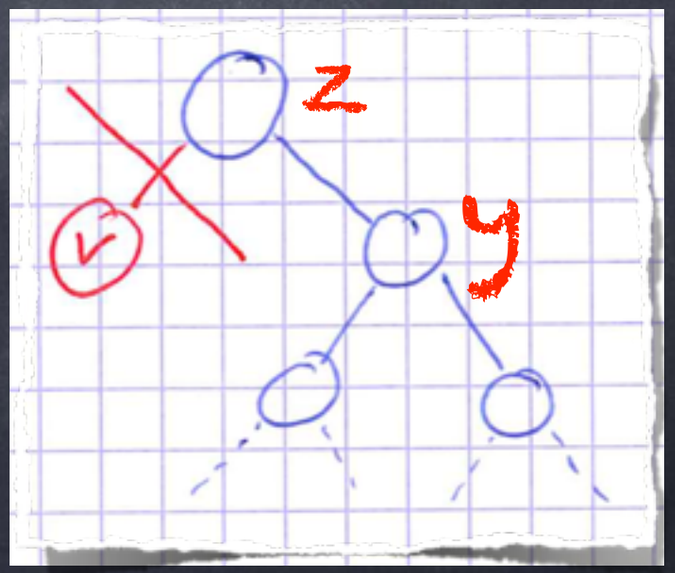
Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist;** der Teilbaum von z mit y muss 2 höher sein als der andere
Sei x das **Kind von y** mit nicht kleinerer Höhe

Warum?

Weil z unbalanciert



Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Hinzufügen eines Knotens v ist der Baum unbalanciert geworden.
Sei z der nach dem Einfügen niedrigste unbalancierte Vorfahre von v.
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Beispiel an der Tafel

Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Einfügen niedrigste unbalancierte Vorfahre von v.
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Beispiel an der Tafel

Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen eines Knotens v ist der Baum unbalanciert geworden.
Sei z der nach dem Löschen niedrigste unbalancierte Vorfahre von v.
Sei y das Kind von z, das Vorfahre von v ist; y muss zwei höher sein als das andere Kind von z.
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Beispiel an der Tafel

Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist; y muss zwei höher sein als das andere Kind von z.**
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Beispiel an der Tafel

Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist;** der Teilbaum von z mit y muss 2 höher sein als der andere
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Beispiel an der Tafel

Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist;** der Teilbaum von z mit y muss 2 höher sein als der andere
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Warum?

Beispiel an der Tafel

Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist;** der Teilbaum von z mit y muss 2 höher sein als der andere
Sei x das Kind von y, das im selben Teilbaum wie v liegt.

Warum?

Weil z unbalanciert

Beispiel an der Tafel

Aufrechterhaltung: bal. Höhe -Löschen

Anwenden von RESTRUCTURE:

Angenommen, durch Löschen **eines Knotens v ist der Baum unbalanciert geworden.**
Sei z der nach dem Löschen **niedrigste unbalancierte Vorfahre von v.**
Sei y das Kind von z, das nicht Vorfahre **von v ist;** der Teilbaum von z mit y muss 2 höher sein als der andere
Sei x das **Kind von y** mit nicht kleinerer Höhe

Warum?

Weil z unbalanciert

Beispiel an der Tafel