Technische Universität Braunschweig

Kapitel 4.8 - 4.11
Andere dynamische Datenstrukturen

Algorithmen und Datenstrukturen
Wintersemester 2021/22

Sándor Fekete, Matthias Konitzny, Arne Schmidt

Agenda

Agenda

- Wiederholung: Binäre Suchbäume / AVL-Bäume

Agenda

- Wiederholung: Binäre Suchbäume / AVL-Bäume
- 4.8 Rot-Schwarz-Bäume

Agenda

- Wiederholung: Binäre Suchbäume / AVL-Bäume
- 4.8 Rot-Schwarz-Bäume
- 4.9 B-Bäume

Agenda

- Wiederholung: Binäre Suchbäume / AVL-Bäume
- 4.8 Rot-Schwarz-Bäume
- 4.9 B-Bäume
- 4.10 Heaps

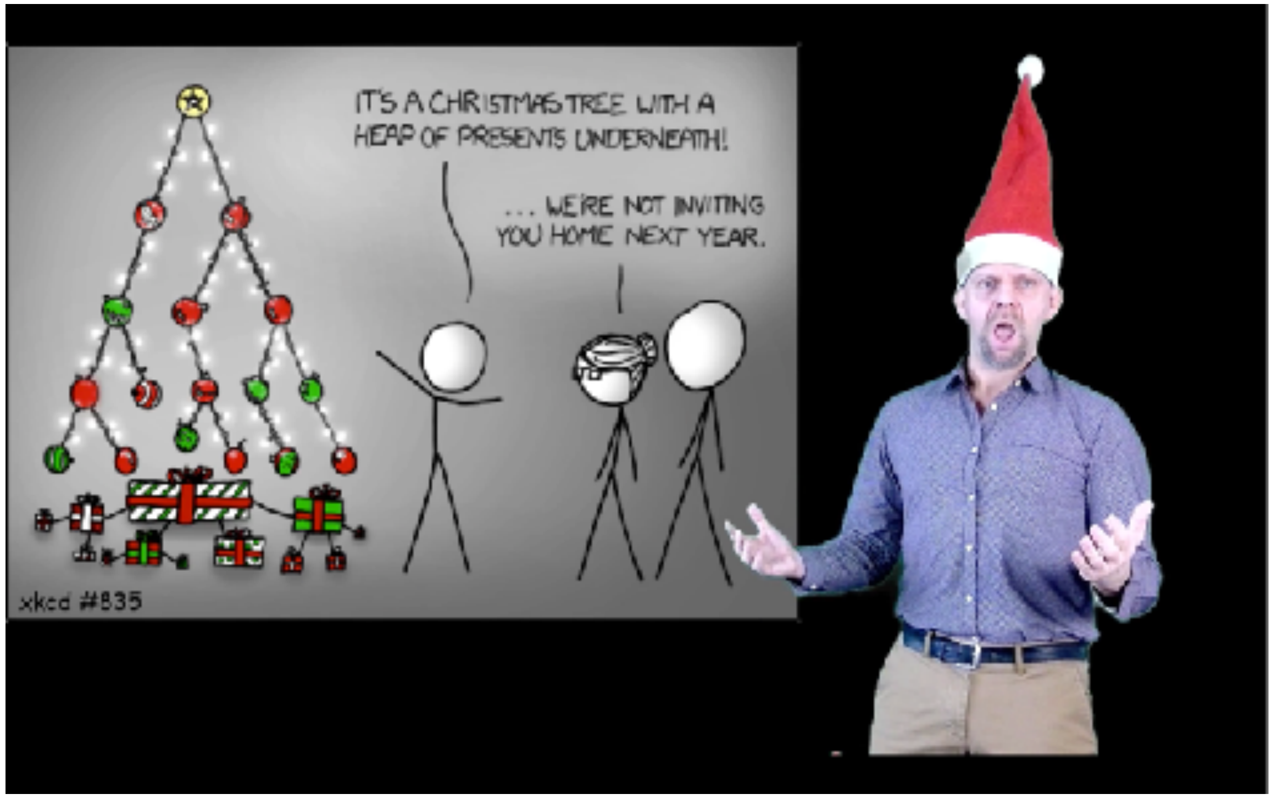
Agenda

- Wiederholung: Binäre Suchbäume / AVL-Bäume
- 4.8 Rot-Schwarz-Bäume
- 4.9 B-Bäume
- 4.10 Heaps
- 4.11 Andere Strukturen

Wiederholung: Binäre Bäume

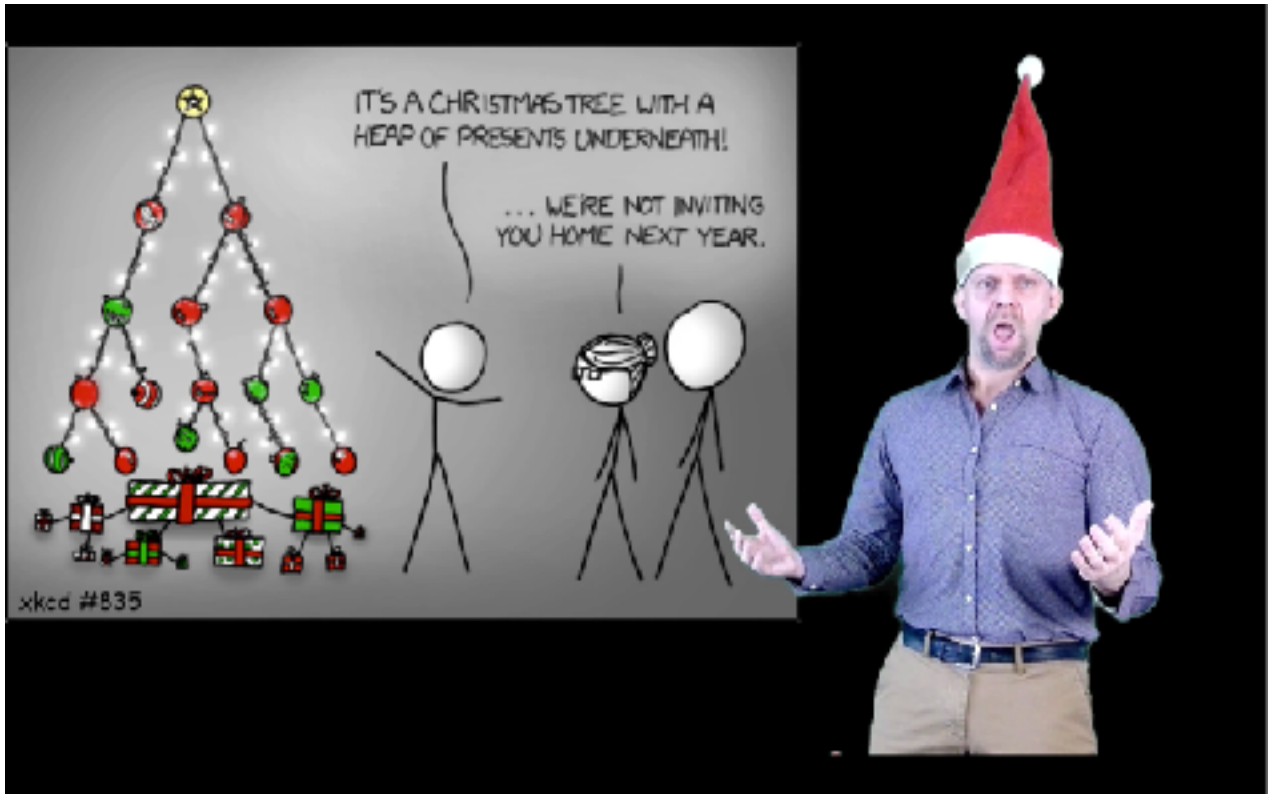
Binäre Bäume (Wiederholung)

Binäre Bäume (Wiederholung)



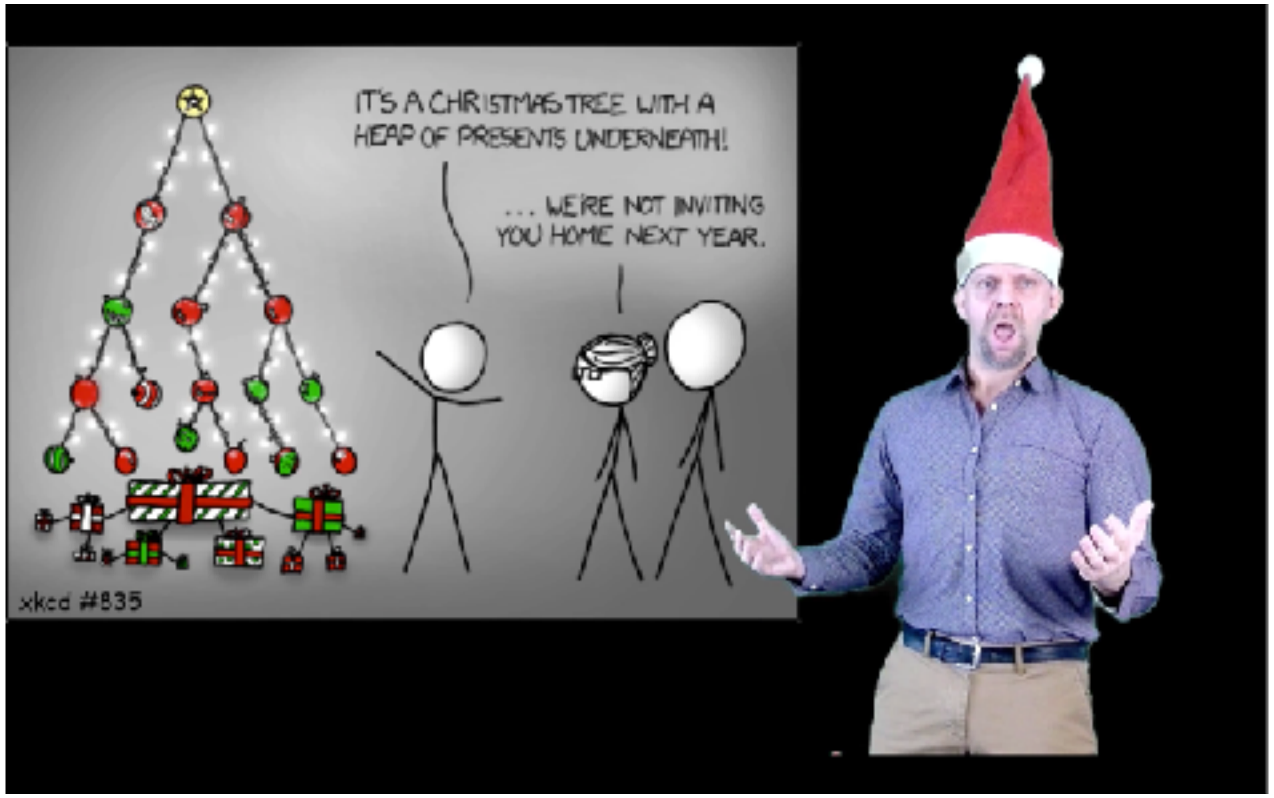
Binäre Bäume (Wiederholung)

Binäre Bäume sind gewurzelte und gerichtete Bäume.



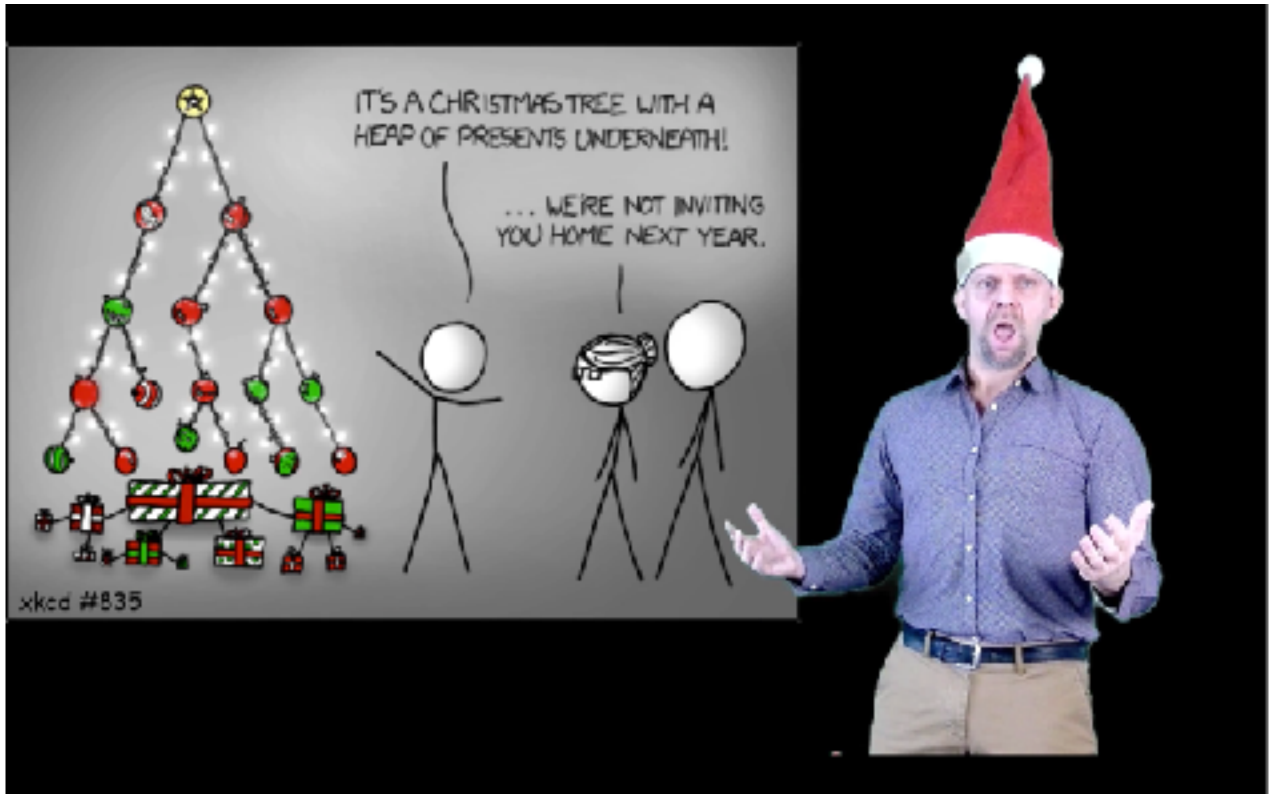
Binäre Bäume (Wiederholung)

Binäre Bäume sind gewurzelte und gerichtete Bäume.
Jeder Knoten hat kein, ein oder zwei Kind(er).



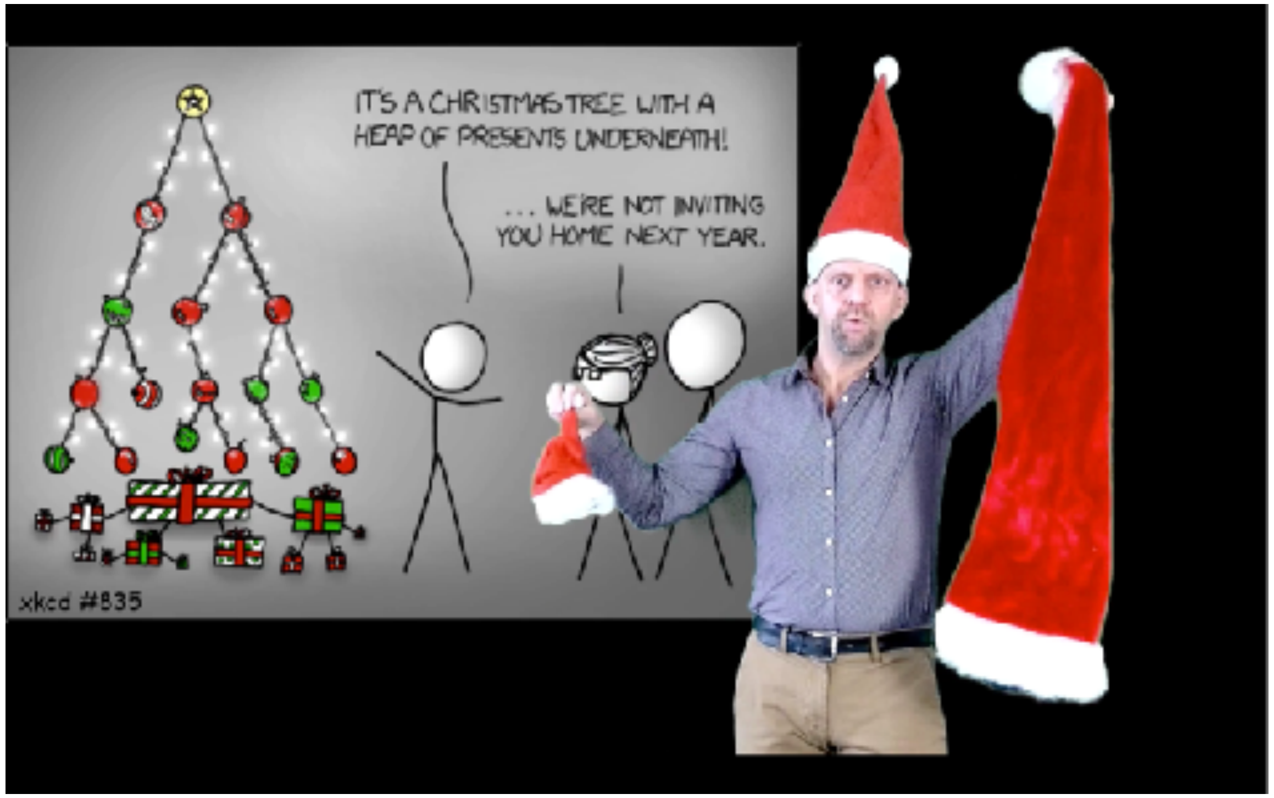
Binäre Bäume (Wiederholung)

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.



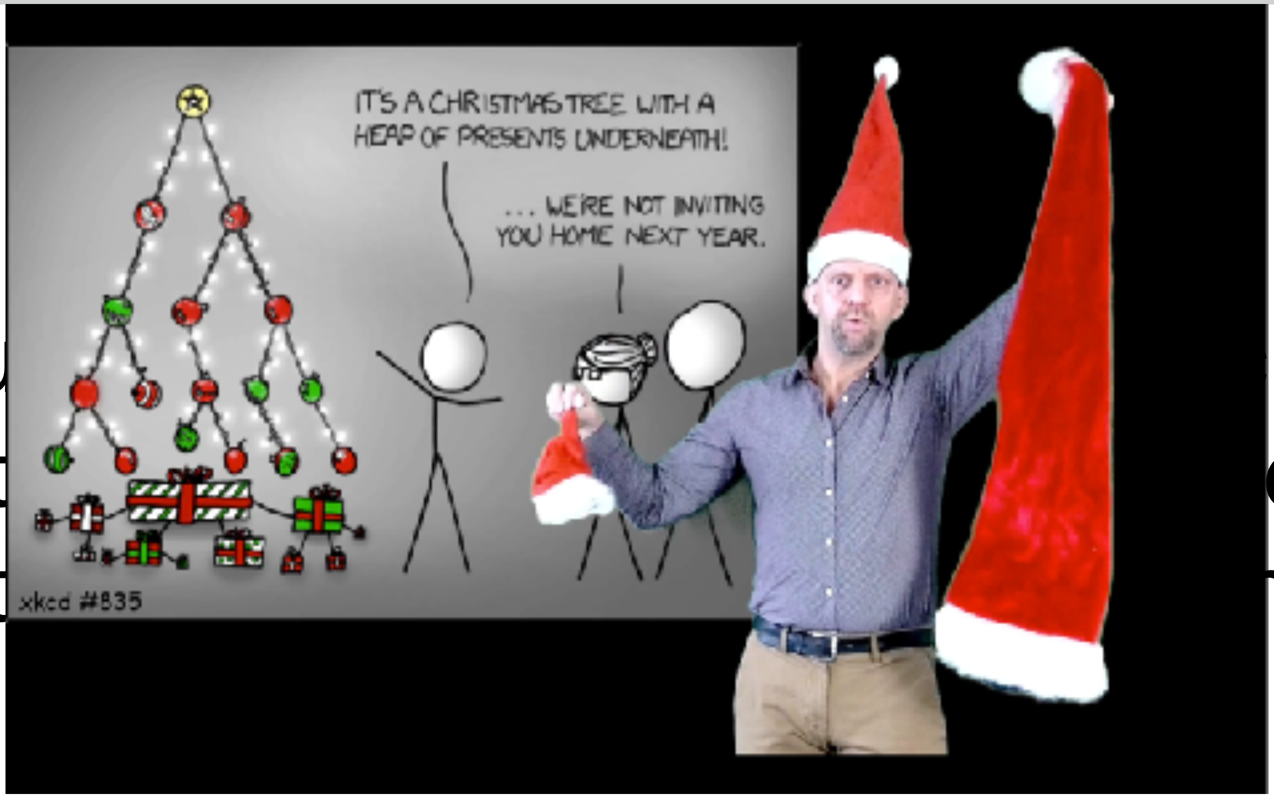
Binäre Bäume (Wiederholung)

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.

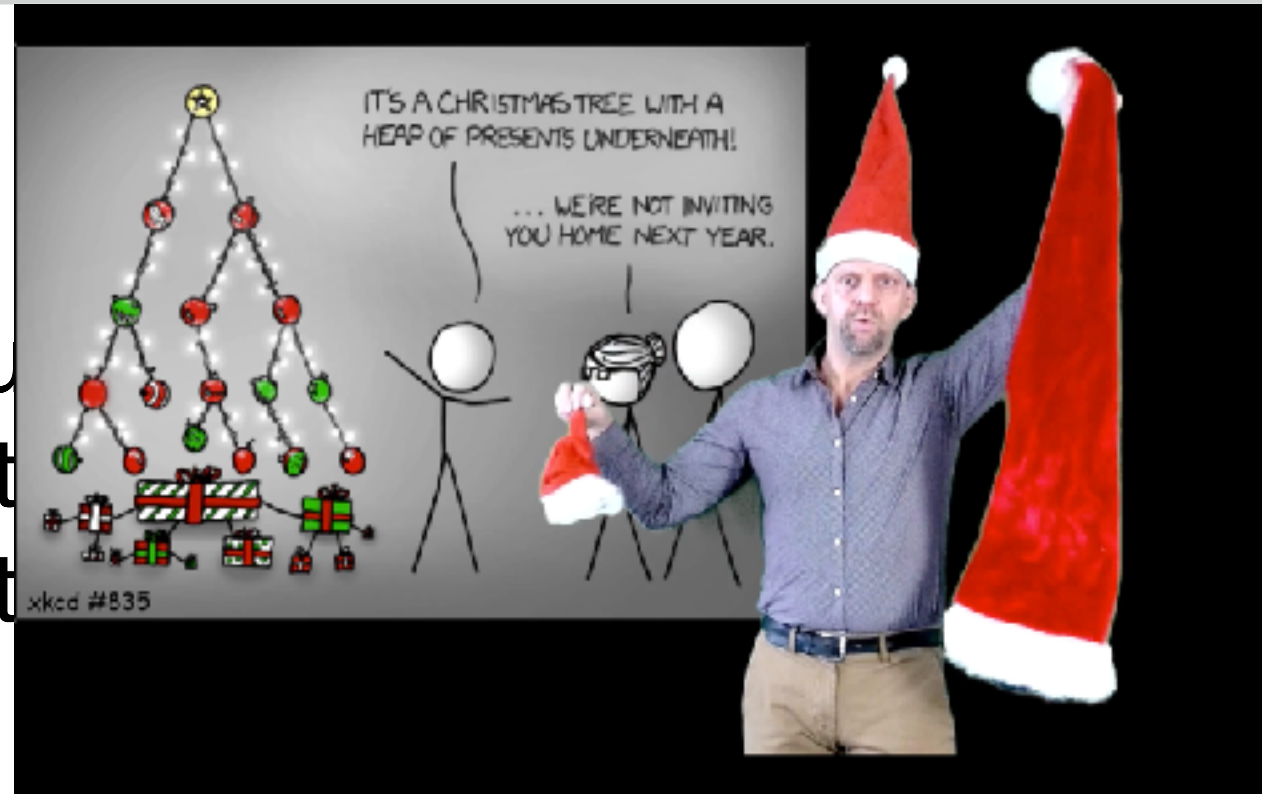


Binäre Bäume (Wiederholung)

Binäre Bäu htete Bäume.
Jeder Knot d(er).
Jeder Knot nau einen Vater.



Binäre Bäume (Wiederholung)



Binäre Bäu htete Bäume.
Jeder Knot d(er).
Jeder Knot nau einen Vater.

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

Binäre Bäume (Wiederholung)

Binäre Bäume (Wiederholung)

Seien folgende Elemente gegeben:

Binäre Bäume (Wiederholung)

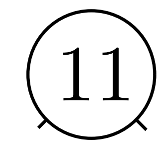
Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

Binäre Bäume (Wiederholung)

Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

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

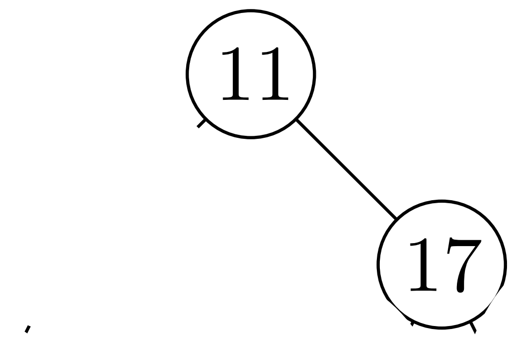
Binäre Bäume (Wiederholung)



Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

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

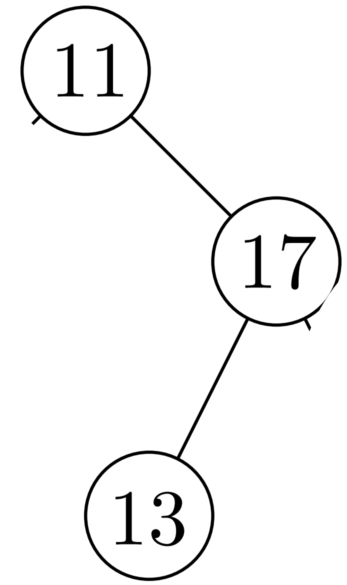
Binäre Bäume (Wiederholung)



Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

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

Binäre Bäume (Wiederholung)

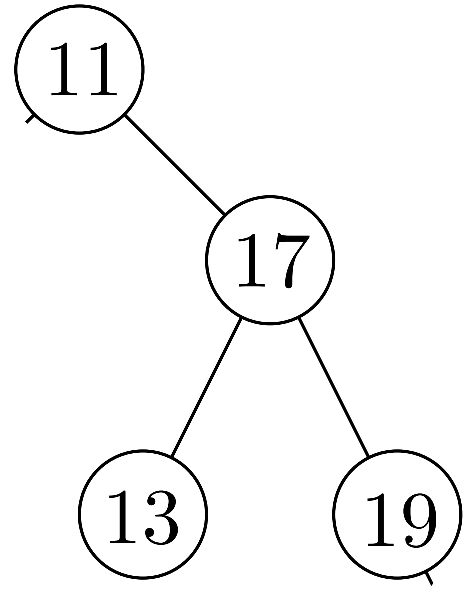


Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

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

Binäre Bäume (Wiederholung)

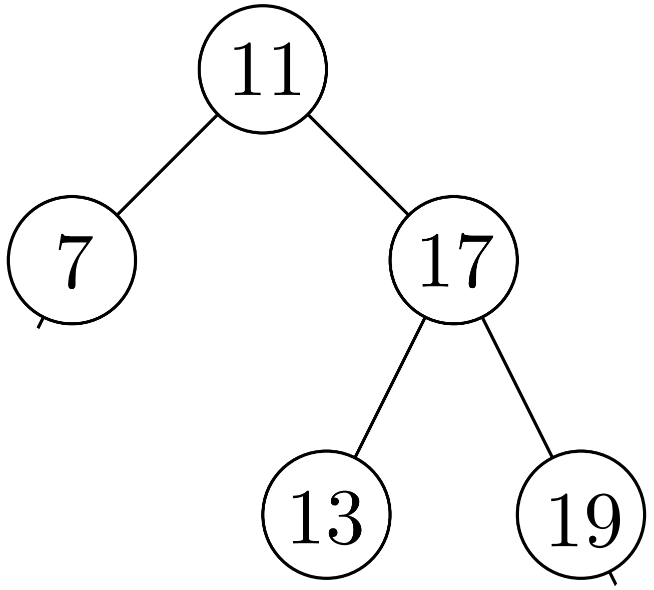
Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.



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

Binäre Bäume (Wiederholung)

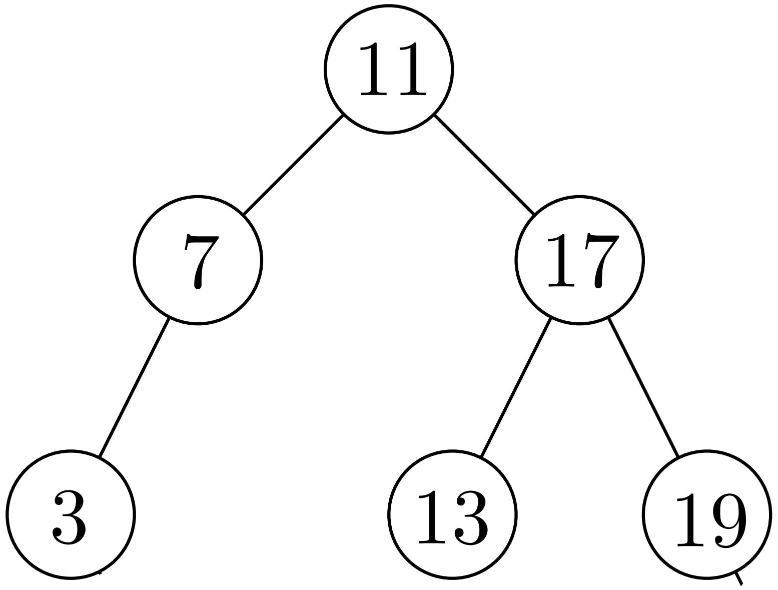
Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.



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

Binäre Bäume (Wiederholung)

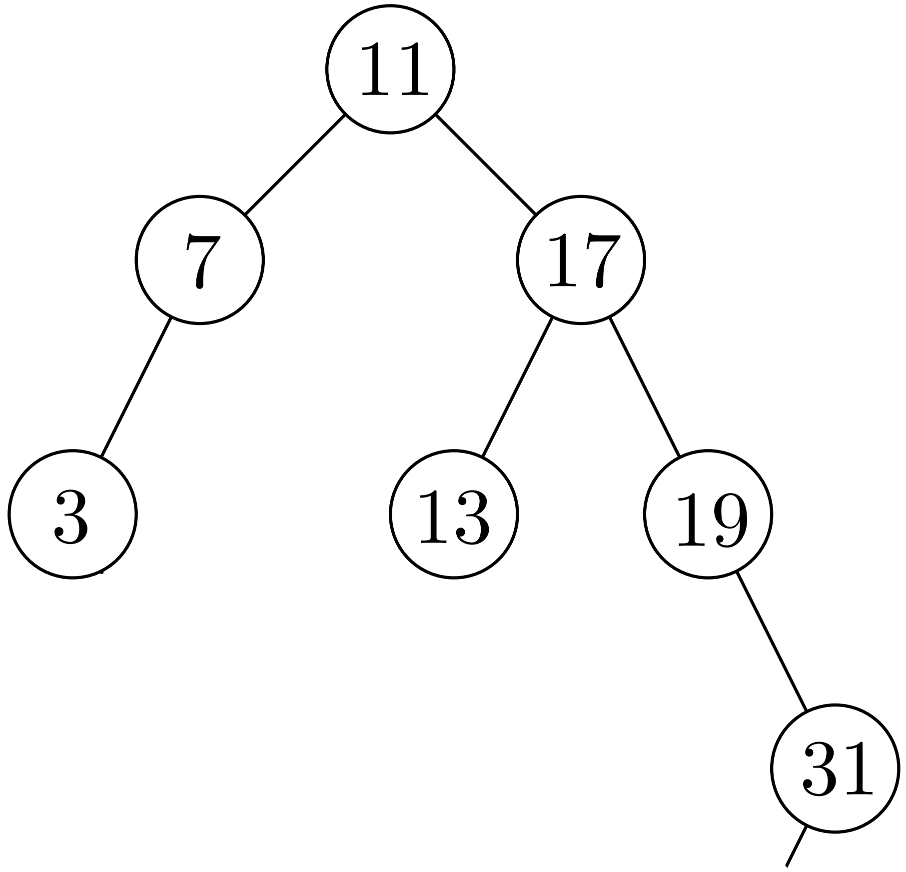
Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.



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

Binäre Bäume (Wiederholung)

Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

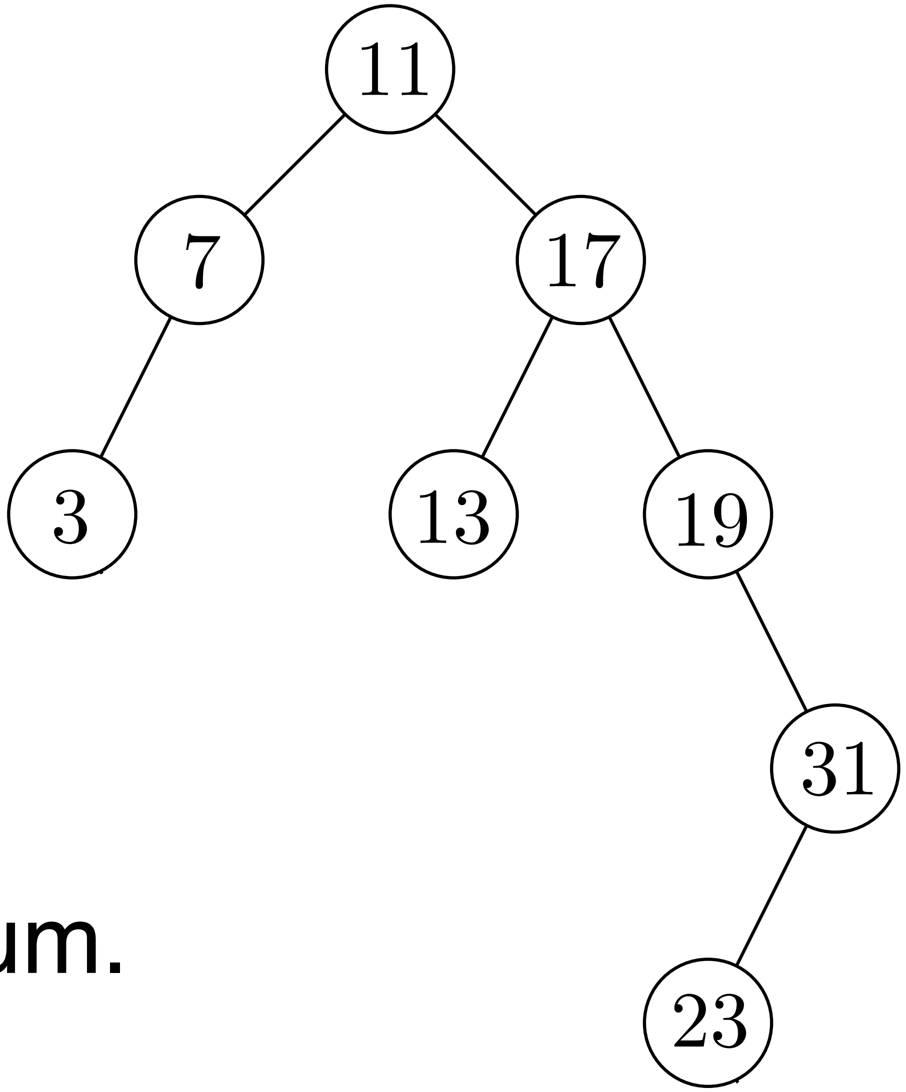


Das Einfügen dieser Elemente von links nach rechts in einen binären Suchbaum ergibt den folgenden Baum.

Binäre Bäume (Wiederholung)

Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

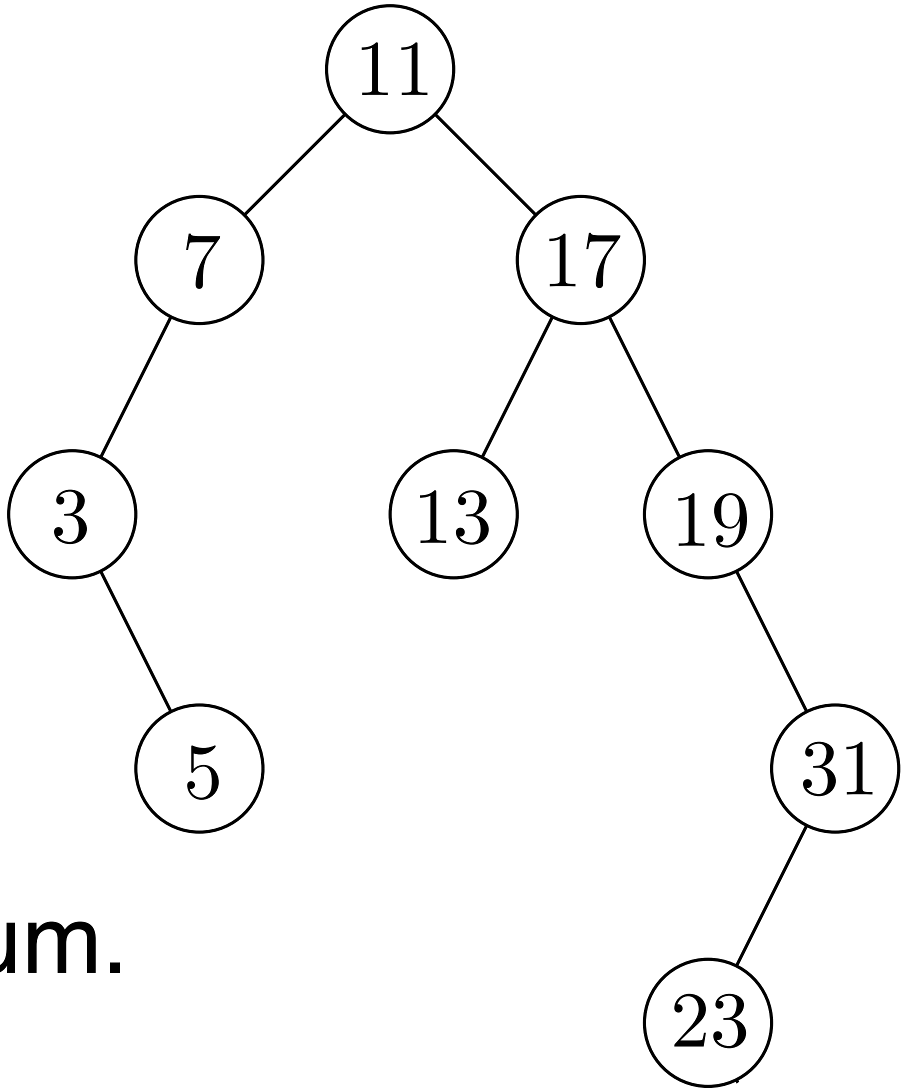
Das Einfügen dieser Elemente von links nach rechts in einen binären Suchbaum ergibt den folgenden Baum.



Binäre Bäume (Wiederholung)

Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

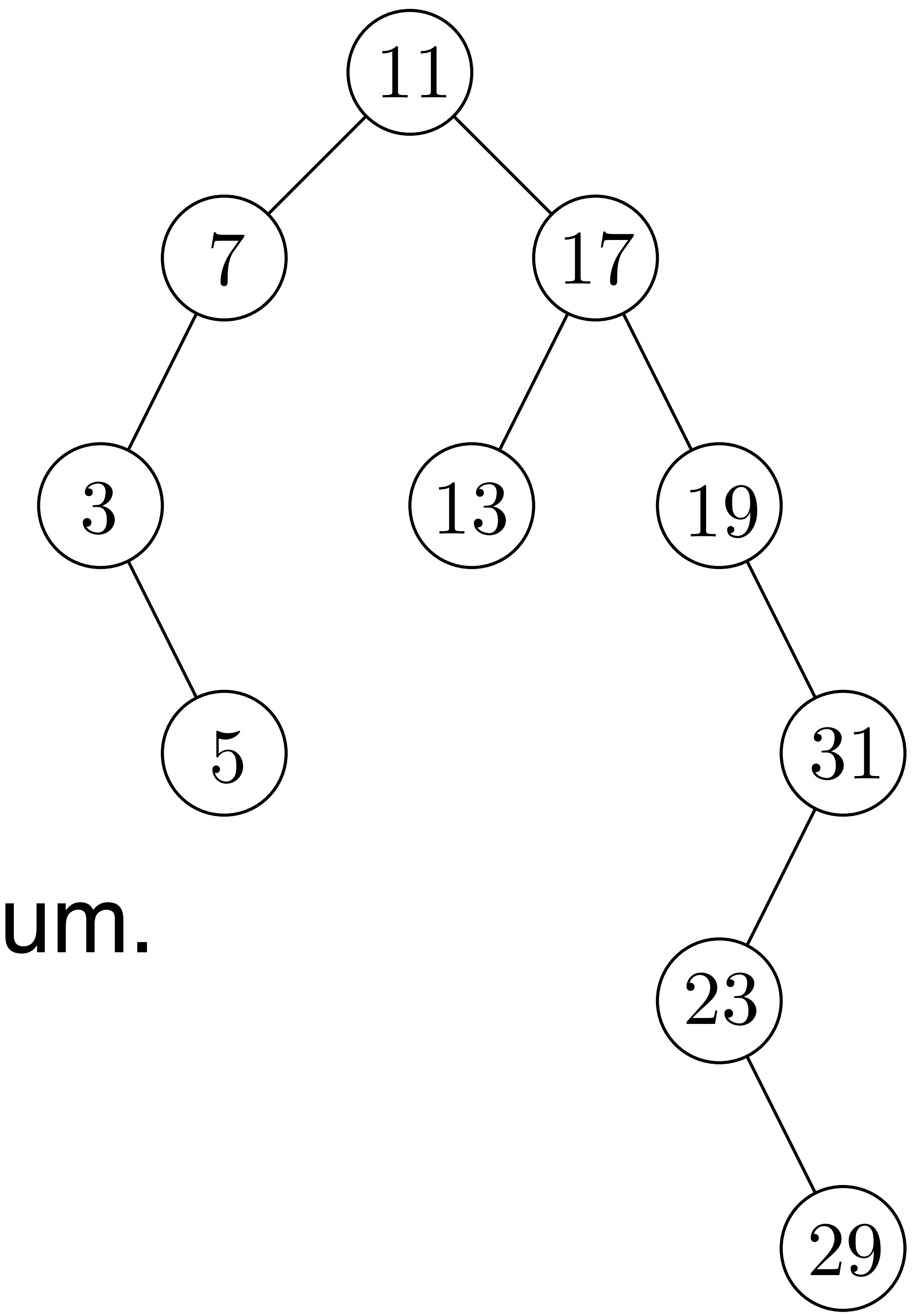
Das Einfügen dieser Elemente von links nach rechts in einen binären Suchbaum ergibt den folgenden Baum.



Binäre Bäume (Wiederholung)

Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.

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



Binäre Bäume (Wiederholung)

Voller binärer Baum

Vollständiger binärer Baum

Degenerierter binärer Baum

Binäre Bäume (Wiederholung)

Voller binärer Baum

- jeder Knoten hat kein oder zwei Kind(er)

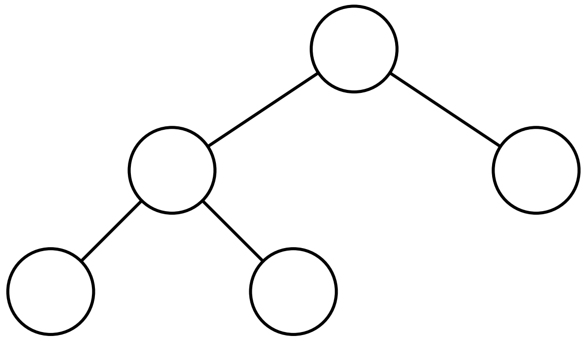
Vollständiger binärer Baum

Degenerierter binärer Baum

Binäre Bäume (Wiederholung)

Voller binärer Baum

- jeder Knoten hat kein oder zwei Kind(er)

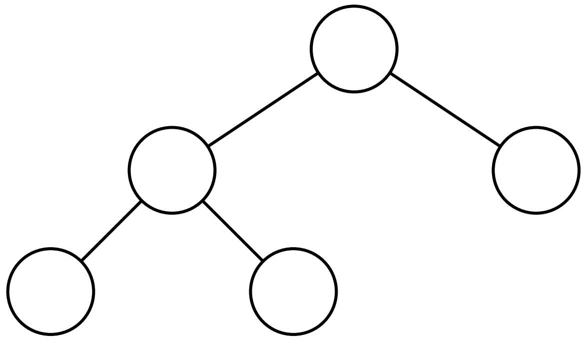


Vollständiger binärer Baum

Degenerierter binärer Baum

Binäre Bäume (Wiederholung)

Voller binärer Baum
- jeder Knoten hat kein oder zwei Kind(er)



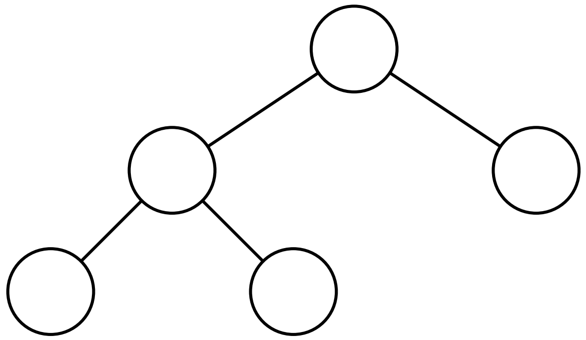
Vollständiger binärer Baum
- voller Baum und alle Blätter haben den gleichen Abstand zur Wurzel

Degenerierter binärer Baum

Binäre Bäume (Wiederholung)

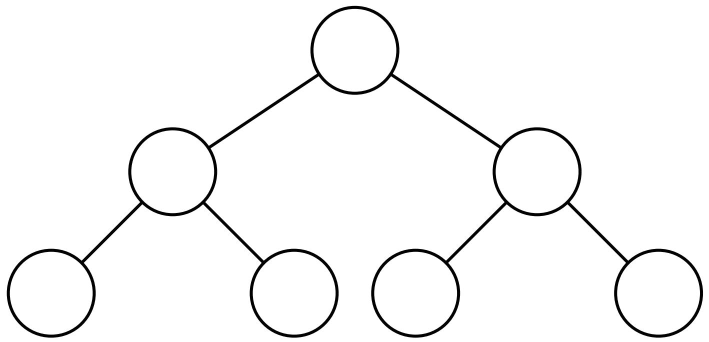
Voller binärer Baum

- jeder Knoten hat kein oder zwei Kind(er)



Vollständiger binärer Baum

- voller Baum und alle Blätter haben den gleichen Abstand zur Wurzel

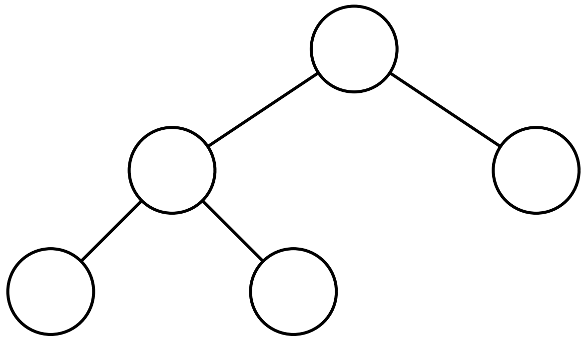


Degenerierter binärer Baum

Binäre Bäume (Wiederholung)

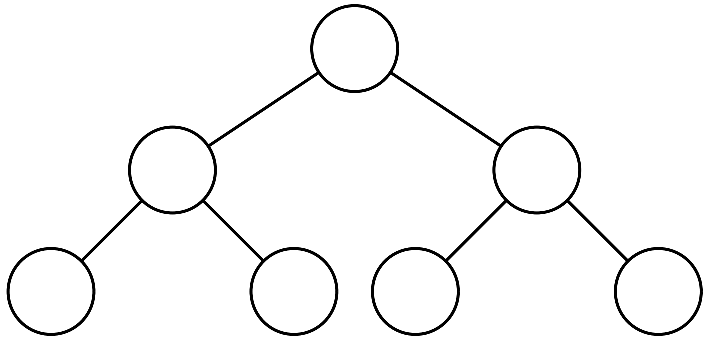
Voller binärer Baum

- jeder Knoten hat kein oder zwei Kind(er)



Vollständiger binärer Baum

- voller Baum und alle Blätter haben den gleichen Abstand zur Wurzel



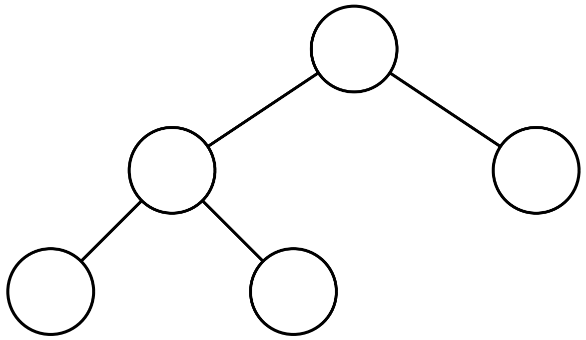
Degenerierter binärer Baum

- jeder Knoten hat maximal ein Kind

Binäre Bäume (Wiederholung)

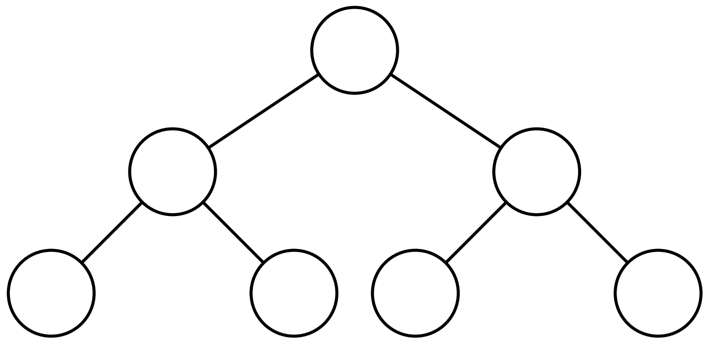
Voller binärer Baum

- jeder Knoten hat kein oder zwei Kind(er)



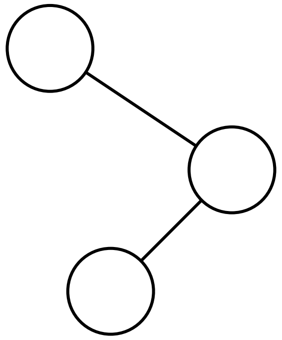
Vollständiger binärer Baum

- voller Baum und alle Blätter haben den gleichen Abstand zur Wurzel

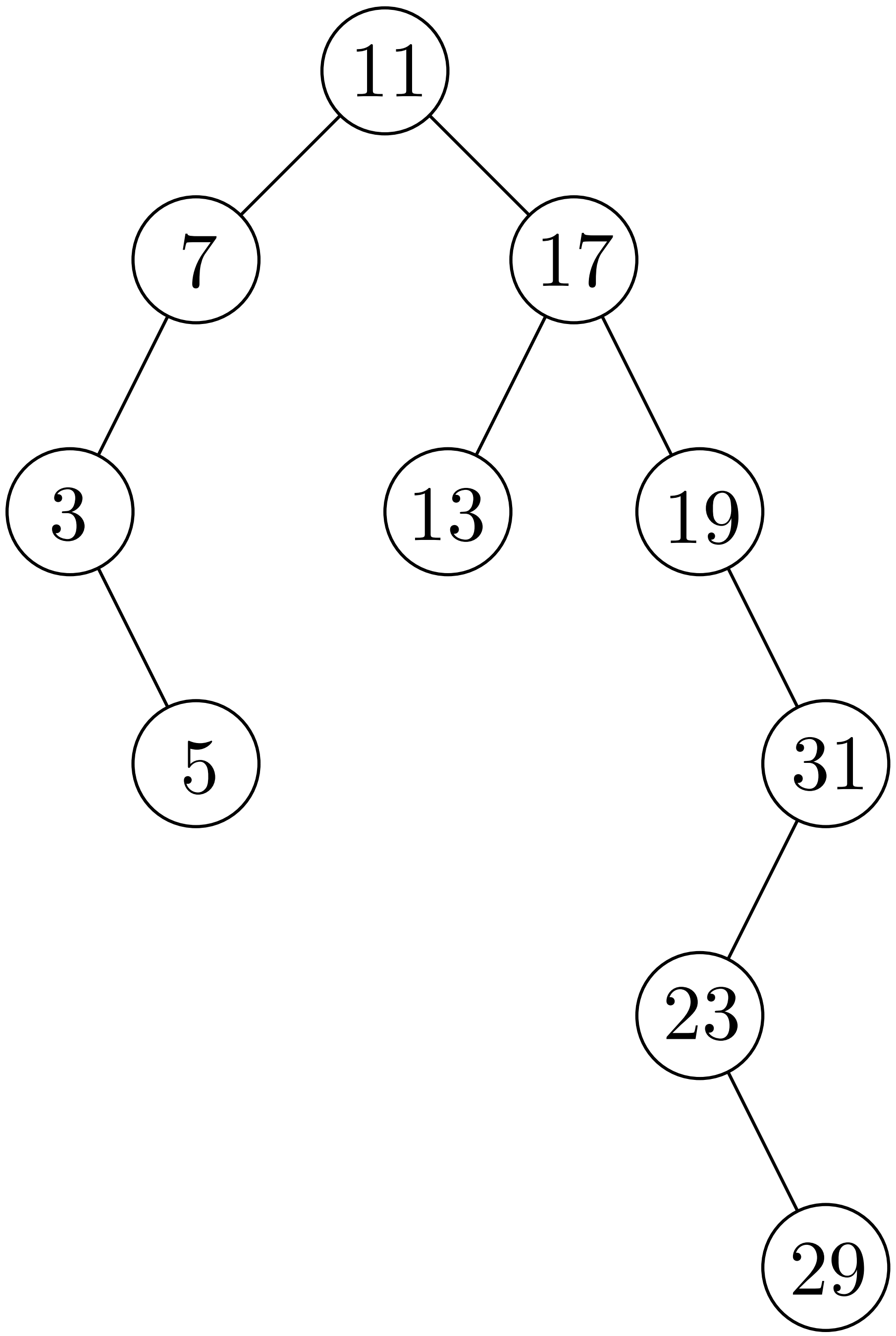


Degenerierter binärer Baum

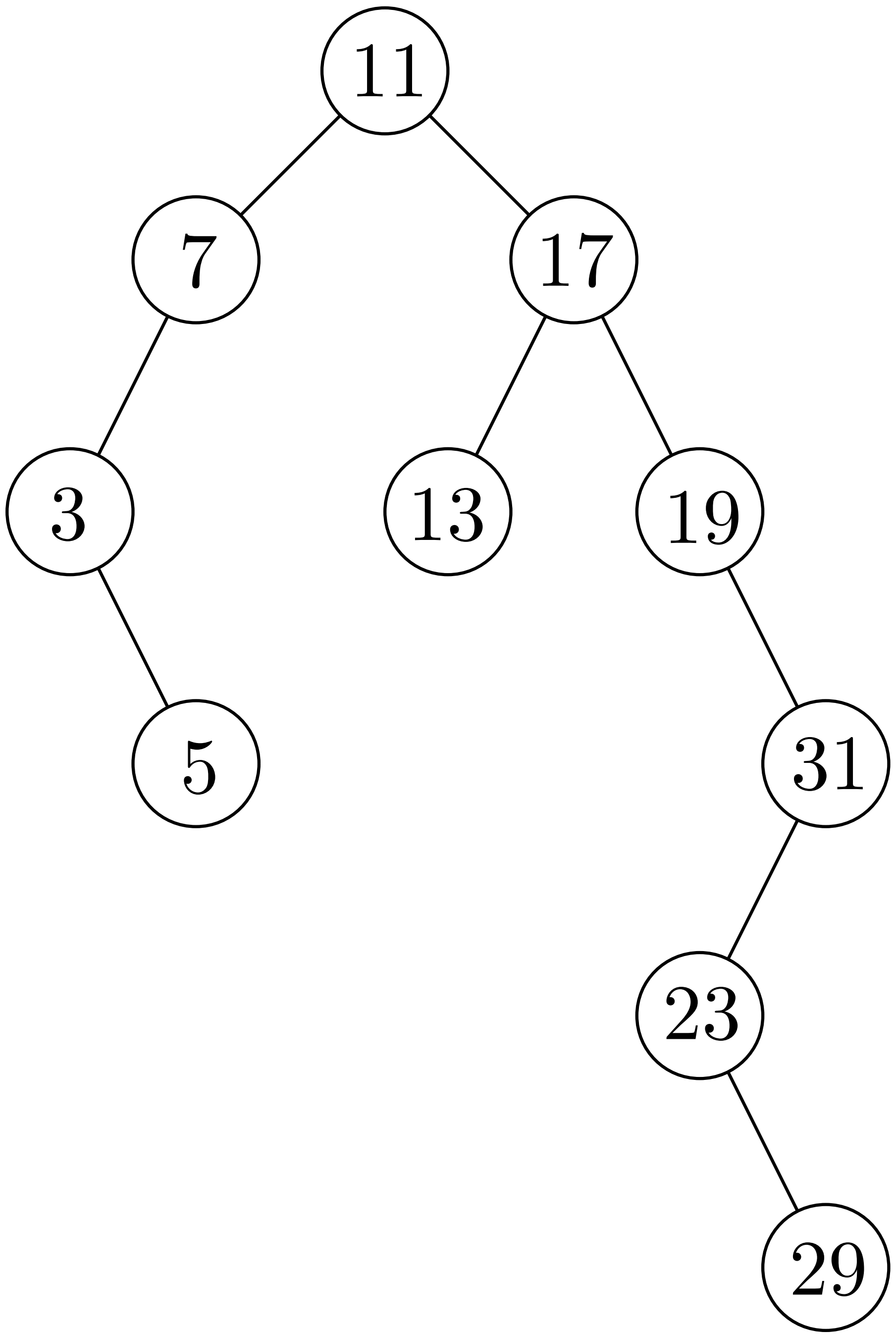
- jeder Knoten hat maximal ein Kind



Binäre Bäume (Wiederholung)

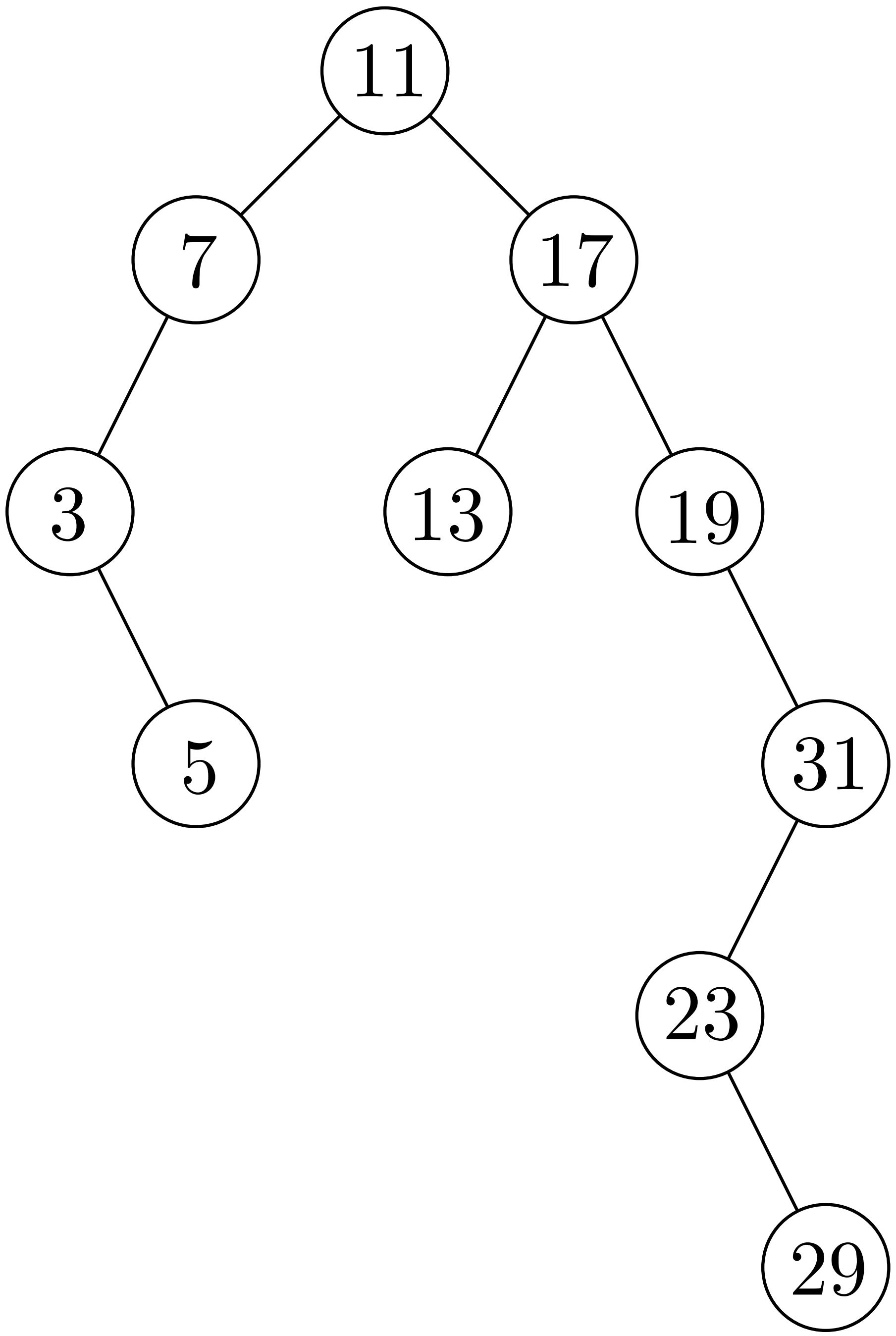


Binäre Bäume (Wiederholung)



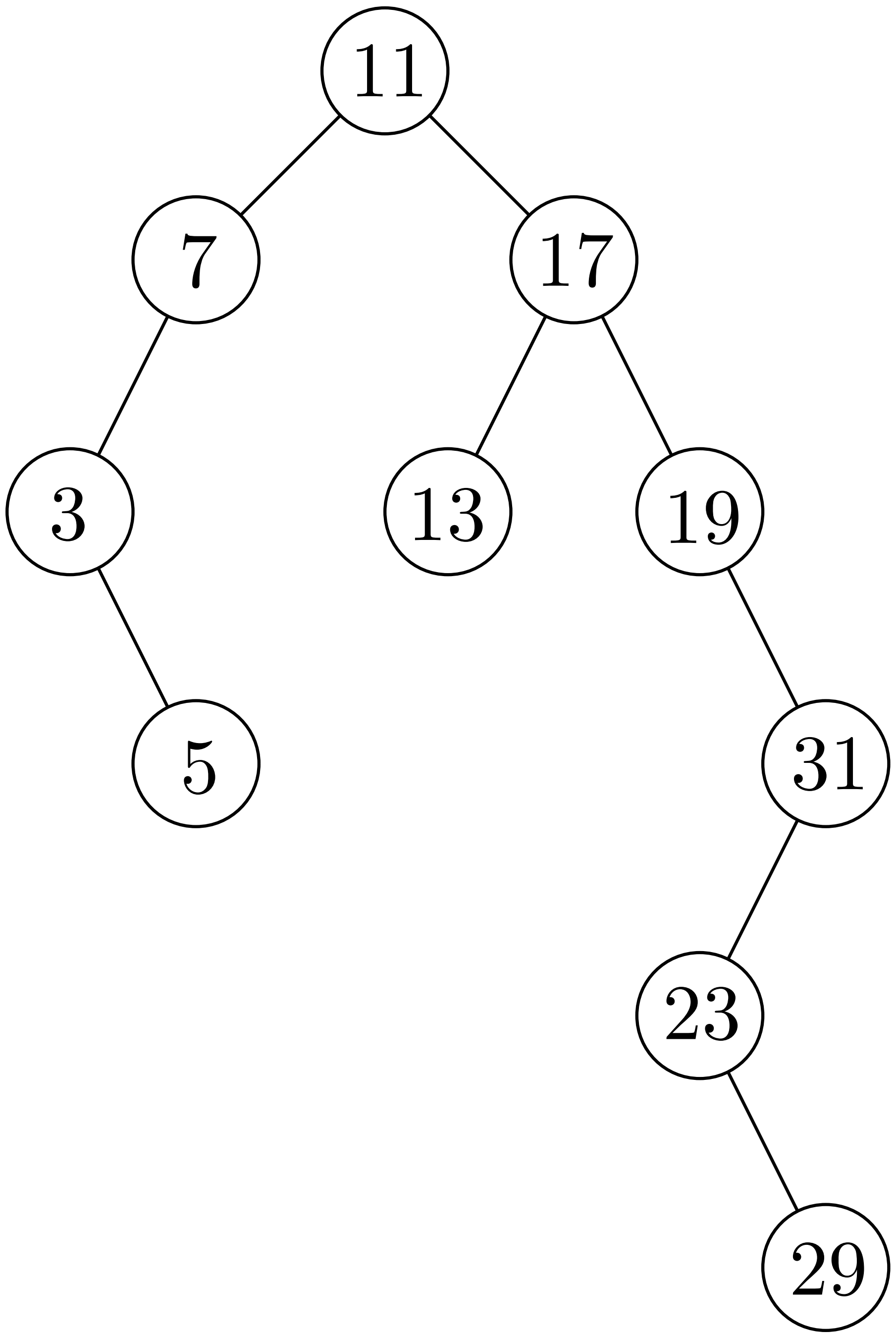
- Suchen

Binäre Bäume (Wiederholung)



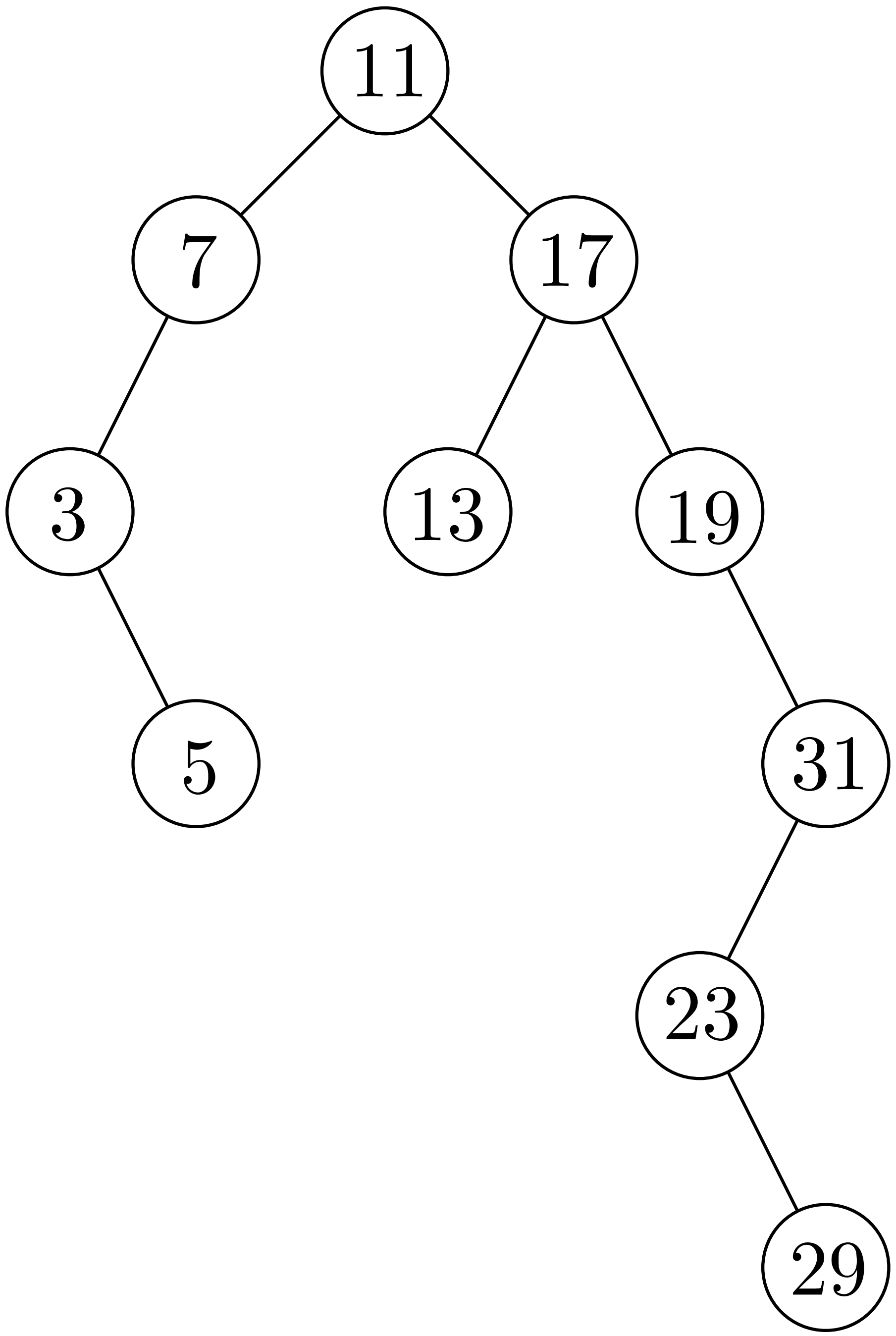
- Suchen
- Minimum

Binäre Bäume (Wiederholung)



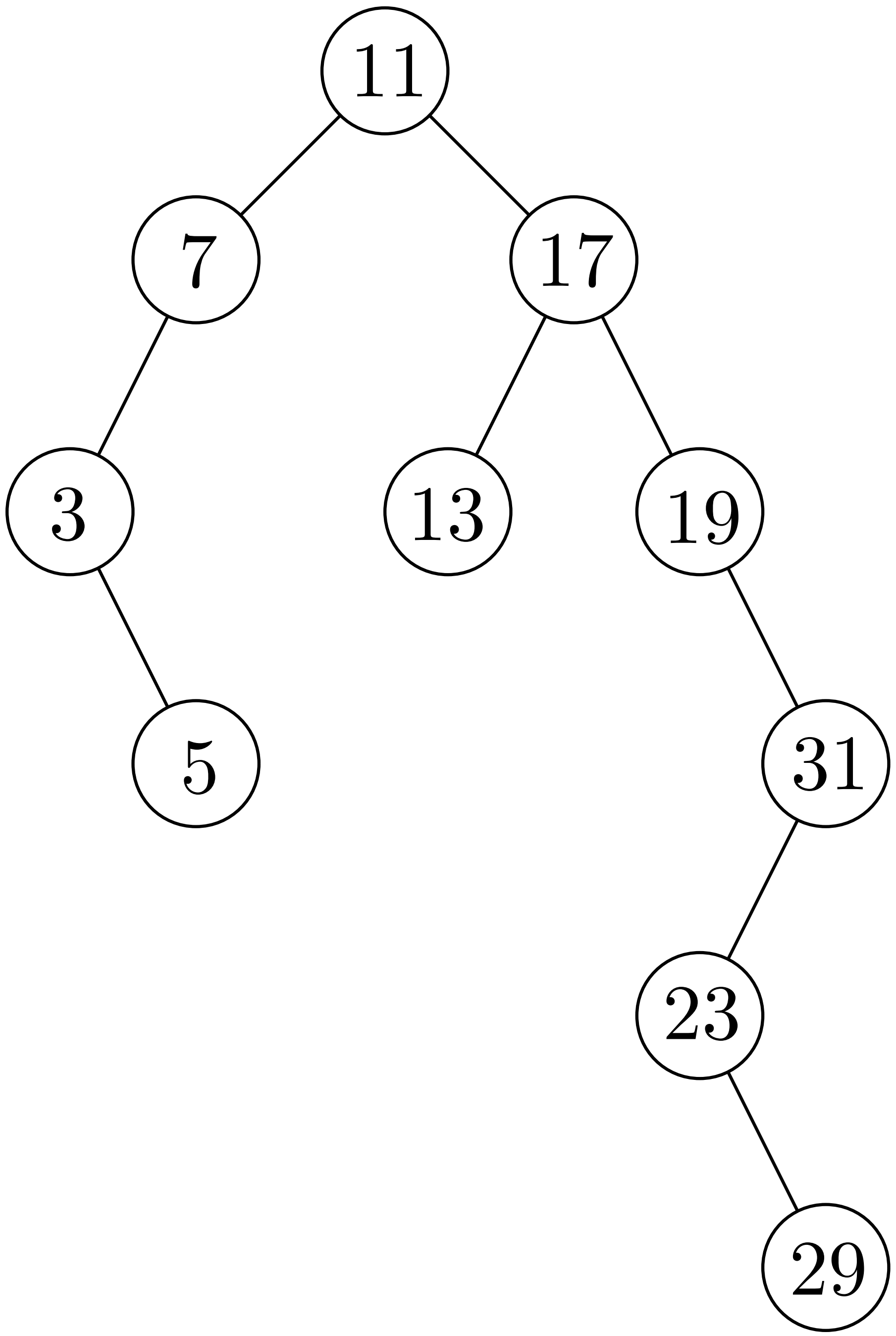
- Suchen
- Minimum
- Maximum

Binäre Bäume (Wiederholung)



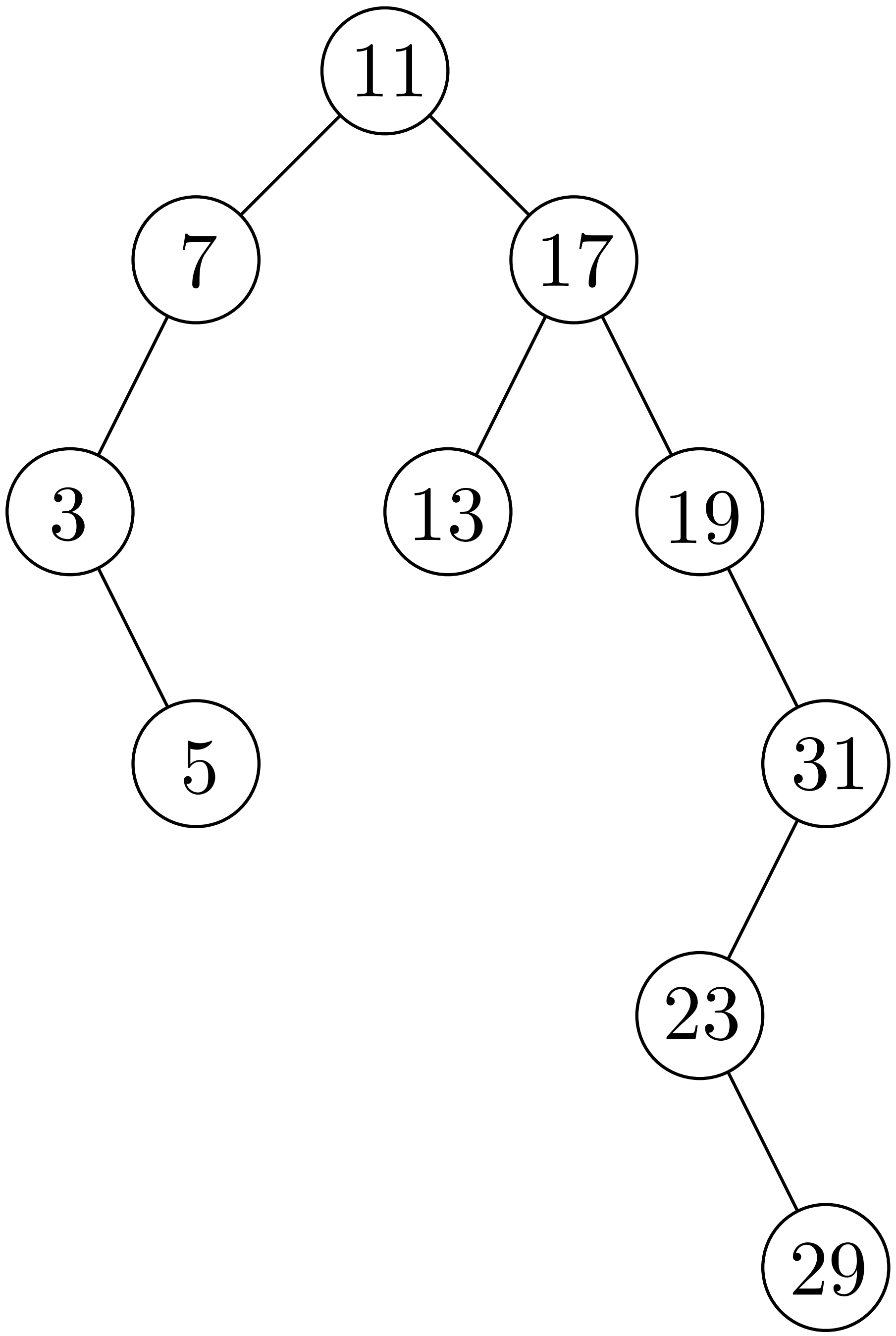
- Suchen
- Minimum
- Maximum
- Nachfolger

Binäre Bäume (Wiederholung)



- Suchen
- Minimum
- Maximum
- Nachfolger
- Vorgänger

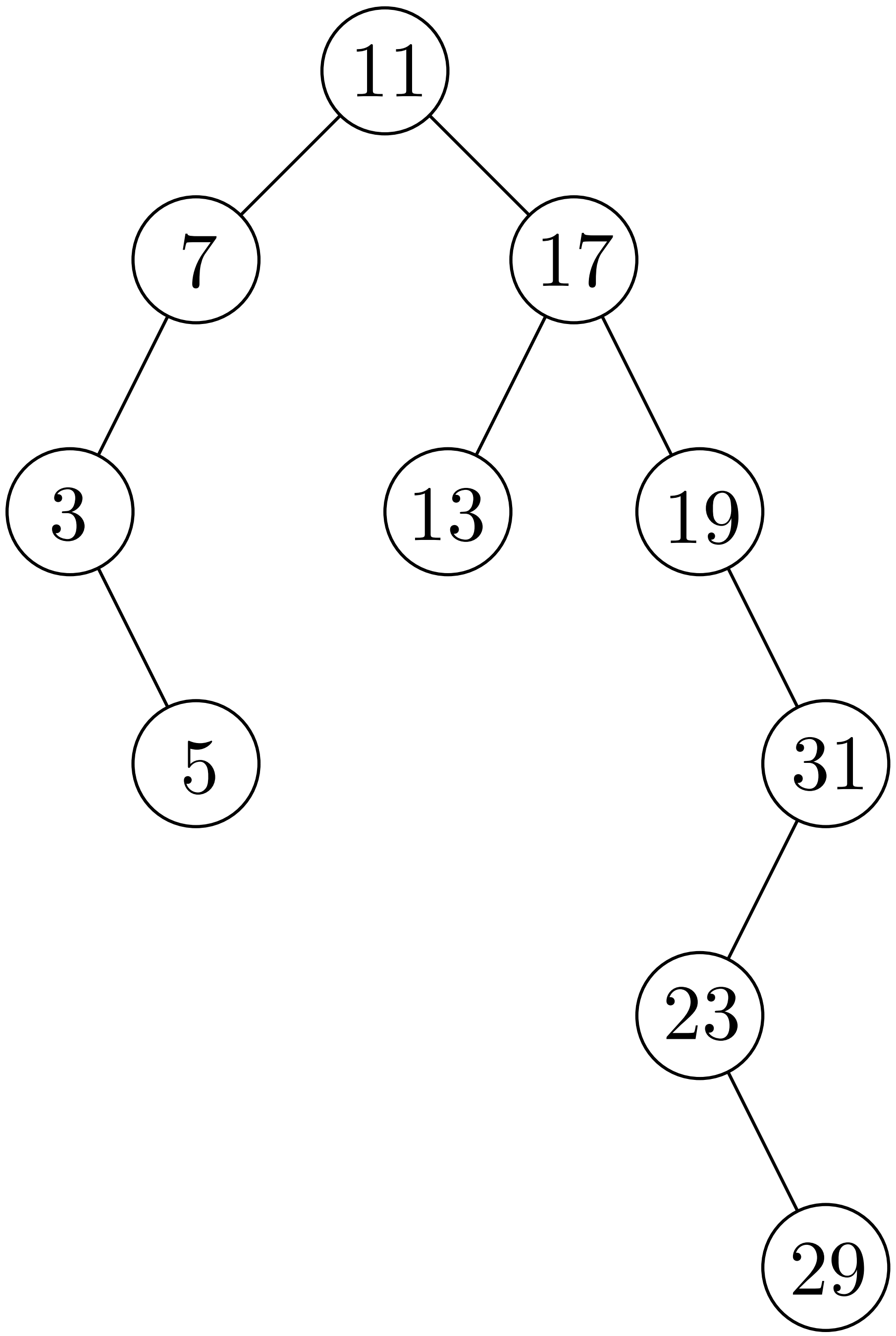
Binäre Bäume (Wiederholung)



- Suchen
- Minimum
- Maximum
- Nachfolger
- Vorgänger

Satz 4.4

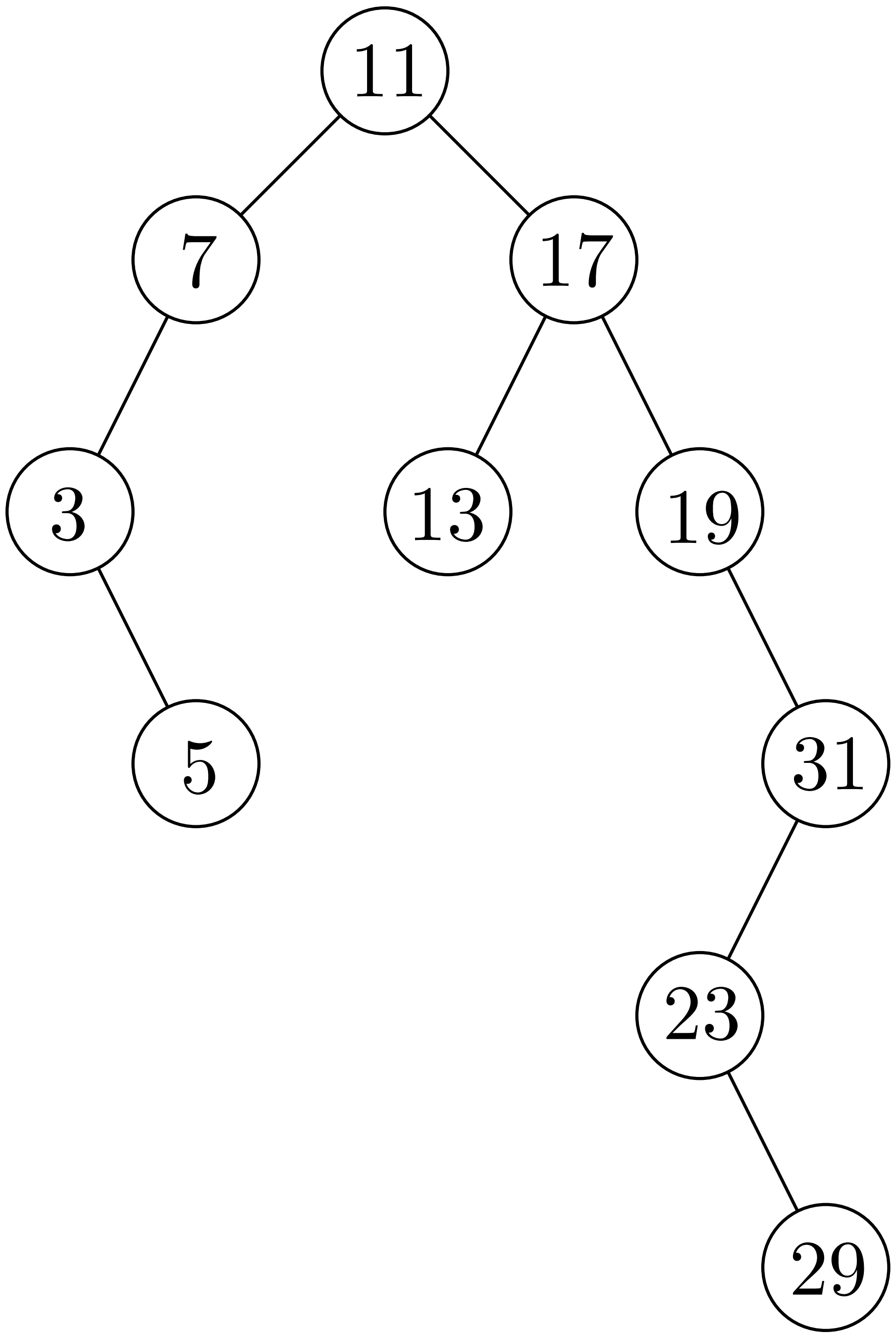
Binäre Bäume (Wiederholung)



- Suchen
- Minimum
- Maximum
- Nachfolger
- Vorgänger
- Einfügen

Satz 4.4

Binäre Bäume (Wiederholung)

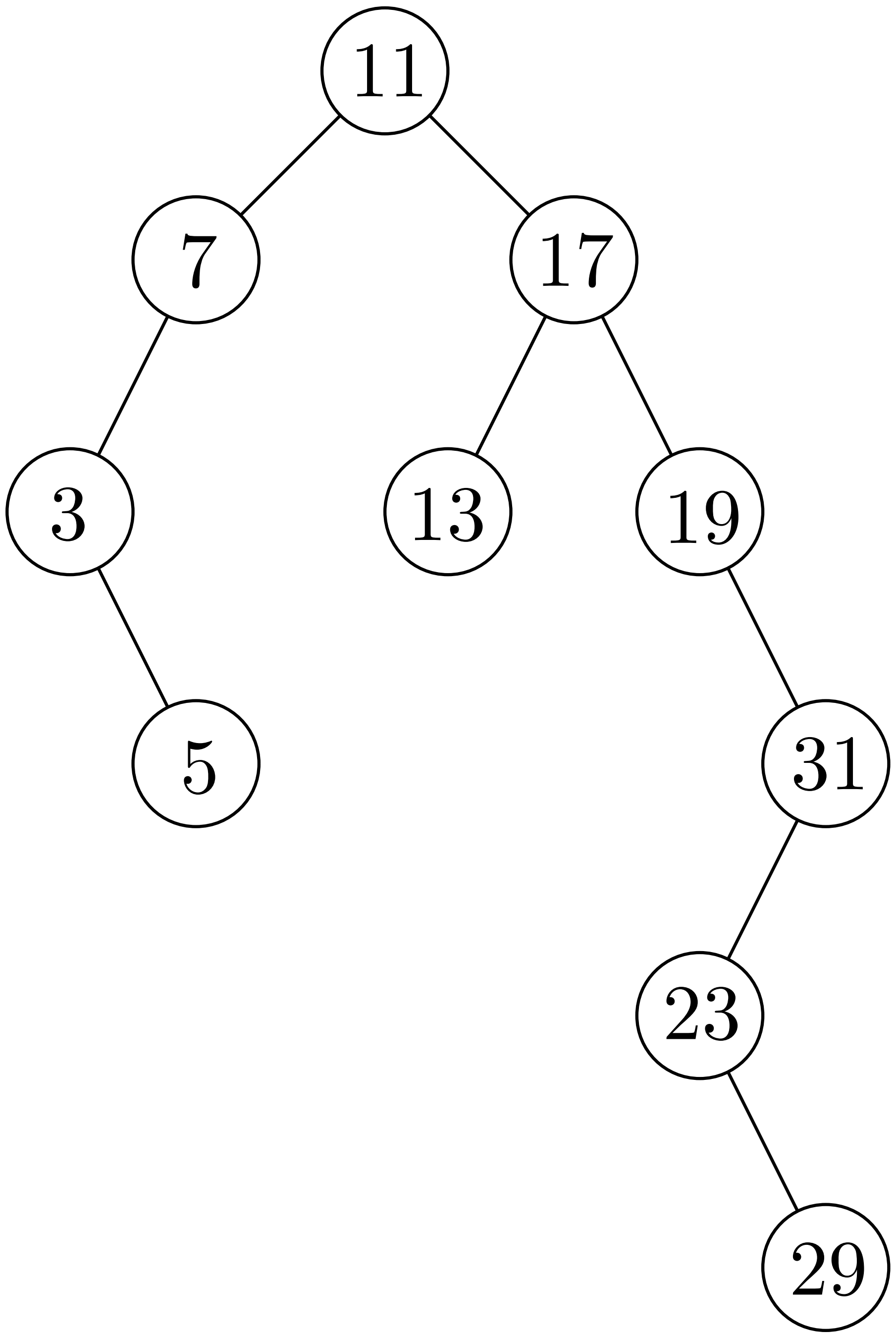


- Suchen
- Minimum
- Maximum
- Nachfolger
- Vorgänger
- Einfügen

Satz 4.4

Satz 4.5

Binäre Bäume (Wiederholung)

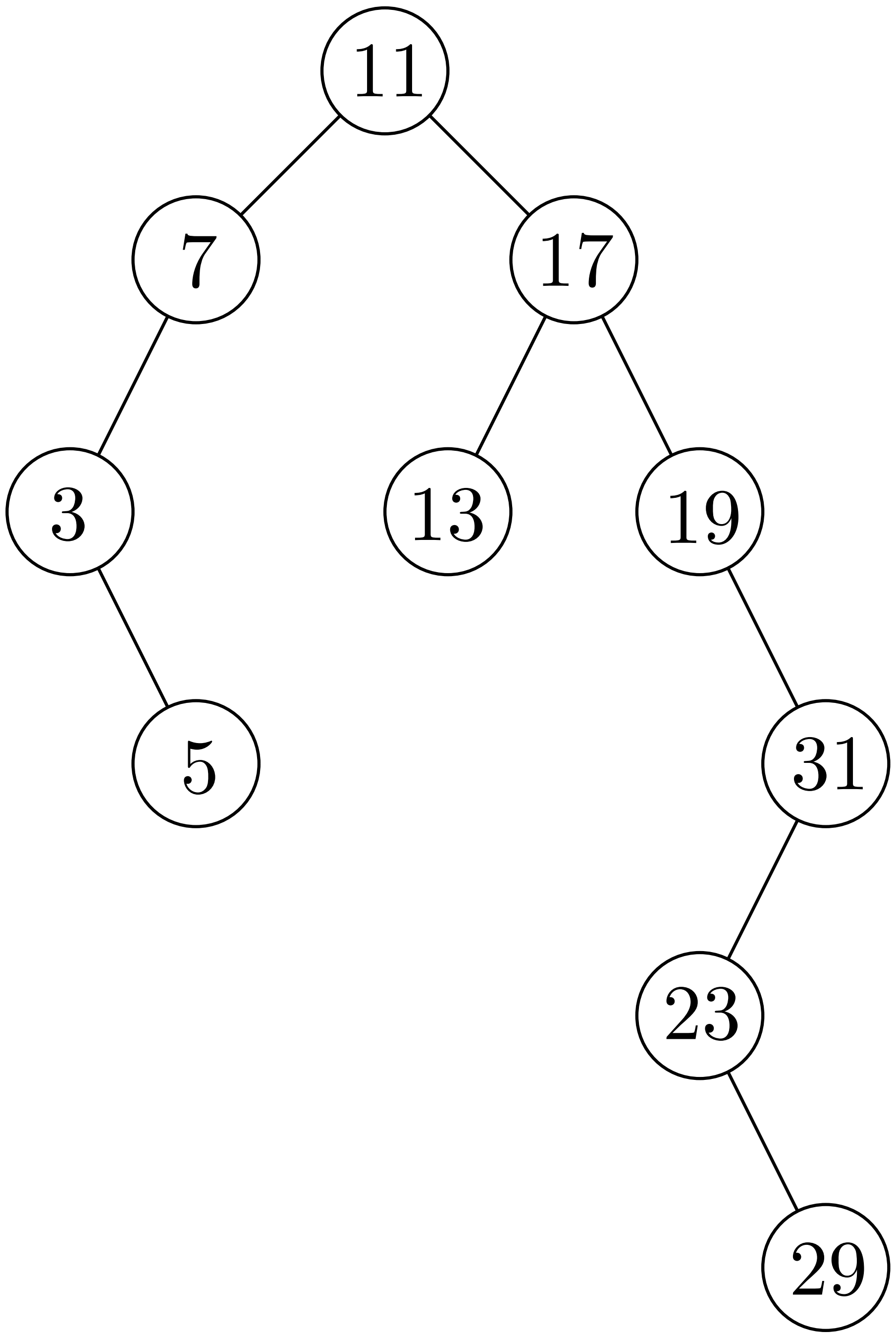


- Suchen
- Minimum
- Maximum
- Nachfolger
- Vorgänger
- Einfügen
- Löschen

Satz 4.4

Satz 4.5

Binäre Bäume (Wiederholung)



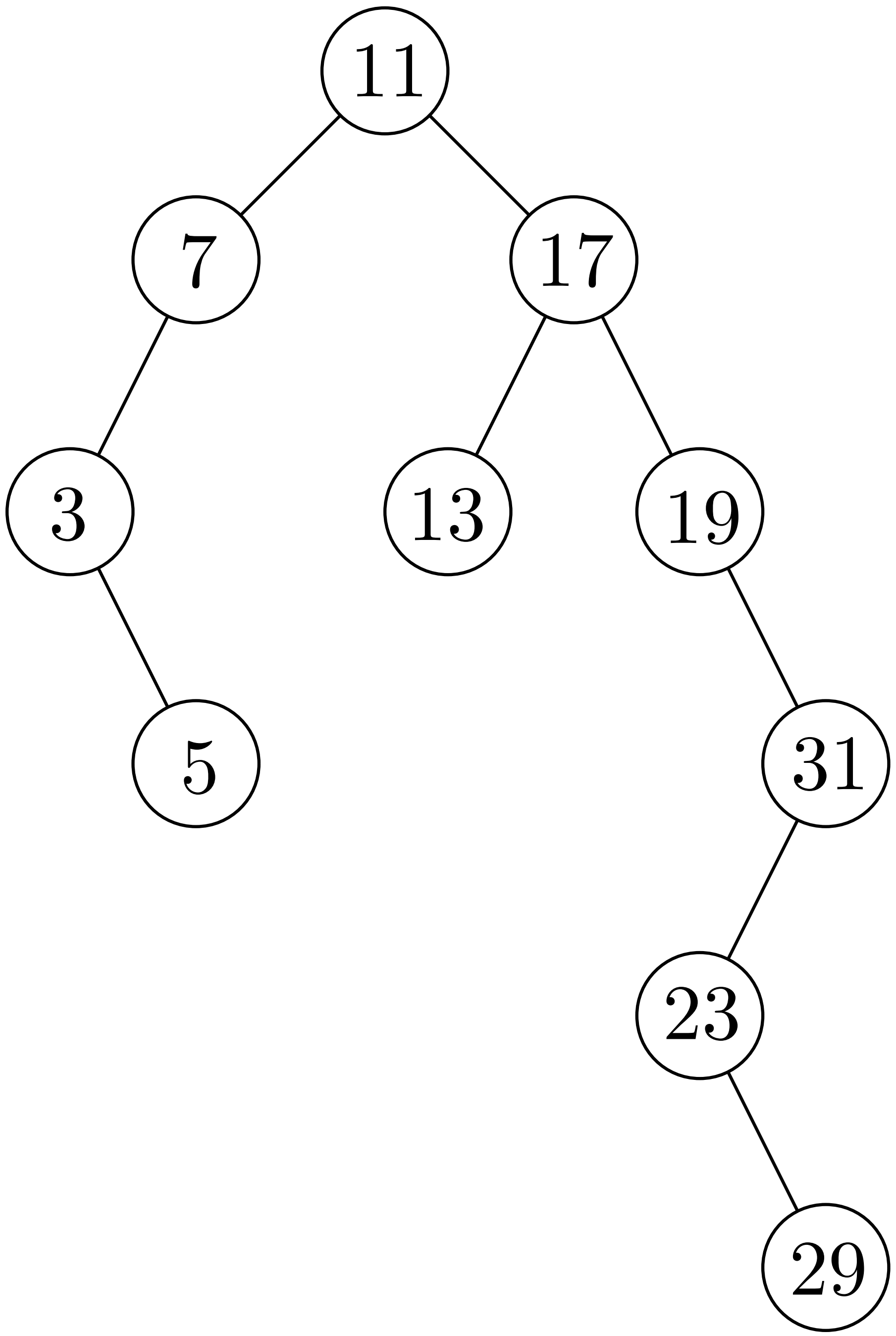
- Suchen
- Minimum
- Maximum
- Nachfolger
- Vorgänger
- Einfügen
- Löschen

Satz 4.4

Satz 4.5

Satz 4.6

Binäre Bäume (Wiederholung)



- Suchen
- Minimum
- Maximum
- Nachfolger
- Vorgänger
- Einfügen
- Löschen

Satz 4.4

Satz 4.5

Satz 4.6

O(h)

Höhenbalanciert (Wiederholung)

Wann ist ein binärer Suchbaum (mit vielen Knoten) gut?

Höhenbalanciert (Wiederholung)

Wann ist ein binärer Suchbaum (mit vielen Knoten) gut?

- Wenn die Teilbäume eines Knotens gleich groß sind?

Höhenbalanciert (Wiederholung)

Wann ist ein binärer Suchbaum (mit vielen Knoten) gut?

- Wenn die Teilbäume eines Knotens gleich groß sind?
- Wenn die Teilbäume unterschiedlich groß sind?

Höhenbalanciert (Wiederholung)

Wann ist ein binärer Suchbaum (mit vielen Knoten) gut?

- Wenn die Teilbäume eines Knotens gleich groß sind?
- Wenn die Teilbäume unterschiedlich groß sind?
- Wenn die Teilbäume unterschiedlich groß sind, aber einem gewissen Verhältnis / Regeln genügen?

Höhenbalanciert (Wiederholung)

Wann ist ein binärer Suchbaum (mit vielen Knoten) gut?

- Wenn die Teilbäume eines Knotens gleich groß sind?
- Wenn die Teilbäume unterschiedlich groß sind?
- Wenn die Teilbäume unterschiedlich groß sind, aber einem gewissen Verhältnis / Regeln genügen?
- Wenn es nur einen Teilbaum in jedem Knoten gibt?

Höhenbalanciert (Wiederholung)

Wann ist ein binärer Suchbaum (mit vielen Knoten) gut?

- Wenn die Teilbäume eines Knotens gleich groß sind?
- Wenn die Teilbäume unterschiedlich groß sind?
- Wenn die Teilbäume unterschiedlich groß sind, aber einem gewissen Verhältnis / Regeln genügen?
- Wenn es nur einen Teilbaum in jedem Knoten gibt?

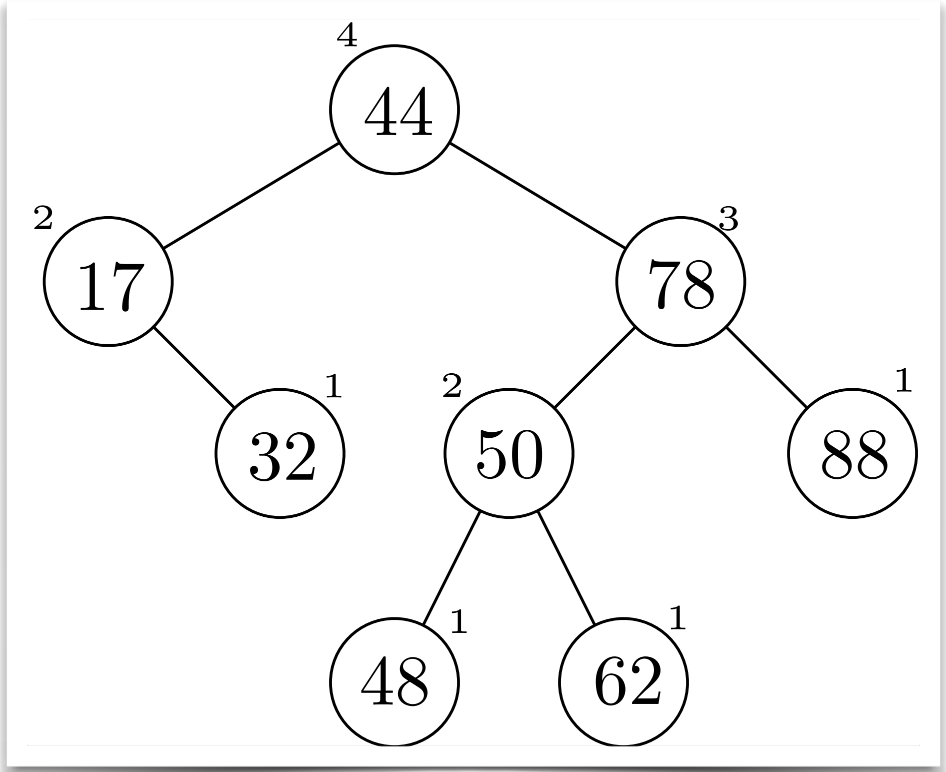
Wir wollen eine Höhe von O(log n) erreichen und beibehalten!

AVL-Bäume (Wiederholung)

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

AVL-Bäume (Wiederholung)

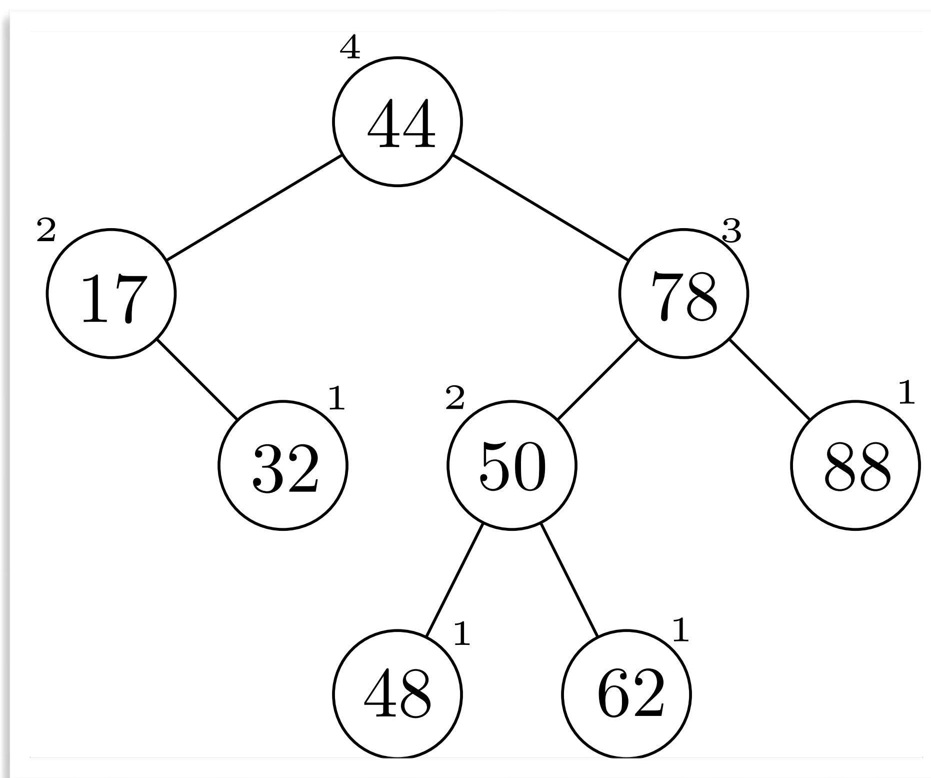
Definition 4.7 (Nach Adelson-Velski und Landis, 1962)



AVL-Bäume (Wiederholung)

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

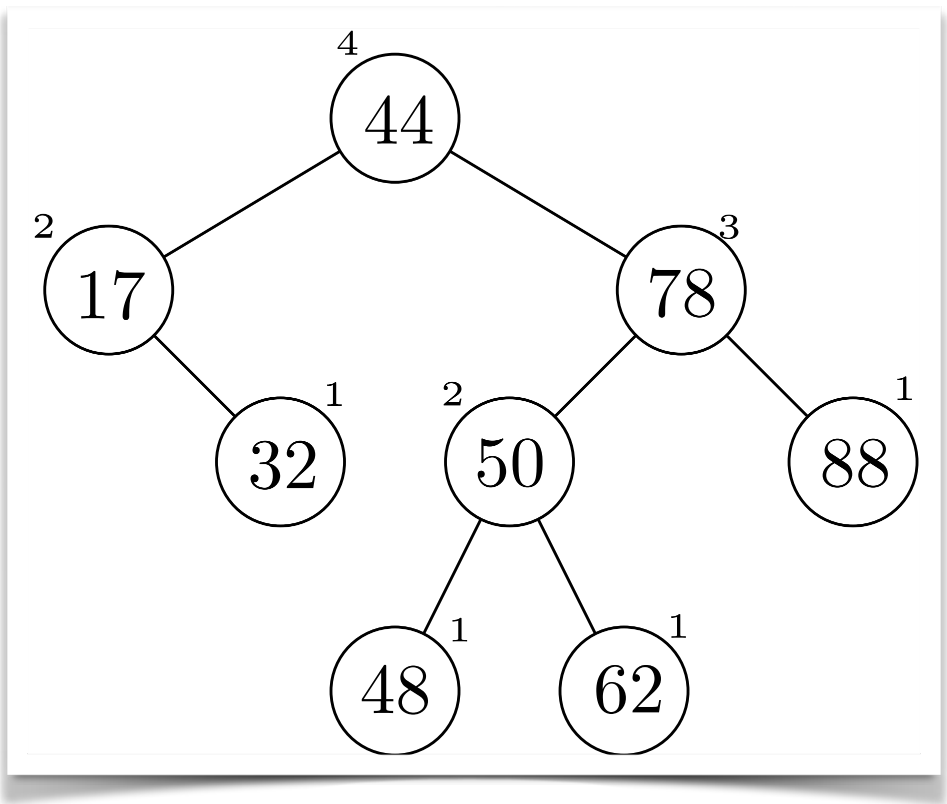
1. Ein binärer Suchbaum ist höhenbalanciert, wenn sich für jeden inneren Knoten v die Höhe der beiden Kinder von v um höchstens 1 unterscheidet.



AVL-Bäume (Wiederholung)

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.

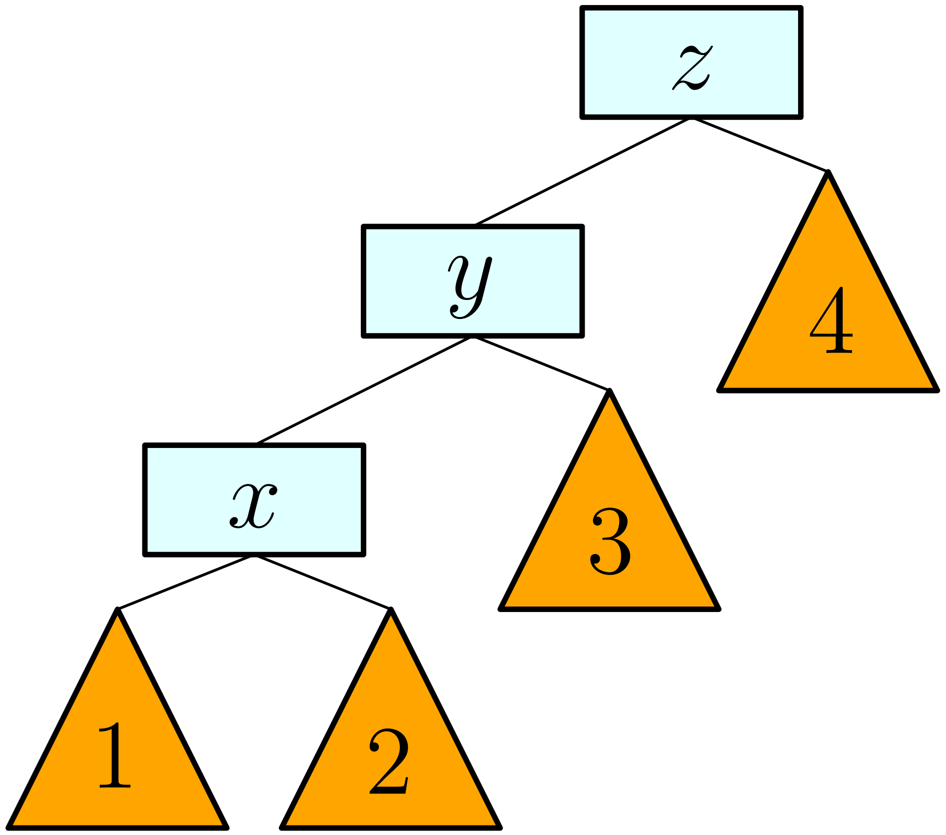


Rotation / Restructure

Falls ein binärer Suchbaum nicht mehr balanciert ist, kann man über unterschiedliche Rotationen lokal rebalancieren.

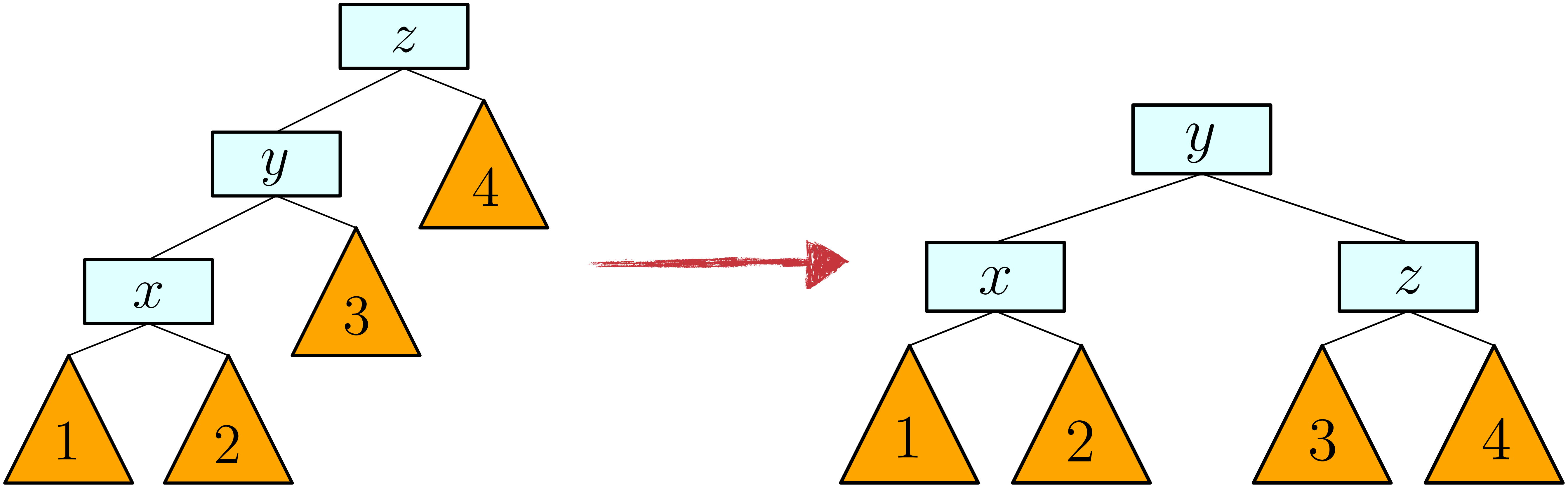
Rotation / Restructure

Falls ein binärer Suchbaum nicht mehr balanciert ist, kann man über unterschiedliche Rotationen lokal rebalancieren.



Rotation / Restructure

Falls ein binärer Suchbaum nicht mehr balanciert ist, kann man über unterschiedliche Rotationen lokal rebalancieren.



4.8 Rot-Schwarz-Bäume

Quiz!

Definition 4.12 (Rot-Schwarz-Baum)

4.8 Rot-Schwarz-Bäume

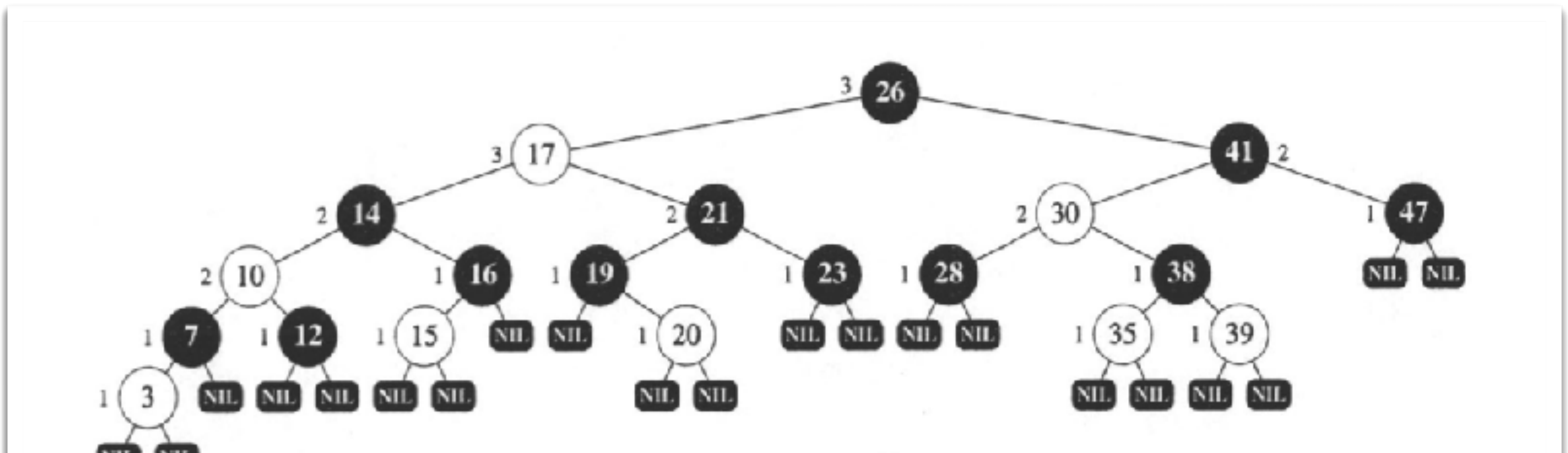
Definition 4.12 (Rot-Schwarz-Baum)

Ein binärer Suchbaum heißt Rot-Schwarz-Baum, wenn er die folgenden Eigenschaften erfüllt:

4.8 Rot-Schwarz-Bäume

Definition 4.12 (Rot-Schwarz-Baum)

Ein binärer Suchbaum heißt Rot-Schwarz-Baum, wenn er die folgenden Eigenschaften erfüllt:

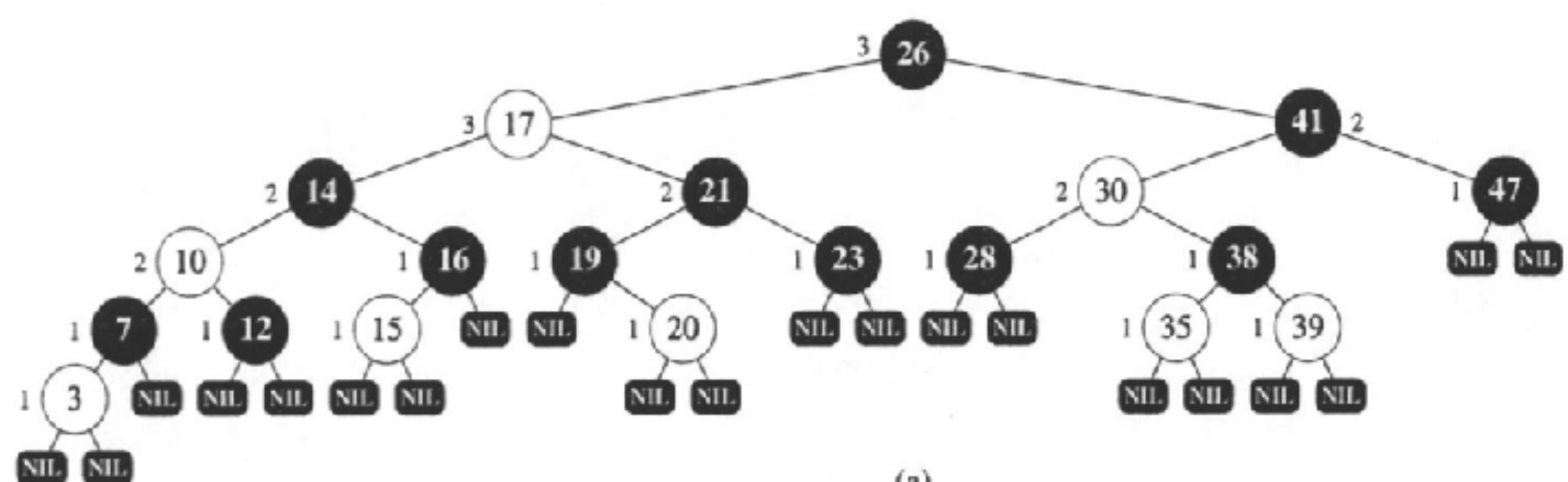


(a)

Definition 4.12 (Rot-Schwarz-Baum)

Ein binärer Suchbaum heißt Rot-Schwarz-Baum, wenn er die folgenden Eigenschaften erfüllt:

1. Jeder Knoten ist entweder rot oder schwarz.



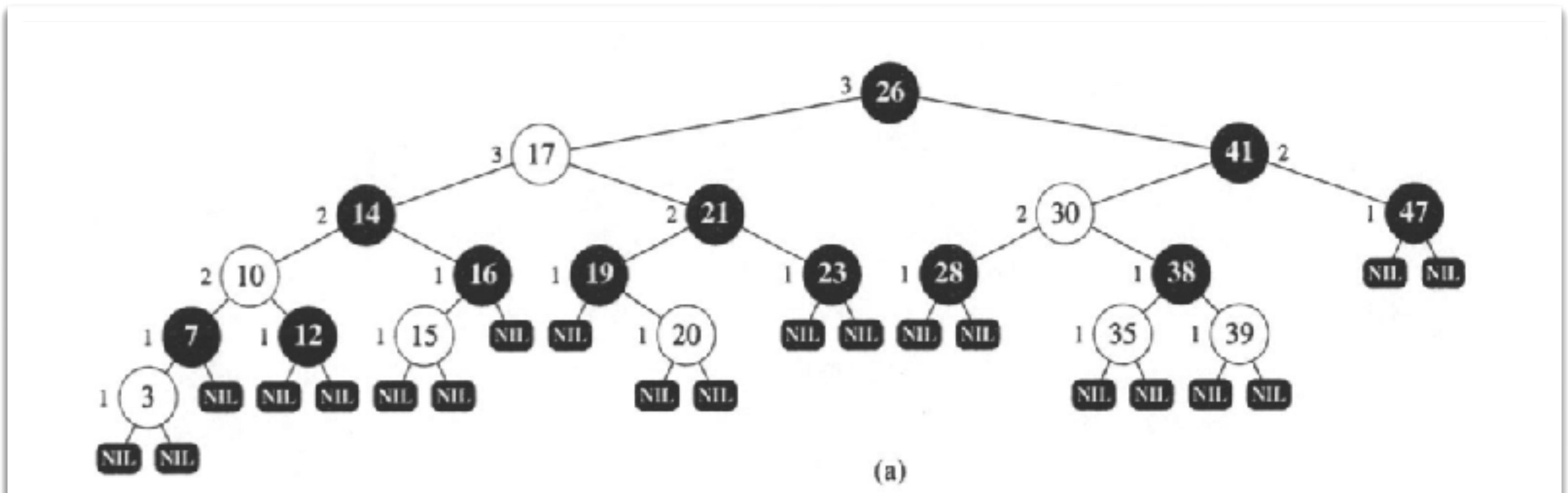
(a)

4.8 Rot-Schwarz-Bäume

Definition 4.12 (Rot-Schwarz-Baum)

Ein binärer Suchbaum heißt Rot-Schwarz-Baum, wenn er die folgenden Eigenschaften erfüllt:

1. Jeder Knoten ist entweder rot oder schwarz.
2. Die Wurzel ist schwarz.

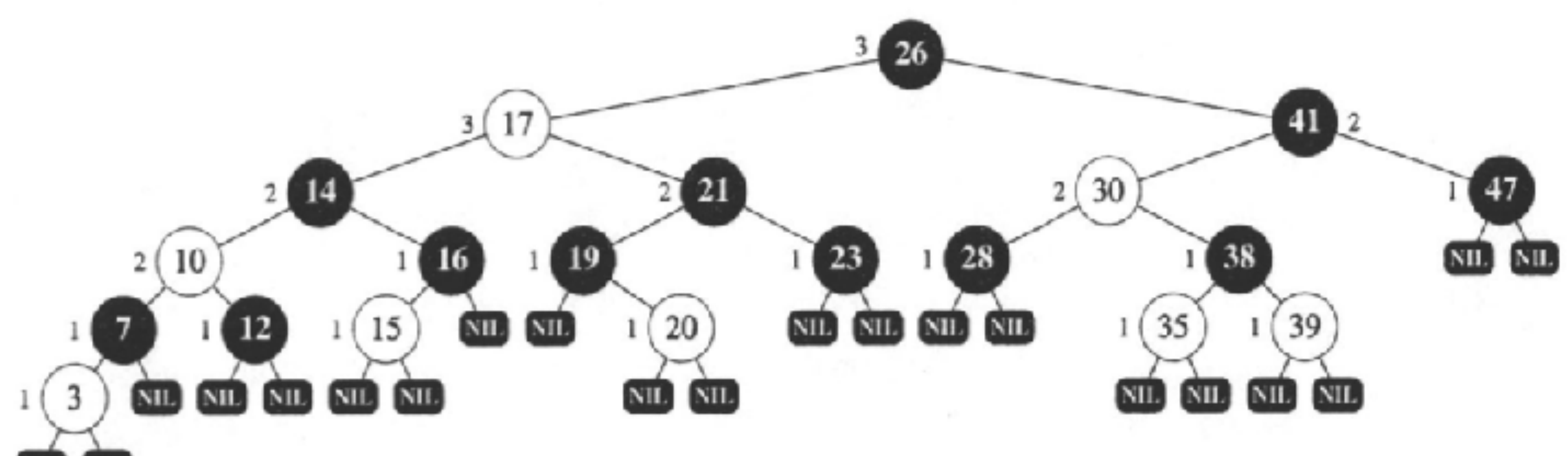


(a)

Definition 4.12 (Rot-Schwarz-Baum)

Ein binärer Suchbaum heißt Rot-Schwarz-Baum, wenn er die folgenden Eigenschaften erfüllt:

1. Jeder Knoten ist entweder rot oder schwarz.
2. Die Wurzel ist schwarz.
3. Jedes Blatt (NIL) ist schwarz.



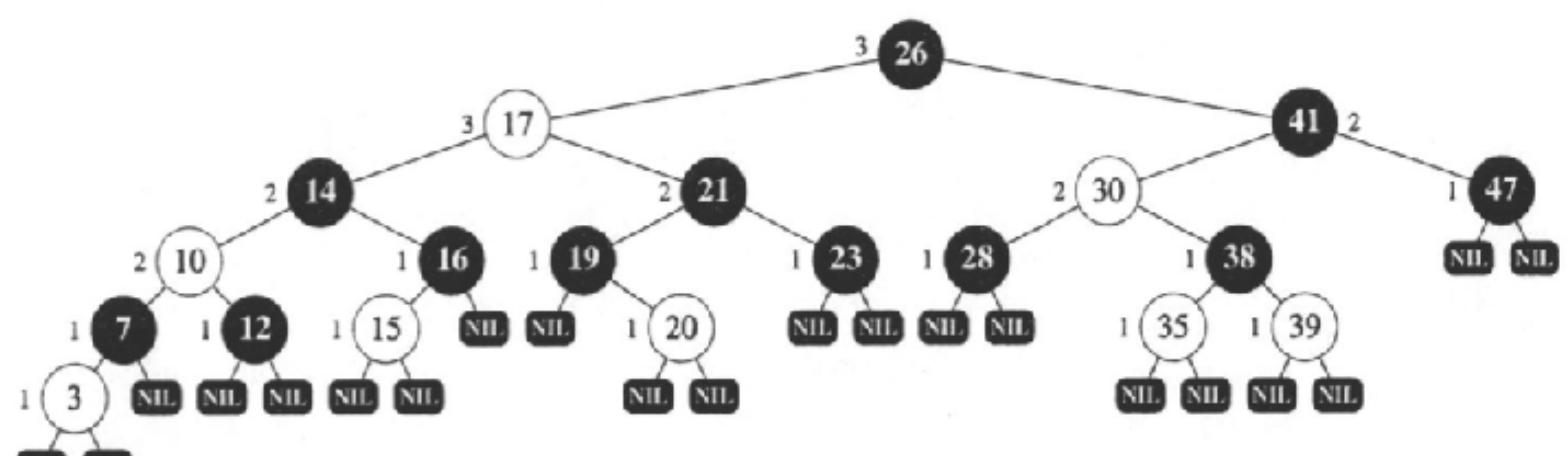
(a)

4.8 Rot-Schwarz-Bäume

Definition 4.12 (Rot-Schwarz-Baum)

Ein binärer Suchbaum heißt Rot-Schwarz-Baum, wenn er die folgenden Eigenschaften erfüllt:

1. Jeder Knoten ist entweder rot oder schwarz.
2. Die Wurzel ist schwarz.
3. Jedes Blatt (NIL) ist schwarz.
4. Wenn ein Knoten rot ist, sind seine beiden Kinder schwarz.

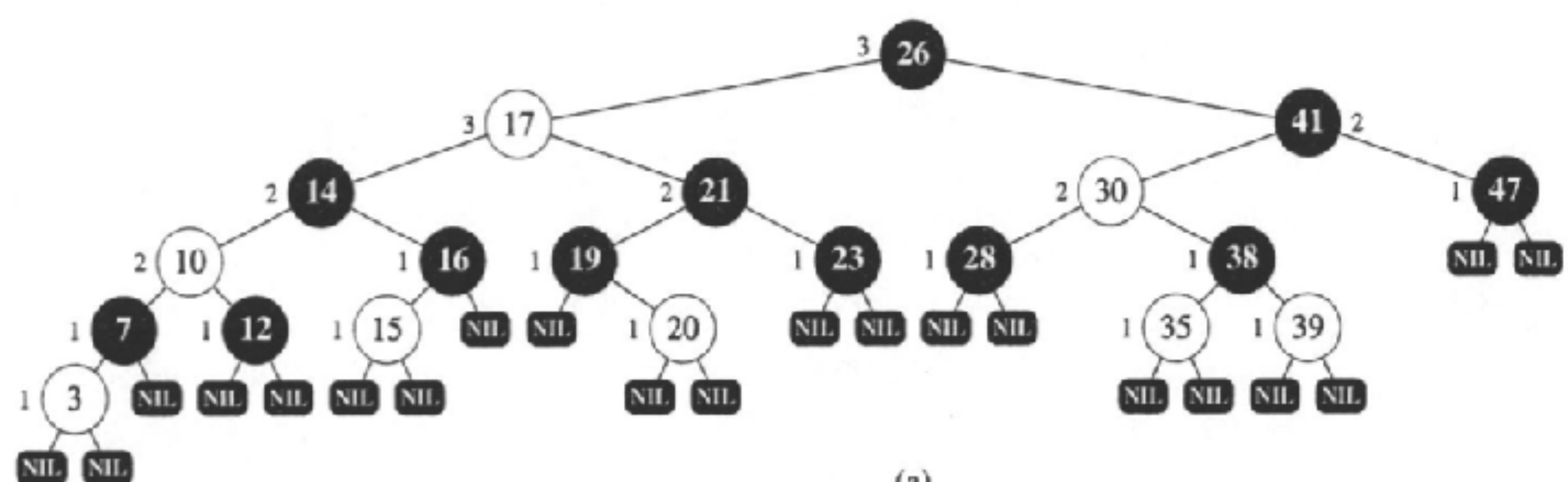


(a)

Definition 4.12 (Rot-Schwarz-Baum)

Ein binärer Suchbaum heißt Rot-Schwarz-Baum, wenn er die folgenden Eigenschaften erfüllt:

1. Jeder Knoten ist entweder rot oder schwarz.
2. Die Wurzel ist schwarz.
3. Jedes Blatt (NIL) ist schwarz.
4. Wenn ein Knoten rot ist, sind seine beiden Kinder schwarz.
5. Für jeden Knoten enthalten alle Pfade nach unten zu einem Blatt des Teilbaumes die gleiche Anzahl schwarzer Knoten.



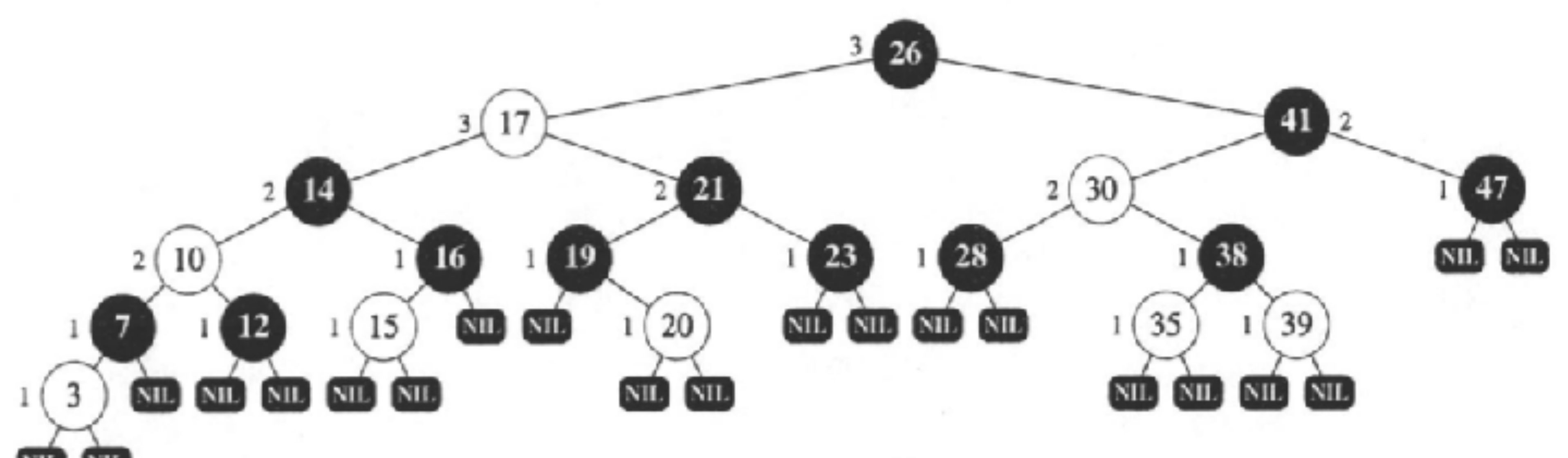
(a)

4.8 Rot-Schwarz-Bäume

Definition 4.12 (Rot-Schwarz-Baum)

Ein binärer Suchbaum heißt Rot-Schwarz-Baum, wenn er die folgenden Eigenschaften erfüllt:

1. Jeder Knoten ist entweder rot oder schwarz.
2. Die Wurzel ist schwarz.
3. Jedes Blatt (NIL) ist schwarz.
4. Wenn ein Knoten rot ist, sind seine beiden Kinder schwarz.
5. Für jeden Knoten enthalten alle Pfade nach unten zu einem Blatt des Teilbaumes die gleiche Anzahl schwarzer Knoten.



(a)

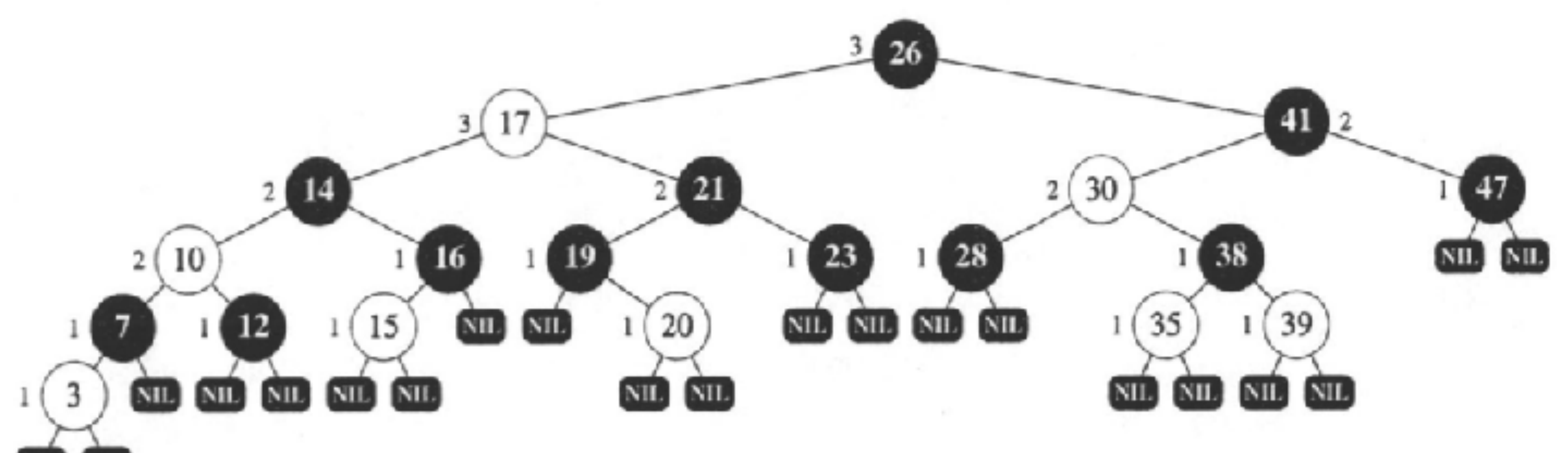
4.8 Rot-Schwarz-Bäume

Definition 4.12 (Rot-Schwarz-Baum)

Schwarz-Höhe!

Ein binärer Suchbaum heißt Rot-Schwarz-Baum, wer folgenden Eigenschaften erfüllt:

1. Jeder Knoten ist entweder rot oder schwarz.
2. Die Wurzel ist schwarz.
3. Jedes Blatt (NIL) ist schwarz.
4. Wenn ein Knoten rot ist, sind seine beiden Kinder schwarz.
5. Für jeden Knoten enthalten alle Pfade nach unten zu einem Blatt des Teilbaumes die gleiche Anzahl schwarzer Knoten.



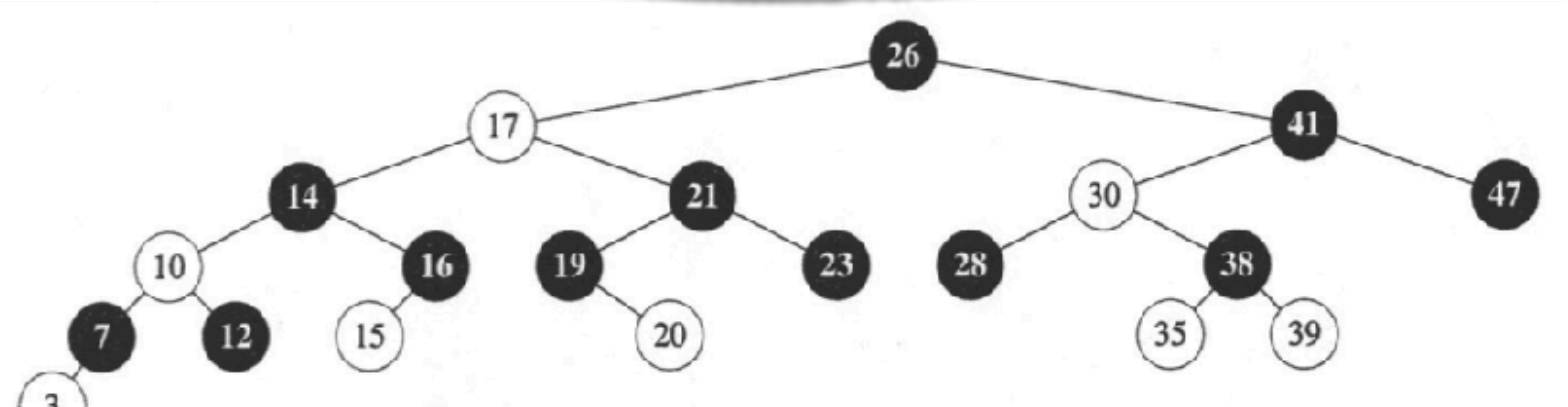
(a)

Definition 4.12 (Rot-Schwarz-Baum)

Schwarz-Höhe!

Ein binärer Suchbaum heißt Rot-Schwarz-Baum, wen
folgenden Eigenschaften erfüllt:

1. Jeder Knoten ist entweder rot oder schwarz.
2. Die Wurzel ist schwarz.
3. Jedes Blatt (NIL) ist schwarz.
4. Wenn ein Knoten rot ist, sind seine beiden Kinder schwarz.
5. Für jeden Knoten enthalten alle Pfade nach unten zu einem Blatt des Teilbaumes die gleiche Anzahl schwarzer Knoten.



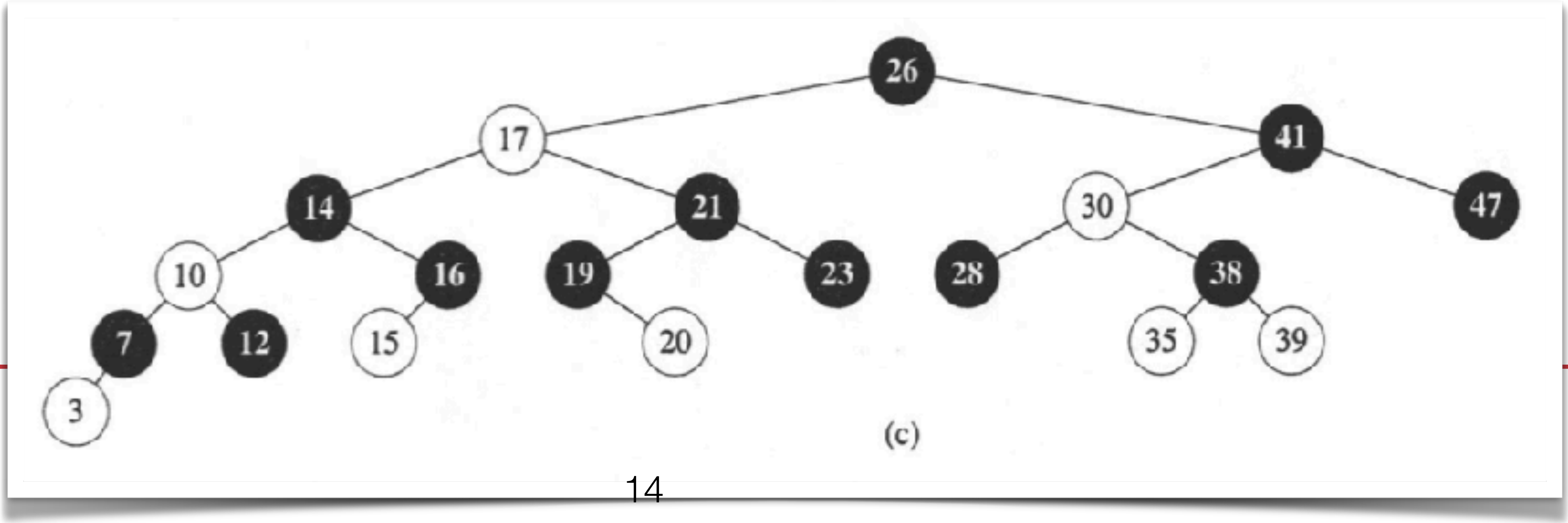
(c)

Rot-Schwarz-Bäume

Warum sorgt dies dafür, dass der Baum balanciert ist?

Rot-Schwarz-Bäume

Warum sorgt dies dafür, dass der Baum balanciert ist?

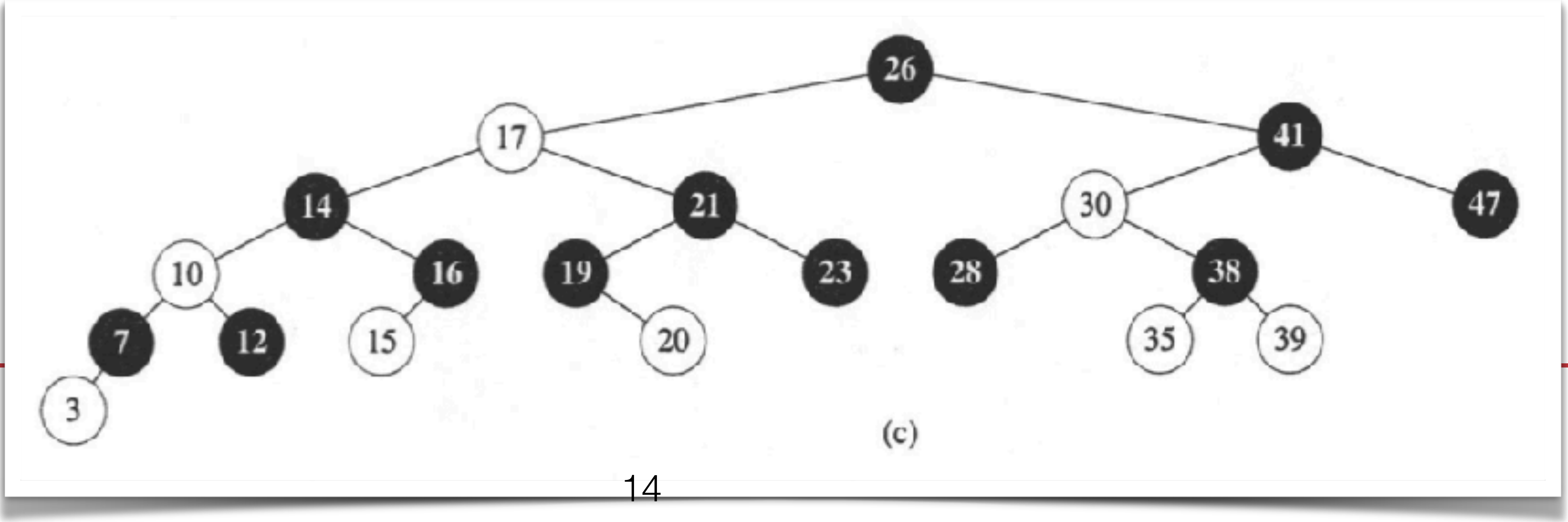


Technische Universität Braunschweig

Rot-Schwarz-Bäume

Warum sorgt dies dafür, dass der Baum balanciert ist?

- Wegen 4. kann es auf keinem Pfad von der Wurzel bis zum Blatt mehr rote als schwarze Knoten geben.

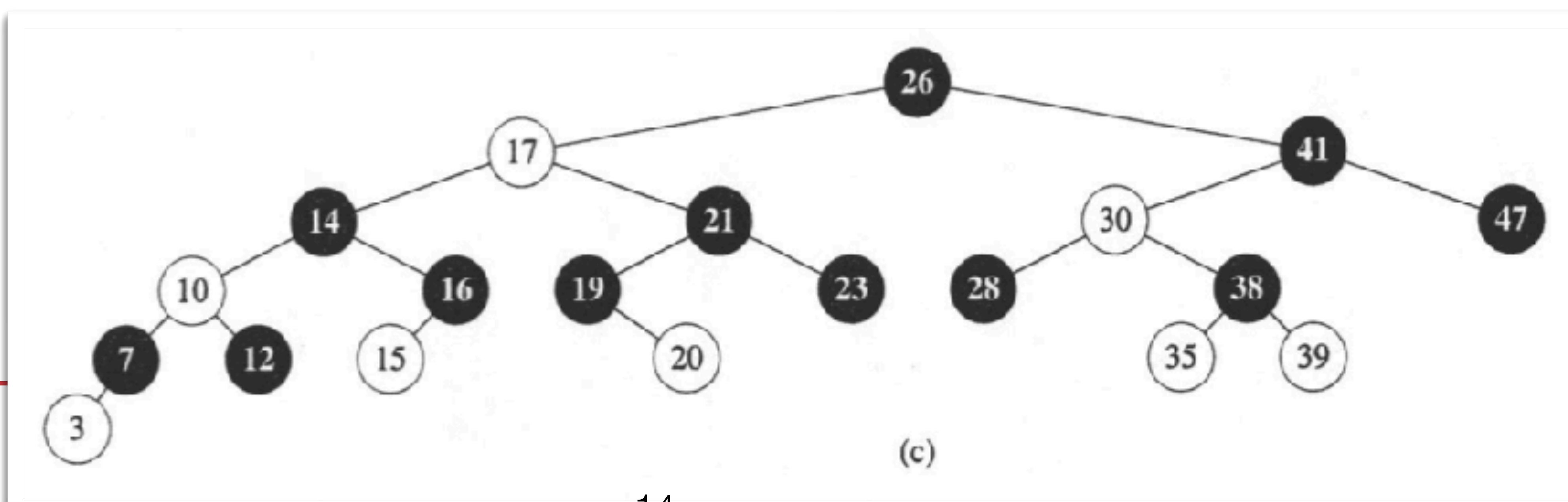


Rot-Schwarz-Bäume

Warum sorgt dies dafür, dass der Baum balanciert ist?

- Wegen 4. kann es auf keinem Pfad von der Wurzel bis zum Blatt mehr rote als schwarze Knoten geben.

Wenn ein Knoten rot ist, sind seine beiden Kinder schwarz.

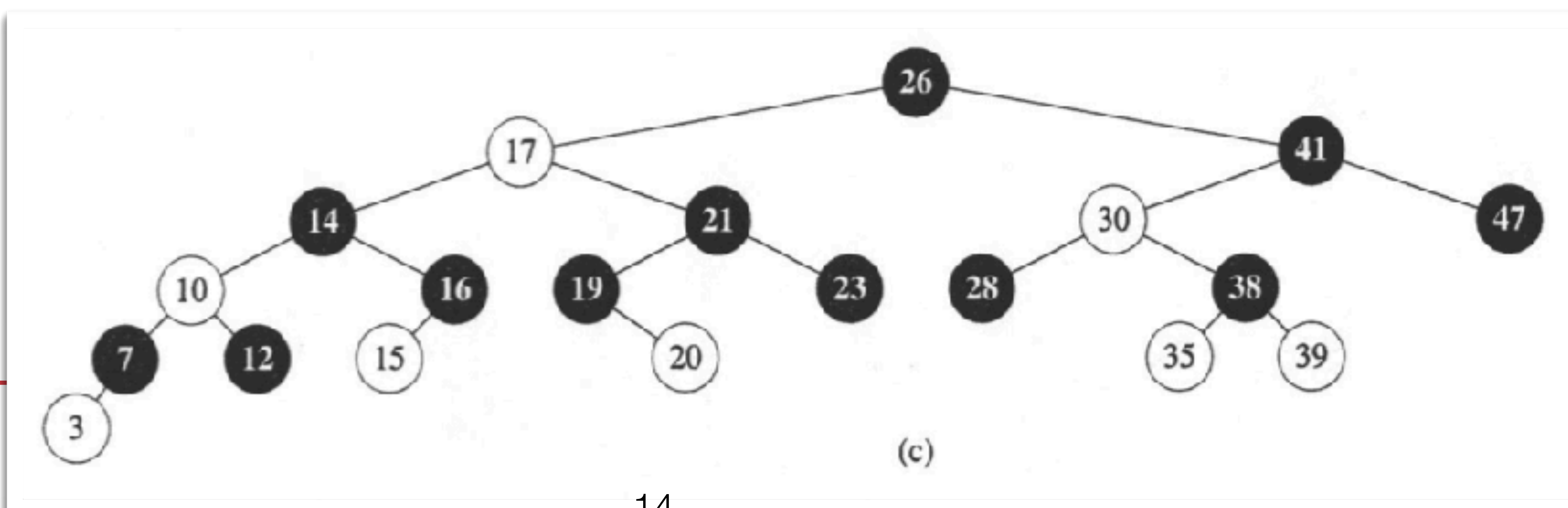


Technische Universität Braunschweig

Rot-Schwarz-Bäume

Warum sorgt dies dafür, dass der Baum balanciert ist?

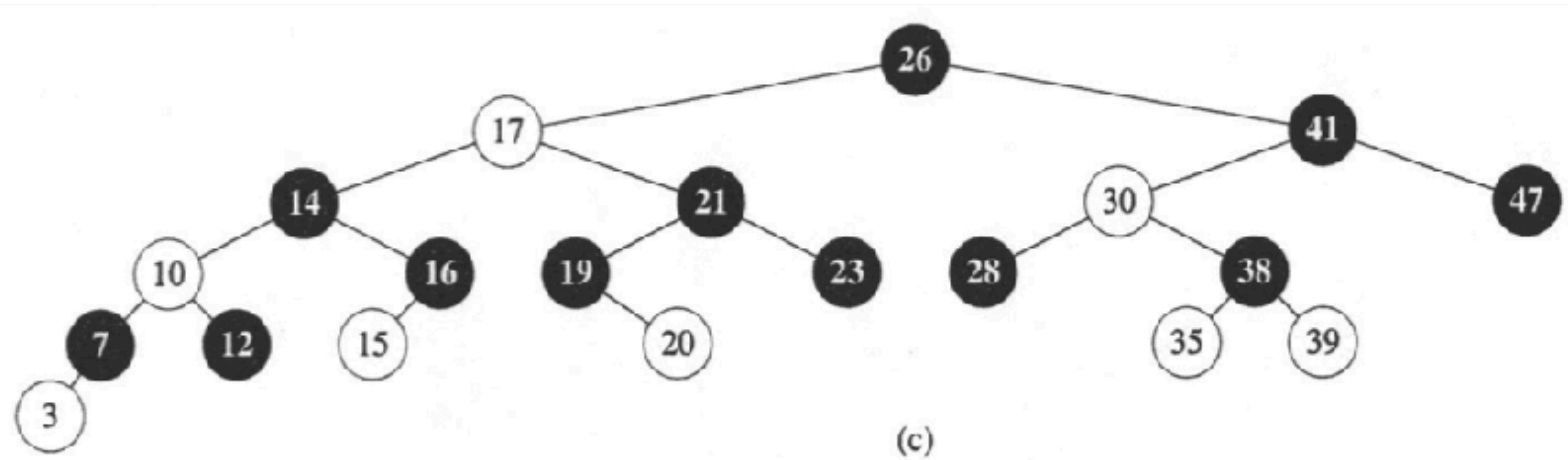
- Wegen 4. kann es auf keinem Pfad von der Wurzel bis zum Blatt mehr rote als schwarze Knoten geben.



Rot-Schwarz-Bäume

Warum sorgt dies dafür, dass der Baum balanciert ist?

- Wegen 4. kann es auf keinem Pfad von der Wurzel bis zum Blatt mehr rote als schwarze Knoten geben.
- Auf dem kürzesten Pfad können nur schwarze Knoten vorkommen.



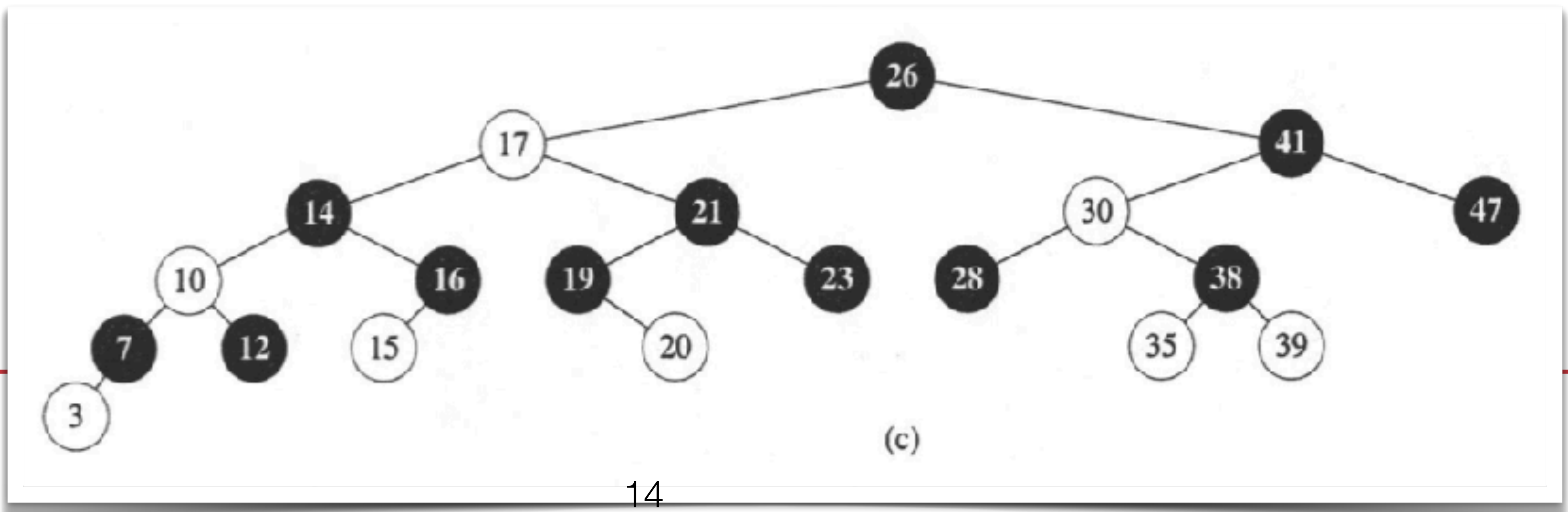
(c)

Technische Universität Braunschweig

Rot-Schwarz-Bäume

Warum sorgt dies dafür, dass der Baum balanciert ist?

- Wegen 4. kann es auf keinem Pfad von der Wurzel bis zum Blatt mehr rote als schwarze Knoten geben.
- Auf dem kürzesten Pfad können nur schwarze Knoten vorkommen.
- Wegen 5. können auf einem Pfad bei dem schwarze und rote Knoten vorkommen, maximal doppelt so viele Knoten liegen, wie auf dem mit nur schwarzen Knoten.



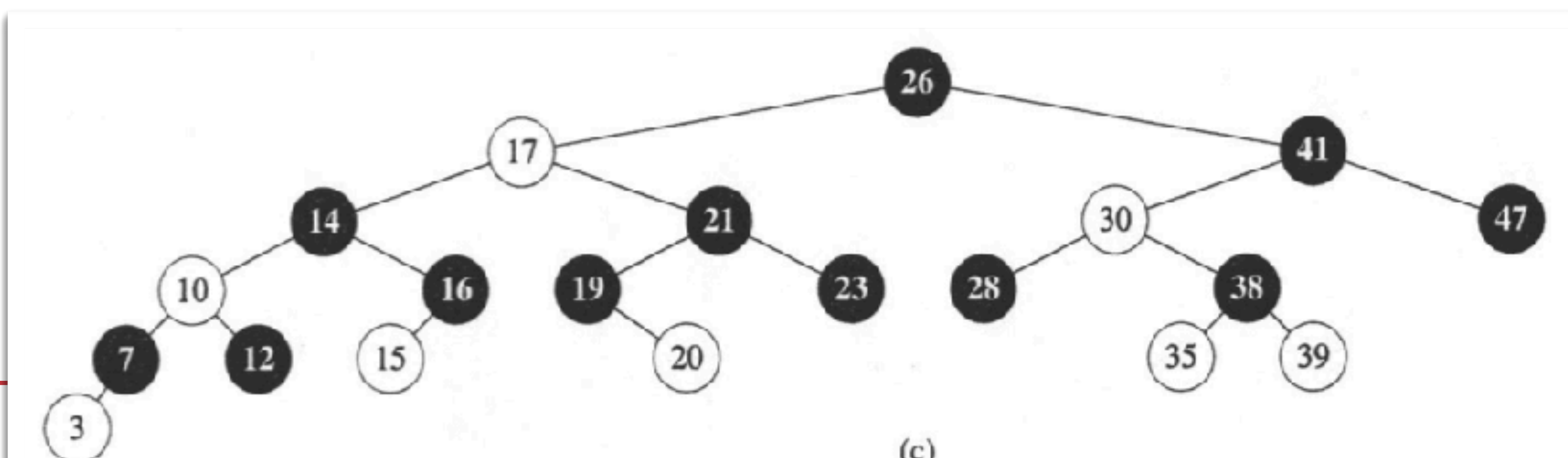
Technische Universität Braunschweig

Rot-Schwarz-Bäume

Warum sorgt dies dafür, dass der Baum balanciert ist?

Für jeden Knoten enthalten alle Pfade, die in diesem Knoten starten und in einem Blatt des Teilbaumes dieses Knotens enden, die gleiche Anzahl schwarzer Knoten.

- Auf dem kürzesten Pfad können nur schwarze Knoten vorkommen.
- Wegen 5. können auf einem Pfad bei dem schwarze und rote Knoten vorkommen, maximal doppelt so viele Knoten liegen, wie auf dem mit nur schwarzen Knoten.

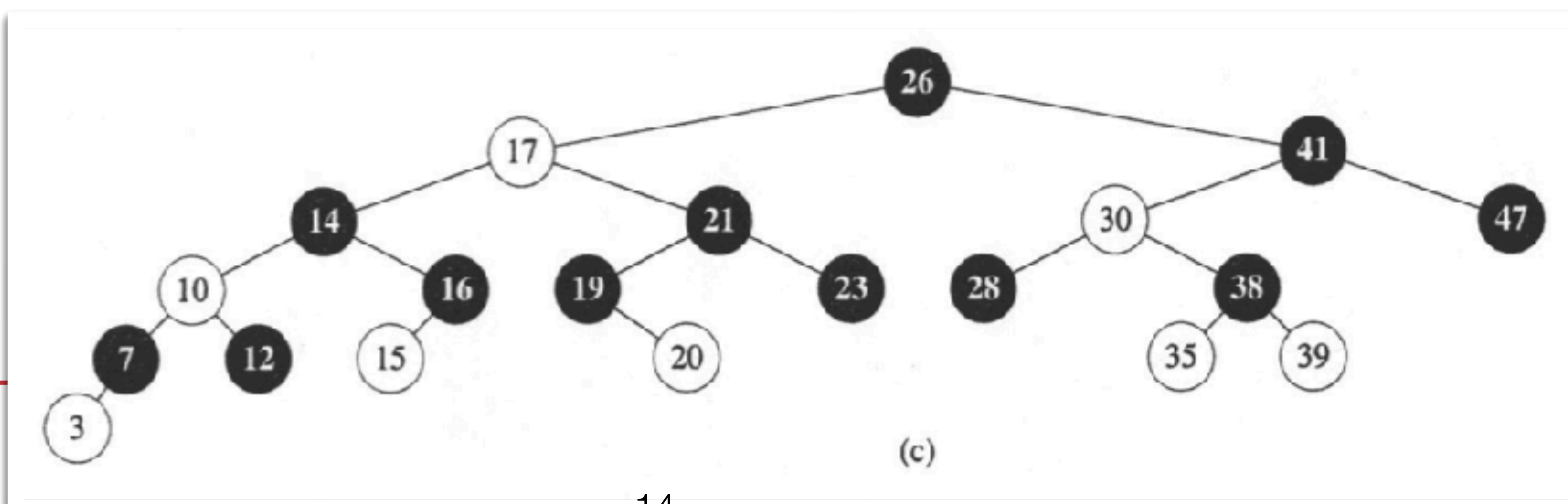


(c)

Rot-Schwarz-Bäume

Warum sorgt dies dafür, dass der Baum balanciert ist?

- Wegen 4. kann es auf keinem Pfad von der Wurzel bis zum Blatt mehr rote als schwarze Knoten geben.
- Auf dem kürzesten Pfad können nur schwarze Knoten vorkommen.
- Wegen 5. können auf einem Pfad bei dem schwarze und rote Knoten vorkommen, maximal doppelt so viele Knoten liegen, wie auf dem mit nur schwarzen Knoten.

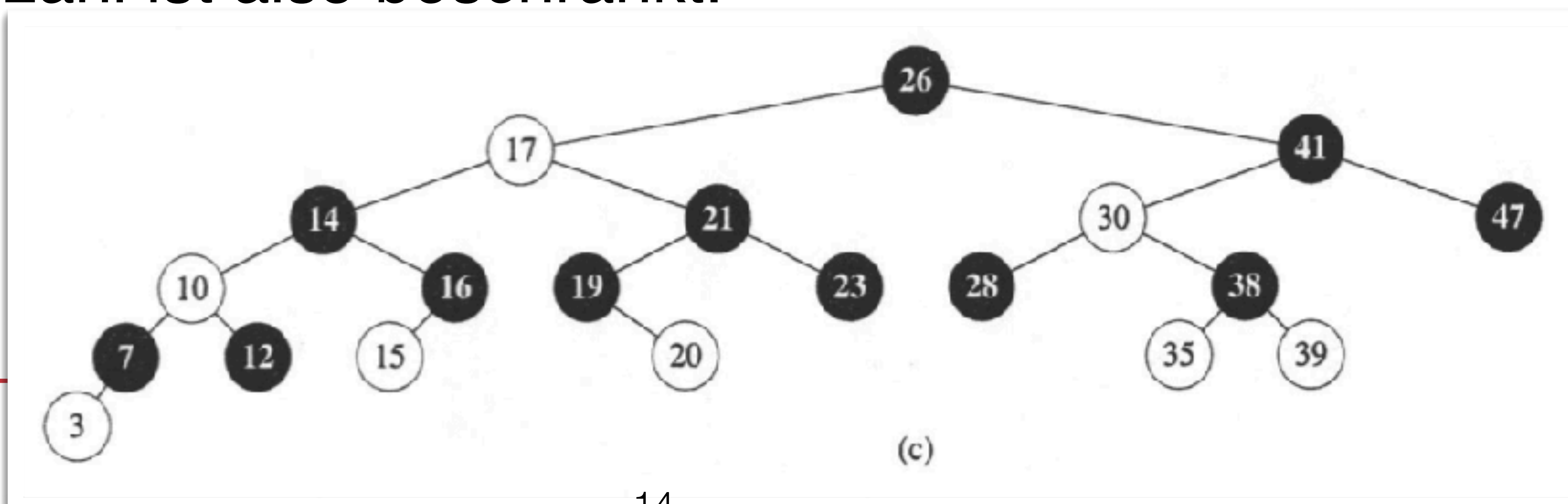


Technische Universität Braunschweig

Rot-Schwarz-Bäume

Warum sorgt dies dafür, dass der Baum balanciert ist?

- Wegen 4. kann es auf keinem Pfad von der Wurzel bis zum Blatt mehr rote als schwarze Knoten geben.
- Auf dem kürzesten Pfad können nur schwarze Knoten vorkommen.
- Wegen 5. können auf einem Pfad bei dem schwarze und rote Knoten vorkommen, maximal doppelt so viele Knoten liegen, wie auf dem mit nur schwarzen Knoten.
- Das Verhältnis der Höhe und dem Logarithmus der Knotenanzahl ist also beschränkt.



Technische Universität Braunschweig

Rot-Schwarz-Bäume

Satz 4.13
Ein Rot-Schwarz-Baum mit n Knoten hat Höhe O(log n).

Rot-Schwarz-Bäume

Satz 4.13
Ein Rot-Schwarz-Baum mit n Knoten hat Höhe O(log n).

Rot-Schwarz-Bäume

Satz 4.13
Ein Rot-Schwarz-Baum mit n Knoten hat Höhe O(log n).

Beweis
Durch Induktion: Jeder Teilbaum zu einem Knoten v mit der Schwarz-Höhe bh(v) hat mindestens $2^{bh(v)} - 1$ innere Knoten.

Rot-Schwarz-Bäume

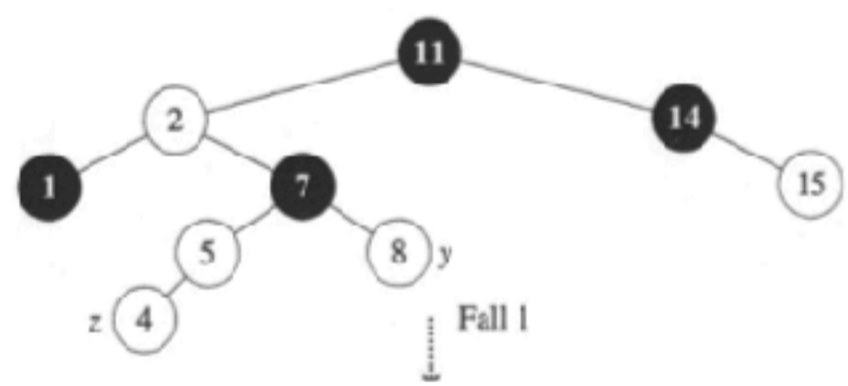
Satz 4.13
Ein Rot-Schwarz-Baum mit n Knoten hat Höhe O(log n).

Beweis
Durch Induktion: Jeder Teilbaum zu einem Knoten v mit der Schwarz-Höhe bh(v) hat mindestens $2^{bh(v)} - 1$ innere Knoten.

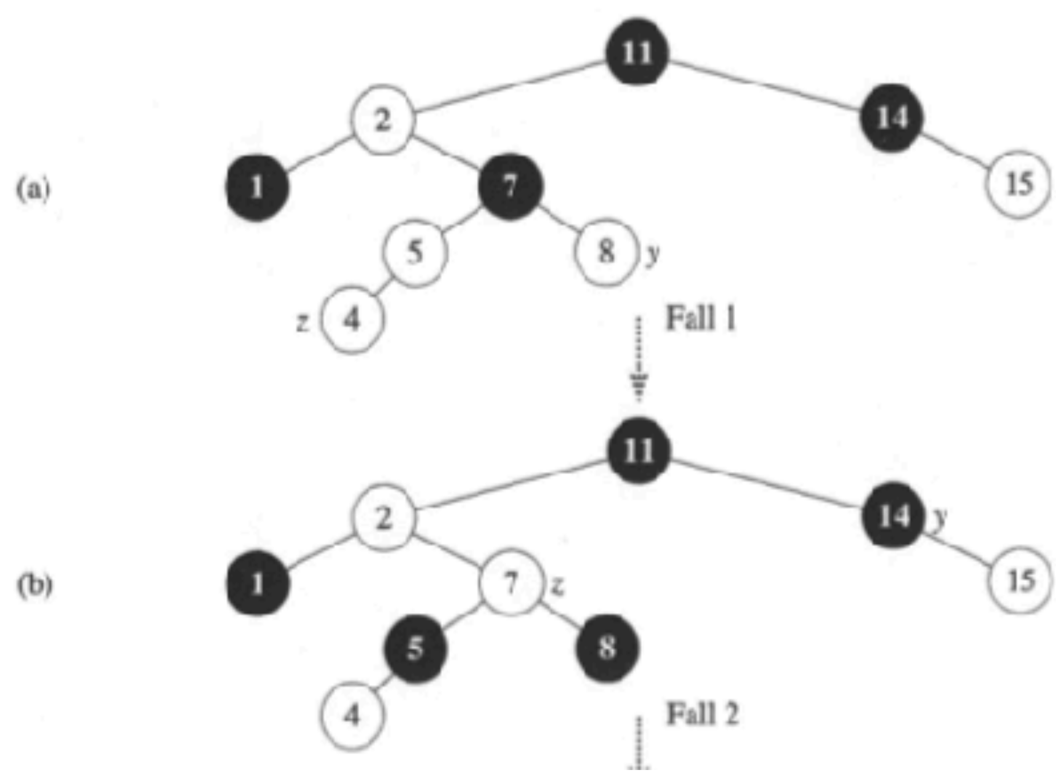
Induktion als Übung!

Rot-Schwarz-Bäume — Einfügen

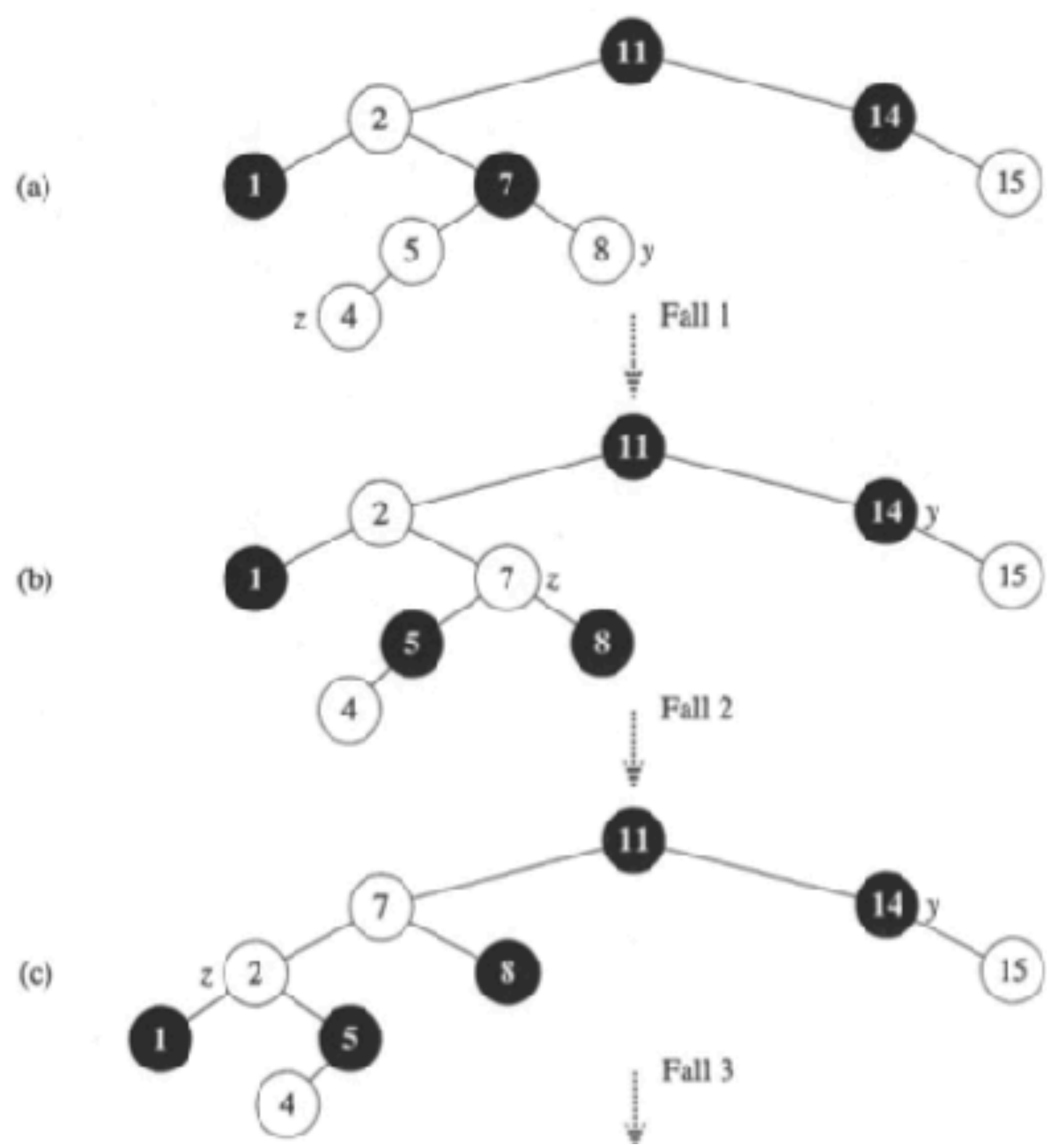
(a)



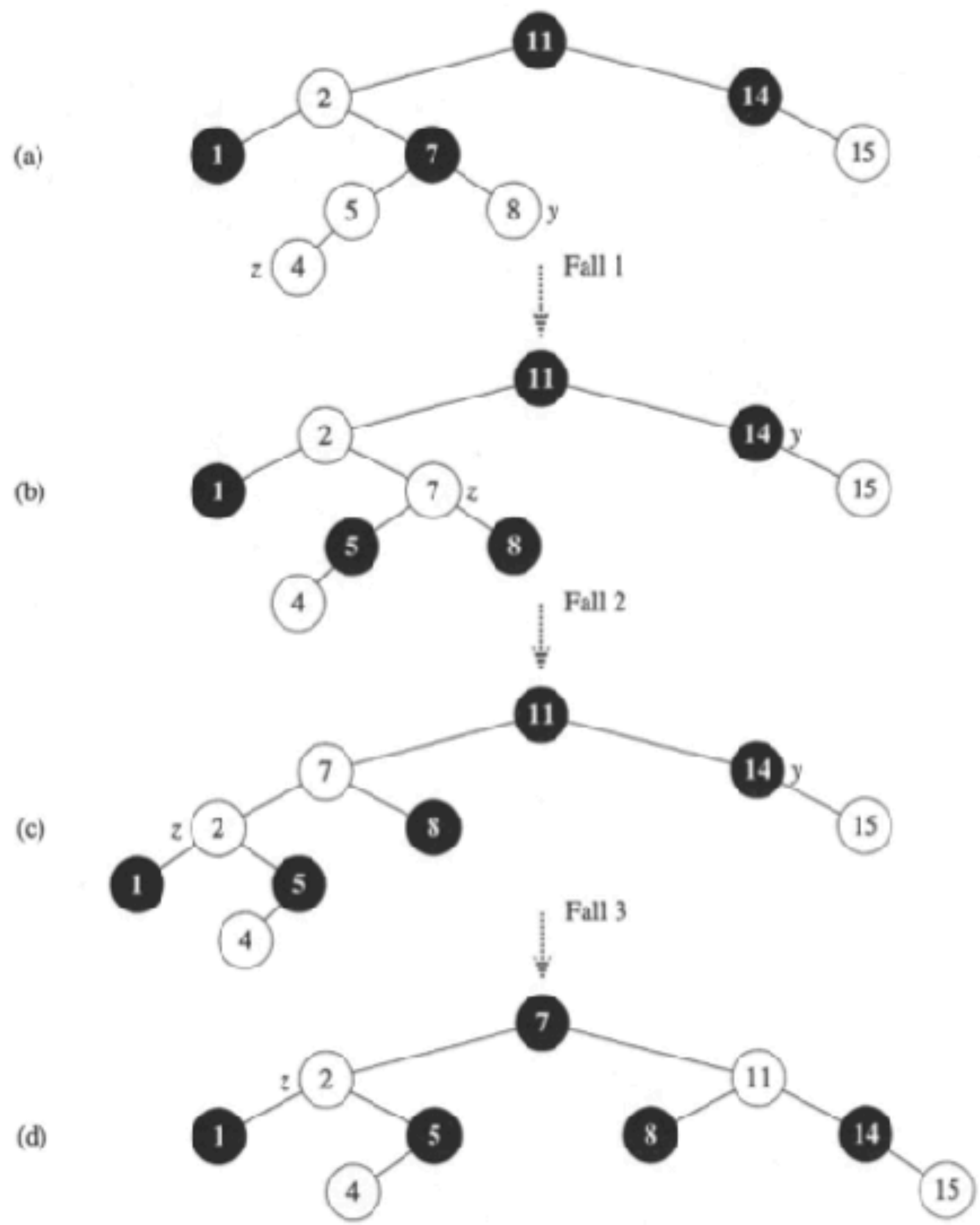
Rot-Schwarz-Bäume — Einfügen



Rot-Schwarz-Bäume — Einfügen



Rot-Schwarz-Bäume — Einfügen



Rot-Schwarz-Bäume

Satz 4.14
Ein Rot-Schwarz-Baum benötigt $O(\log n)$ für dynamische Operationen auf Datenmengen mit n Objekten.

Rot-Schwarz-Bäume

Satz 4.14
Ein Rot-Schwarz-Baum benötigt $O(\log n)$ für dynamische Operationen auf Datenmengen mit n Objekten.

(d.h. genauso lange wie ein AVL-Baum)

4.9 B-Bäume

4.9 B-Bäume

4.9 B-Bäume

(Balancierte) Binäre Bäume haben viele gute Eigenschaften. Wie sieht das in der Praxis aus, zum Beispiel für den Einsatz als Index bei Datenbanken?

4.9 B-Bäume

(Balancierte) Binäre Bäume haben viele gute Eigenschaften. Wie sieht das in der Praxis aus, zum Beispiel für den Einsatz als Index bei Datenbanken?

- Gut bei internen Speicher (Hauptspeicher, Cache,…)

4.9 B-Bäume

(Balancierte) Binäre Bäume haben viele gute Eigenschaften. Wie sieht das in der Praxis aus, zum Beispiel für den Einsatz als Index bei Datenbanken?

- Gut bei internen Speicher (Hauptspeicher, Cache,…)
- Sehr schlecht bei externem Speicher (HDDs)

4.9 B-Bäume

(Balancierte) Binäre Bäume haben viele gute Eigenschaften. Wie sieht das in der Praxis aus, zum Beispiel für den Einsatz als Index bei Datenbanken?

- Gut bei internen Speicher (Hauptspeicher, Cache,…)
- Sehr schlecht bei externem Speicher (HDDs)

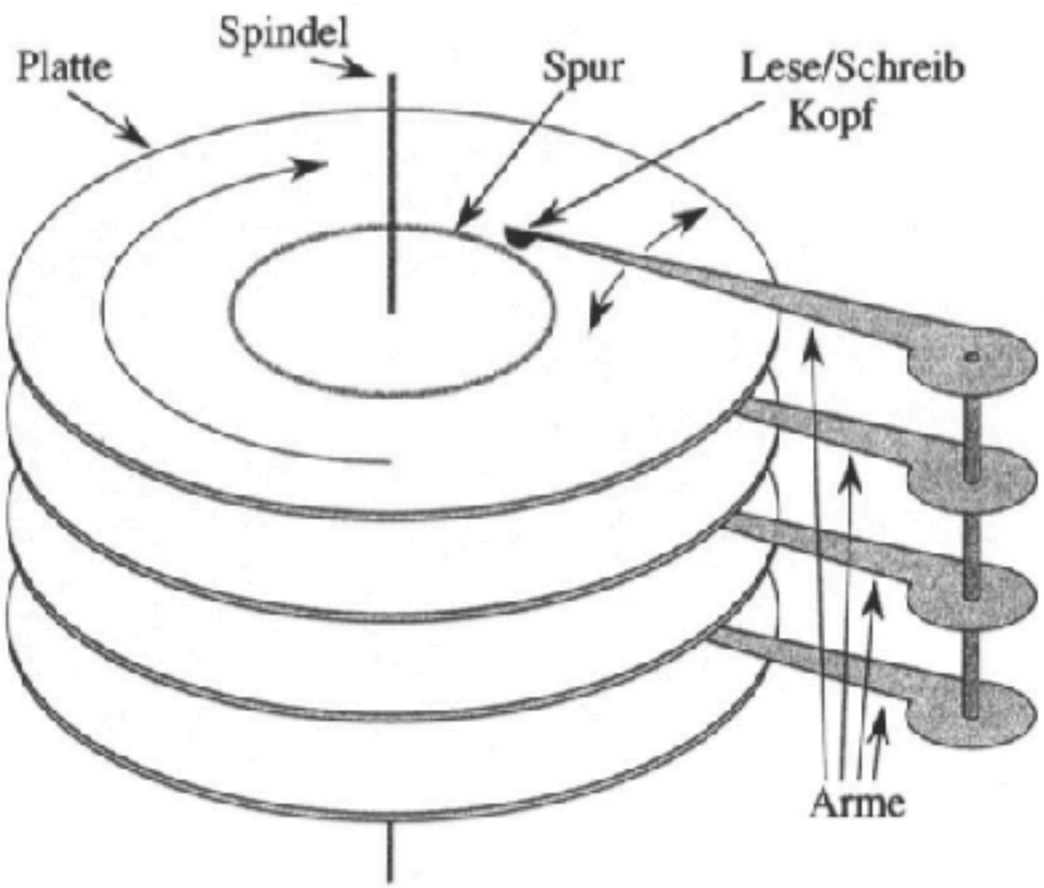
Die Knoten werden hintereinander auf die Platte geschrieben. Wenn der Baum groß ist, muss man ggf. für jeden Knoten einen neuen Block lesen; das dauert!

B-Bäume

B-Bäume

Kontext: Speicherhierarchien!

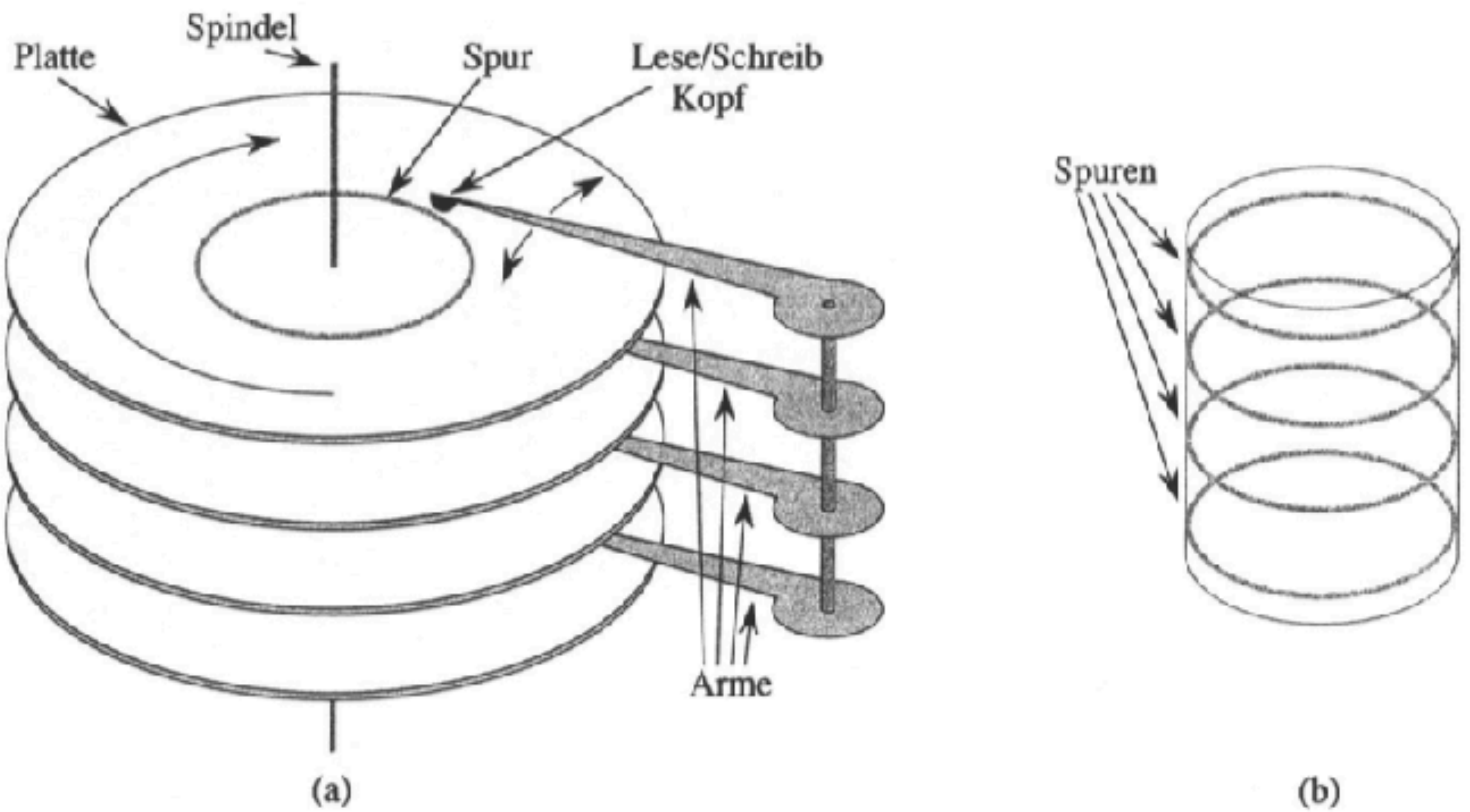
B-Bäume



(a)

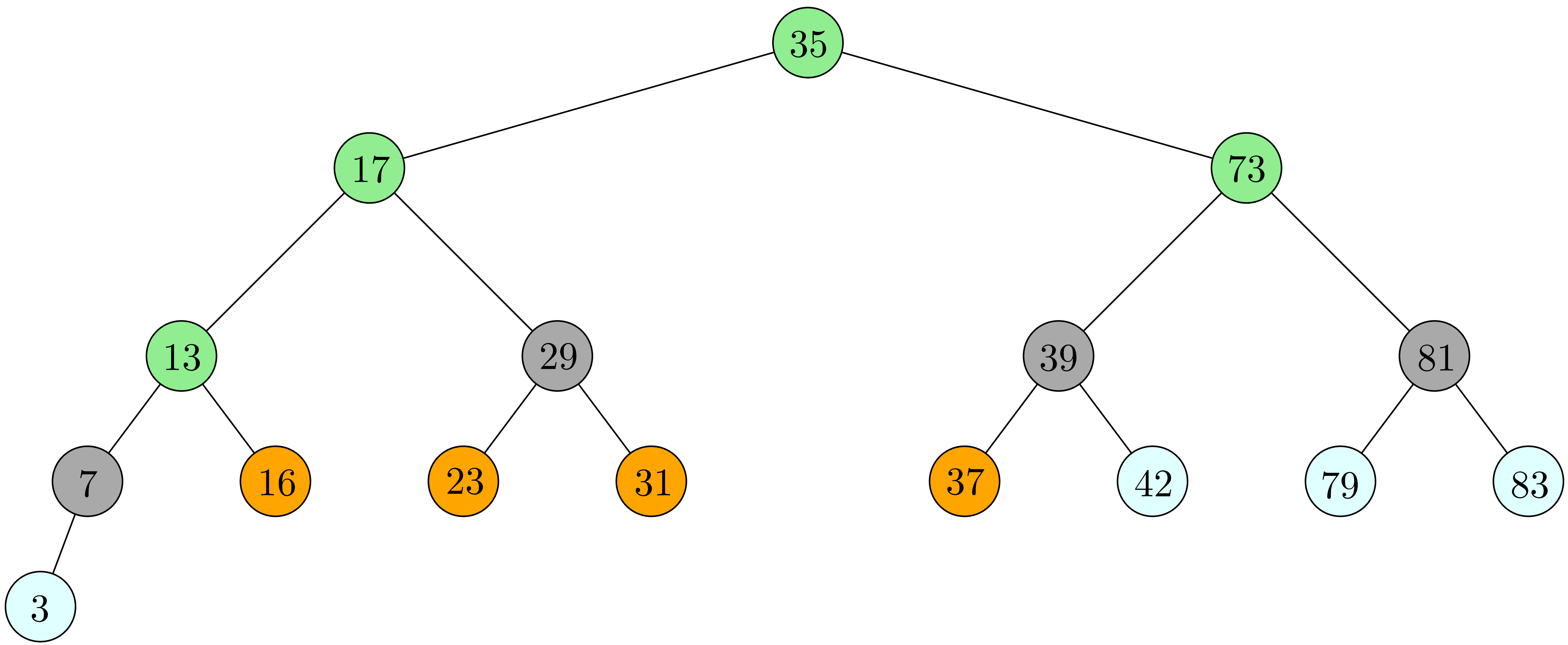
Kontext: Speicherhierarchien!

B-Bäume



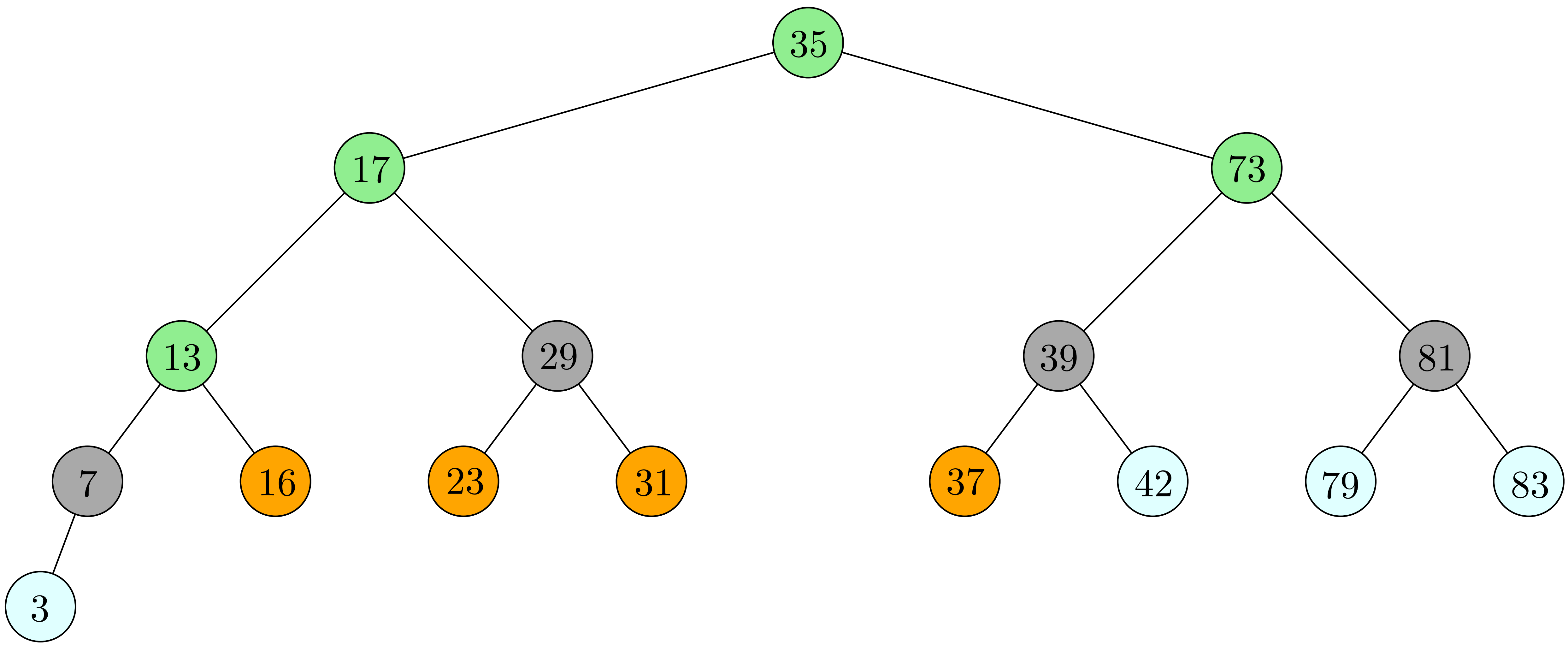
Kontext: Speicherhierarchien!

B-Bäume



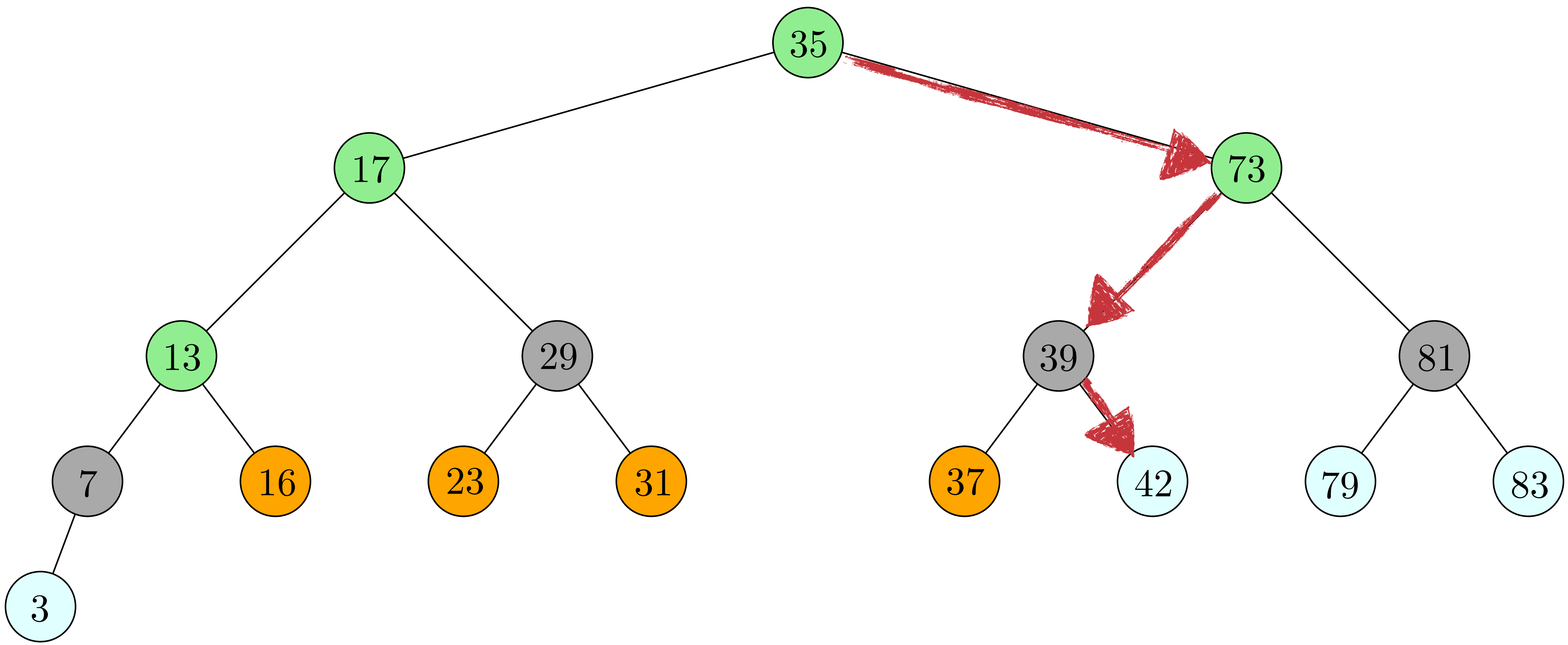
B-Bäume

Angenommen, wir suchen die 42.



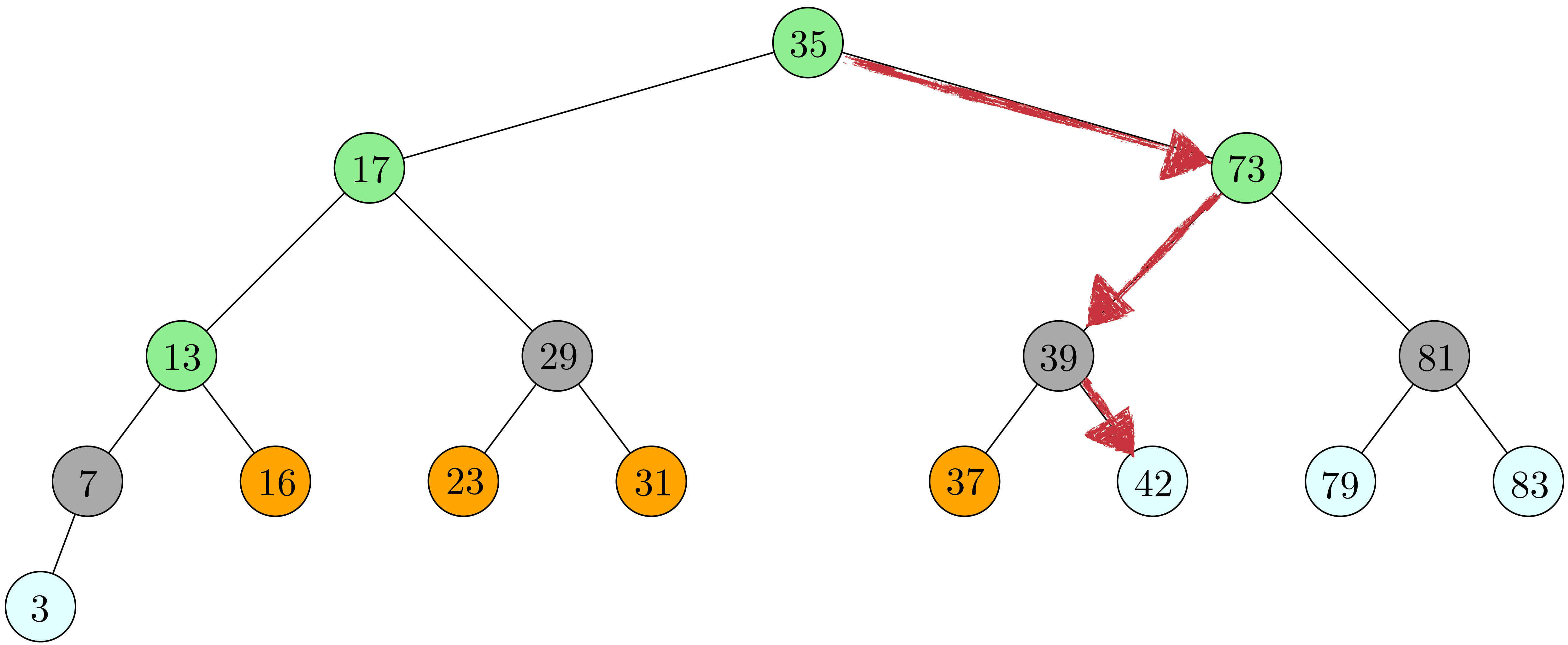
B-Bäume

Angenommen, wir suchen die 42.



B-Bäume

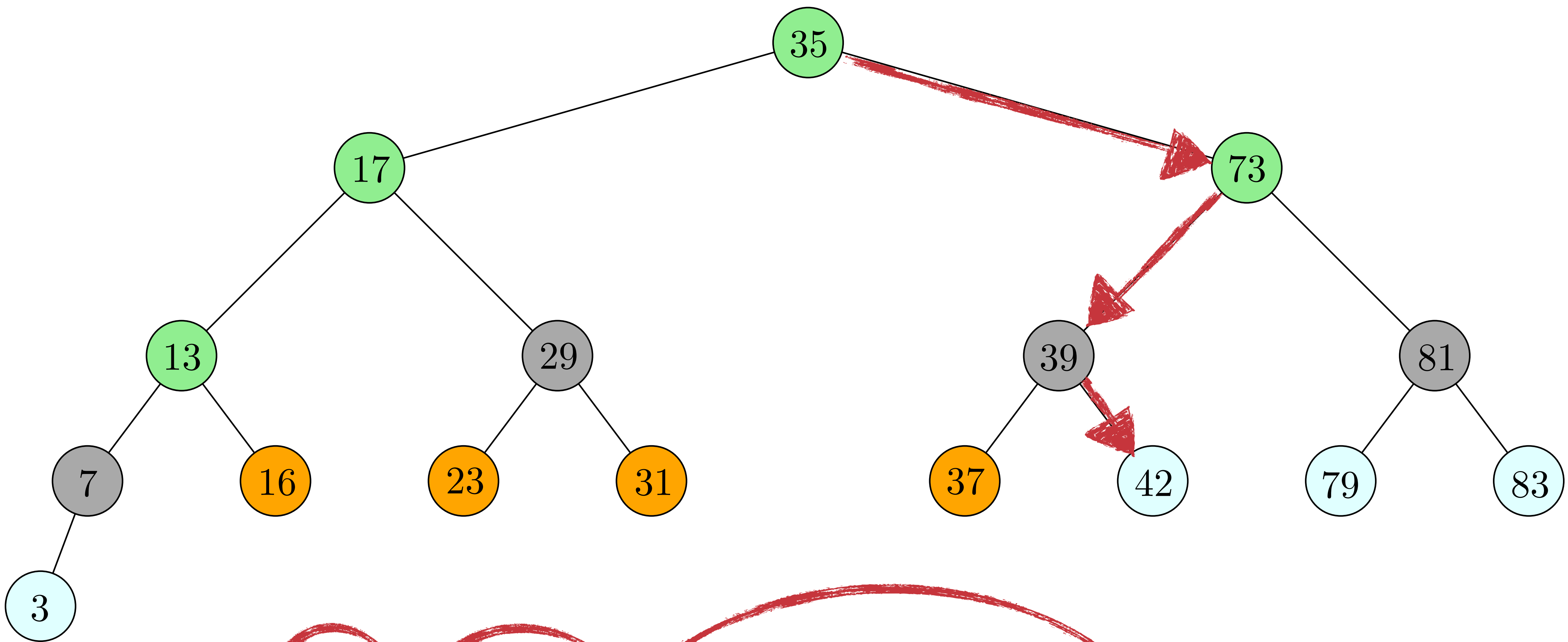
Angenommen, wir suchen die 42.



35	17	73	13	29	39	81	7	16	23	31	37	42	79	83	3

B-Bäume

Angenommen, wir suchen die 42.



35	17	73	13	29	39	81	7	16	23	31	37	42	79	83	3

B-Bäume

- Konzepte von binären Suchbäumen übernehmen

B-Bäume

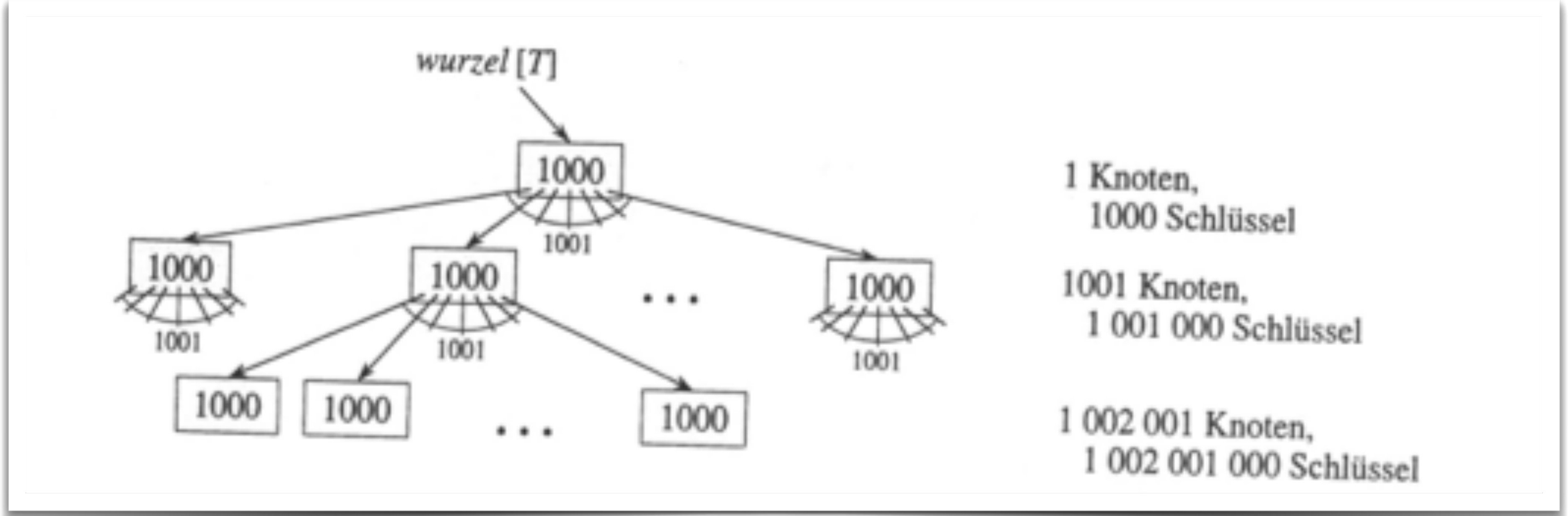
- Konzepte von binären Suchbäumen übernehmen
  - balancieren

B-Bäume

- Konzepte von binären Suchbäumen übernehmen
  - balancieren
  - …

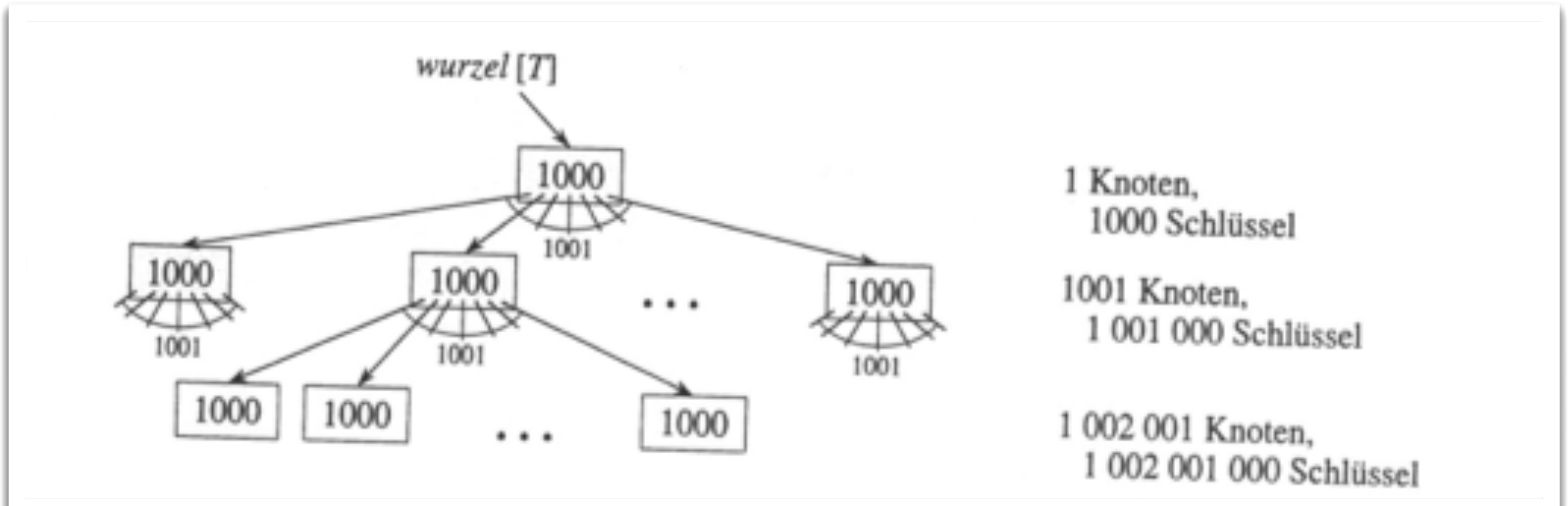
B-Bäume

- Konzepte von binären Suchbäumen übernehmen
  - balancieren
  - …



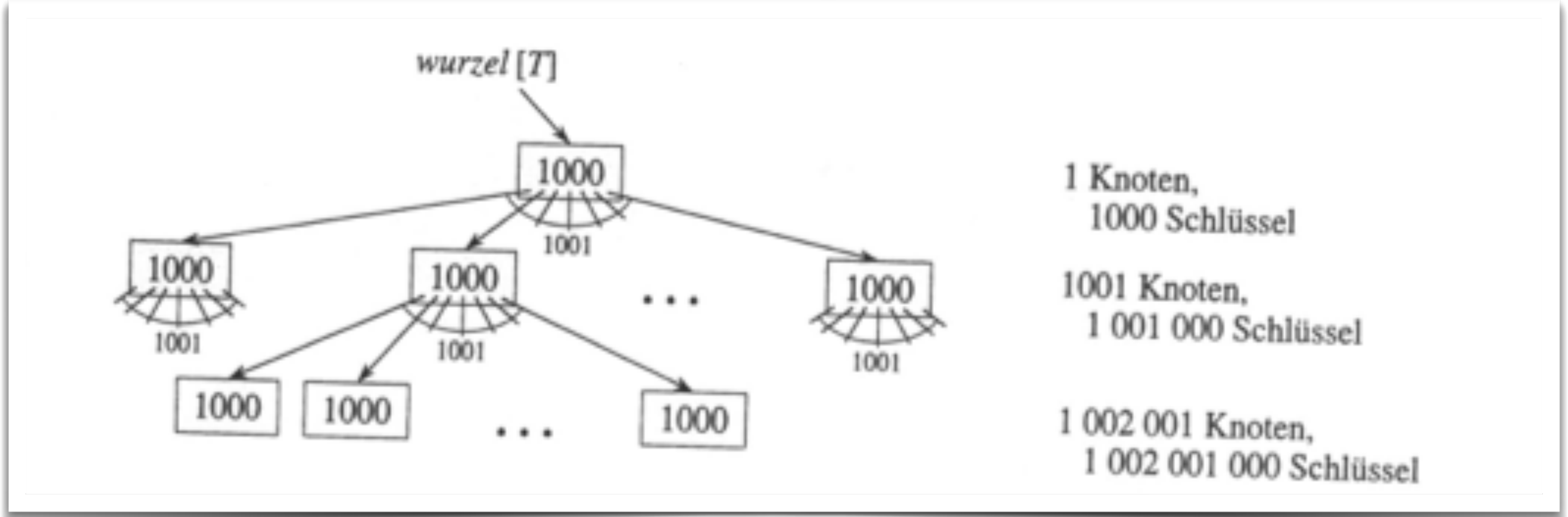
B-Bäume

- Konzepte von binären Suchbäumen übernehmen
  - balancieren
  - …
- Für den Einsatz auf HDDs optimieren



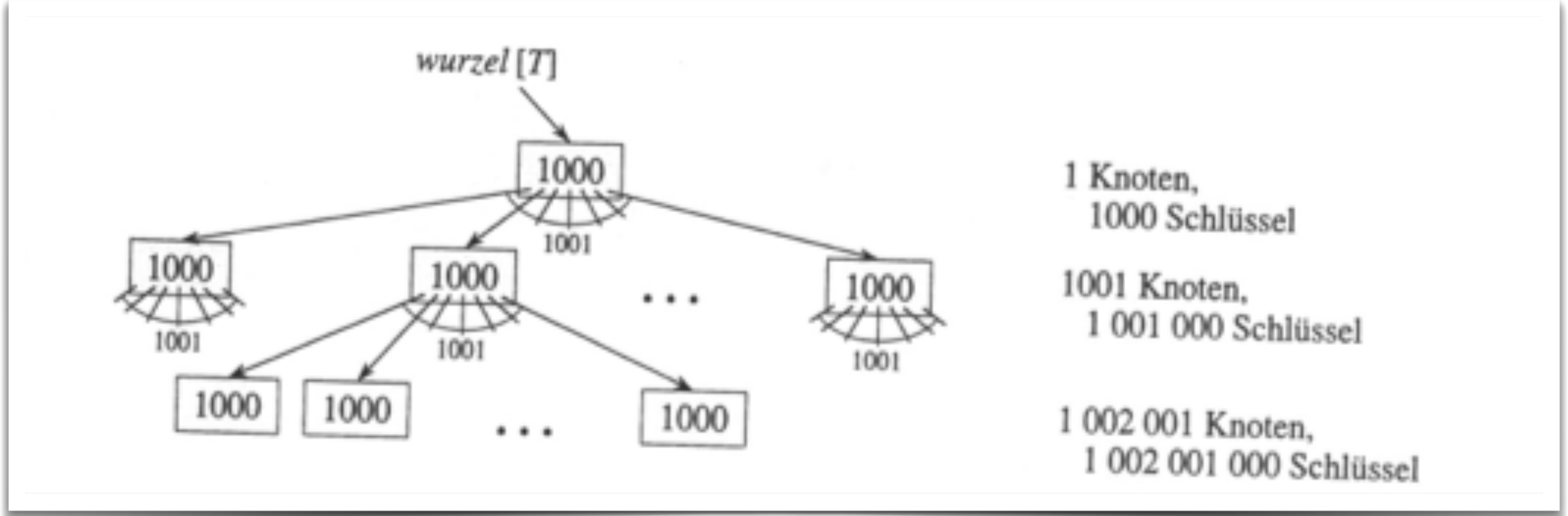
B-Bäume

- Konzepte von binären Suchbäumen übernehmen
  - balancieren
  - …
- Für den Einsatz auf HDDs optimieren
  - Höhe des Baumes minimieren



B-Bäume

- Konzepte von binären Suchbäumen übernehmen
  - balancieren
  - …
- Für den Einsatz auf HDDs optimieren
  - Höhe des Baumes minimieren
  - mehr Schlüssel pro Knoten

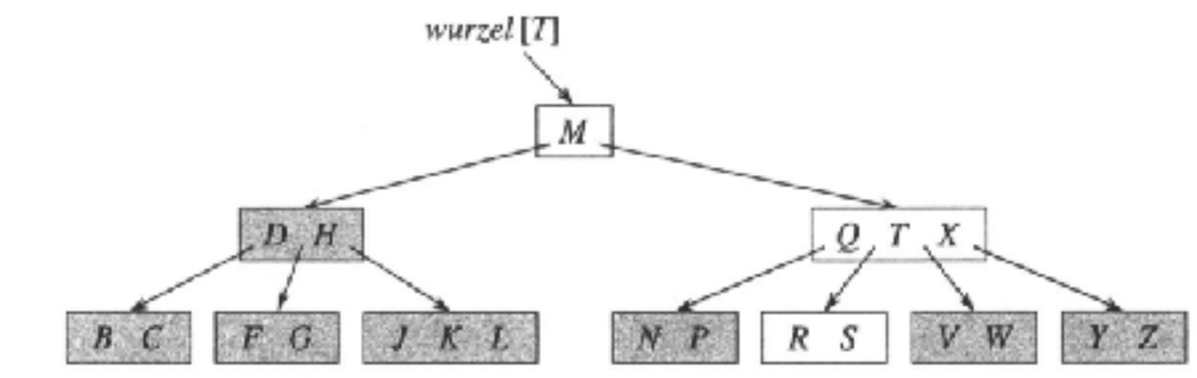


B-Bäume

B-Bäume

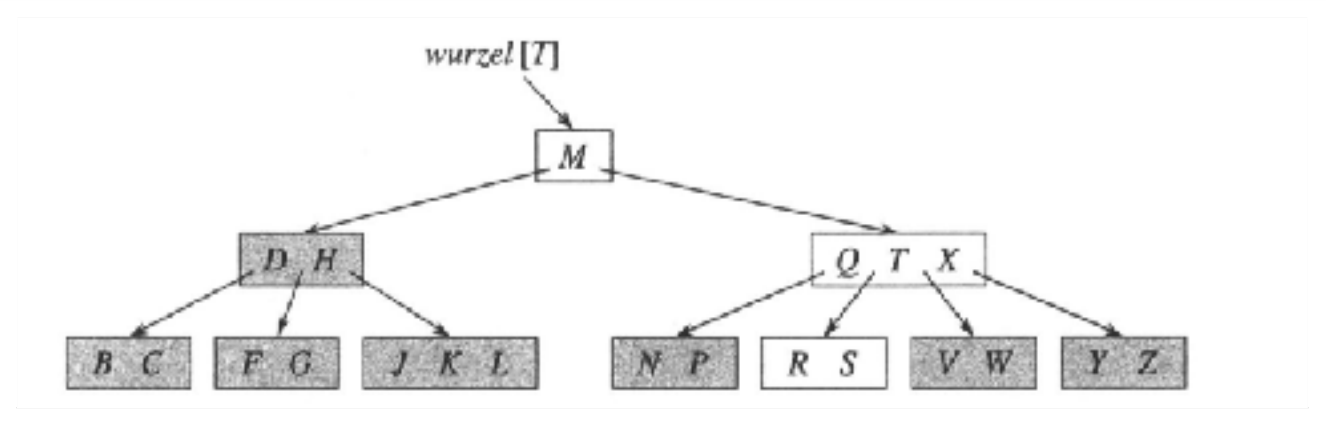
Definition 4.13 (B-Baum)

B-Bäume



Definition 4.13 (B-Baum)

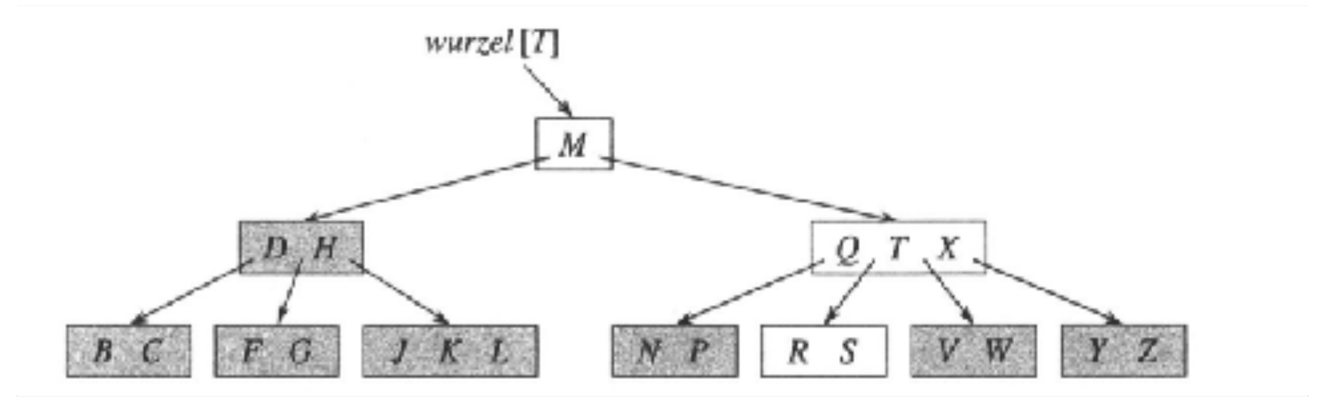
B-Bäume



Definition 4.13 (B-Baum)

Ein B-Baum ist ein gerichteter Baum mit den Eigenschaften:

B-Bäume

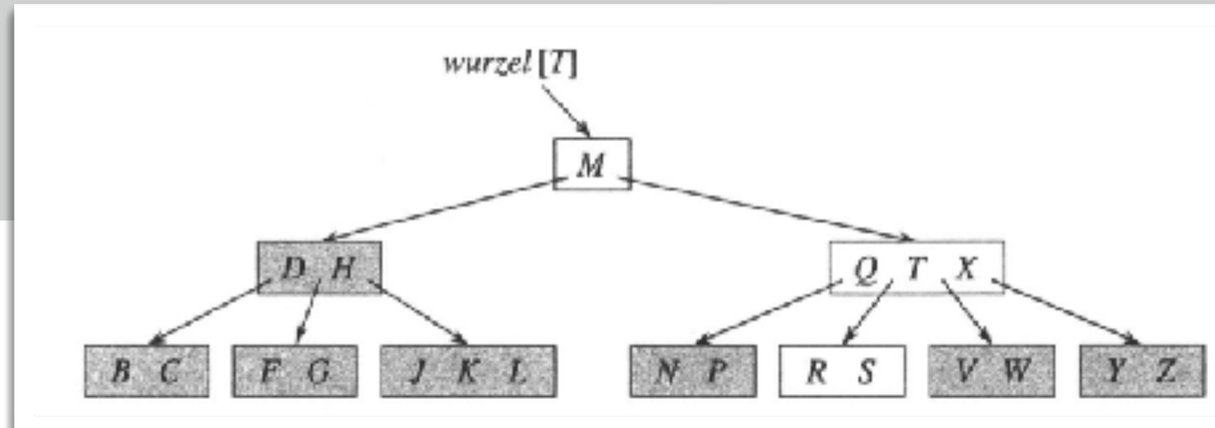


Definition 4.13 (B-Baum)

Ein B-Baum ist ein gerichteter Baum mit den Eigenschaften:

1. Jeder Knoten hat die folgenden Attribute:
   a. die Anzahl n[x] der in x gespeicherten Schlüssel
   b. die sortiert gespeicherten Schlüssel
   c. ein boolscher Wert, der anzeigt, ob x ein Blatt ist

B-Bäume

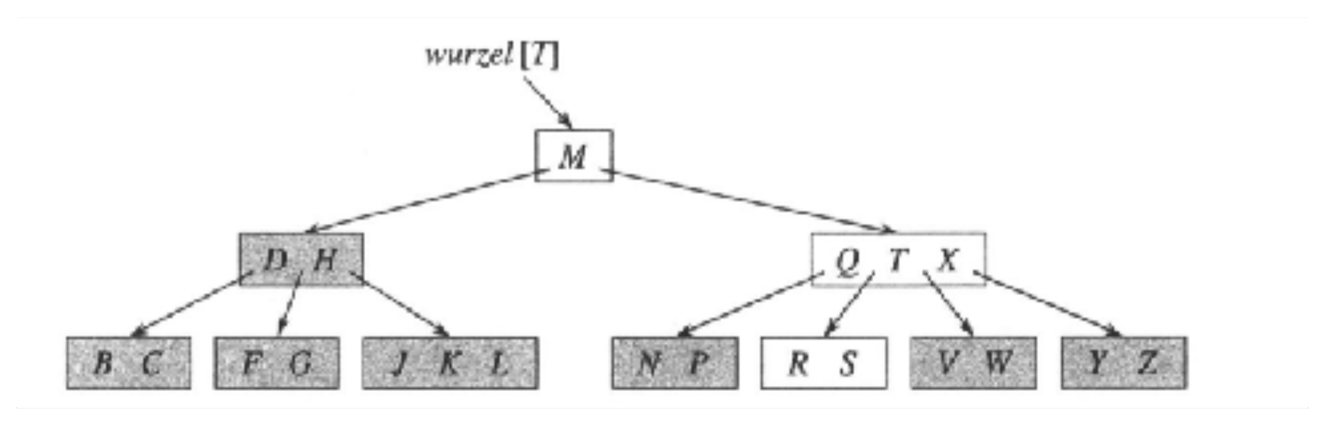


Definition 4.13 (B-Baum)

Ein B-Baum ist ein gerichteter Baum mit den Eigenschaften:

1. Jeder Knoten hat die folgenden Attribute:
   a. die Anzahl n[x] der in x gespeicherten Schlüssel
   b. die sortiert gespeicherten Schlüssel
   c. ein boolscher Wert, der anzeigt, ob x ein Blatt ist
2. Jeder innere Knoten enthält n[x]+1 Zeiger auf seine Kinder

B-Bäume

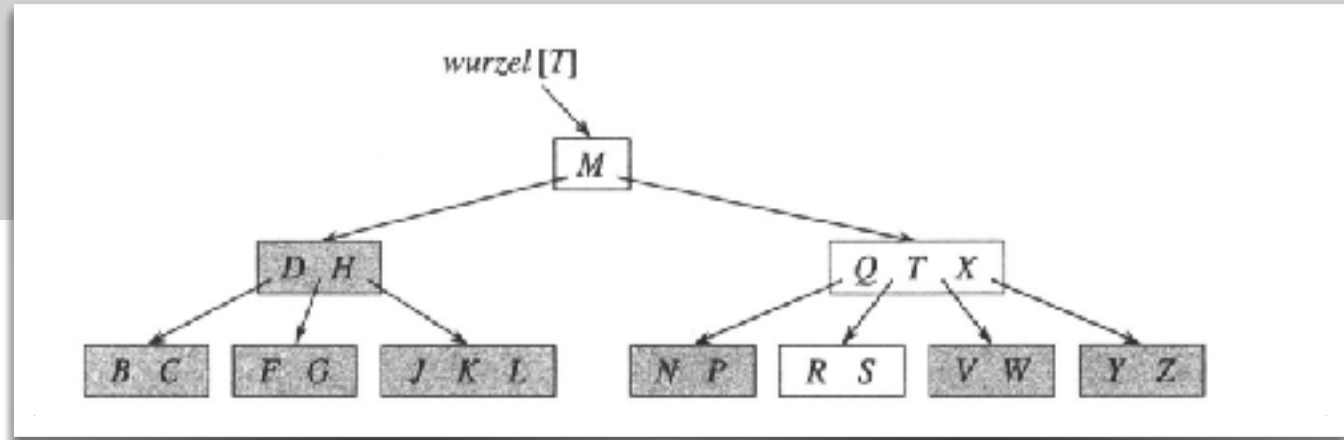


Definition 4.13 (B-Baum)

Ein B-Baum ist ein gerichteter Baum mit den Eigenschaften:

1. Jeder Knoten hat die folgenden Attribute:
   a. die Anzahl n[x] der in x gespeicherten Schlüssel
   b. die sortiert gespeicherten Schlüssel
   c. ein boolscher Wert, der anzeigt, ob x ein Blatt ist
2. Jeder innere Knoten enthält n[x]+1 Zeiger auf seine Kinder
3. Die Schlüssel unterteilen die darunter stehenden Teilbäume nach Größe

B-Bäume

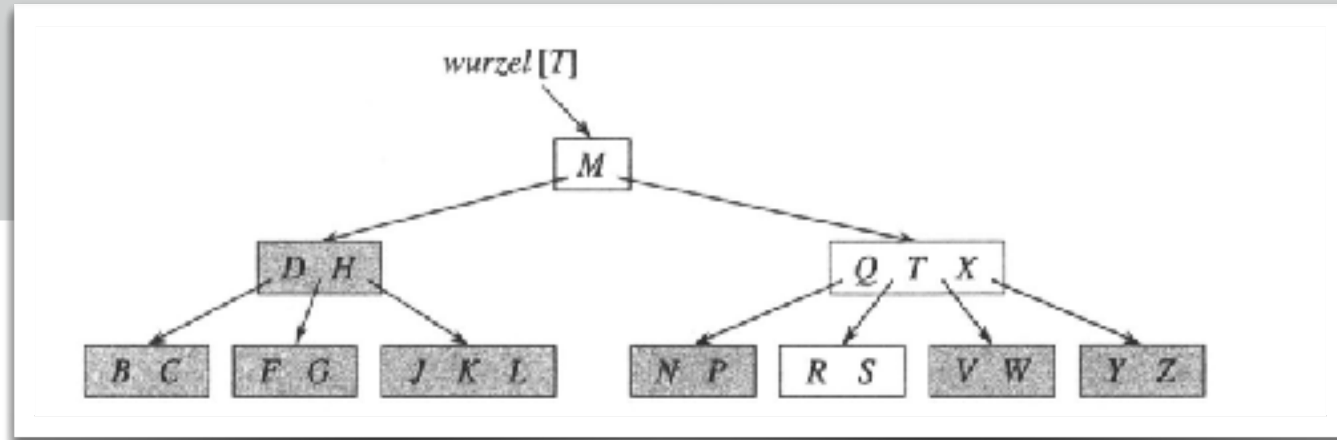


Definition 4.13 (B-Baum)

Ein B-Baum ist ein gerichteter Baum mit den Eigenschaften:

1. Jeder Knoten hat die folgenden Attribute:
   a. die Anzahl n[x] der in x gespeicherten Schlüssel
   b. die sortiert gespeicherten Schlüssel
   c. ein boolscher Wert, der anzeigt, ob x ein Blatt ist
2. Jeder innere Knoten enthält n[x]+1 Zeiger auf seine Kinder
3. Die Schlüssel unterteilen die darunter stehenden Teilbäume nach Größe
4. Alle Blätter haben gleiche Tiefe

B-Bäume

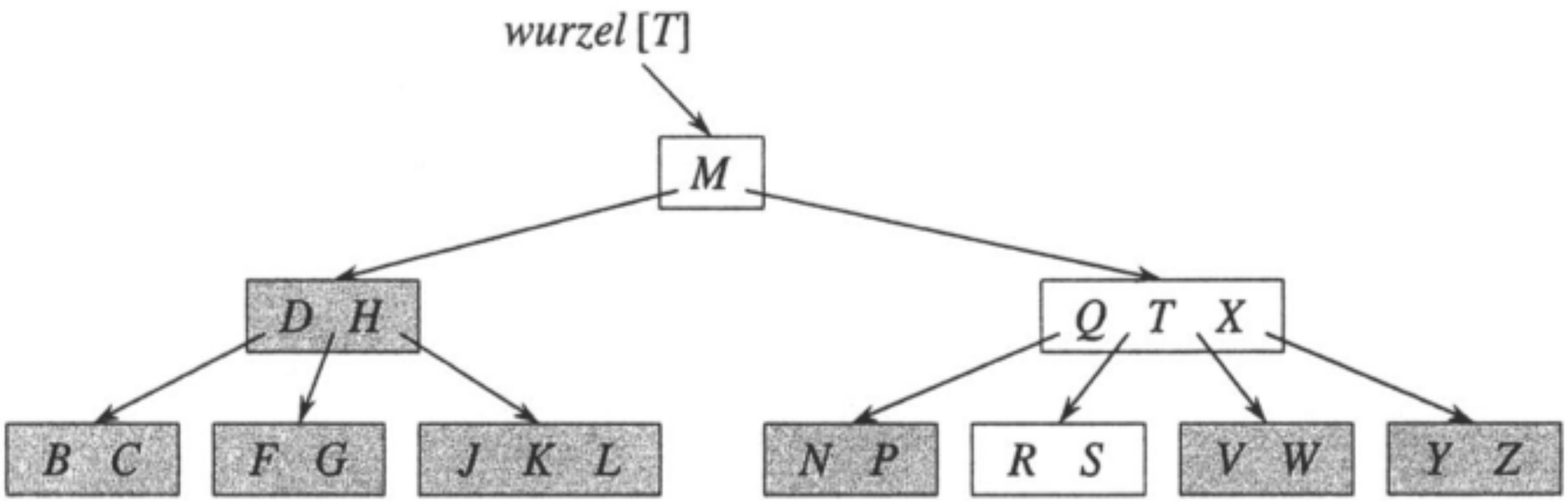


Definition 4.13 (B-Baum)

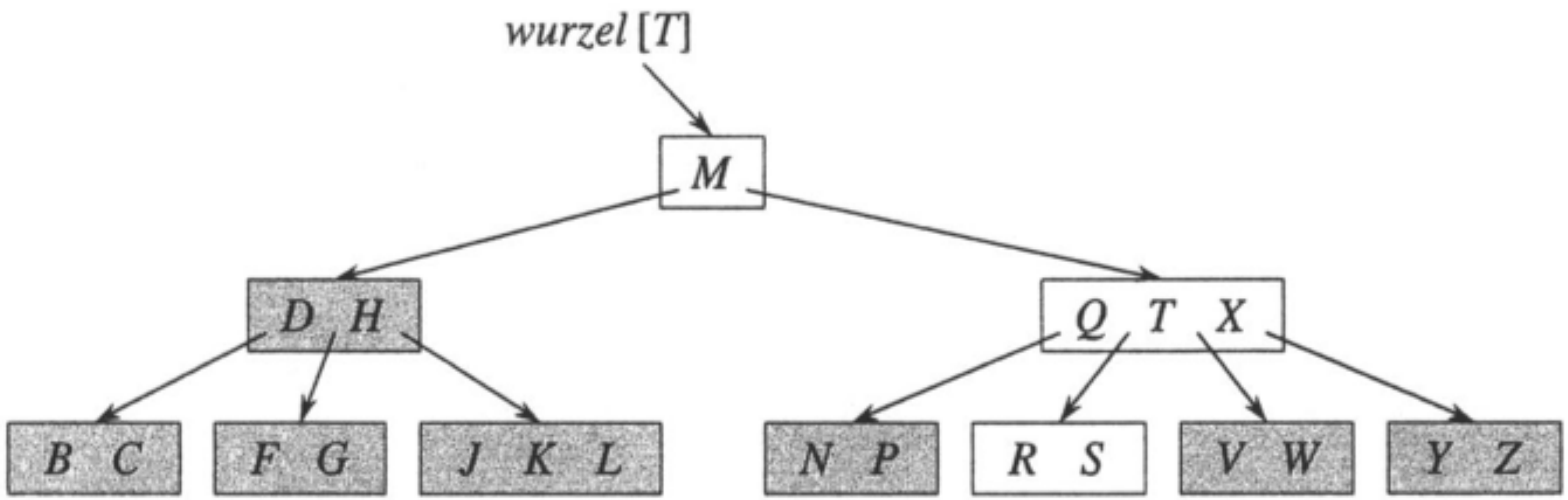
Ein B-Baum ist ein gerichteter Baum mit den Eigenschaften:

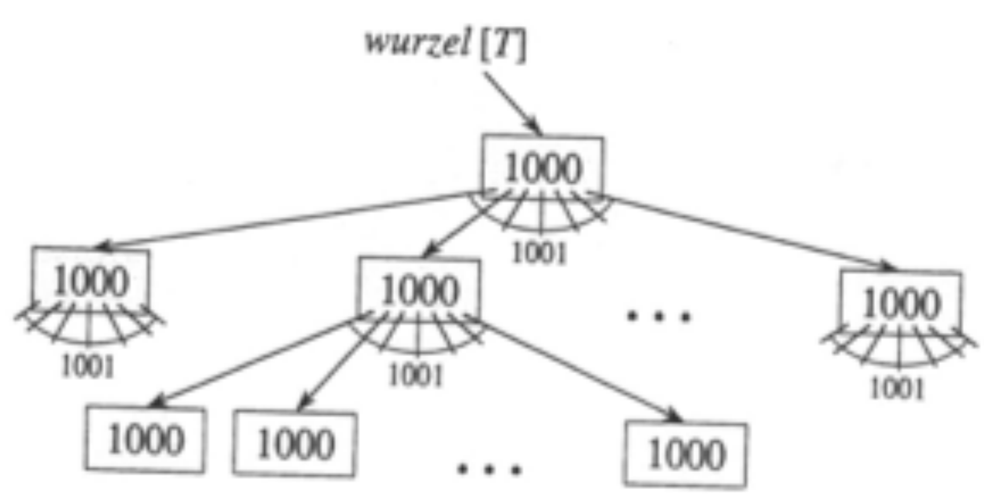
1. Jeder Knoten hat die folgenden Attribute:
   a. die Anzahl n[x] der in x gespeicherten Schlüssel
   b. die sortiert gespeicherten Schlüssel
   c. ein boolscher Wert, der anzeigt, ob x ein Blatt ist
2. Jeder innere Knoten enthält n[x]+1 Zeiger auf seine Kinder
3. Die Schlüssel unterteilen die darunter stehenden Teilbäume nach Größe
4. Alle Blätter haben gleiche Tiefe
5. Jeder Knoten hat Mindest- und Maximalzahl von Schlüsseln
   a. Jeder Knoten (außer der Wurzel) hat mindestens t-1 Schlüssel (Also hat jeder innere Knoten mindestens t Kinder)
   b. Jeder Knoten hat höchstens 2t-1 Schlüssel, also höchstens 2t Kinder

B-Bäume



B-Bäume





1 Knoten,
1000 Schlüssel

1001 Knoten,
1 001 000 Schlüssel

1 002 001 Knoten,
1 002 001 000 Schlüssel

B-Bäume

B-Bäume

B-Baum der Höhe h=3 und Fan-Out-Factor q = 2048

B-Bäume

B-Baum der Höhe h=3 und Fan-Out-Factor q = 2048

- jeder Knoten speichert bis zu 2047 Schlüssel und 2048 Links

B-Bäume

B-Baum der Höhe h=3 und Fan-Out-Factor q = 2048

- jeder Knoten speichert bis zu 2047 Schlüssel und 2048 Links
- vollständig: n = 8581000000

B-Bäume

B-Baum der Höhe h=3 und Fan-Out-Factor q = 2048

- jeder Knoten speichert bis zu 2047 Schlüssel und 2048 Links
- vollständig: n = 8581000000
- balanciert: n ≥ 4000000

B-Bäume

B-Baum der Höhe h=3 und Fan-Out-Factor q = 2048

- jeder Knoten speichert bis zu 2047 Schlüssel und 2048 Links
- vollständig: n = 8581000000
- balanciert: n ≥ 4000000

Binärer Baum der Höhe h=3 und Fan-Out-Factor q = 2

B-Bäume

B-Baum der Höhe h=3 und Fan-Out-Factor q = 2048

- jeder Knoten speichert bis zu 2047 Schlüssel und 2048 Links
- vollständig: n = 8581000000
- balanciert: n ≥ 4000000

Binärer Baum der Höhe h=3 und Fan-Out-Factor q = 2

- jeder Knoten speichert 1 Schlüssel und 2 Links

B-Bäume

B-Baum der Höhe h=3 und Fan-Out-Factor q = 2048

- jeder Knoten speichert bis zu 2047 Schlüssel und 2048 Links
- vollständig: n = 8581000000
- balanciert: n ≥ 4000000

Binärer Baum der Höhe h=3 und Fan-Out-Factor q = 2

- jeder Knoten speichert 1 Schlüssel und 2 Links
- vollständig: n = 7

B-Bäume

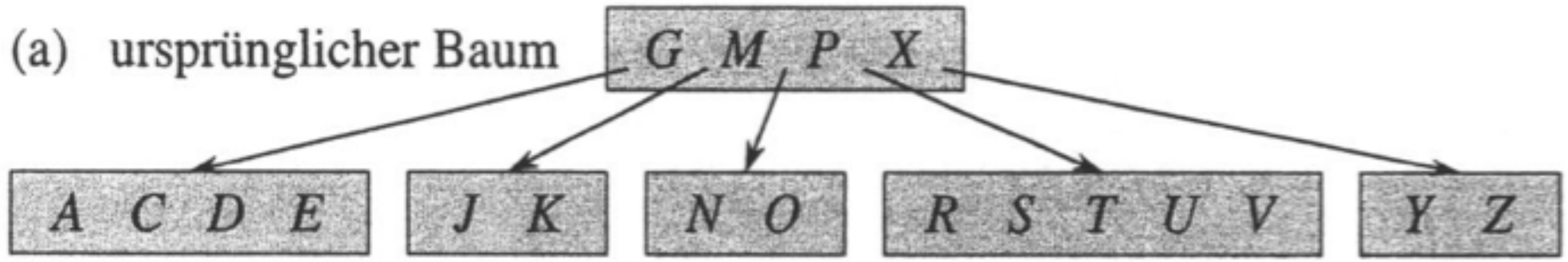
B-Baum der Höhe h=3 und Fan-Out-Factor q = 2048

- jeder Knoten speichert bis zu 2047 Schlüssel und 2048 Links
- vollständig: n = 8581000000
- balanciert: n ≥ 4000000

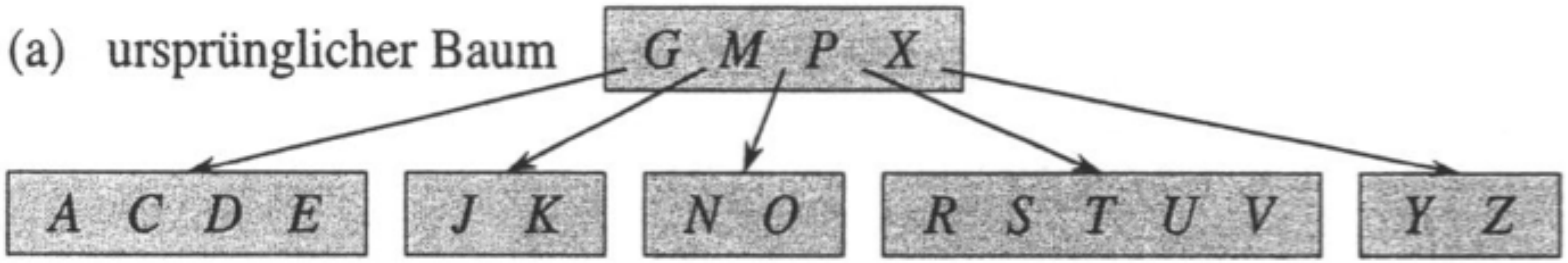
Binärer Baum der Höhe h=3 und Fan-Out-Factor q = 2

- jeder Knoten speichert 1 Schlüssel und 2 Links
- vollständig: n = 7
- balanciert: n ≥ 4

B-Bäume — Einfügen

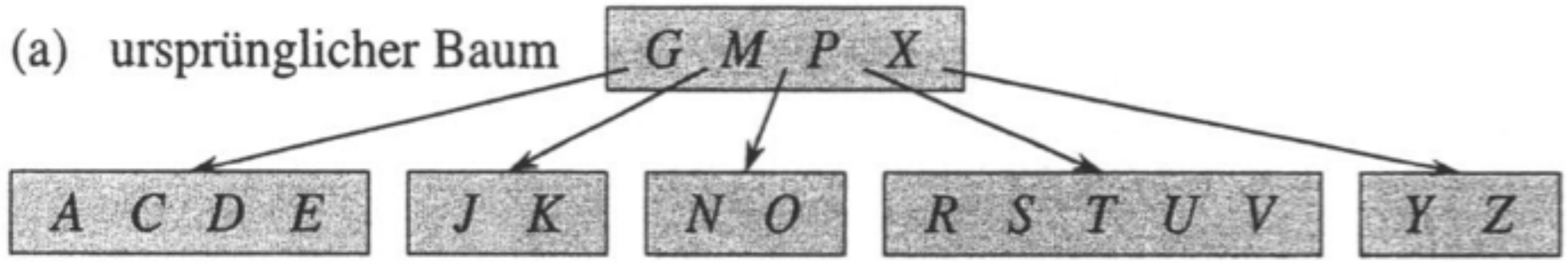


B-Bäume — Einfügen

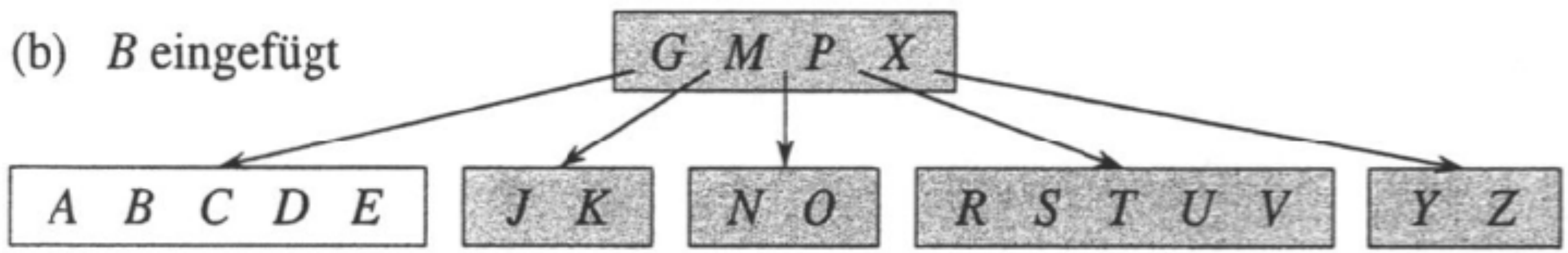


Füge B ein!

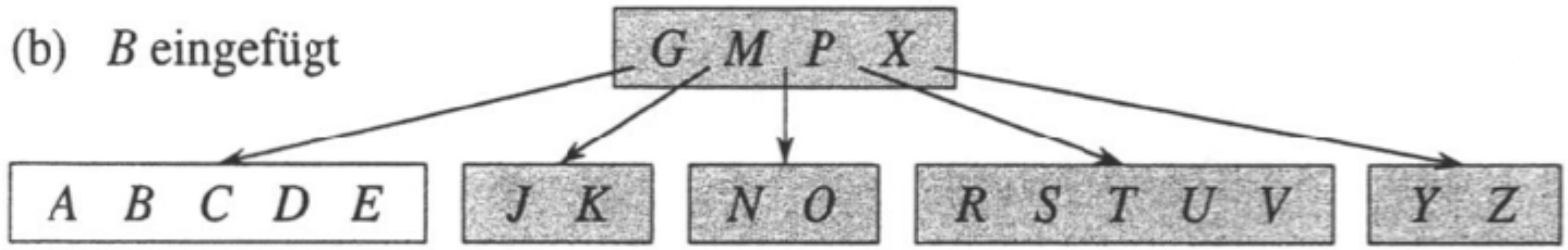
B-Bäume — Einfügen



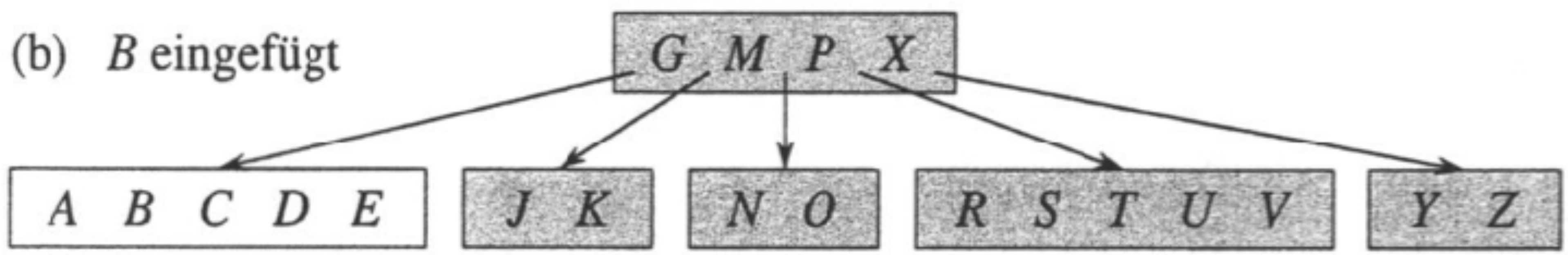
Füge B ein!



B-Bäume — Einfügen

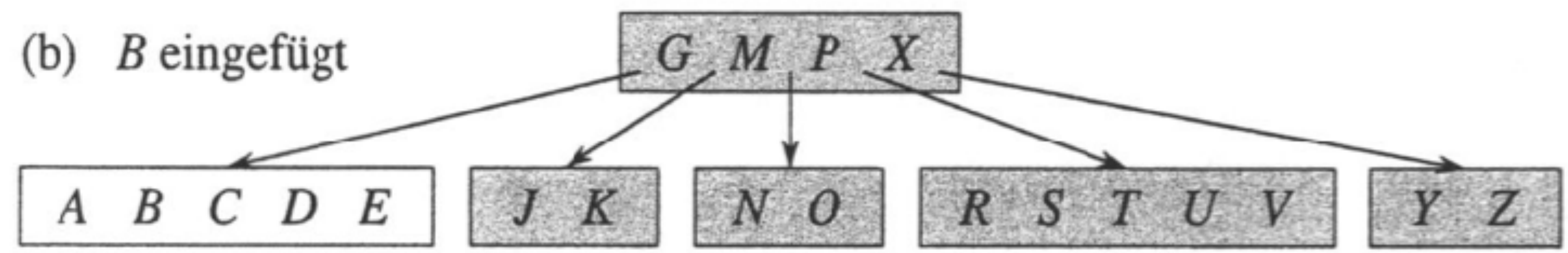


B-Bäume — Einfügen

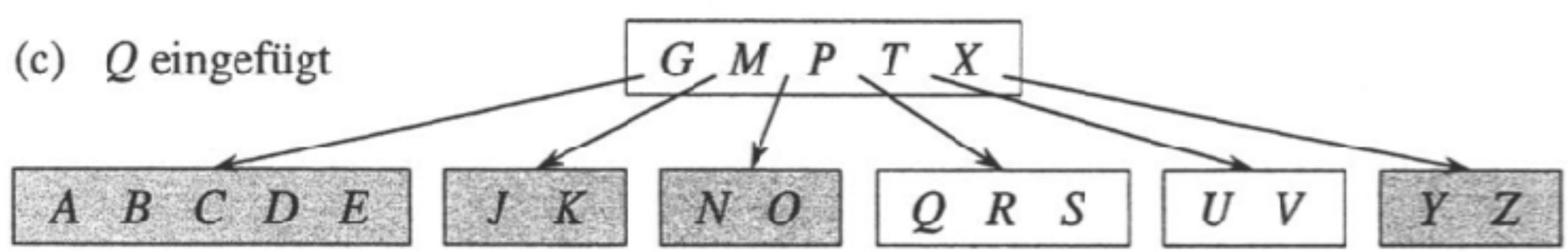


Füge Q ein!

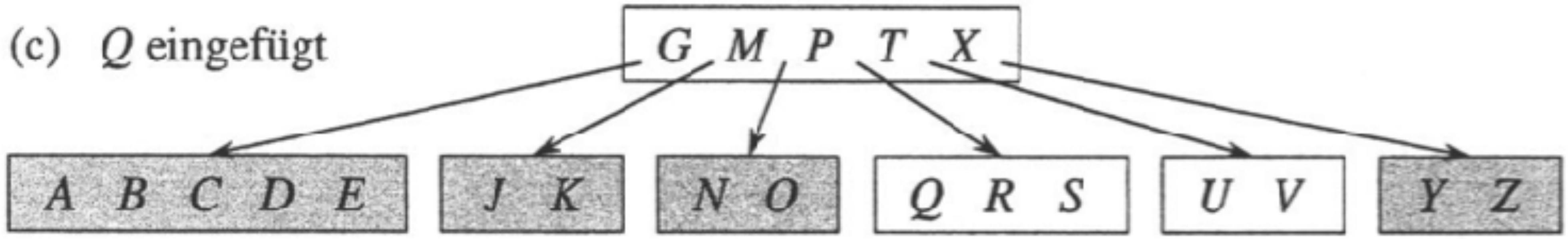
B-Bäume — Einfügen



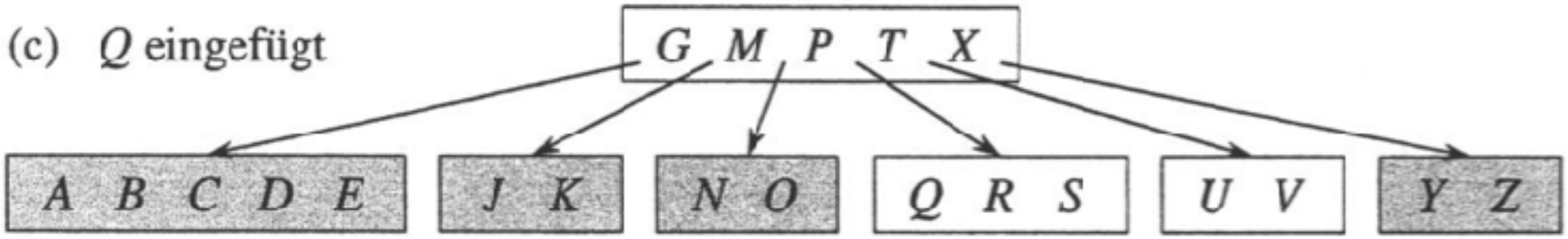
Füge Q ein!



B-Bäume — Einfügen



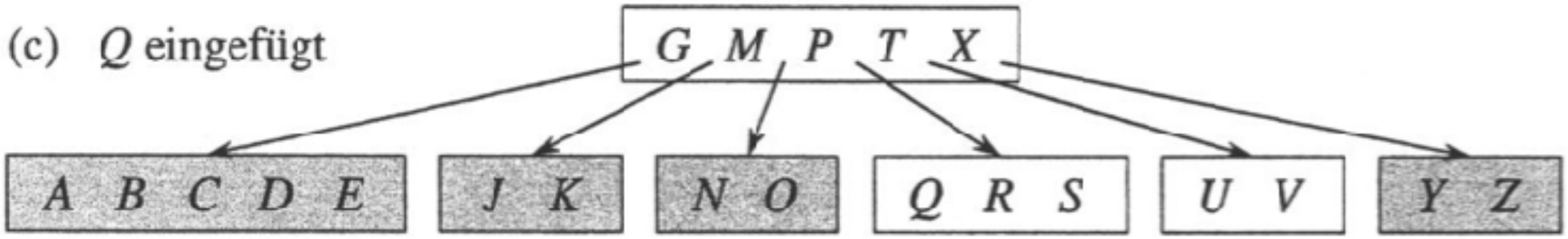
B-Bäume — Einfügen



Füge L ein!

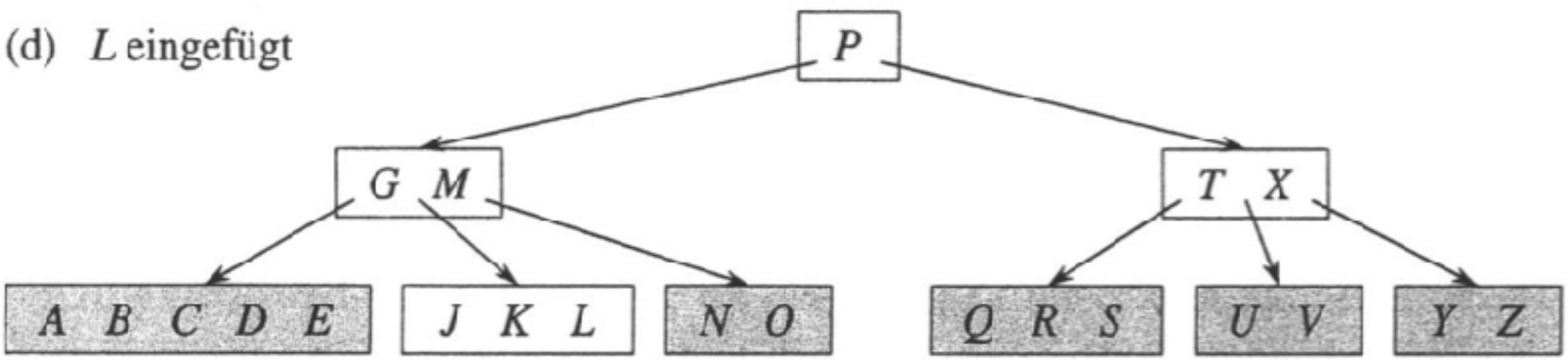
B-Bäume — Einfügen

(c) *Q* eingefügt

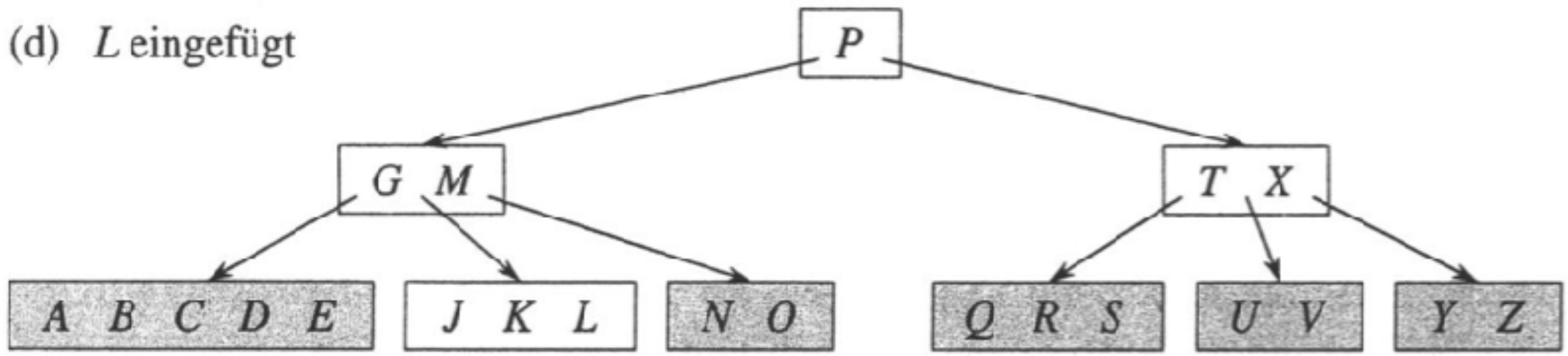


Füge L ein!

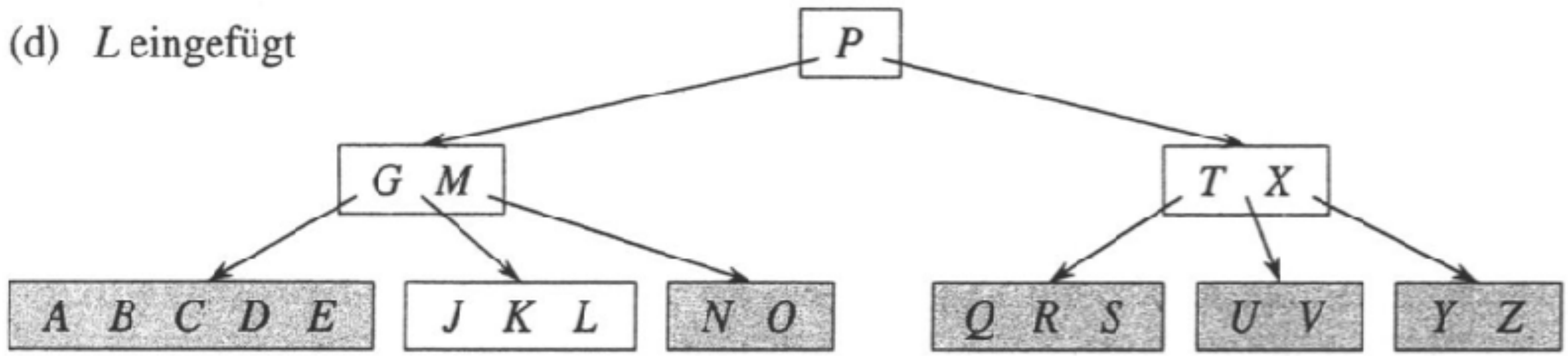
(d) *L* eingefügt



B-Bäume — Einfügen

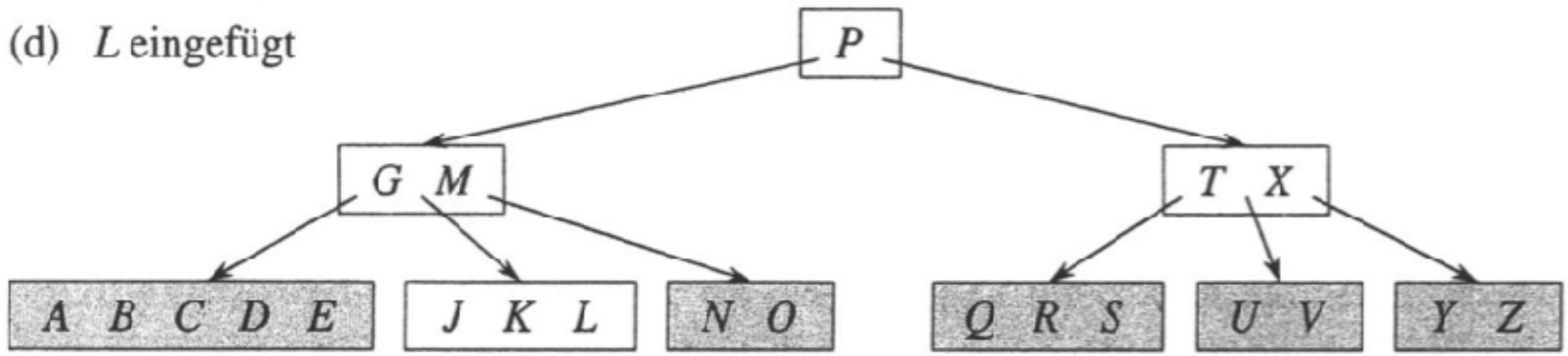


B-Bäume — Einfügen

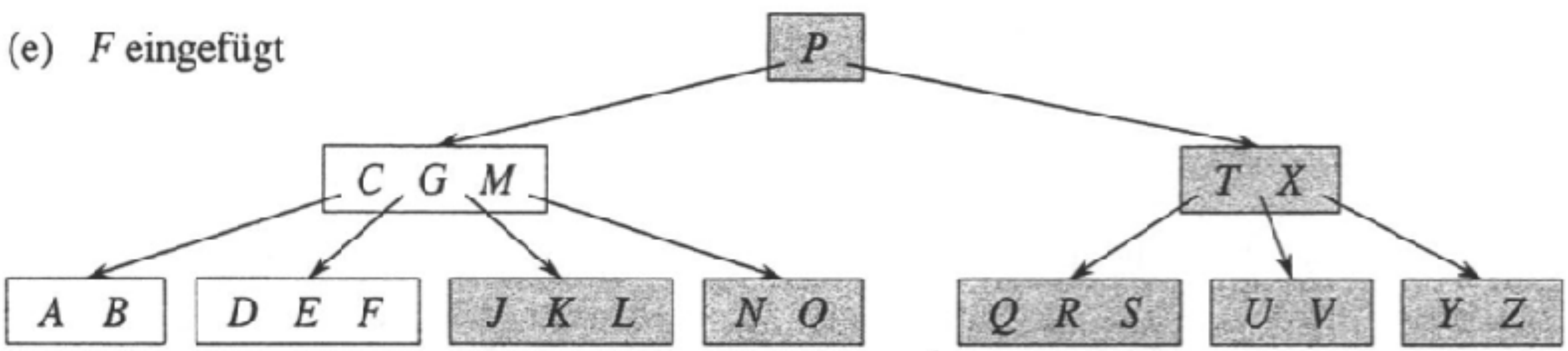


Füge F ein!

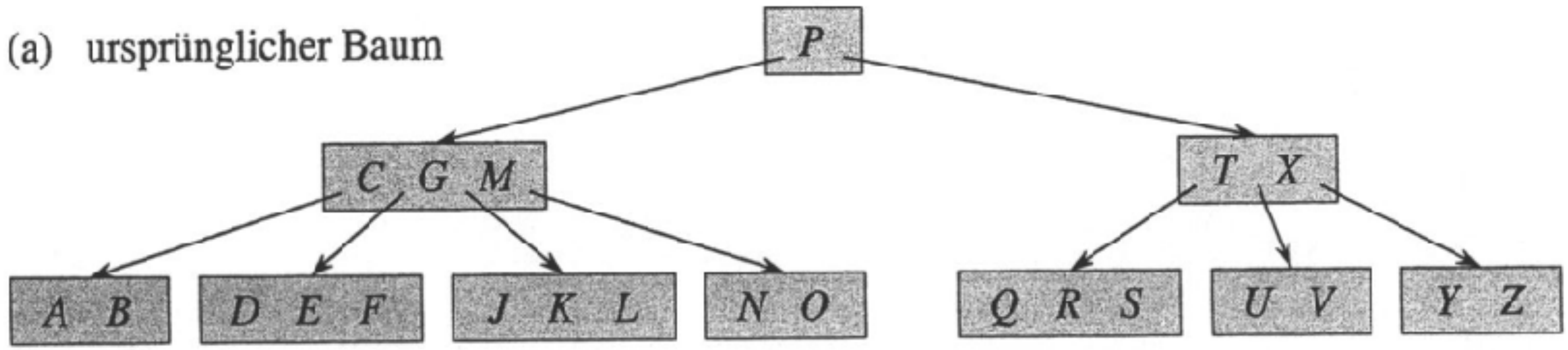
B-Bäume — Einfügen



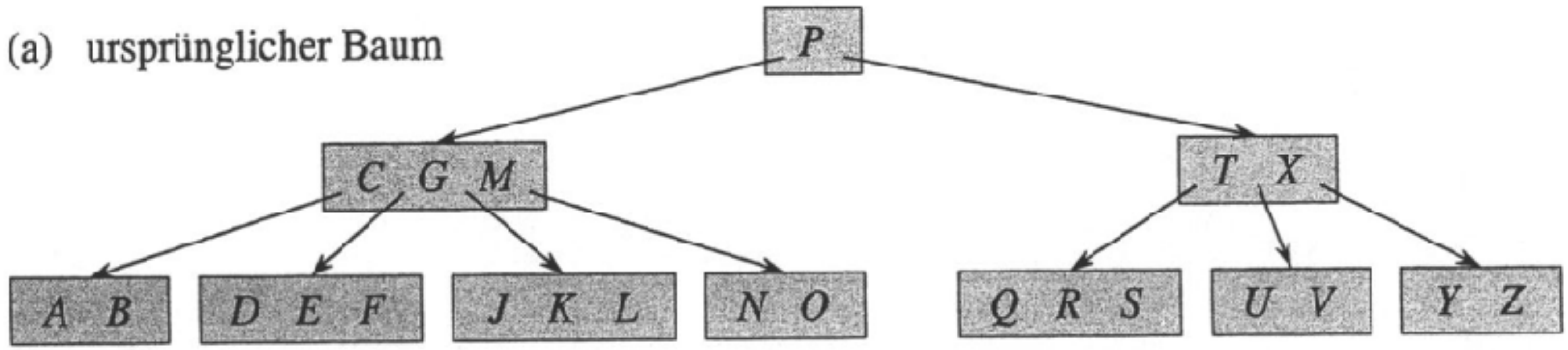
Füge F ein!



B-Bäume — Löschen

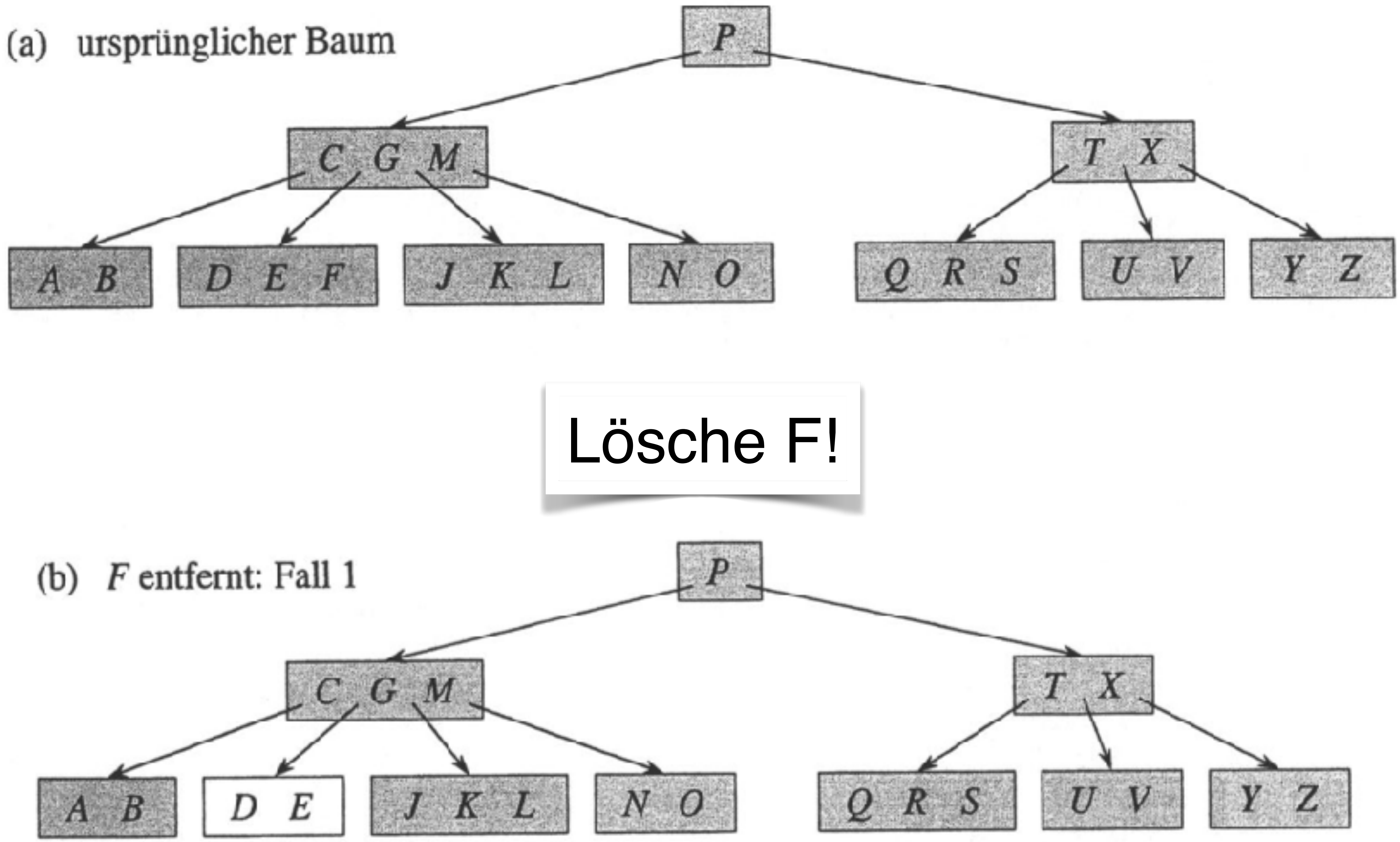


B-Bäume — Löschen



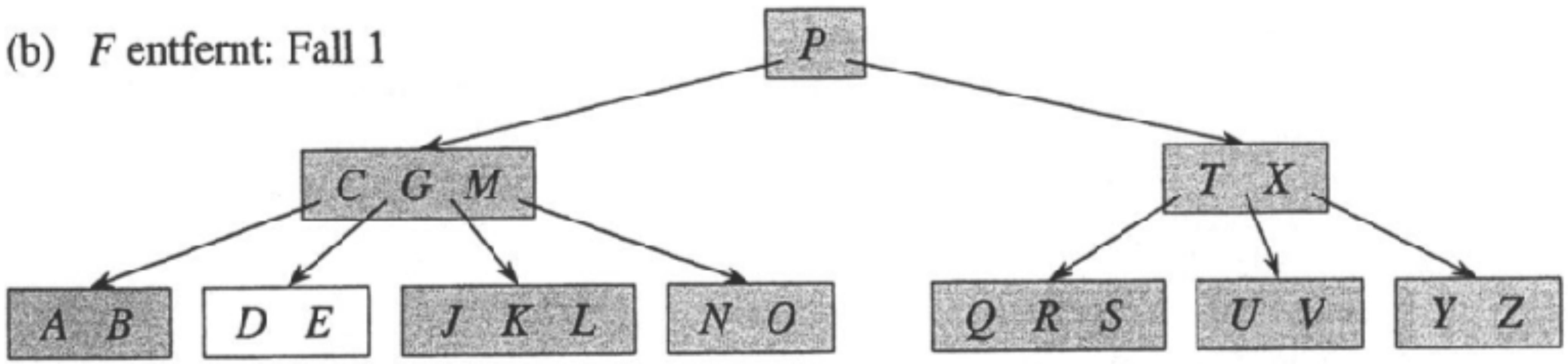
Lösche F!

B-Bäume — Löschen

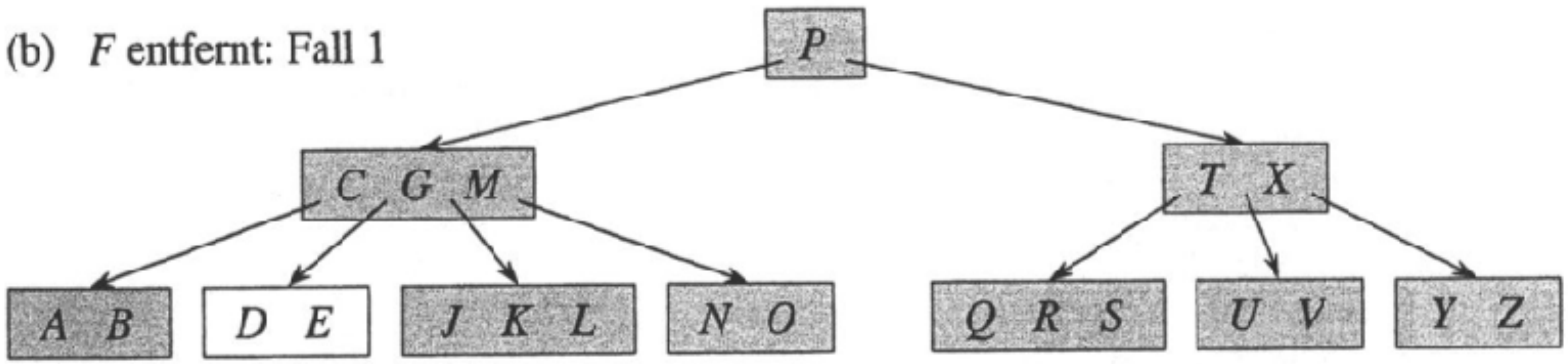


Technische Universität Braunschweig

B-Bäume — Löschen



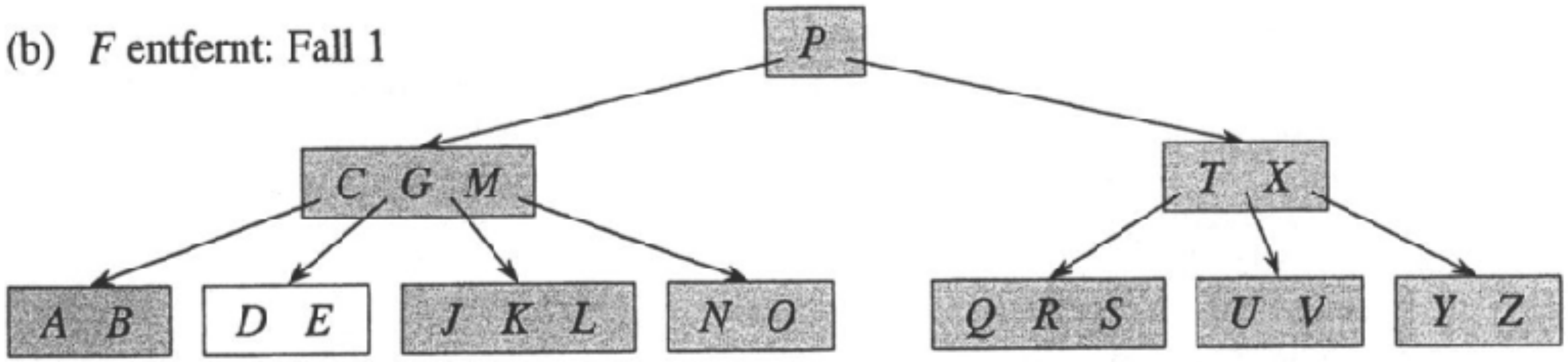
B-Bäume — Löschen



Lösche M!

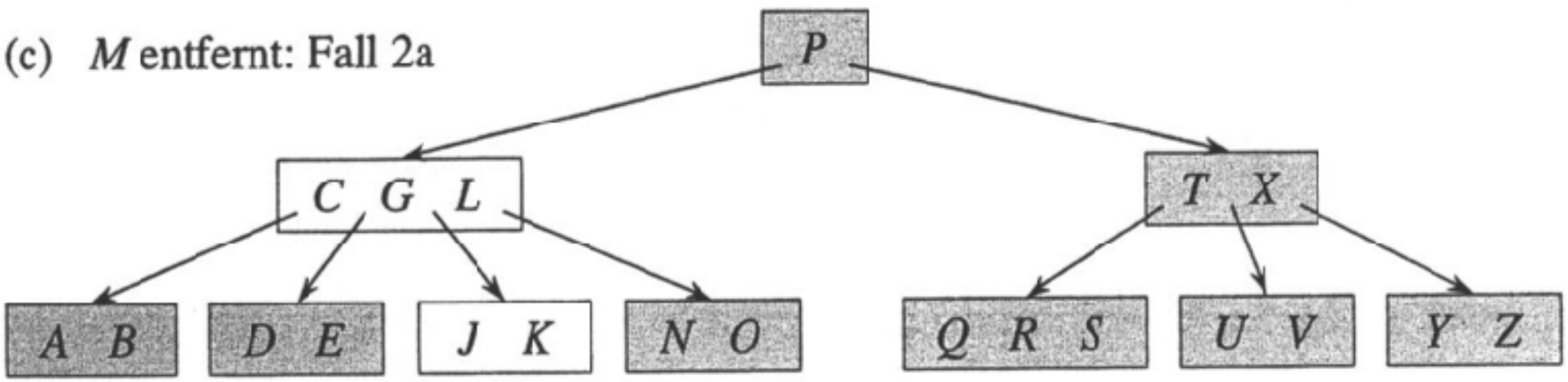
B-Bäume — Löschen

(b) *F* entfernt: Fall 1

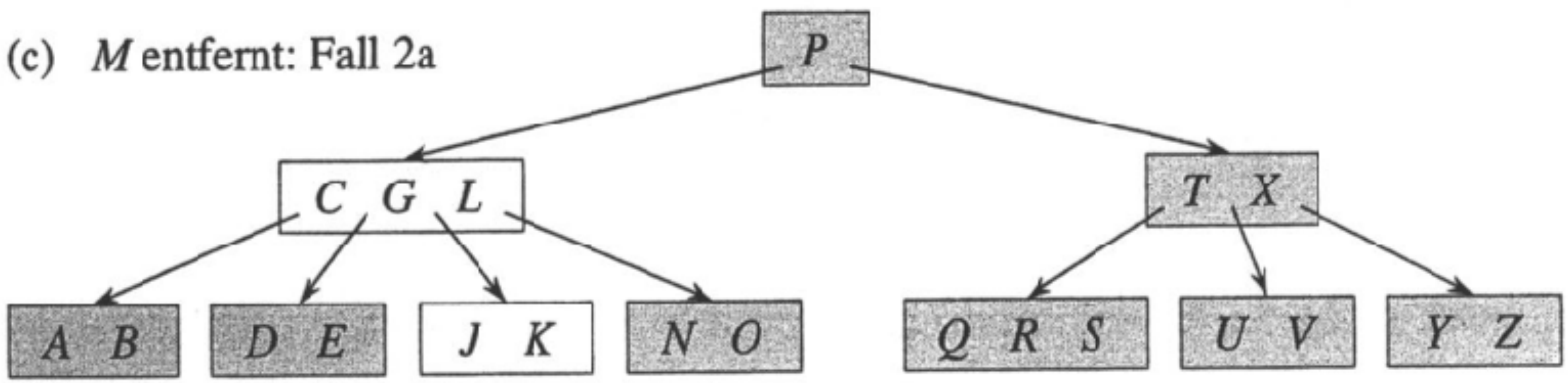


Lösche M!

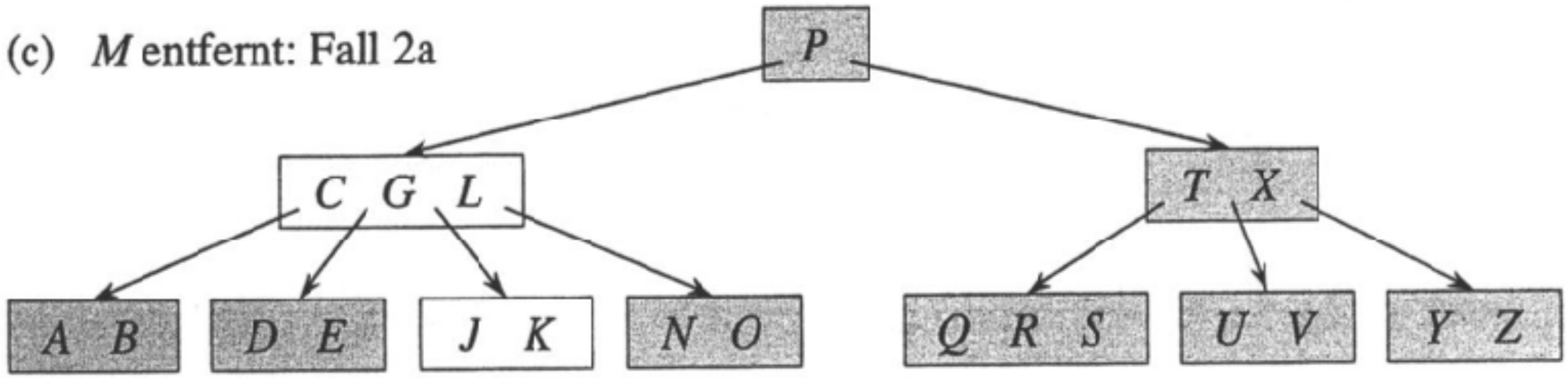
(c) *M* entfernt: Fall 2a



B-Bäume — Löschen

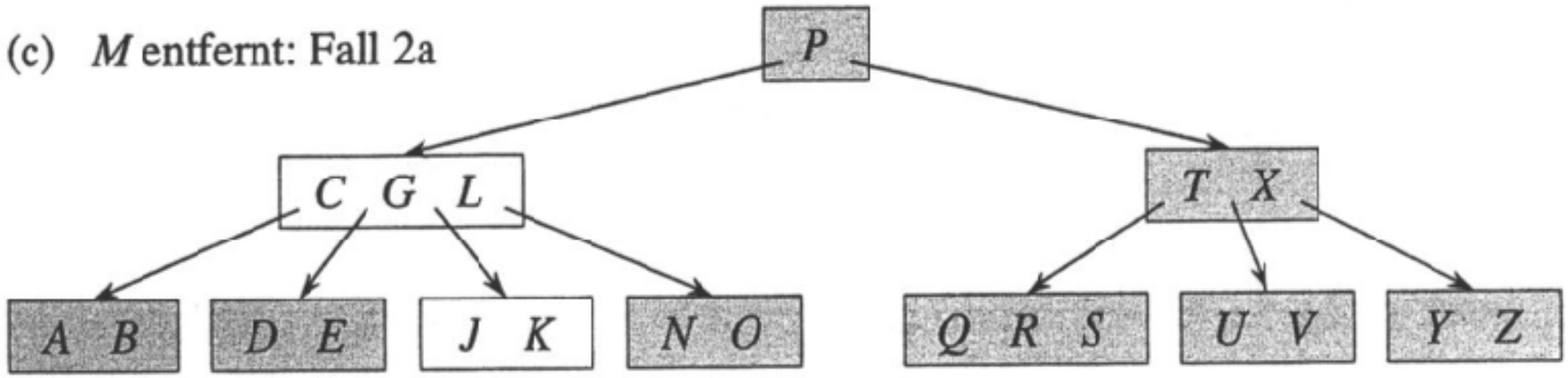


B-Bäume — Löschen

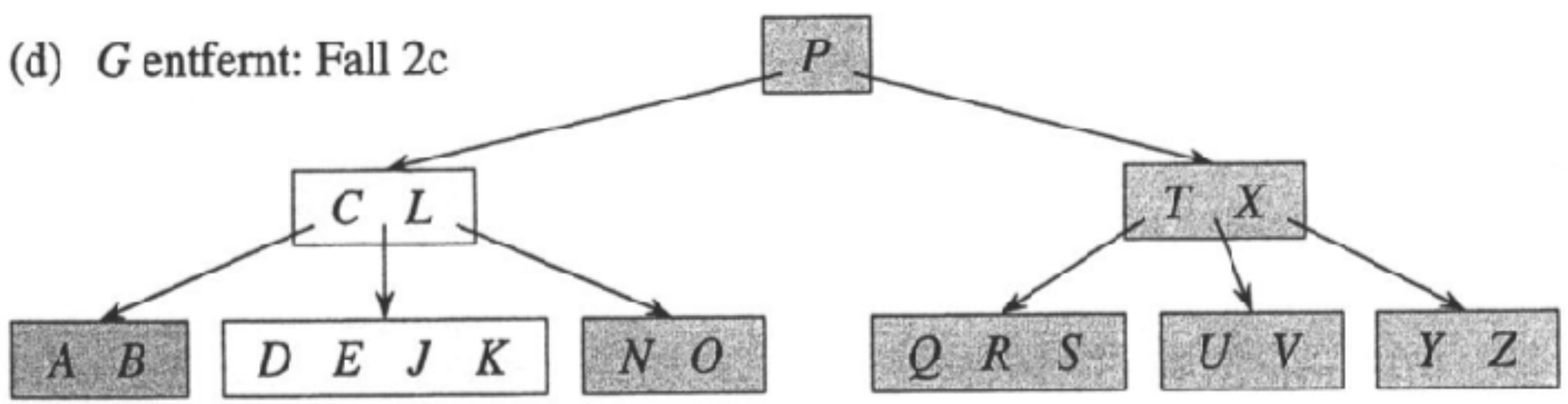


Lösche G!

B-Bäume — Löschen

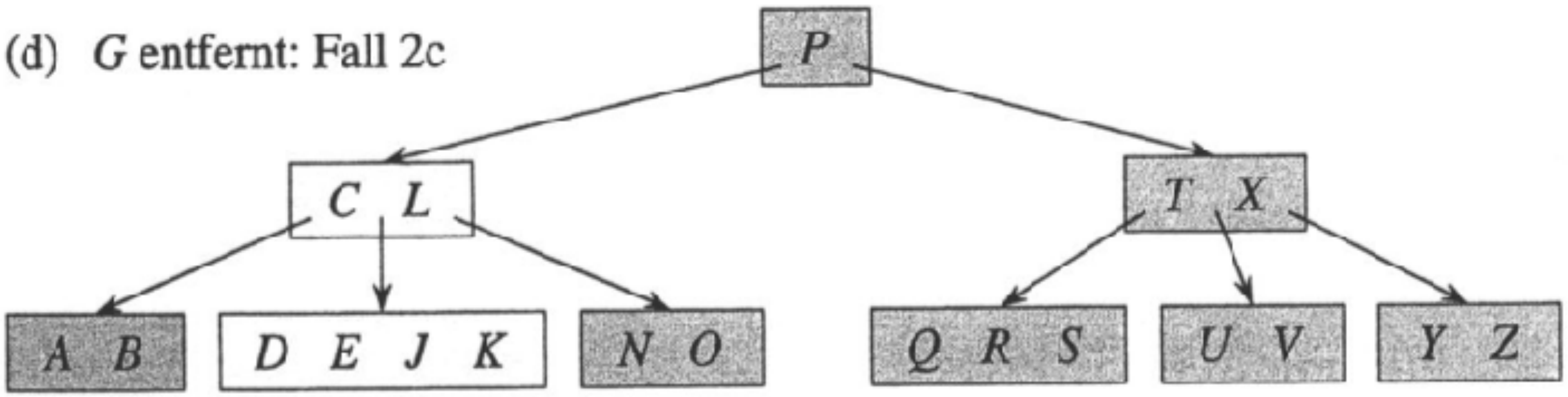


Lösche G!

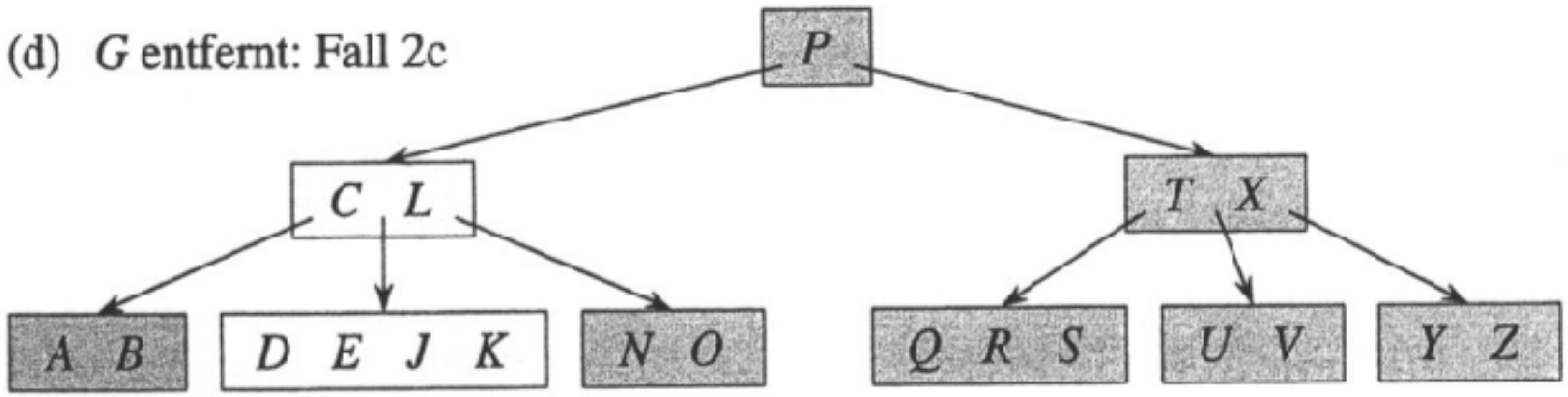


B-Bäume — Löschen

(d) *G* entfernt: Fall 2c



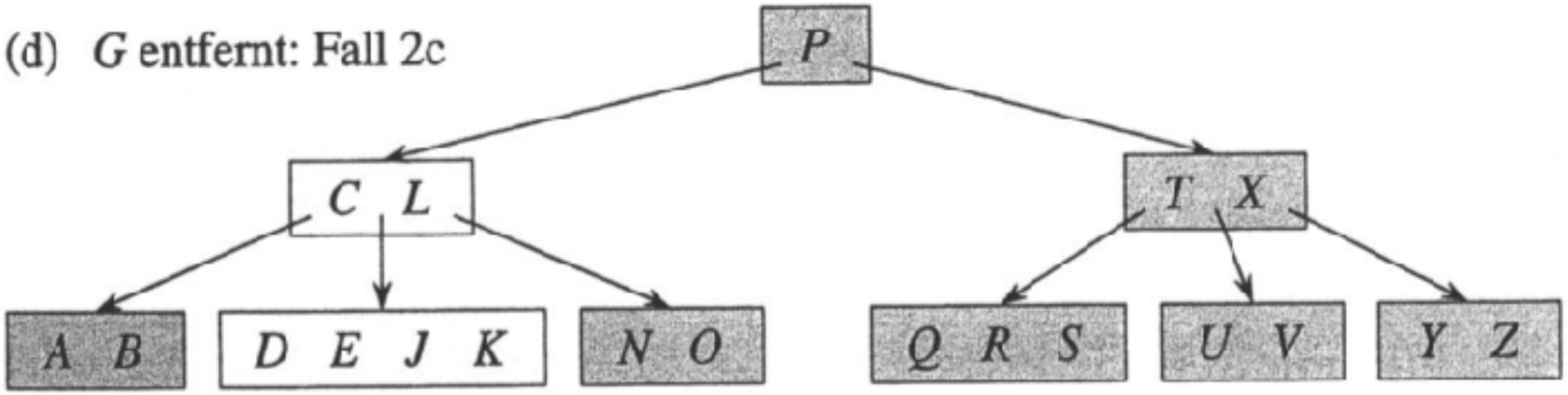
B-Bäume — Löschen



Lösche D!

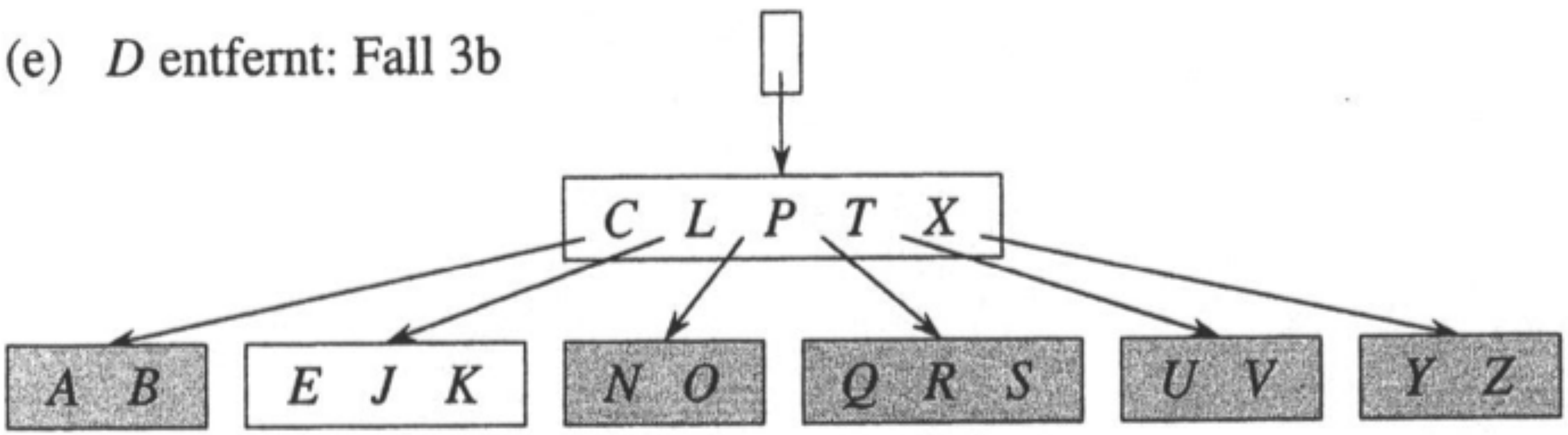
B-Bäume — Löschen

(d) *G* entfernt: Fall 2c

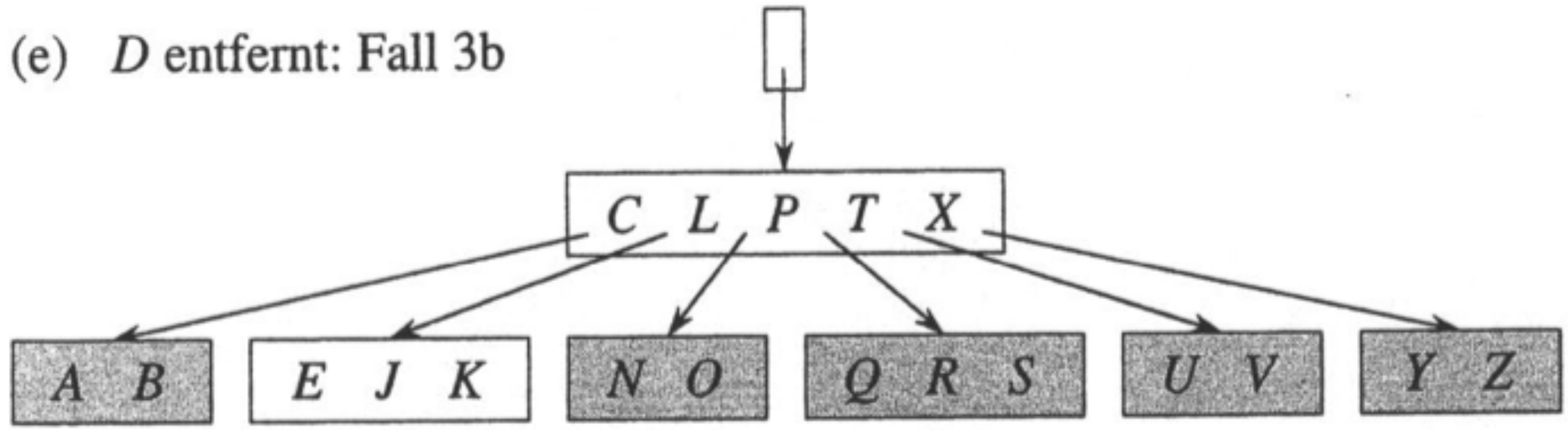


Lösche D!

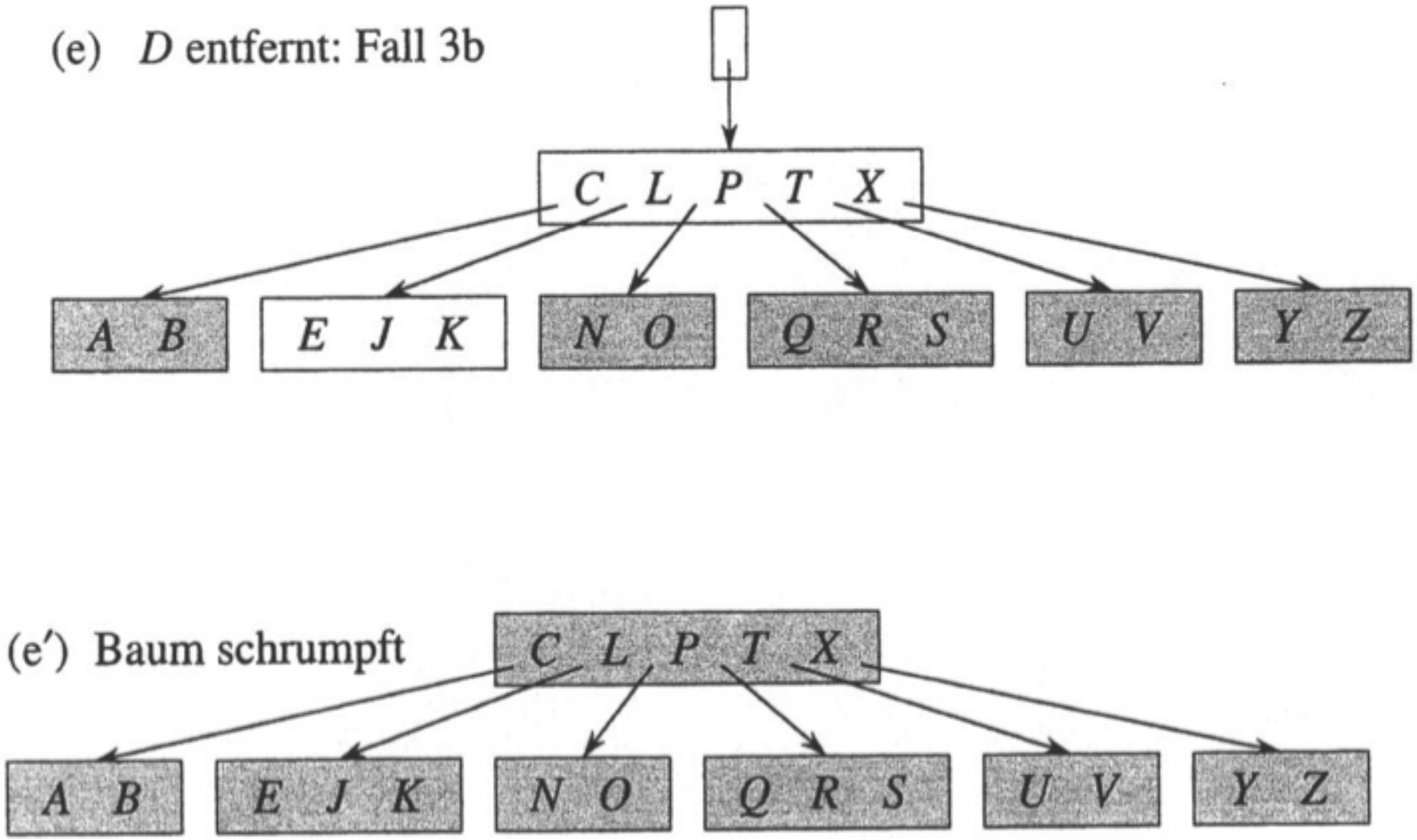
(e) *D* entfernt: Fall 3b



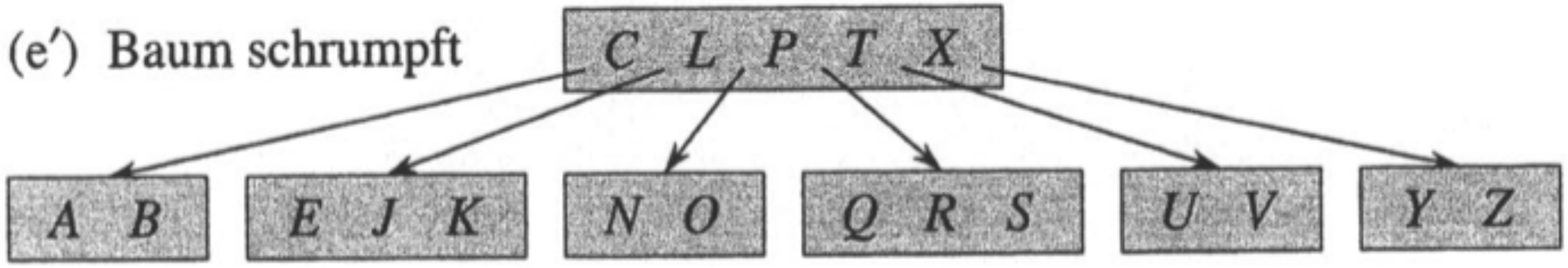
B-Bäume — Löschen



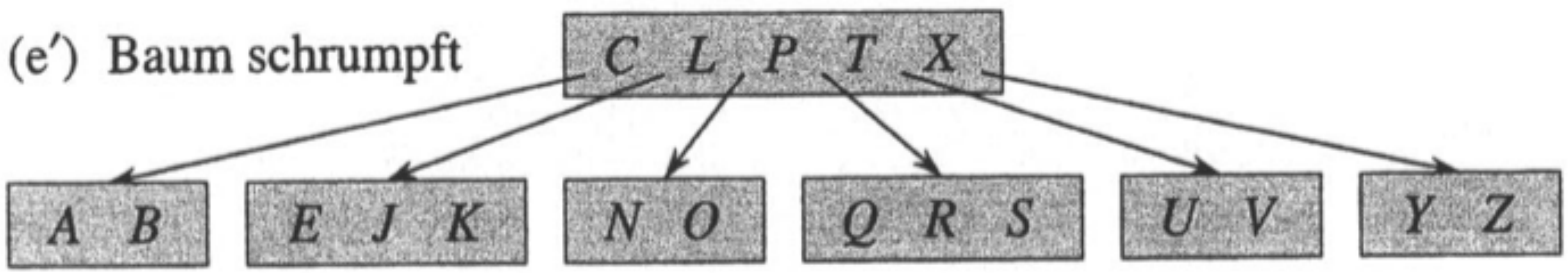
B-Bäume — Löschen



B-Bäume — Löschen

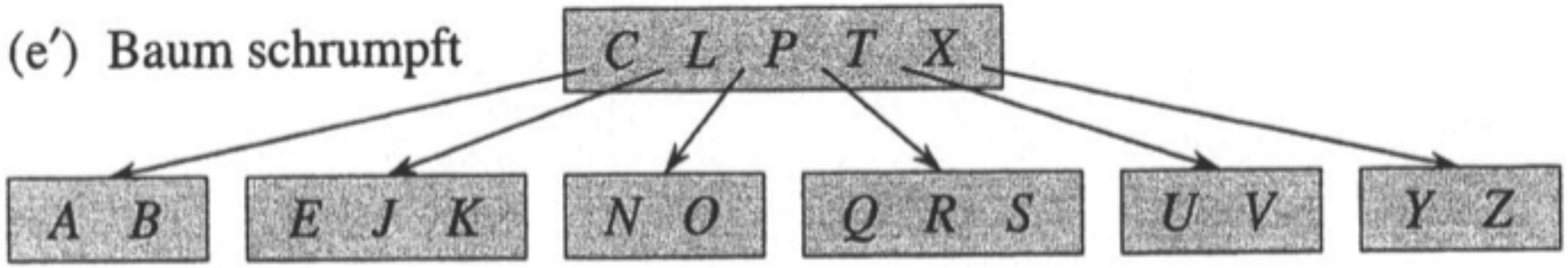


B-Bäume — Löschen

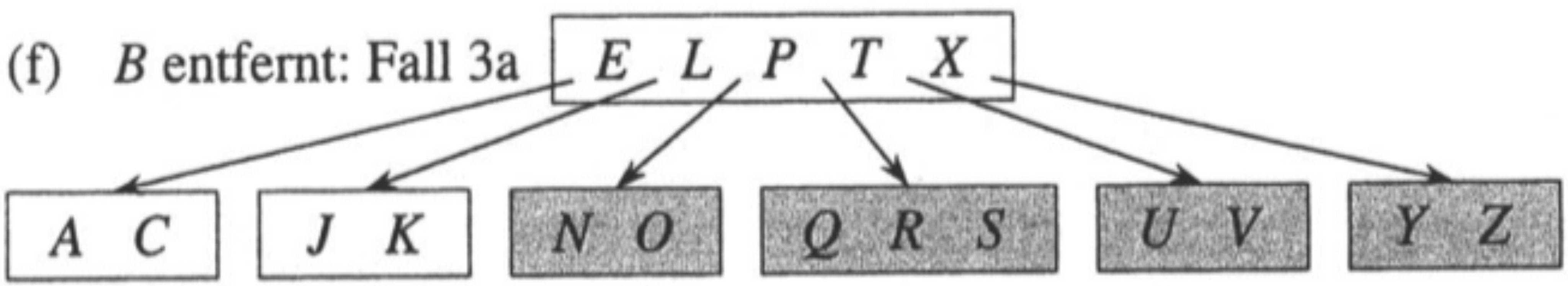


Lösche B!

B-Bäume — Löschen



Lösche B!



B-Bäume

Satz 4.14
Ein B-Baum der Höhe h benötigt zur dynamischen Datenverwaltung im schlimmsten Fall O(h) Plattenoperationen und $O(t \log_t n)$ CPU-Zeit.

B-Bäume

Satz 4.14
Ein B-Baum der Höhe h benötigt zur dynamischen Datenverwaltung im schlimmsten Fall O(h) Plattenoperationen und $O(t \log_t n)$ CPU-Zeit.

mehr dazu: Cormen, Kapitel 18.

B-Bäume

Satz 4.14
Ein B-Baum der Höhe h benötigt zur dynamischen Datenverwaltung im schlimmsten Fall O(h) Plattenoperationen und $O(t \log_t n)$ CPU-Zeit.

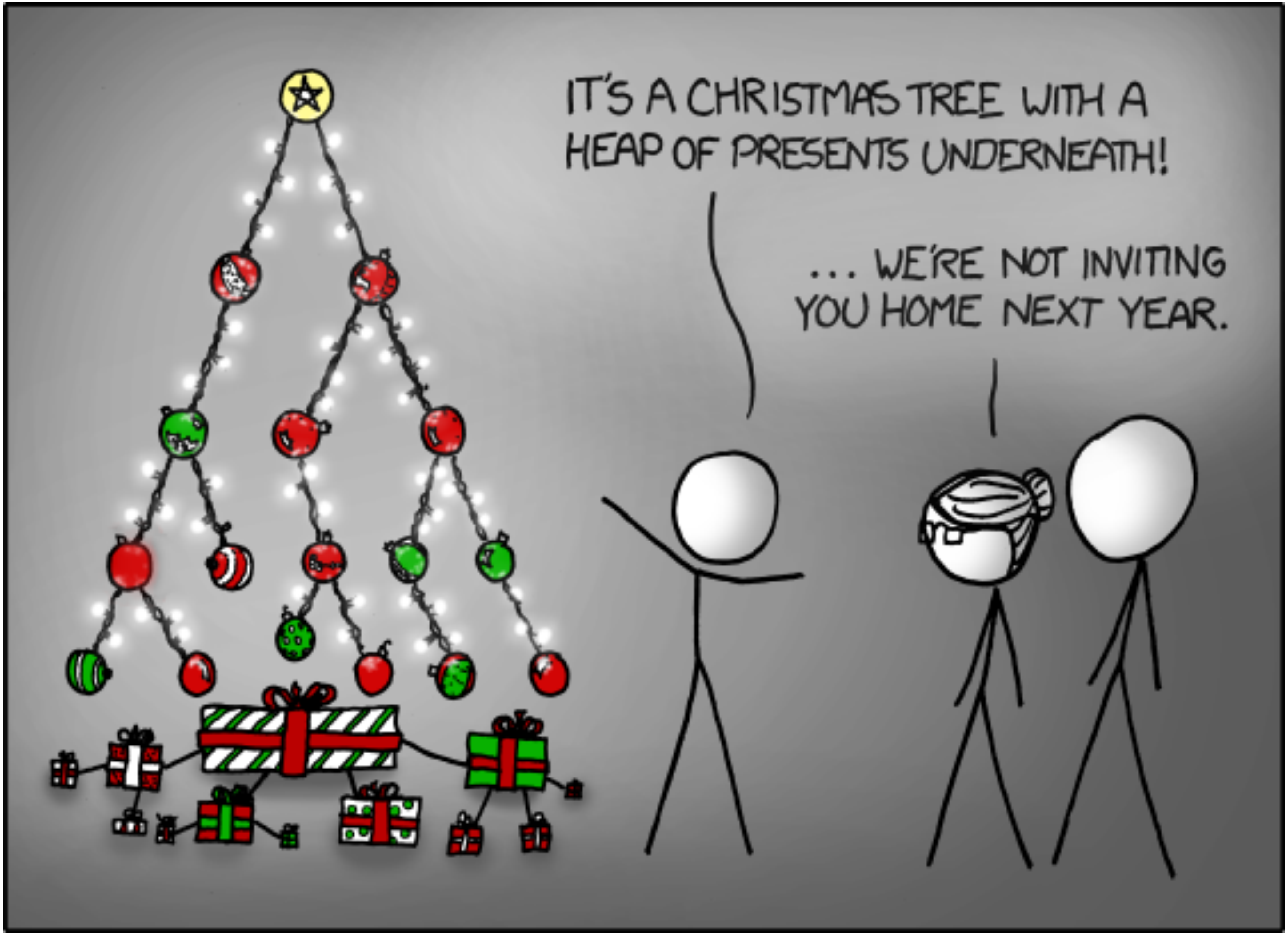
mehr dazu: Cormen, Kapitel 18.

Relationale Datenbanksysteme 2

4.10 Heaps

4.10 Heaps

4.10 Heaps


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.

4.10 Heaps

4.10 Heaps

Idee: Ordne Baum so, dass größere Elemente immer oben/unten stehen.

4.10 Heaps

Idee: Ordne Baum so, dass größere Elemente immer oben/unten stehen.

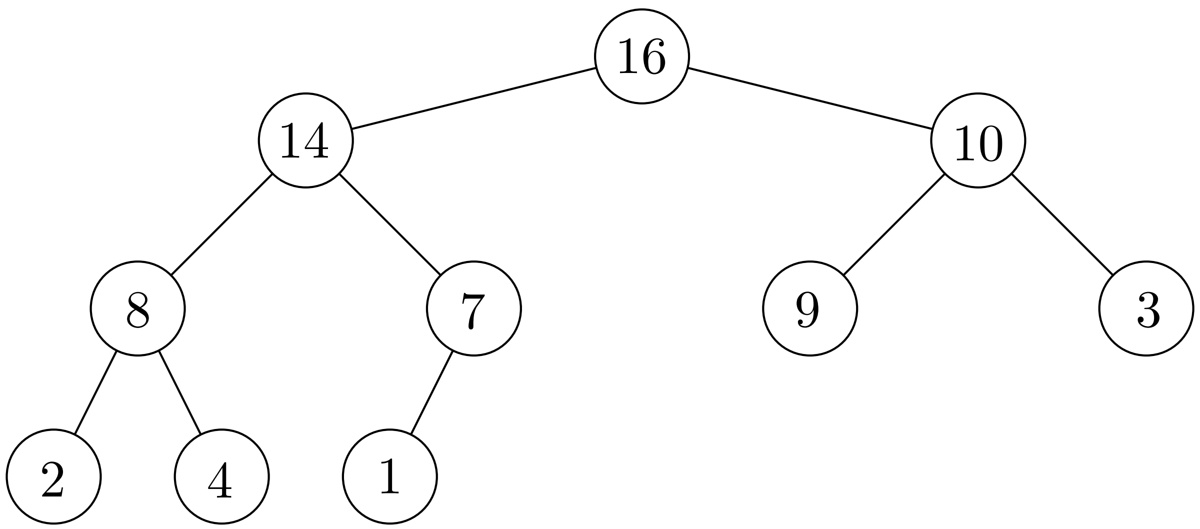
Idee: Ordne Baum so, dass größere Elemente immer oben/unten stehen.

Max-Heap

4.10 Heaps

Idee: Ordne Baum so, dass größere Elemente immer oben/unten stehen.

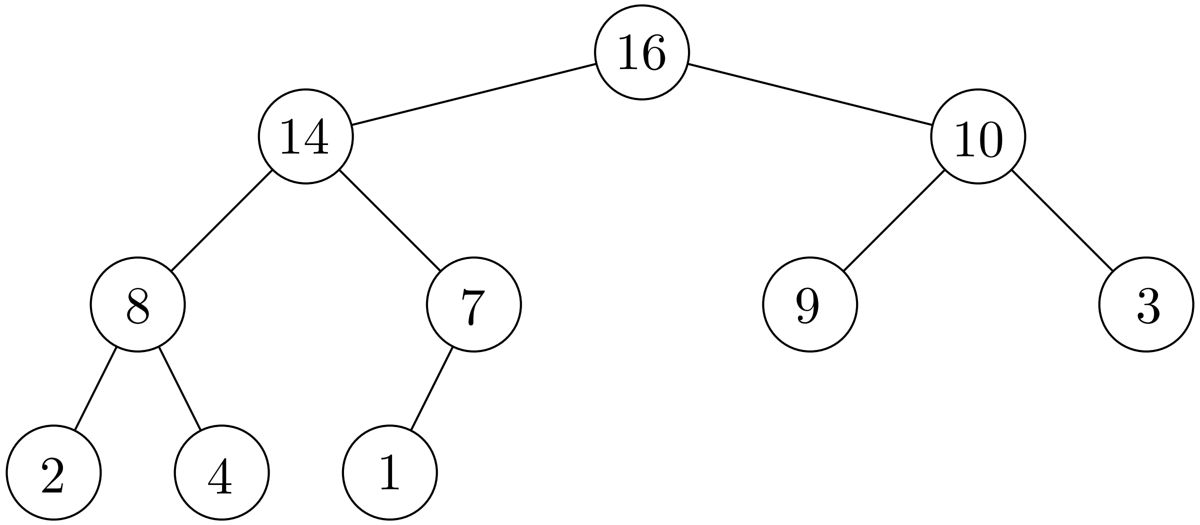
Max-Heap



4.10 Heaps

Idee: Ordne Baum so, dass größere Elemente immer oben/unten stehen.

Max-Heap

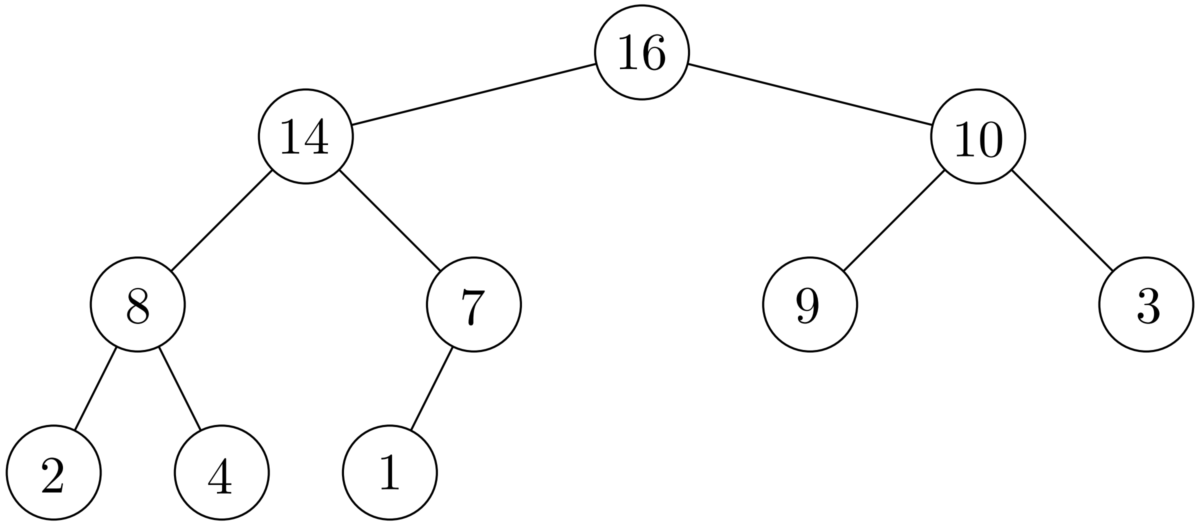


4.10 Heaps

Idee: Ordne Baum so, dass größere Elemente immer oben/unten stehen.

Max-Heap

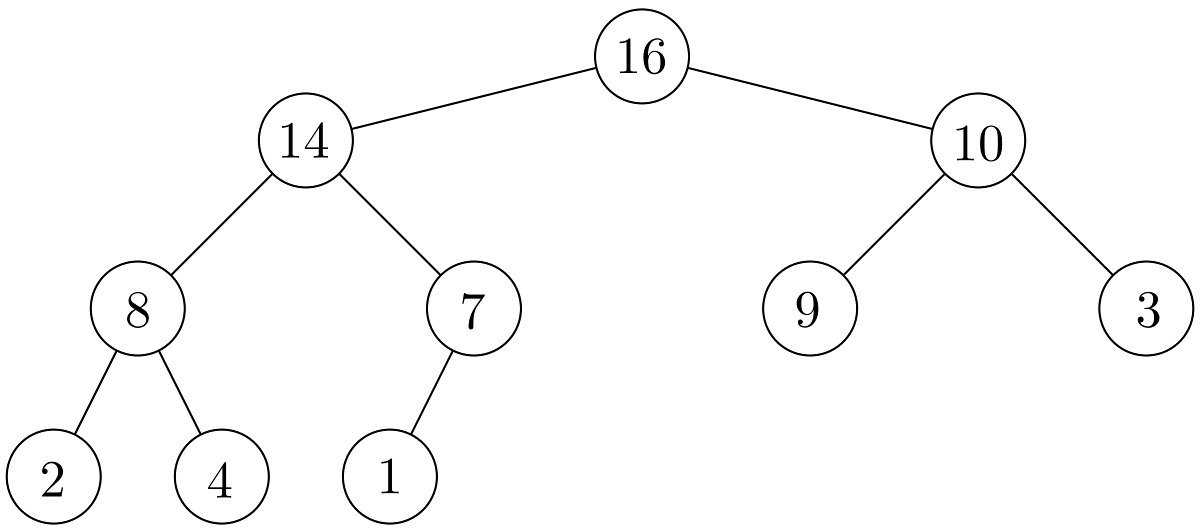
Min-Heap



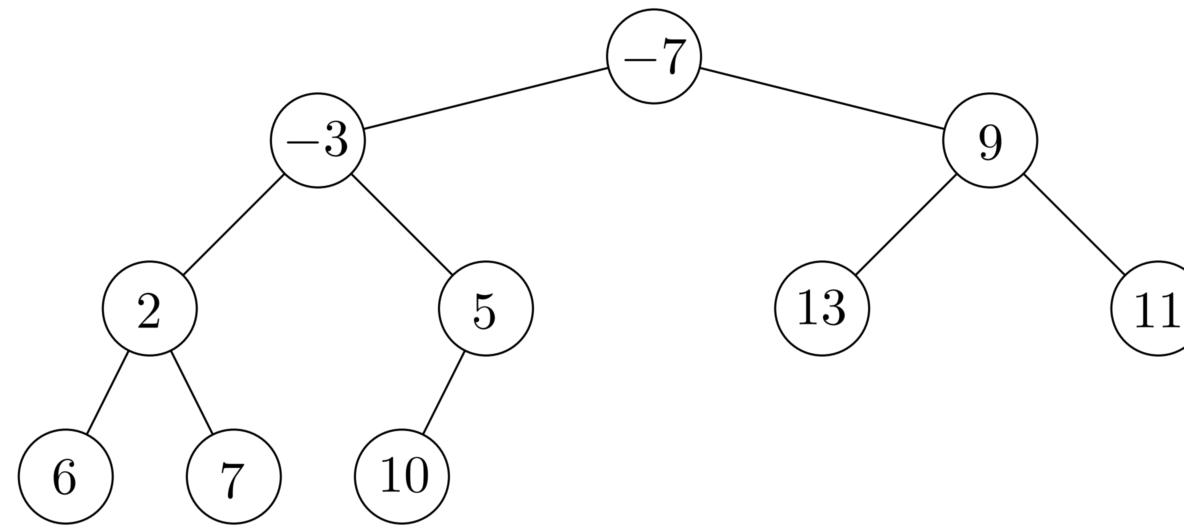
4.10 Heaps

Idee: Ordne Baum so, dass größere Elemente immer oben/unten stehen.

Max-Heap

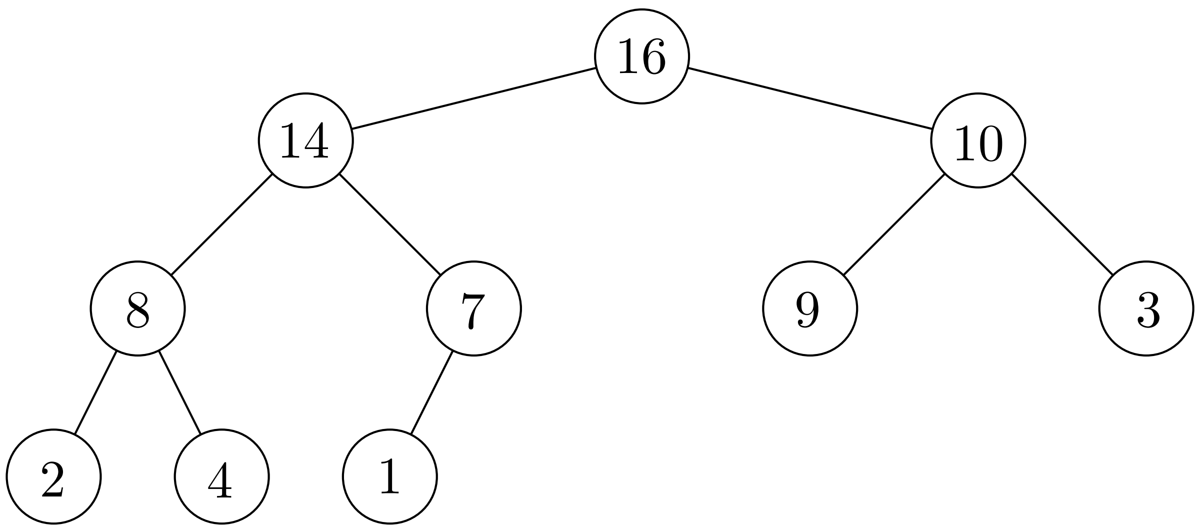


Min-Heap

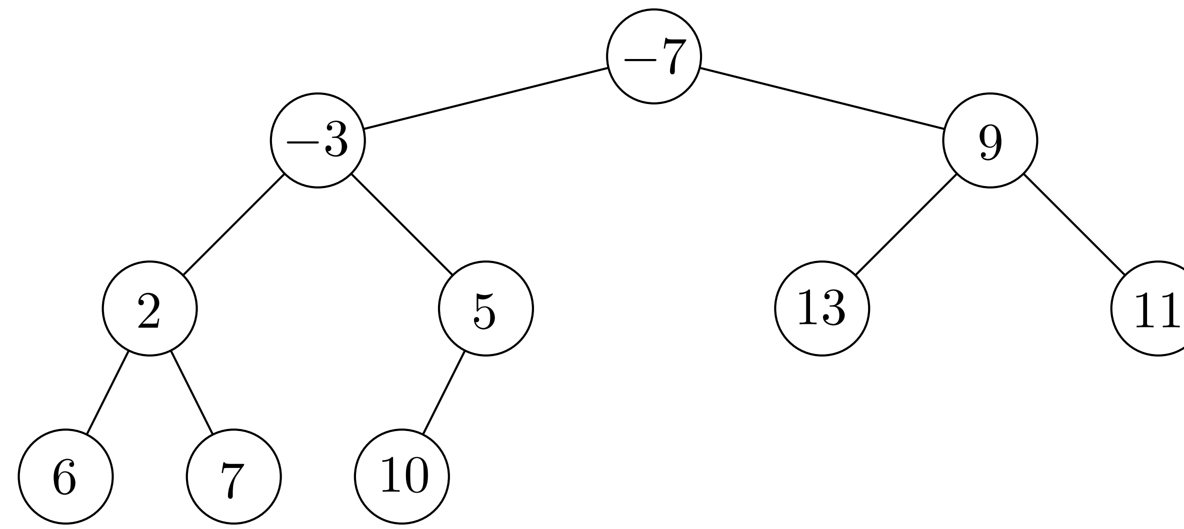


Idee: Ordne Baum so, dass größere Elemente immer oben/unten stehen.

Max-Heap



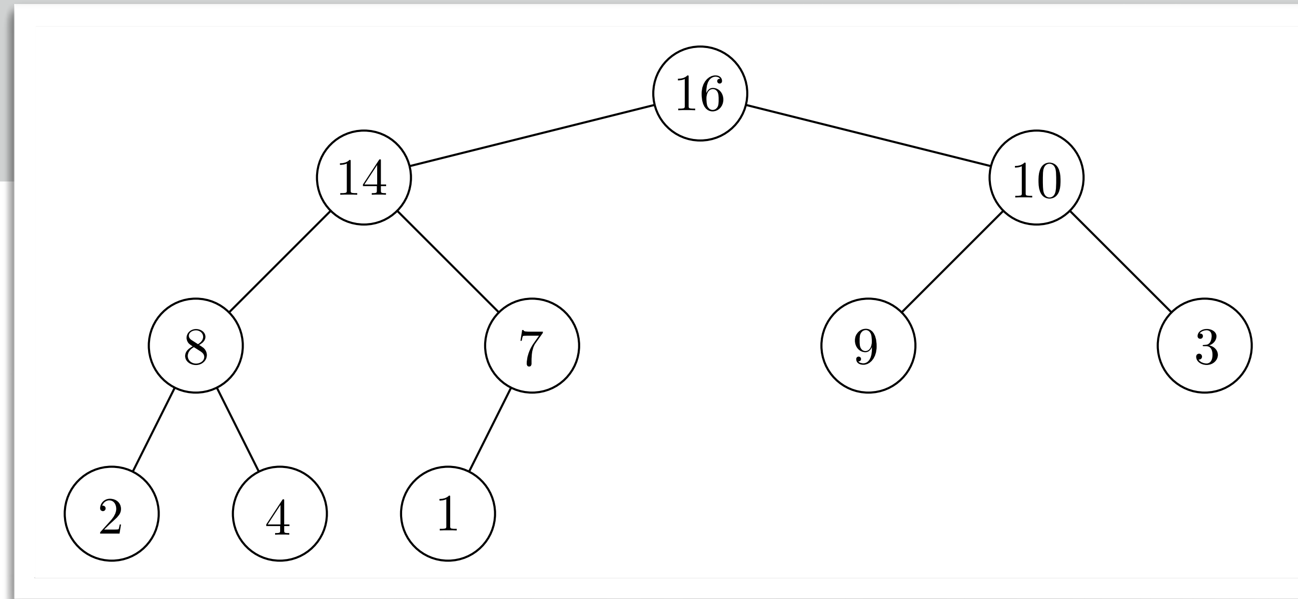
Min-Heap



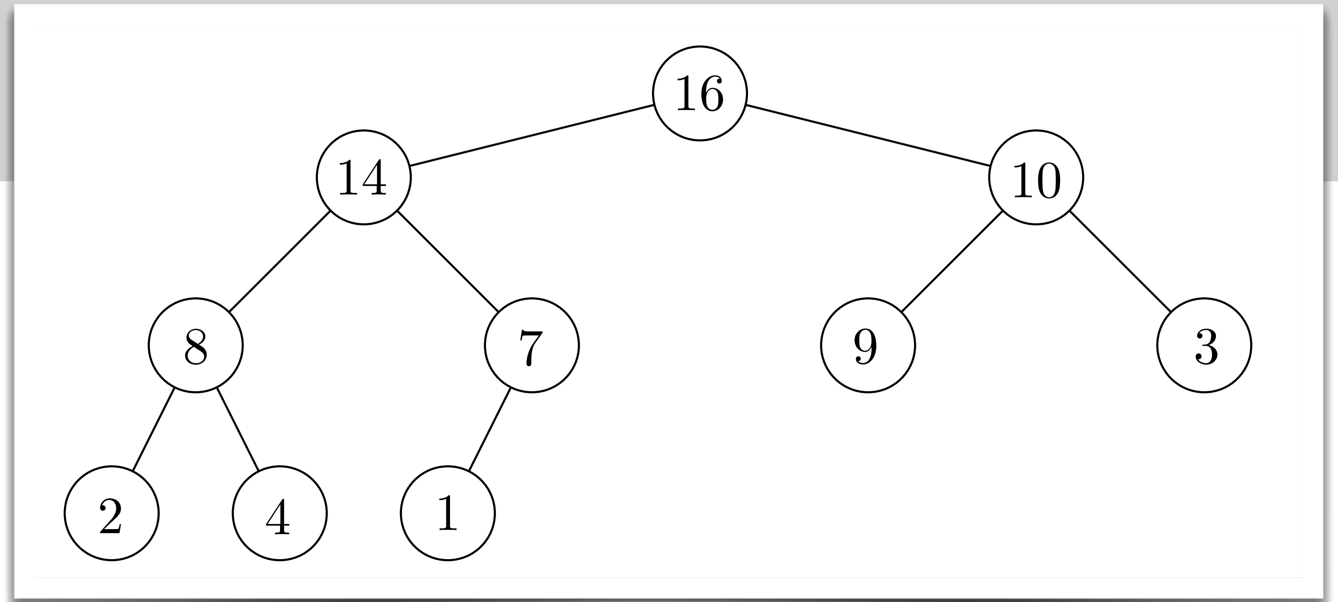
Eine Beziehung zwischen den Teilbäumen existiert nicht!

Heaps

Heaps

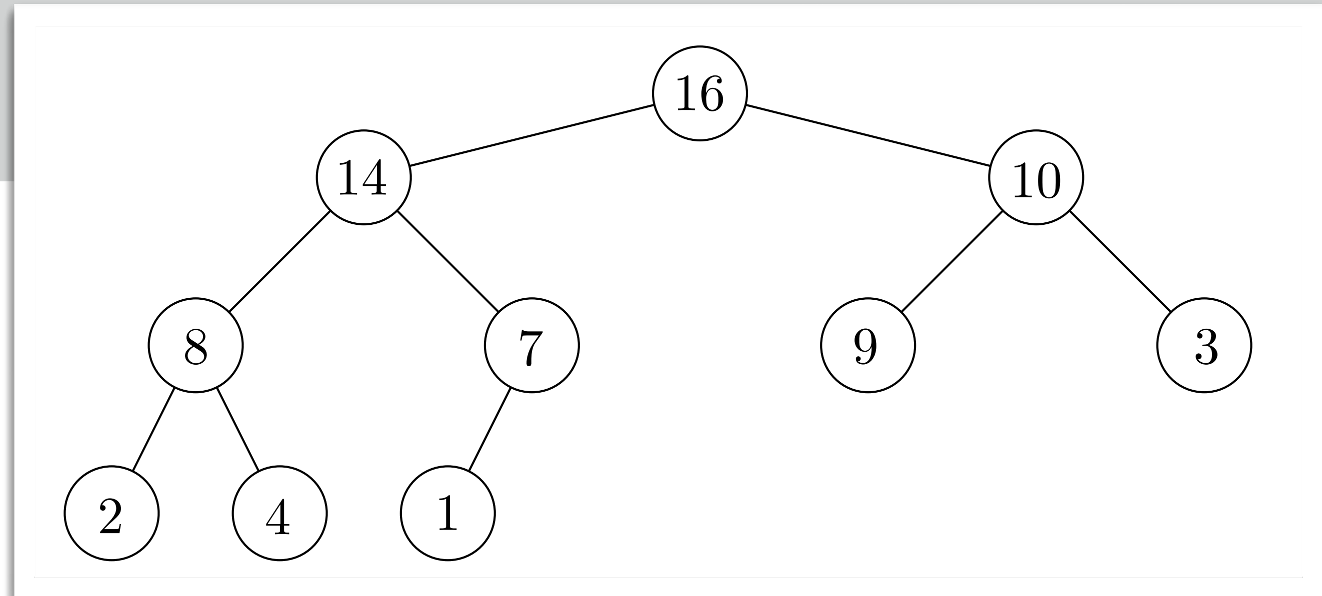


Heaps



Definition 4.14 (Heap)

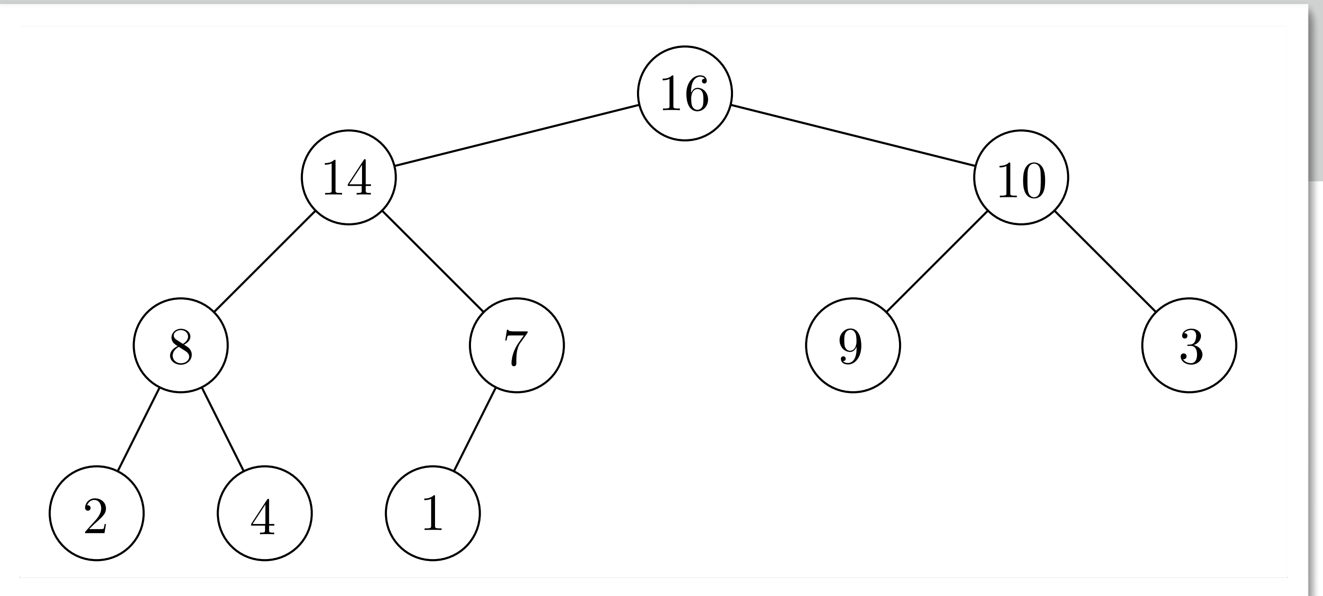
Heaps



Definition 4.14 (Heap)

Ein gerichteter binärer Baum heißt binärer Max-Heap, wenn folgende Eigenschaften erfüllt sind:

Heaps

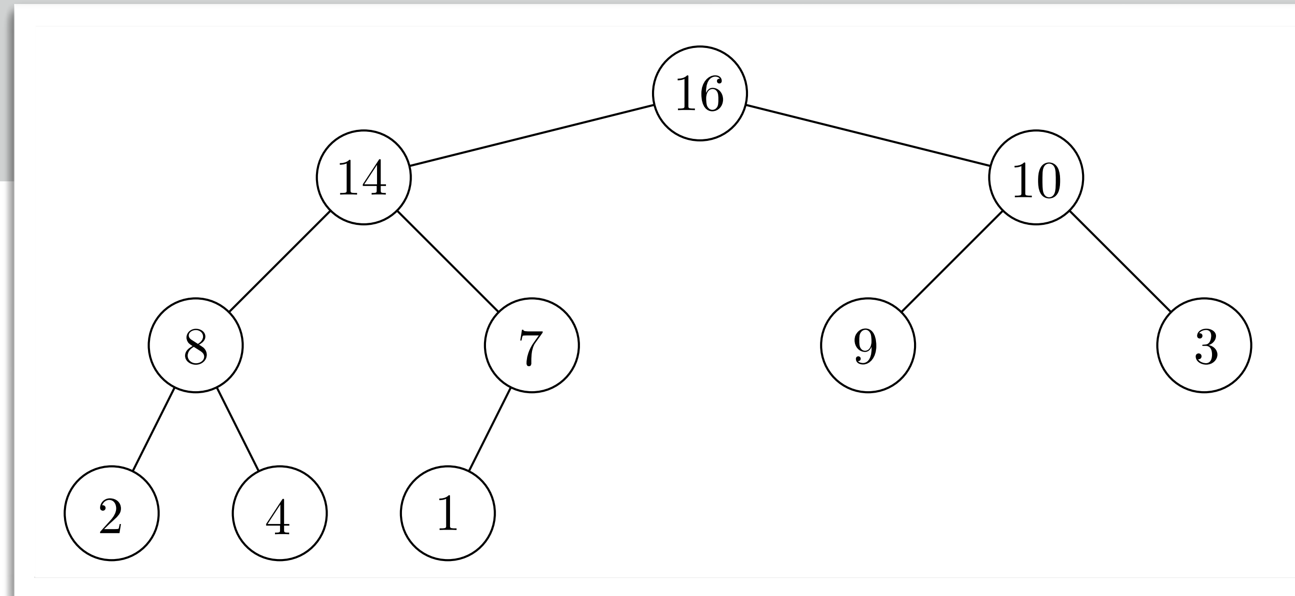


Definition 4.14 (Heap)

Ein gerichteter binärer Baum heißt binärer Max-Heap, wenn folgende Eigenschaften erfüllt sind:

1. Jeder Knoten hat einen Schlüssel.

Heaps

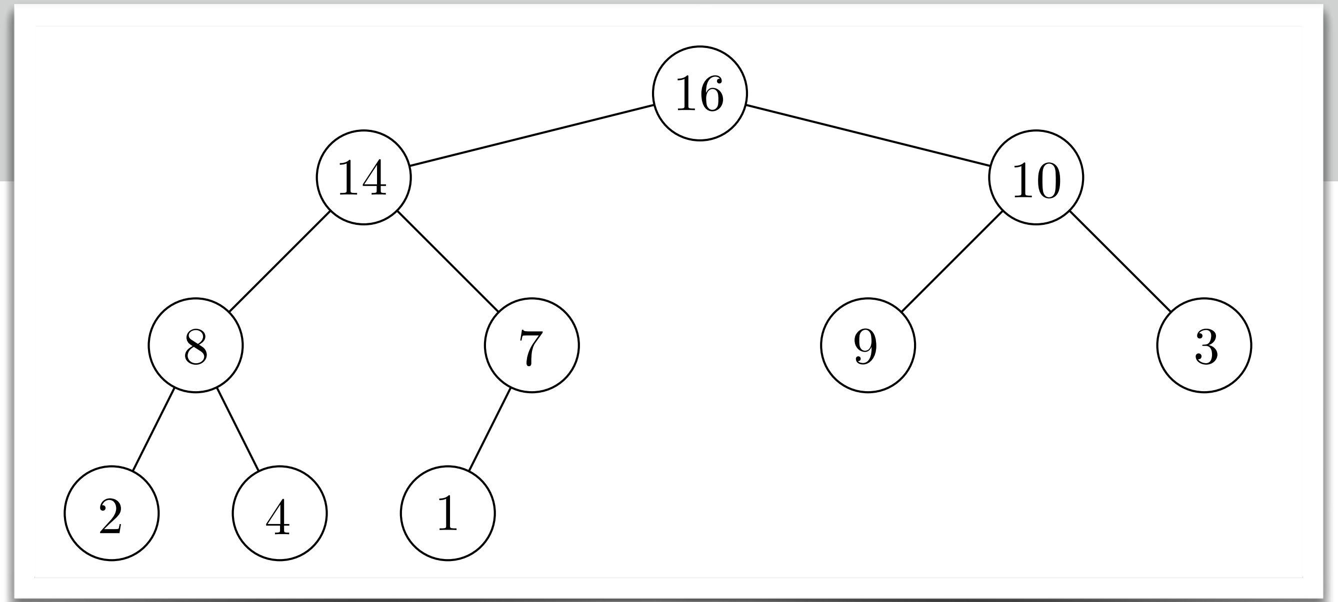


Definition 4.14 (Heap)

Ein gerichteter binärer Baum heißt binärer Max-Heap, wenn folgende Eigenschaften erfüllt sind:

1. Jeder Knoten hat einen Schlüssel.
2. Ist h die maximale Distanz von der Wurzel, dann haben alle Ebenen i < h genau zwei Knoten.

Heaps

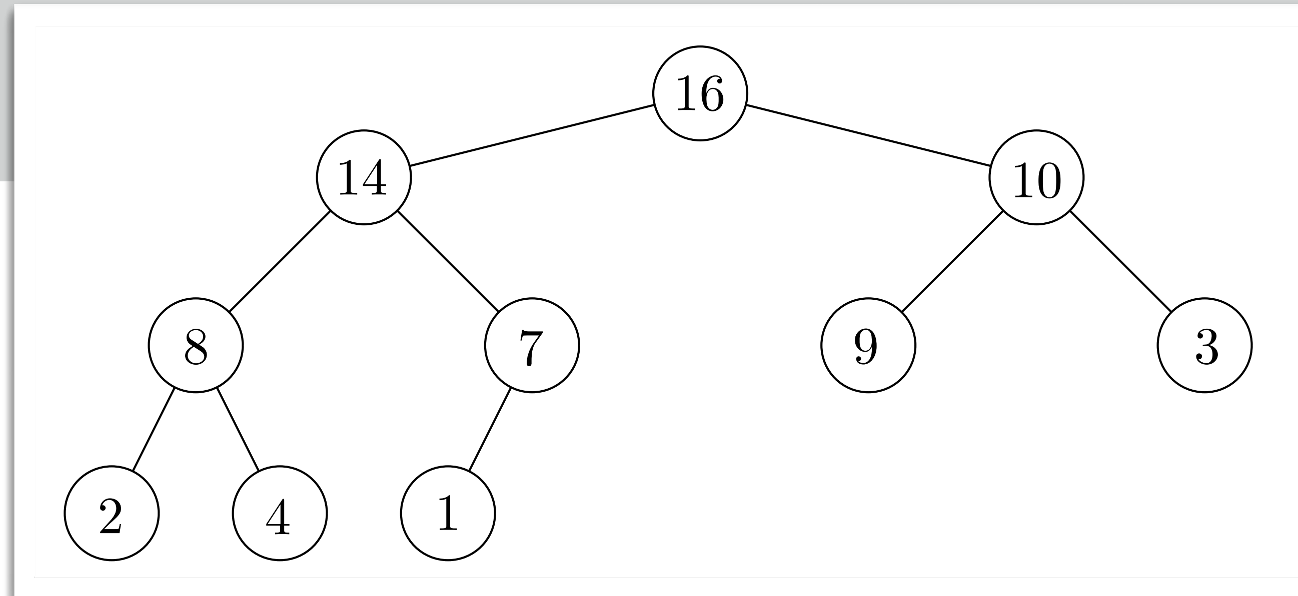


Definition 4.14 (Heap)

Ein gerichteter binärer Baum heißt binärer Max-Heap, wenn folgende Eigenschaften erfüllt sind:

1. Jeder Knoten hat einen Schlüssel.
2. Ist h die maximale Distanz von der Wurzel, dann haben alle Ebenen i < h genau zwei Knoten.
3. Auf Ebene h sind die linken $n - 2^h + 1$ Positionen besetzt.

Heaps

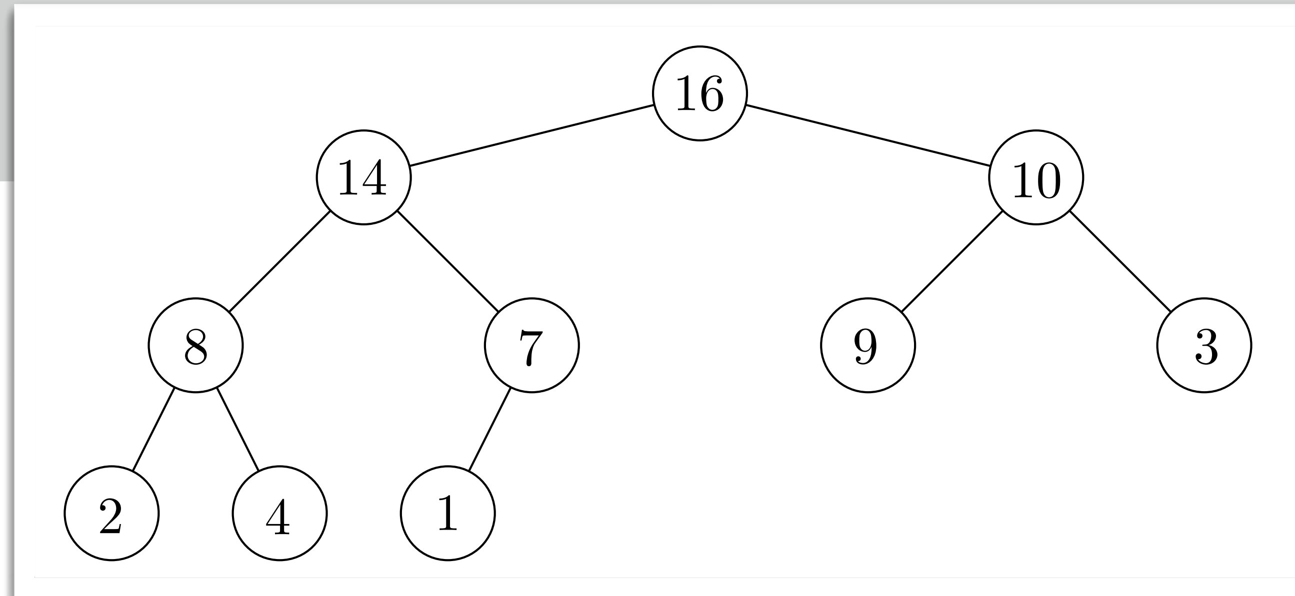


Definition 4.14 (Heap)

Ein gerichteter binärer Baum heißt binärer Max-Heap, wenn folgende Eigenschaften erfüllt sind:

1. Jeder Knoten hat einen Schlüssel.
2. Ist h die maximale Distanz von der Wurzel, dann haben alle Ebenen i < h genau zwei Knoten.
3. Auf Ebene h sind die linken $n - 2^h + 1$ Positionen besetzt.
4. Der Schlüssel jedes Knotens ist mindestens so groß wie die seiner Kinder.

Heaps



Definition 4.14 (Heap)

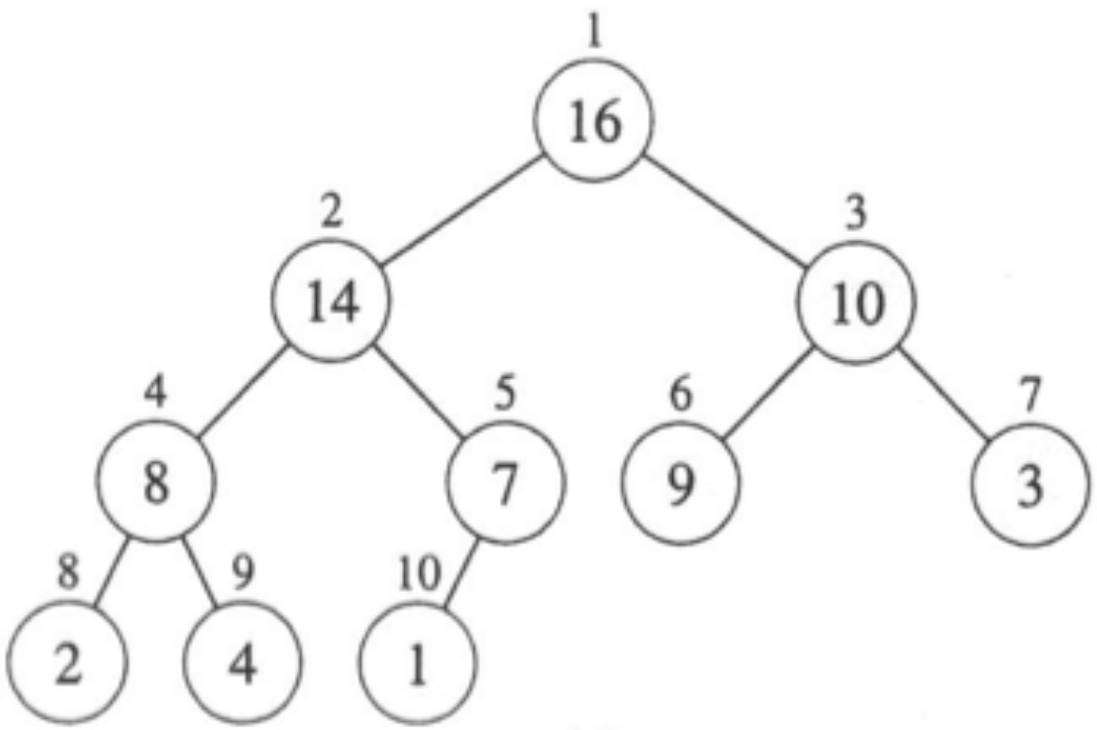
Ein gerichteter binärer Baum heißt binärer Max-Heap, wenn folgende Eigenschaften erfüllt sind:

1. Jeder Knoten hat einen Schlüssel.
2. Ist h die maximale Distanz von der Wurzel, dann haben alle Ebenen i < h genau zwei Knoten.
3. Auf Ebene h sind die linken $n - 2^h + 1$ Positionen besetzt.
4. Der Schlüssel jedes Knotens ist mindestens so groß wie die seiner Kinder.

Bei einem Min-Heap sind die Schlüssel eines Knotens höchstens so groß wie die seiner Kinder.

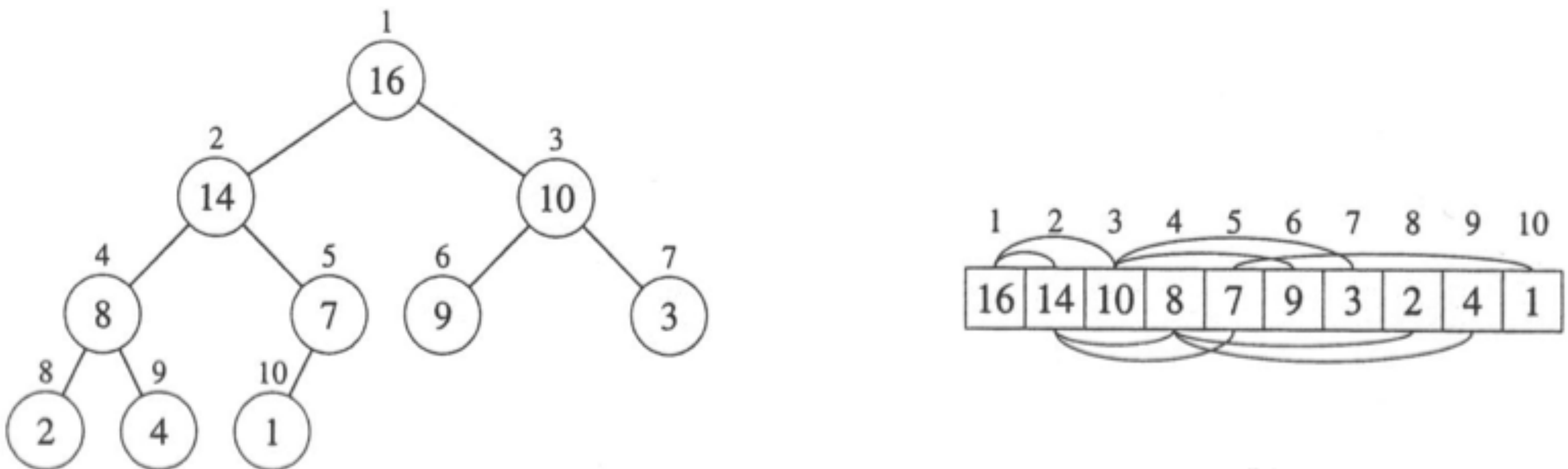
Heaps

Heaps



(a)

Heaps



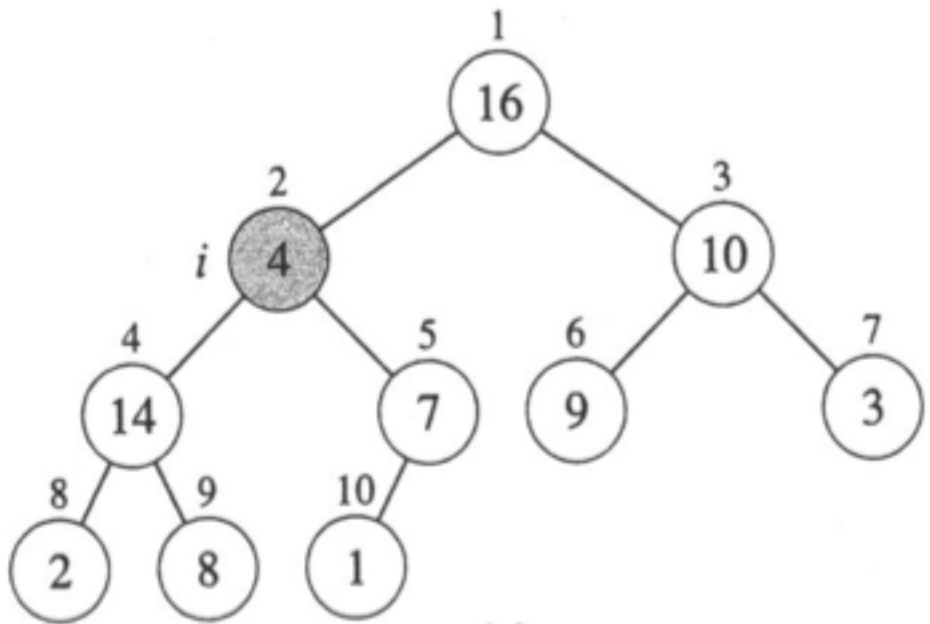
(a) (b)

Heaps — Umordnen

Heaps — Umordnen

```
MAX-HEAPIFY(A, i)
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ heap-größe[A] und A[l] > A[i]
    then maximum ← l
    else maximum ← i
if r ≤ heap-größe[A] und A[r] > A[maximum]
    then maximum ← r
if maximum ≠ i
    then vertausche A[i] ↔ A[maximum]
         MAX-HEAPIFY(A, maximum)
```

Heaps — Umordnen

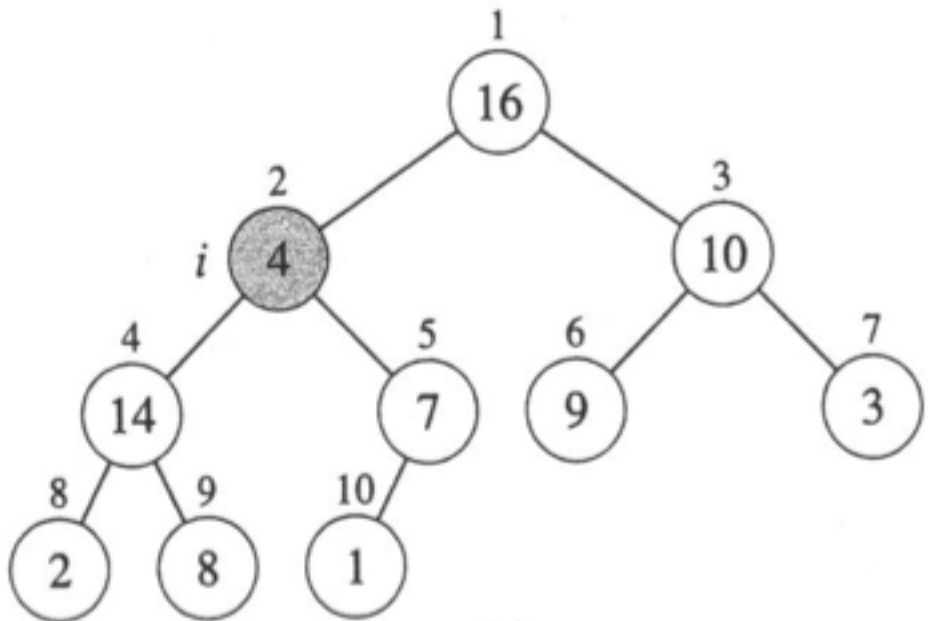


(a)

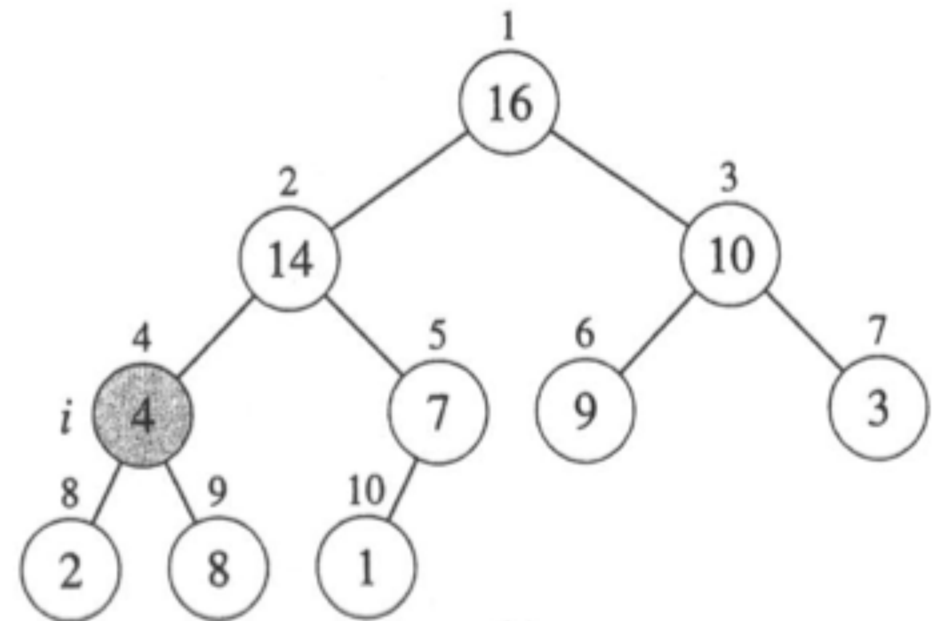
MAX-HEAPIFY(A, i)

```
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ heap-größe[A] und A[l] > A[i]
    then maximum ← l
    else maximum ← i
if r ≤ heap-größe[A] und A[r] > A[maximum]
    then maximum ← r
if maximum ≠ i
    then vertausche A[i] ↔ A[maximum]
            MAX-HEAPIFY(A, maximum)
```

Heaps — Umordnen



(a)

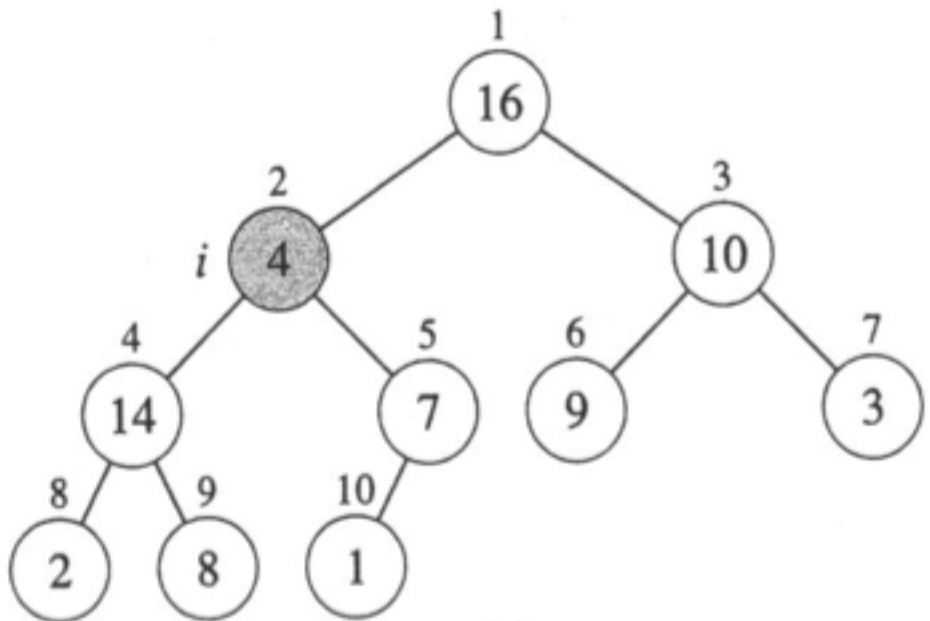


(b)

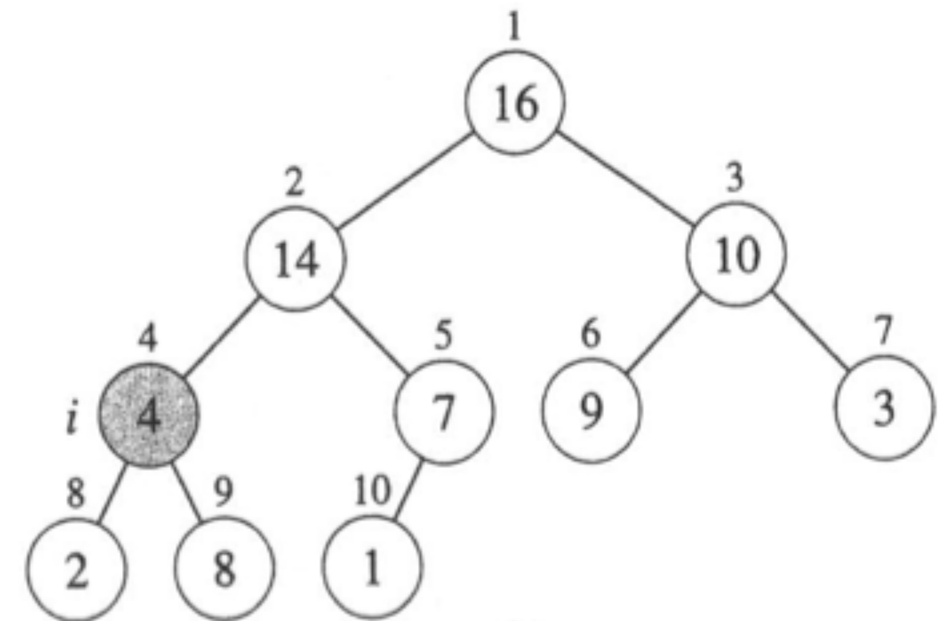
Max-Heapify(A, i)

$l \leftarrow$ Left(i)
$r \leftarrow$ Right(i)
if $l \leq$ *heap-größe*$[A]$ und $A[l] > A[i]$
  then *maximum* $\leftarrow l$
  else *maximum* $\leftarrow i$
if $r \leq$ *heap-größe*$[A]$ und $A[r] > A[$*maximum*$]$
  then *maximum* $\leftarrow r$
if *maximum* $\neq i$
  then vertausche $A[i] \leftrightarrow A[$*maximum*$]$
    Max-Heapify$(A,$ *maximum*$)$

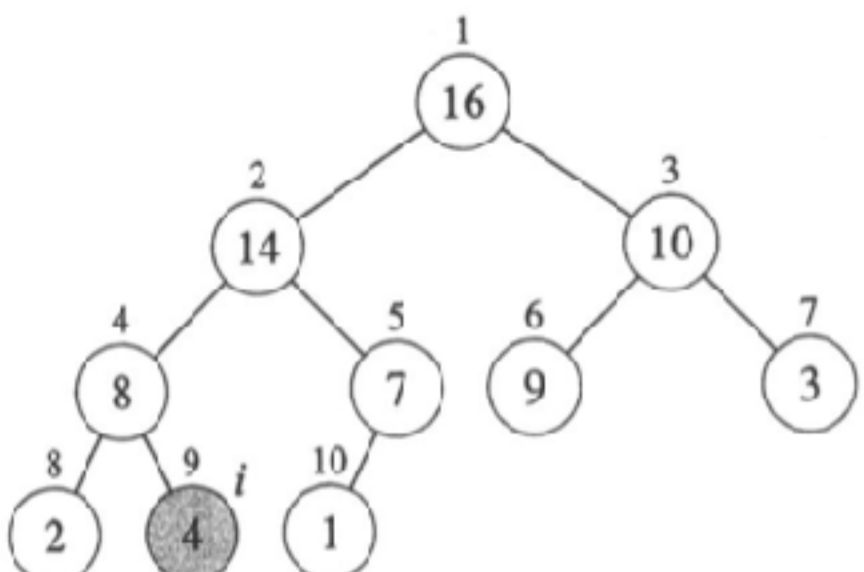
Heaps — Umordnen



(a)



(b)



(c)

Max-Heapify(A, i)

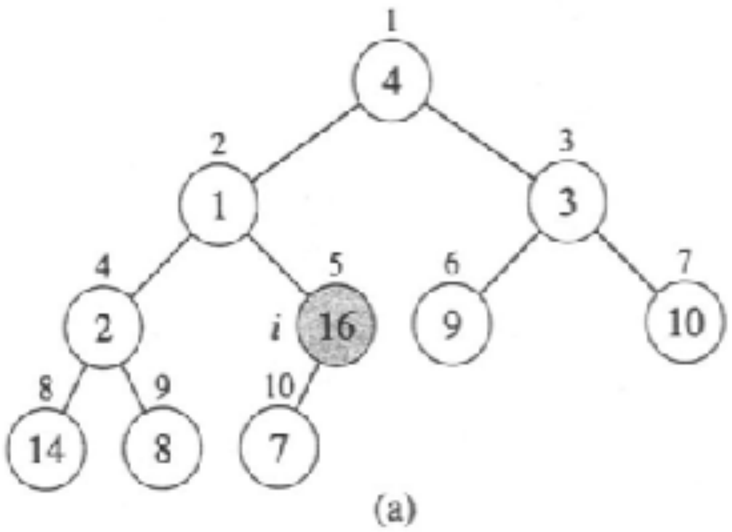
```
l ← Left(i)
r ← Right(i)
if l ≤ heap-größe[A] und A[l] > A[i]
    then maximum ← l
    else maximum ← i
if r ≤ heap-größe[A] und A[r] > A[maximum]
    then maximum ← r
if maximum ≠ i
    then vertausche A[i] ↔ A[maximum]
         Max-Heapify(A, maximum)
```

Heaps — Bauen

A	4	1	3	2	16	9	10	14	8	7

```
BUILD-MAX-HEAP(A)
    heap-größe[A] ← länge[A]
    for i ← ⌊länge[A]/2⌋ downto 1
            do MAX-HEAPIFY(A, i)
```

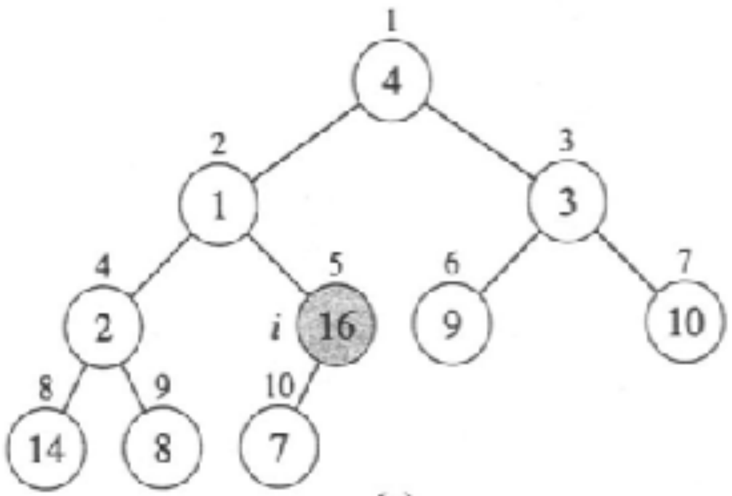
Heaps — Bauen



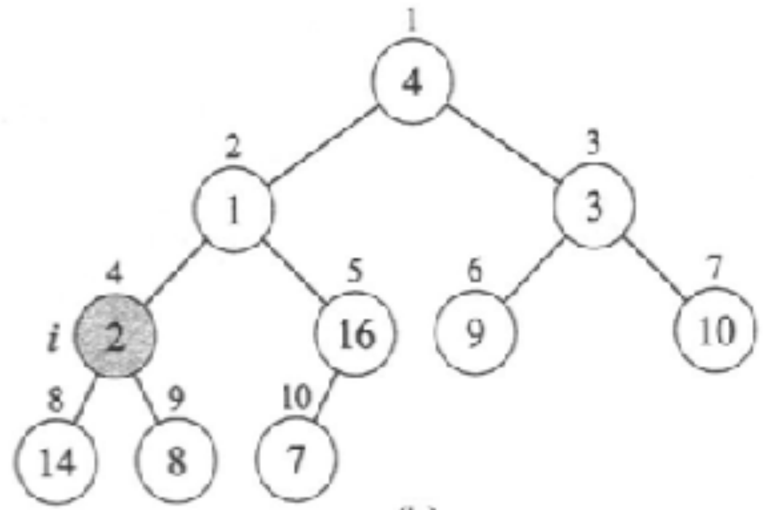
A									
4	1	3	2	16	9	10	14	8	7

BUILD-MAX-HEAP(A)

1 $\mathit{heap\text{-}gr\ddot{o}\ss e}[A] \leftarrow \mathit{l\ddot{a}nge}[A]$
2 **for** $i \leftarrow \lfloor \mathit{l\ddot{a}nge}[A]/2 \rfloor$ **downto** 1
3 **do** MAX-HEAPIFY(A, i)



(a)



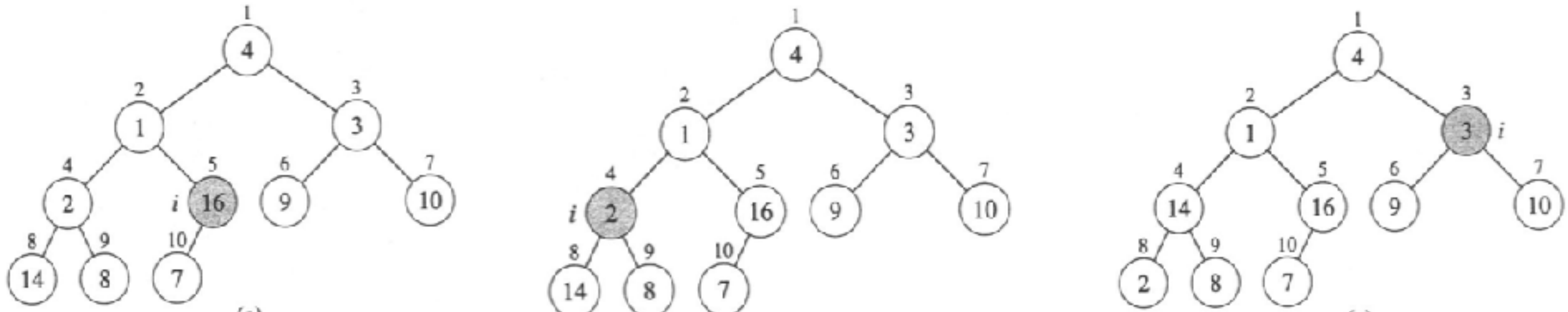
(b)

A	4	1	3	2	16	9	10	14	8	7

BUILD-MAX-HEAP(A)

$heap\text{-}größe[A] \leftarrow länge[A]$
for $i \leftarrow \lfloor länge[A]/2 \rfloor$ **downto** 1
  do MAX-HEAPIFY(A, i)

Heaps — Bauen



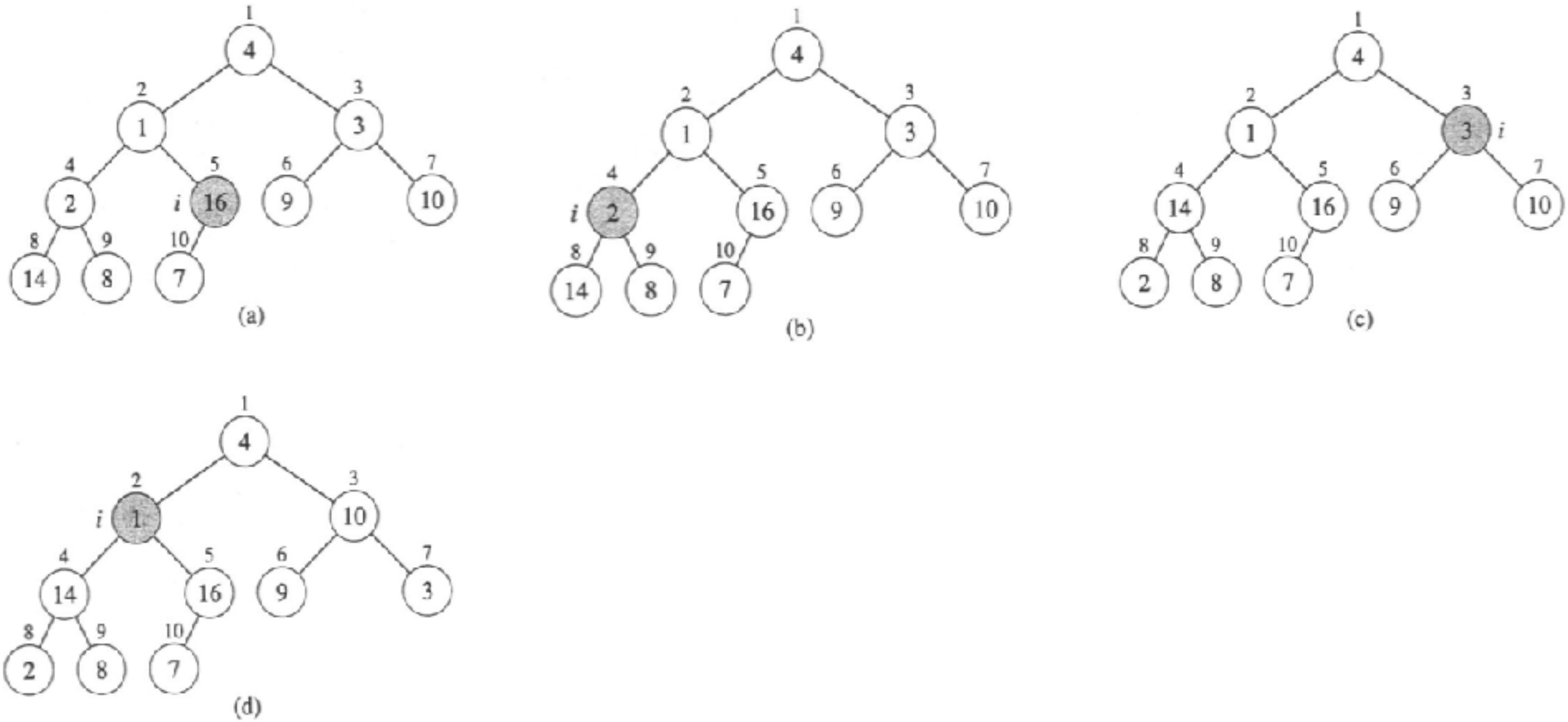
(a) (b) (c)

A									
4	1	3	2	16	9	10	14	8	7

BUILD-MAX-HEAP(A)

```
heap-größe[A] ← länge[A]
for i ← ⌊länge[A]/2⌋ downto 1
    do MAX-HEAPIFY(A, i)
```

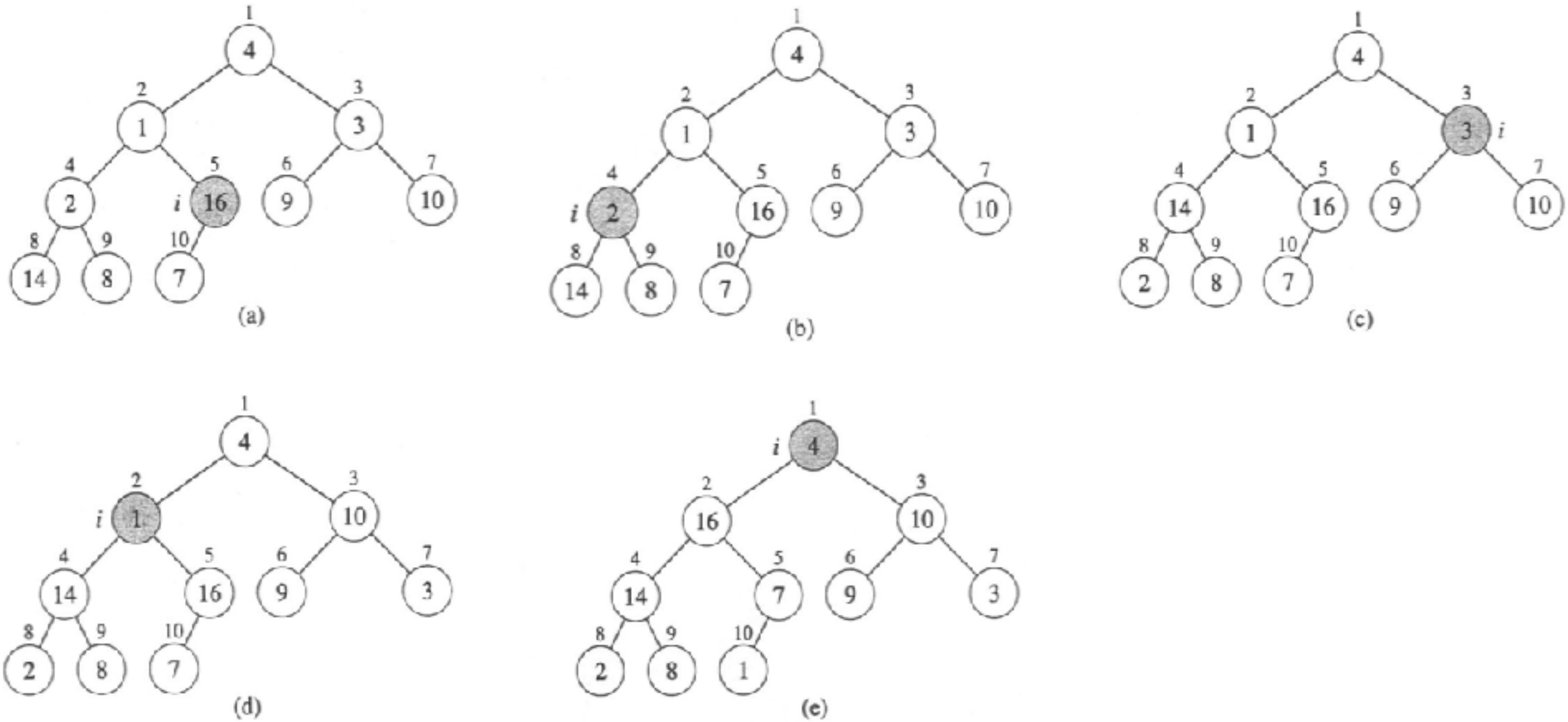
Heaps — Bauen



A									
4	1	3	2	16	9	10	14	8	7

```
BUILD-MAX-HEAP(A)
heap-größe[A] ← länge[A]
for i ← ⌊länge[A]/2⌋ downto 1
    do MAX-HEAPIFY(A, i)
```

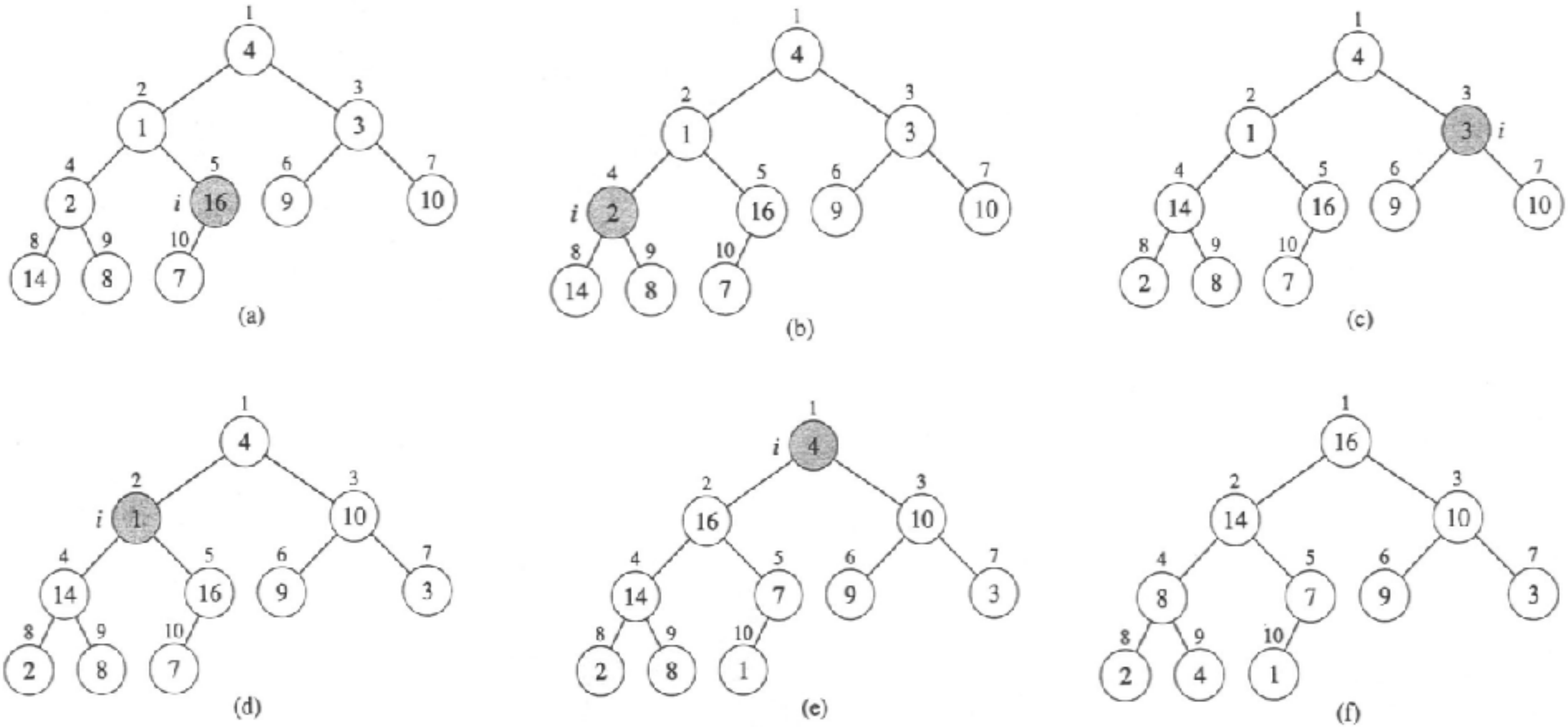
Heaps — Bauen



A									
4	1	3	2	16	9	10	14	8	7

```
BUILD-MAX-HEAP(A)
heap-größe[A] ← länge[A]
for i ← ⌊länge[A]/2⌋ downto 1
        do MAX-HEAPIFY(A, i)
```

Heaps — Bauen



A									
4	1	3	2	16	9	10	14	8	7

```
BUILD-MAX-HEAP(A)
heap-größe[A] ← länge[A]
for i ← ⌊länge[A]/2⌋ downto 1
        do MAX-HEAPIFY(A, i)
```

Heaps

Heaps

Da man $O(\log n)$ Operationen pro Level benötigt, ist klar, dass ein Max-Heap in $O(n \log n)$ gebaut werden kann.

Heaps

Da man $O(\log n)$ Operationen pro Level benötigt, ist klar, dass ein Max-Heap in $O(n \log n)$ gebaut werden kann.

Da aber mehr Knoten auf niedrigerem Niveau sind, was weniger Arbeit erfordert, kann man diese Abschätzung noch verbessern:

Heaps

Da man $O(\log n)$ Operationen pro Level benötigt, ist klar, dass ein Max-Heap in $O(n \log n)$ gebaut werden kann.

Da aber mehr Knoten auf niedrigerem Niveau sind, was weniger Arbeit erfordert, kann man diese Abschätzung noch verbessern:

Satz 4.15
Ein Max-Heap mit n Knoten kann in $O(n)$ gebaut werden.

4.11 Weitere Varianten

4.11.1 Fibonacci Heaps

4.11.1 Fibonacci Heaps

Heap-Struktur mit sehr schneller amortisierter Zugriffszeit.

4.11.1 Fibonacci Heaps

Heap-Struktur mit sehr schneller amortisierter Zugriffszeit.

4.11.1 Fibonacci Heaps

Heap-Struktur mit sehr schneller amortisierter Zugriffszeit.

“durchschnittliche” Kosten

4.11.1 Fibonacci Heaps

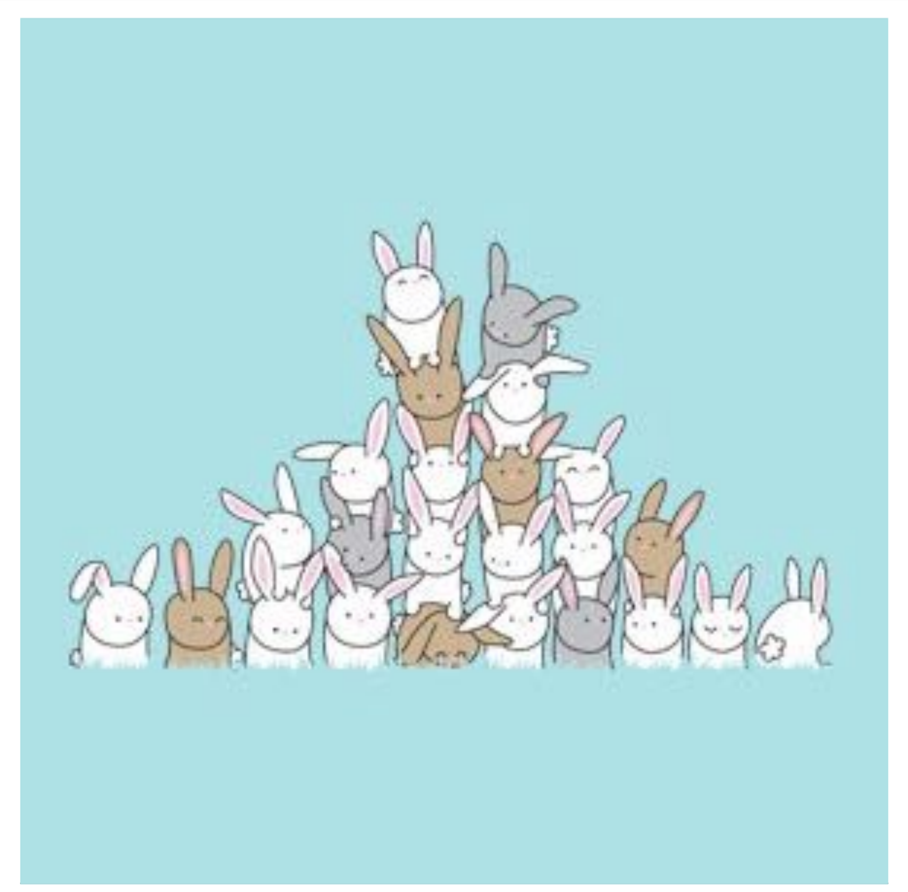
Heap-Struktur mit sehr schneller amortisierter Zugriffszeit.

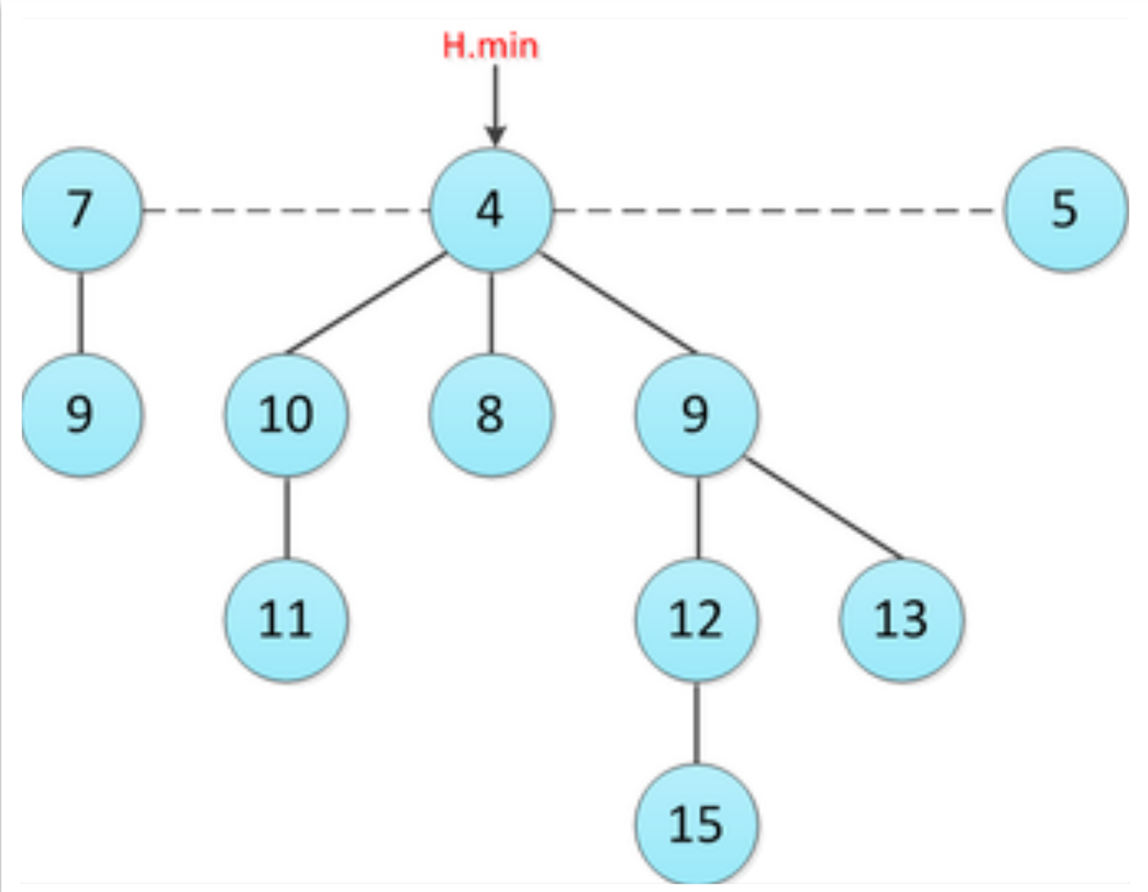
“durchschnittliche” Kosten

4.11.1 Fibonacci Heaps

Heap-Struktur mit sehr schneller amortisierter Zugriffszeit.

“durchschnittliche” Kosten





4.11.1 Fibonacci Heaps

Heap-Struktur mit sehr schneller amortisierter Zugriffszeit.

“durchschnittliche” Kosten

Operation	Lineare Liste	Sortierte Liste	(Min-)Heap	Unbalancierter Binärbaum	Fibonacci-Heap
insert	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)^*$	$\mathcal{O}(1)$
getMin	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
extractMin	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)^*$
decreaseKey	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)^*$	$\mathcal{O}(1)^*$
remove	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)^{**}$	$\mathcal{O}(\log n)^*$
merge	$\mathcal{O}(1)$	$\mathcal{O}(n+m)$	$\mathcal{O}(m \cdot \log(n+m))$	$\mathcal{O}(n+m)$	$\mathcal{O}(1)$

H.min

15

4.11.2 Cache-Oblivious B-Trees

Umgang mit großen Datenmengen bei unbekannter Cache-Größe

4.11.2 Cache-Oblivious B-Trees

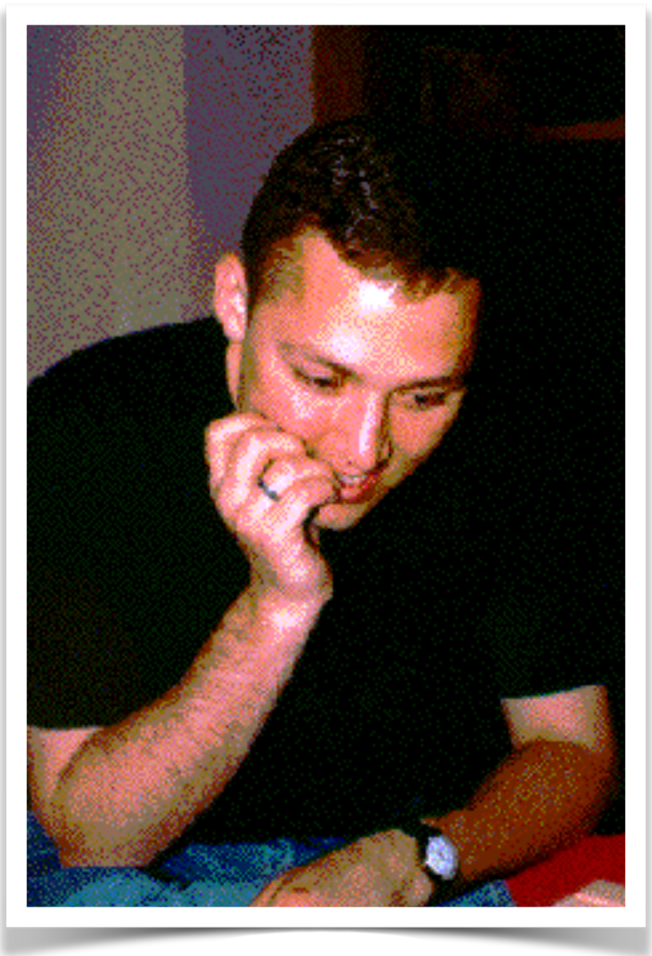
Umgang mit großen Datenmengen bei unbekannter Cache-Größe

Michael Bender

4.11.2 Cache-Oblivious B-Trees

Umgang mit großen Datenmengen bei unbekannter Cache-Größe

Michael Bender



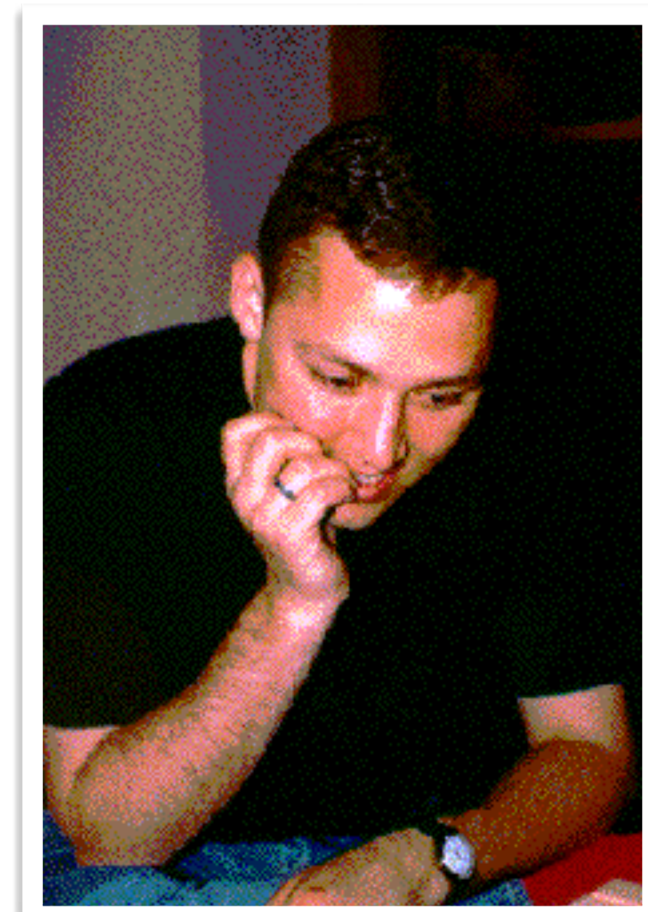
Martin Farach-Colton

4.11.2 Cache-Oblivious B-Trees

Umgang mit großen Datenmengen bei unbekannter Cache-Größe

Michael Bender

Erik Demaine



Martin Farach-Colton

4.11.2 Cache-Oblivious B-Trees

Umg … röße

CACHE-OBLIVIOUS B-TREES*

MICHAEL A. BENDER†, ERIK D. DEMAINE‡, AND MARTIN FARACH-COLTON§

Abstract. This paper presents two dynamic search trees attaining near-optimal performance on any hierarchical memory. The data structures are independent of the parameters of the memory hierarchy, e.g., the number of memory levels, the block-transfer size at each level, and the relative speeds of memory levels. The performance is analyzed in terms of the number of memory transfers between two memory levels with an arbitrary block-transfer size of B; this analysis can then be applied to every adjacent pair of levels in a multilevel memory hierarchy. Both search trees match the optimal search bound of $\Theta(1+\log_{B+1} N)$ memory transfers. This bound is also achieved by the classic B-tree data structure on a two-level memory hierarchy with a known block-transfer size B. The first search tree supports insertions and deletions in $\Theta(1+\log_{B+1} N)$ amortized memory transfers, which matches the B-tree's worst-case bounds. The second search tree supports scanning S consecutive elements optimally in $\Theta(1+S/B)$ memory transfers and supports insertions and deletions in $\Theta(1+\log_{B+1} N + \frac{\log^2 N}{B})$ amortized memory transfers, matching the performance of the B-tree for $B = \Omega(\log N \log\log N)$.

Key words. Memory hierarchy, cache efficiency, data structures, search trees

AMS subject classifications. 68P05, 68P30, 68P20

DOI. 10.1137/S0097539701389956

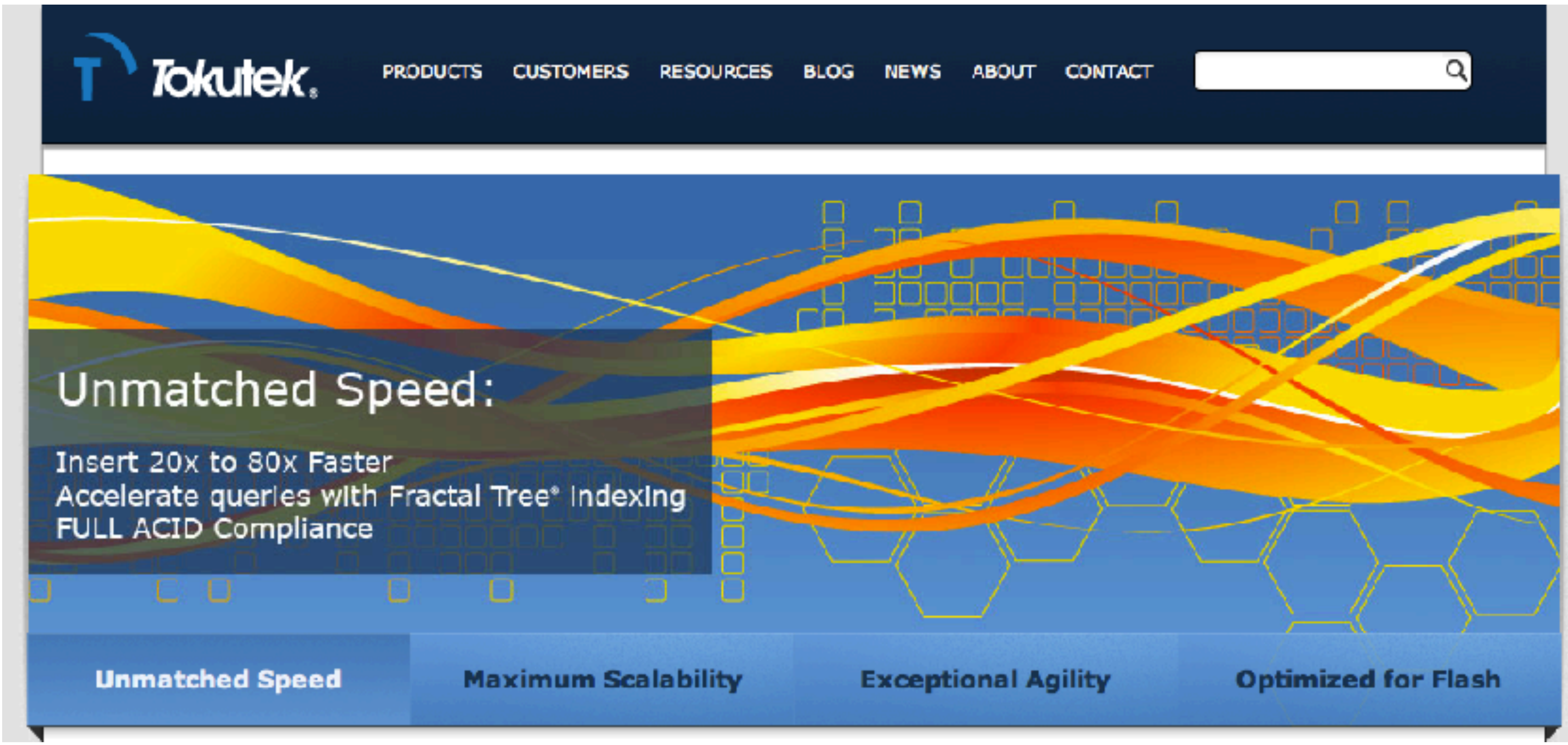
ton

4.11.2 Cache-Oblivious B-Trees

Umg … röße

CACHE-OBLIVIOUS B-TREES*

MICHAEL A. BENDER†, ERIK D. DEMAINE‡, AND MARTIN FARACH-COLTON§



… ton

DOI. 10.1137/S0097539701389956

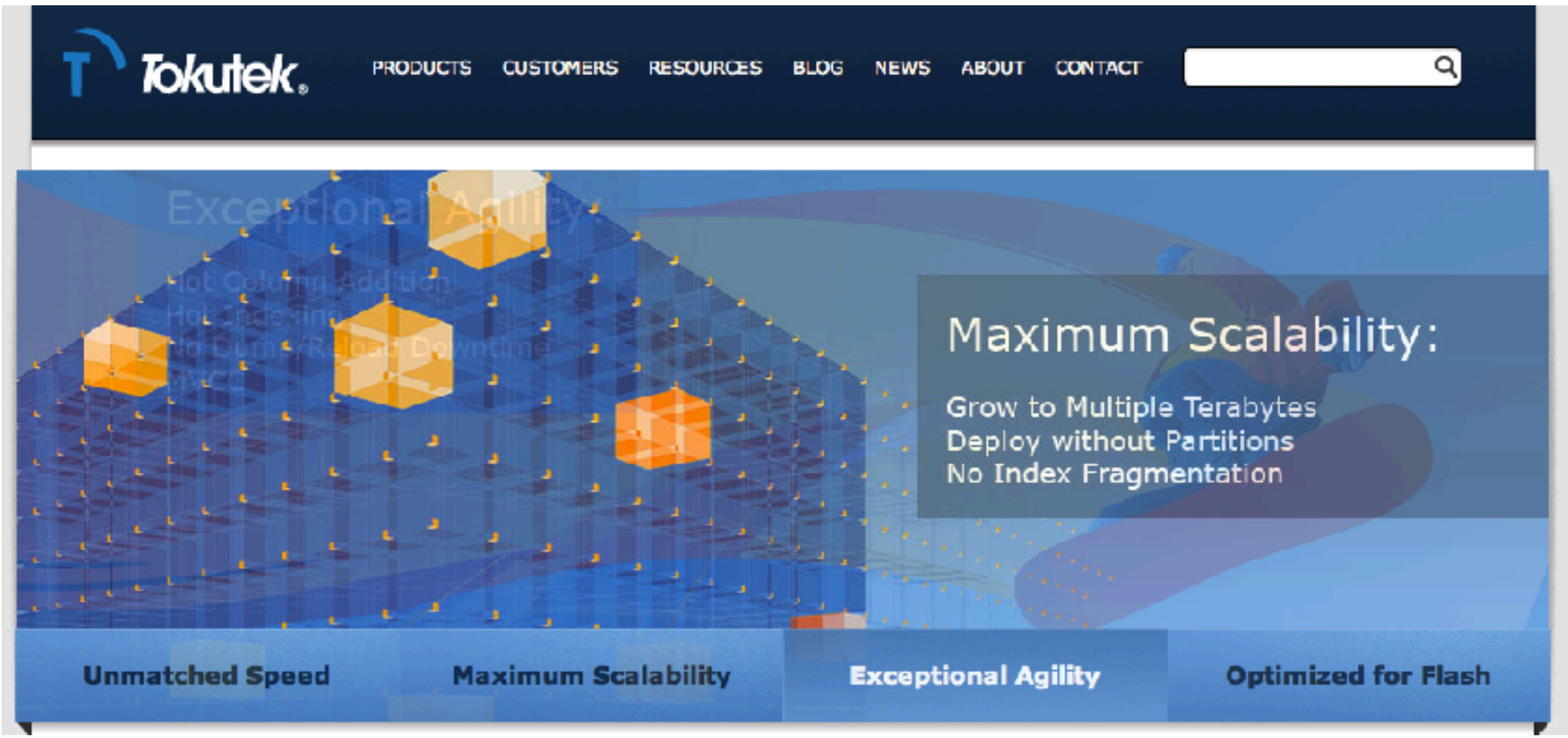
4.11.2 Cache-Oblivious B-Trees

Umg

öße

CACHE-OBLIVIOUS B-TREES*

MICHAEL A. BENDER†, ERIK D. DEMAINE‡, AND MARTIN FARACH-COLTON§

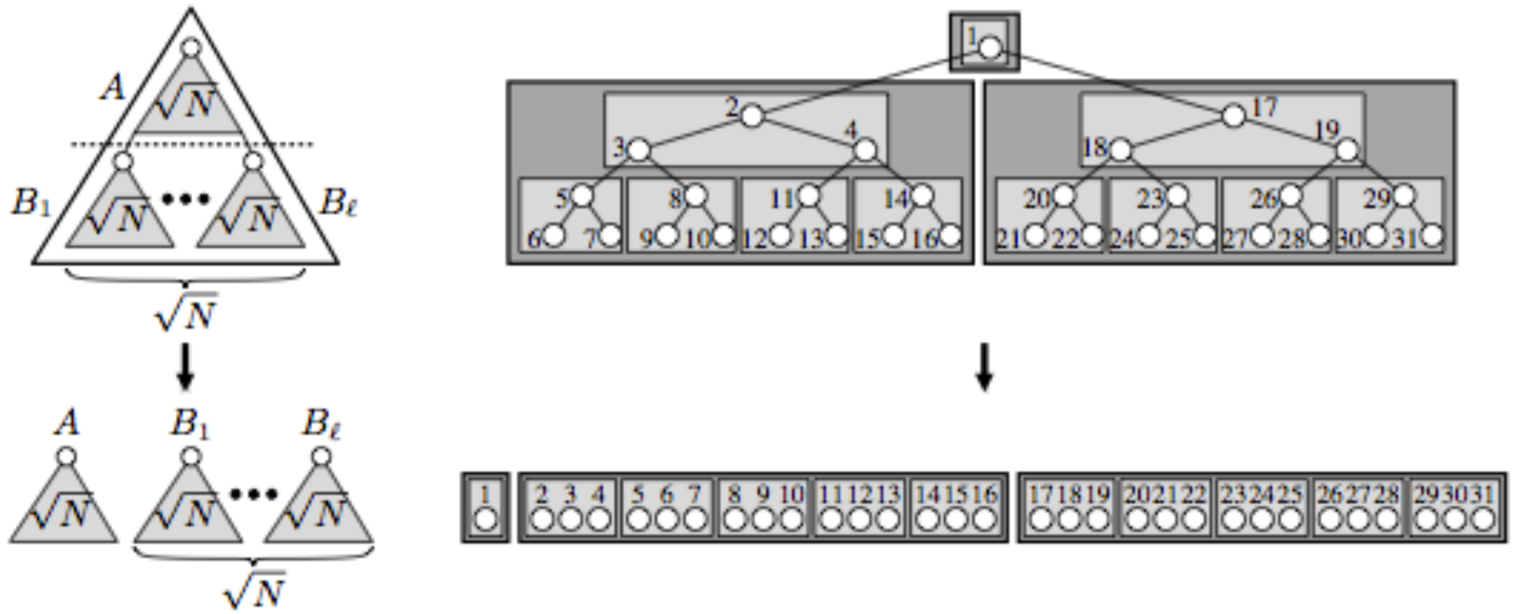


ton

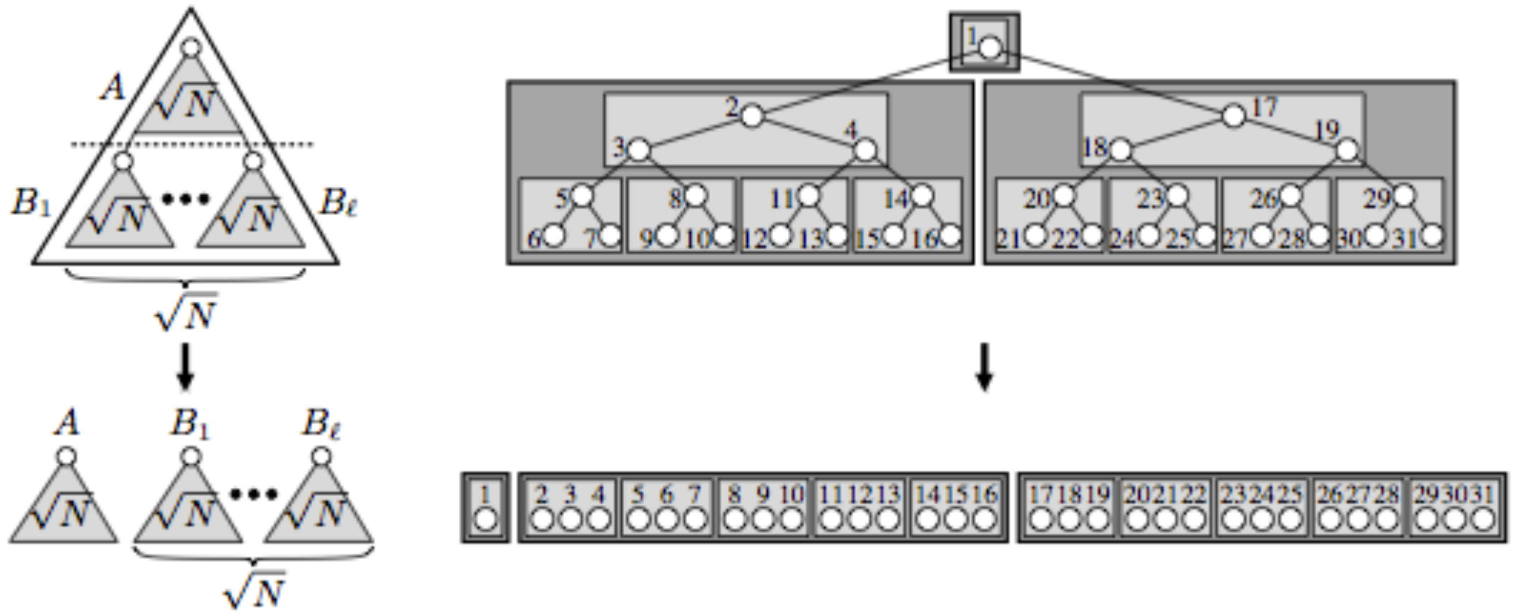
DOI. 10.1137/S0097539701389956

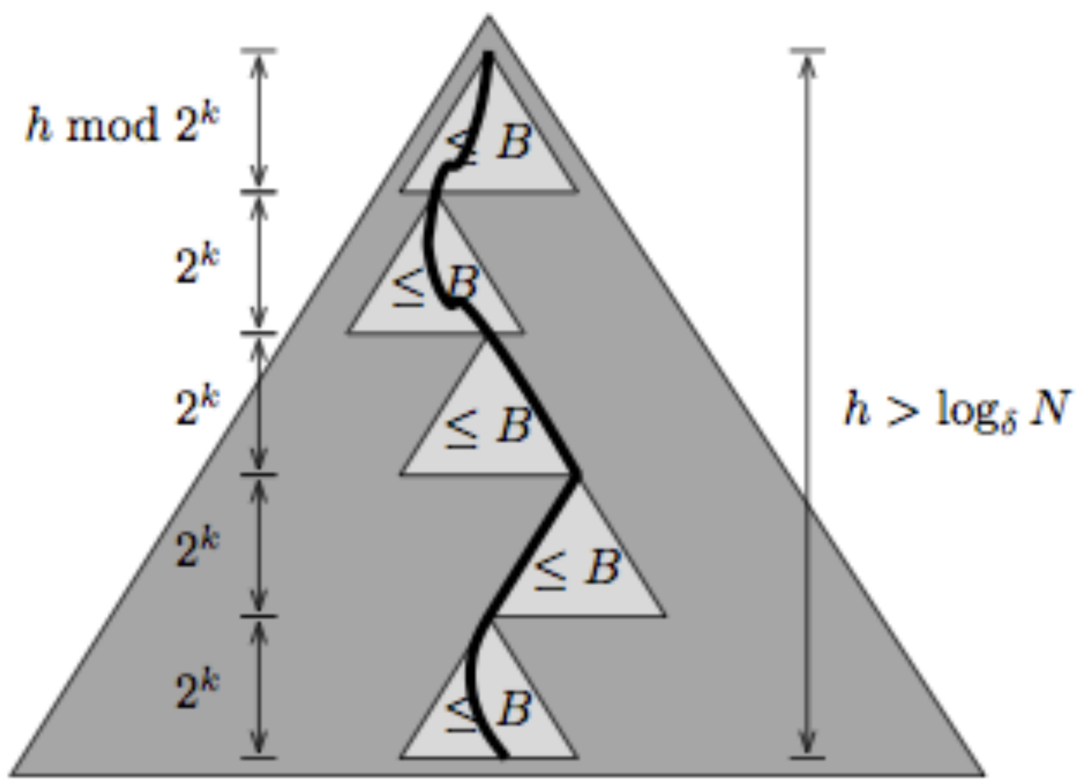
Cache-Oblivious B-Trees

Cache-Oblivious B-Trees



Cache-Oblivious B-Trees





Vielen Dank!