Technische Universität Braunschweig

Kapitel 4.8 - 4.11
Andere dynamische Datenstrukturen

Algorithmen und Datenstrukturen
Wintersemester 2023/24

Sándor Fekete, Ramin Kosfeld, Chek-Manh Loi

Prof. Michael Bender, Stony Brook University, New York

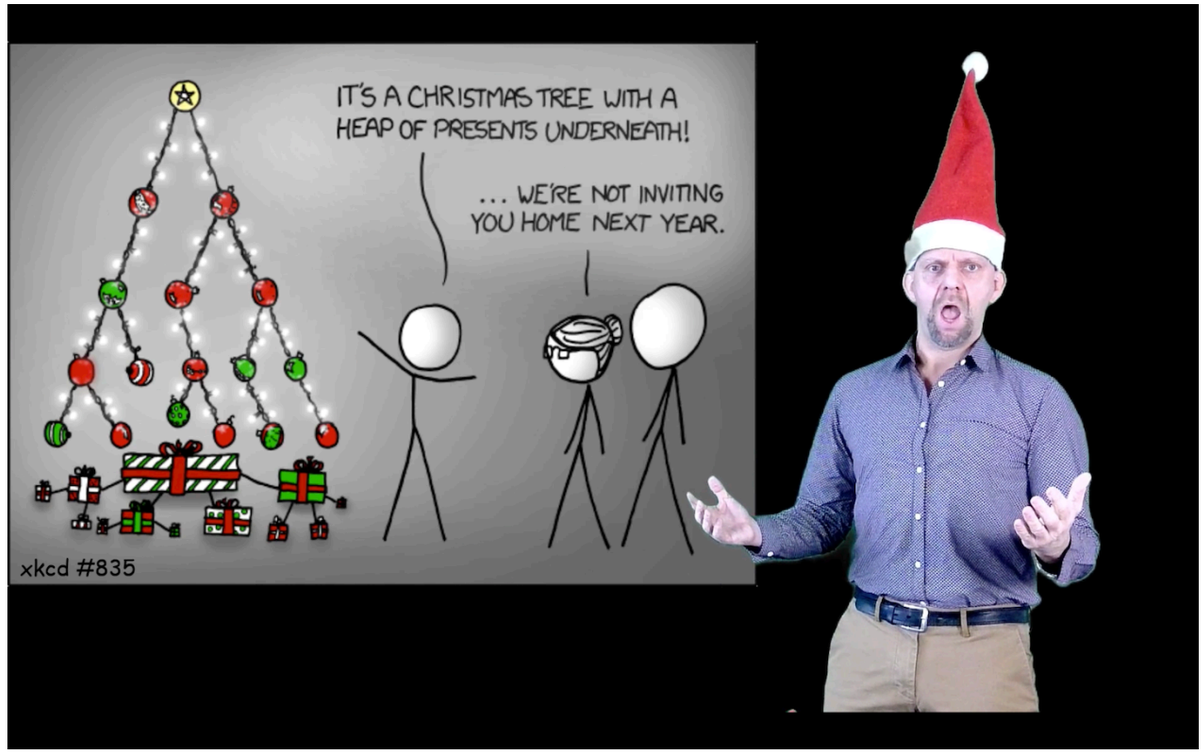
Agenda

- 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 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.

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

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.

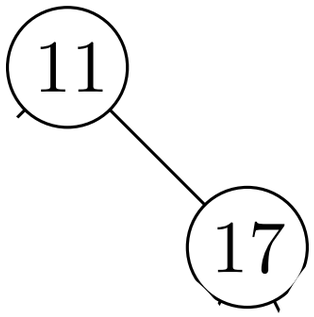
Binäre Bäume (Wiederholung)

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)

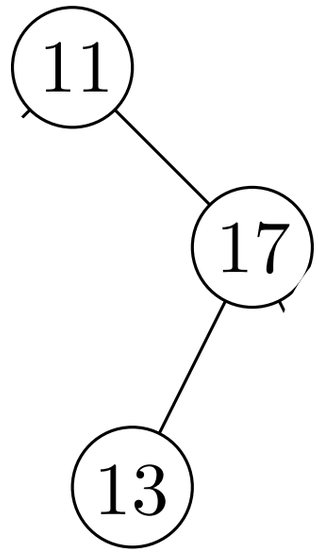
Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.



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

Binäre Bäume (Wiederholung)

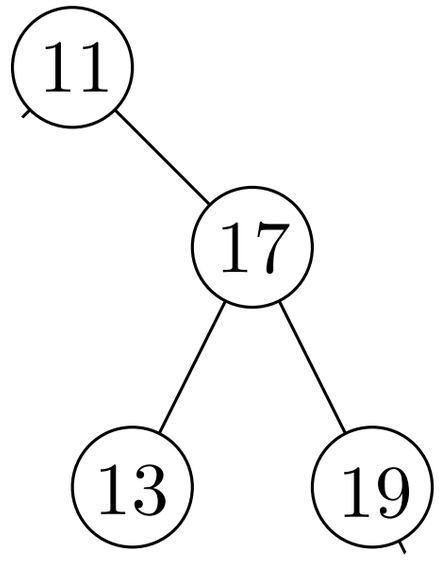
Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.



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

Binäre Bäume (Wiederholung)

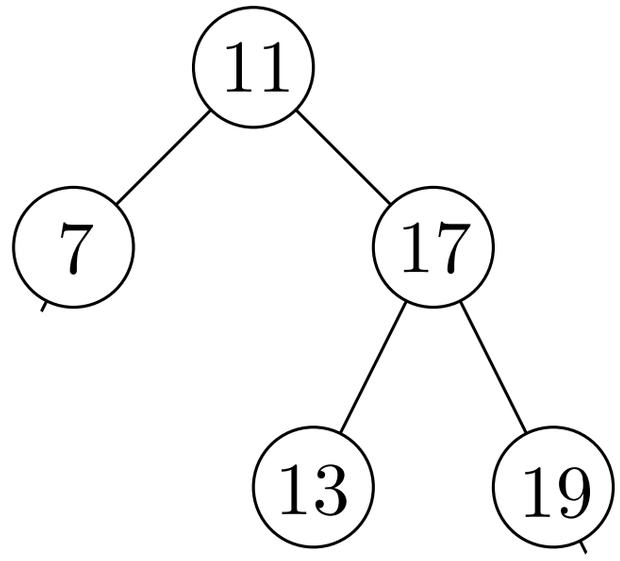
Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.



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

Binäre Bäume (Wiederholung)

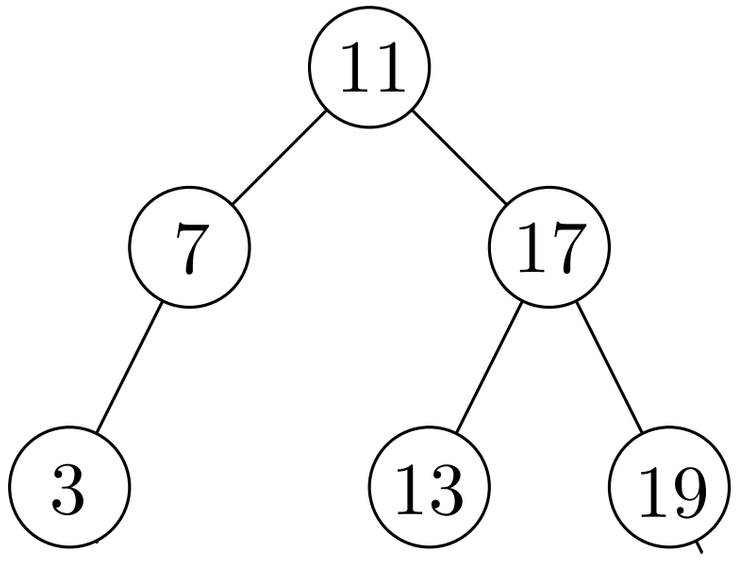
Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.



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

Binäre Bäume (Wiederholung)

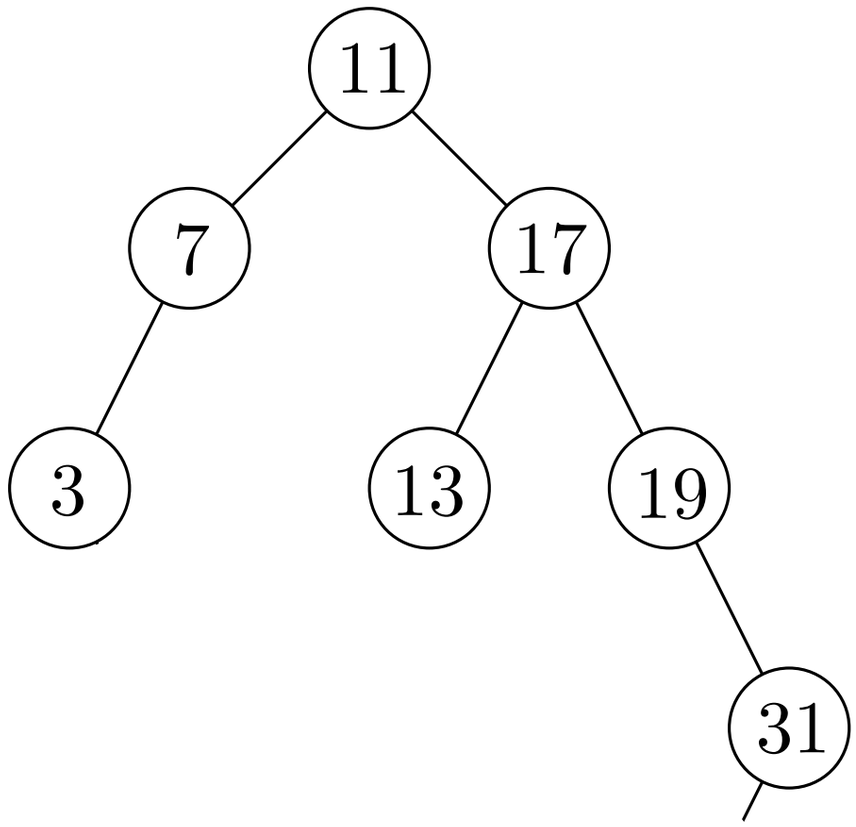
Seien folgende Elemente gegeben:
11, 17, 13, 19, 7, 3, 31, 23, 5, 29.



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.

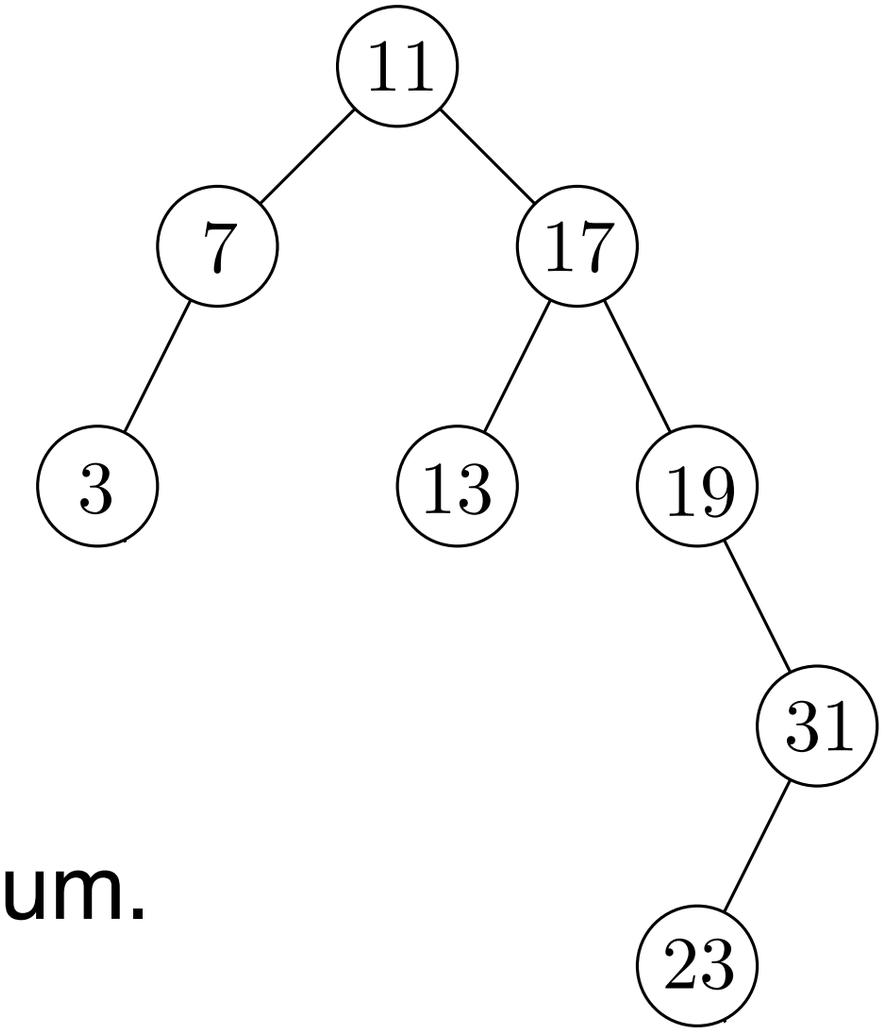


Das Einfügen dieser Elemente von links nach rechts in einen binären Suchbaum ergibt den folgenden Baum.

Binäre Bäume (Wiederholung)

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

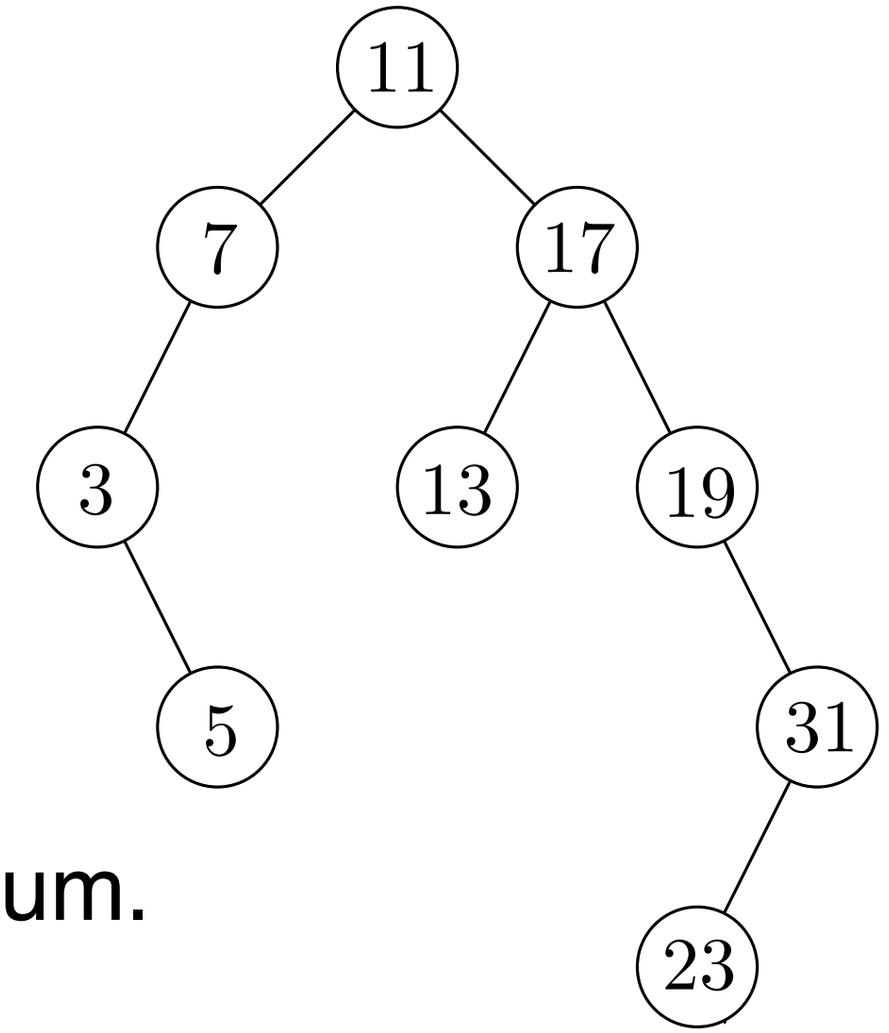
Das Einfügen dieser Elemente von links nach rechts in einen binären Suchbaum ergibt den folgenden Baum.



Binäre Bäume (Wiederholung)

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

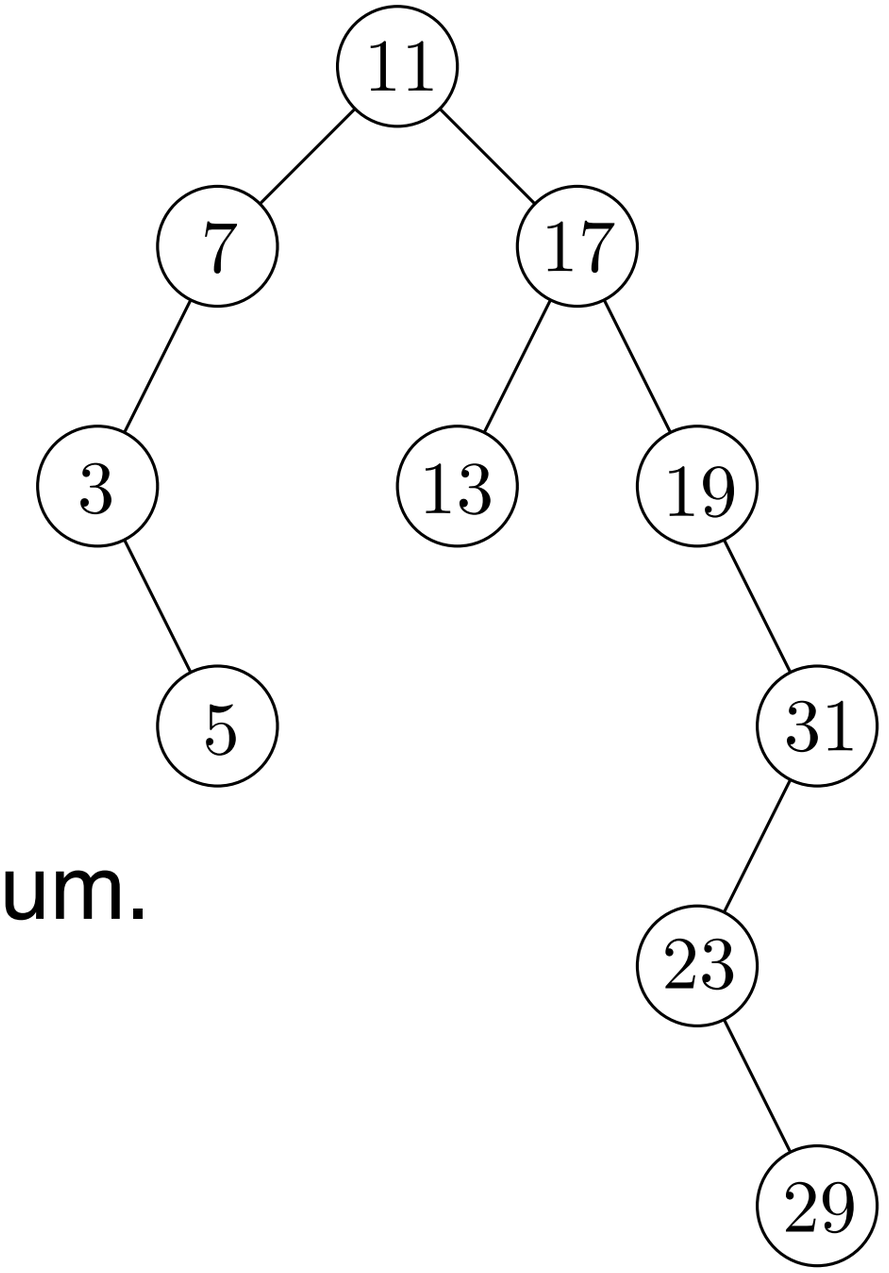
Das Einfügen dieser Elemente von links nach rechts in einen binären Suchbaum ergibt den folgenden Baum.



Binäre Bäume (Wiederholung)

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

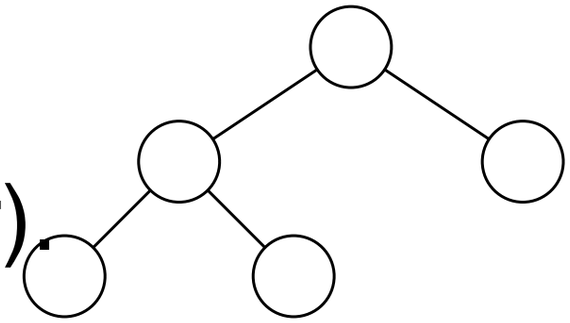
Das Einfügen dieser Elemente von links nach rechts in einen binären Suchbaum ergibt den folgenden Baum.



Binäre Bäume (Wiederholung)

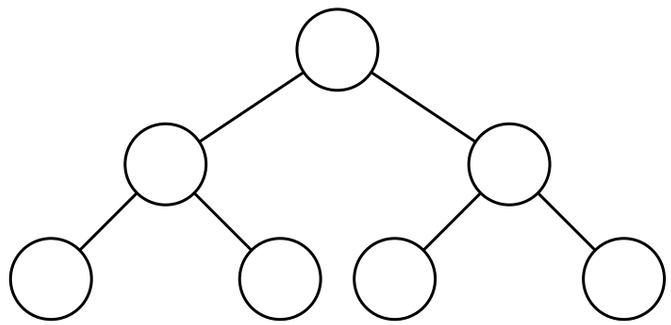
Voller binärer Baum:

- Jeder Knoten hat kein oder zwei Kind(er).



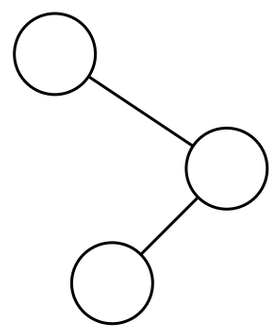
Vollständiger binärer Baum:

- Voller Baum und alle Blätter haben den gleichen Abstand zur Wurzel.

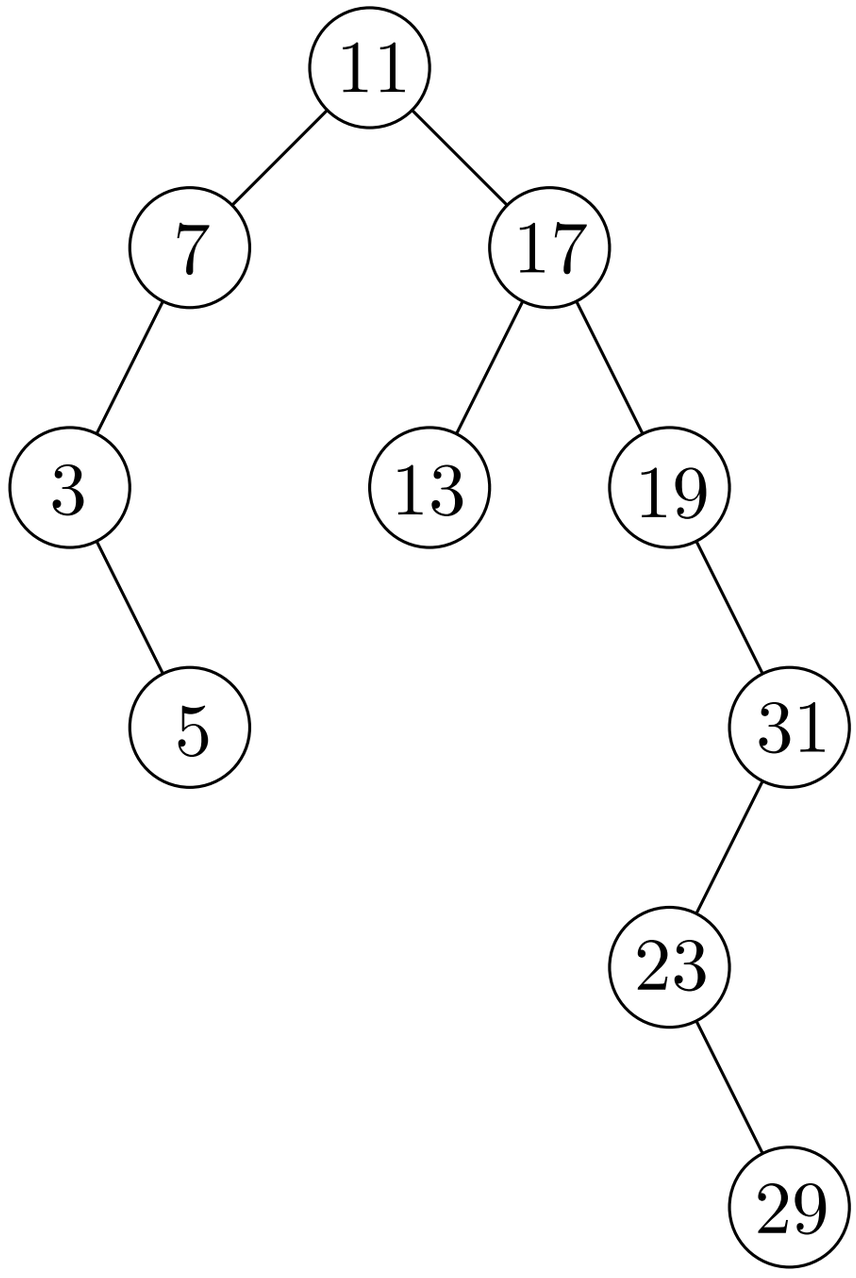


Degenerierter binärer Baum:

- Jeder Knoten hat maximal ein Kind.



Binäre Bäume (Wiederholung)



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

Satz 4.4 → O(h)

Satz 4.5

Satz 4.6

Höhenbalanciert (Wiederholung)

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

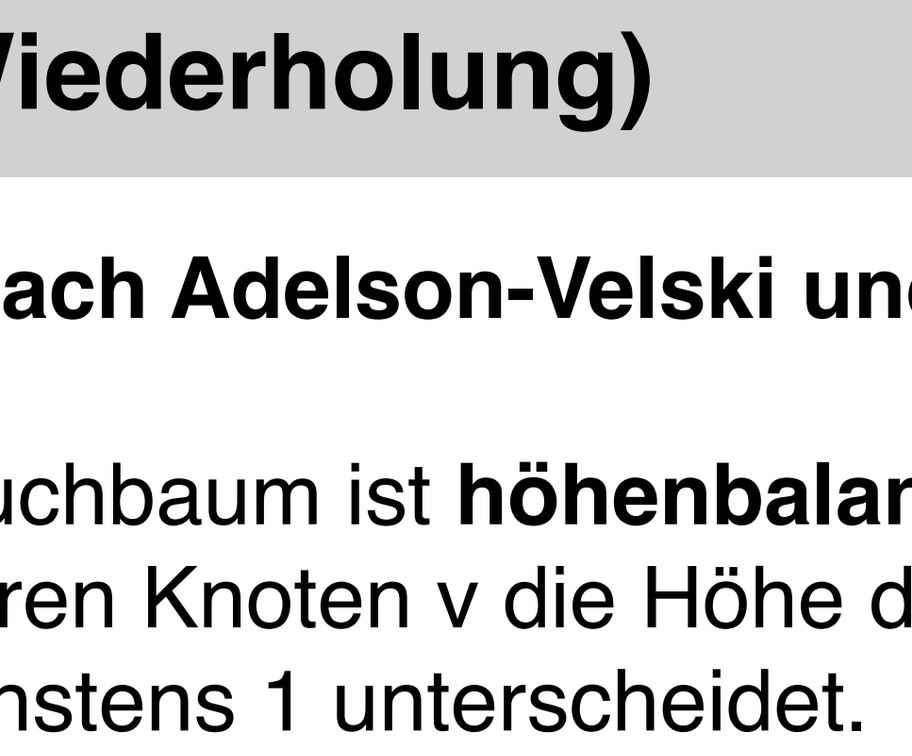
- Wenn die Teilbäume eines Knotens gleich groß sind?
- Wenn die Teilbäume unterschiedlich groß sind?
- Wenn die Teilbäume unterschiedlich groß sind, aber einem gewissen Verhältnis / Regeln genügen?
- Wenn es nur einen Teilbaum in jedem Knoten gibt?

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

AVL-Bäume (Wiederholung)

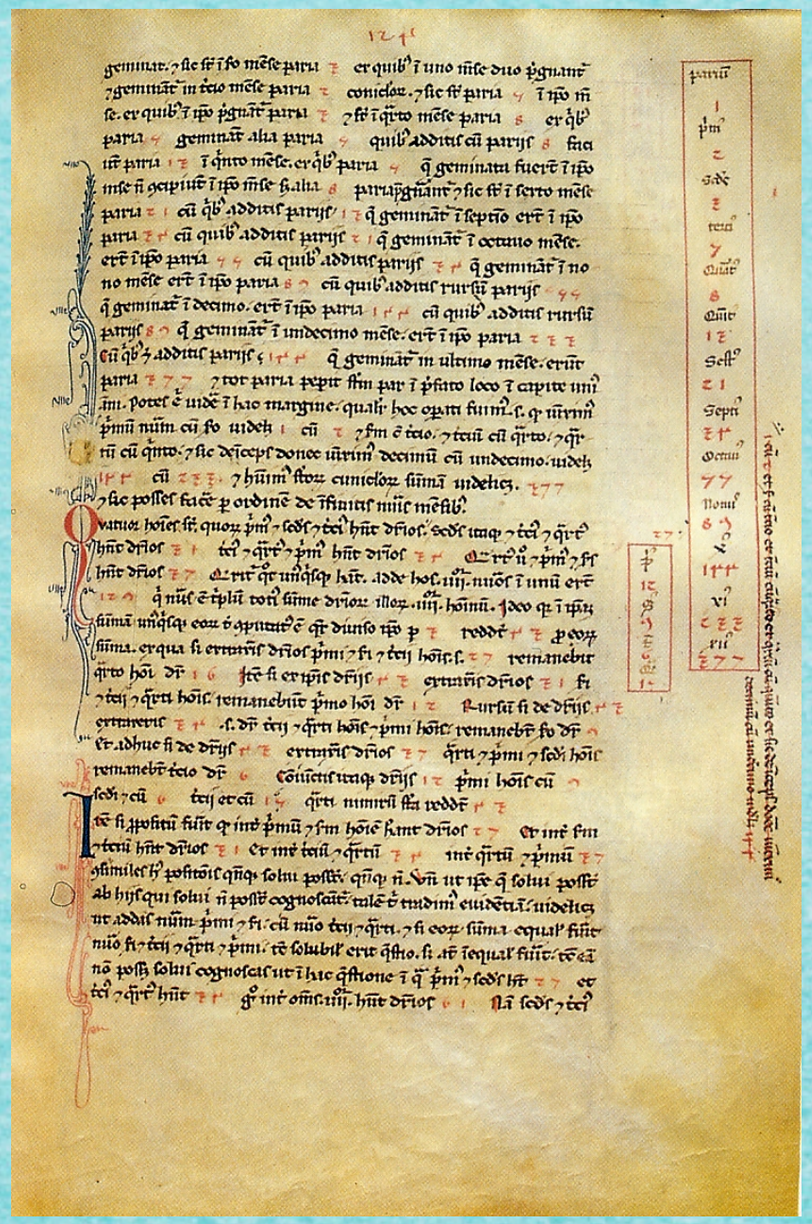
Definition 4.7 (Nach Adelson-Velski und Landis, 1962)

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.

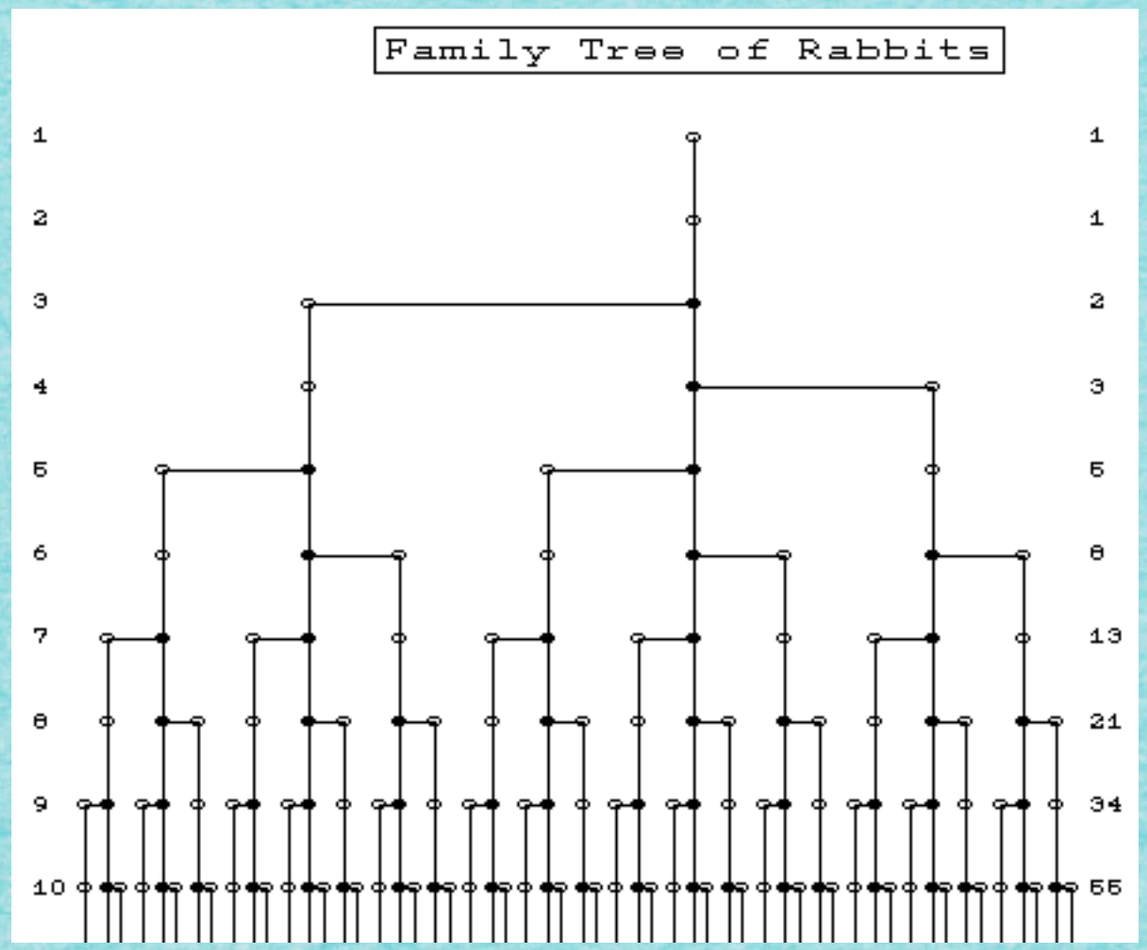


4.7 Fibonacci-Zahlen

Leonardo da Pisa,
gen. Fibonacci
1180-1241

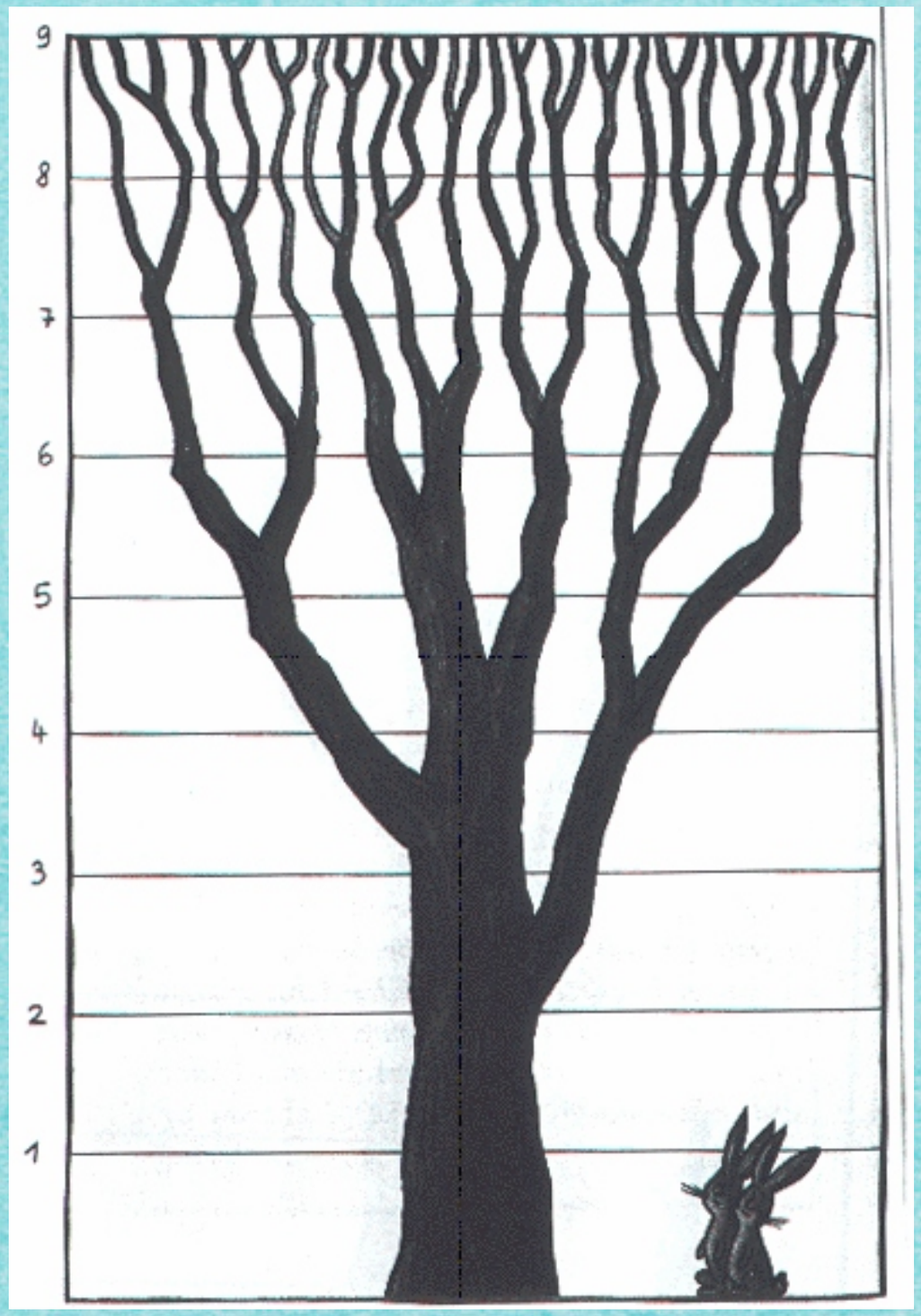


Fibonacci-Zahlen

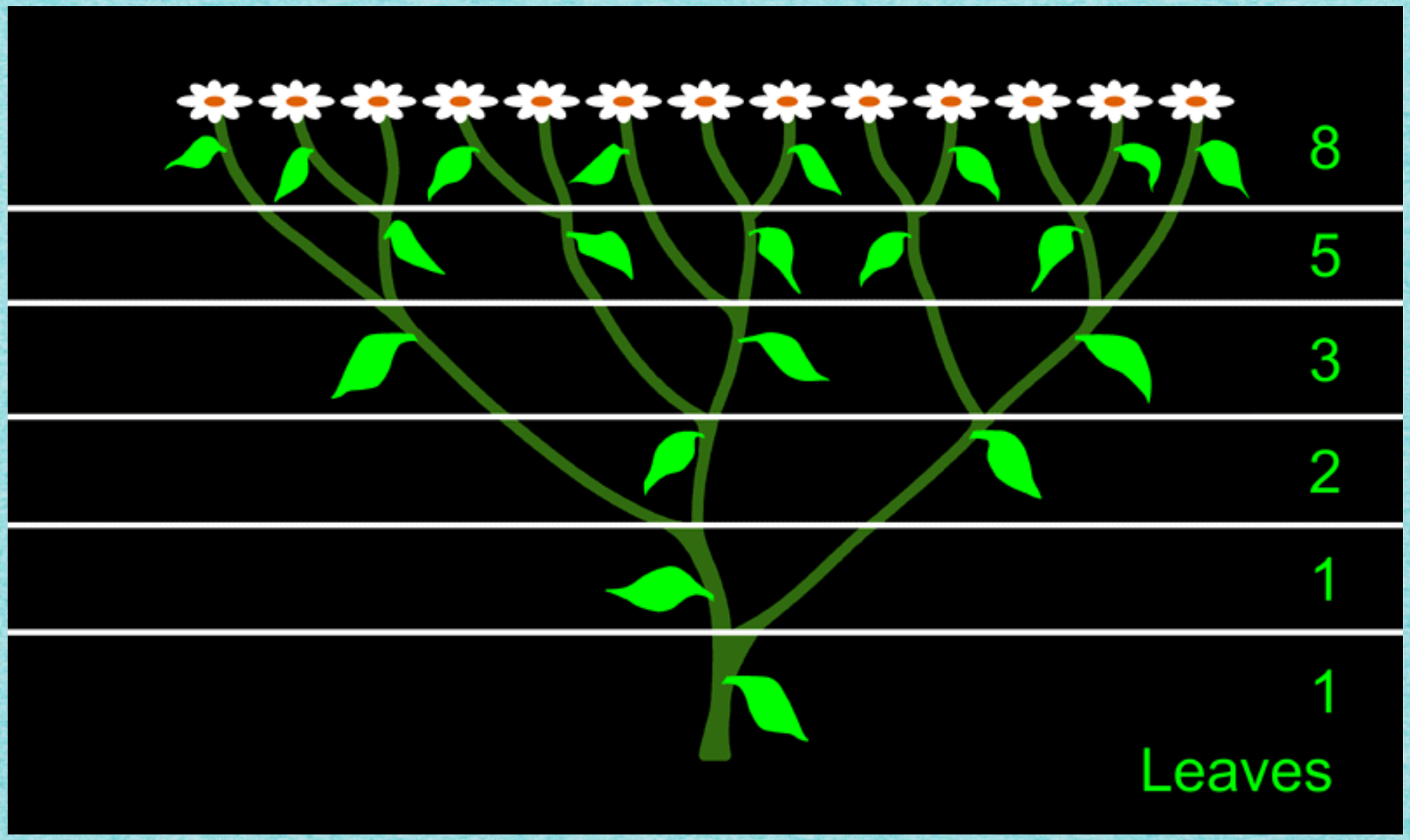


$$F(n)=F(n-1)+F(n-2)$$

Fibonacci-Zahlen



Fibonacci-Zahlen



Fibonacci-Zahlen

1

Fibonacci-Zahlen

2

Fibonacci-Zahlen

3

Fibonacci-Zahlen

5

Fibonacci-Zahlen

8

Fibonacci-Zahlen

1 3

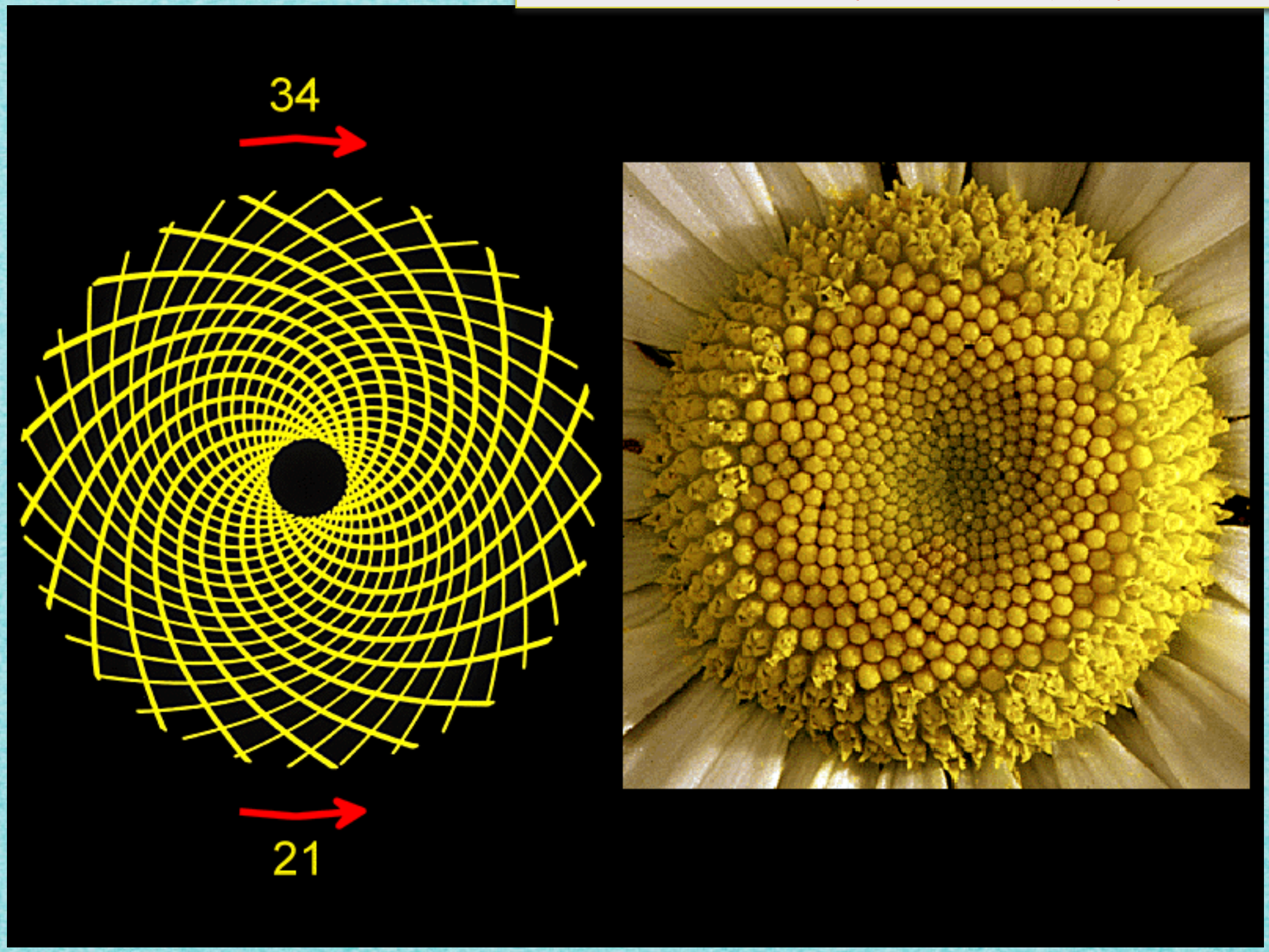
Fibonacci-Zahlen

2.1

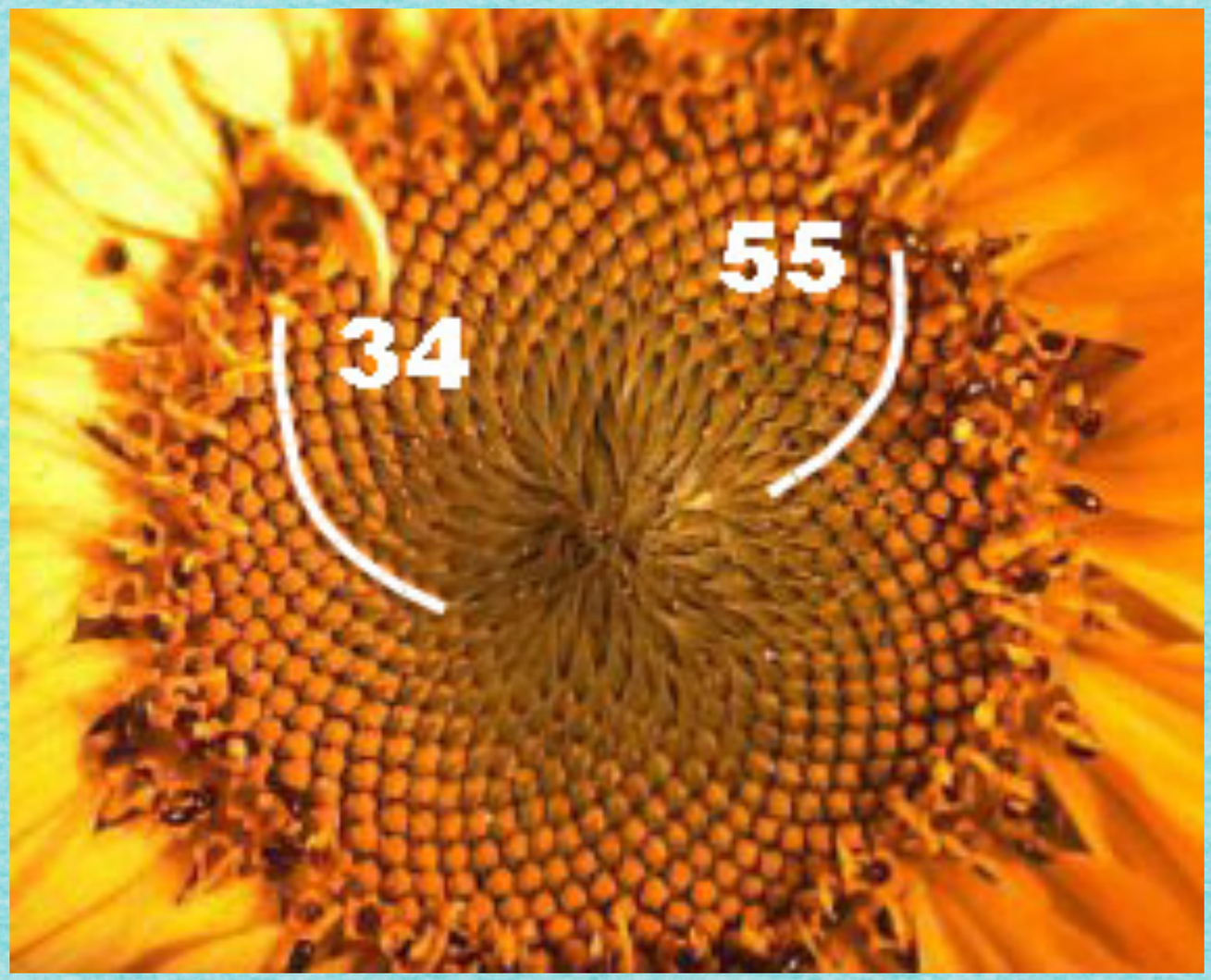
Fibonacci-Zahlen

34

Fibonacci-Zahlen



Fibonacci-Zahlen



Fibonacci-Zahlen

3:2=1.500

5:3=1.666

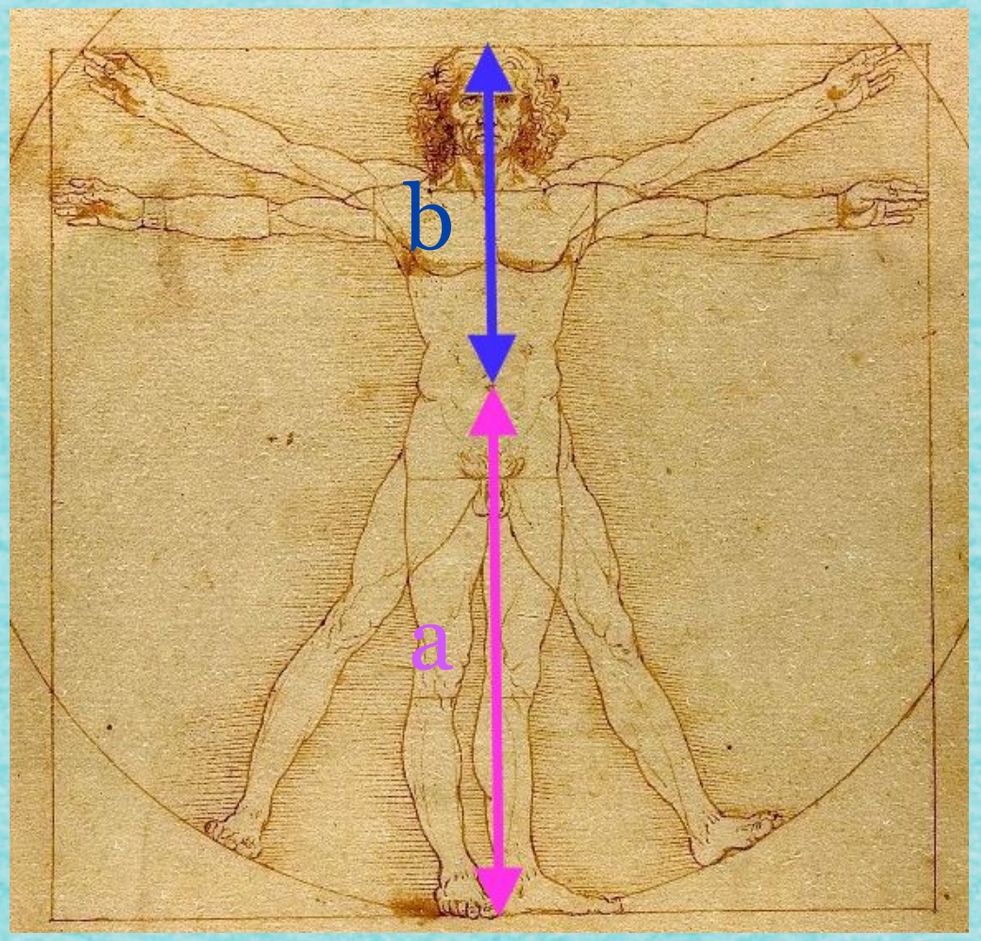
8:5=1.600

13:8=1.625

21:13=1.615

34:21=1.619

Fibonacci-Zahlen

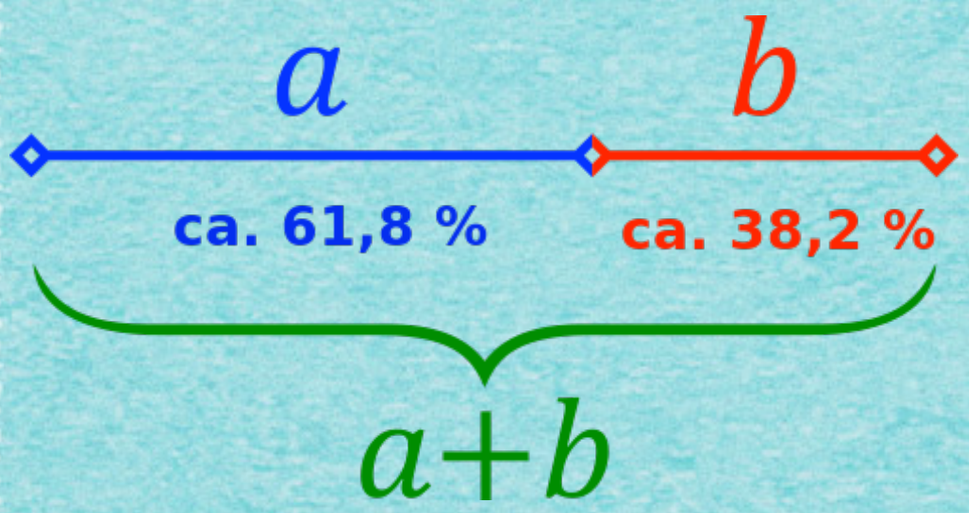


Goldener Schnitt:

$$\frac{a}{a+b} = \frac{b}{a}$$

Fibonacci-Zahlen

$$\frac{a}{a+b} = \frac{b}{a}$$



Herleitung des Zahlenwertes [Bearbeiten]

Aus der oben angegebenen Definition

$$\frac{a}{b} = \frac{a+b}{a} = 1 + \frac{b}{a}$$

bzw.

$$\frac{a}{b} - 1 - \frac{b}{a} = 0$$

folgt mit $\Phi = \frac{a}{b}$ und $\frac{1}{\Phi} = \frac{b}{a}$

$$\Phi - 1 - \frac{1}{\Phi} = 0\,.$$

Multiplikation mit Φ ergibt die quadratische Gleichung

$$\Phi^2 - \Phi - 1 = 0\,.$$

Diese Gleichung hat genau die beiden Lösungen

$$\Phi = \frac{1+\sqrt{5}}{2} \approx 1{,}618\,033$$

und

$$\bar{\Phi} = \frac{1-\sqrt{5}}{2} = 1 - \Phi = -\frac{1}{\Phi} \approx -0{,}618\,033\,.$$

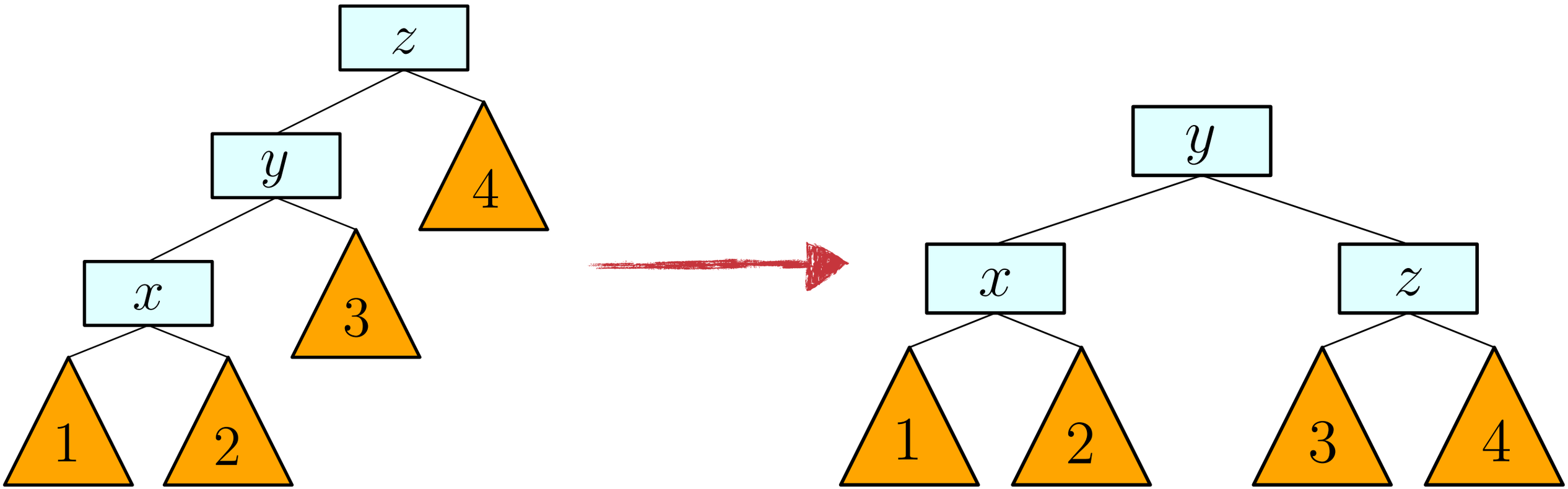
Da $\bar{\Phi}$ negativ ist, ist Φ die gesuchte Goldene Zahl.

Aus diesen Betrachtungen folgt unmittelbar die interessante Beziehung:

$$\frac{1}{\Phi} + 1 = \Phi = \Phi^2 - 1$$

Rotation / Restructure

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



QUIZ!

4.8 Rot-Schwarz-Bäume

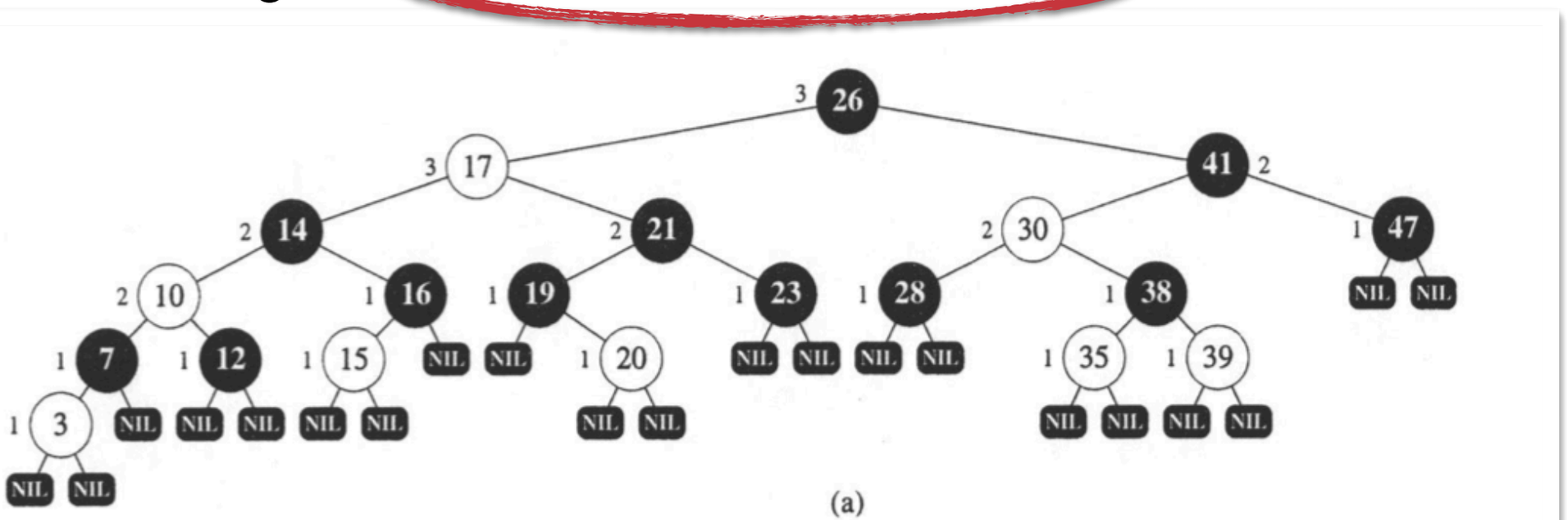
4.8 Rot-Schwarz-Bäume

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.



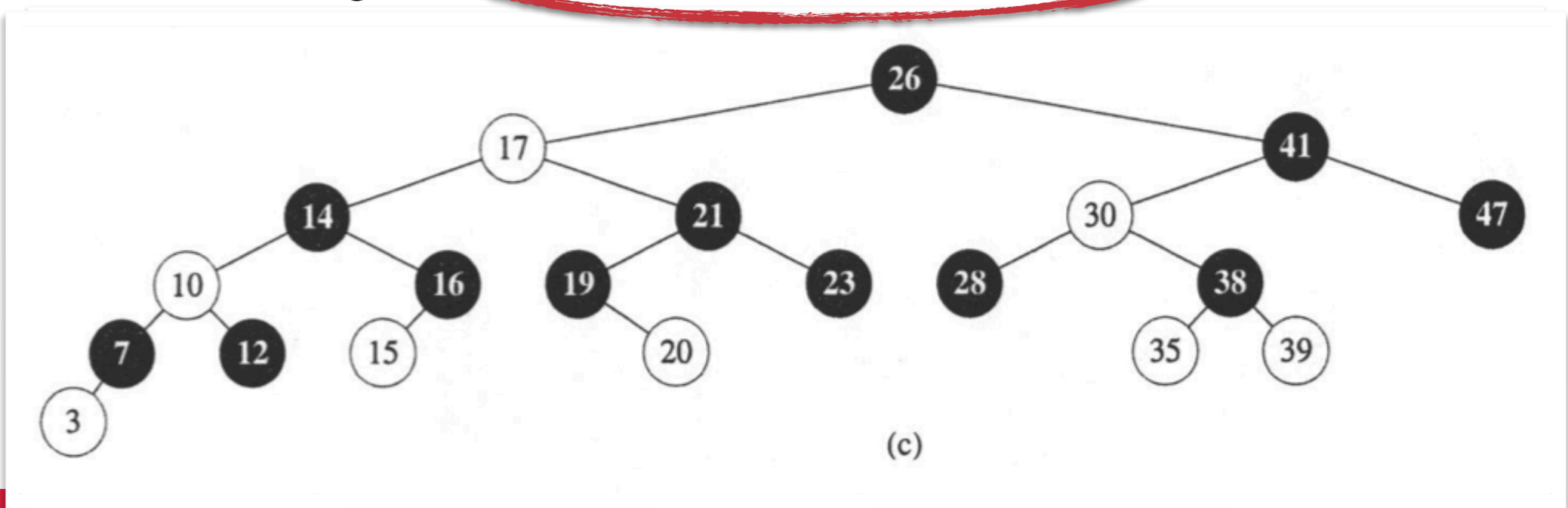
4.8 Rot-Schwarz-Bäume

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.

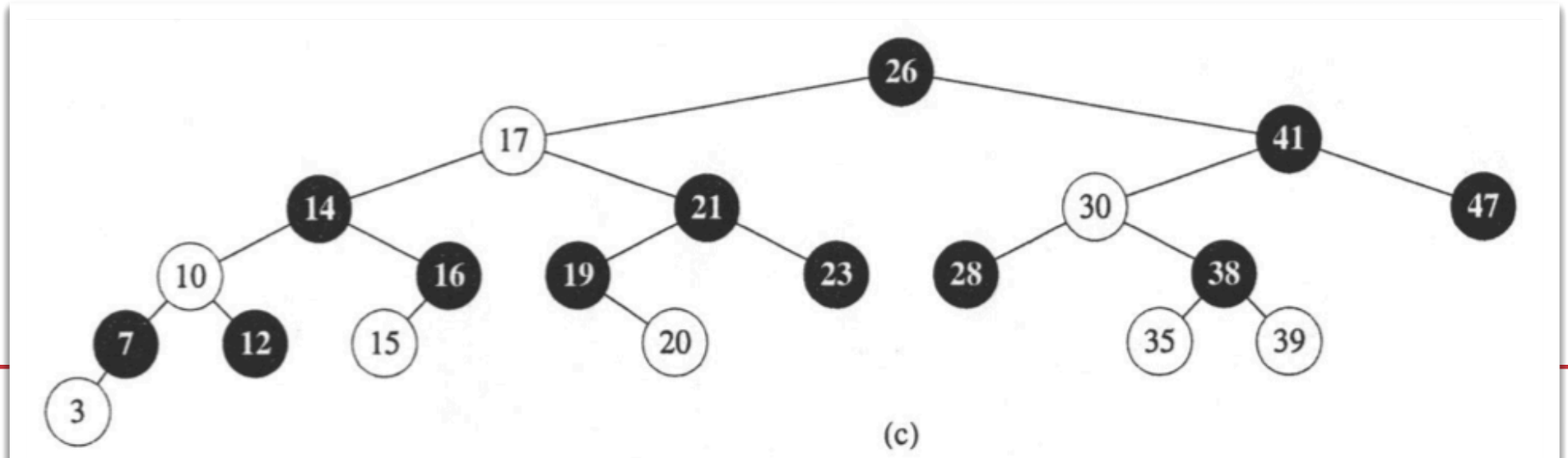


Rot-Schwarz-Bäume

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.

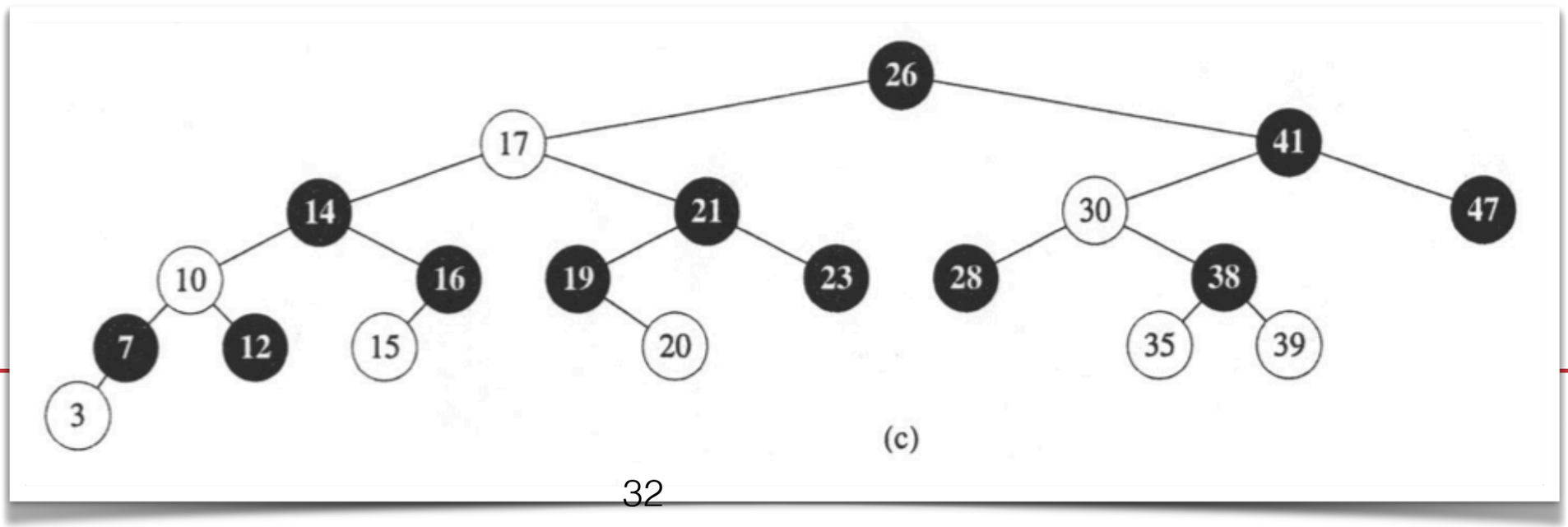
> Wenn ein Knoten rot ist, sind seine beiden Kinder schwarz.



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.

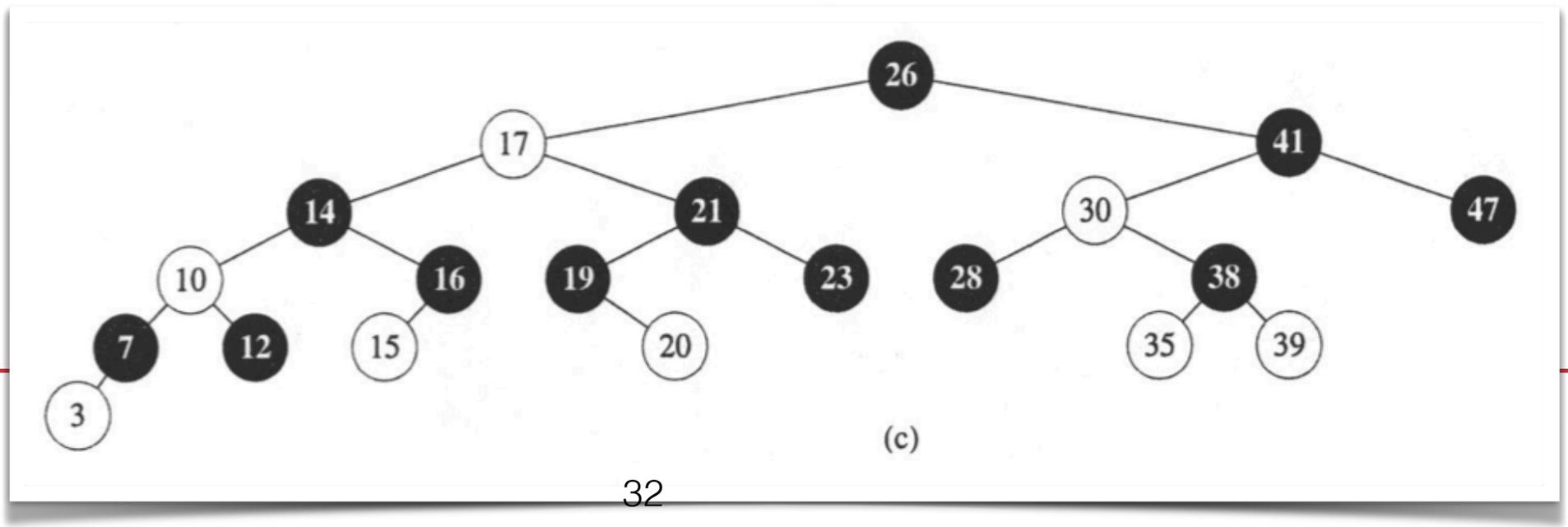


Rot-Schwarz-Bäume

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.

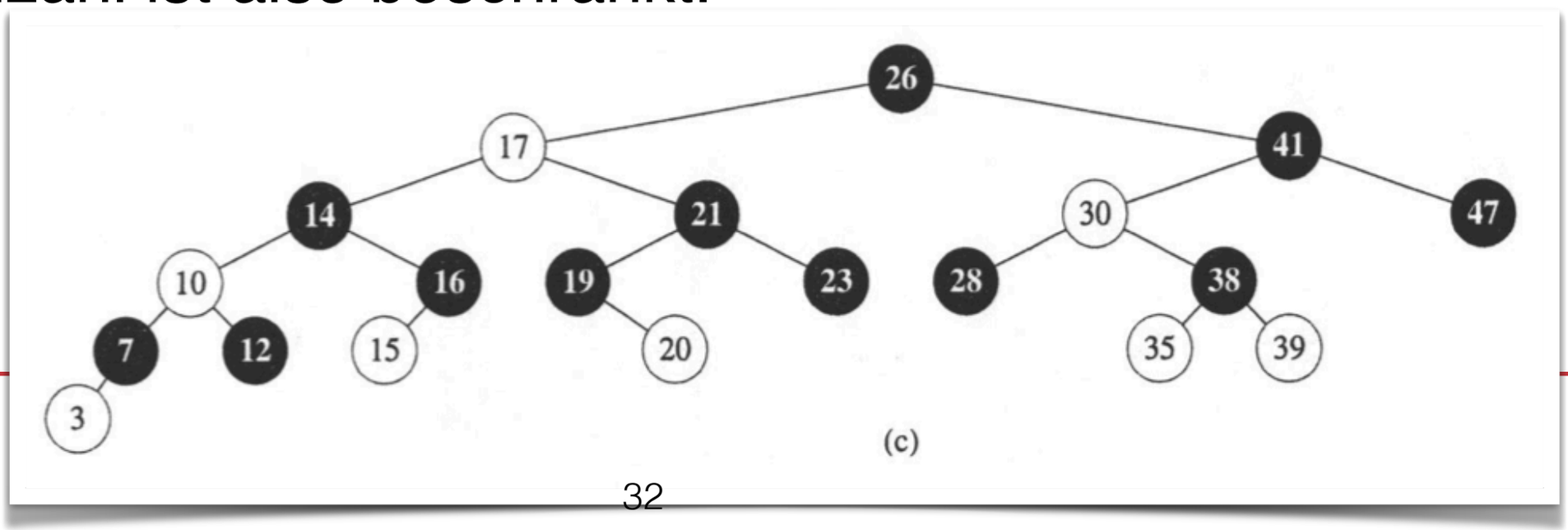
- Auf dem kürzesten Pfad können nur schwarze Knoten vorkommen.
- Wegen 5. können auf einem Pfad bei dem schwarze und rote Knoten vorkommen, maximal doppelt so viele Knoten liegen, wie auf dem mit nur schwarzen Knoten.



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.

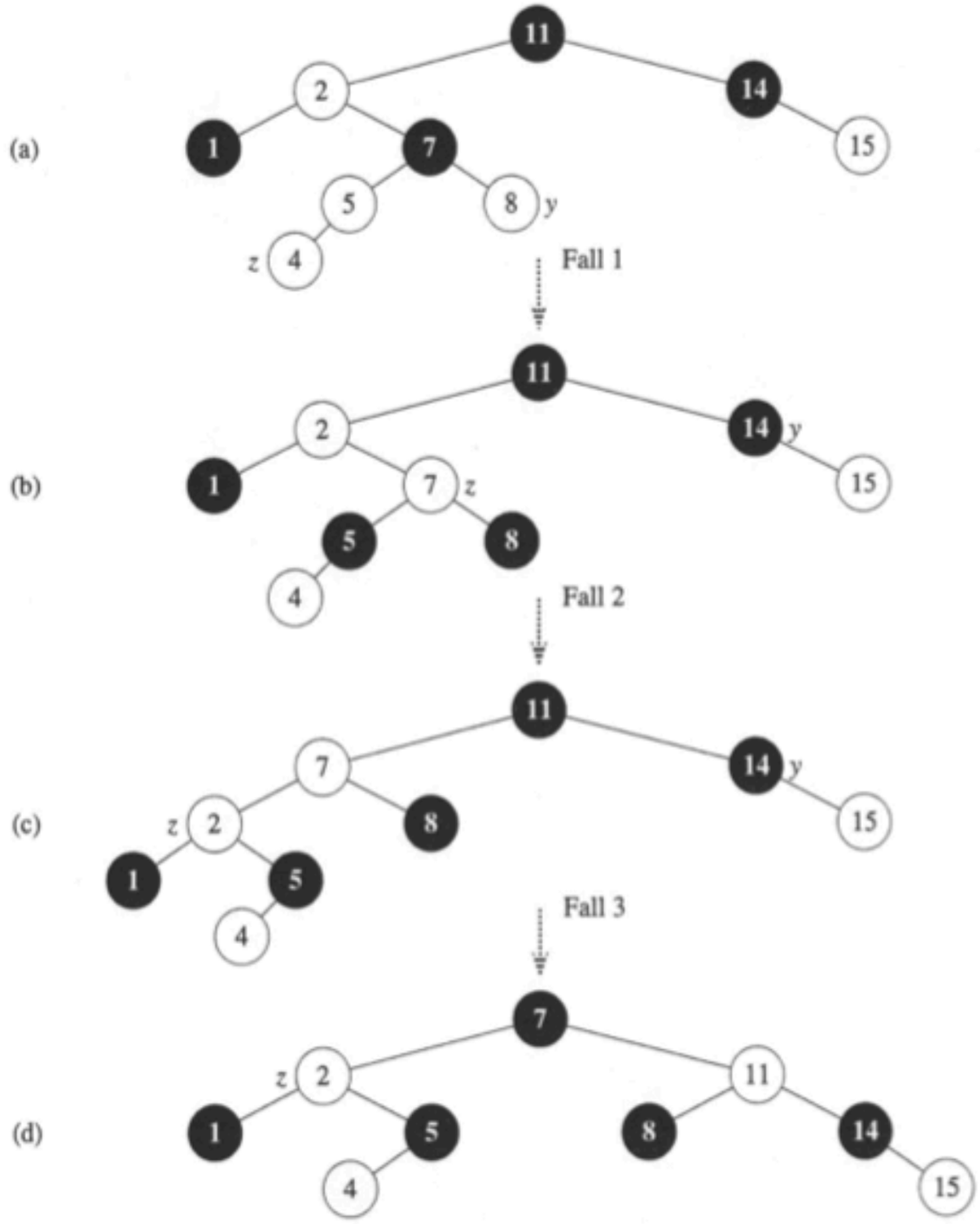


Satz 4.13
Ein Rot-Schwarz-Baum mit n Knoten hat Höhe O(log n).

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



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

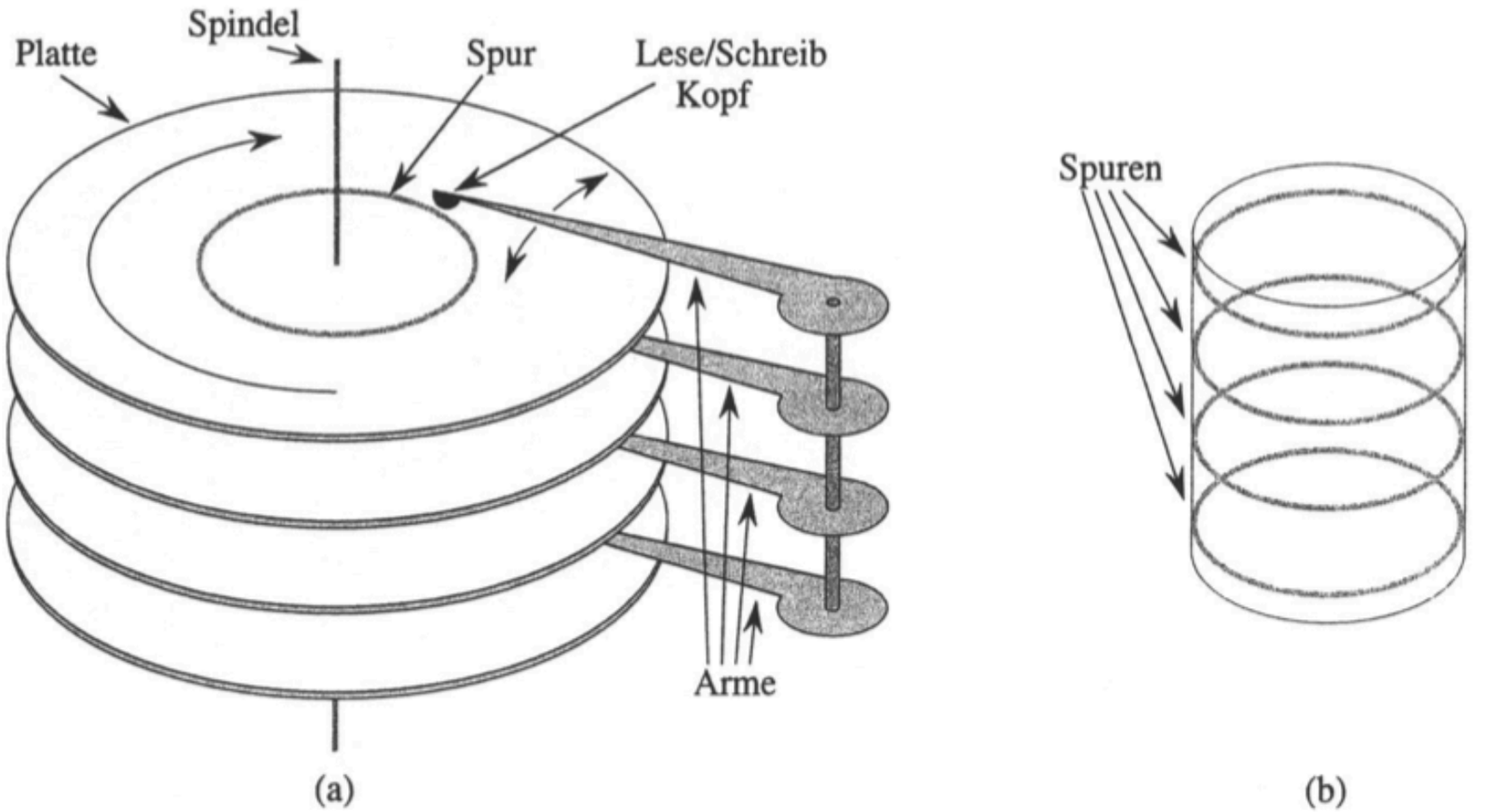
4.9 B-Bäume

(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!

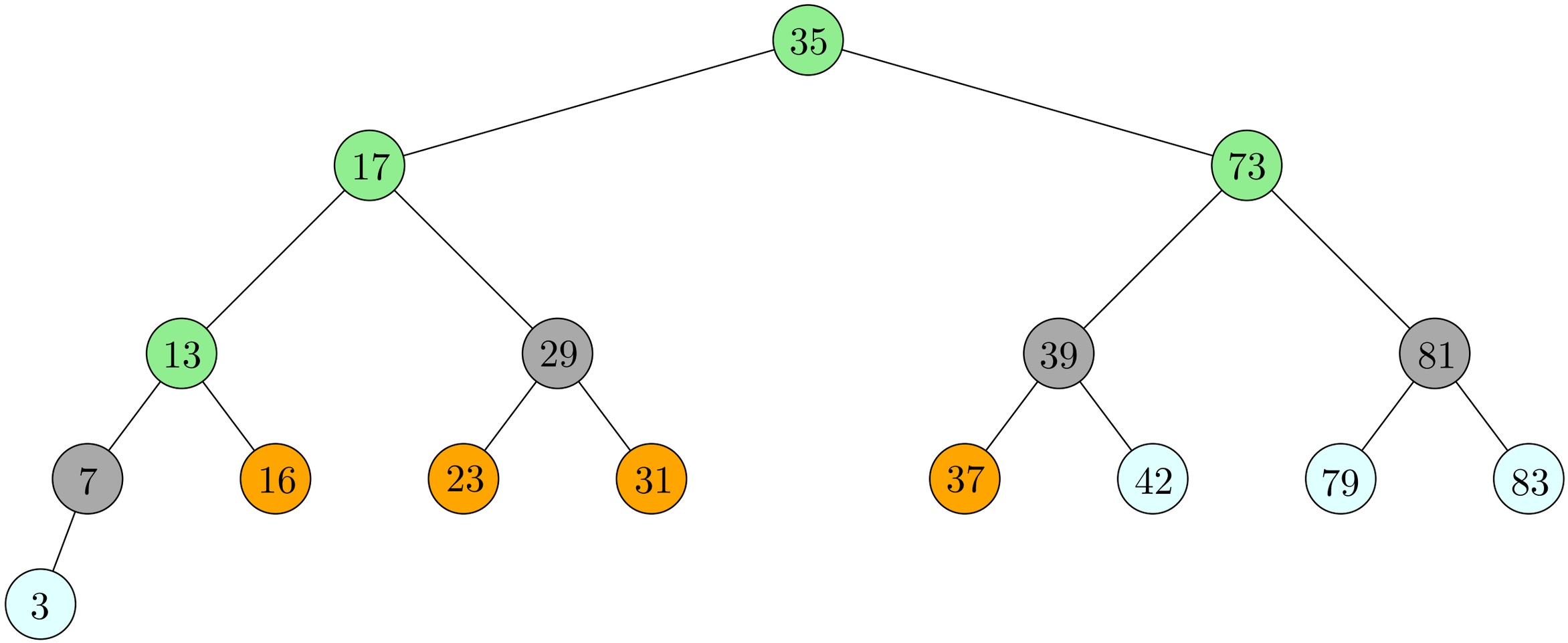
B-Bäume



Kontext: Speicherhierarchien!

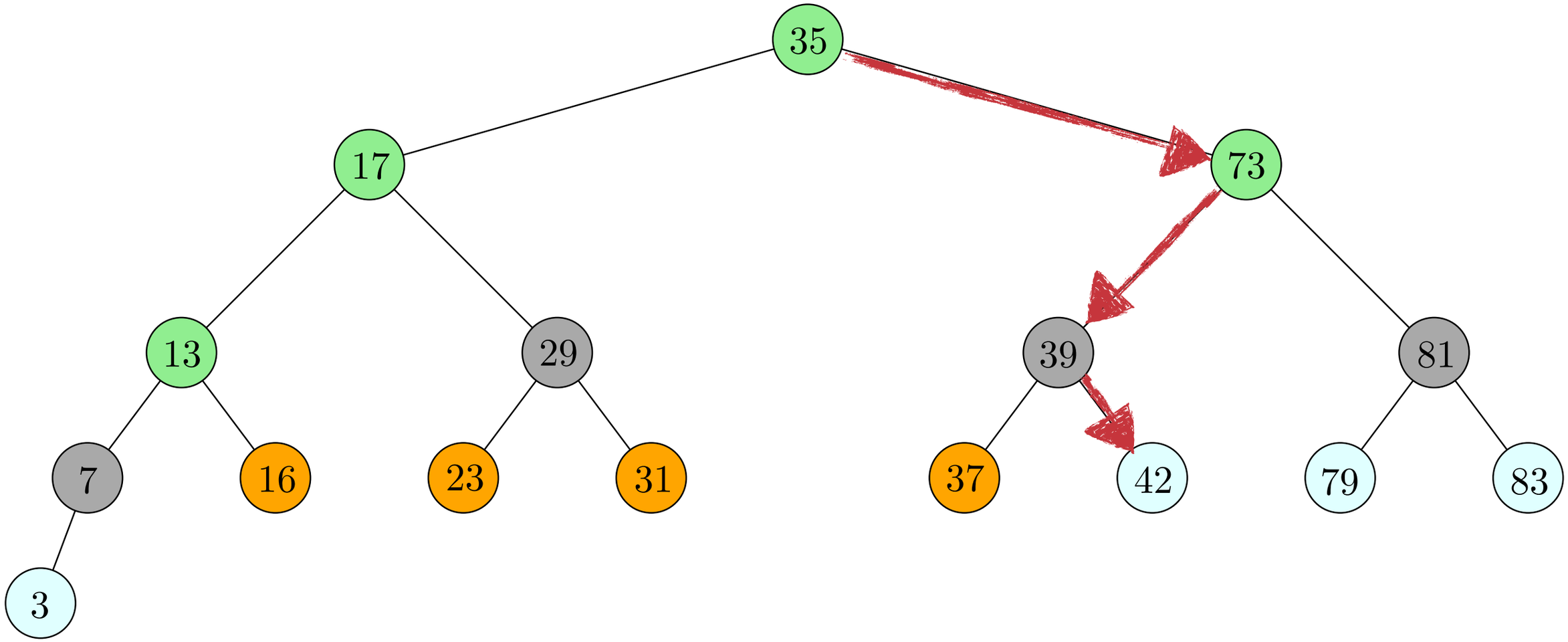
B-Bäume

Angenommen, wir suchen die 42.



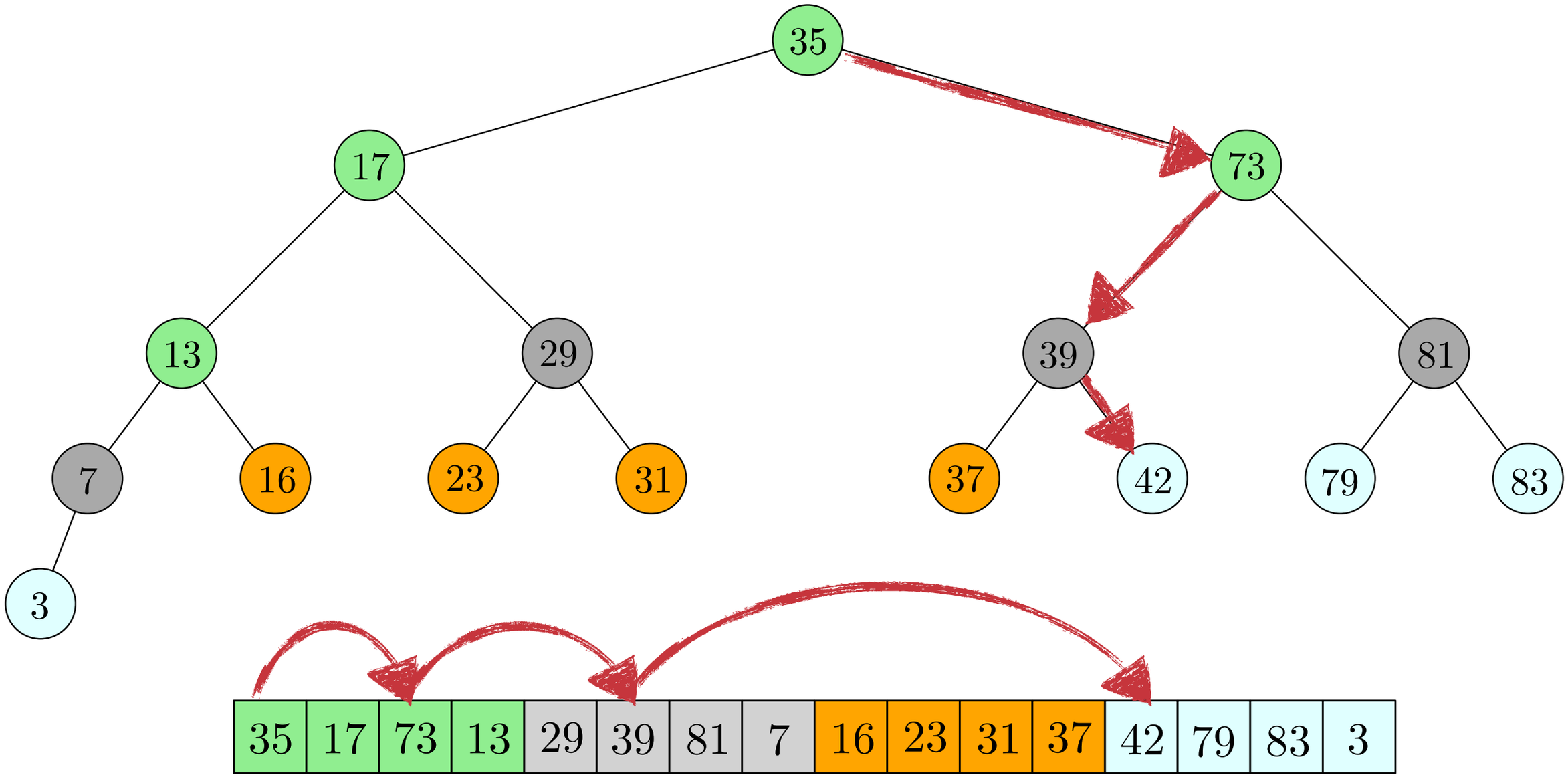
B-Bäume

Angenommen, wir suchen die 42.



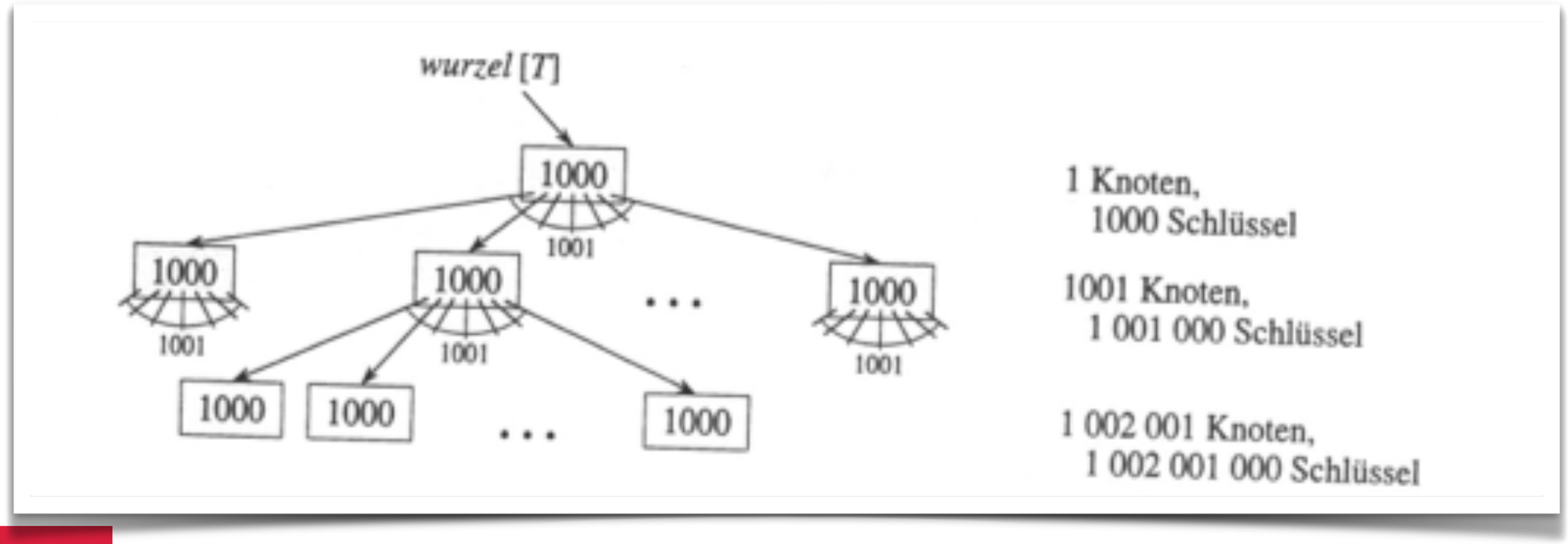
B-Bäume

Angenommen, wir suchen die 42.

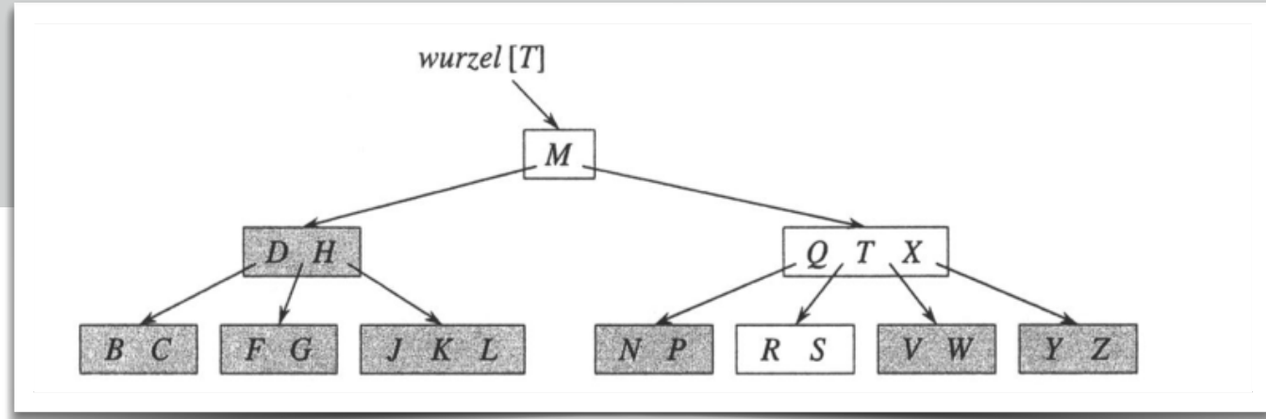


B-Bäume

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



B-Bäume

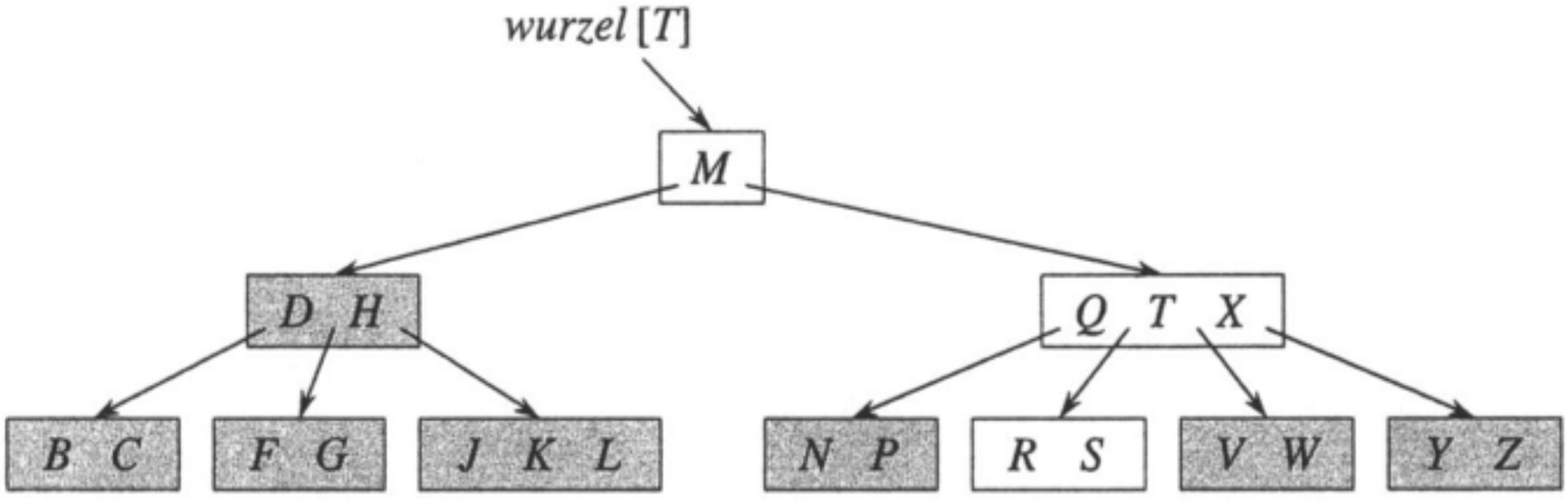


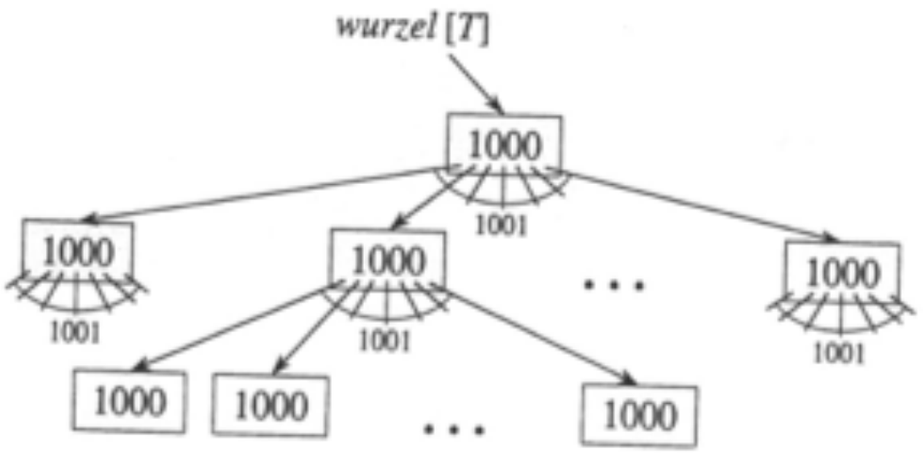
Definition 4.13 (B-Baum)

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





1 Knoten,
1000 Schlüssel

1001 Knoten,
1 001 000 Schlüssel

1 002 001 Knoten,
1 002 001 000 Schlüssel

B-Bäume

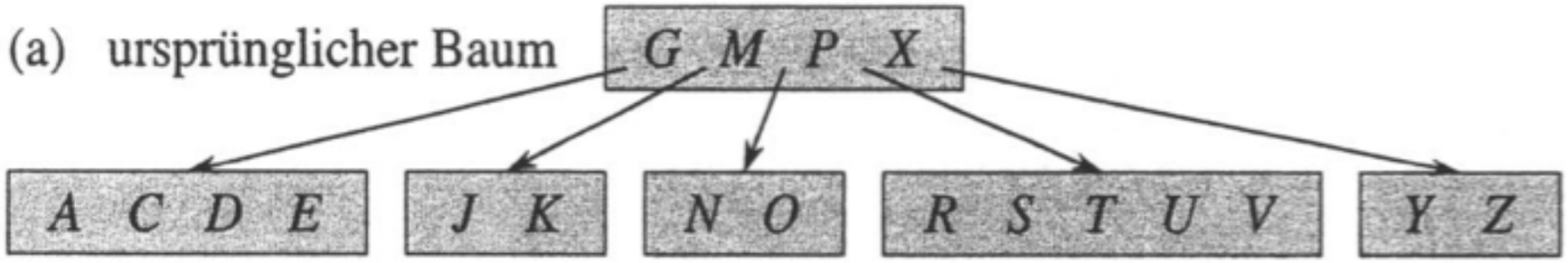
B-Baum der Höhe h=3 und Fan-Out-Factor q = 2048

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

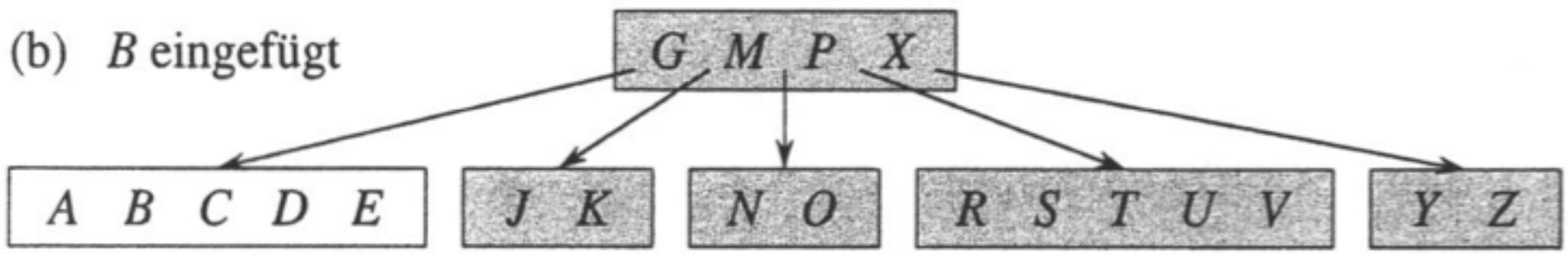
Binärer Baum der Höhe h=3 und Fan-Out-Factor q = 2

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

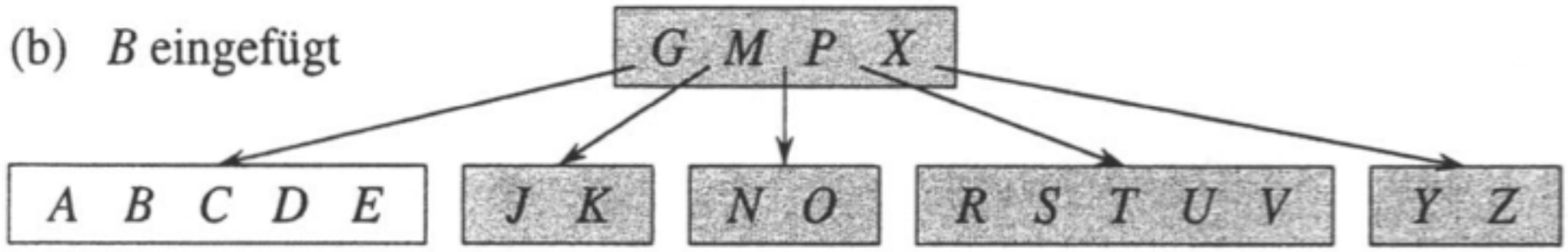
B-Bäume — Einfügen



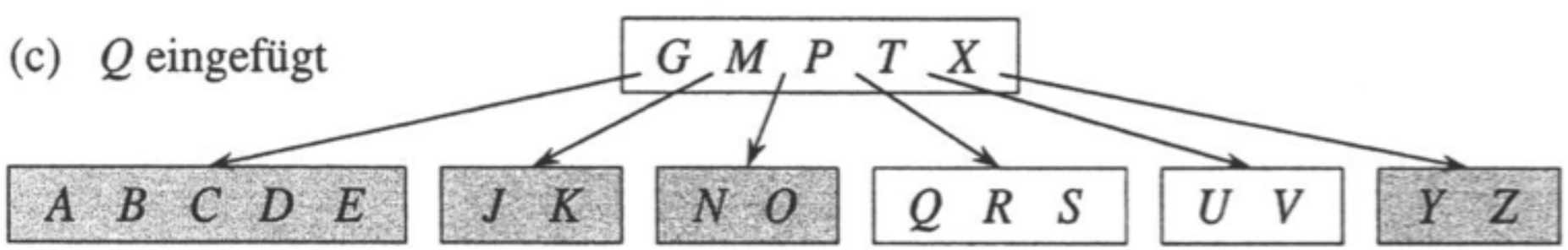
Füge B ein!



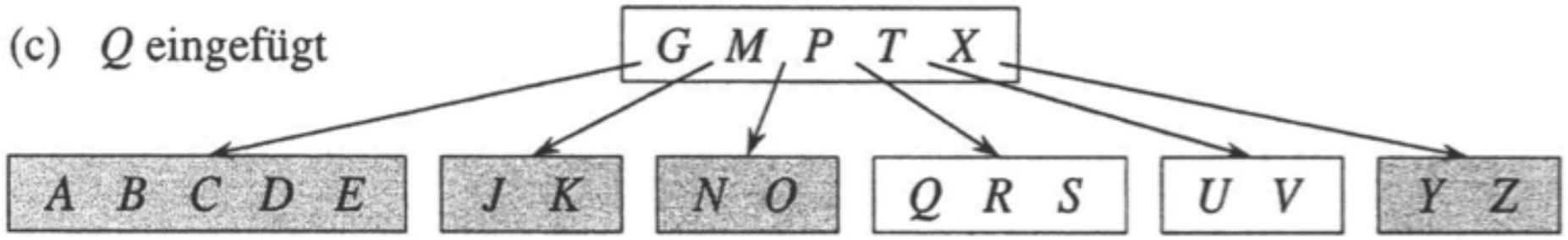
B-Bäume — Einfügen



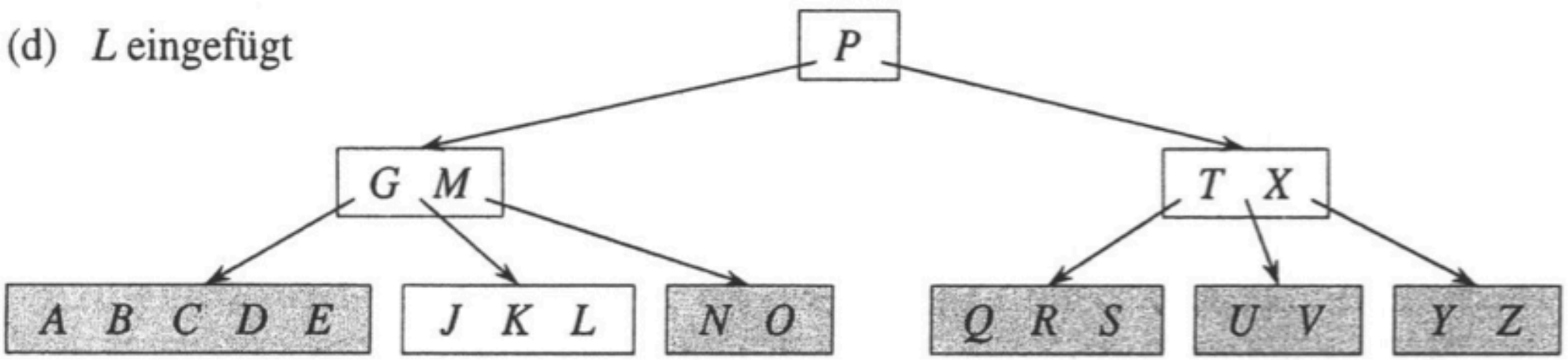
Füge Q ein!



B-Bäume — Einfügen

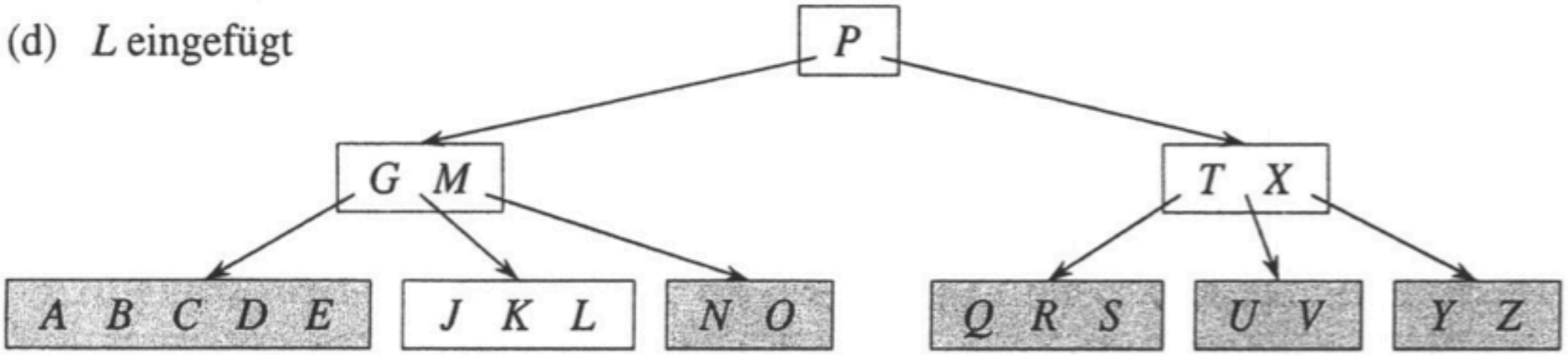


Füge L ein!



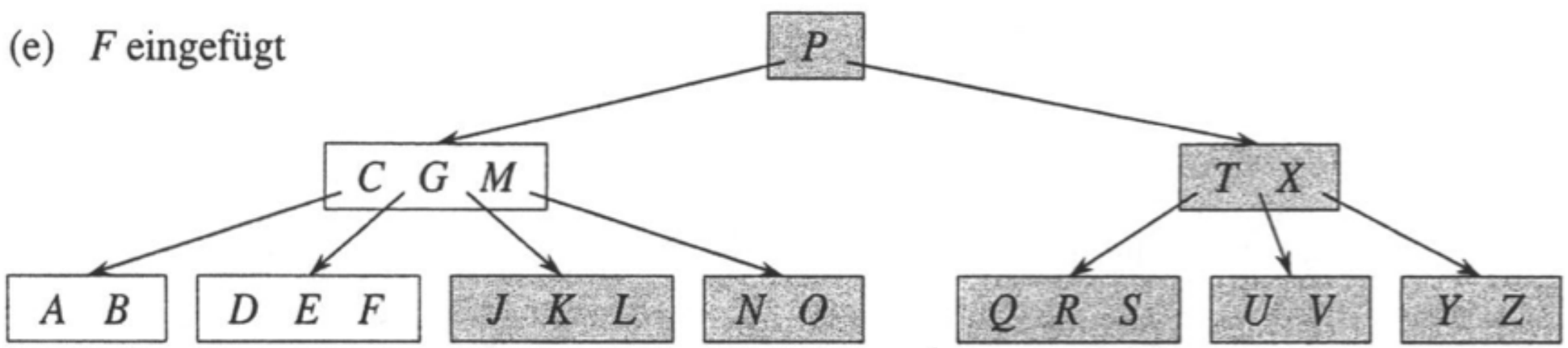
B-Bäume — Einfügen

(d) *L* eingefügt



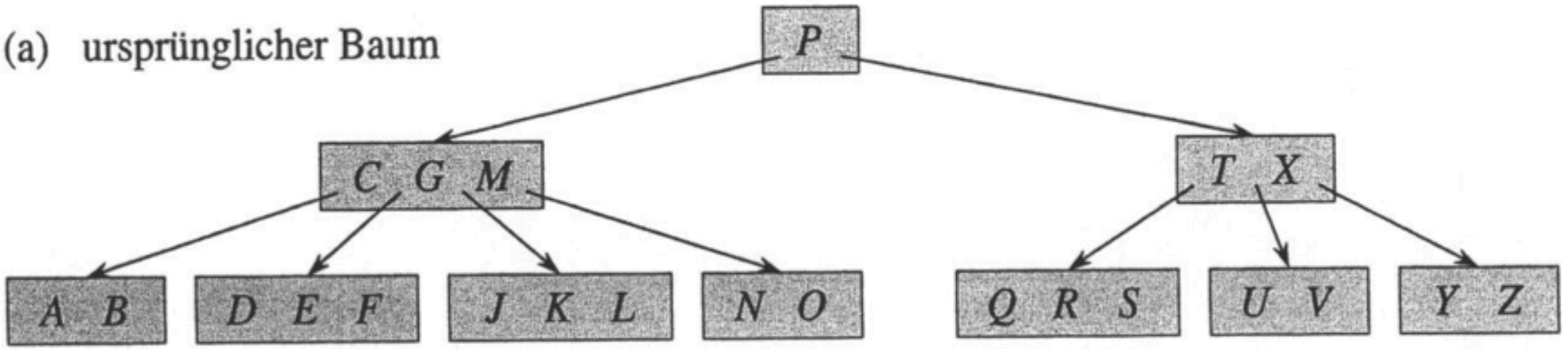
Füge F ein!

(e) *F* eingefügt



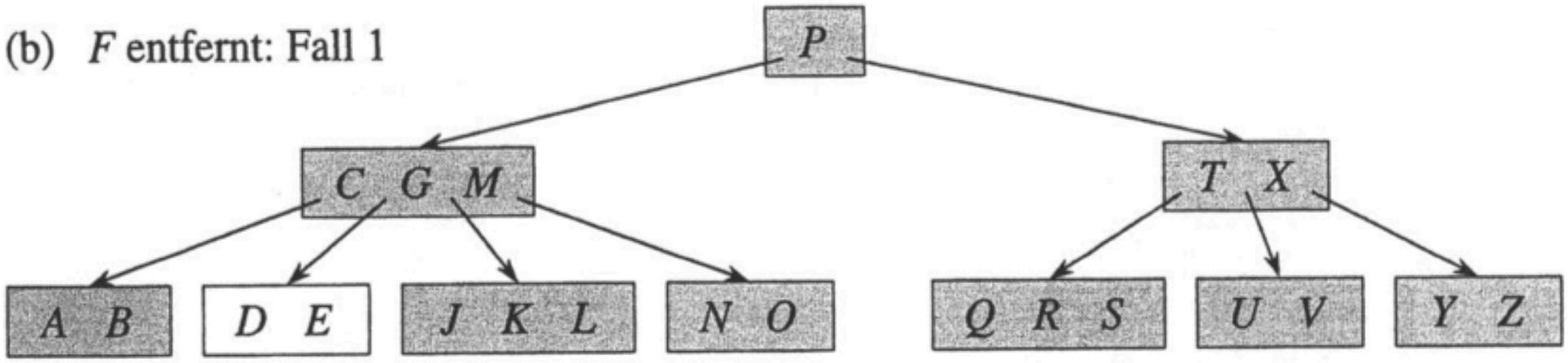
B-Bäume — Löschen

(a) ursprünglicher Baum



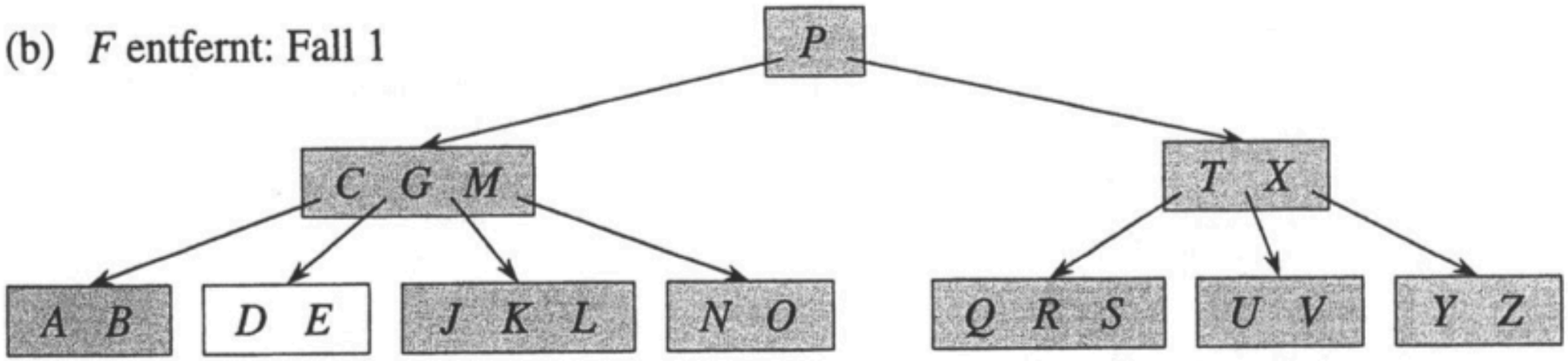
Lösche F!

(b) *F* entfernt: Fall 1



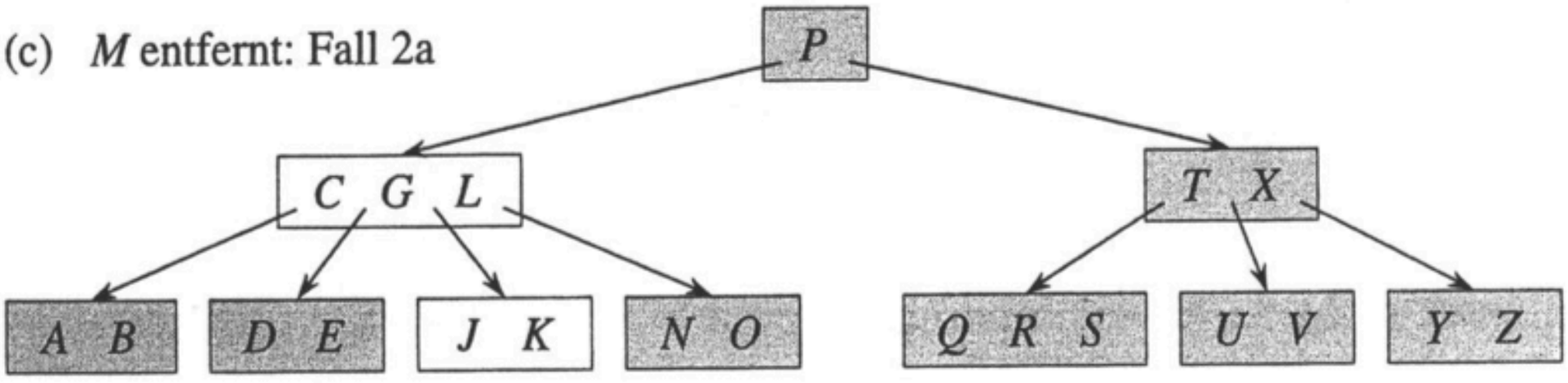
B-Bäume — Löschen

(b) *F* entfernt: Fall 1



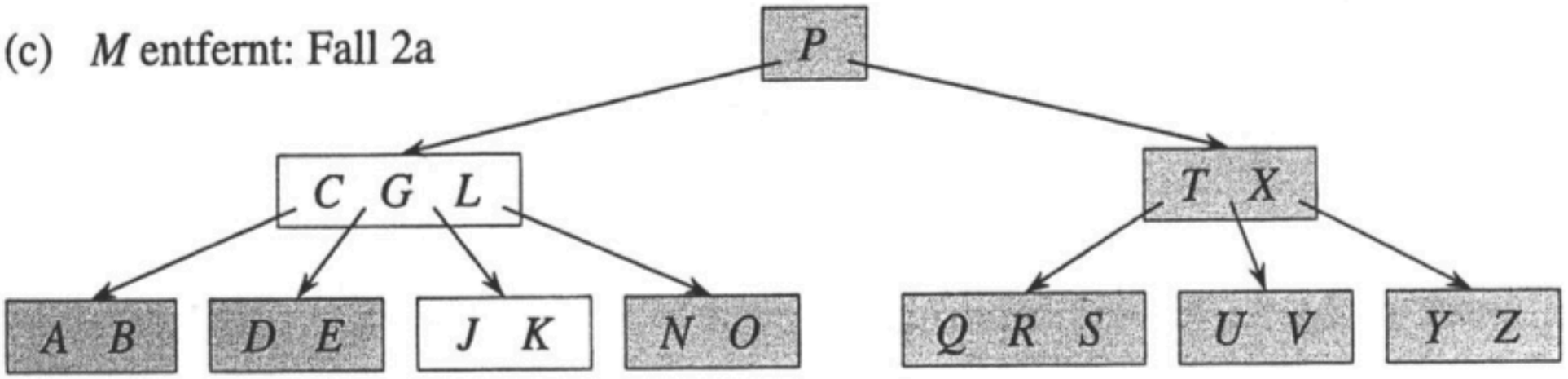
Lösche M!

(c) *M* entfernt: Fall 2a



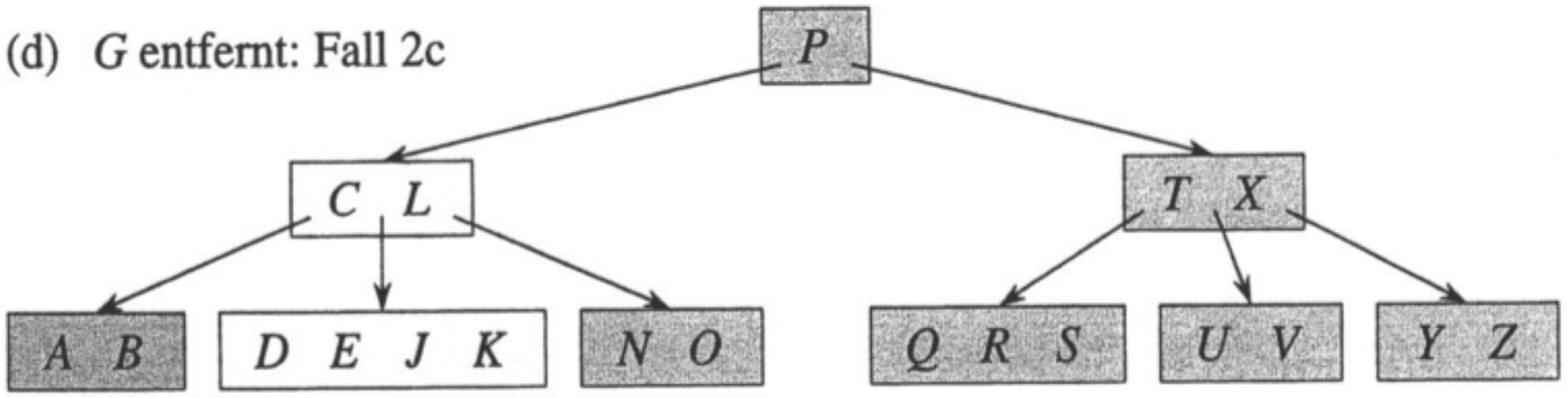
B-Bäume — Löschen

(c) *M* entfernt: Fall 2a

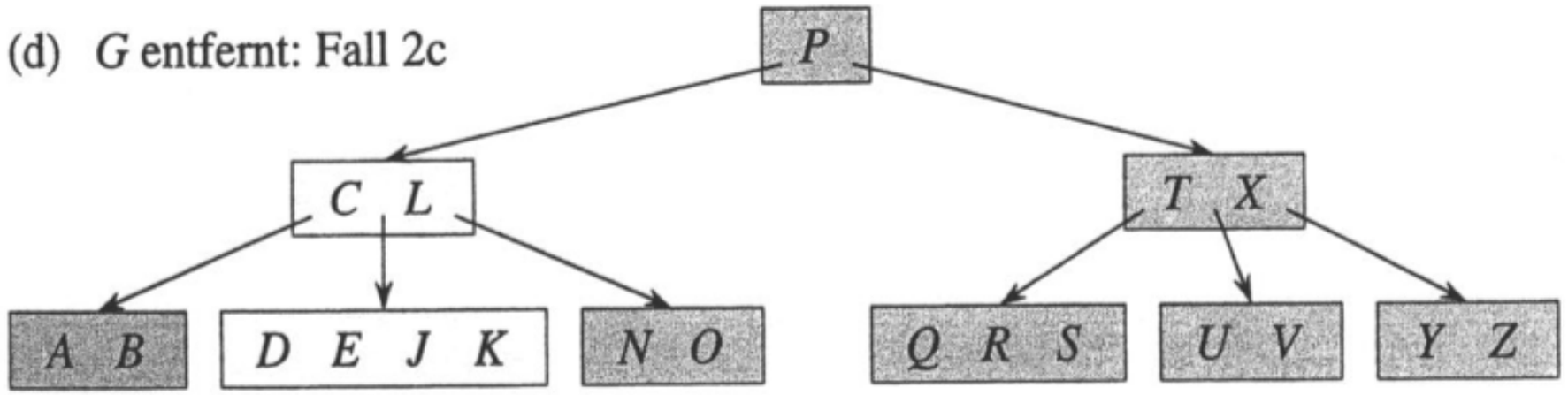


Lösche G!

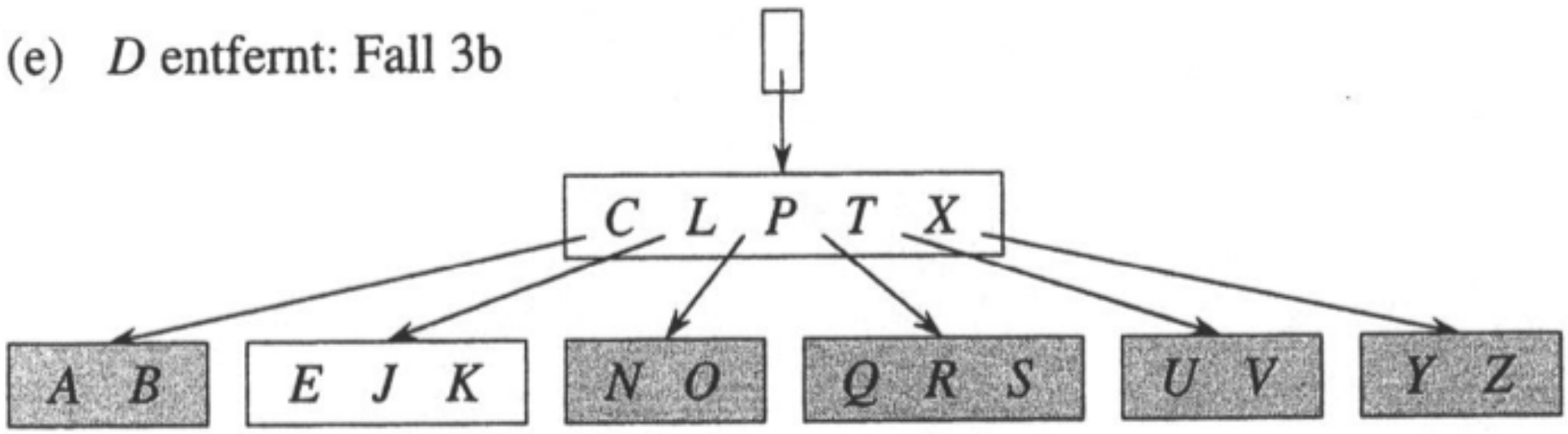
(d) *G* entfernt: Fall 2c



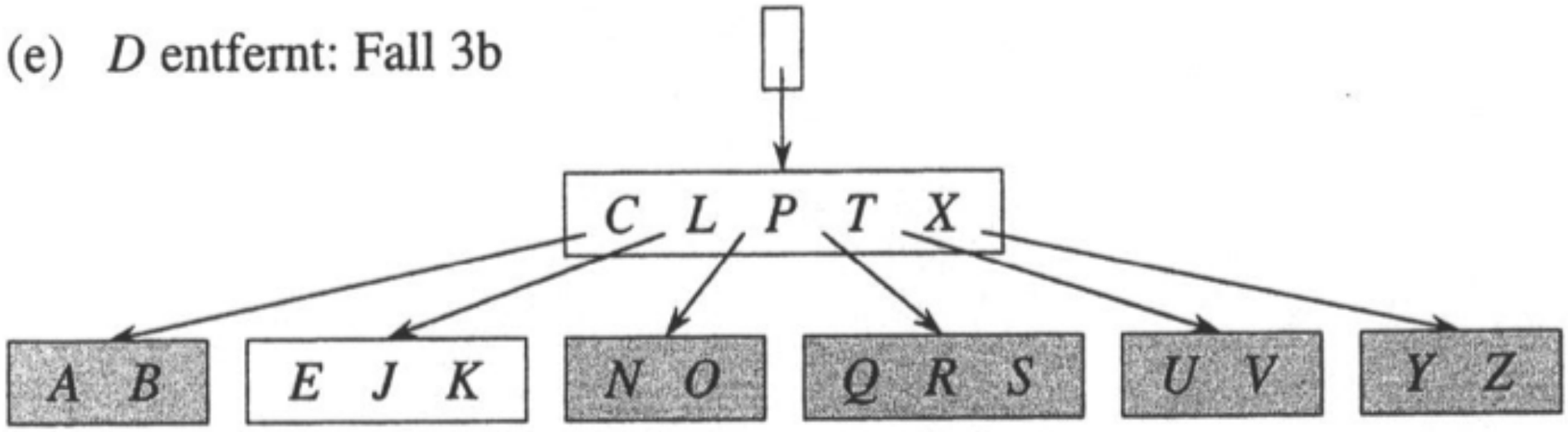
B-Bäume — Löschen

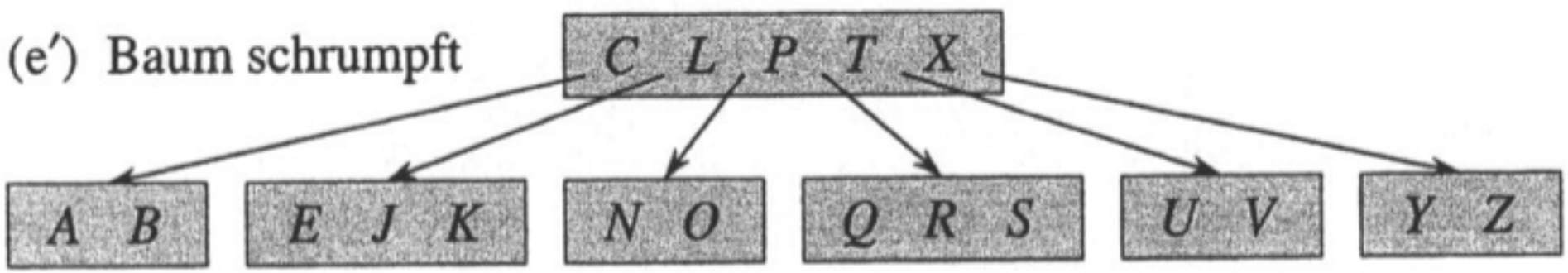


Lösche D!

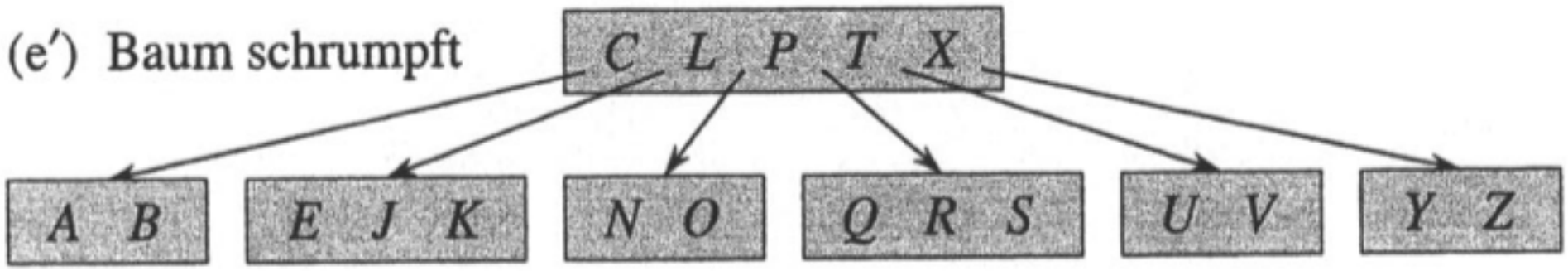


B-Bäume — Löschen

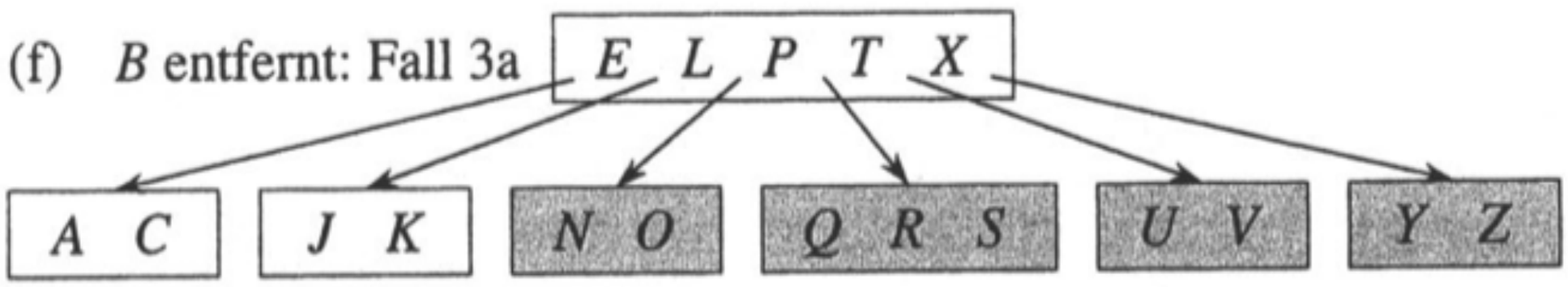




B-Bäume — Löschen



Lösche B!



B-Bäume

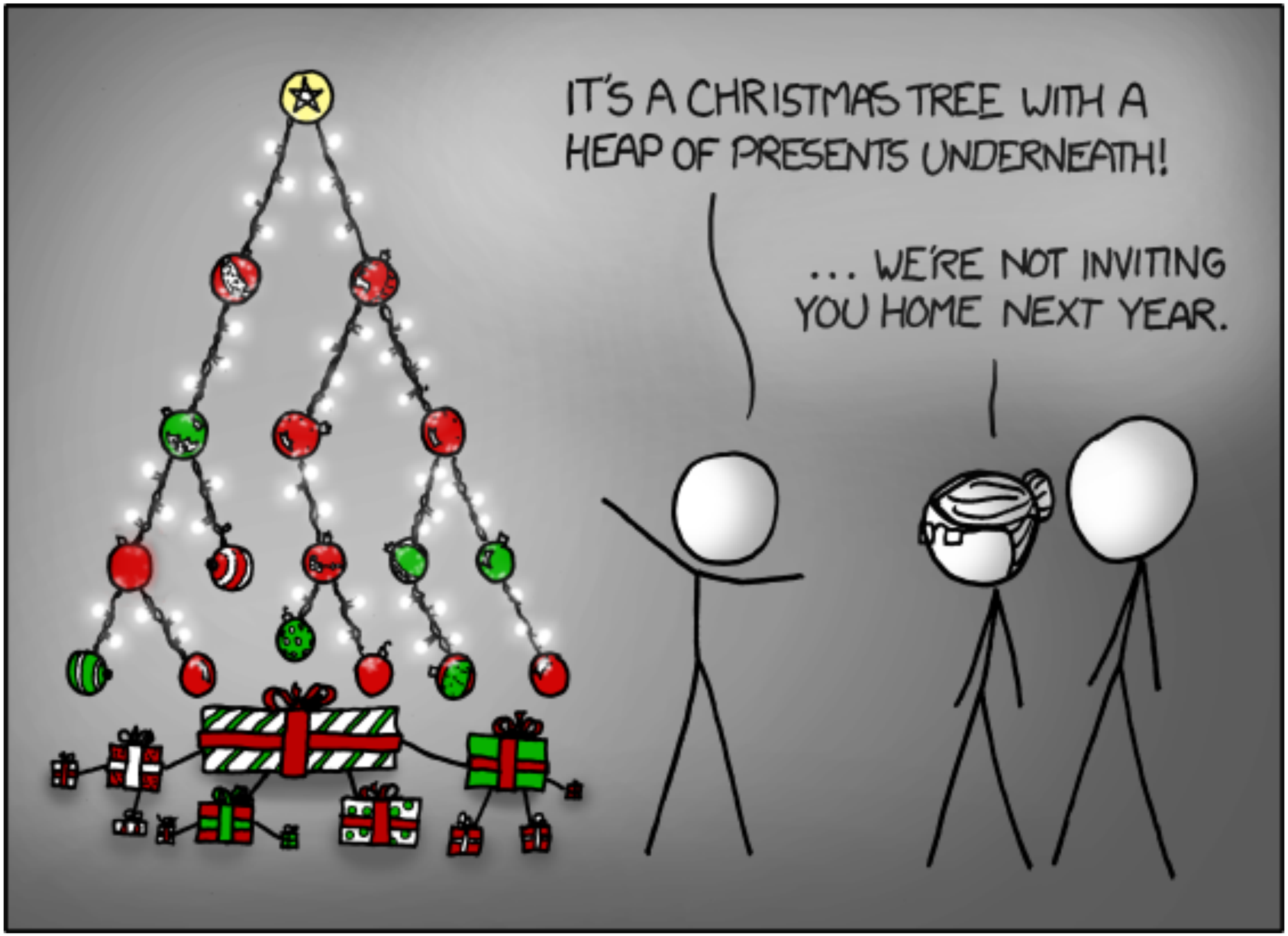
Satz 4.14
Ein B-Baum der Höhe h benötigt zur dynamischen Datenverwaltung im schlimmsten Fall O(h) Plattenoperationen und $O(t \log_t n)$ CPU-Zeit.

mehr dazu: Cormen, Kapitel 18.

Relationale Datenbanksysteme 2

4.10 Heaps

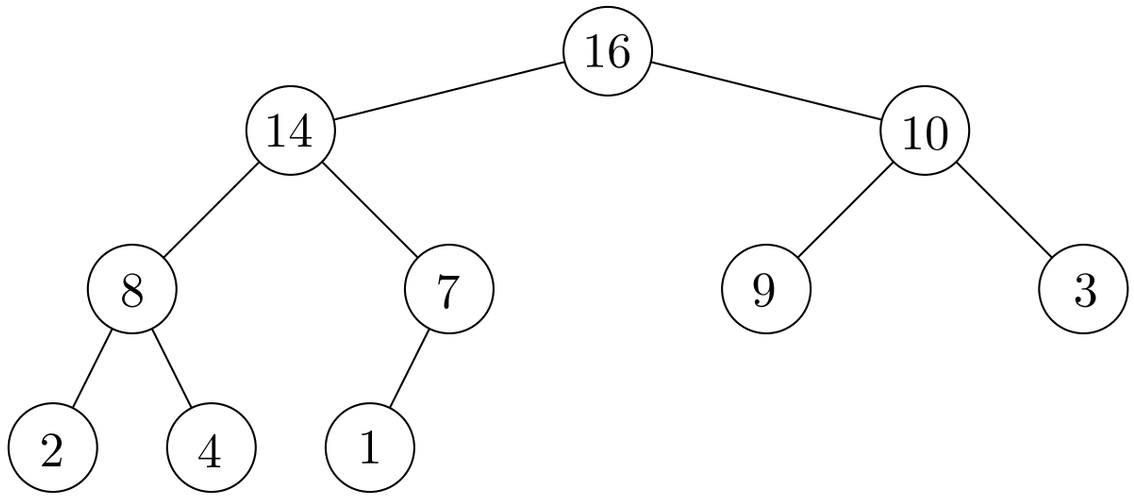
4.10 Heaps


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

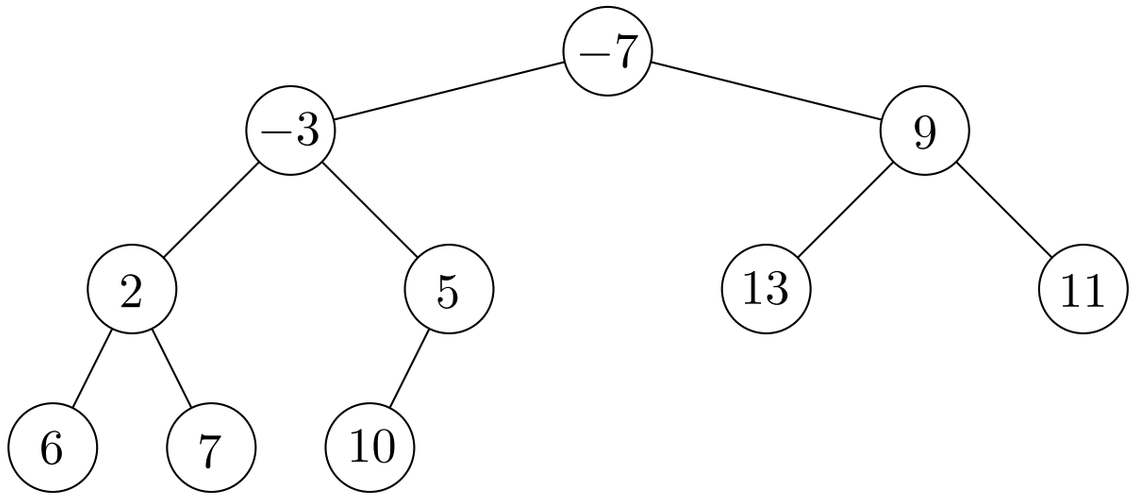
4.10 Heaps

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

Max-Heap

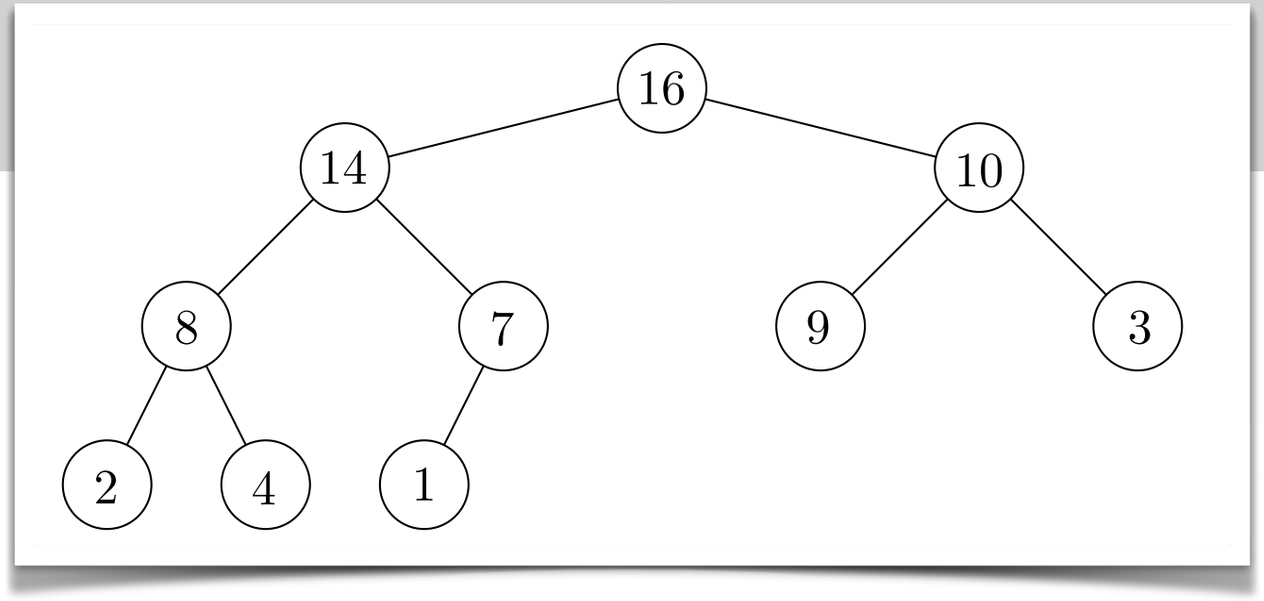


Min-Heap



Eine Beziehung zwischen den Teilbäumen existiert nicht!

Heaps



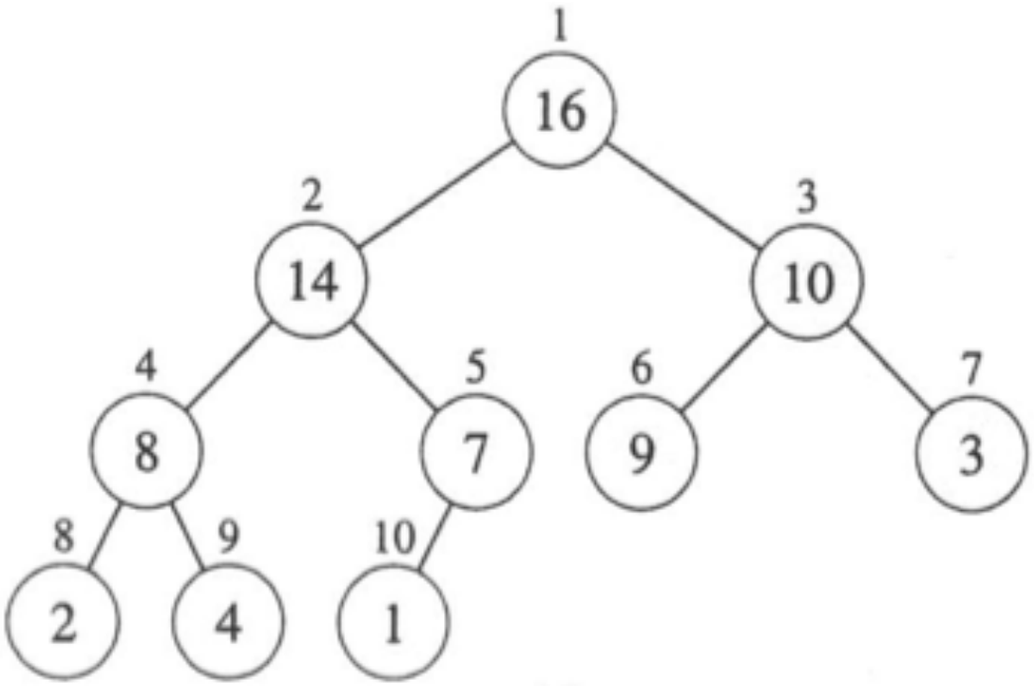
Definition 4.14 (Heap)

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

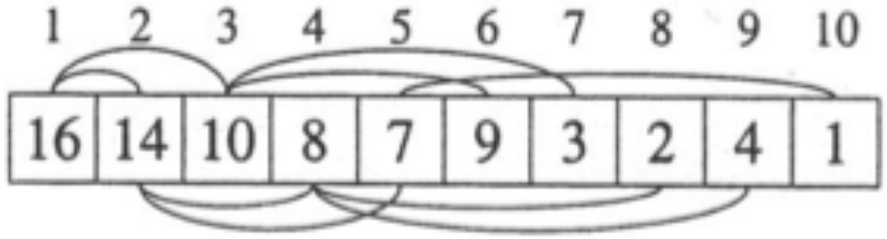
1. Jeder Knoten hat einen Schlüssel.
2. Ist h die maximale Distanz von der Wurzel, dann haben alle Ebenen i < h genau zwei Knoten.
3. Auf Ebene h sind die linken $n - 2^h + 1$ Positionen besetzt.
4. Der Schlüssel jedes Knotens ist mindestens so groß wie die seiner Kinder.

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

Heaps

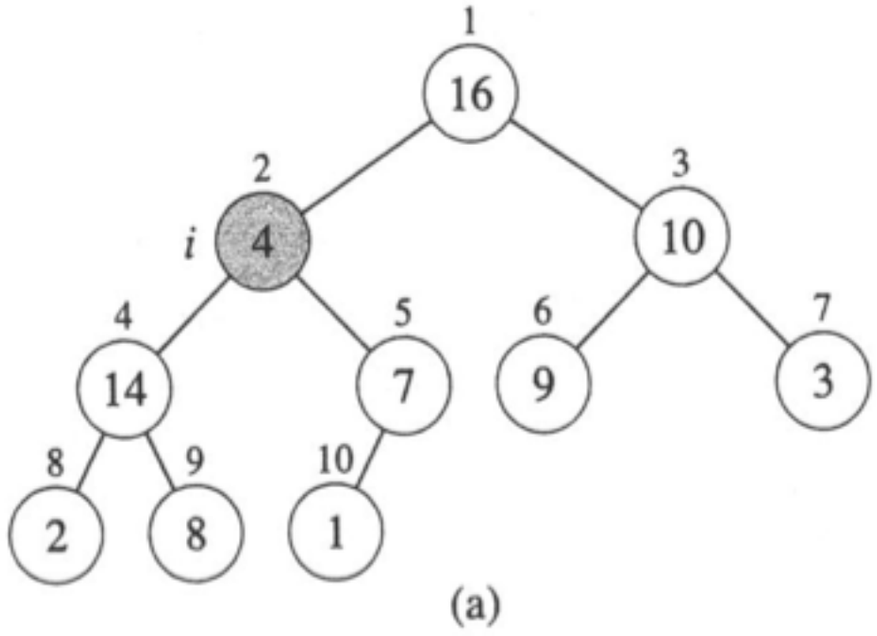


(a)

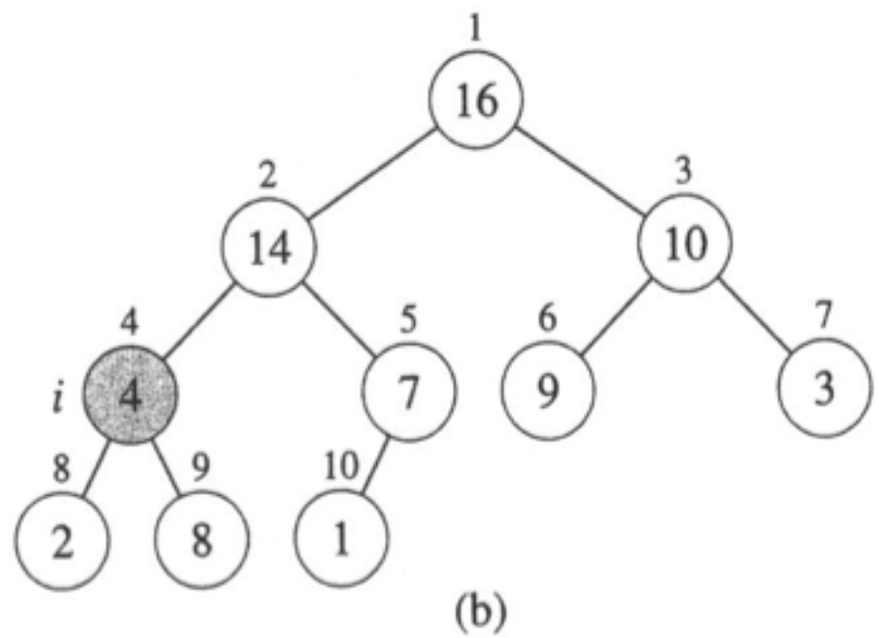


(b)

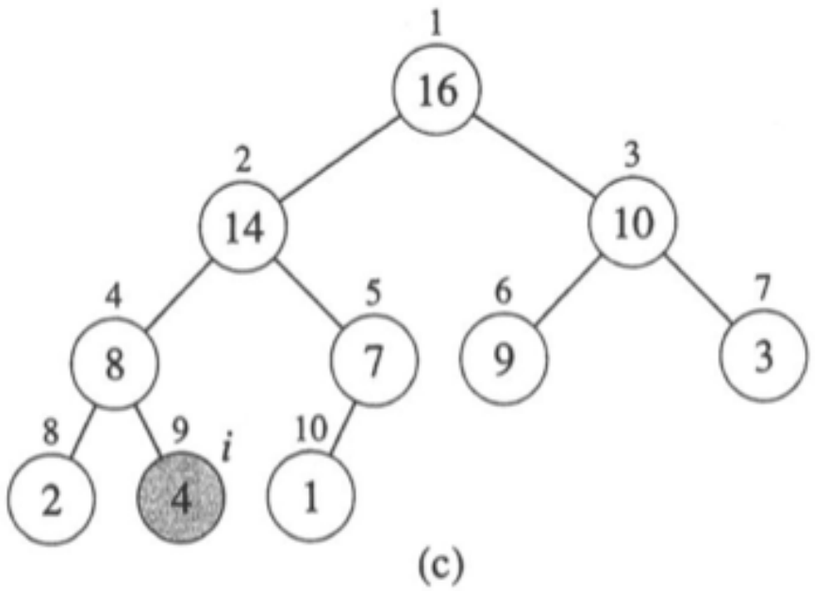
Heaps — Umordnen



(a)



(b)

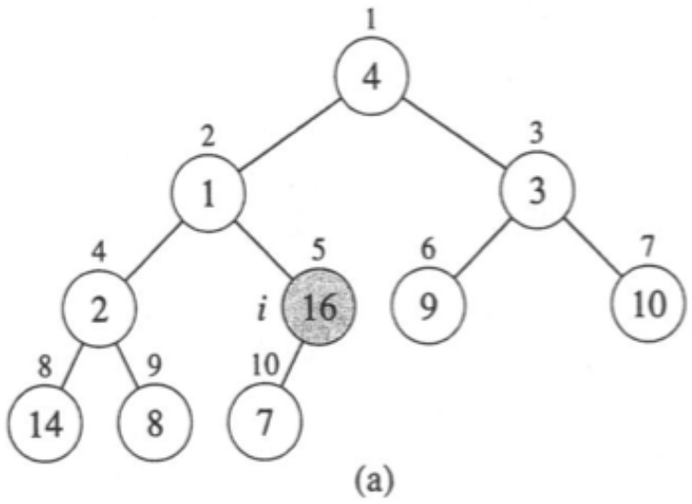


(c)

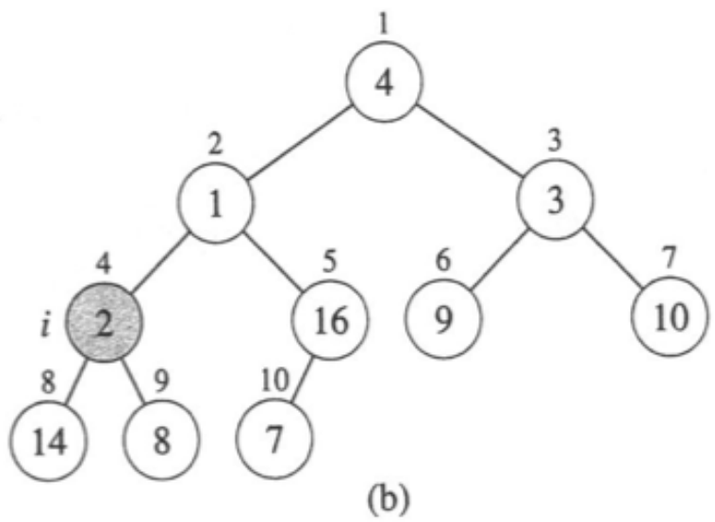
MAX-HEAPIFY(A, i)

```
1  l ← LEFT(i)
2  r ← RIGHT(i)
3  if l ≤ heap-größe[A] und A[l] > A[i]
4      then maximum ← l
5      else maximum ← i
6  if r ≤ heap-größe[A] und A[r] > A[maximum]
7      then maximum ← r
8  if maximum ≠ i
9      then vertausche A[i] ↔ A[maximum]
10          MAX-HEAPIFY(A, maximum)
```

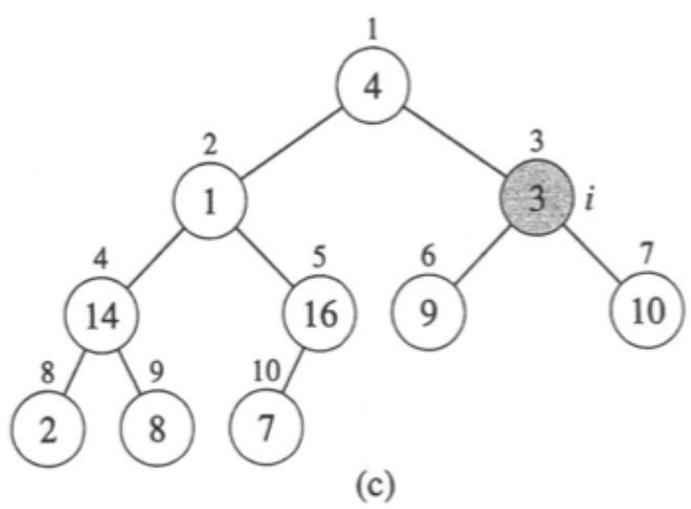
Heaps — Bauen



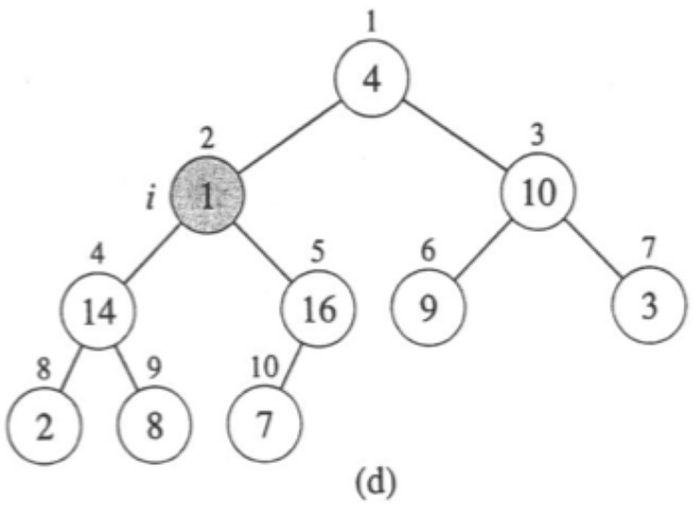
(a)



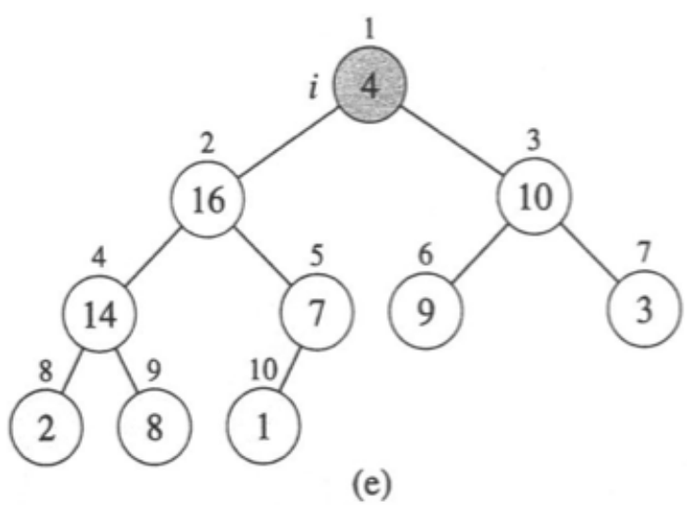
(b)



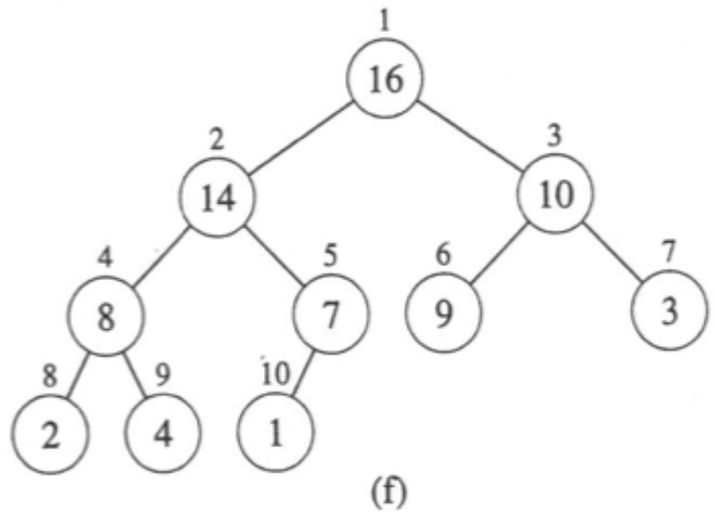
(c)



(d)



(e)



(f)

A	4	1	3	2	16	9	10	14	8	7

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

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:

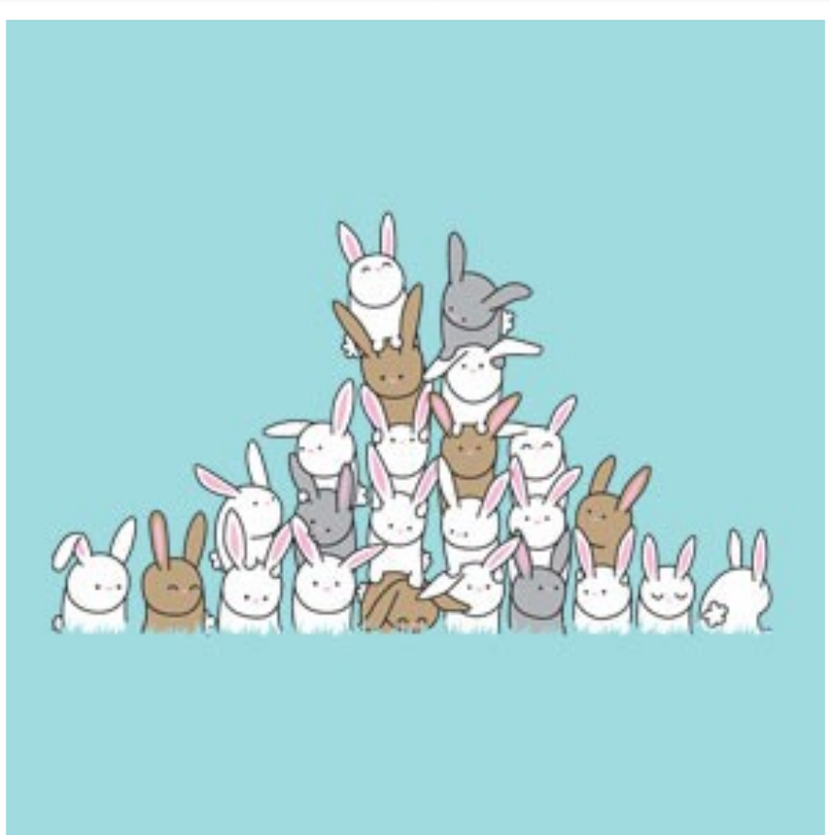
Satz 4.15
Ein Max-Heap mit n Knoten kann in $O(n)$ gebaut werden.

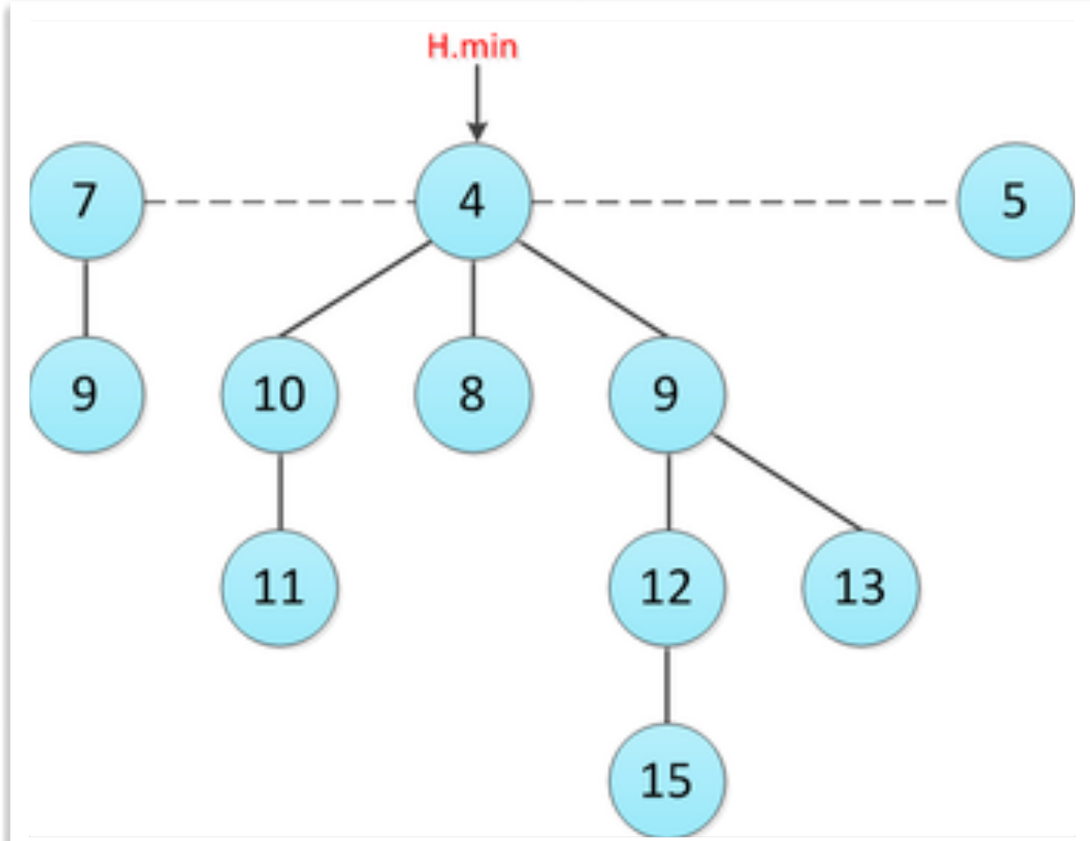
4.11 Weitere Varianten

4.11.1 Fibonacci Heaps

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

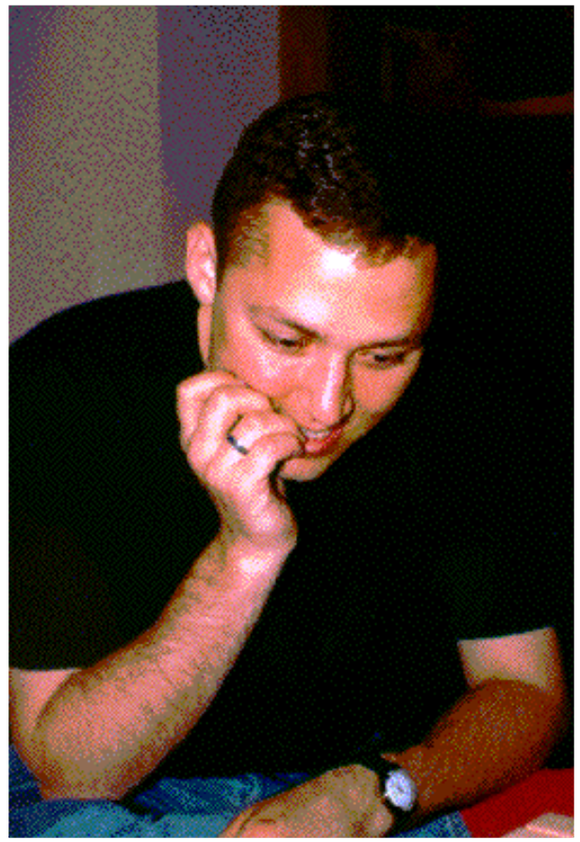
15

4.11.2 Cache-Oblivious B-Trees

Umgang mit großen Datenmengen bei unbekannter Cache-Größe

Michael Bender

Erik Demaine



Martin Farach-Colton

4.11.2 Cache-Oblivious B-Trees

Umg ...größe

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

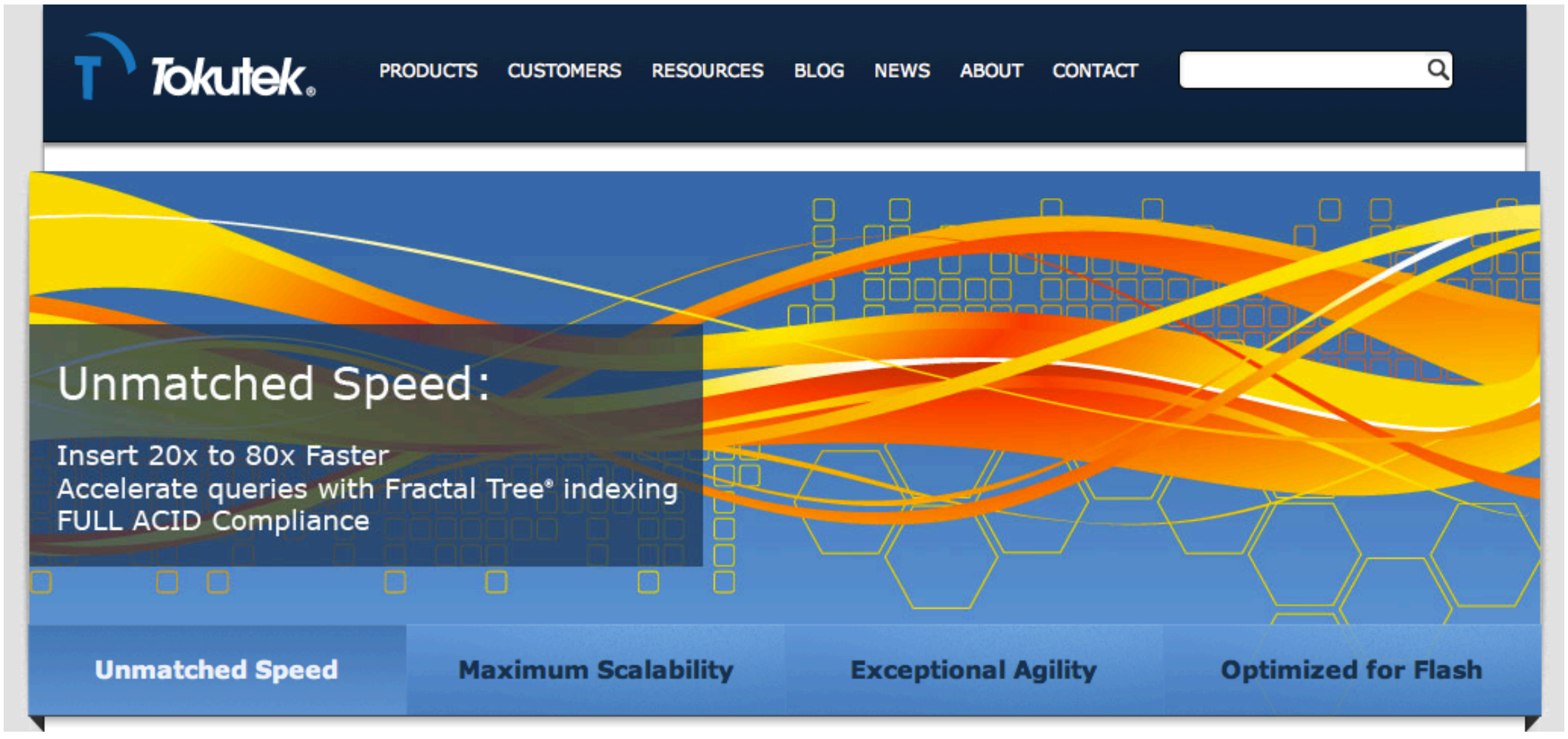
4.11.2 Cache-Oblivious B-Trees

Umg

größe

CACHE-OBLIVIOUS B-TREES*

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



ton

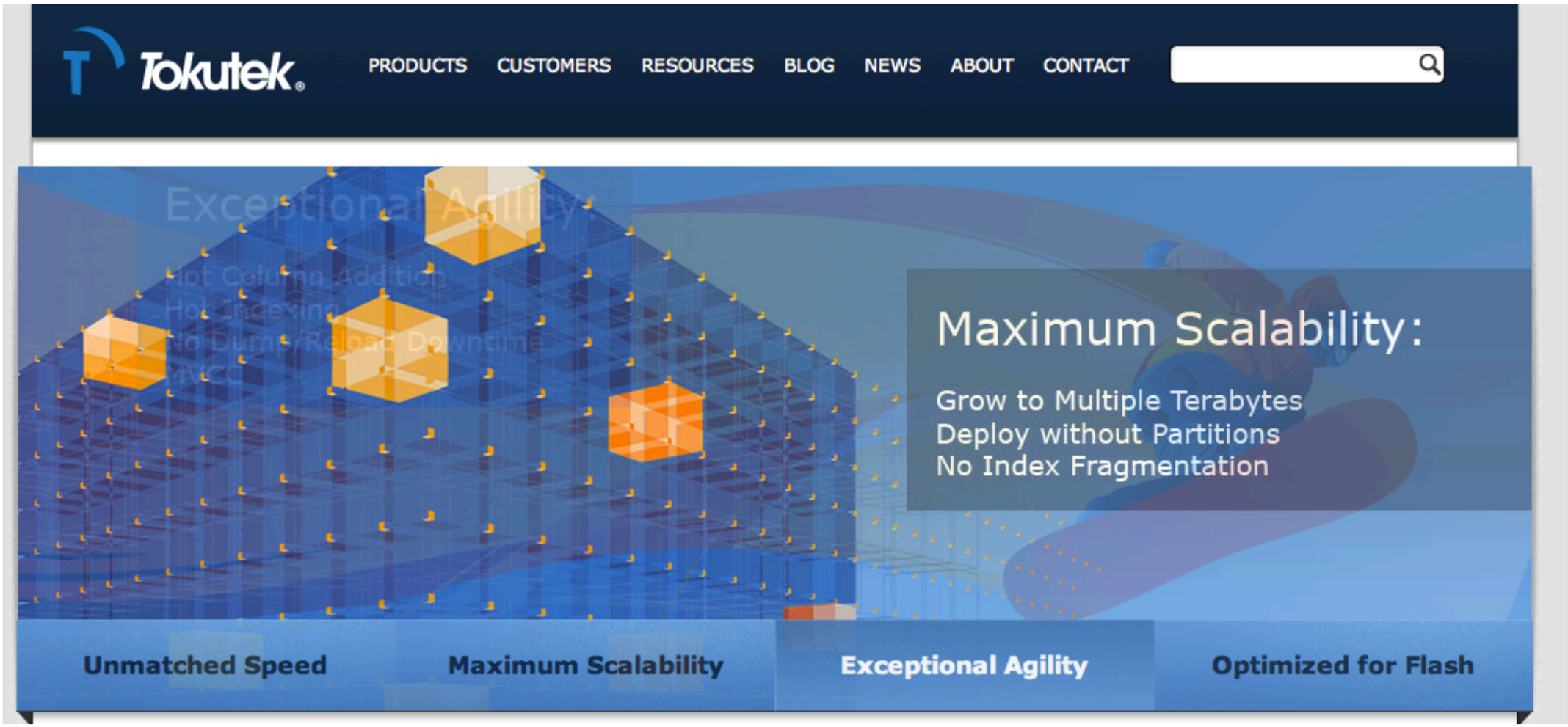
DOI. 10.1137/S0097539701389956

4.11.2 Cache-Oblivious B-Trees

Umg größe

CACHE-OBLIVIOUS B-TREES*

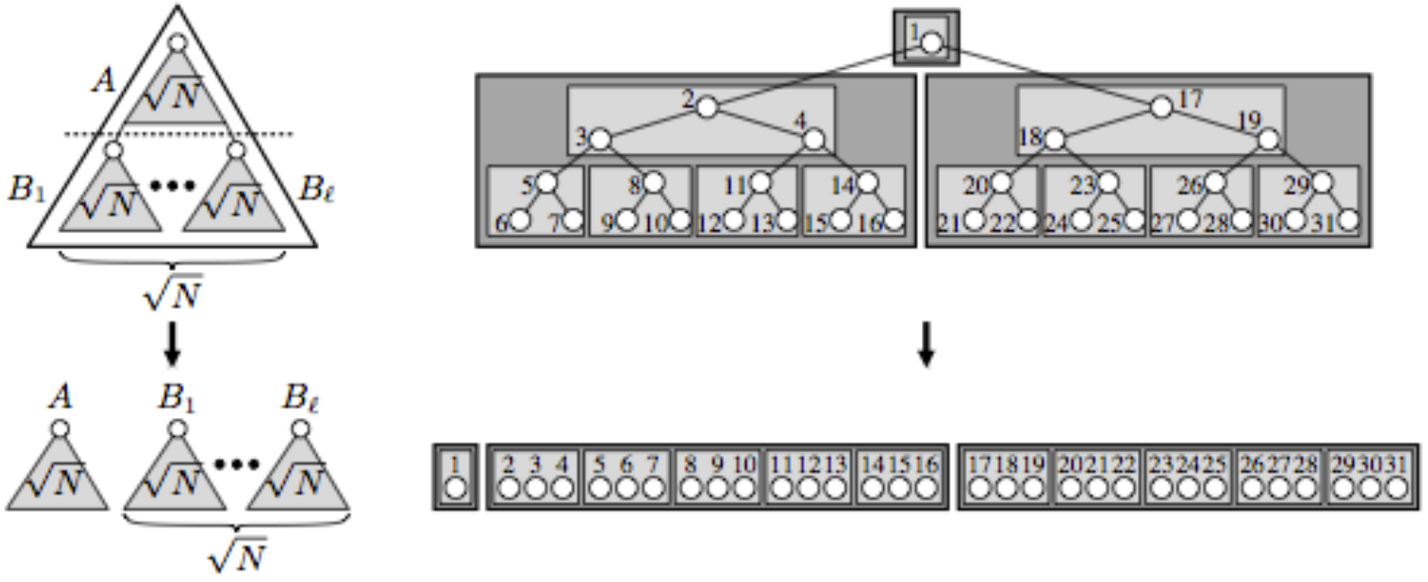
MICHAEL A. BENDER†, ERIK D. DEMAINE‡, AND MARTIN FARACH-COLTON§

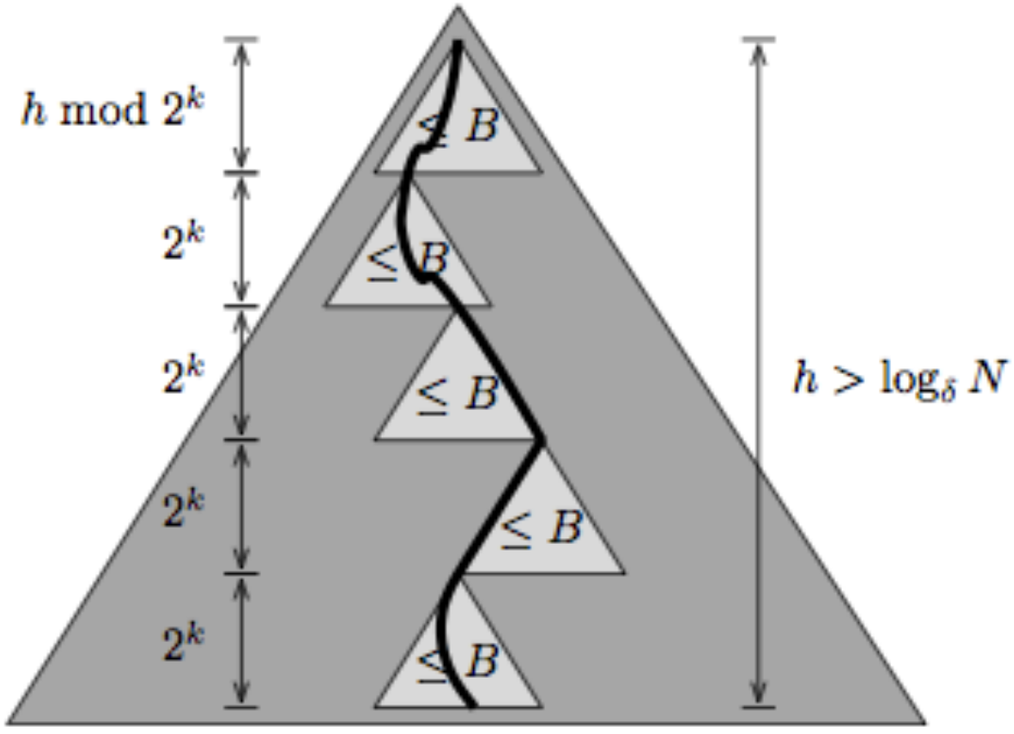


DOI. 10.1137/S0097539701389956

ton

Cache-Oblivious B-Trees





Vielen Dank!