Technische Universität Braunschweig

Institute of Operating Systems
and Computer Networks
Algorithms Group

Algorithmen und Datenstrukturen — Wintersemester 2017/2018
Übung#5, 21.12.2017

Christian Rieck, Arne Schmidt

Bäume

Satz

Jeder gerichtete Baum hat einen Knoten mit Eingangsgrad 0.

Beweis

Angenommen, sei T ein gerichteter Baum und es existiert kein solcher Knoten. Sei $v \in V$. Betrachte Sequenz $v = v_0, v_1, v_2, \ldots, v_n$ wobei v_i der Vater von $v_{i-1} (\forall\ 1 \leq i \leq n)$ ist. Mindestens ein Knoten muss in der Sequenz zweimal auftreten. Sei u der Knoten, der zuerst zweimal auftritt. Betrachte Teilsequenz vom ersten Auftreten von u zum zweiten Auftreten in umgekehrter Reihenfolge. Dies ist ein Kreis in T. Widerspruch dazu, dass T ein Baum ist.

Bäume

Satz

In einem gewurzelten Baum T gibt es einen eindeutigen Weg von der Wurzel zu jedem anderen Knoten.

(Teil-)Beweis

Annahme: Für einen Knoten v gibt es zwei unterschiedliche Wege $A = (w = a_0, a_1, \ldots, a_n = v)$ und $B = (w = b_0, b_1, \ldots, b_m = v)$ von der Wurzel w zu v. Betrachte den vom Ende gezählt letzten Knoten u, der in beiden Sequenzen gleich ist ($a_i = b_j = u, \ i, j > 0$). Dann hat u zwei unterschiedliche Väter a_{i-1} und b_{j-1}, im Widerspruch dazu, dass T ein gewurzelter Baum ist.

Bäume

Satz

In einem gewurzelten Baum T gibt es einen eindeutigen Weg von der Wurzel zu jedem anderen Knoten.

(Teil-)Beweis

Annahme: Für einen Knoten v gibt es zwei unterschiedliche Wege

$A = (w = a_0, a_1, \ldots, a_{i-1}, a_i = v)$ und $B = (w = b_0, b_1, \ldots, b_{j-1}, b_j = v)$

von de

Knoter > 0).

Dann hat u zwei unterschiedliche Väter a_{i-1} und b_{j-1}, im Widerspruch dazu, dass T ein gewurzelter Baum ist.

Selbst überlegen: Warum gibt es für jeden Knoten v einen Weg von w zu v?

Binäre Bäume

Binäre Bäume sind gewurzelte und gerichtete Bäume.
Jeder Knoten hat kein, ein oder zwei Kind(er).
Jeder Knoten (außer der Wurzel) hat genau einen Vater.

Mit Totalordnung der Elemente gibt das einen binären Suchbaum, bei dem Suchen, Einfügen, Löschen,... realisiert werden kann.

Binäre Bäume

Seien folgende Elemente gegeben:

11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

Schlüssel im linken Teilbaum: ≤
Schlüssel im rechten Teilbaum: >

Das Einfügen dieser Elemente von links nach rechts in einen binären Suchbaum ergibt den folgenden Baum.

Binäre Bäume

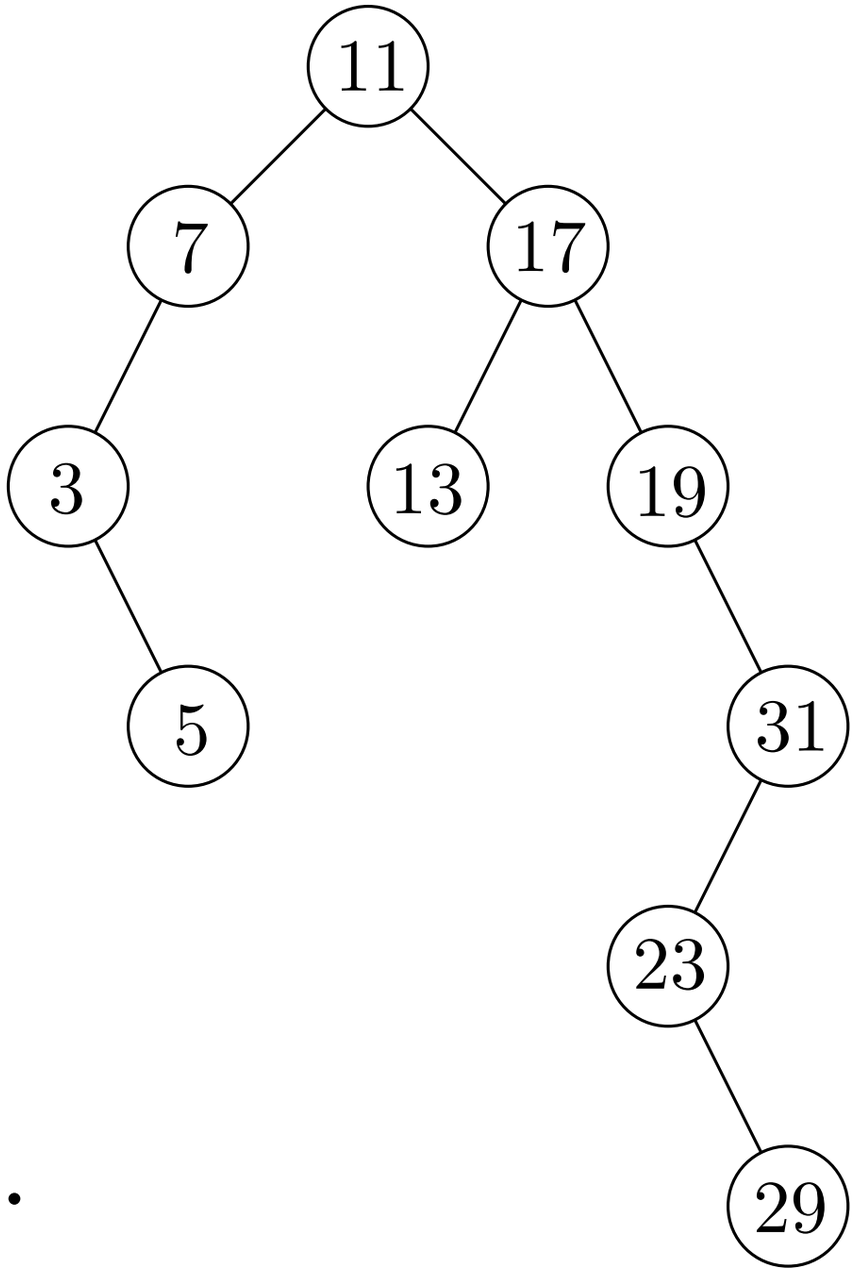
Seien folgende Elemente gegeben:

11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

Schlüssel im linken Teilbaum: ≤

Schlüssel im rechten Teilbaum: >

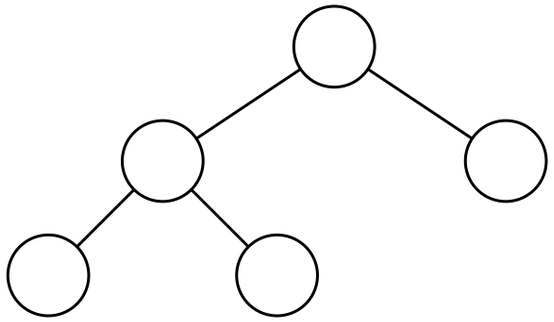
Das Einfügen dieser Elemente von links nach rechts in einen binären Suchbaum ergibt den folgenden Baum.



Binäre Bäume

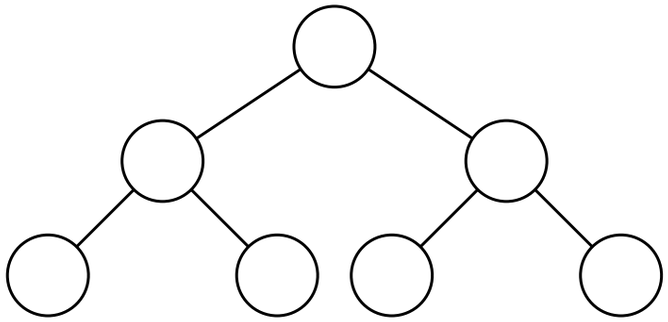
Voller binärer Baum

- jeder Knoten hat kein oder zwei Kind(er)



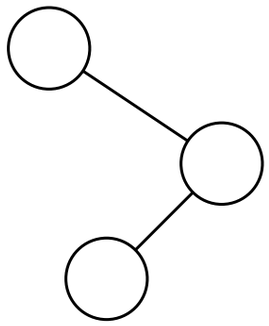
Vollständiger binärer Baum

- alle Blätter haben den gleichen Abstand zur Wurzel

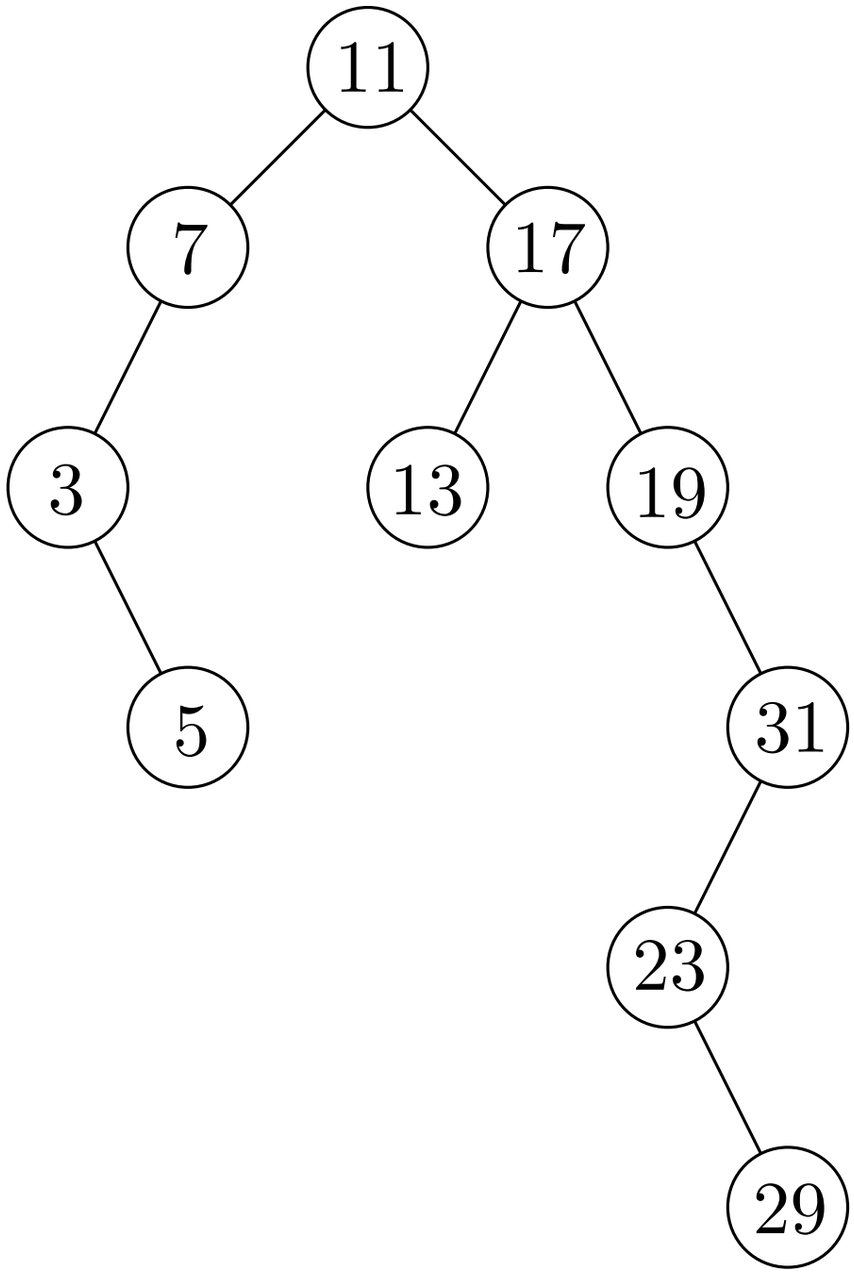


Degenerierter binärer Baum

- jeder Knoten hat maximal ein Kind



Binäre Bäume



- Suchen
- Minimum
- Maximum
- Nachfolger
- Vorgänger
- Einfügen → Satz 4.5
- Löschen → Satz 4.6

Satz 4.4 → O(h)

Binäre Bäume

Satz

Ein vollständiger binärer Suchbaum der Höhe h hat 2^{h-1} Blätter.

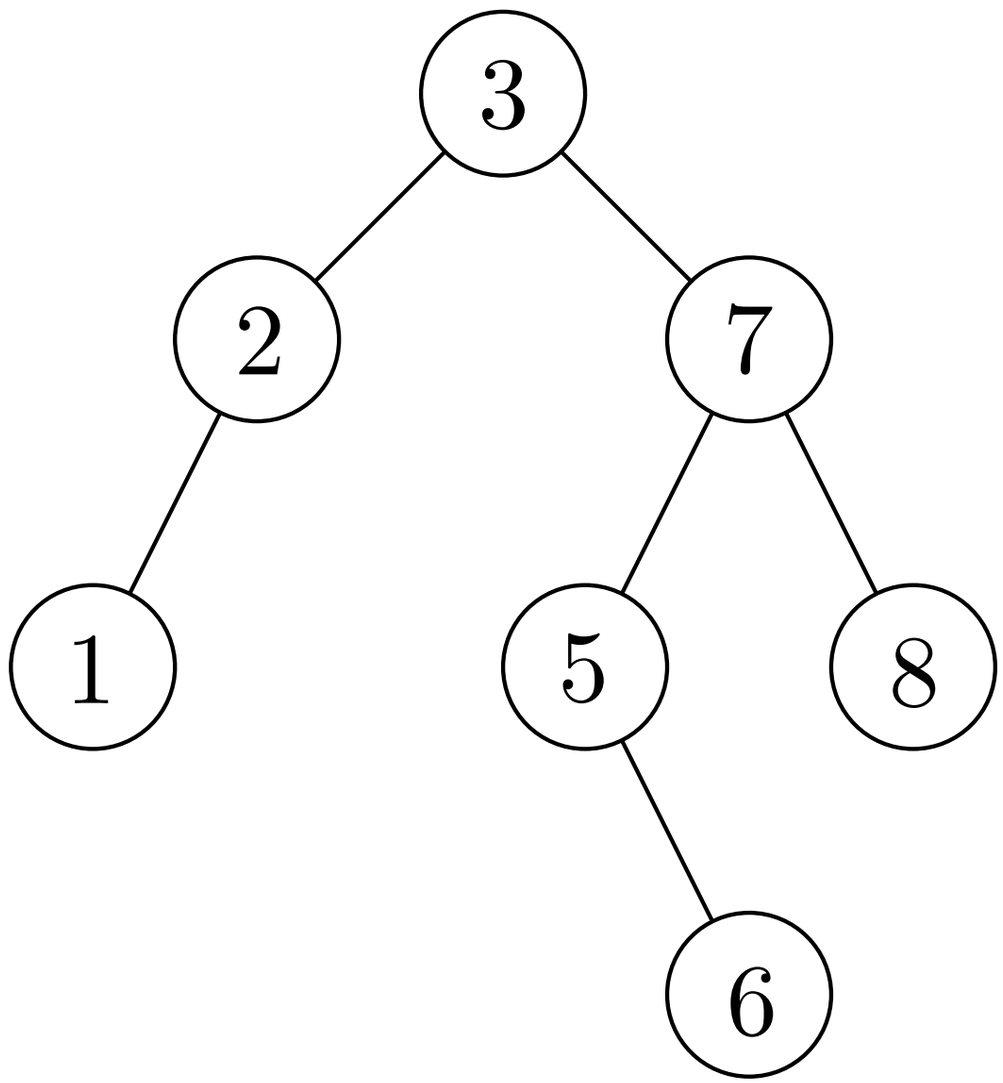
Beweis

Durch Induktion über h. Für h = 1 hat ein binärer Baum nur einen Knoten, der gleichzeitig ein Blatt ist.

Angenommen, jeder vollständige binäre Baum der Höhe h-1 hat 2^{h-2} Blätter.

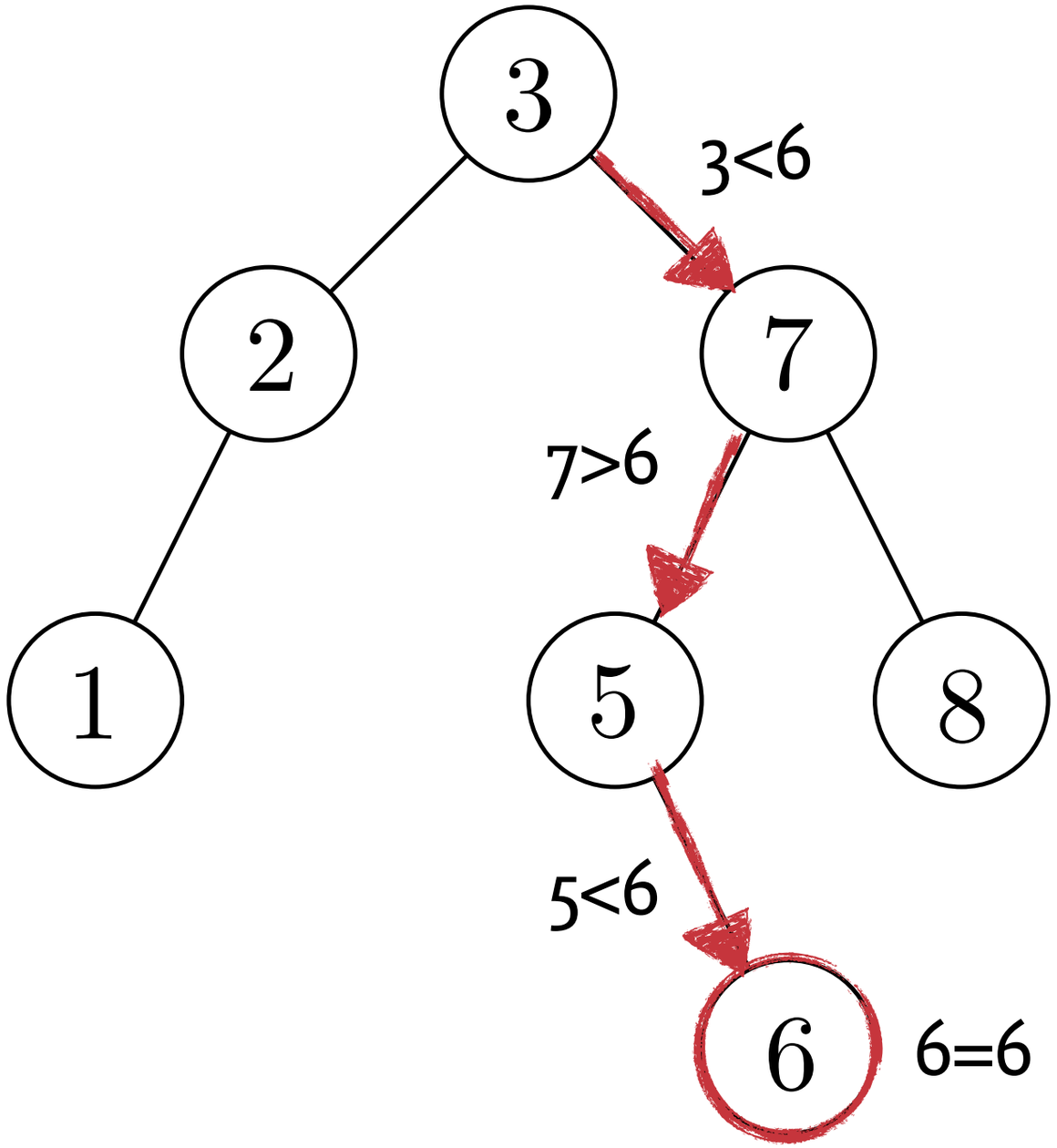
Einen vollständigen binären Baum der Höhe h erhält man aus einem Baum der Höhe h-1, in dem man jedem Blatt 2 Kinder gibt. Dies ergibt $2 \cdot 2^{h-2} = 2^{h-1}$ Blätter.

Traversierung

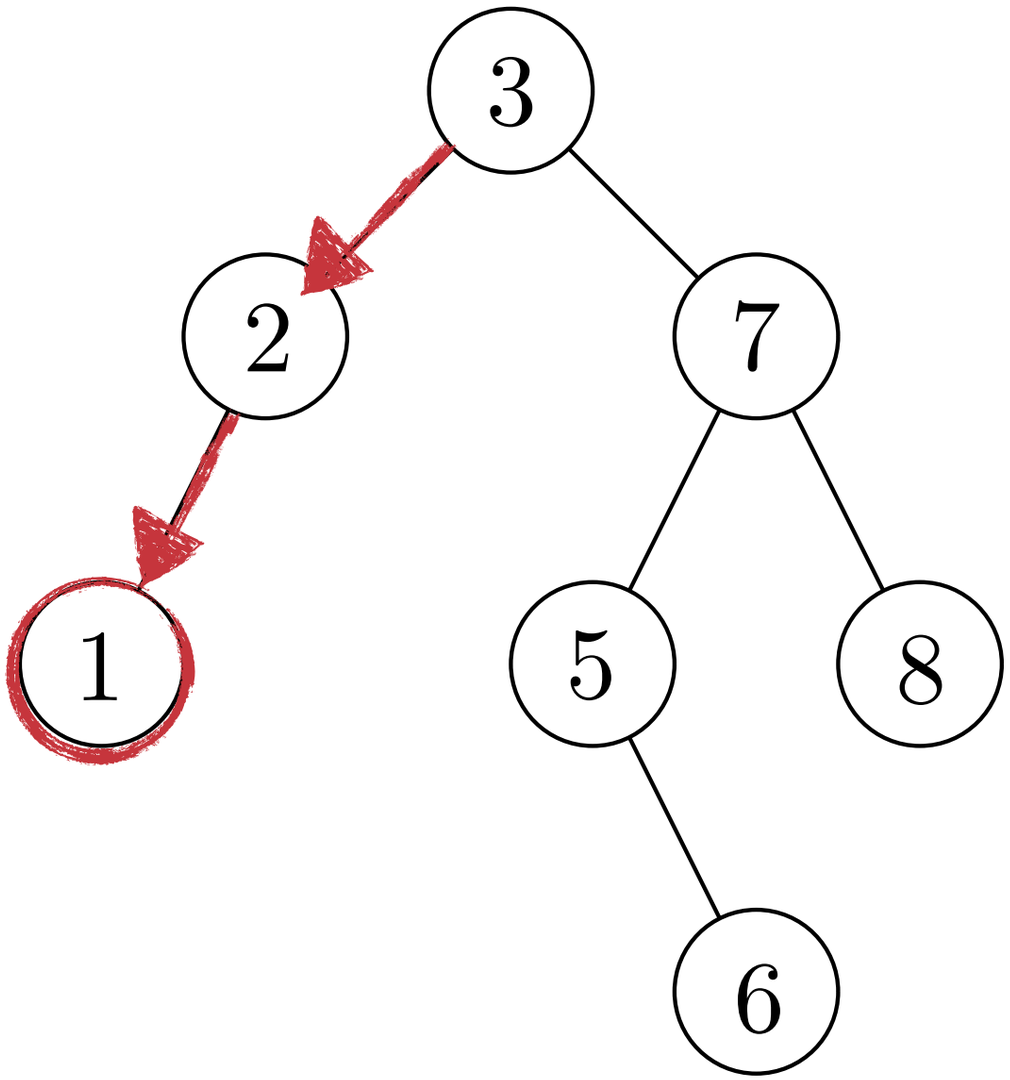


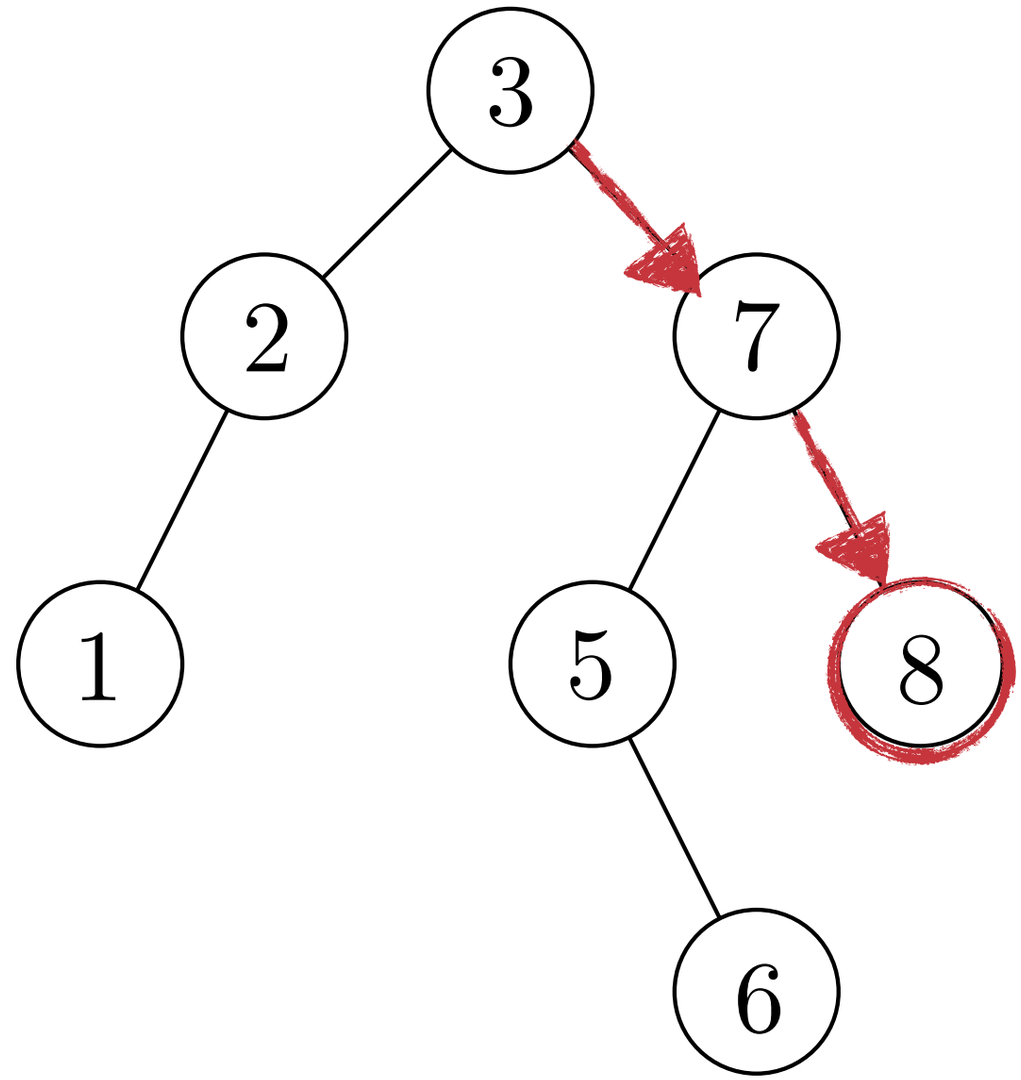
- in-order: 1,2,3,5,6,7,8
- pre-order: 3,2,1,7,5,6,8
- post-order: 1,2,6,5,8,7,3

Operation: Search(6)

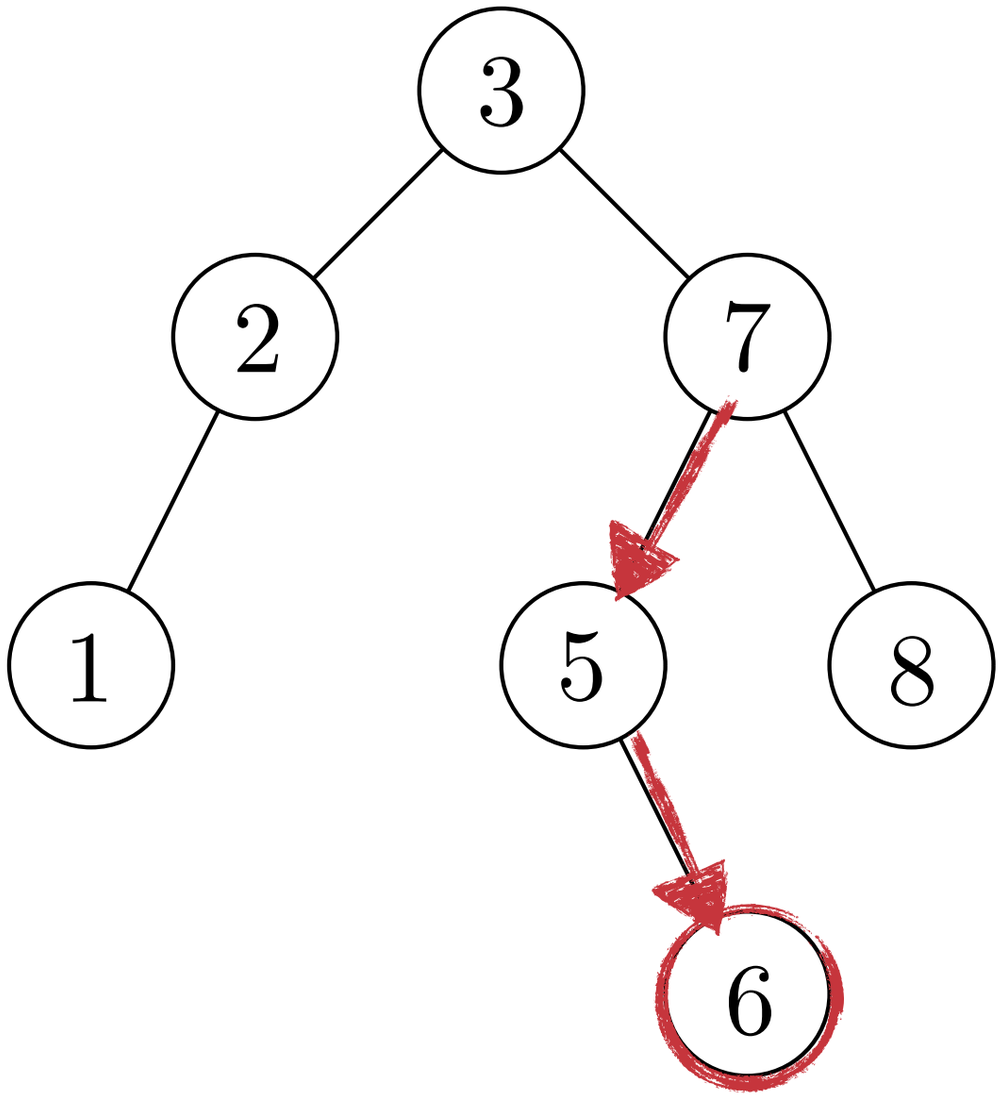


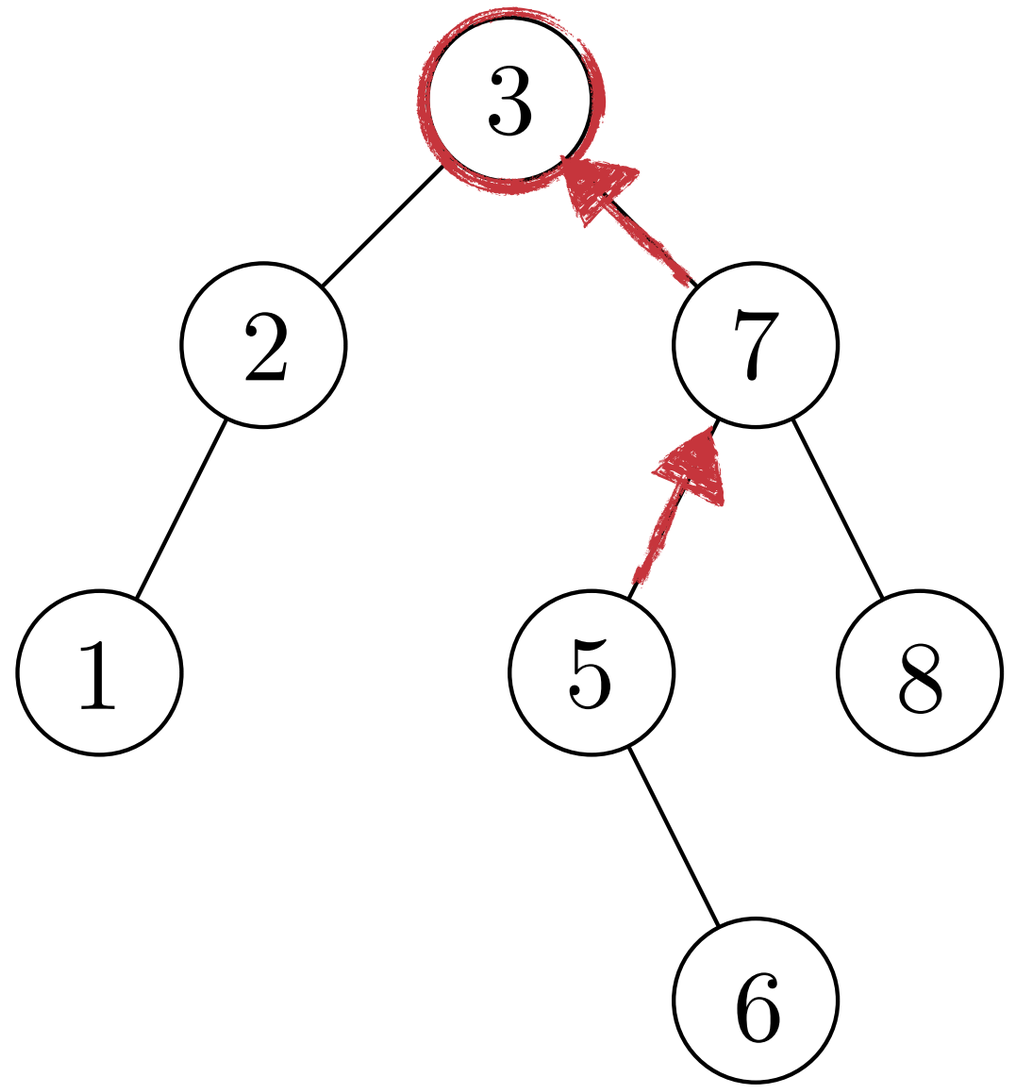
Operation: Minimum, Maximum



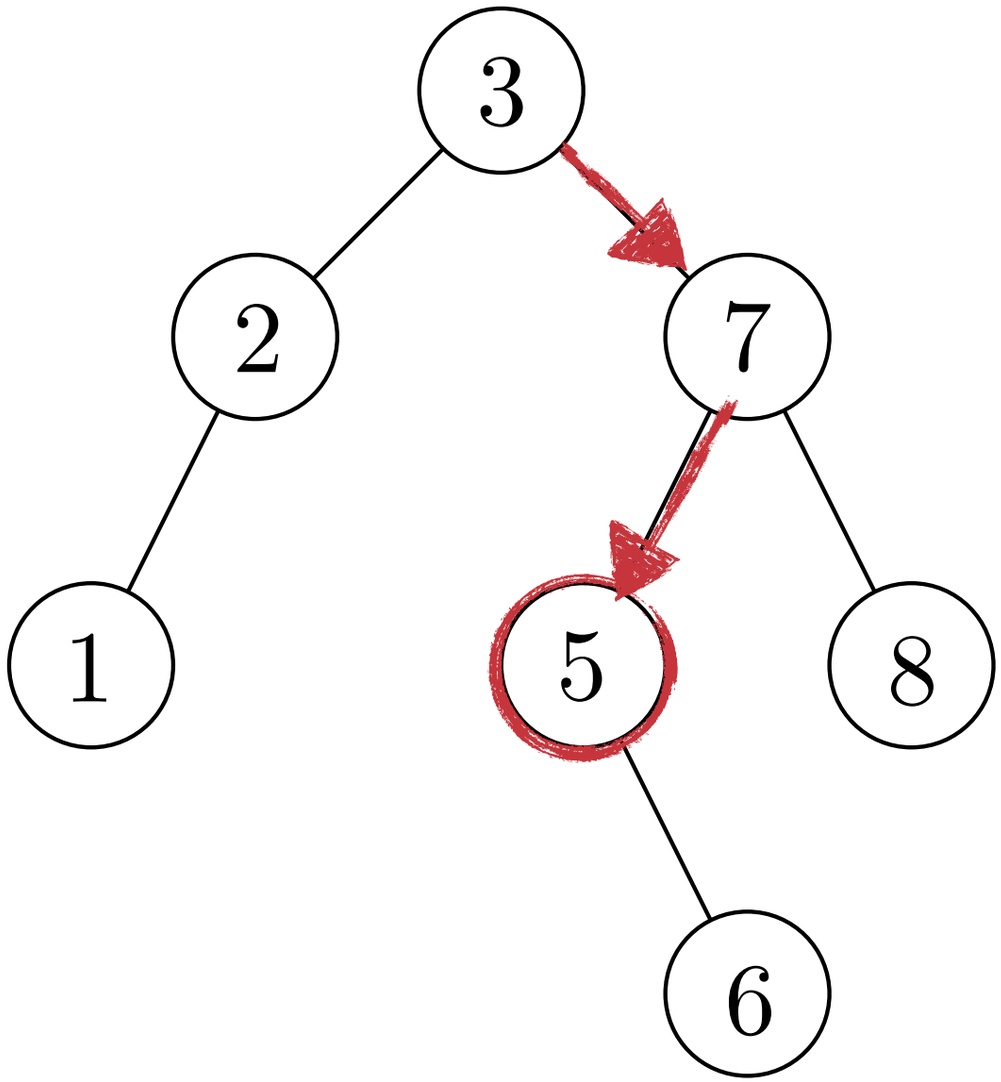


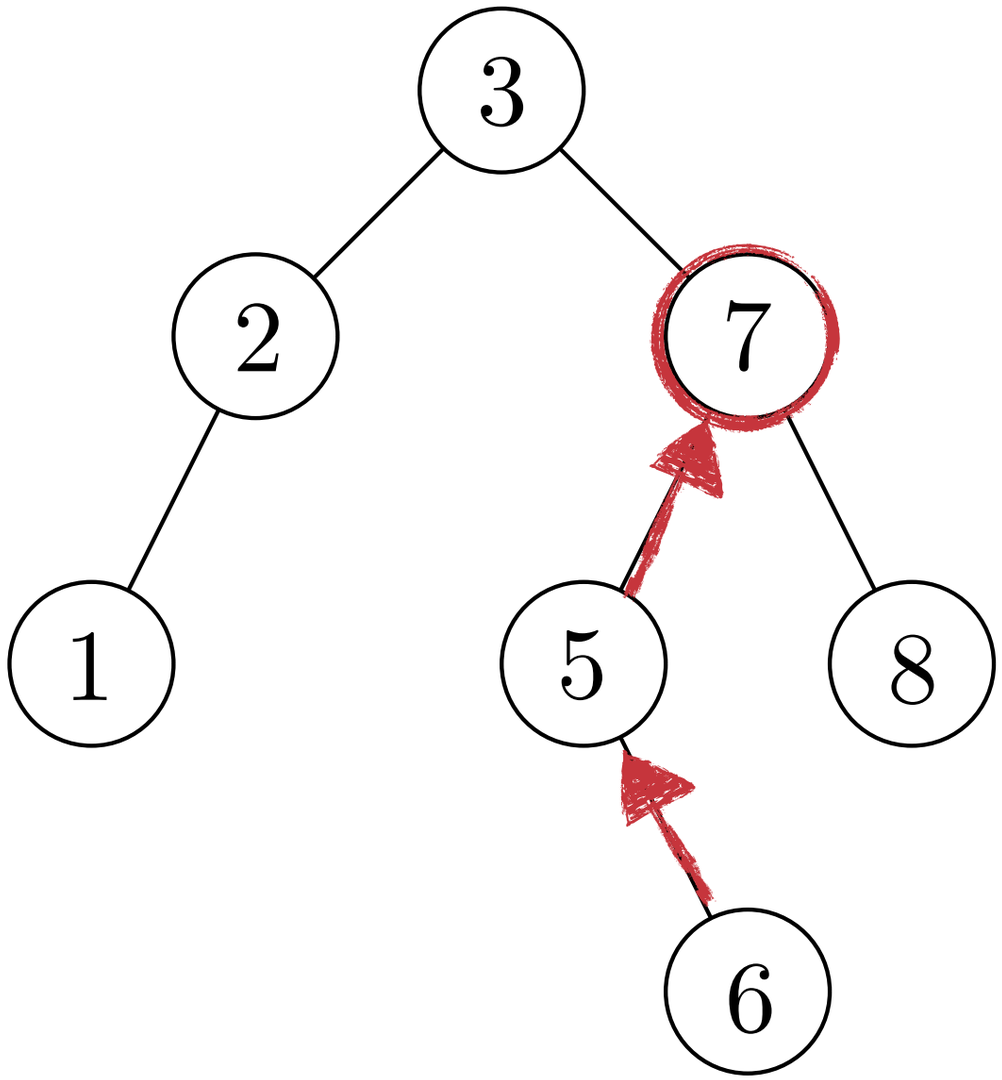
Operation: Predecessor(7), Predecessor(5)



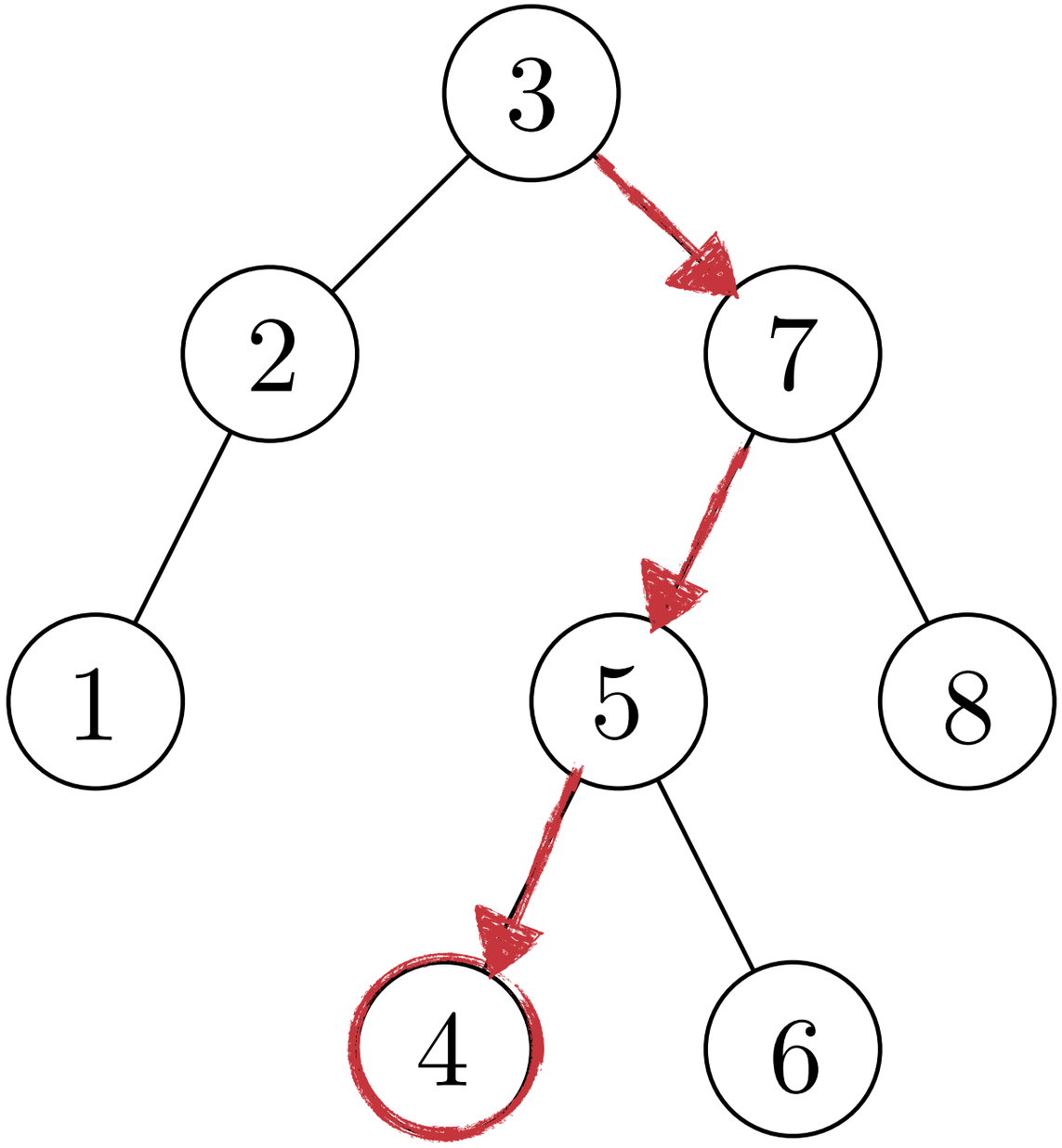


Operation: Successor(3), Successor(6)

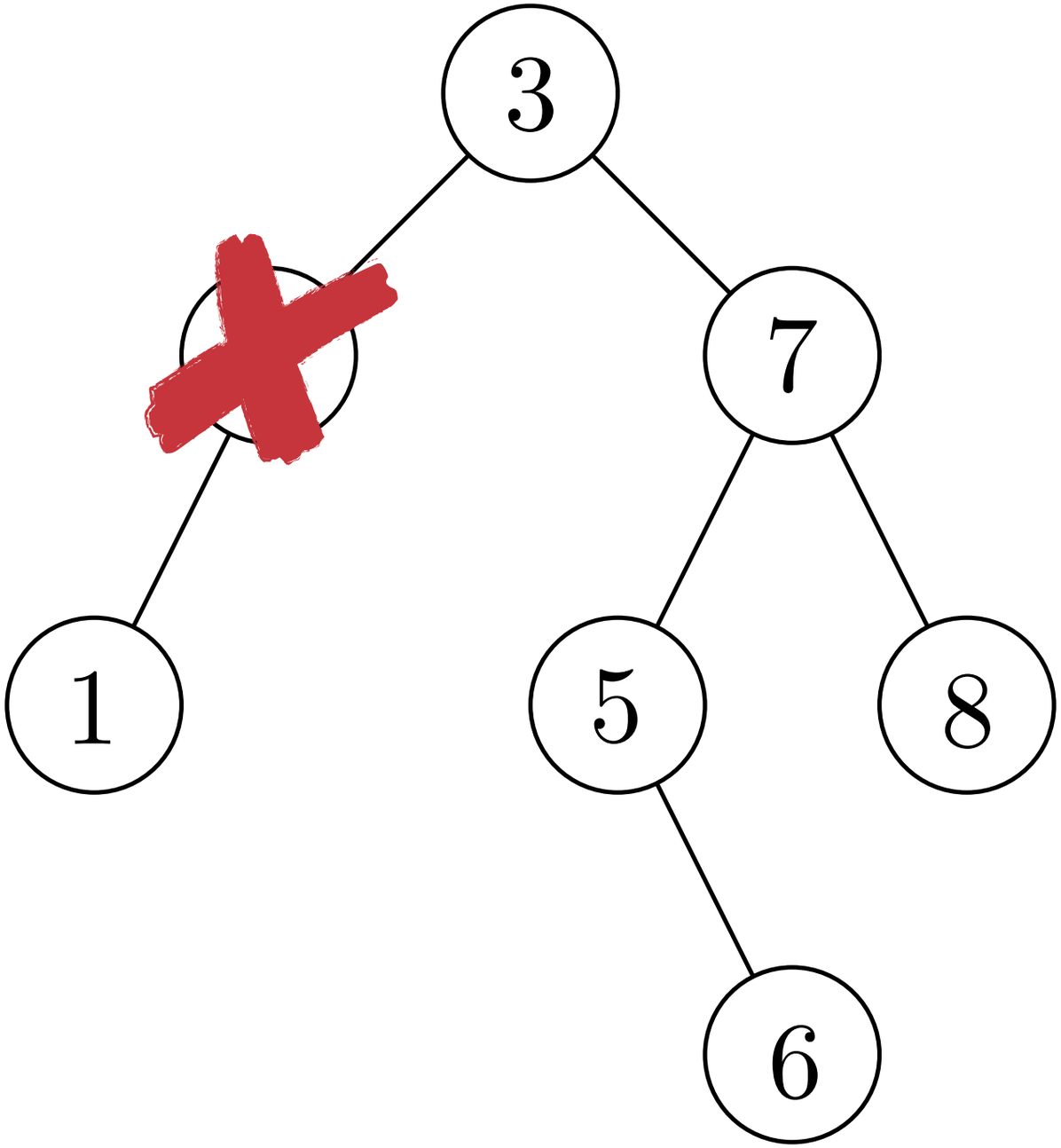




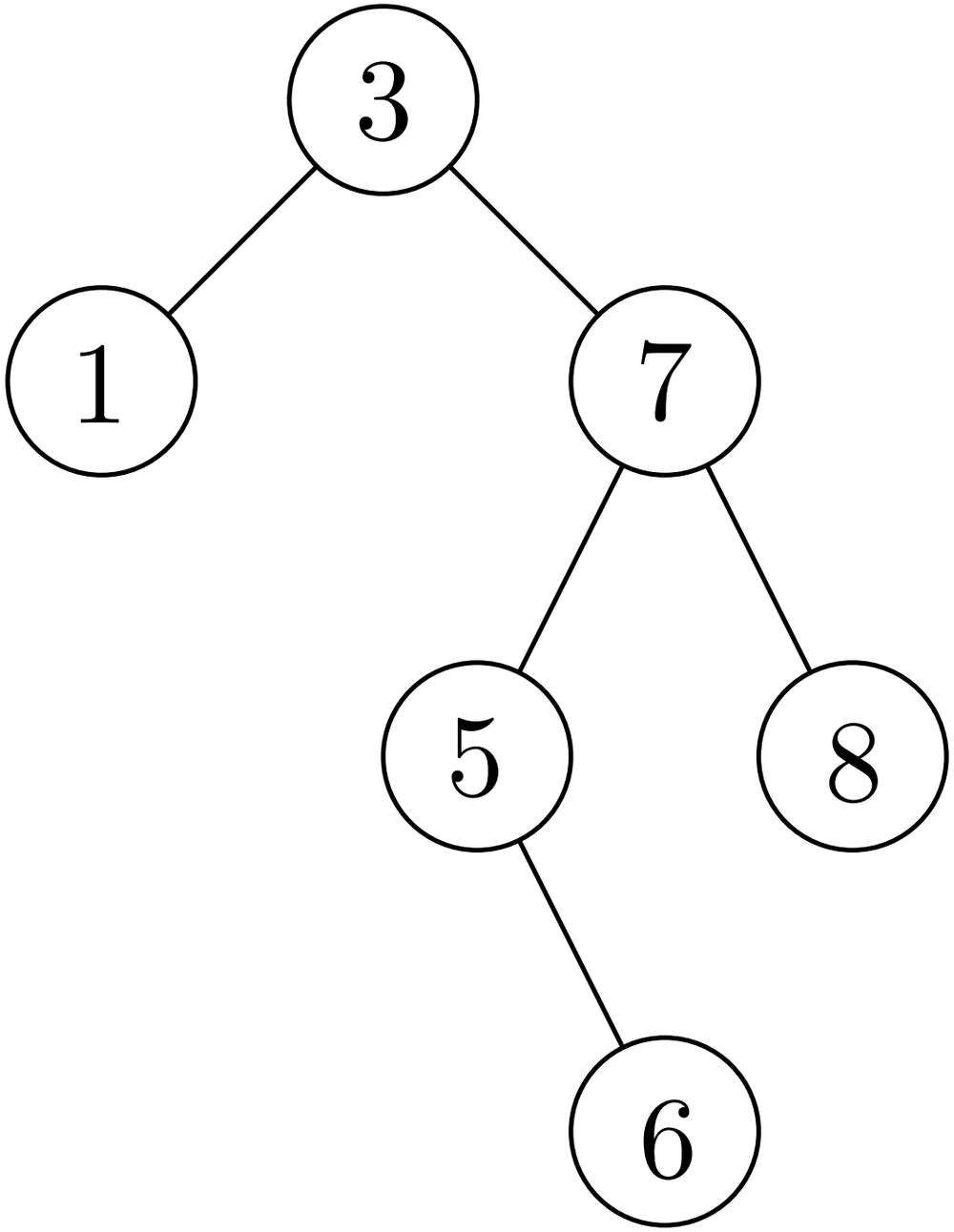
Operation: Insert(4)



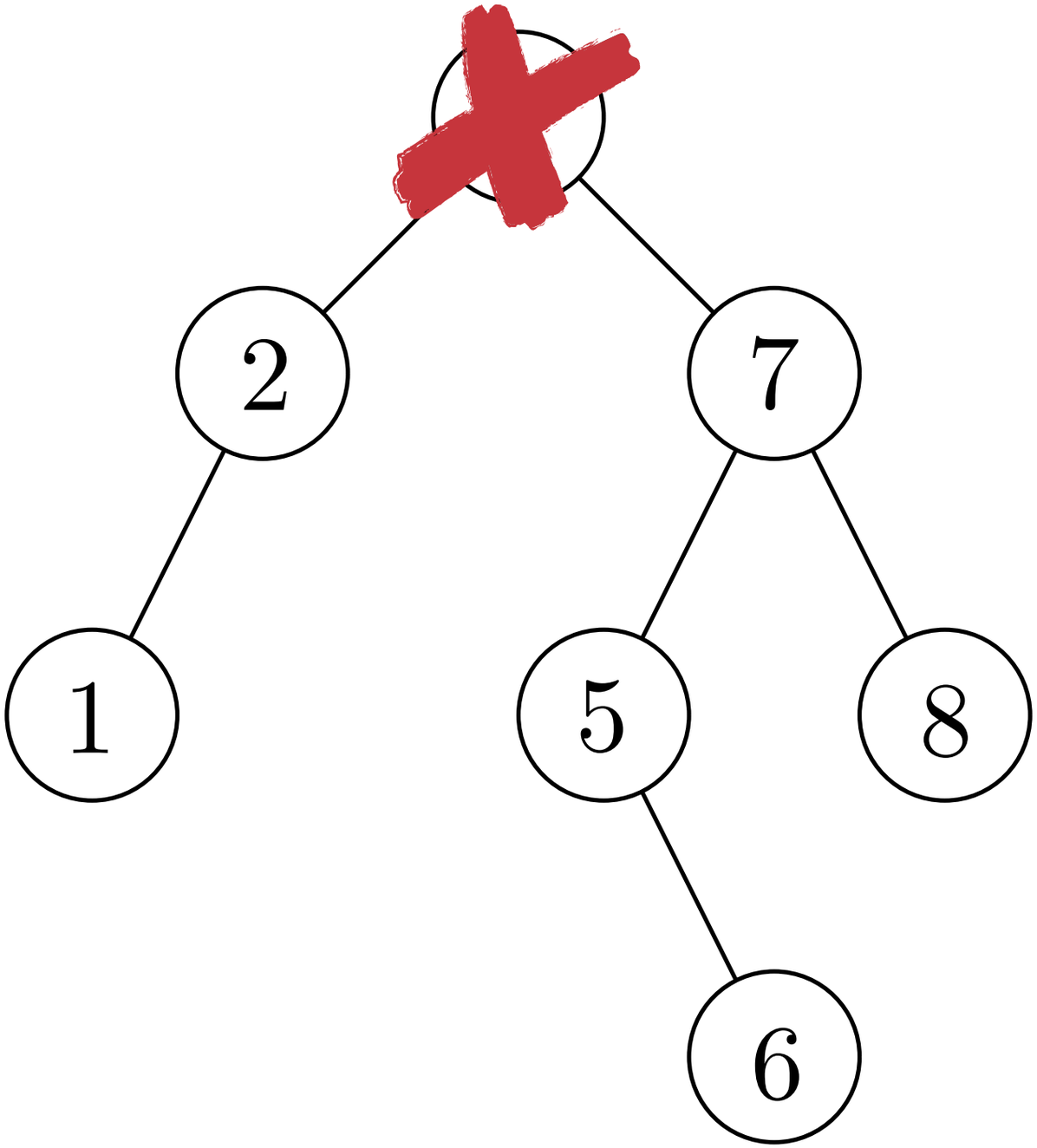
Operation: Delete(2)



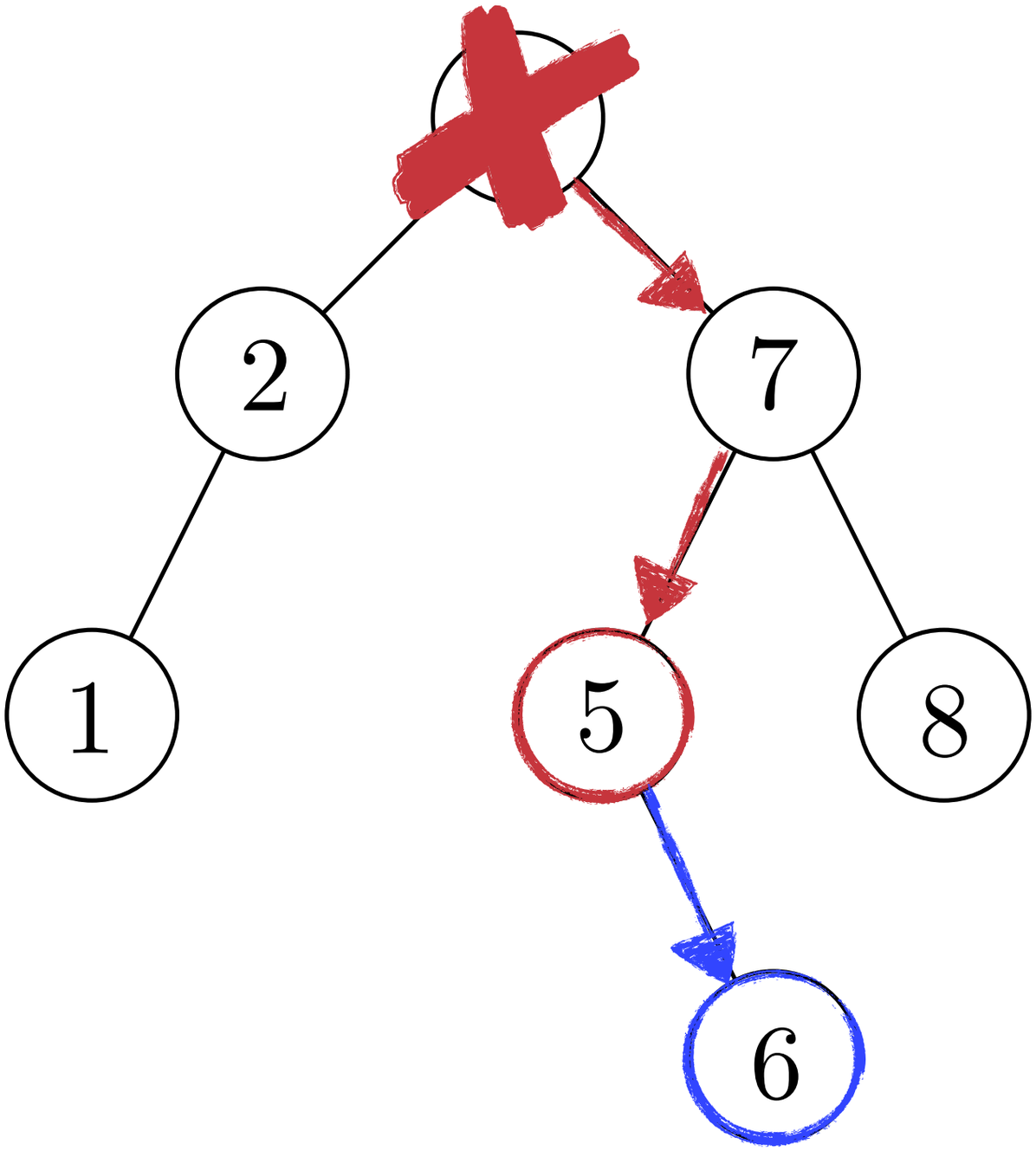
Operation: Delete(2)



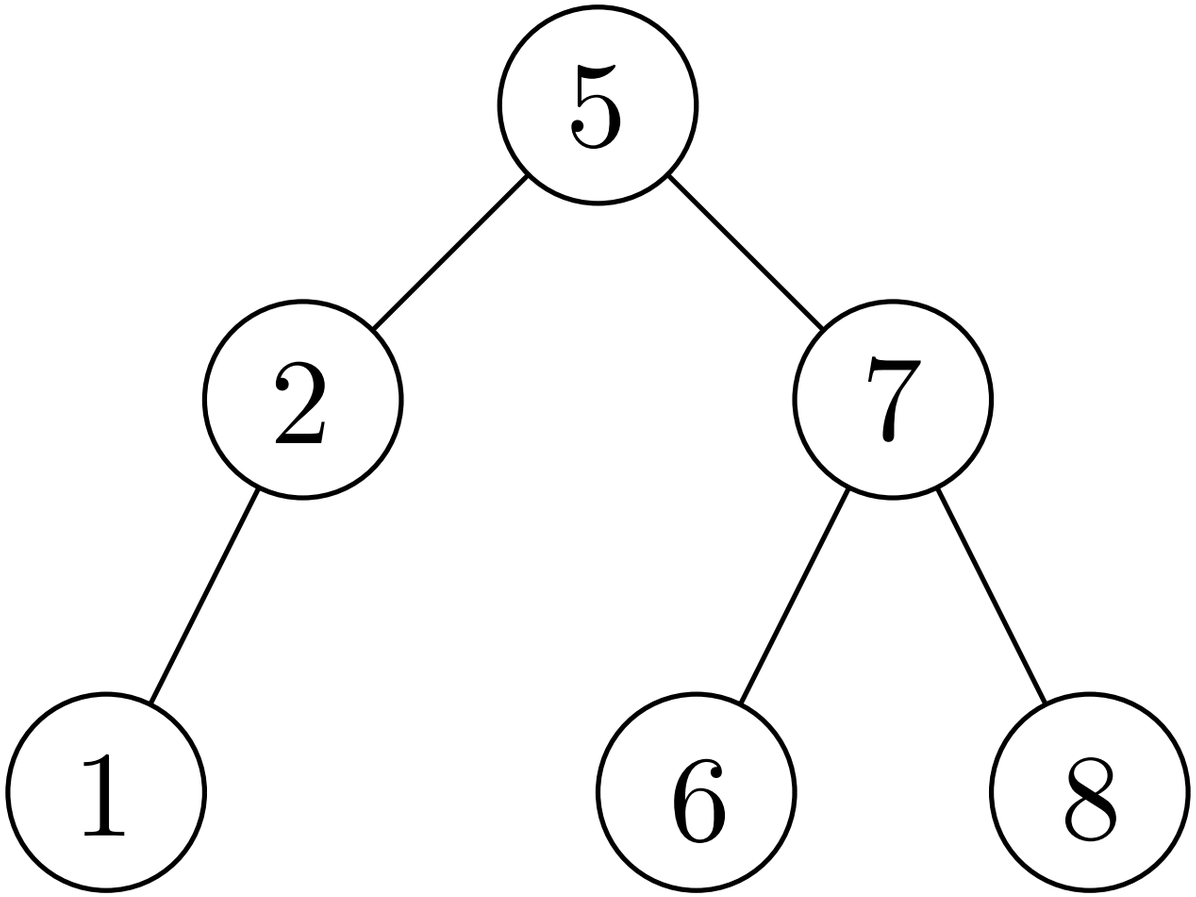
Operation: Delete(3)



Operation: Delete(3)



Operation: Delete(3)

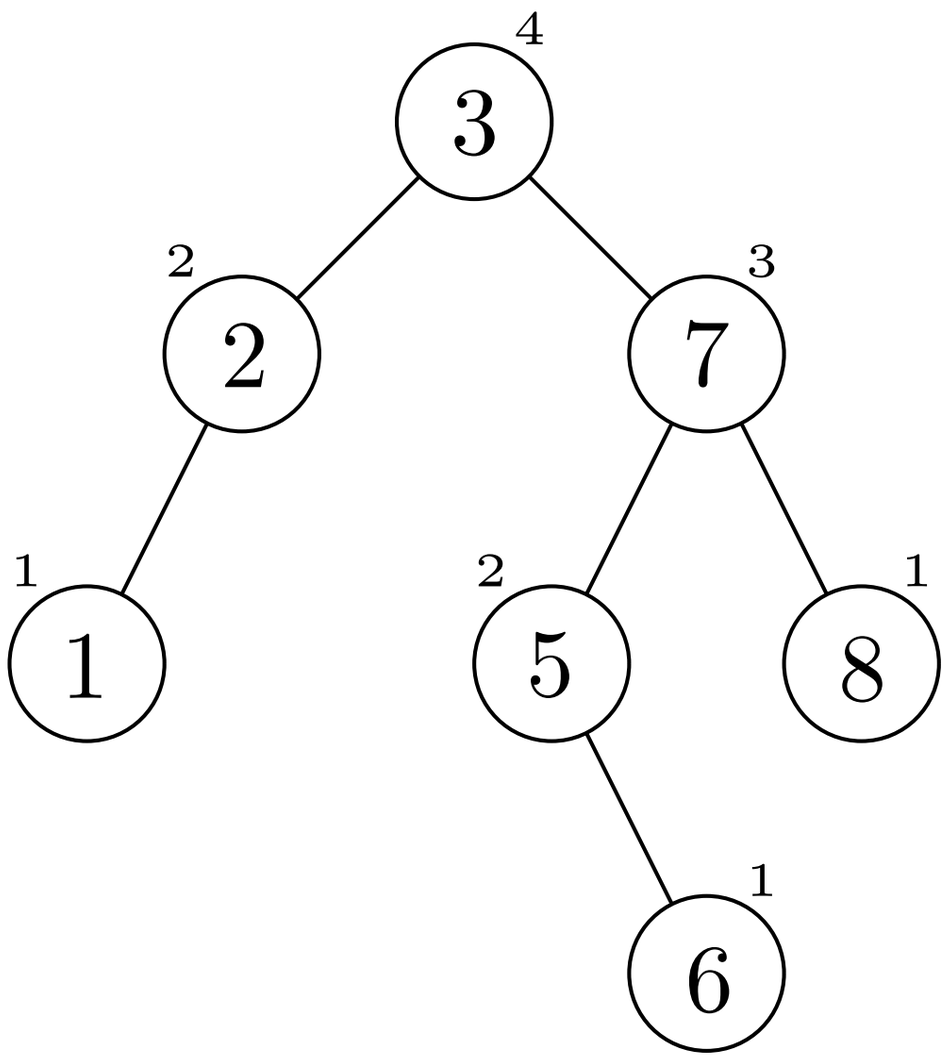


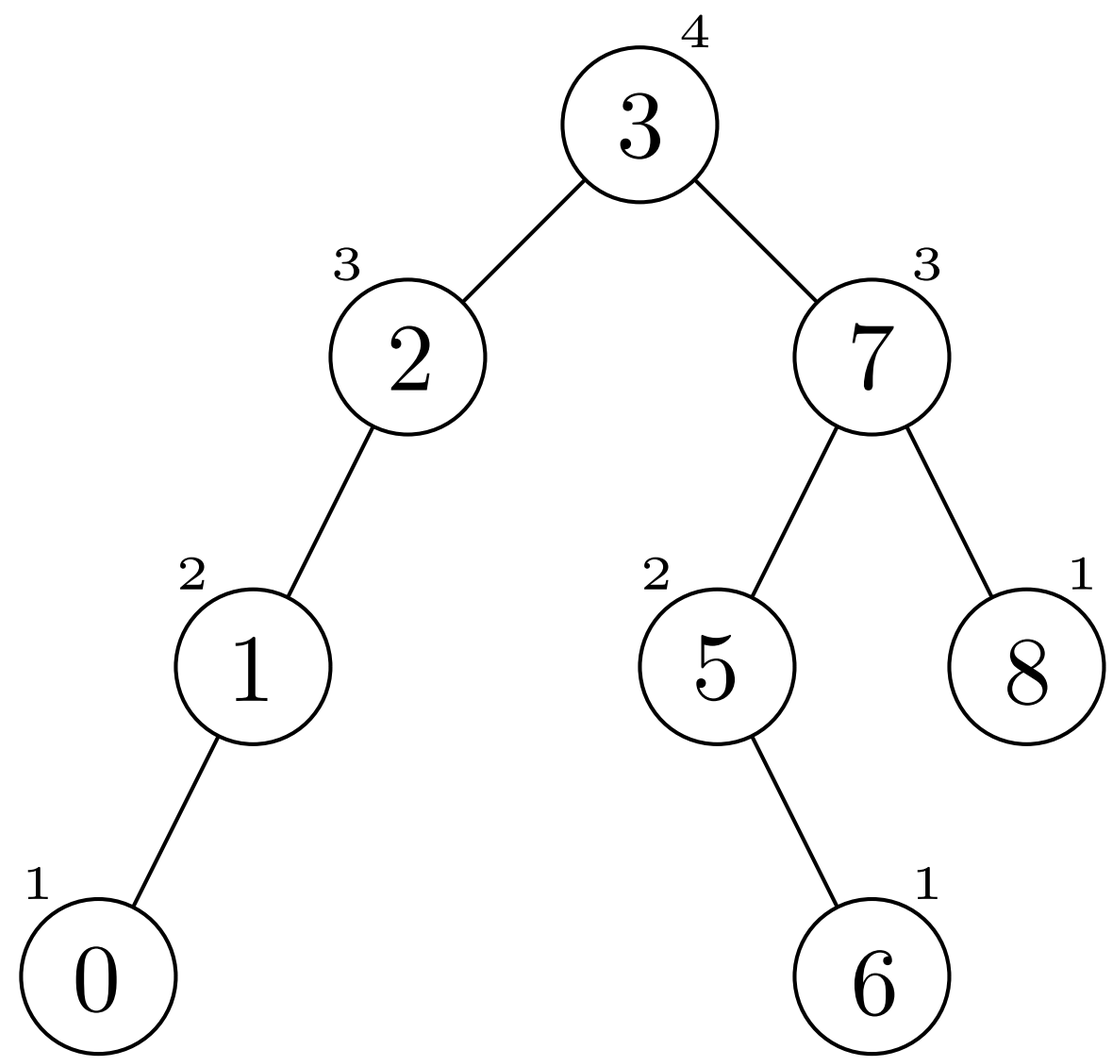
AVL-Bäume

Definition 4.7 (Nach Adelson-Velski und Landis, 1962)

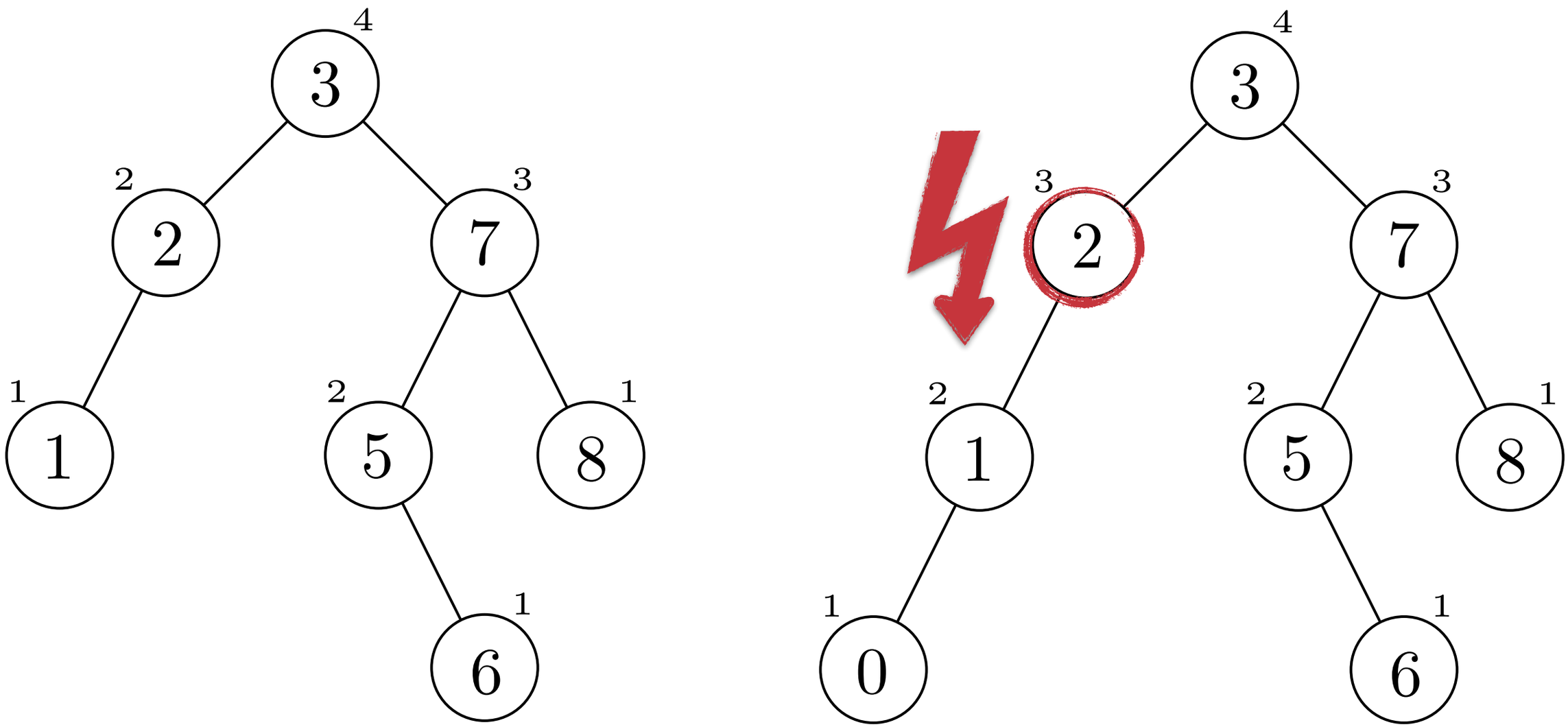
1. Ein binärer Suchbaum ist höhenbalanciert, wenn sich für jeden inneren Knoten v die Höhe der beiden Kinder von v um höchstens 1 unterscheidet.
2. Ein höhenbalancierter Suchbaum heißt auch AVL-Baum.

Operation: Insert(0)





Operation: Insert(0)



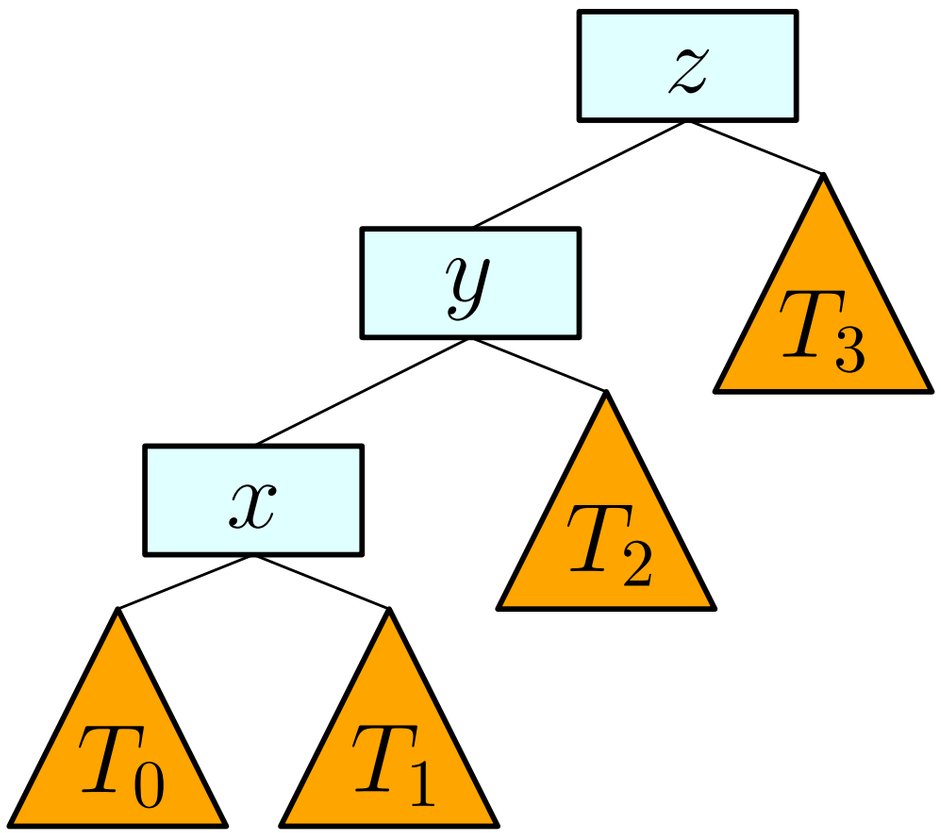
Restructure

Satz

Die Höhe eines AVL-Baumes mit n Knoten ist $\mathcal{O}(\log n)$

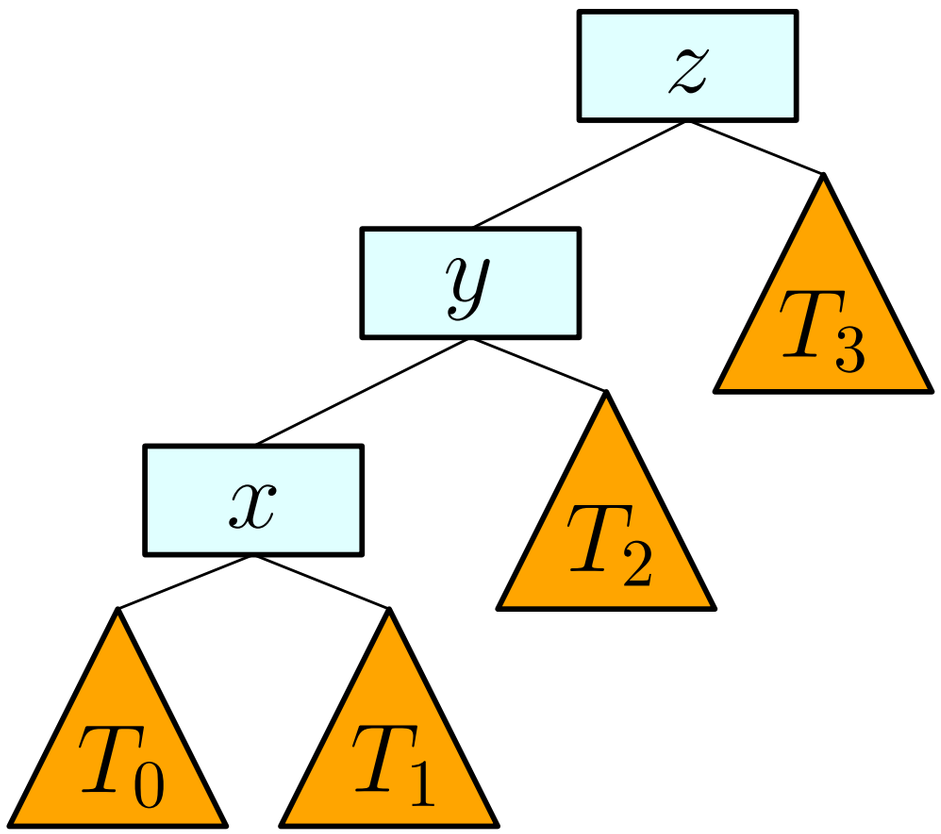
- Wie erhalten wir die AVL-Eigenschaft beim Einfügen und Löschen?
- Lokal restrukturieren!

Restructure

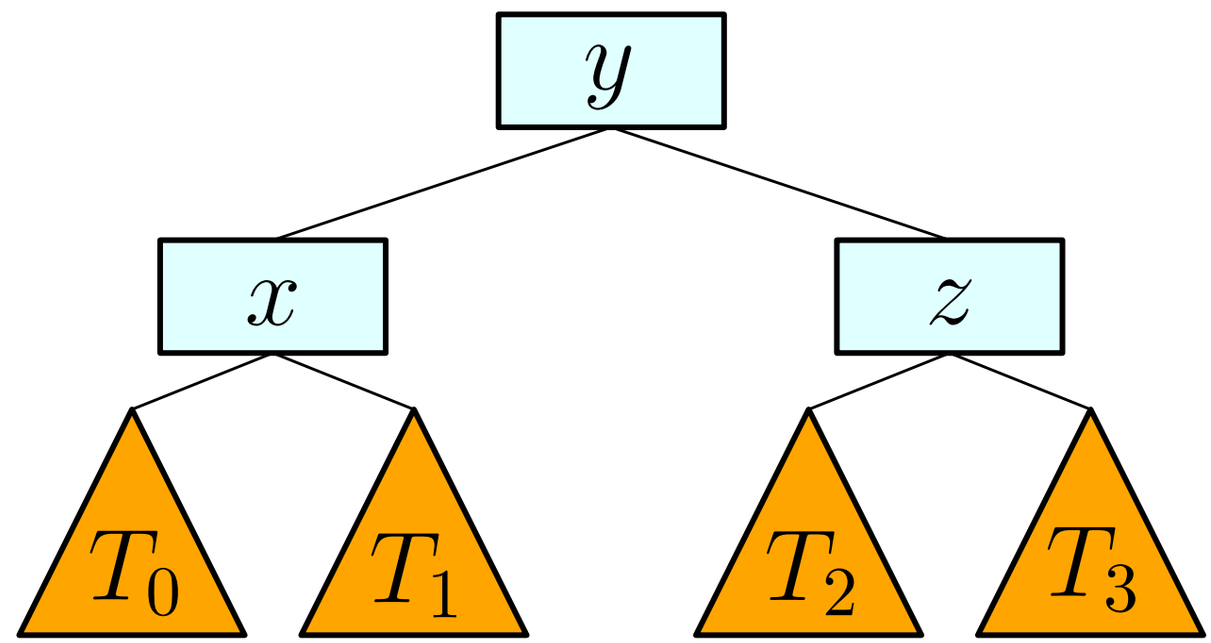


nicht balanciert

Restructure

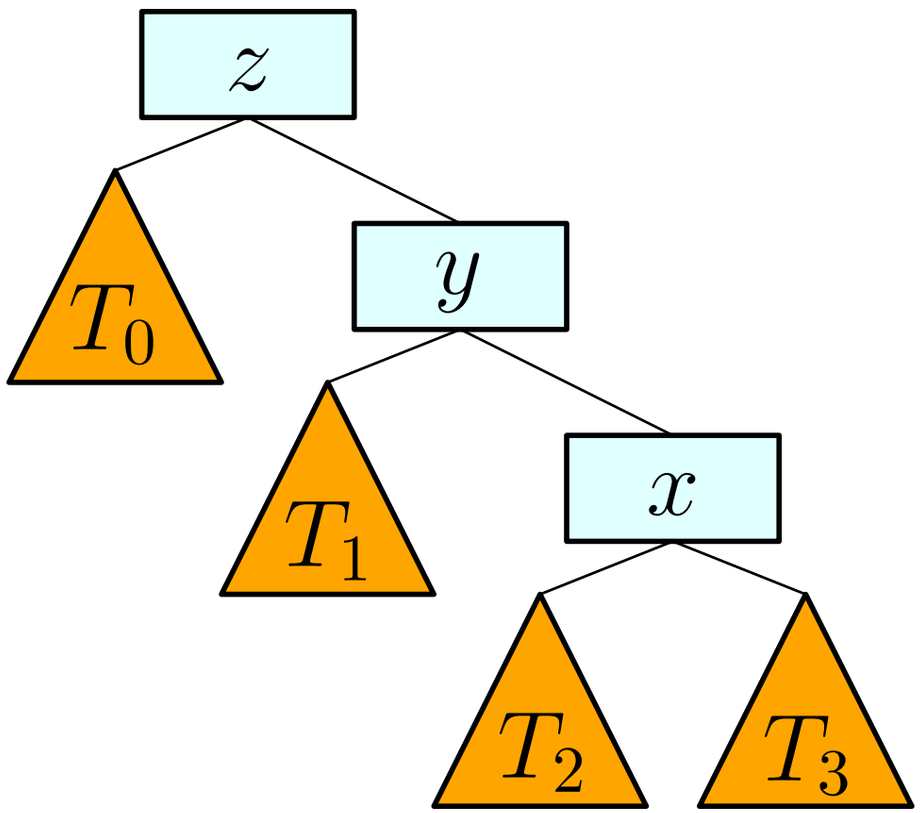


nicht balanciert



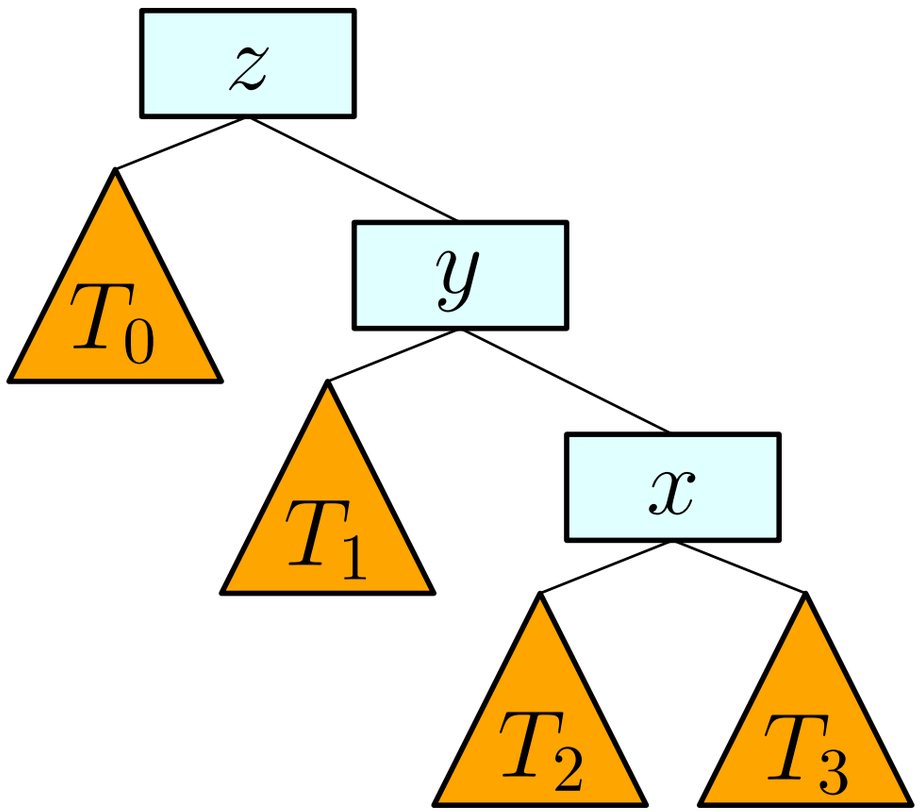
balanciert

Restructure

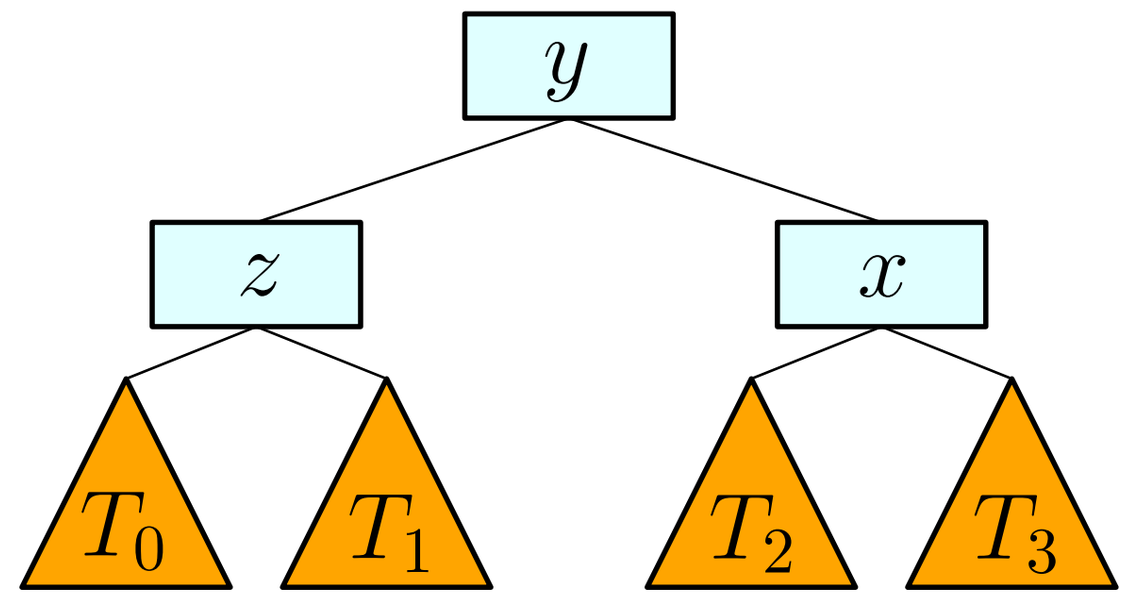


nicht balanciert

Restructure

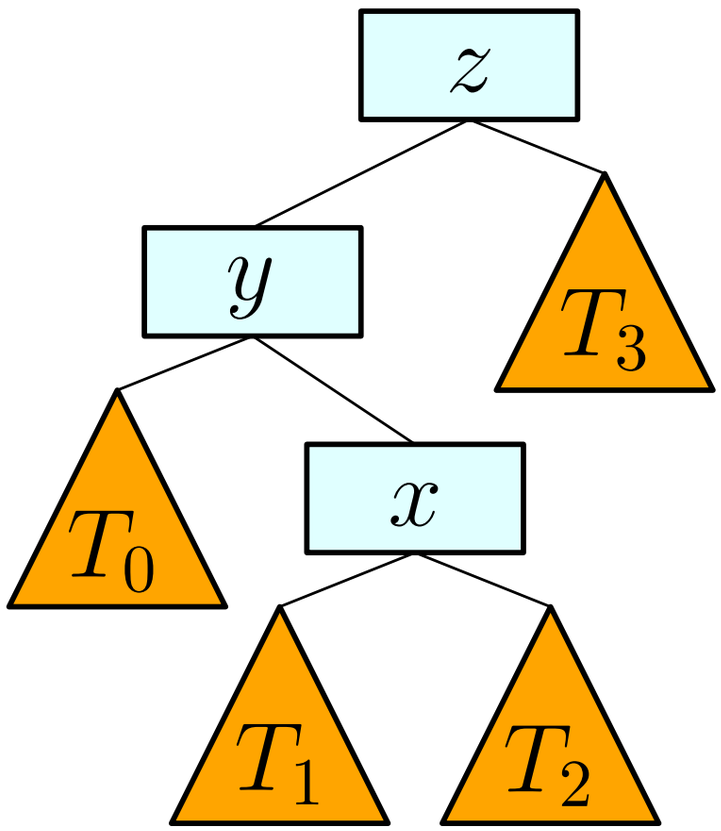


nicht balanciert



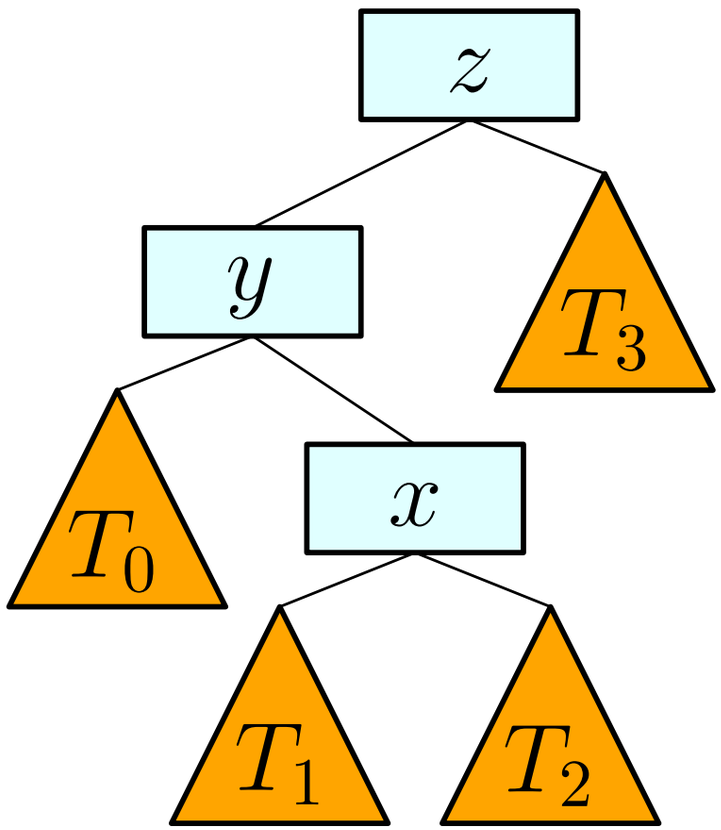
balanciert

Restructure

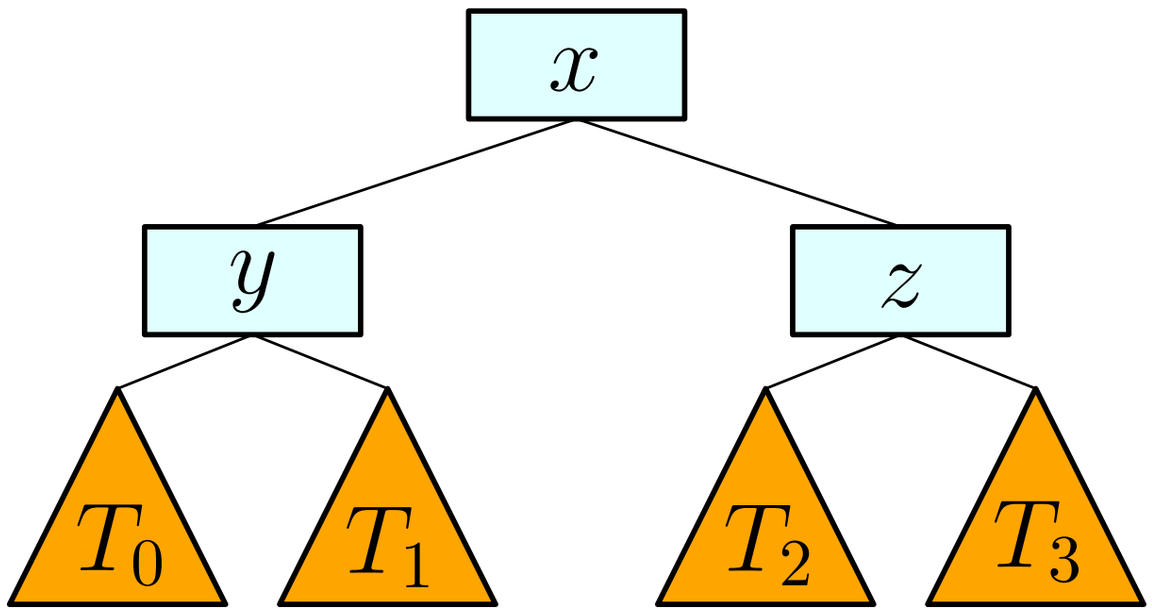


nicht balanciert

Restructure

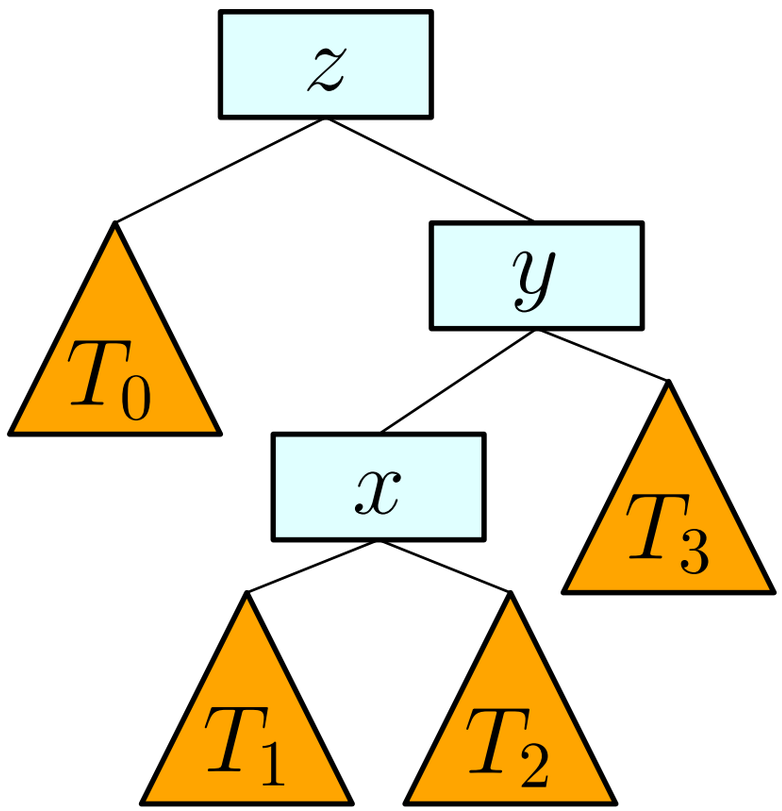


nicht balanciert



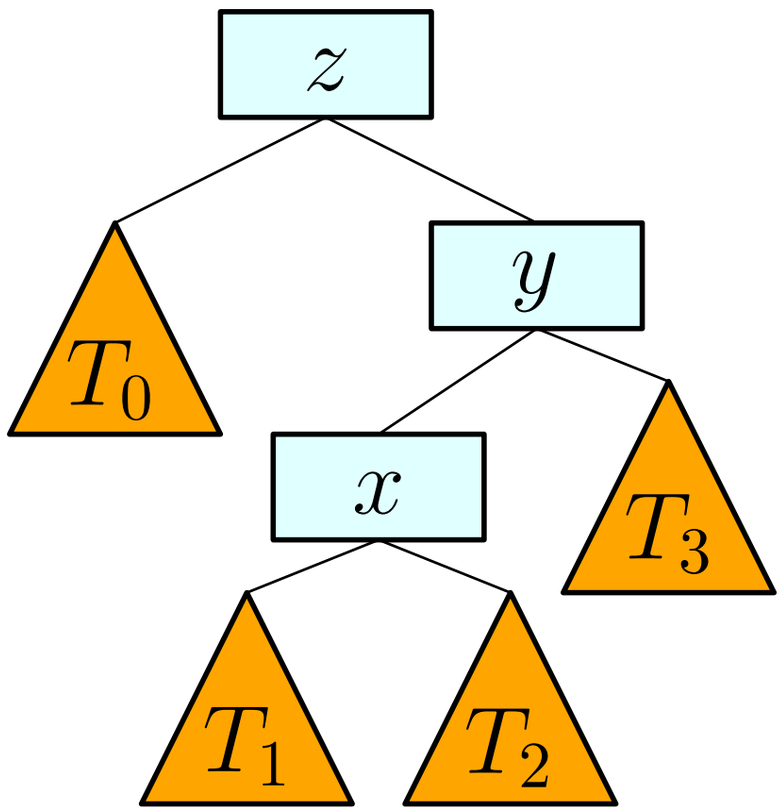
balanciert

Restructure

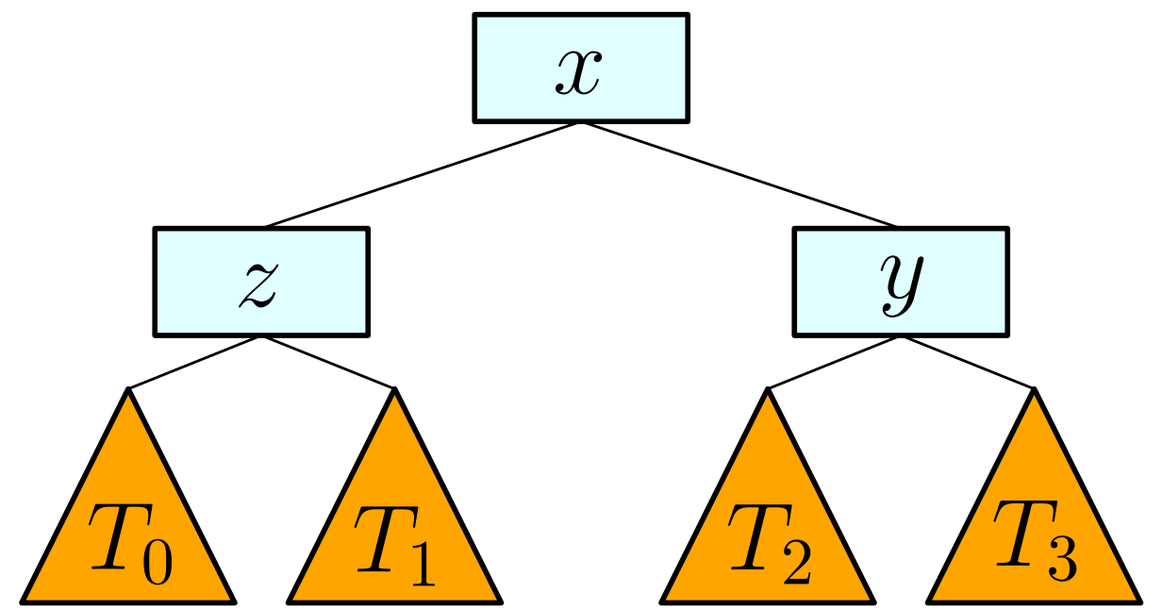


nicht balanciert

Restructure



nicht balanciert

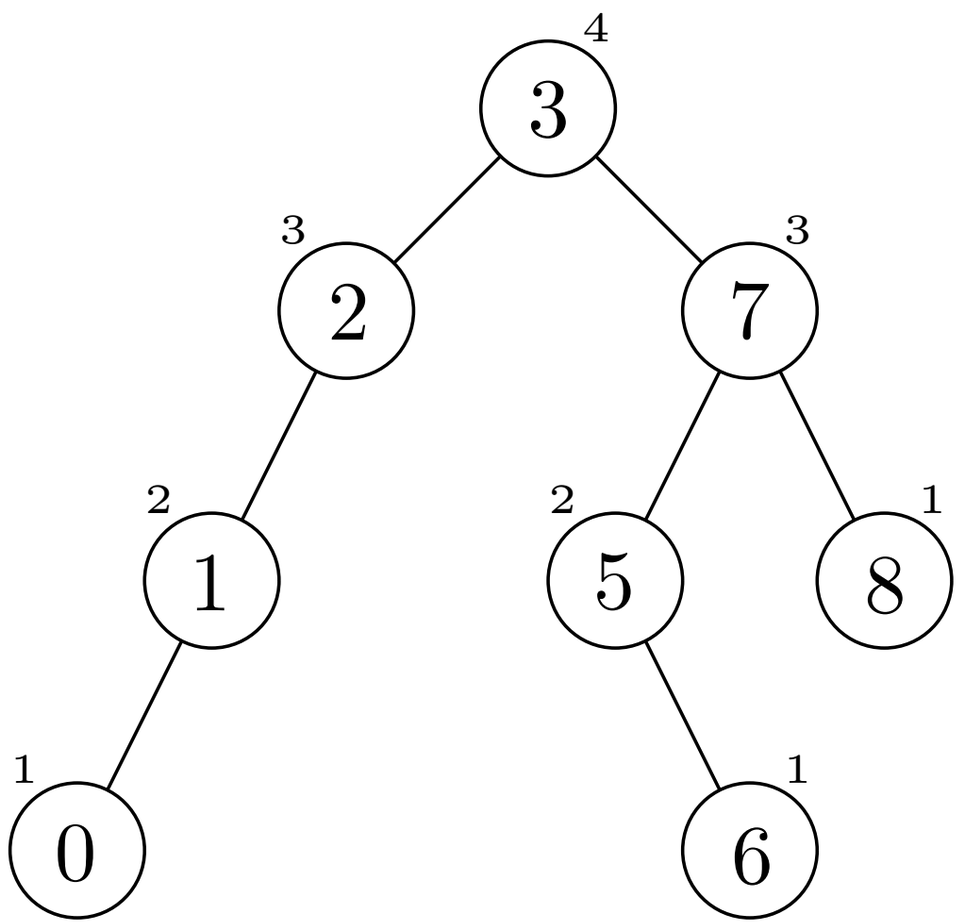


balanciert

Restructure

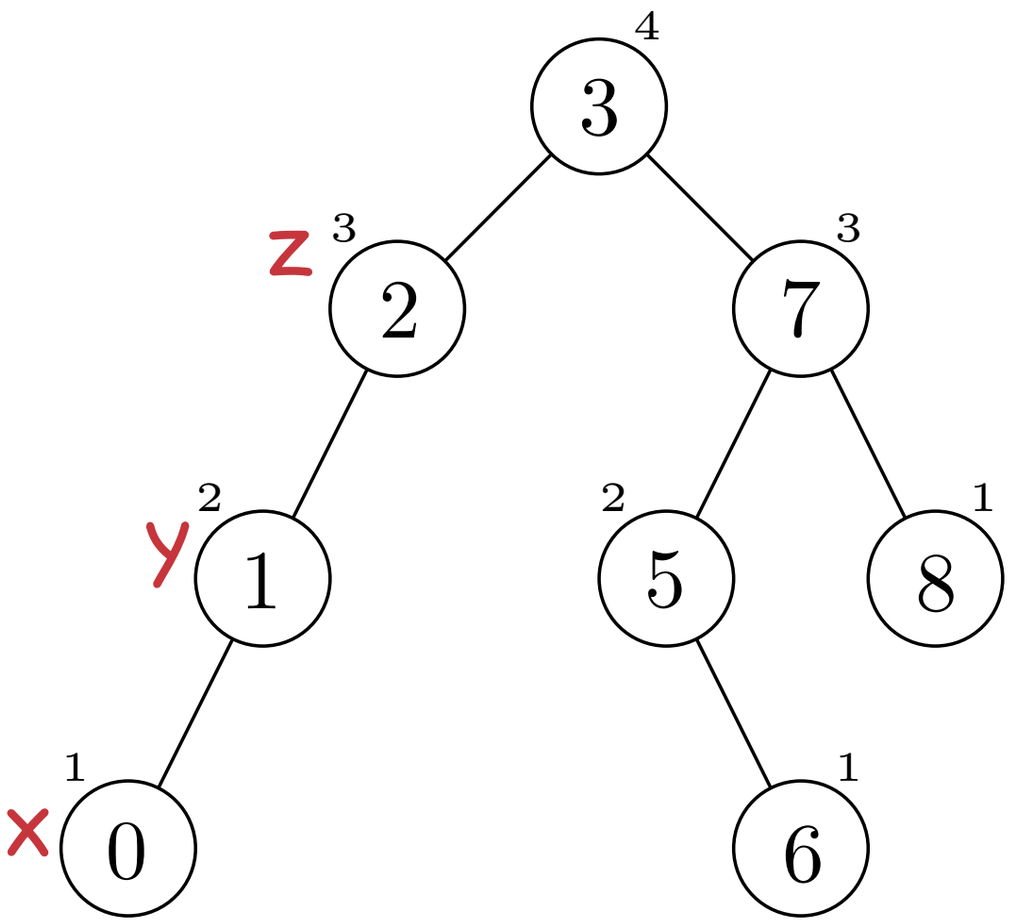
- sei v der eingefügte/entfernte Knoten
- z ist der erste unbalancierte Knoten von v zur Wurzel
- y ist das höhere Kind von z
- x ist das höhere Kind von y (falls gleich hoch: beliebig)
- sei a ≤ b ≤ c die Größensortierung von x,y,z und T_0, T_1, T_2, T_3 die Größensortierung der Teilbäume an den übrigen Kindern von x,y,z
- Dann: Ersetze z durch b mit linkem Kind a, rechtem Kind c und hänge die Teilbäume T_0 und T_1 als linkes/rechtes Kind an a und T_2 und T_3 als linkes/rechtes Kind an c

Insert(0) - Restructure



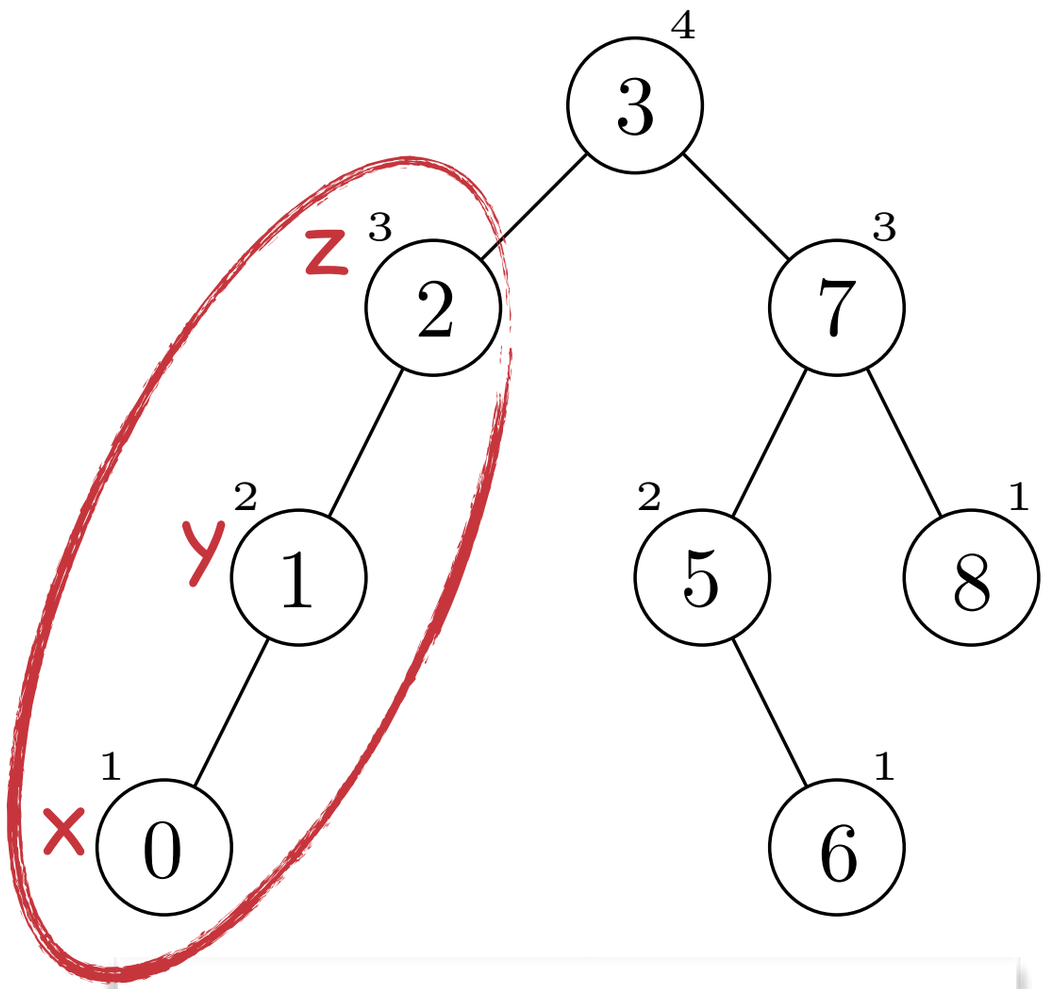
nicht balanciert

Insert(0) - Restructure

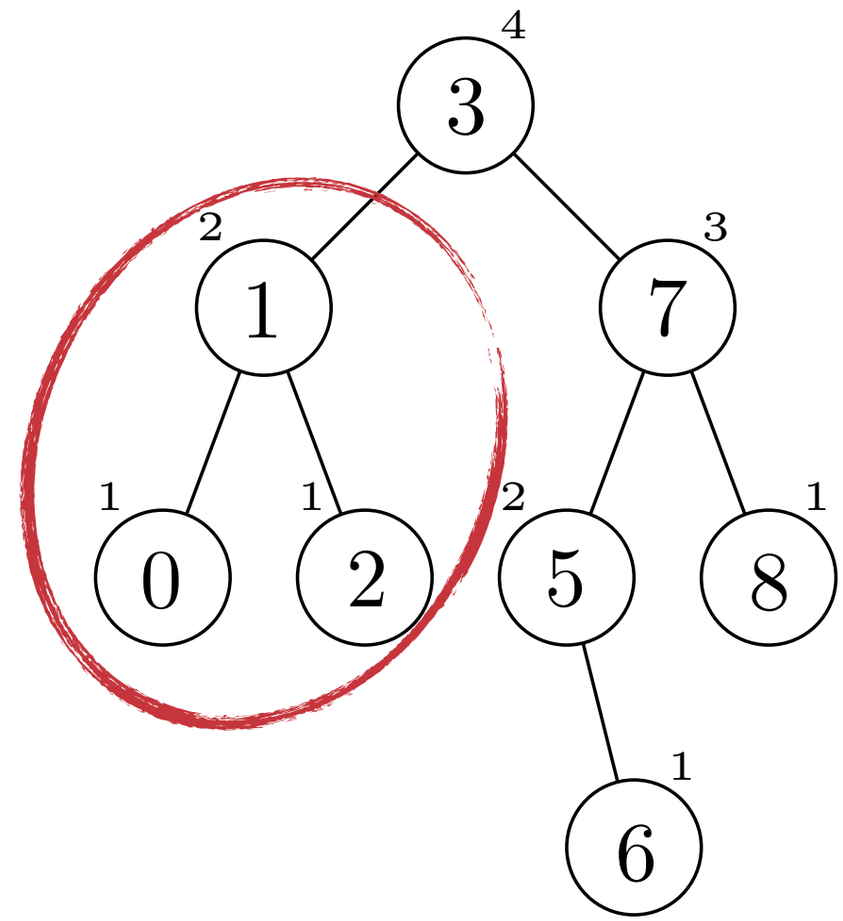


nicht balanciert

Insert(0) - Restructure

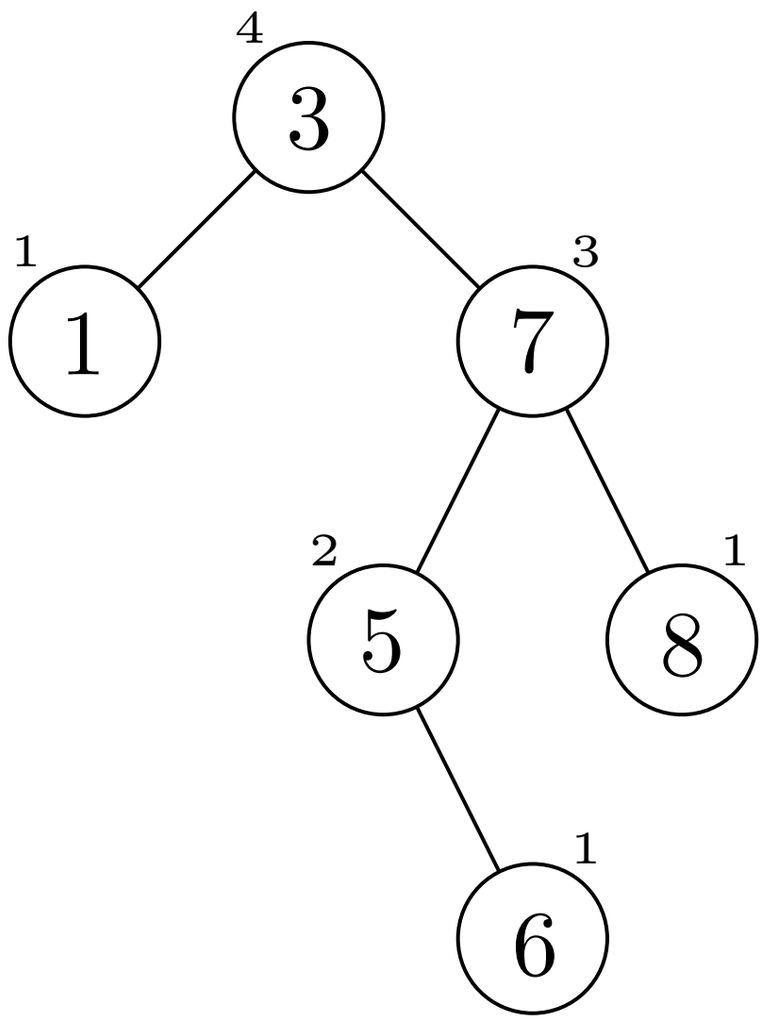


nicht balanciert



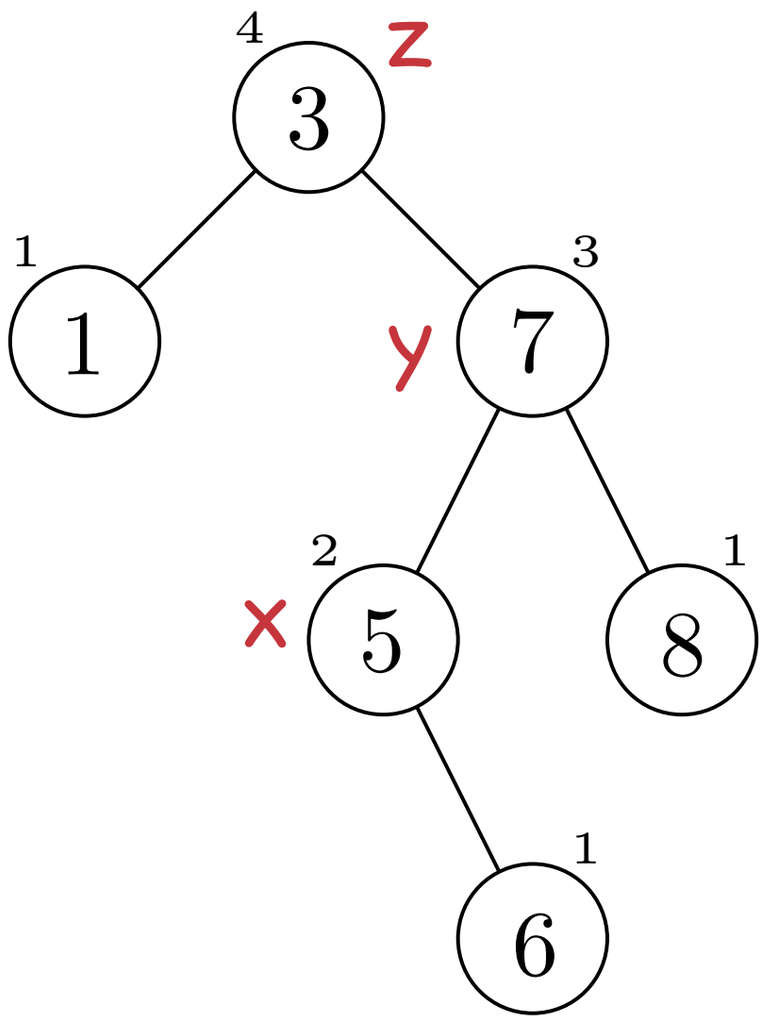
balanciert

Delete(2) - Restructure



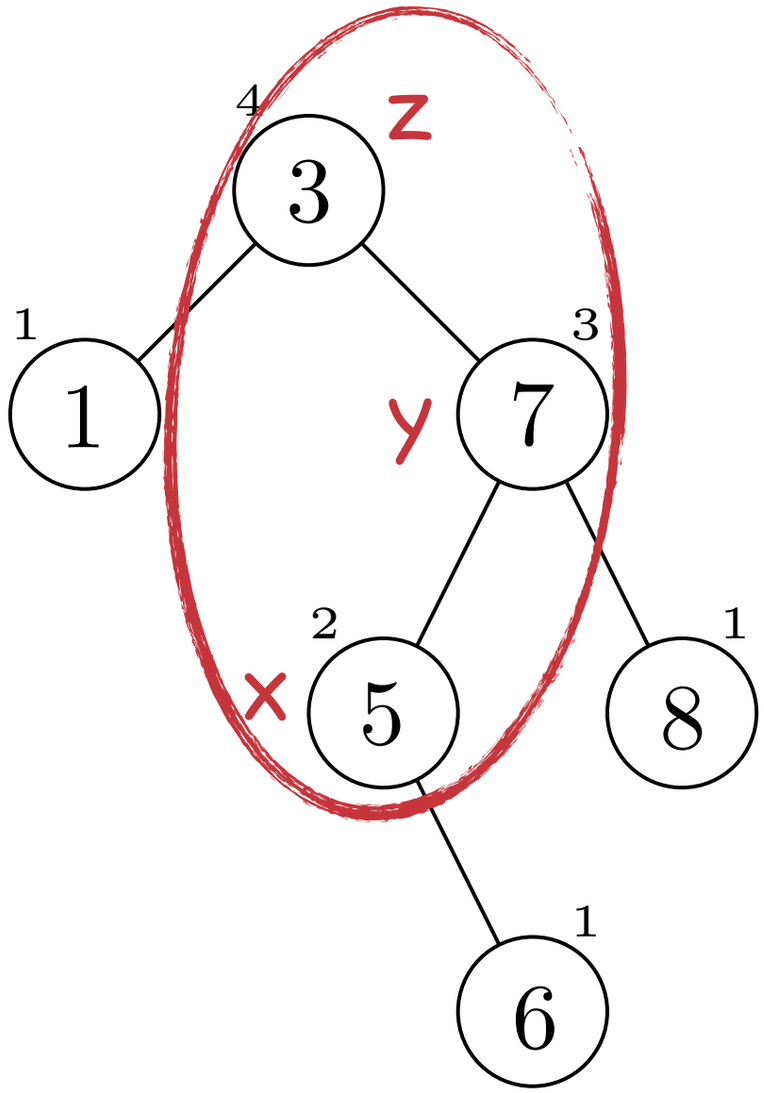
nicht balanciert

Delete(2) - Restructure

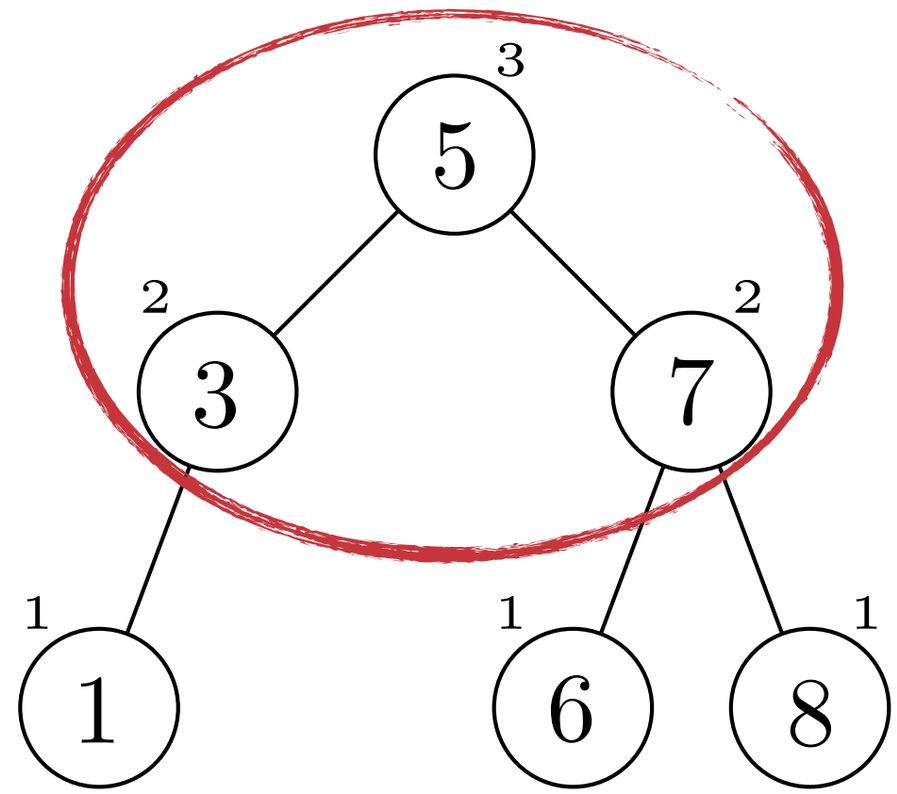


nicht balanciert

Delete(2) - Restructure



nicht balanciert



balanciert

Fragen?

Weitere Beispiele an der Tafel!