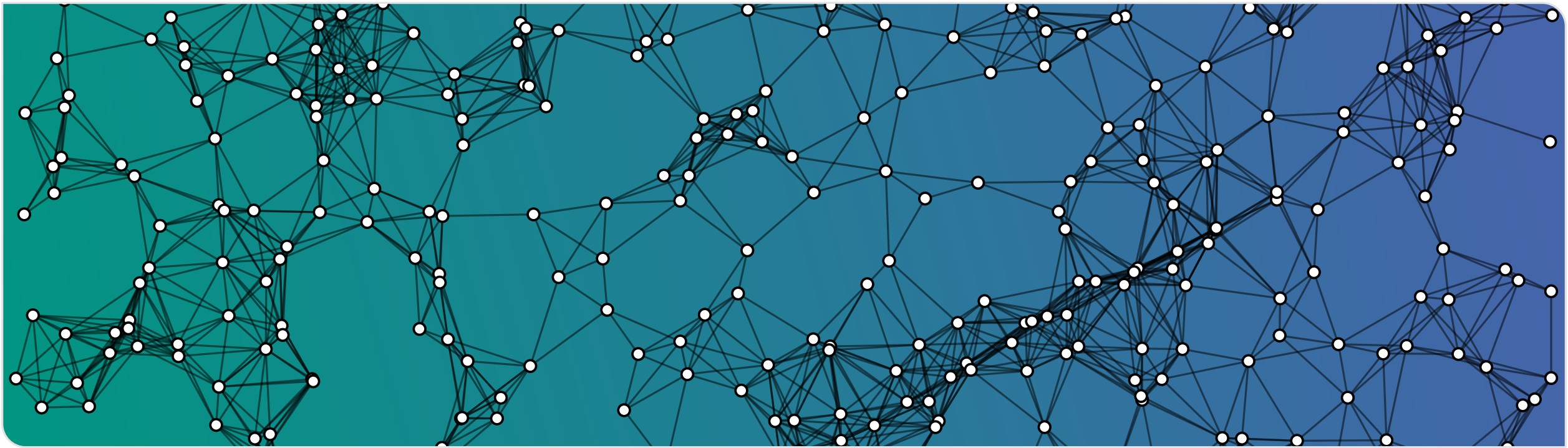
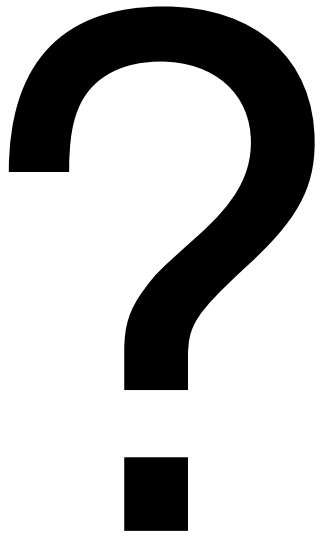
Karlsruher Institut für Technologie

Algorithmen 1

Übung 5
Suchbäume & Suche in Graphen

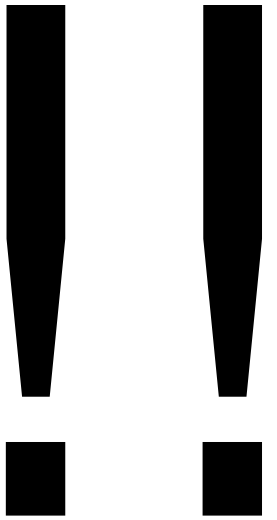


Ankündigung



Digitale Sprechstunde

- Morgen (27.6) um 17:00
- online Fragerunde auf Twitch
- stellt eure Fragen (gerne auf Discord oder auch live im Twitch Chat)



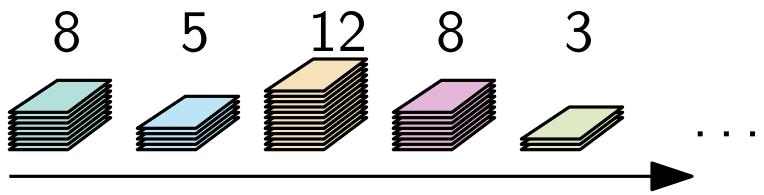
Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



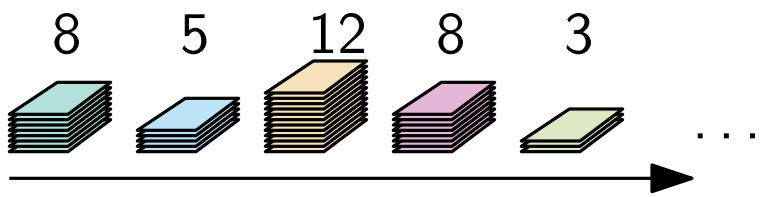
Tutorium 1 **Tutorium 2** **Tutorium 3**

$k = 20$

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



Das ist ein sehr schweres Problem.
Aber wir haben ein paar heuristische Ideen

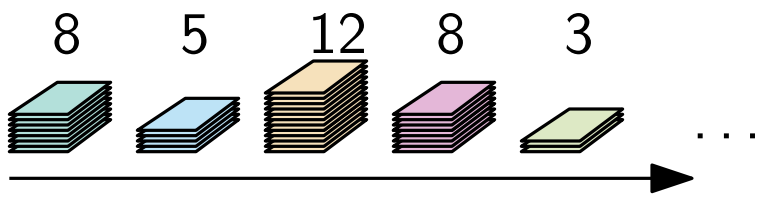
Tutorium 1 **Tutorium 2** **Tutorium 3**

$k = 20$

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



Tutorium 1 **Tutorium 2** **Tutorium 3**

$k = 20$

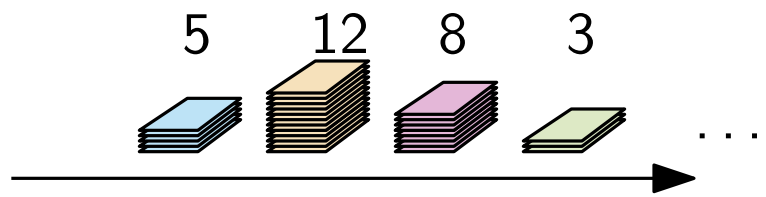
Das ist ein sehr schweres Problem.
Aber wir haben ein paar heuristische Ideen

Idee 1: Wir verteilen die Übungsblätter möglichst gleichmäßig.

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



Tutorium 1 **Tutorium 2** **Tutorium 3**

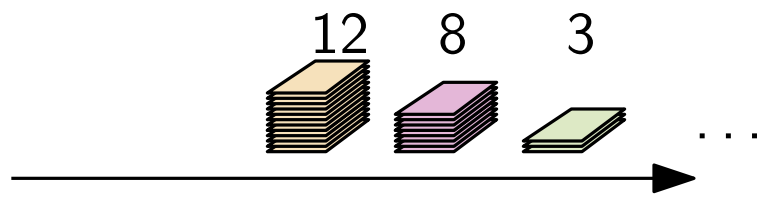
Das ist ein sehr schweres Problem.
Aber wir haben ein paar heuristische Ideen

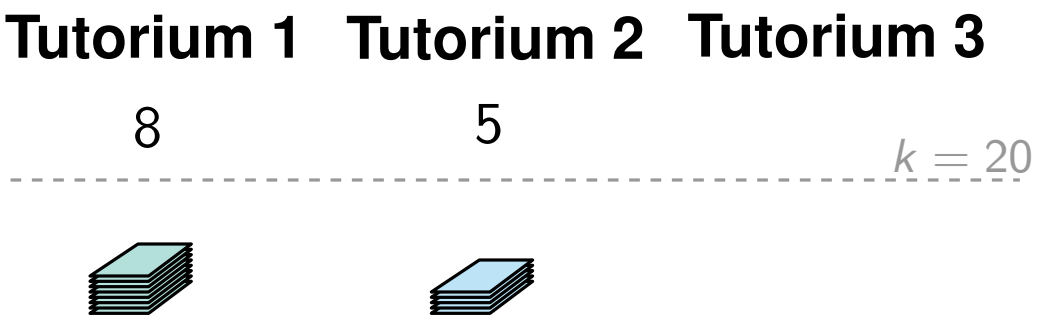
Idee 1: Wir verteilen die Übungsblätter möglichst gleichmäßig.

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen





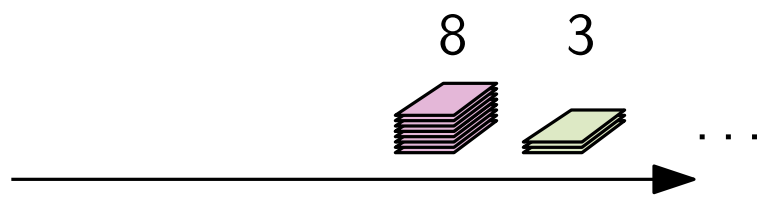
Das ist ein sehr schweres Problem.
Aber wir haben ein paar heuristische Ideen

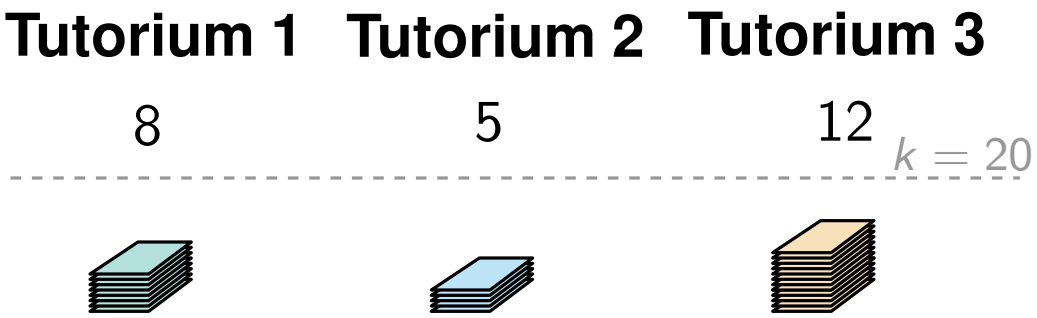
Idee 1: Wir verteilen die Übungsblätter möglichst gleichmäßig.

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen





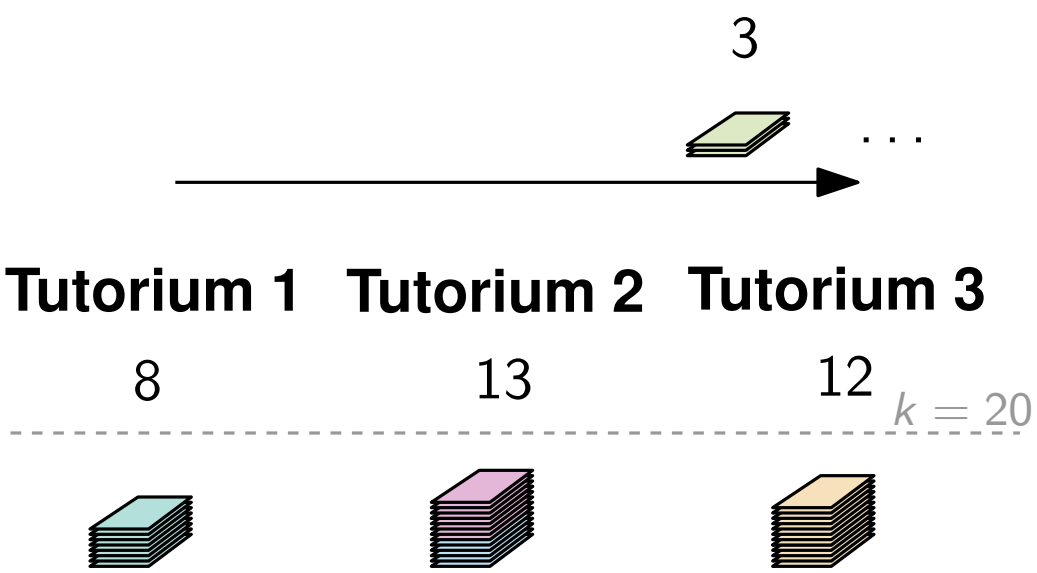
Das ist ein sehr schweres Problem.
Aber wir haben ein paar heuristische Ideen

Idee 1: Wir verteilen die Übungsblätter möglichst gleichmäßig.

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



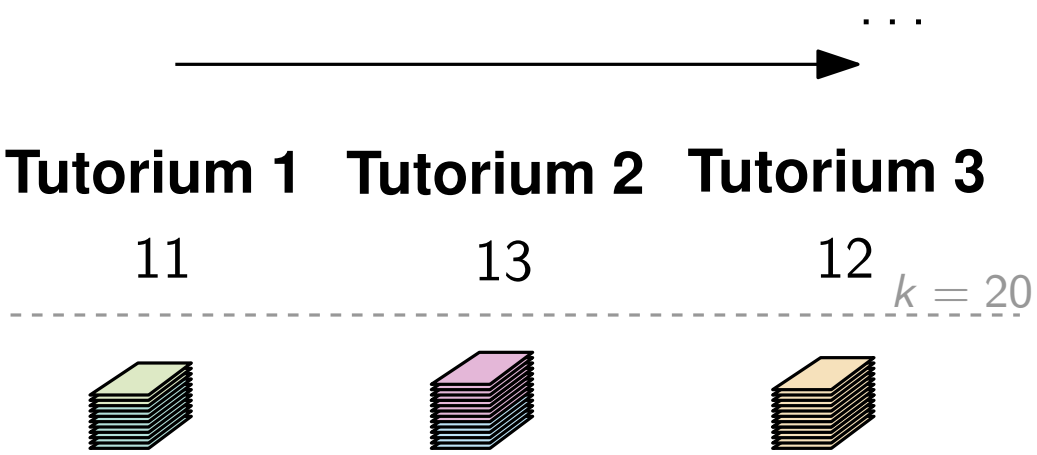
Das ist ein sehr schweres Problem.
Aber wir haben ein paar heuristische Ideen

Idee 1: Wir verteilen die Übungsblätter möglichst gleichmäßig.

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



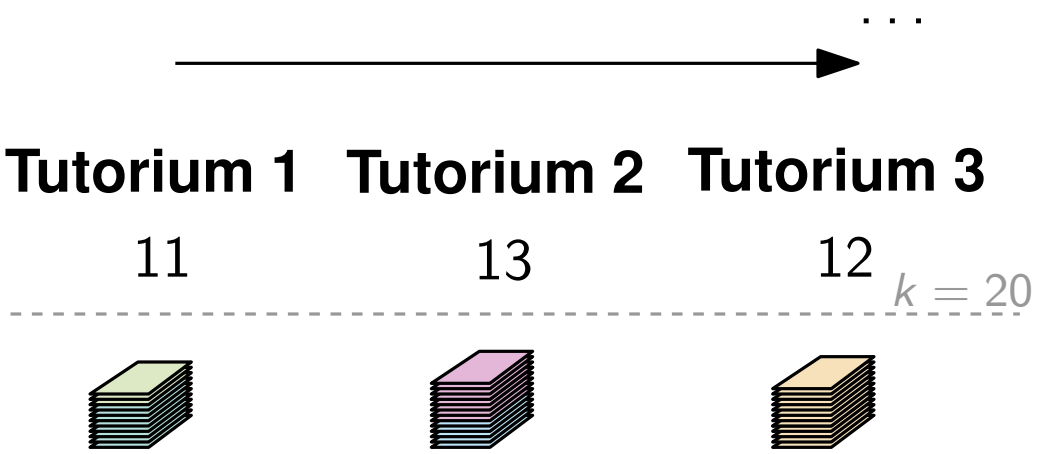
Das ist ein sehr schweres Problem.
Aber wir haben ein paar heuristische Ideen

Idee 1: Wir verteilen die Übungsblätter möglichst gleichmäßig.

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



Das ist ein sehr schweres Problem.
Aber wir haben ein paar heuristische Ideen

Idee 1: Wir verteilen die Übungsblätter möglichst gleichmäßig.

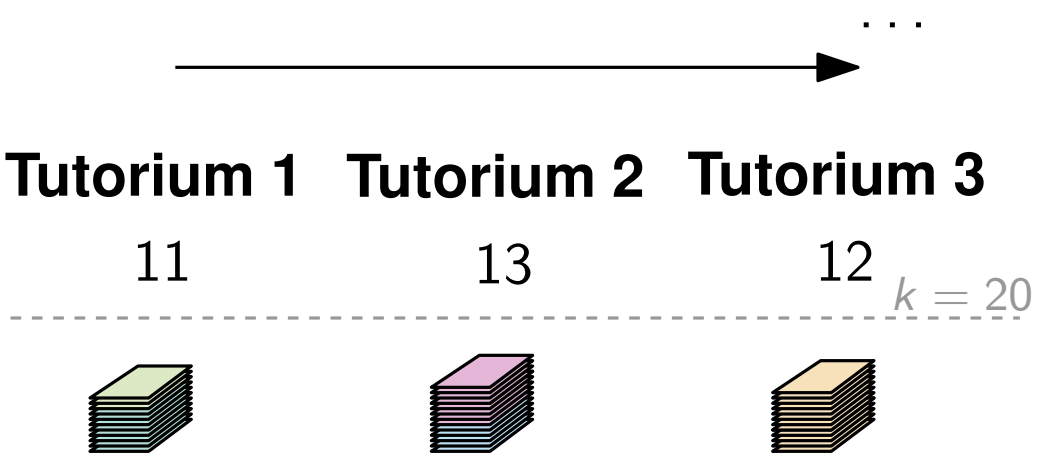
- **Min-Heap**: speichert, wie viele Blätter jedes Tutorium schon korrigieren muss.
- Gib neue Abgabe dem Tutorium mit den wenigsten Blättern

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen

Hoffentlich haben wir genug Tutorien



Das ist ein sehr schweres Problem.
Aber wir haben ein paar heuristische Ideen

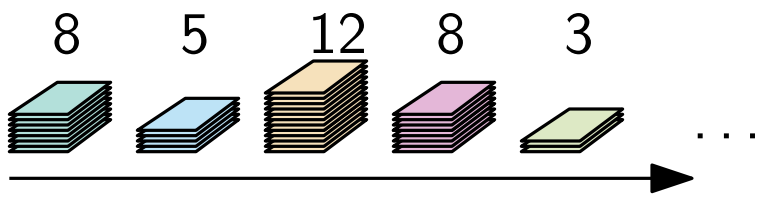
Idee 1: Wir verteilen die Übungsblätter möglichst gleichmäßig.

- **Min-Heap**: speichert, wie viele Blätter jedes Tutorium schon korrigieren muss.
- Gib neue Abgabe dem Tutorium mit den wenigsten Blättern

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



Idee 2: Wir minimieren die Anzahl an Tutorien, die wir brauchen

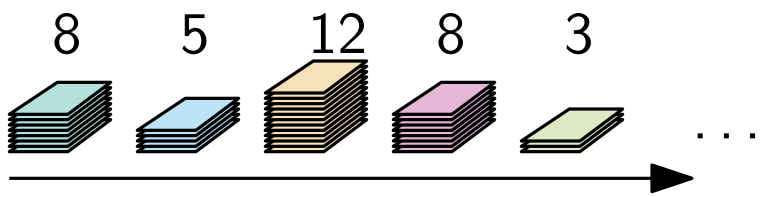
Tutorium 1 **Tutorium 2** **Tutorium 3**

$k = 20$

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



Tutorium 1 **Tutorium 2** **Tutorium 3**

$k = 20$

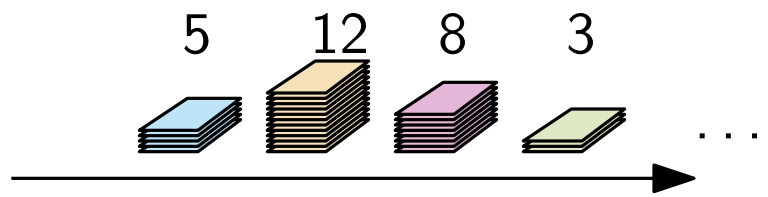
Idee 2: Wir minimieren die Anzahl an Tutorien, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



Tutorium 1 **Tutorium 2** **Tutorium 3**

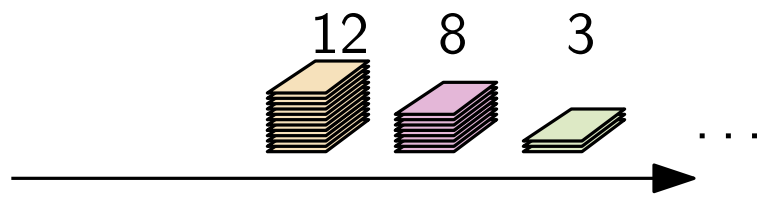
Idee 2: Wir minimieren die Anzahl an Tutorien, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



Tutorium 1 **Tutorium 2** **Tutorium 3**

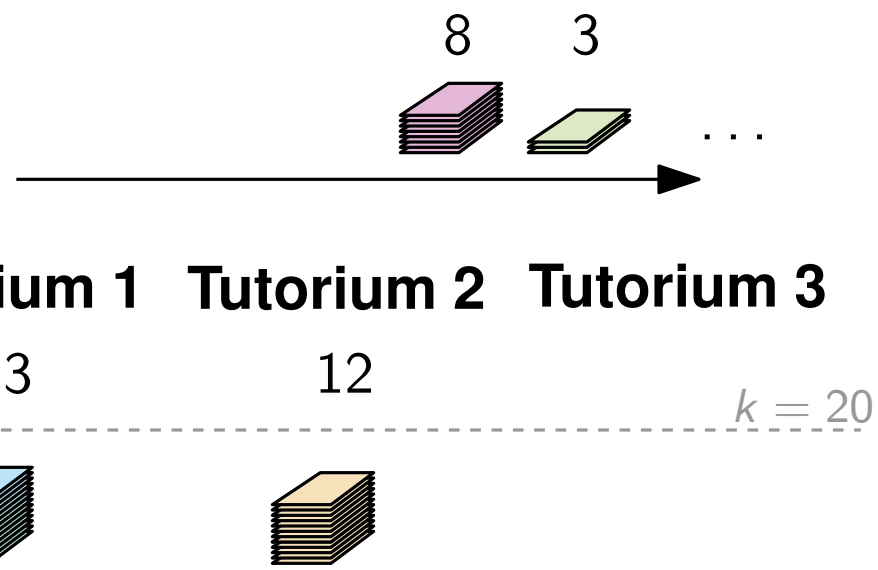
Idee 2: Wir minimieren die Anzahl an Tutorien, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



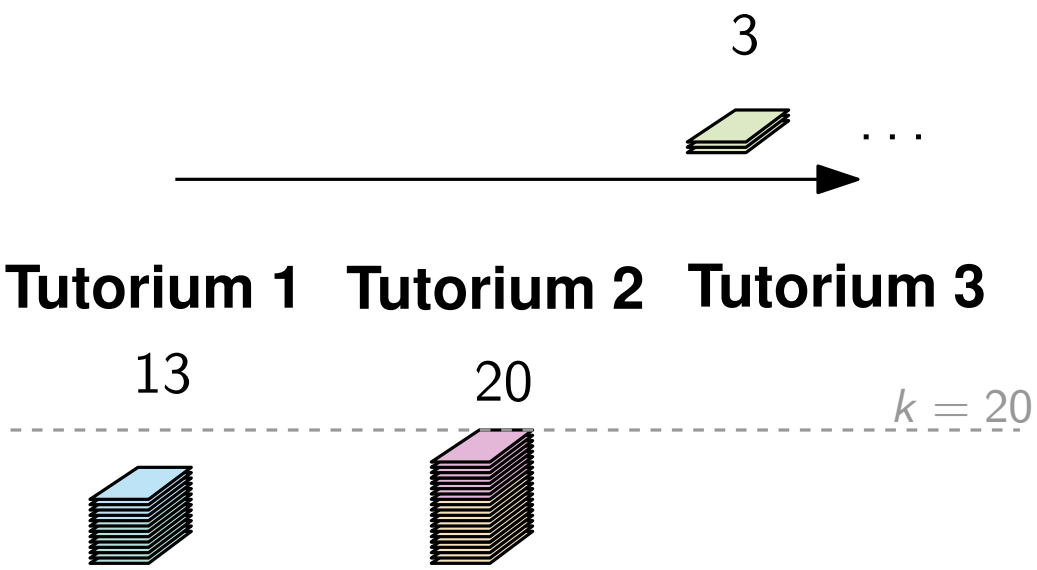
Idee 2: Wir minimieren die Anzahl an Tutorien, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



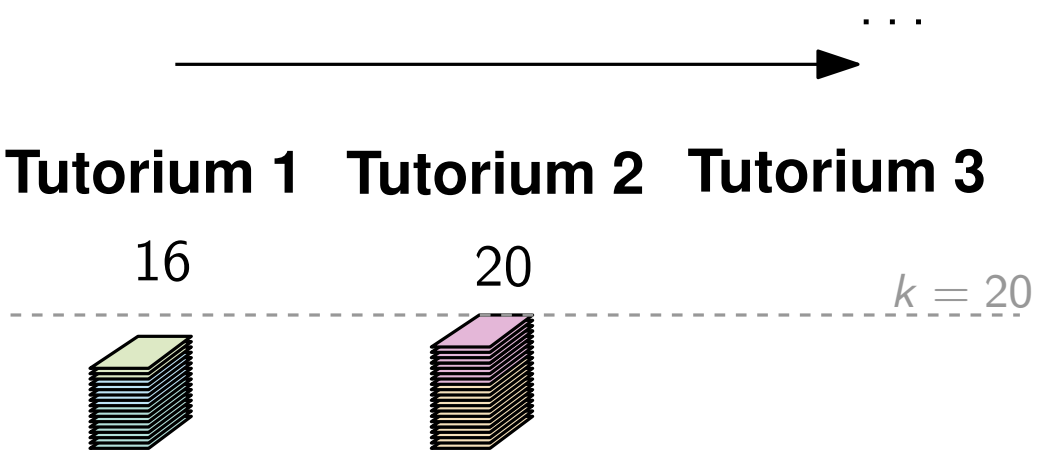
Idee 2: Wir minimieren die Anzahl an Tutorien, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Die Übungsleitung muss sich überlegen, welches Tutorium, welche Abgabe erhält
- Die Abgaben treffen nacheinander bei uns ein. Abgabe i besteht aus a_i Seiten
- Wir haben m Tutorien. Am Ende soll jede Abgabe bei genau einem Tutorium landen



Idee 2: Wir minimieren die Anzahl an Tutorien, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

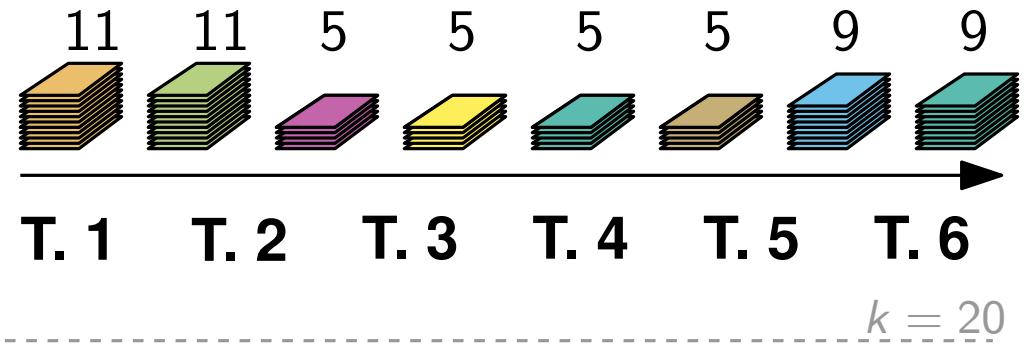
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)



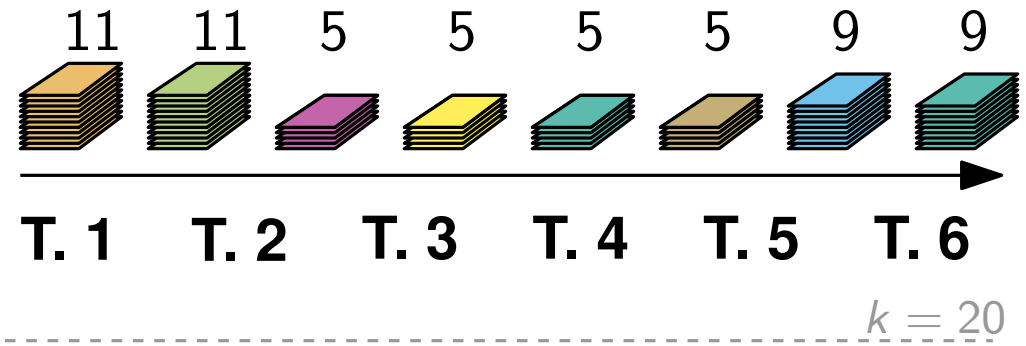
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?



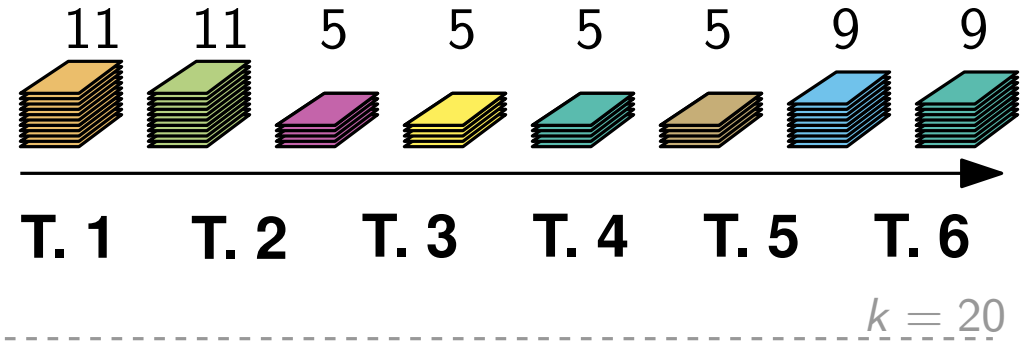
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



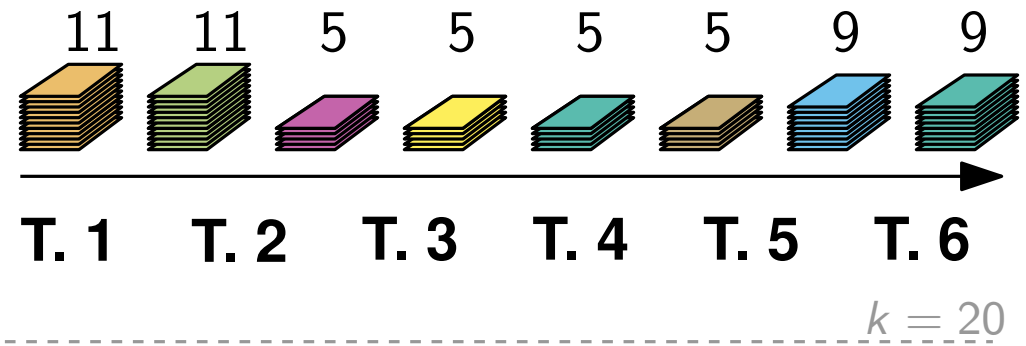
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

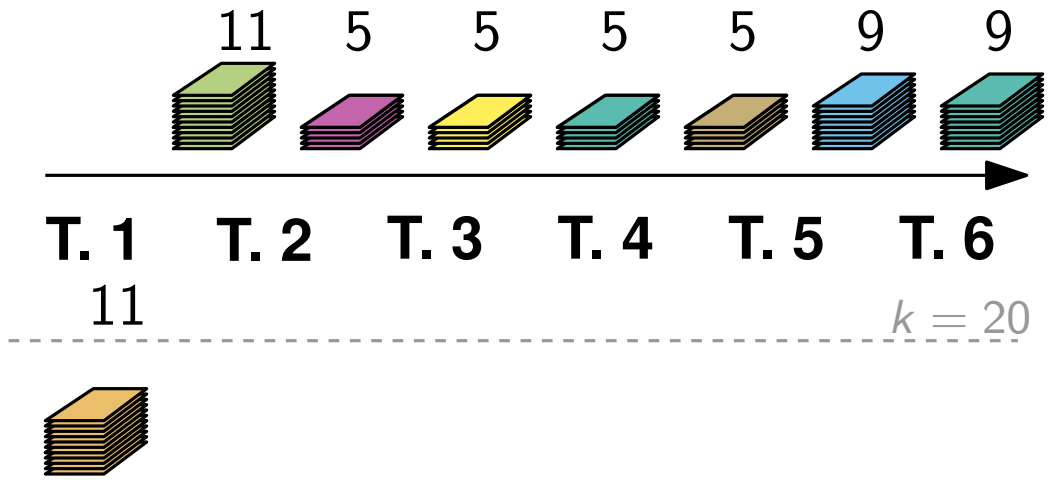
- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

`pwa.klicker.uzh.ch/join/algo1`

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



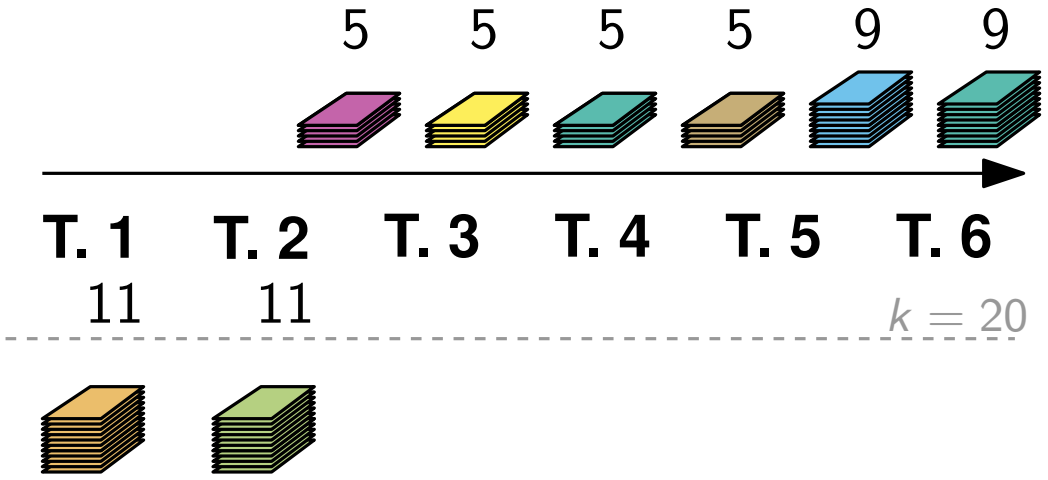
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



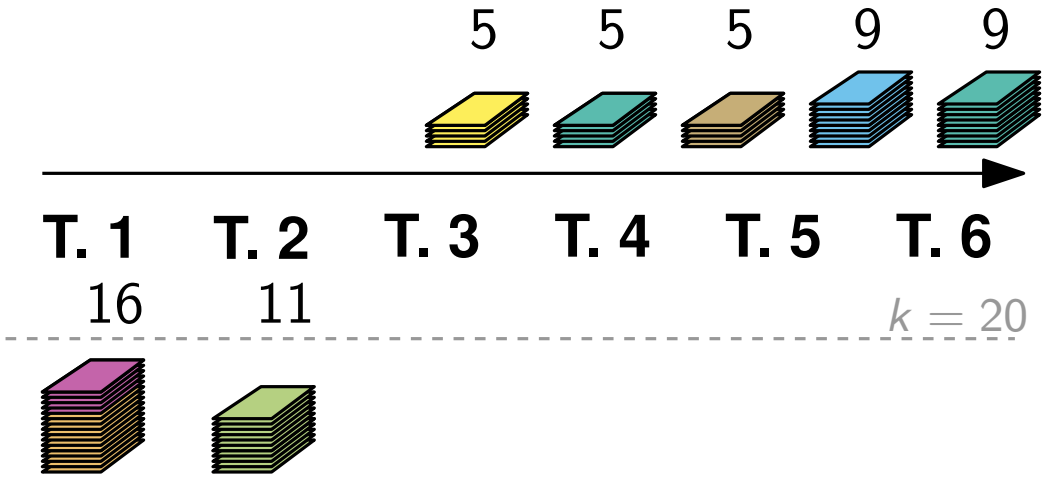
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



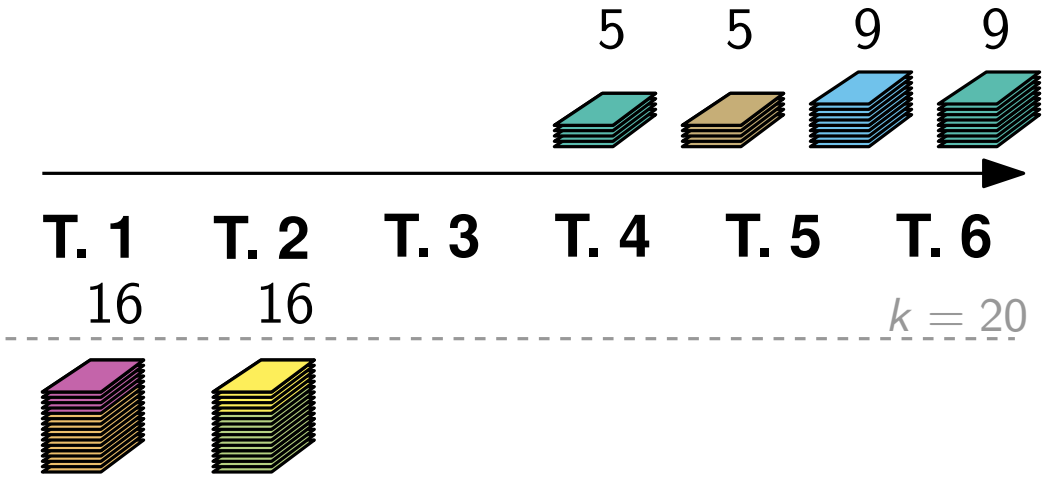
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



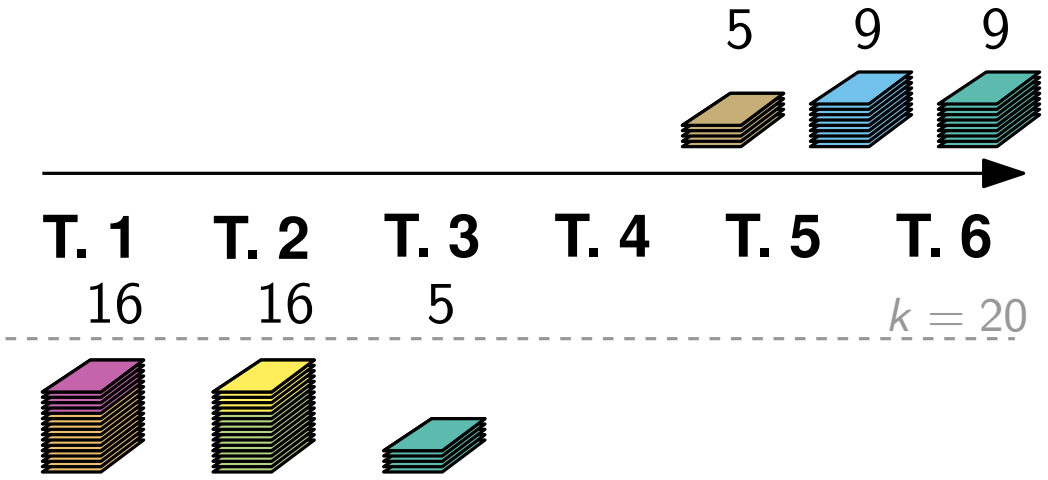
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



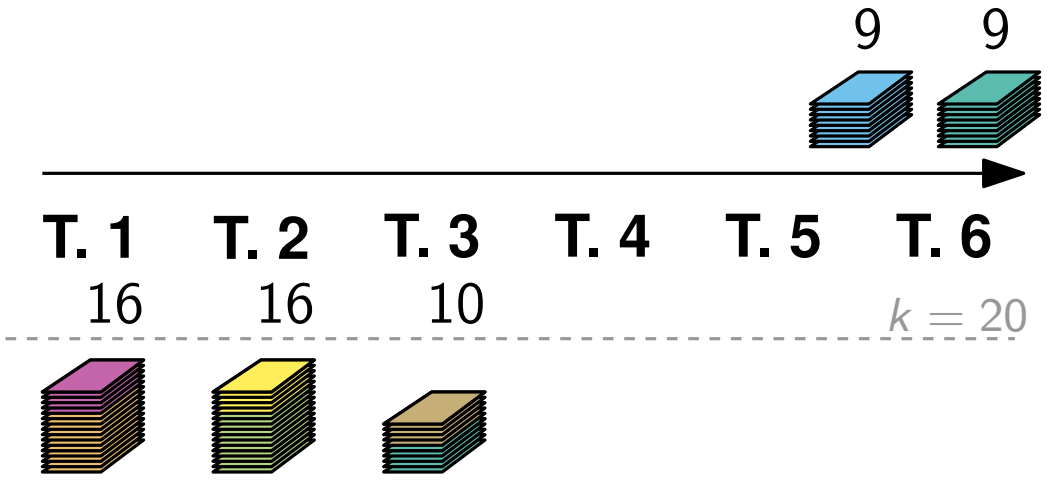
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



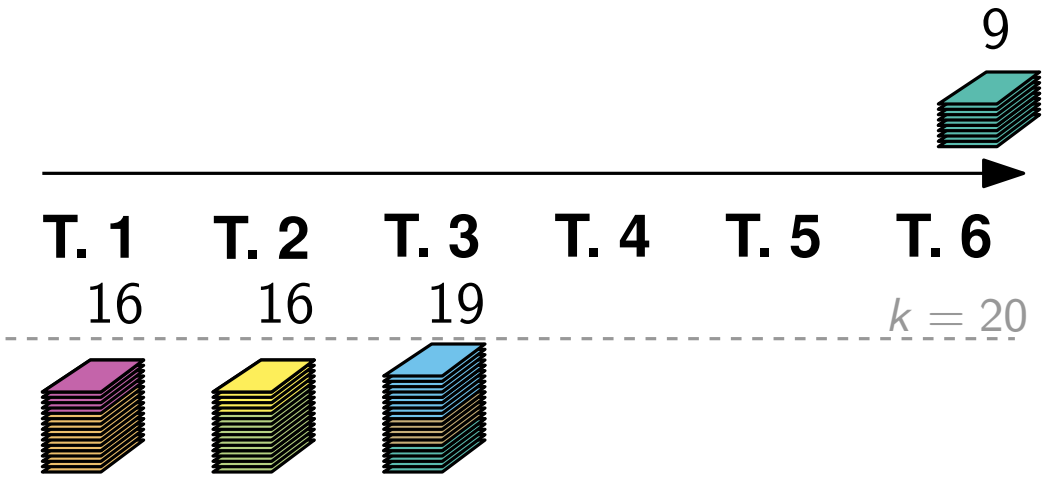
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

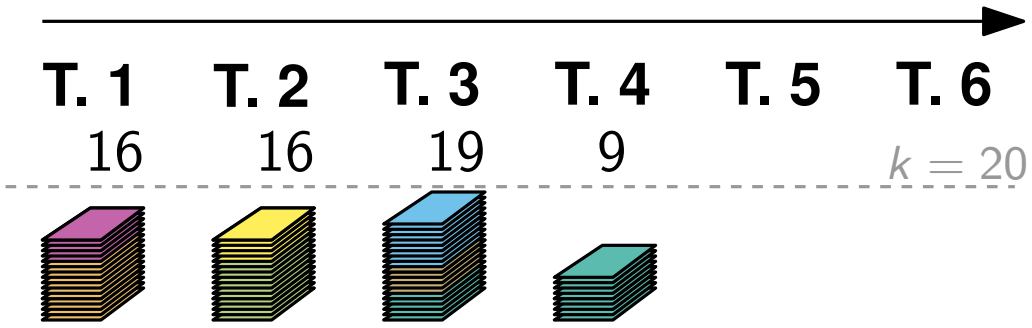
Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$)
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?

Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)



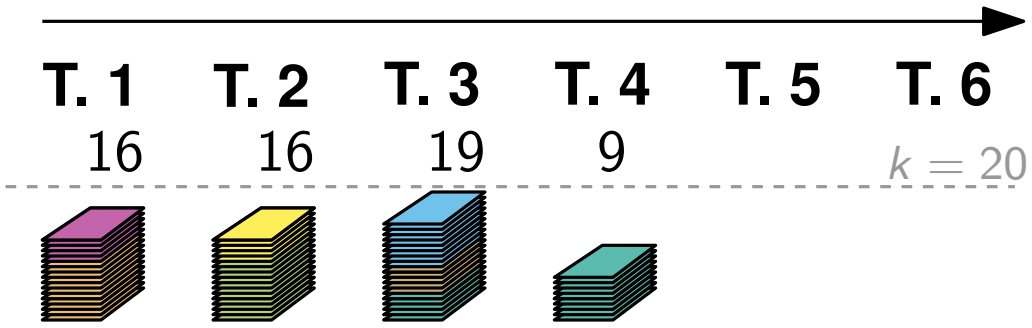
Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$) 4
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?

Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

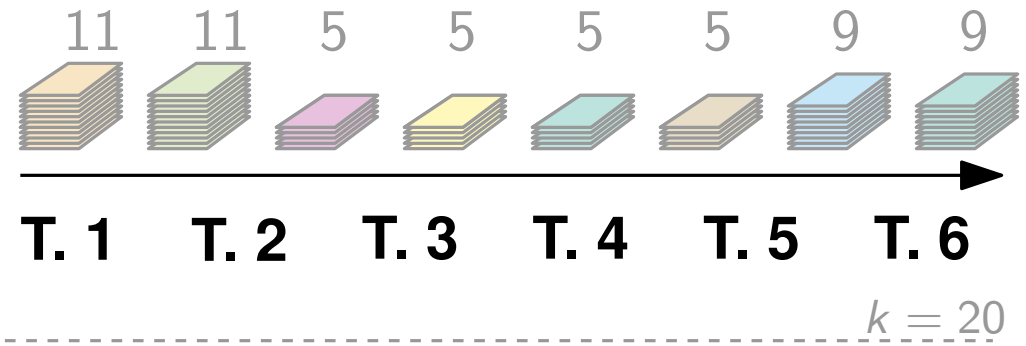
- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)



Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$) 4
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



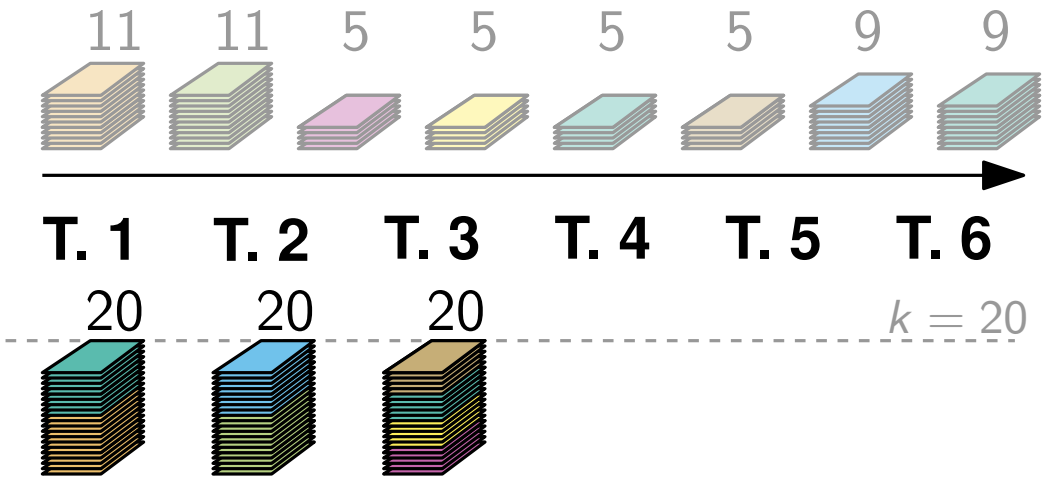
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$) 4
- Was ist die minimale Anzahl an Tutorien, die wir brauchen?
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



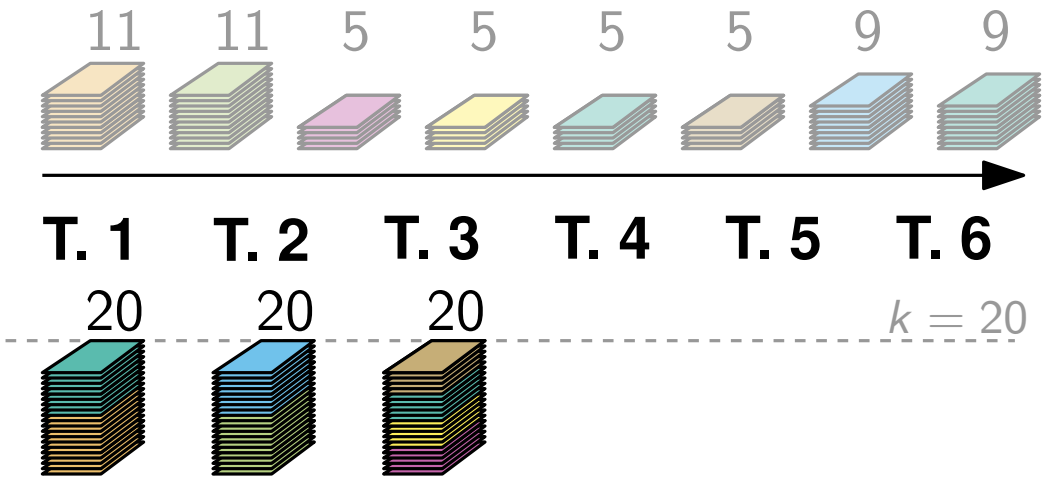
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$) 4
- Was ist die minimale Anzahl an Tutorien, die wir brauchen? 3
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x?



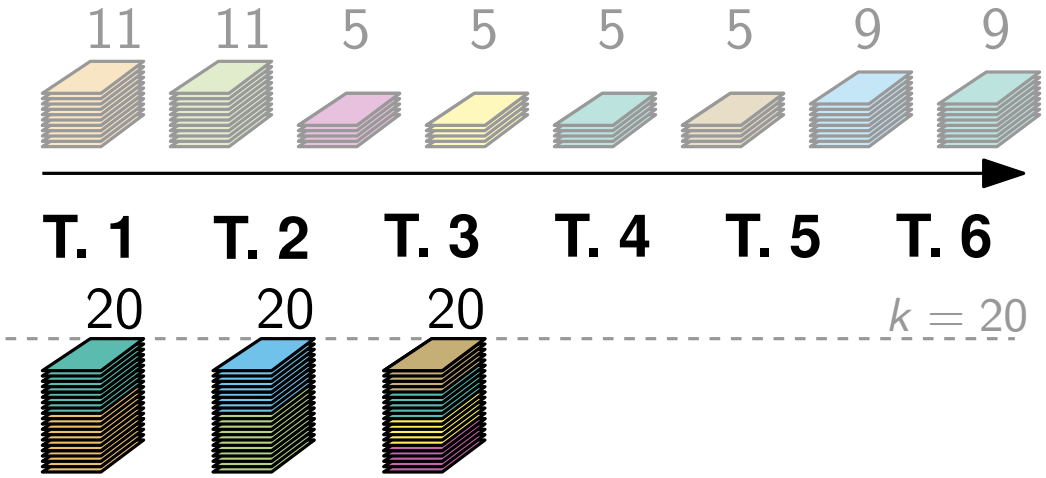
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$) 4
- Was ist die minimale Anzahl an Tutorien, die wir brauchen? 3
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x? $17/10 = 1.7$



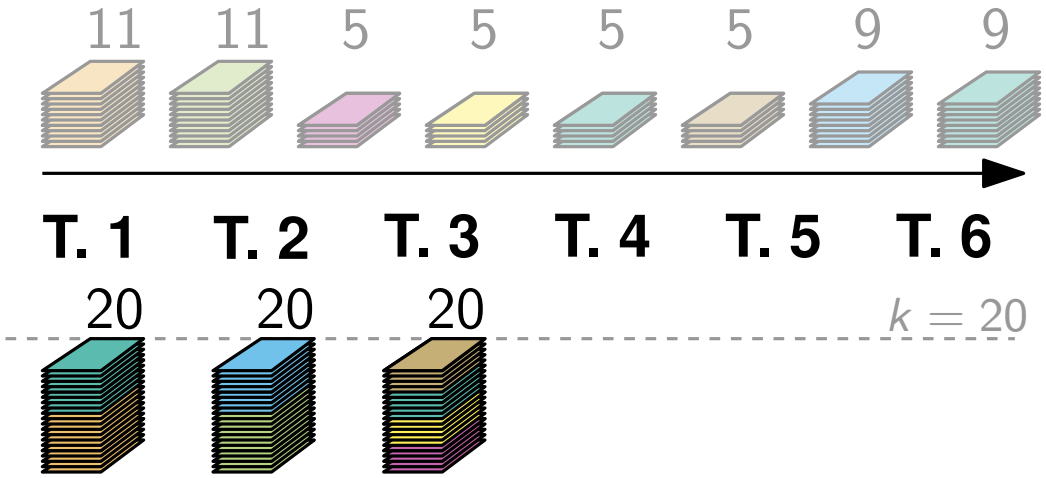
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$) 4
- Was ist die minimale Anzahl an Tutorien, die wir brauchen? 3
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x? $17/10 = 1.7$



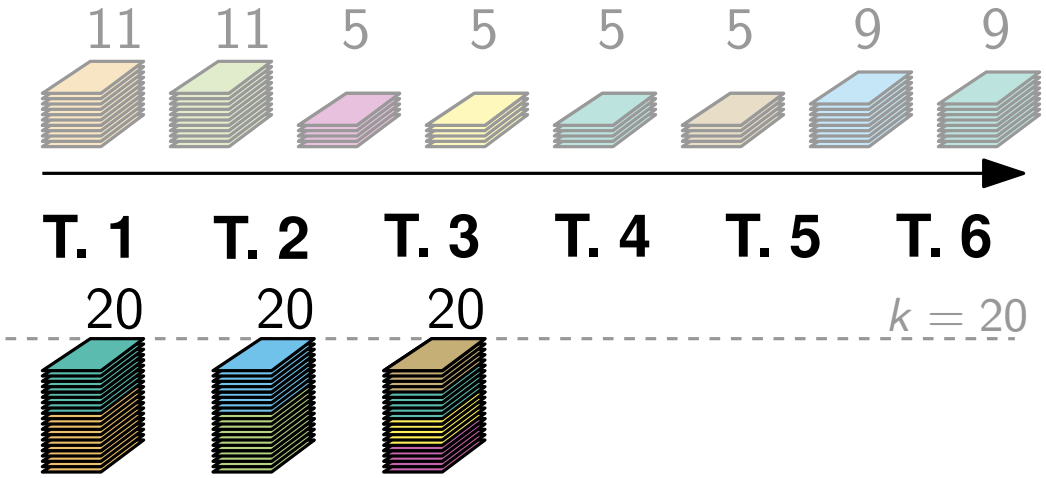
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)
- Der beste Algorithmus brauch ≈ 1.59 mehr Tutorien als eine optimale Lösung

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$) 4
- Was ist die minimale Anzahl an Tutorien, die wir brauchen? 3
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x? $17/10 = 1.7$



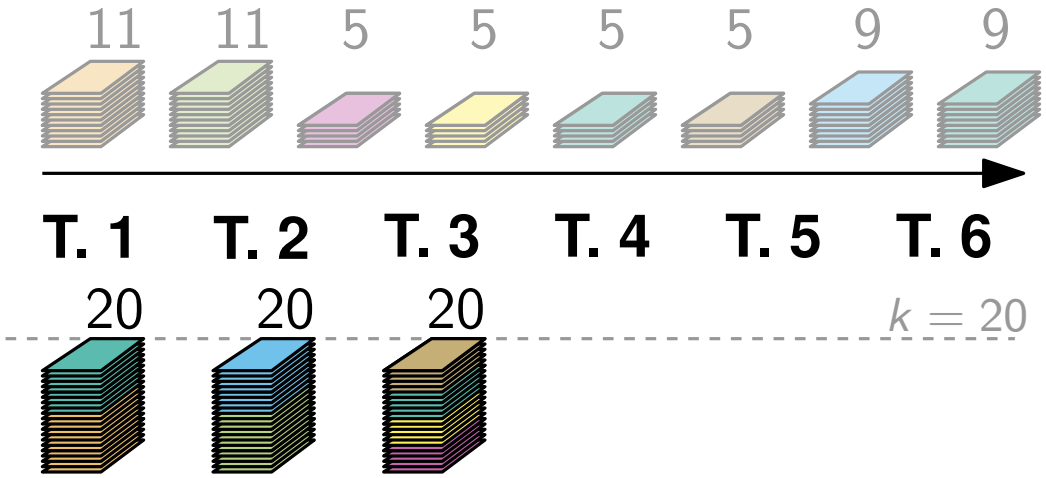
Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)
- Der beste Algorithmus brauch ≈ 1.59 mehr Tutorien als eine optimale Lösung
- Speichere die aktuelle Anzahl an Blättern jedes Tutoriums in einer **sortierten Folge**
- In jedem Schritt: Abgabe hat a_i Seiten $\rightarrow$ finde Tutorium mit den meisten Blättern

Korrekturlimit

Die Tutoren-Gewerkschaft fordert: maximal k Seiten Übungsblatt-Abgaben pro Tutorium

- Wie viele Tutorien braucht unser Verfahren bei folgenden Abgaben? ($k = 20$) 4
- Was ist die minimale Anzahl an Tutorien, die wir brauchen? 3
- Im Allgemeinen gilt: Unser Verfahren braucht maximal x-mal mehr Tutorien als eine optimale Verteilung. Was ist x? $17/10 = 1.7$



Idee 2: Wir minimieren die Anzahl an Tutoren, die wir brauchen

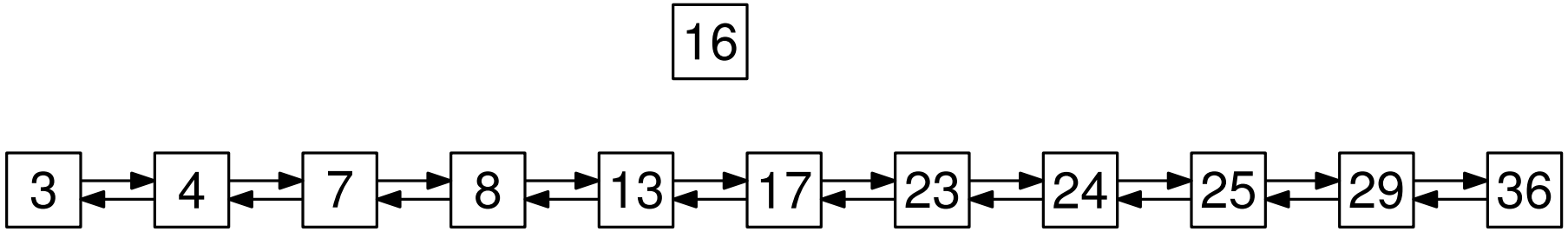
- Gib die Abgabe dem Tutorium, dessen Restlimit am passensten gefüllt wird
- In anderen Worten: Gib die Abgabe dem Tutorium mit den meisten Blättern (ohne k zu überschreiten)
- Der beste Algorithmus brauch ≈ 1.59 mehr Tutorien als eine optimale Lösung
- Speichere die aktuelle Anzahl an Blättern jedes Tutoriums in einer **sortierten Folge**
- In jedem Schritt: Abgabe hat a_i Seiten $\rightarrow$ finde Tutorium mit den meisten Blättern $\leq k - a_i$

Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen

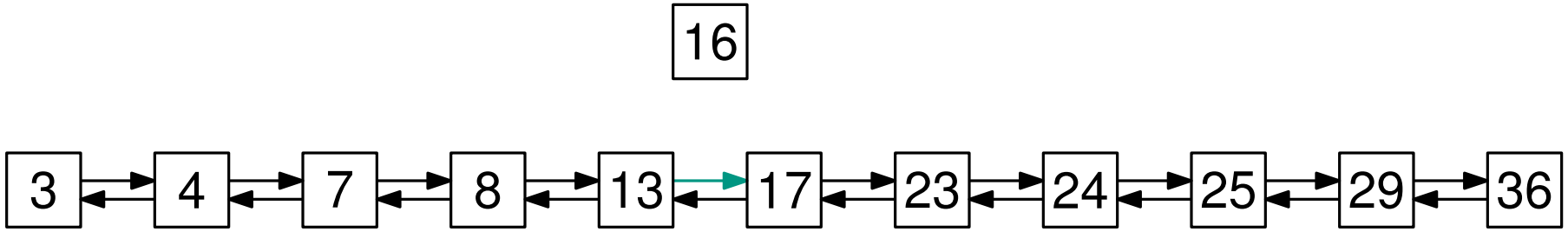
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen



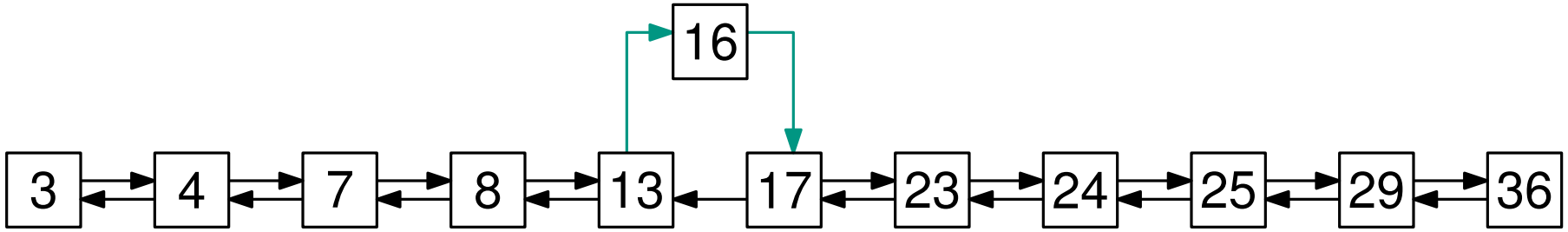
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen



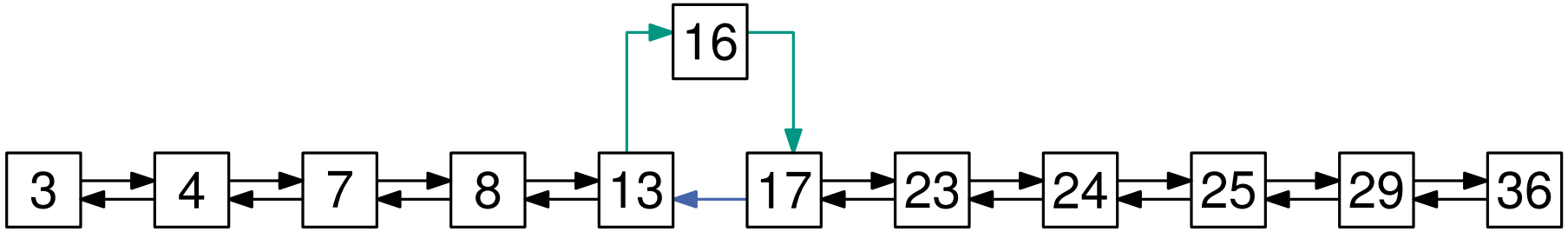
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen



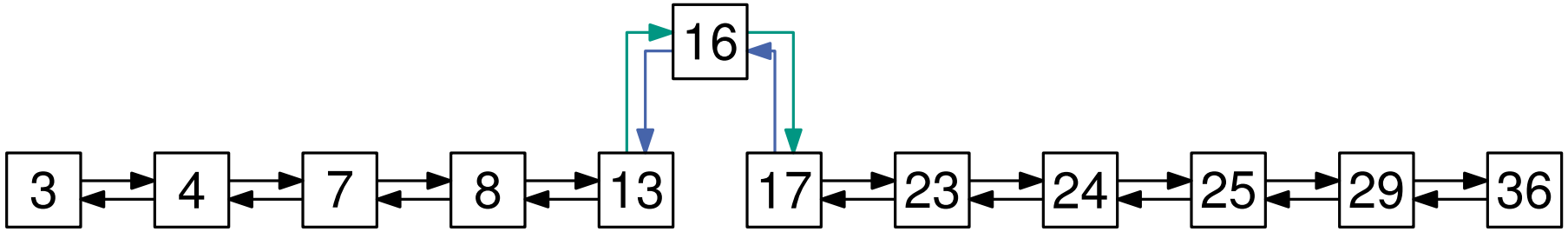
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen



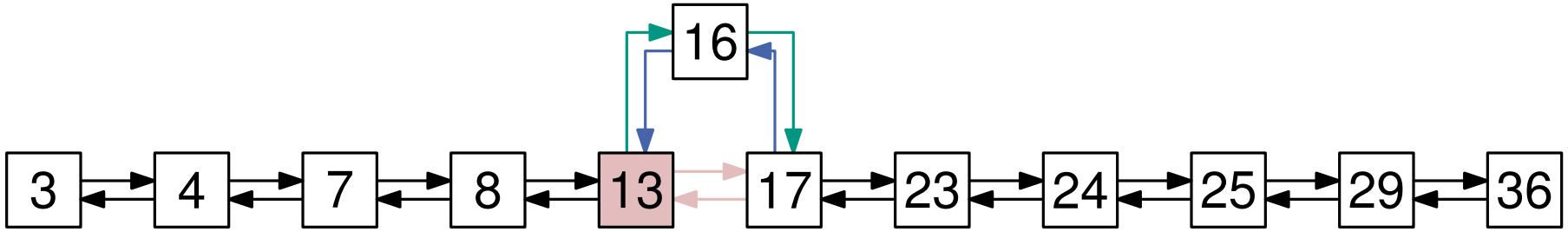
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen



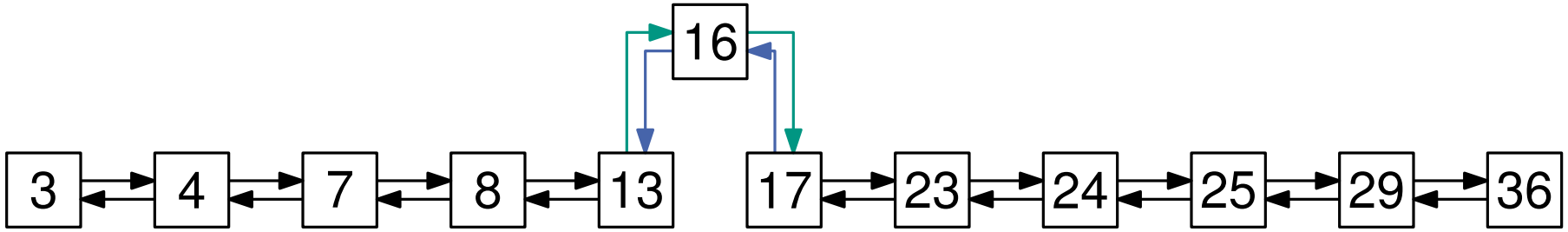
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element



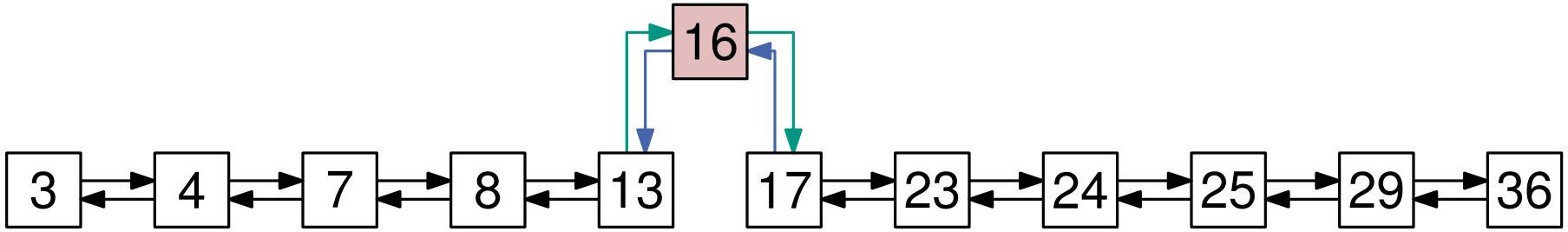
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen



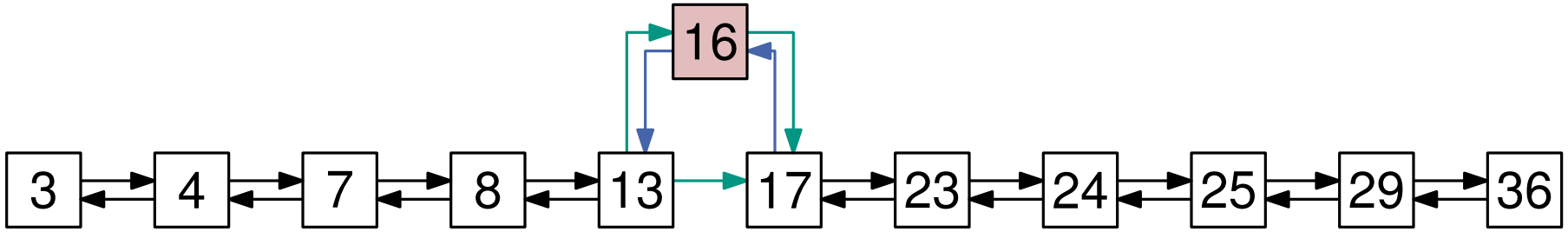
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen



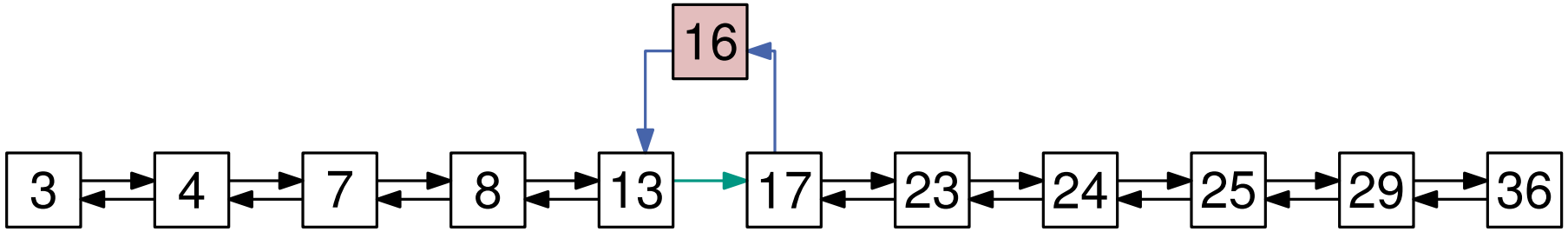
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen



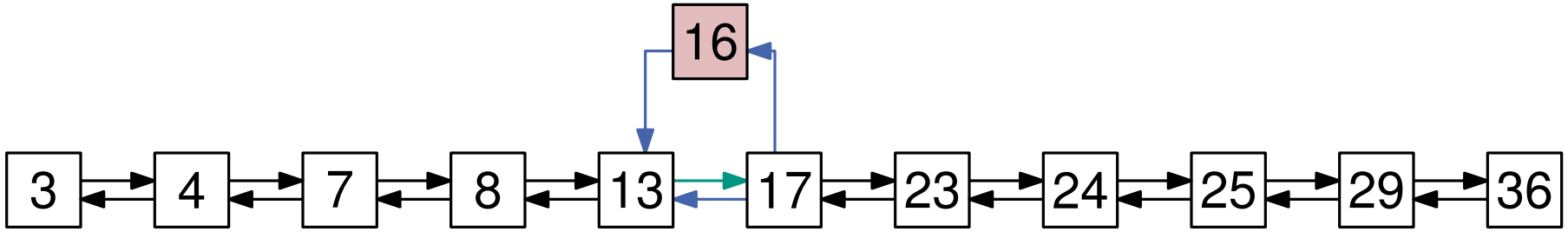
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen



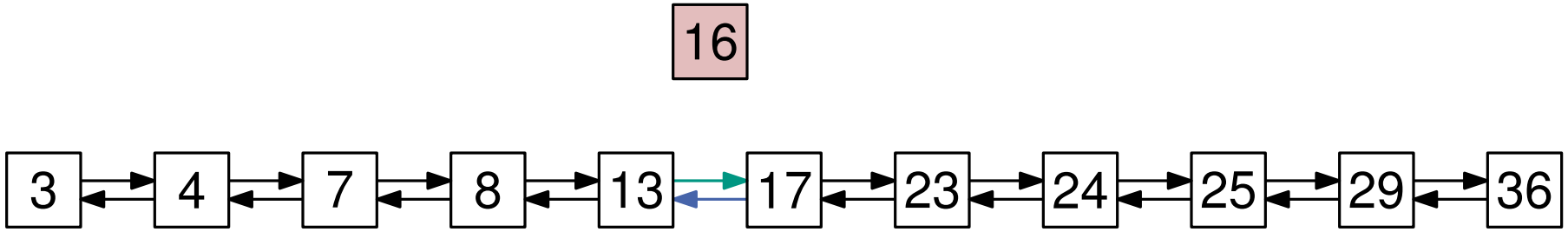
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen



Listen und Binäre Suche

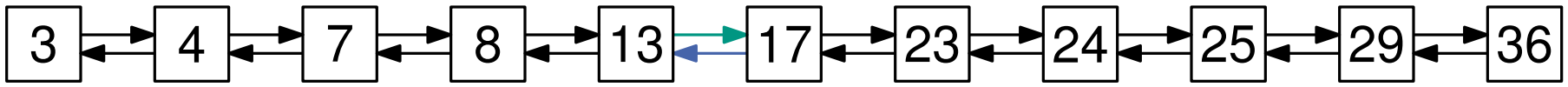
- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen
  - das erfordert Zugang zum zu löschenden Element



Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen
  - das erfordert Zugang zum zu löschenden Element

In Listen können wir schnell umsortieren, wenn wir die Elemente schon in der Hand haben.

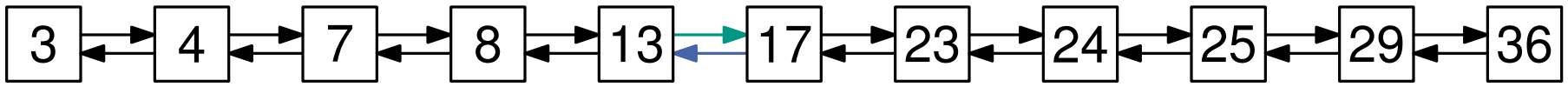


Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen
  - das erfordert Zugang zum zu löschenden Element

In Listen können wir schnell umsortieren, wenn wir die Elemente schon in der Hand haben.

Wie können wir in Listen **schnell** suchen?



Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen
  - das erfordert Zugang zum zu löschenden Element

In Listen können wir schnell umsortieren, wenn wir die Elemente schon in der Hand haben.

Wie können wir in Listen **schnell** suchen?

- Finde Elemente mit linearer Suche. Beispiel: Finde 13.

Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen
  - das erfordert Zugang zum zu löschenden Element

In Listen können wir schnell umsortieren, wenn wir die Elemente schon in der Hand haben.

Wie können wir in Listen **schnell** suchen?

- Finde Elemente mit linearer Suche. Beispiel: Finde 13.

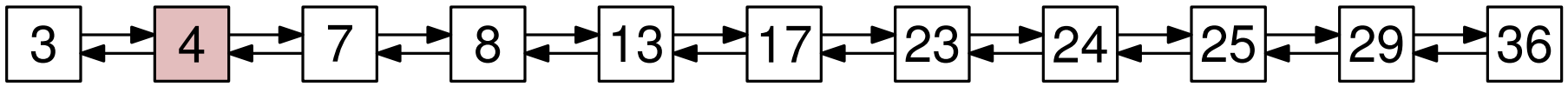
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen
  - das erfordert Zugang zum zu löschenden Element

In Listen können wir schnell umsortieren, wenn wir die Elemente schon in der Hand haben.

Wie können wir in Listen **schnell** suchen?

- Finde Elemente mit linearer Suche. Beispiel: Finde 13.



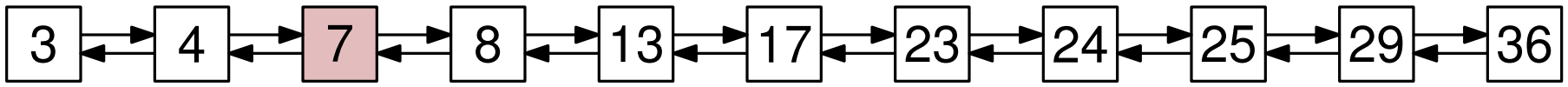
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen
  - das erfordert Zugang zum zu löschenden Element

In Listen können wir schnell umsortieren, wenn wir die Elemente schon in der Hand haben.

Wie können wir in Listen **schnell** suchen?

- Finde Elemente mit linearer Suche. Beispiel: Finde 13.



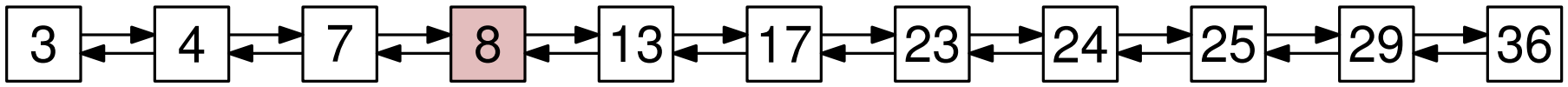
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen
  - das erfordert Zugang zum zu löschenden Element

In Listen können wir schnell umsortieren, wenn wir die Elemente schon in der Hand haben.

Wie können wir in Listen **schnell** suchen?

- Finde Elemente mit linearer Suche. Beispiel: Finde 13.



Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen
  - das erfordert Zugang zum zu löschenden Element

In Listen können wir schnell umsortieren, wenn wir die Elemente schon in der Hand haben.

Wie können wir in Listen **schnell** suchen?

- Finde Elemente mit linearer Suche. Beispiel: Finde 13.

Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Listen erlauben schnelles Einfügen
  - das erfordert Zugang zum Vorgänger Element
  - Listen erlauben einfaches Löschen
  - das erfordert Zugang zum zu löschenden Element

In Listen können wir schnell umsortieren, wenn wir die Elemente schon in der Hand haben.

Wie können wir in Listen **schnell** suchen?

- Finde Elemente mit linearer Suche. Beispiel: Finde 13.

$O(n)$

Listen und Binäre Suche

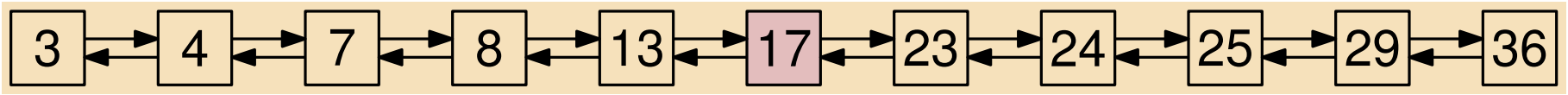
- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8

Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich

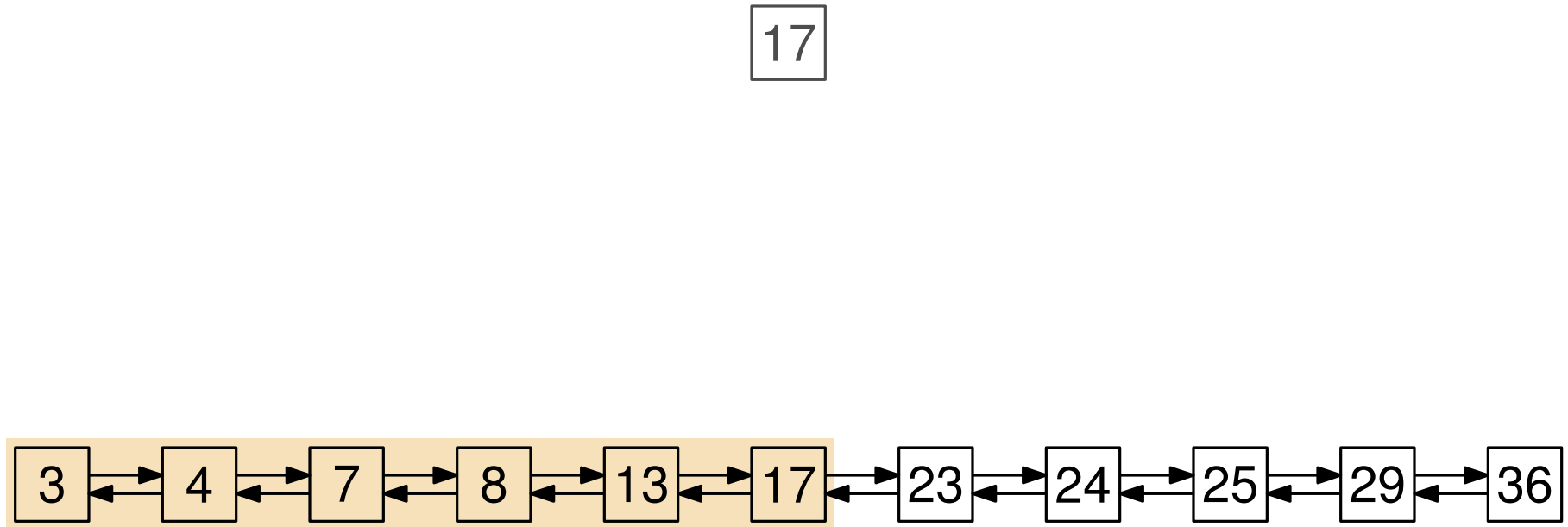
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element



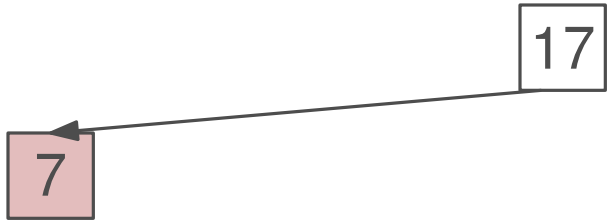
Listen und Binäre Suche

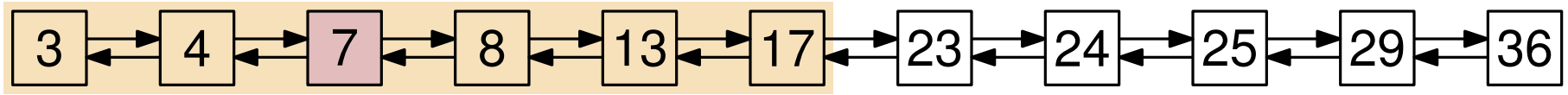
- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element



Listen und Binäre Suche

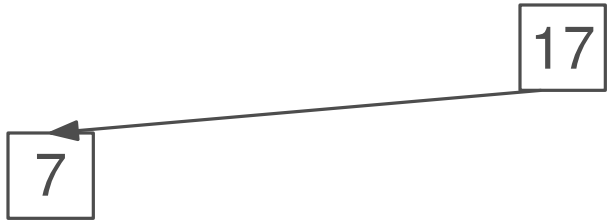
- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element

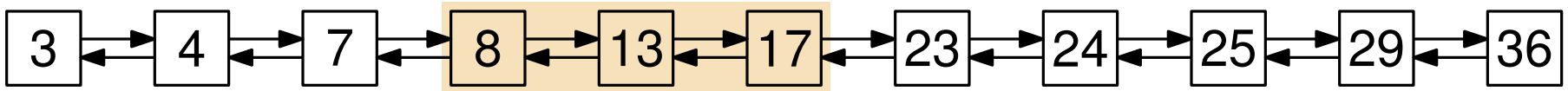




Listen und Binäre Suche

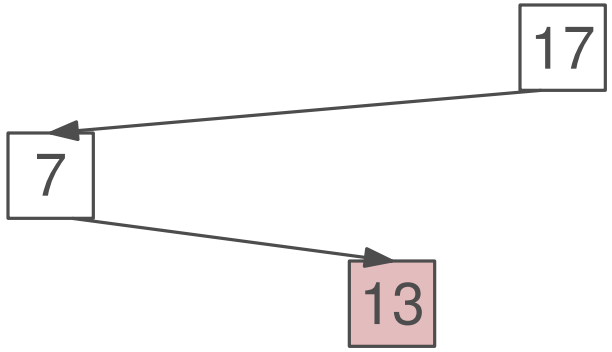
- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element

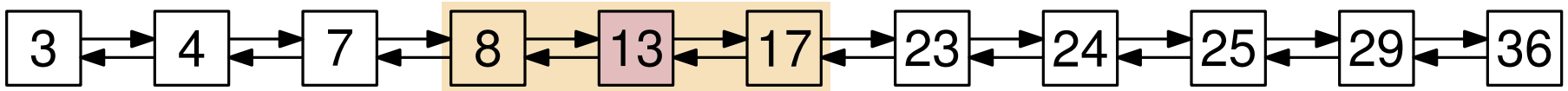




Listen und Binäre Suche

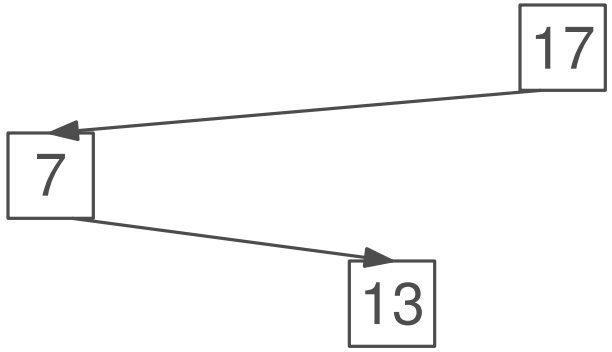
- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element

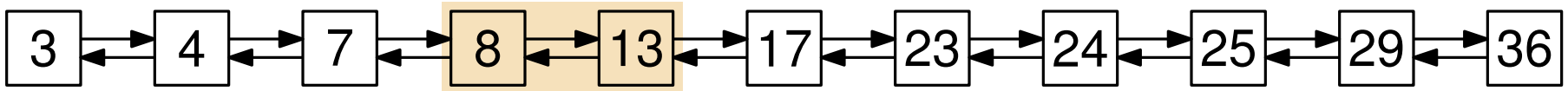




Listen und Binäre Suche

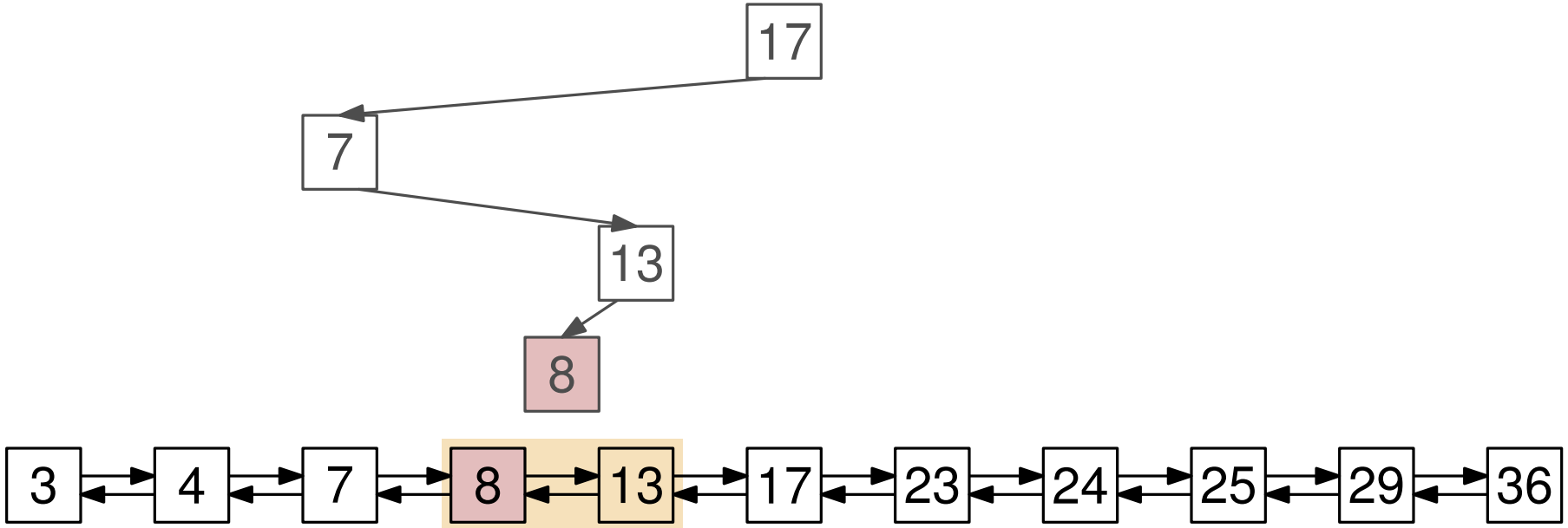
- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element





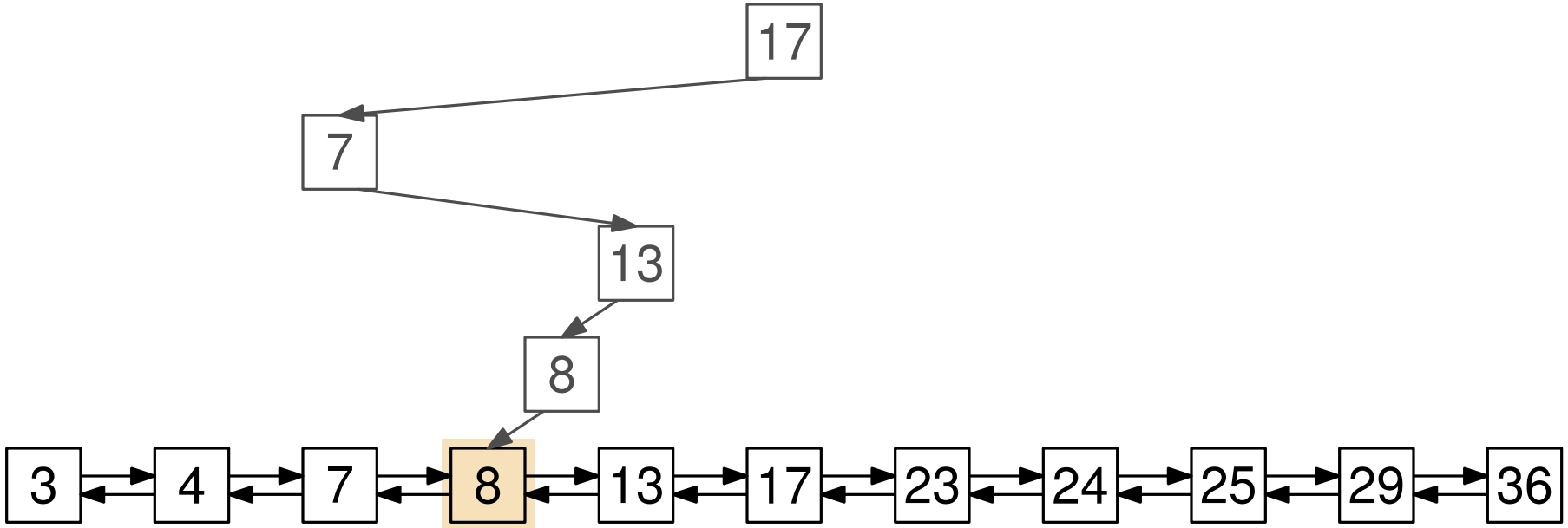
Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element



Listen und Binäre Suche

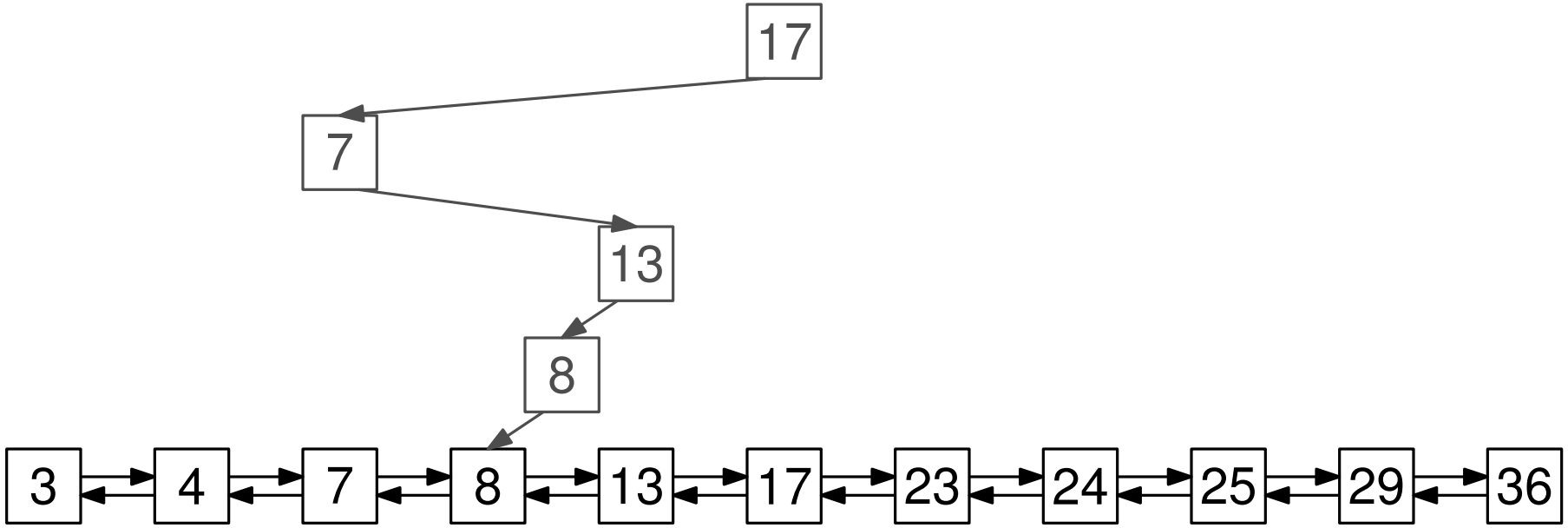
- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element



Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element

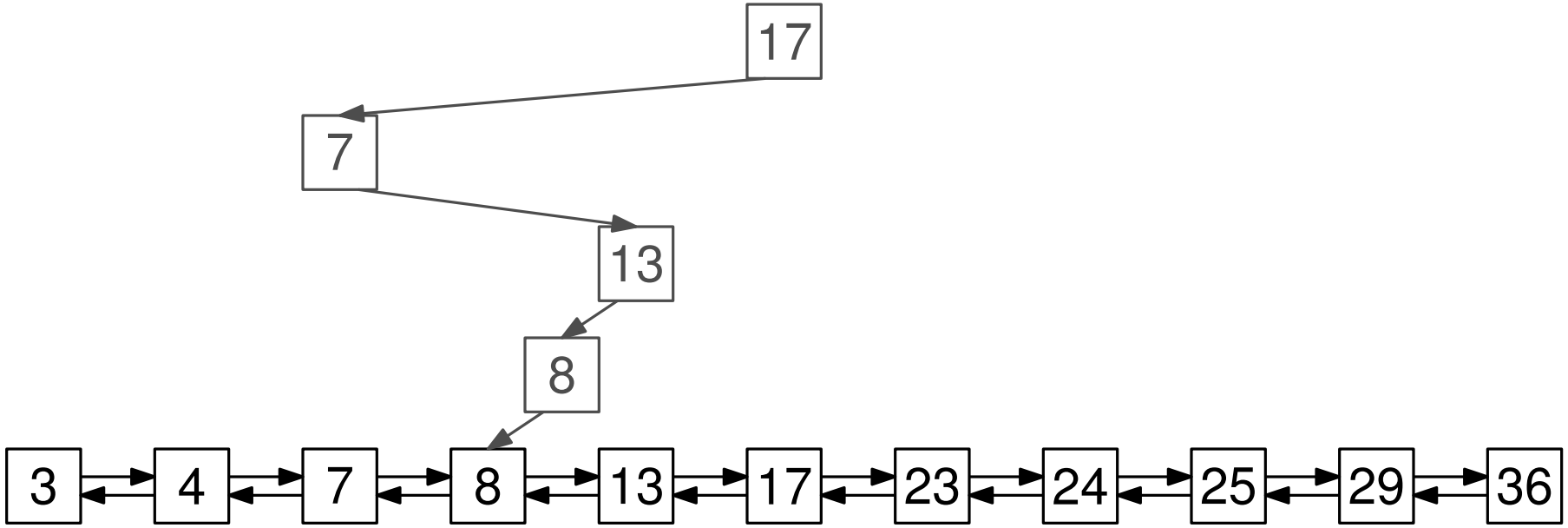
Größe des Suchbereichs halbiert sich in jedem Schritt $\rightarrow O(\log(n))$



Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element

Größe des Suchbereichs halbiert sich in jedem Schritt $\rightarrow O(\log(n))$

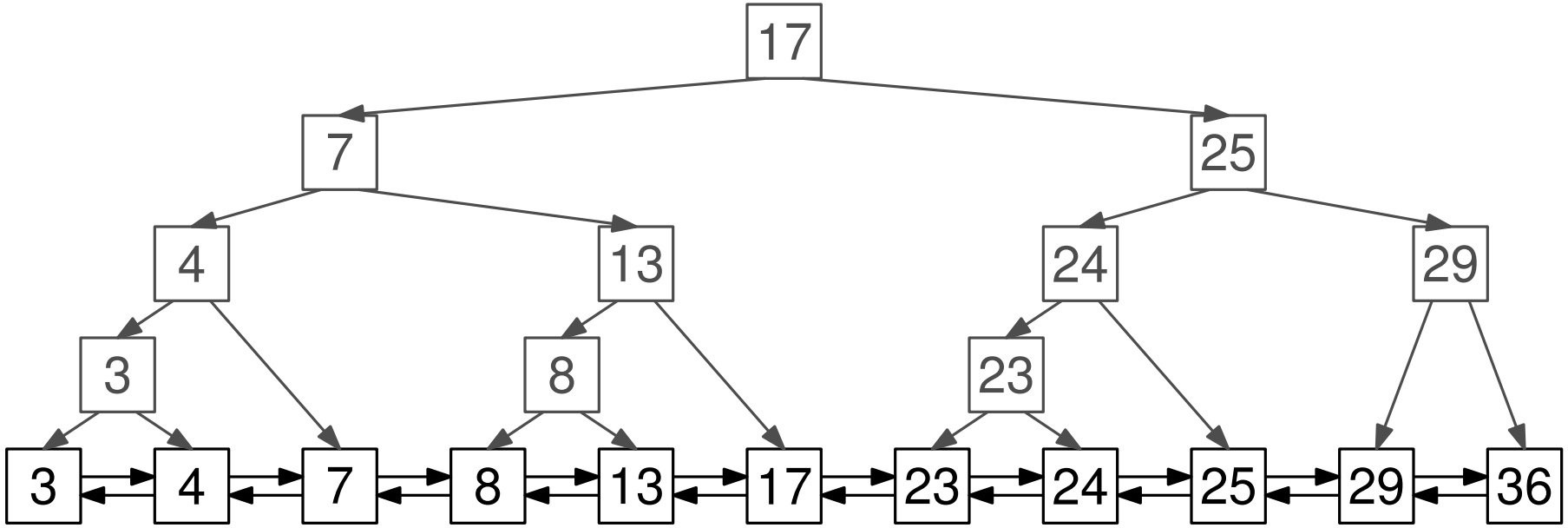


  - Idee: Speichere alle möglichen Suchen als Baum ab

Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element

Größe des Suchbereichs halbiert sich in jedem Schritt $\rightarrow O(\log(n))$

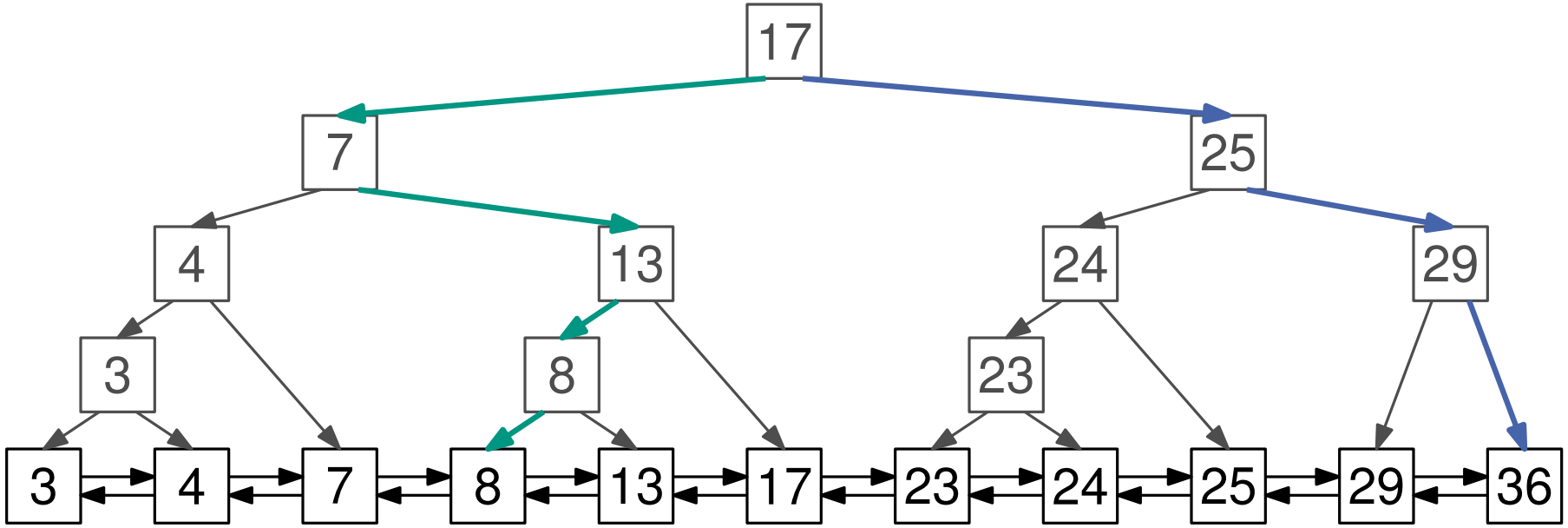


  - Idee: Speichere alle möglichen Suchen als Baum ab

Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element

Größe des Suchbereichs halbiert sich in jedem Schritt $\rightarrow O(\log(n))$

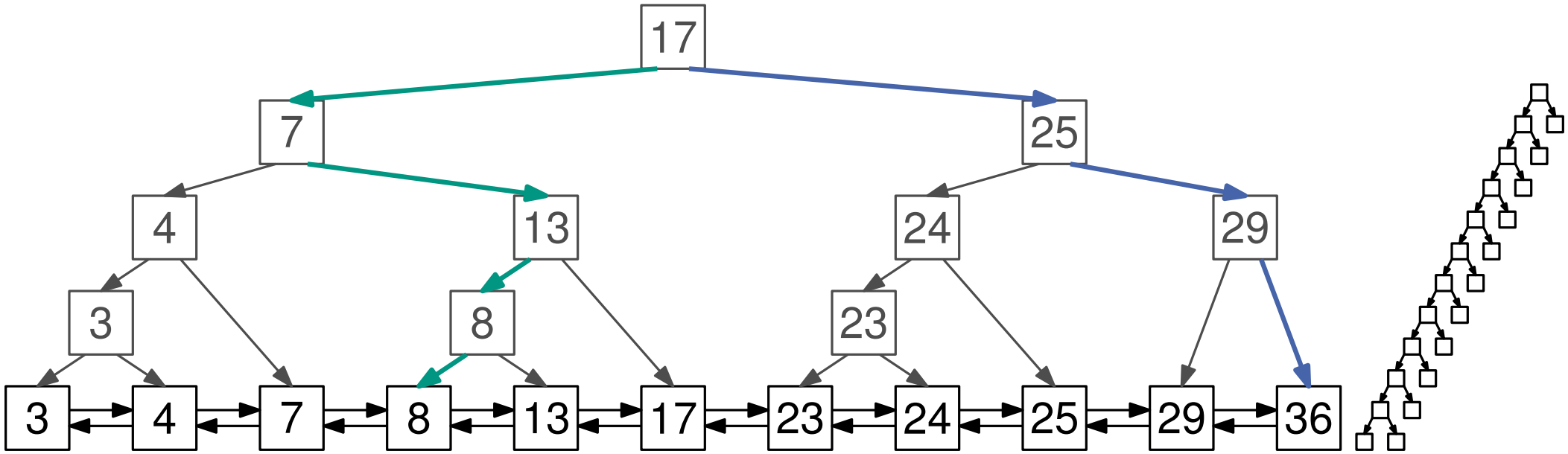


  - Idee: Speichere alle möglichen Suchen als Baum ab
  - Problem(?): Blätter hängen in unterschiedlicher Tiefe

Listen und Binäre Suche

- In Suchbäumen wollen wir die Vorteile von Listen und der binären Suche vereinen
  - Speed-Up der Suche durch Simulation von binärer Suche wie in Arrays
  - Beispiel: Binäre Suche nach 8 Aktueller Suchbereich Mittleres Element

Größe des Suchbereichs halbiert sich in jedem Schritt $\rightarrow O(\log(n))$



- Idee: Speichere alle möglichen Suchen als Baum ab
- Problem(?): Blätter hängen in unterschiedlicher Tiefe

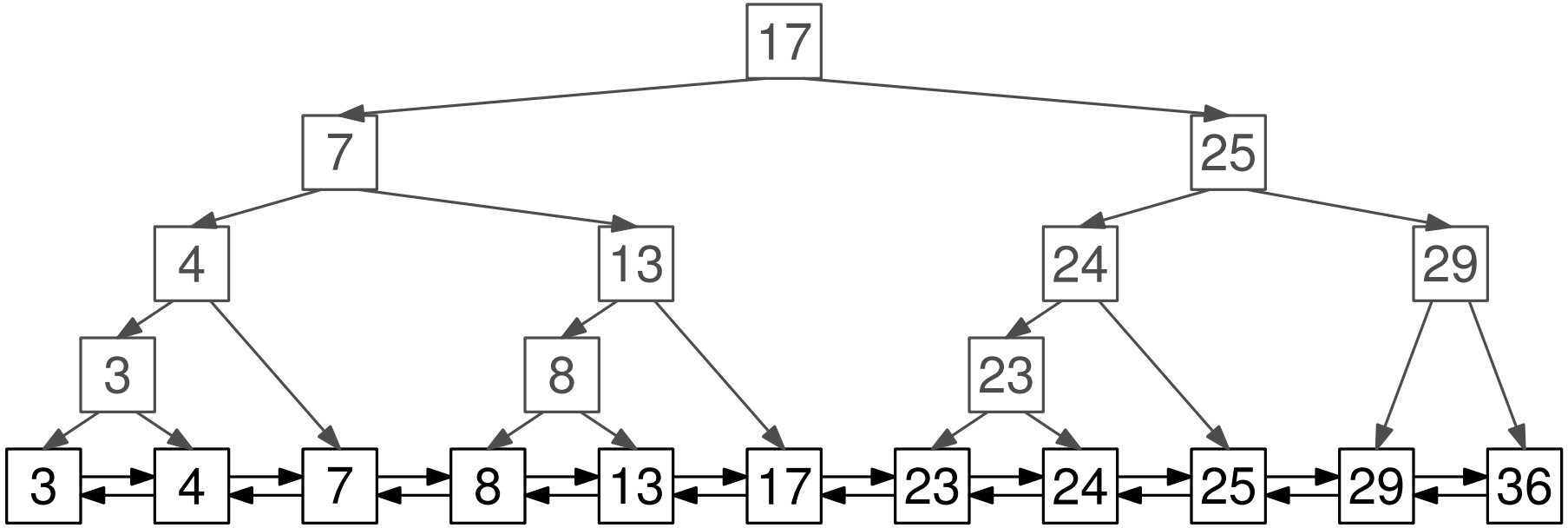
Kann das beliebig schlimm werden?

Nicht beim initialen Aufbau...

... aber beim Einfügen / Löschen?

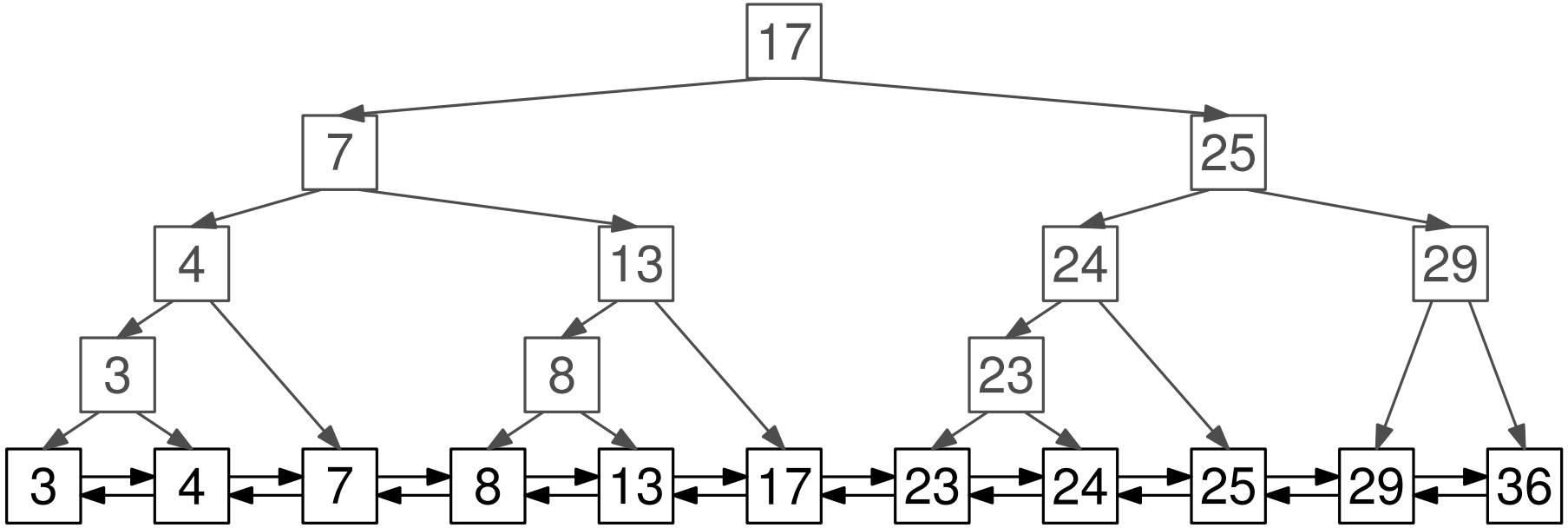
(2, 3)-Bäume

- Idee: **fast** binär und dafür balanciert
  - jeder Knoten hat 2 oder 3 Kinder



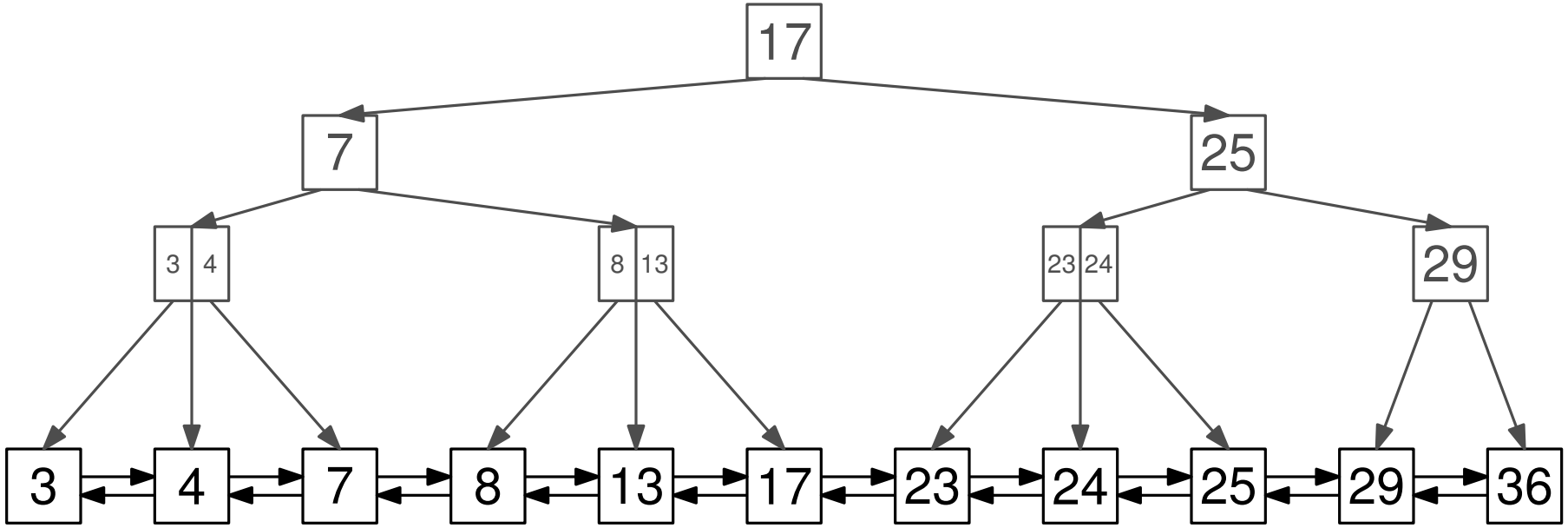
(2, 3)-Bäume

- Idee: **fast** binär und dafür balanciert
  - jeder Knoten hat 2 oder 3 Kinder
  - dadurch können wir dafür sorgen, dass alle Blätter die gleiche Tiefe haben



(2, 3)-Bäume

- Idee: **fast** binär und dafür balanciert
  - jeder Knoten hat 2 oder 3 Kinder
  - dadurch können wir dafür sorgen, dass alle Blätter die gleiche Tiefe haben

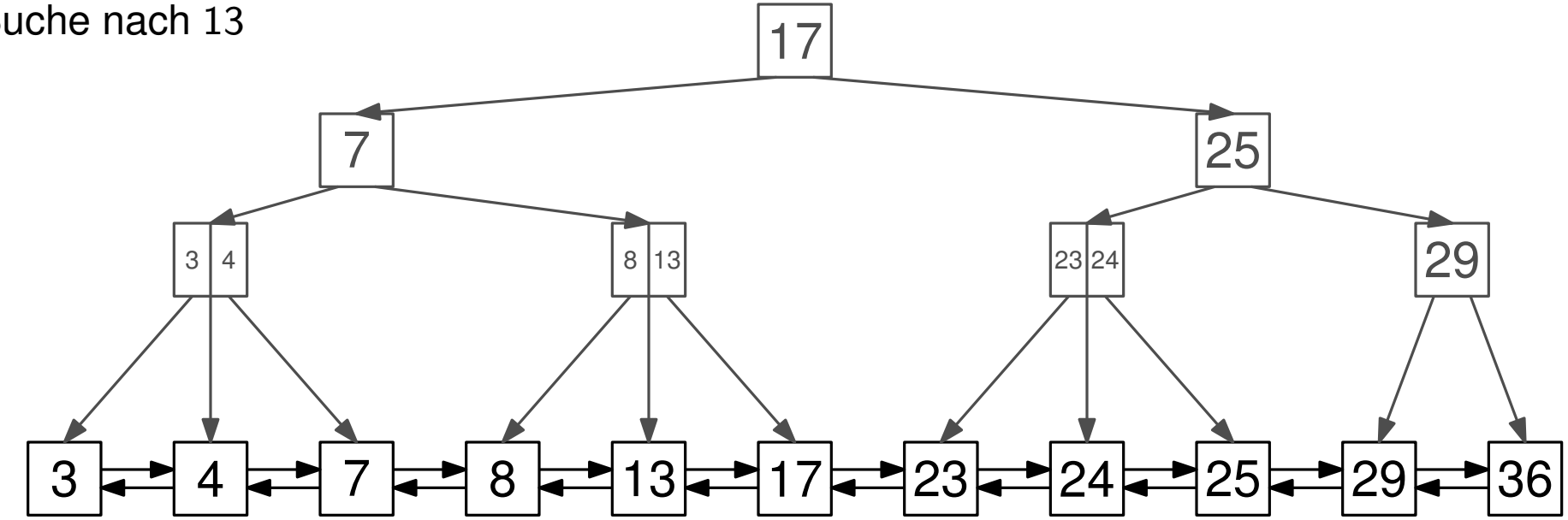


- Knoten halten ggf. zwei Schlüssel k_1, k_2. Abstieg: Links ($x \leq k_1$), Mitte ($k_1 < x \leq k_2$), Rechts ($k_2 < x$)

(2, 3)-Bäume

- Idee: **fast** binär und dafür balanciert
  - jeder Knoten hat 2 oder 3 Kinder
  - dadurch können wir dafür sorgen, dass alle Blätter die gleiche Tiefe haben

Beispiel: Suche nach 13

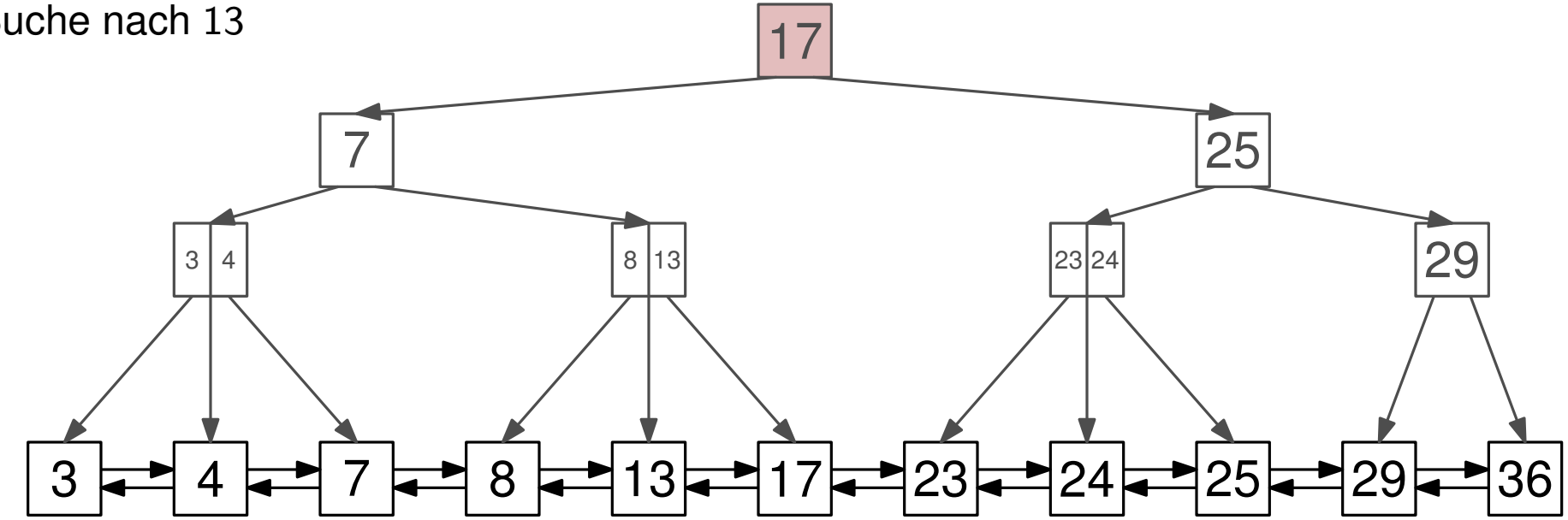


- Knoten halten ggf. zwei Schlüssel k_1, k_2. Abstieg: Links ($x \leq k_1$), Mitte ($k_1 < x \leq k_2$), Rechts ($k_2 < x$)

(2, 3)-Bäume

- Idee: **fast** binär und dafür balanciert
  - jeder Knoten hat 2 oder 3 Kinder
  - dadurch können wir dafür sorgen, dass alle Blätter die gleiche Tiefe haben

Beispiel: Suche nach 13

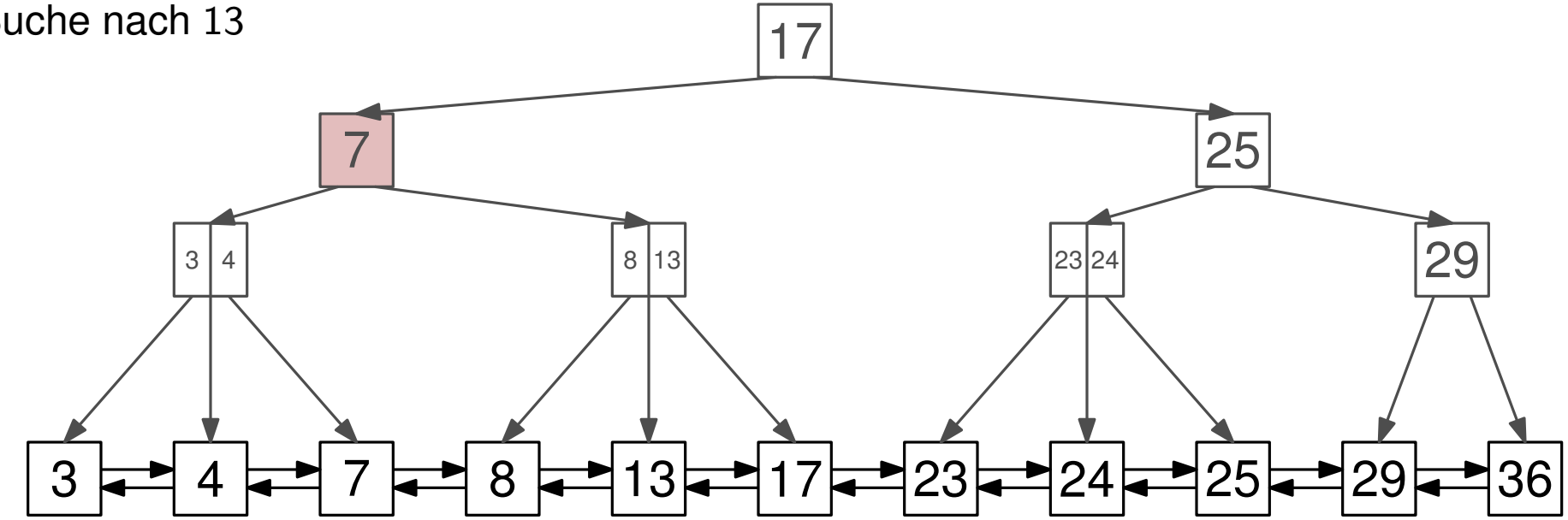


- Knoten halten ggf. zwei Schlüssel k_1, k_2. Abstieg: Links ($x \leq k_1$), Mitte ($k_1 < x \leq k_2$), Rechts ($k_2 < x$)

(2, 3)-Bäume

- Idee: **fast** binär und dafür balanciert
  - jeder Knoten hat 2 oder 3 Kinder
  - dadurch können wir dafür sorgen, dass alle Blätter die gleiche Tiefe haben

Beispiel: Suche nach 13

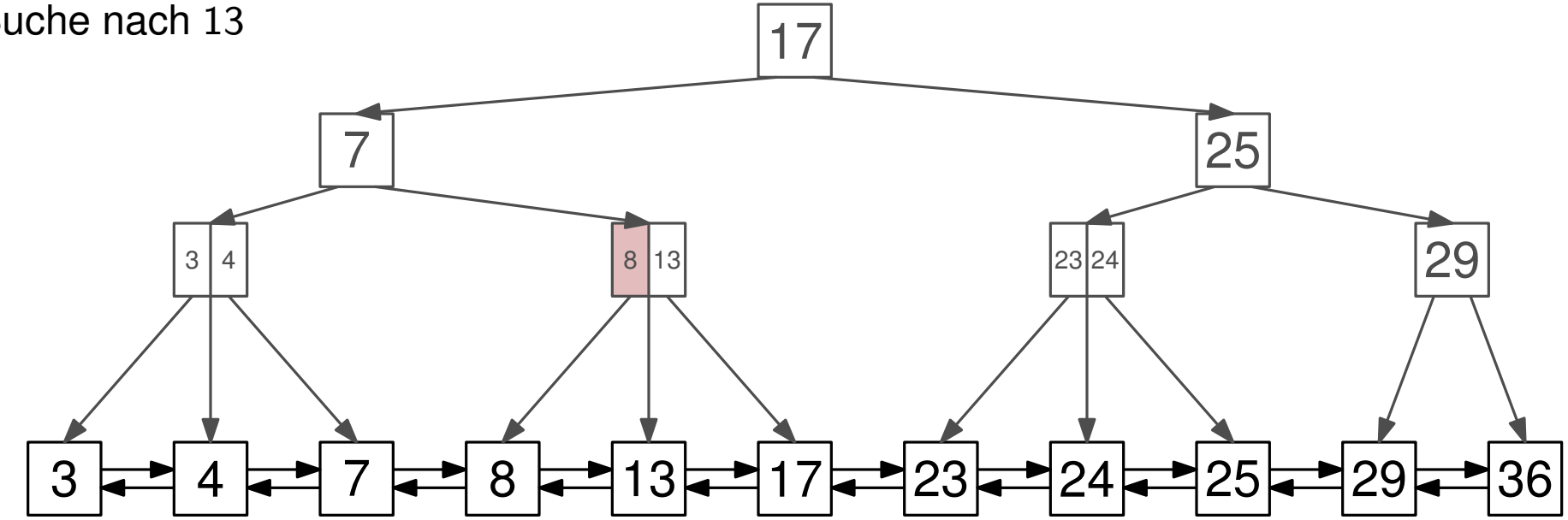


- Knoten halten ggf. zwei Schlüssel k_1, k_2. Abstieg: Links ($x \leq k_1$), Mitte ($k_1 < x \leq k_2$), Rechts ($k_2 < x$)

(2, 3)-Bäume

- Idee: **fast** binär und dafür balanciert
  - jeder Knoten hat 2 oder 3 Kinder
  - dadurch können wir dafür sorgen, dass alle Blätter die gleiche Tiefe haben

Beispiel: Suche nach 13

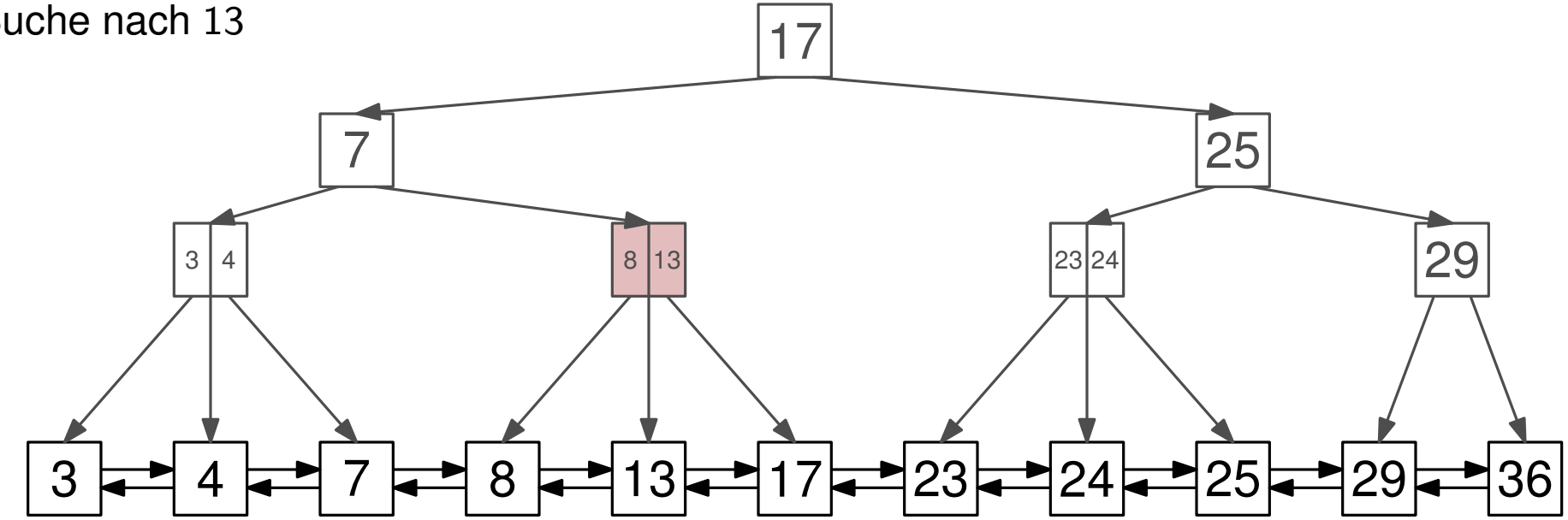


- Knoten halten ggf. zwei Schlüssel k_1, k_2. Abstieg: Links ($x \leq k_1$), Mitte ($k_1 < x \leq k_2$), Rechts ($k_2 < x$)

(2, 3)-Bäume

- Idee: **fast** binär und dafür balanciert
  - jeder Knoten hat 2 oder 3 Kinder
  - dadurch können wir dafür sorgen, dass alle Blätter die gleiche Tiefe haben

Beispiel: Suche nach 13

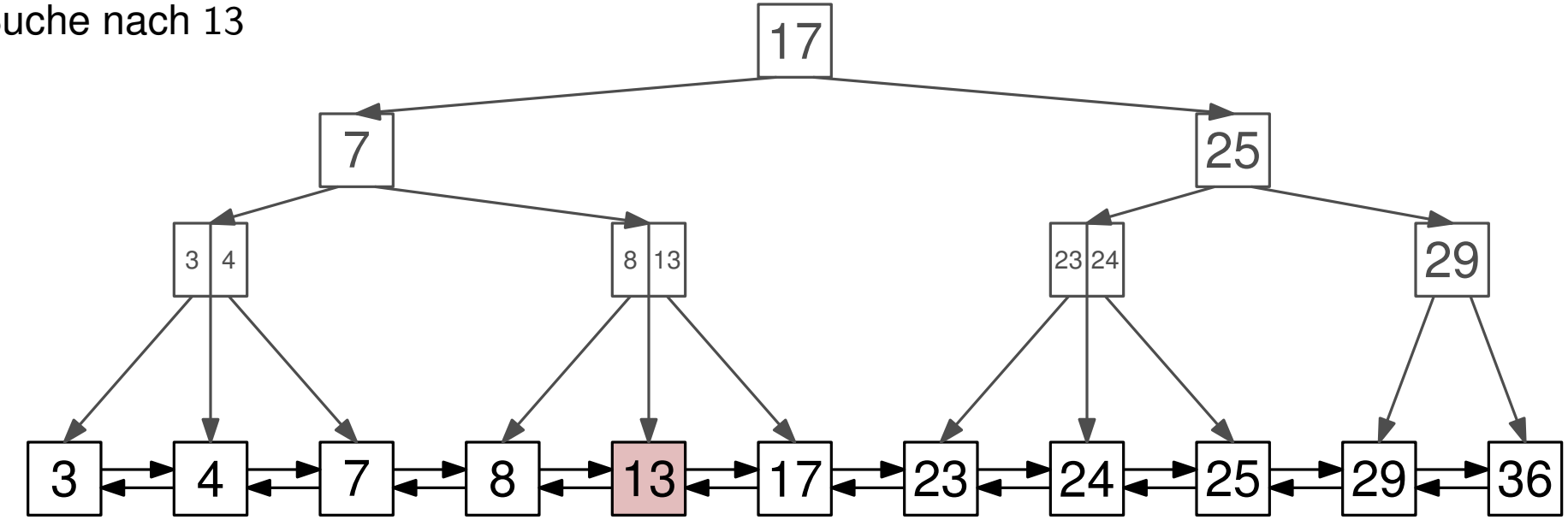


- Knoten halten ggf. zwei Schlüssel k_1, k_2. Abstieg: Links ($x \leq k_1$), Mitte ($k_1 < x \leq k_2$), Rechts ($k_2 < x$)

(2, 3)-Bäume

- Idee: **fast** binär und dafür balanciert
  - jeder Knoten hat 2 oder 3 Kinder
  - dadurch können wir dafür sorgen, dass alle Blätter die gleiche Tiefe haben

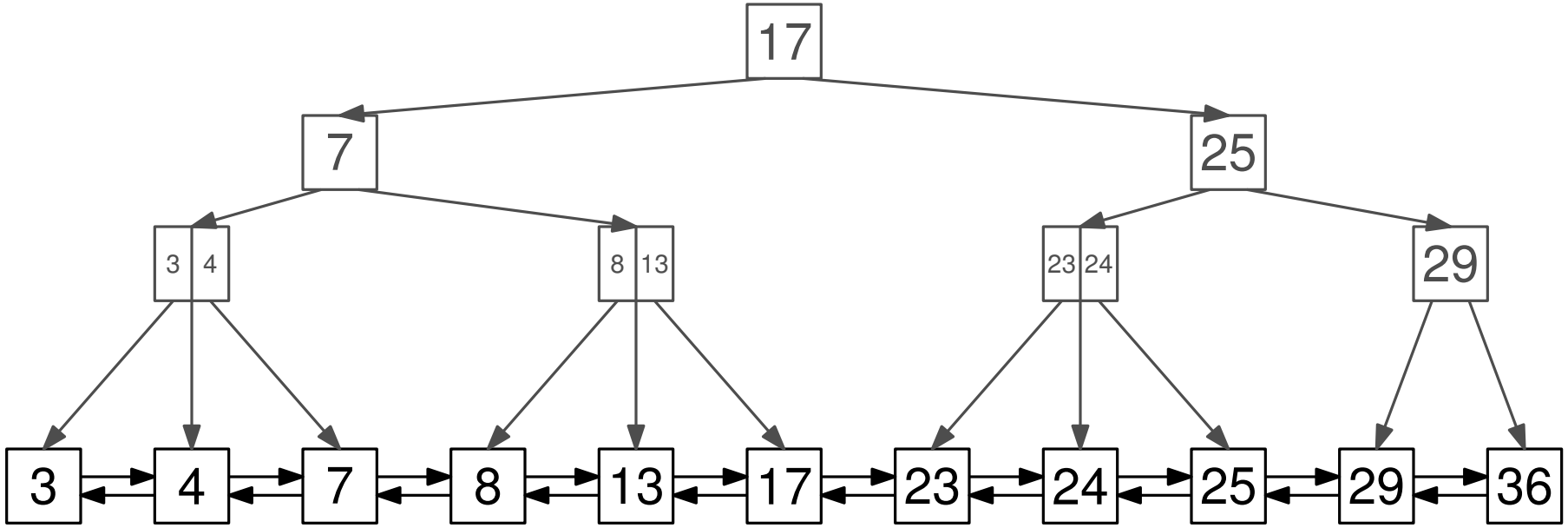
Beispiel: Suche nach 13



- Knoten halten ggf. zwei Schlüssel k_1, k_2. Abstieg: Links ($x \leq k_1$), Mitte ($k_1 < x \leq k_2$), Rechts ($k_2 < x$)

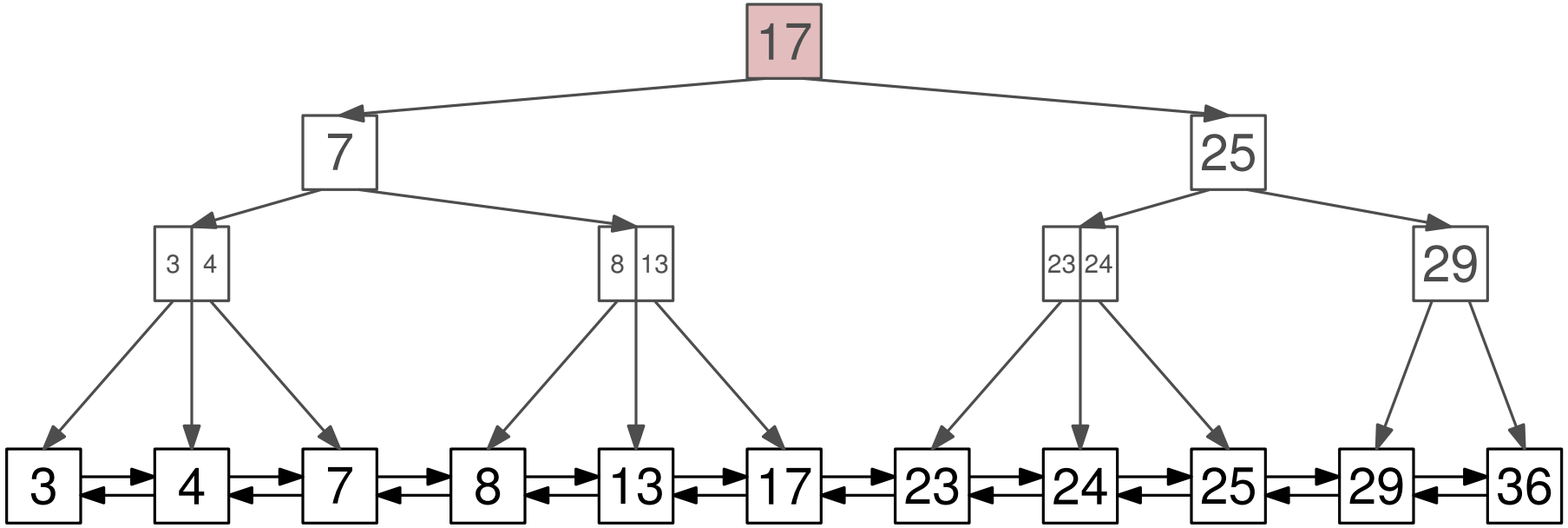
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen



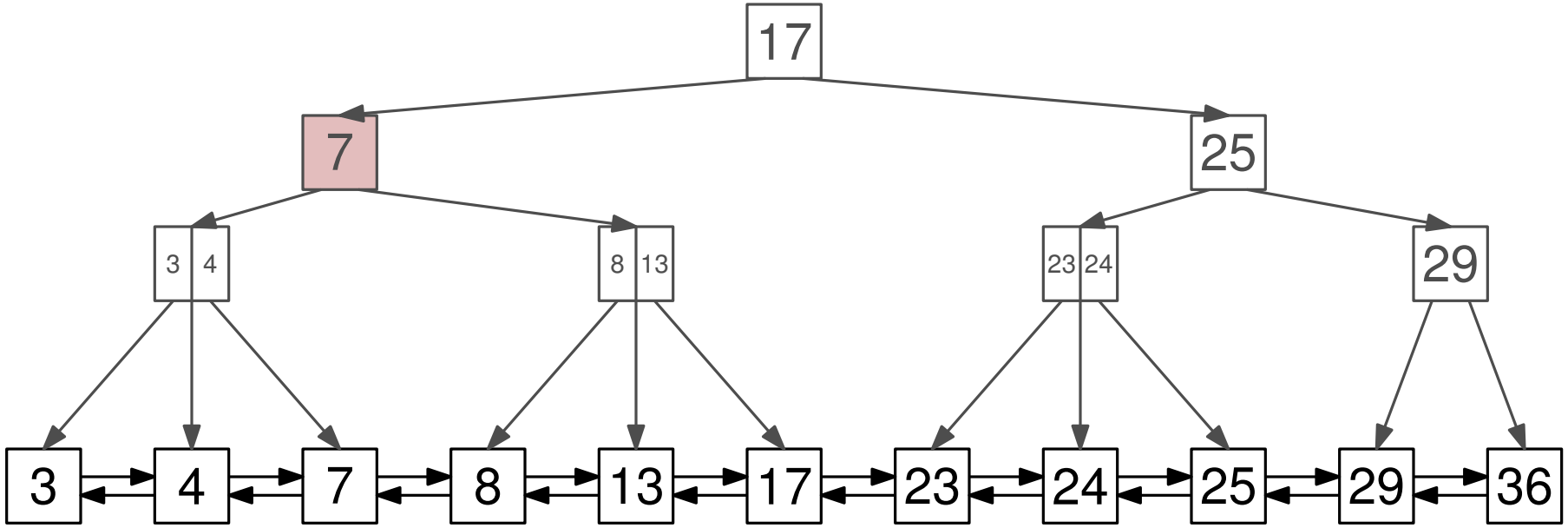
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen



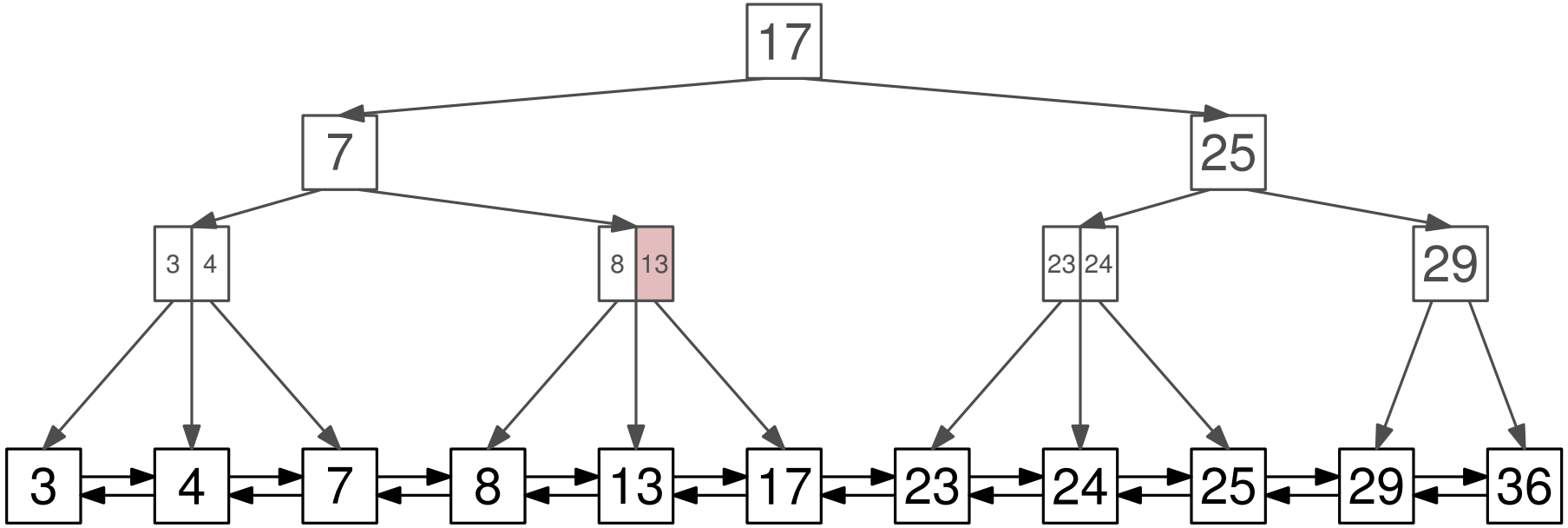
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen



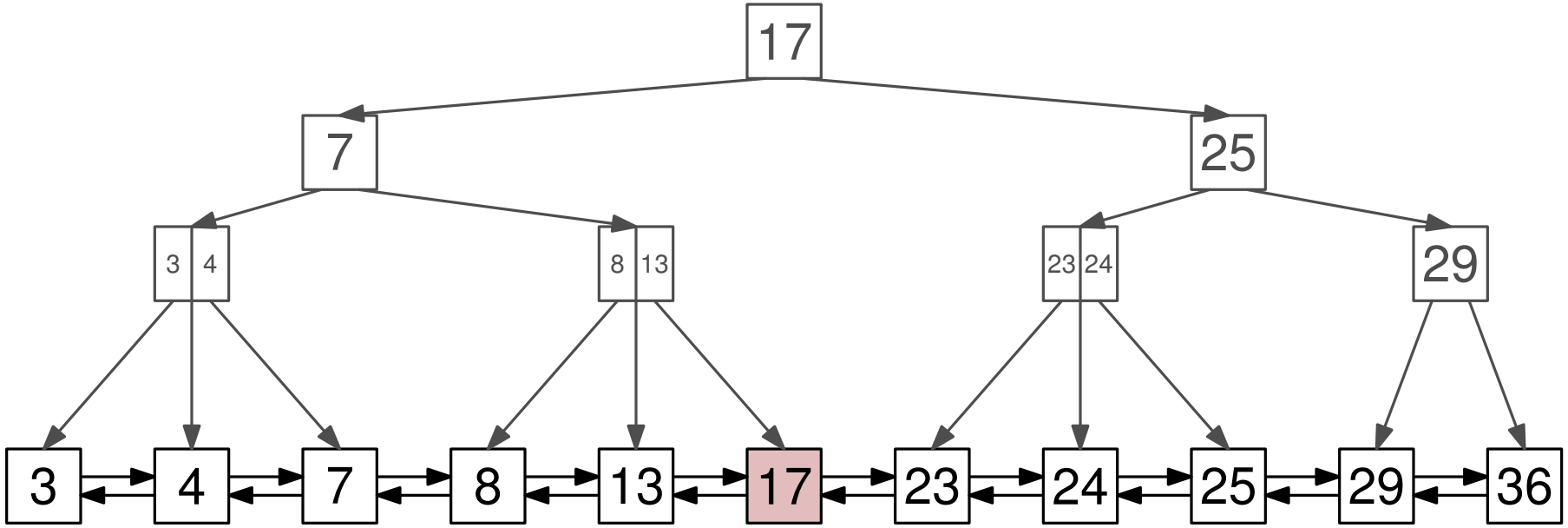
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen



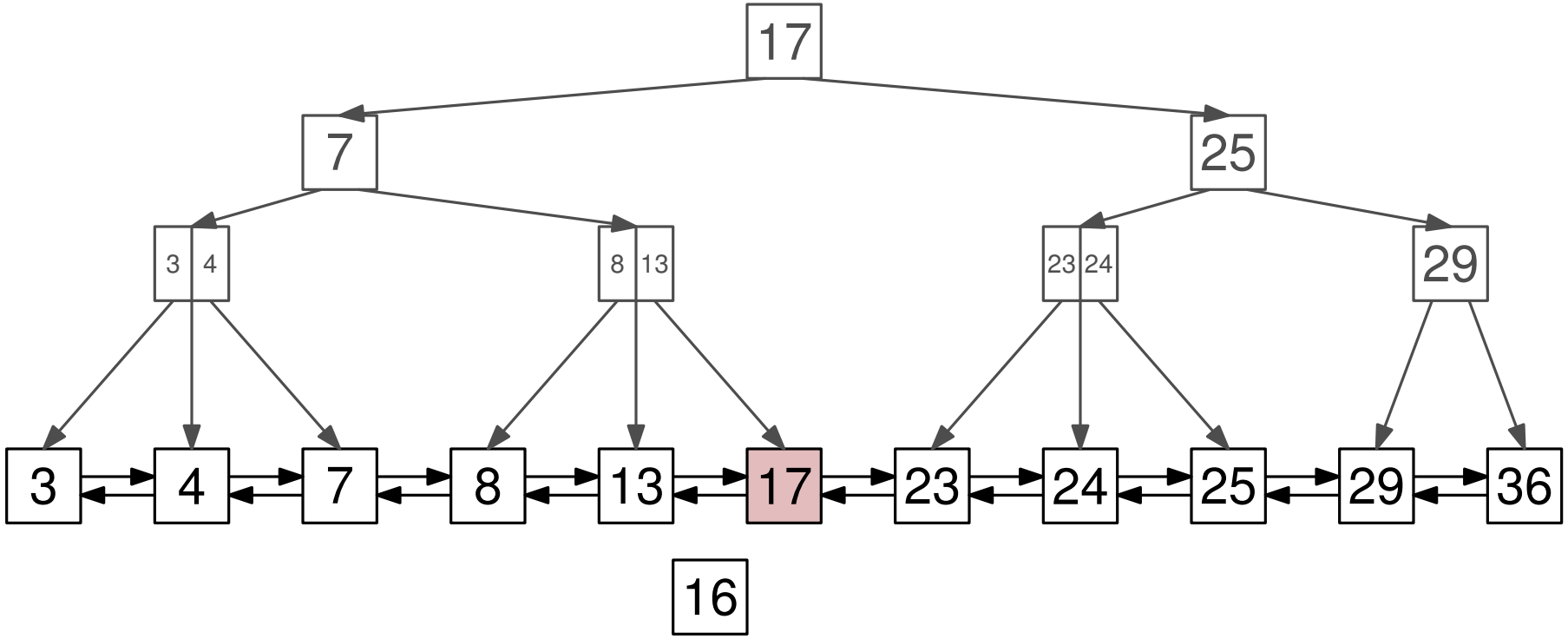
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen



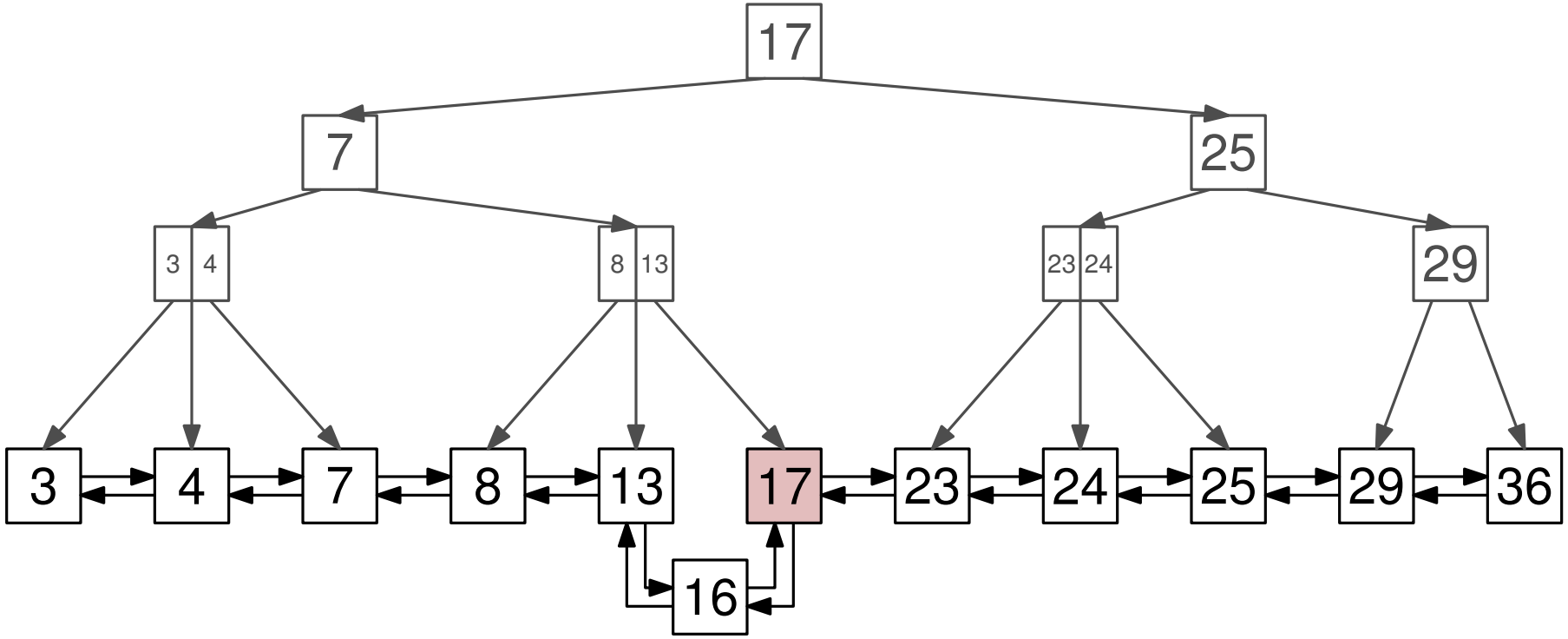
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen



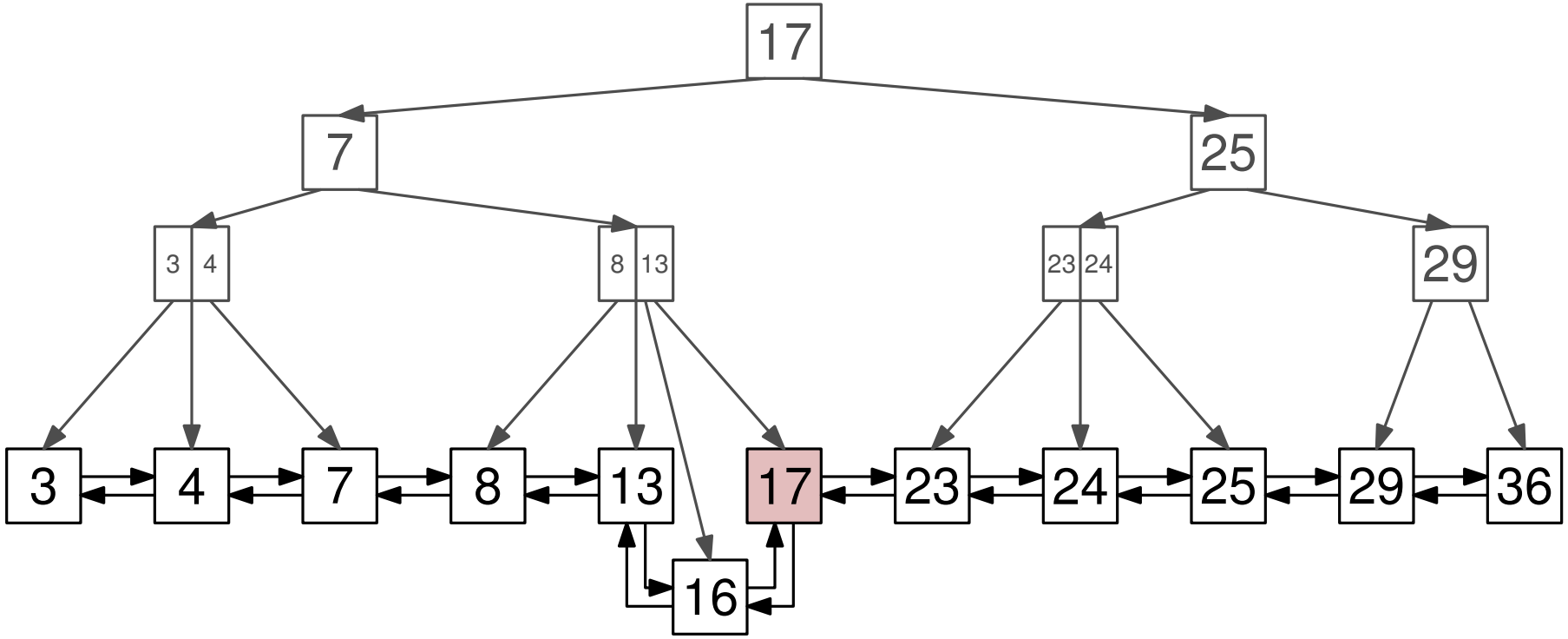
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen



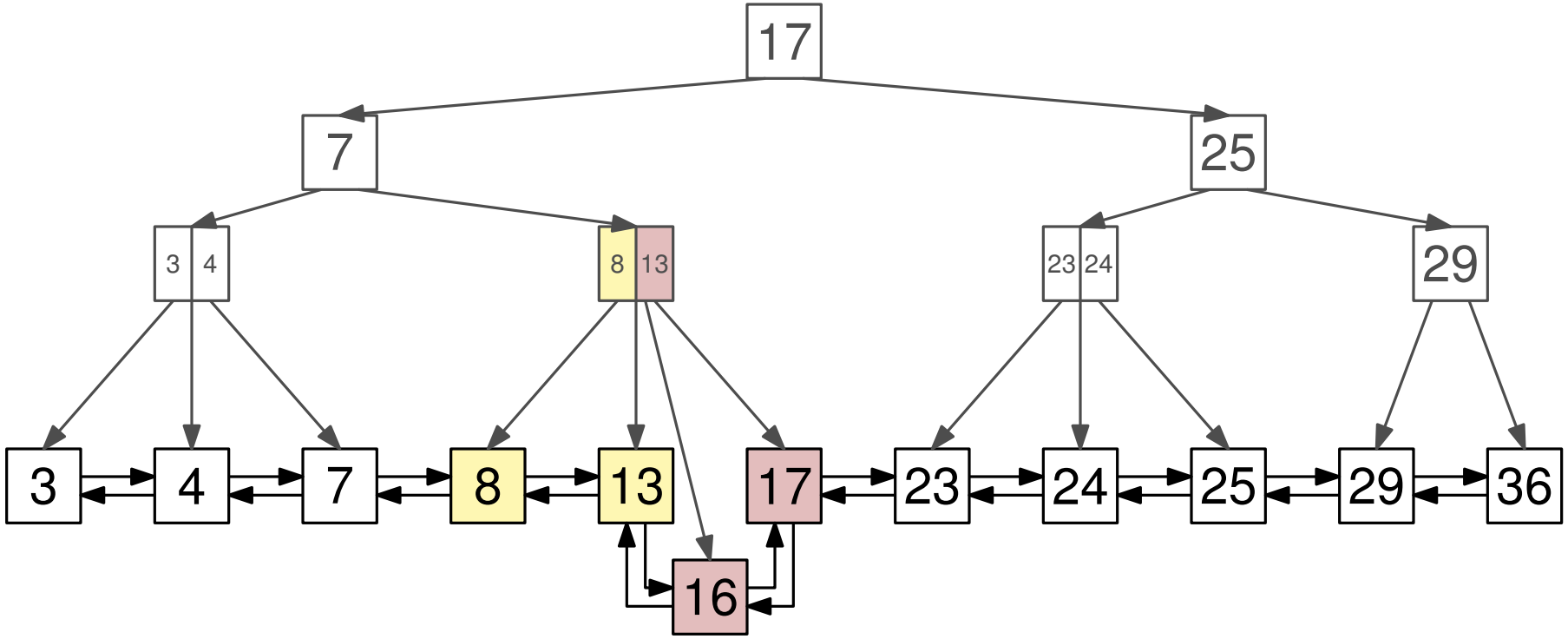
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen



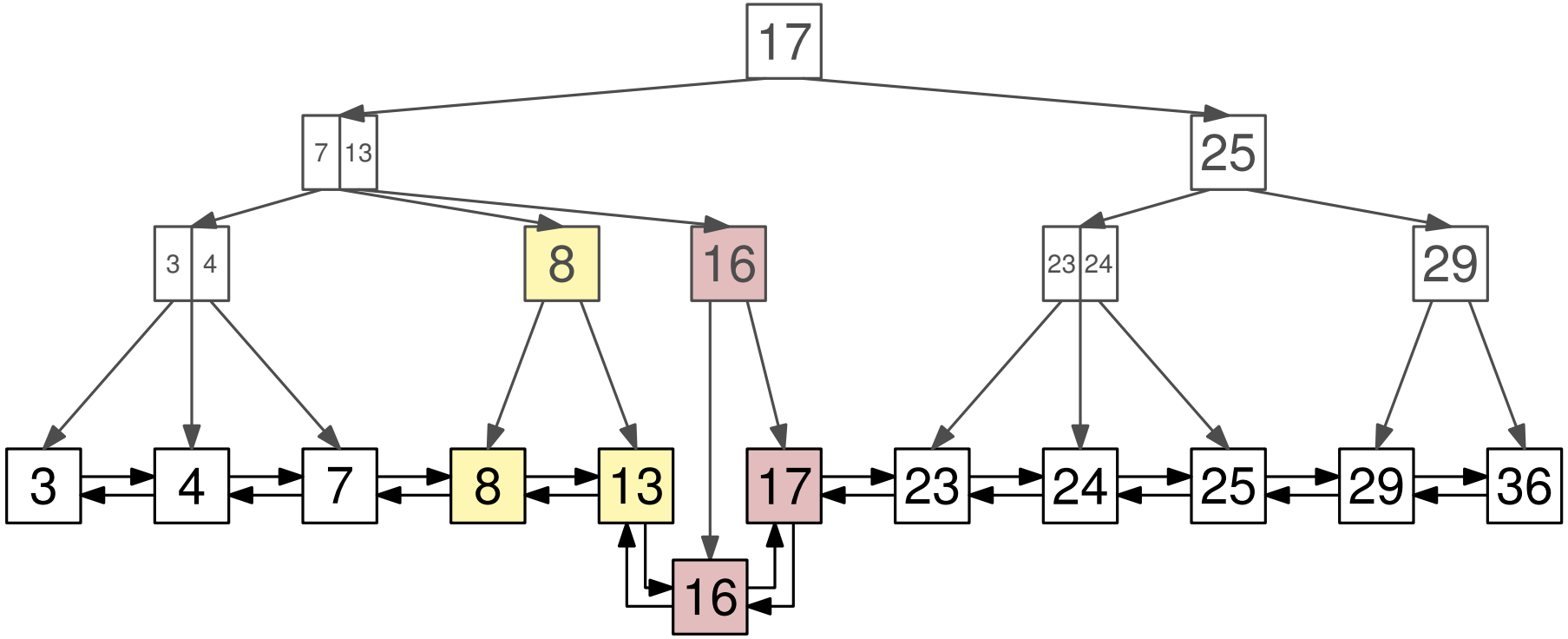
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen



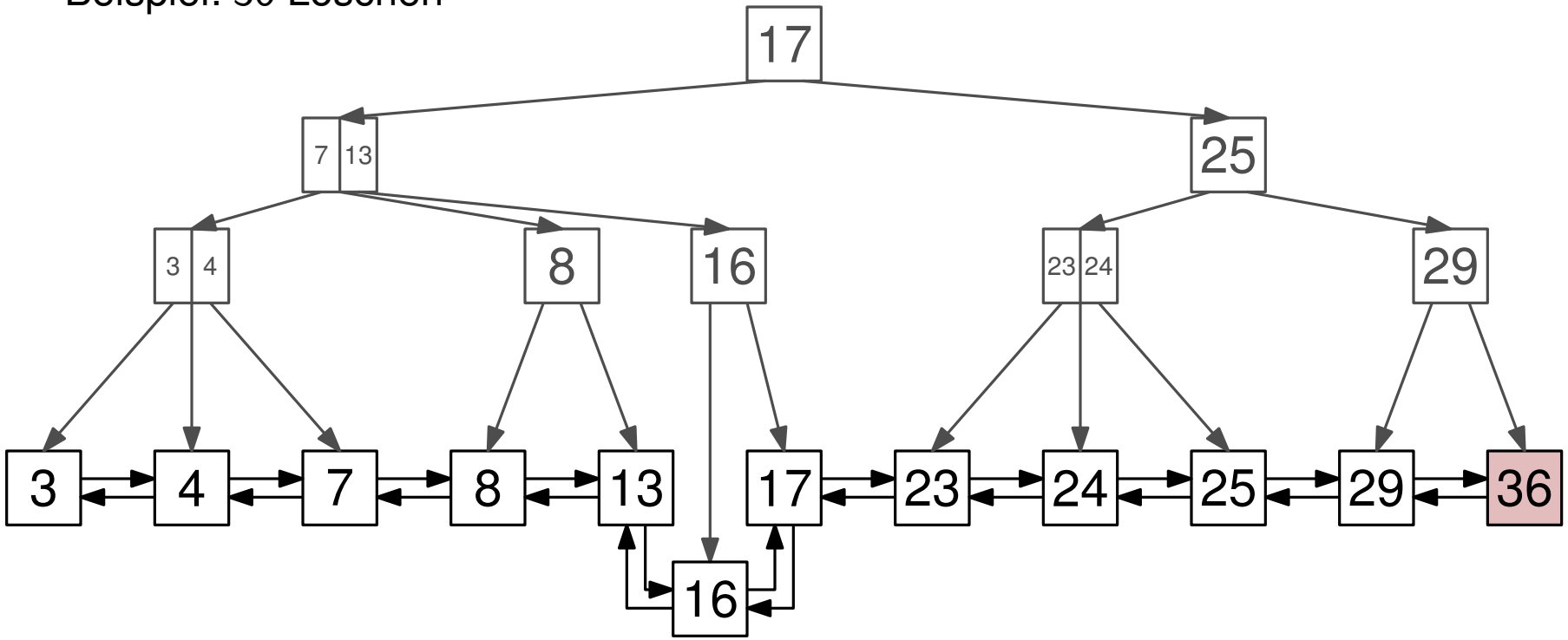
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen



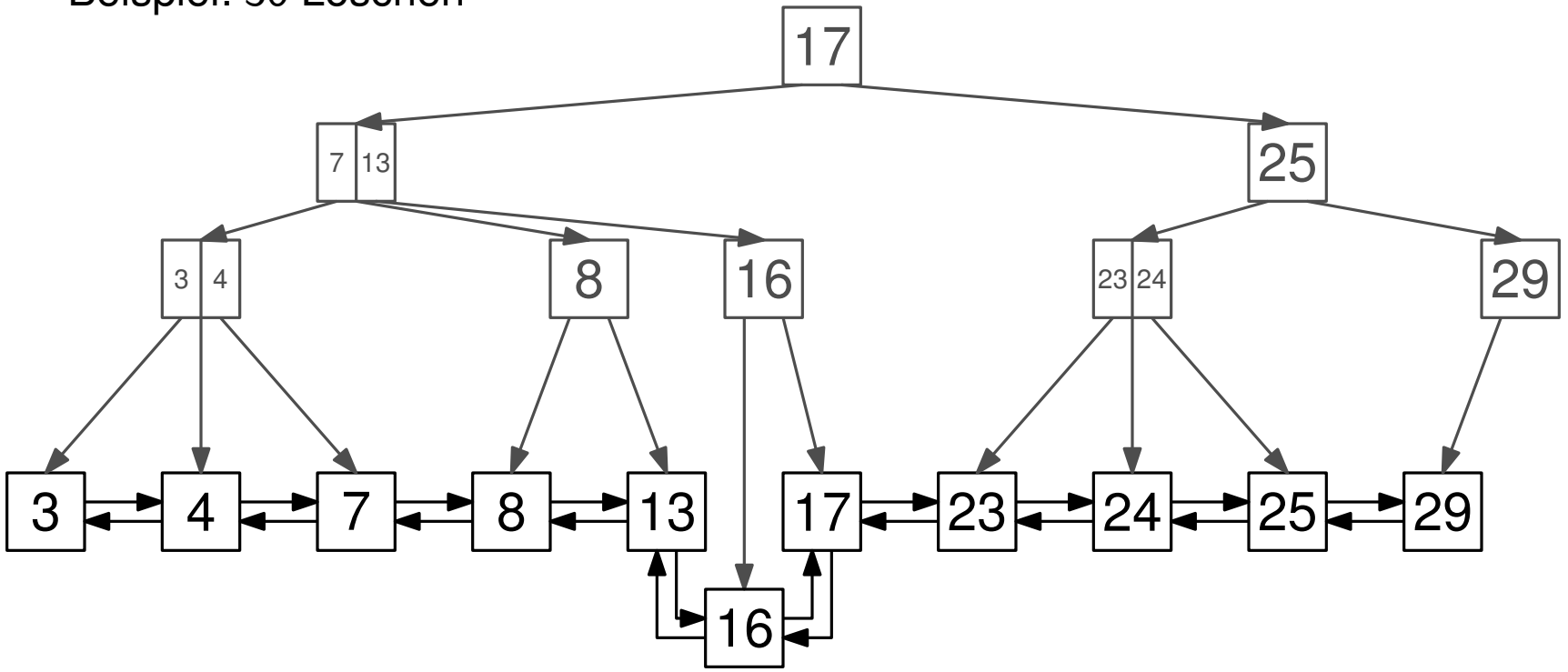
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen
- Löschen: Blatt einfach abschneiden und danach (rekursiv) verschmelzen oder ausbalancieren Beispiel: 36 Löschen



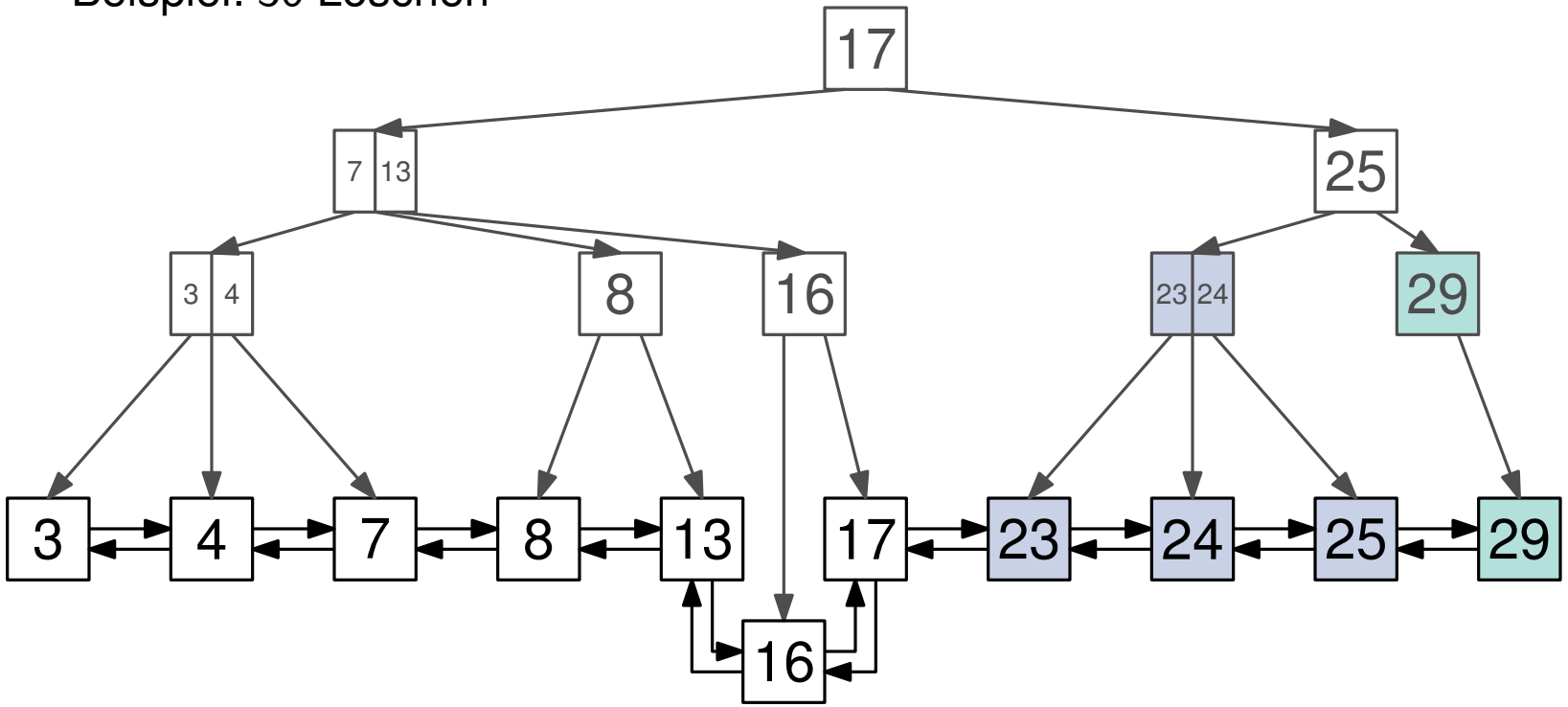
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen
- Löschen: Blatt einfach abschneiden und danach (rekursiv) verschmelzen oder ausbalancieren Beispiel: 36 Löschen



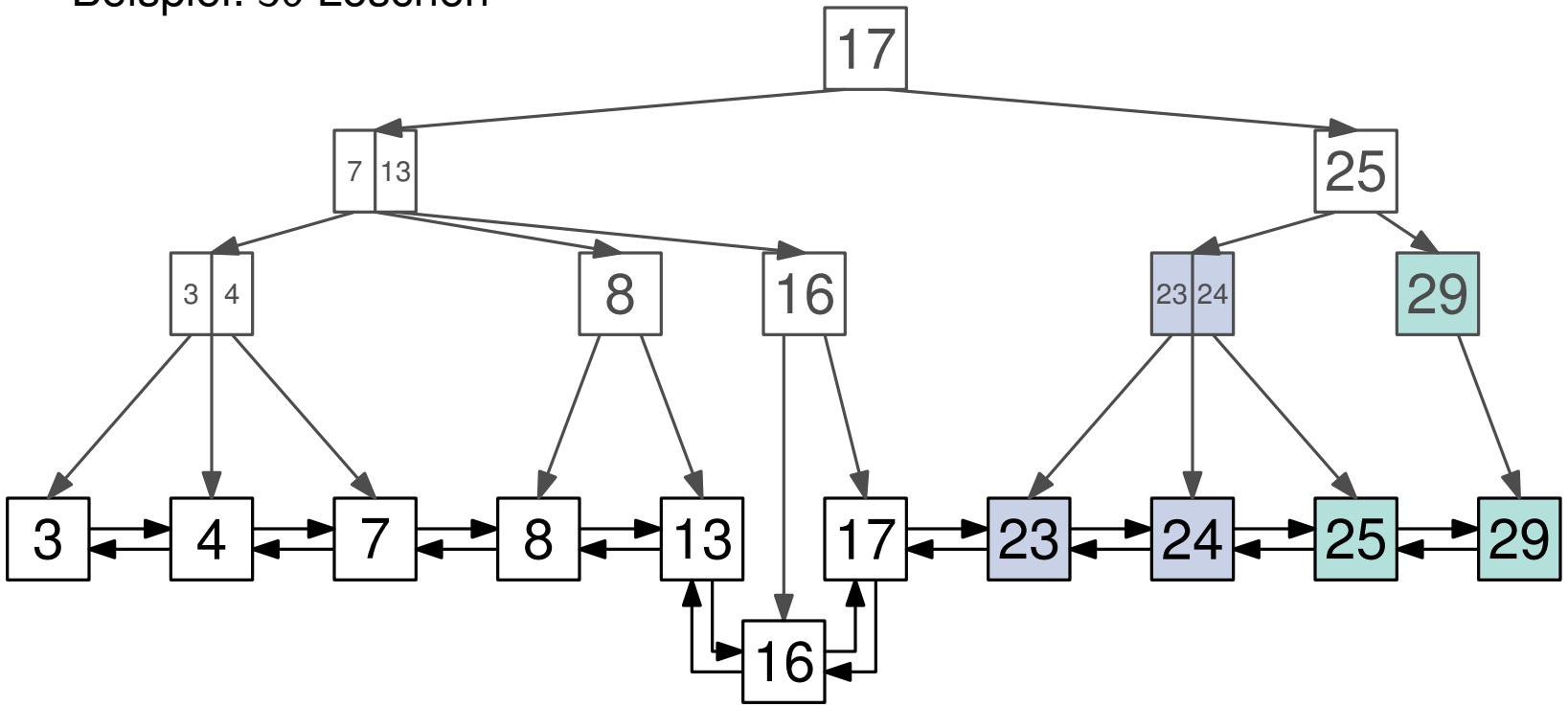
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen
- Löschen: Blatt einfach abschneiden und danach (rekursiv) verschmelzen oder ausbalancieren Beispiel: 36 Löschen



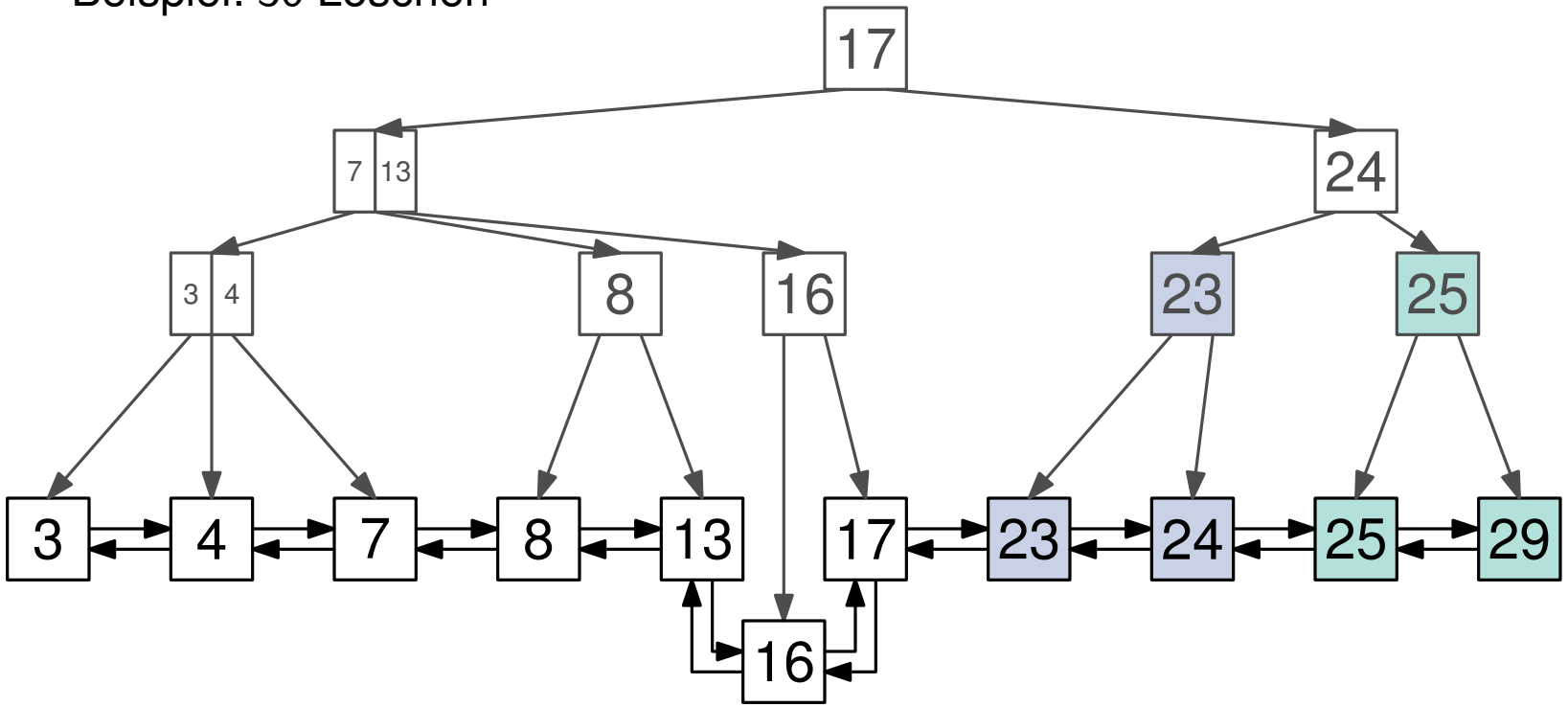
(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen
- Löschen: Blatt einfach abschneiden und danach (rekursiv) verschmelzen oder ausbalancieren Beispiel: 36 Löschen



(2, 3)-Bäume – Einfügen & Löschen

- Einfügen: Neues Blatt an Elter vom Nachfolger anhängen und danach (rekursiv) aufspalten wenn nötig Beispiel: 16 Einfügen
- Löschen: Blatt einfach abschneiden und danach (rekursiv) verschmelzen oder ausbalancieren Beispiel: 36 Löschen



Rot-Schwarz-Bäume

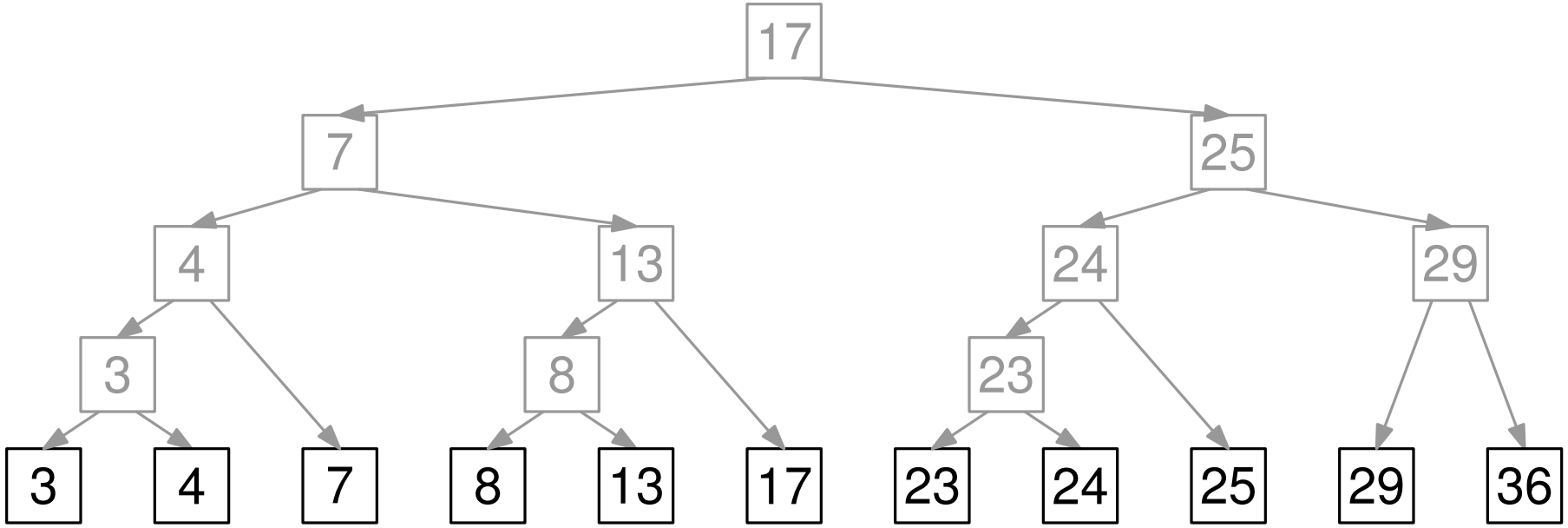
- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Rot-Schwarz-Bäume

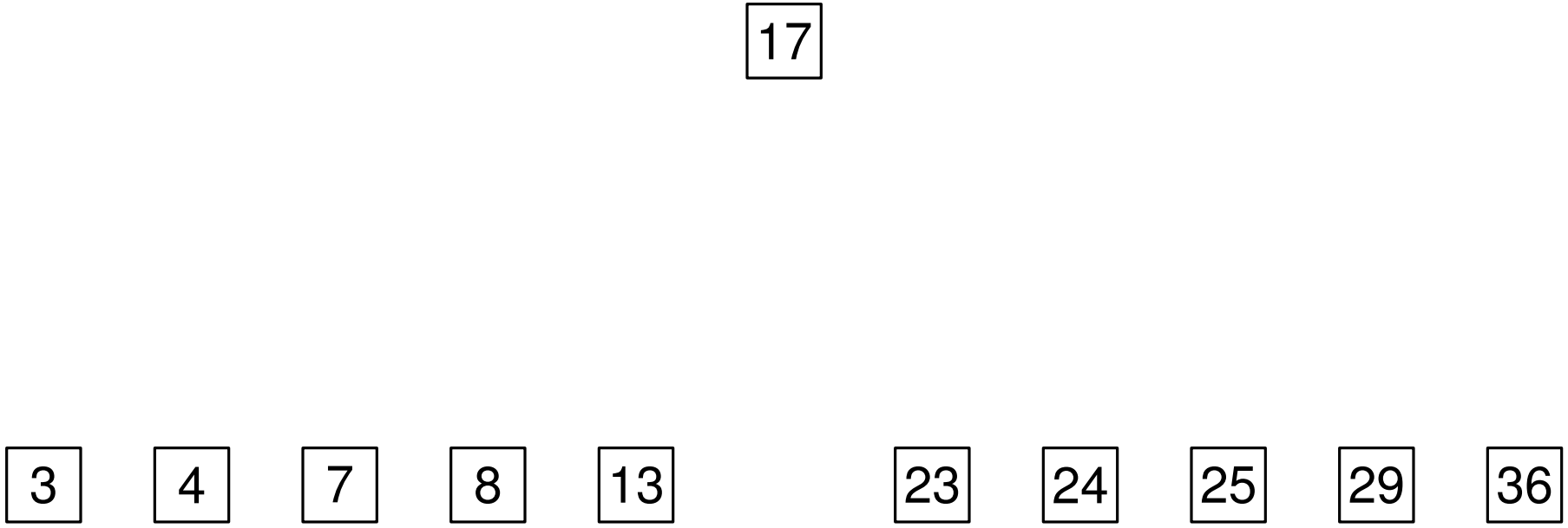
- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

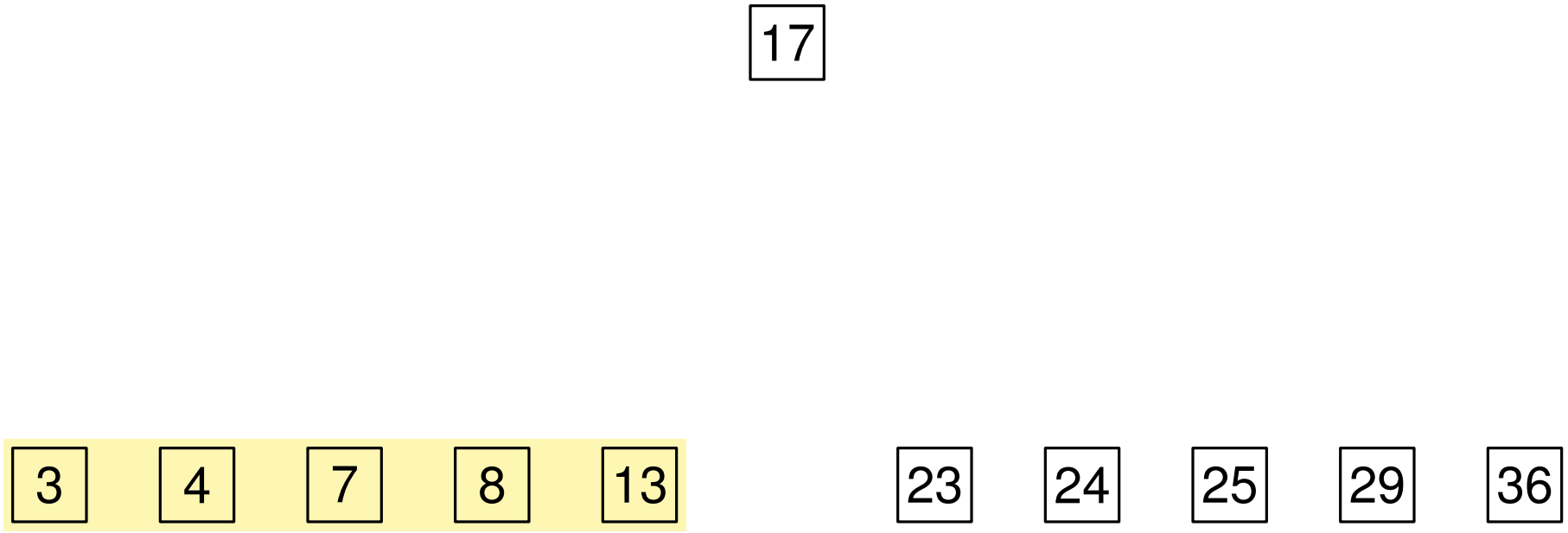
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



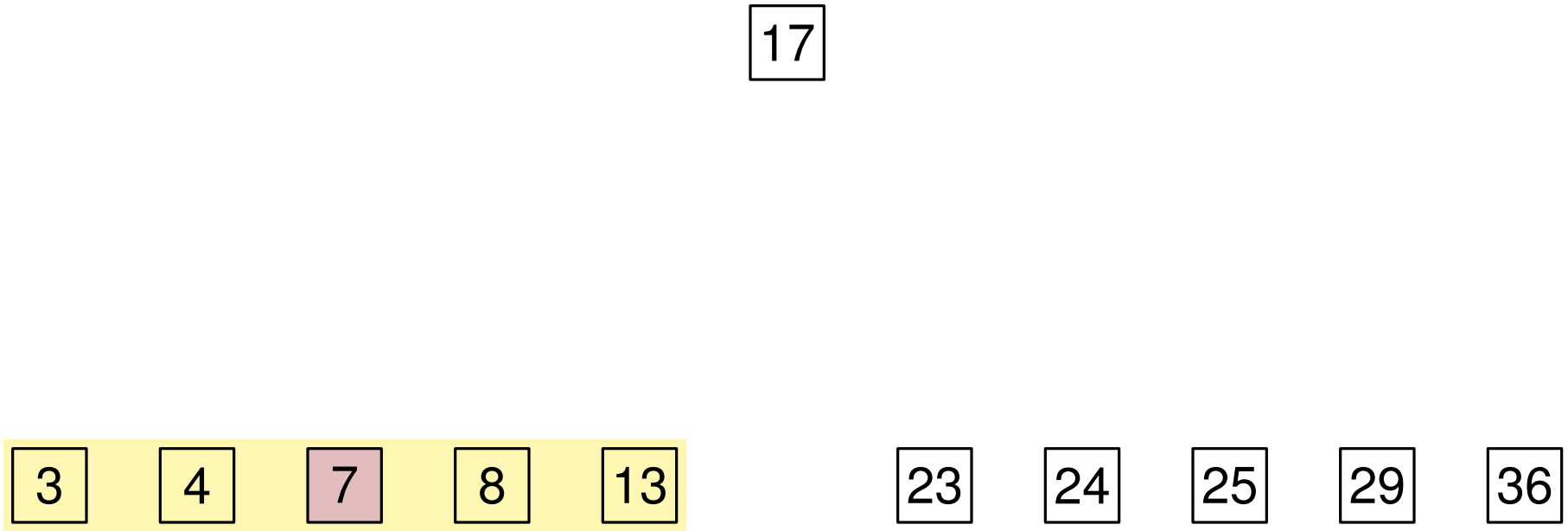
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



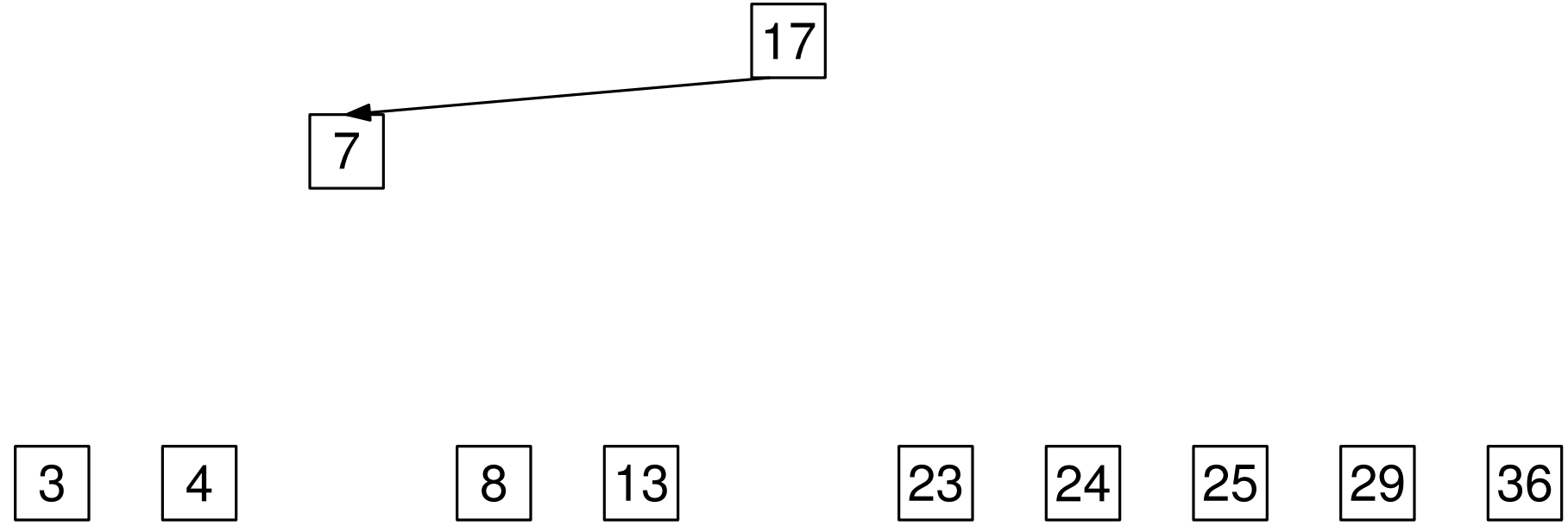
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



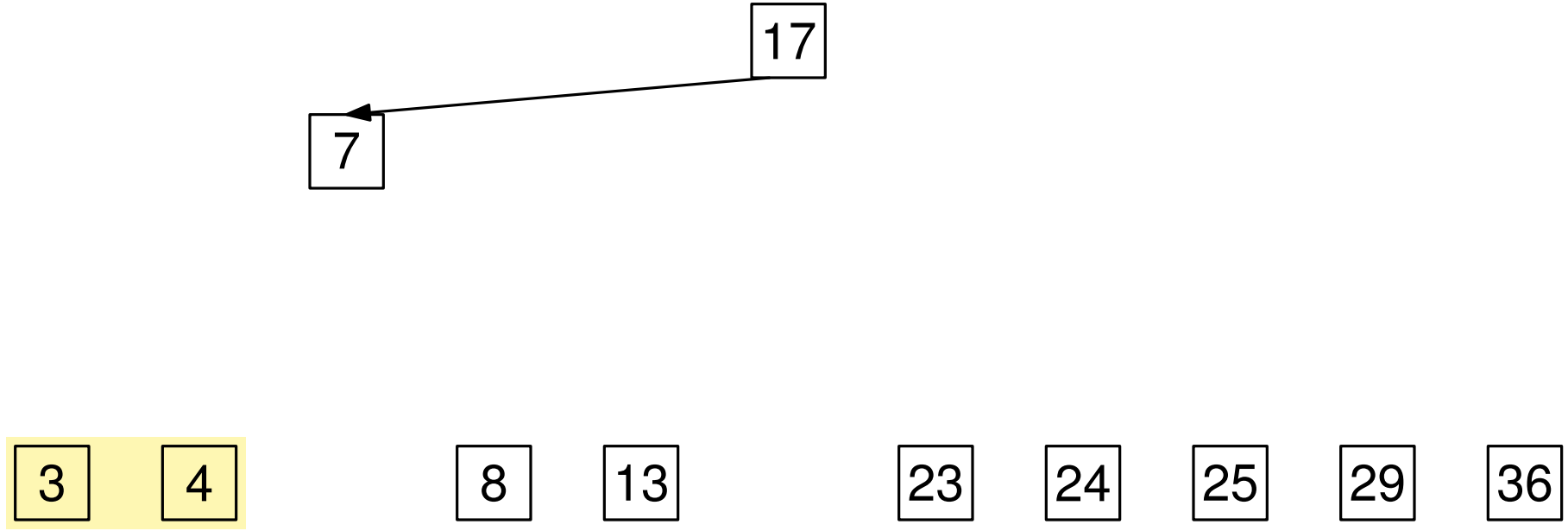
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



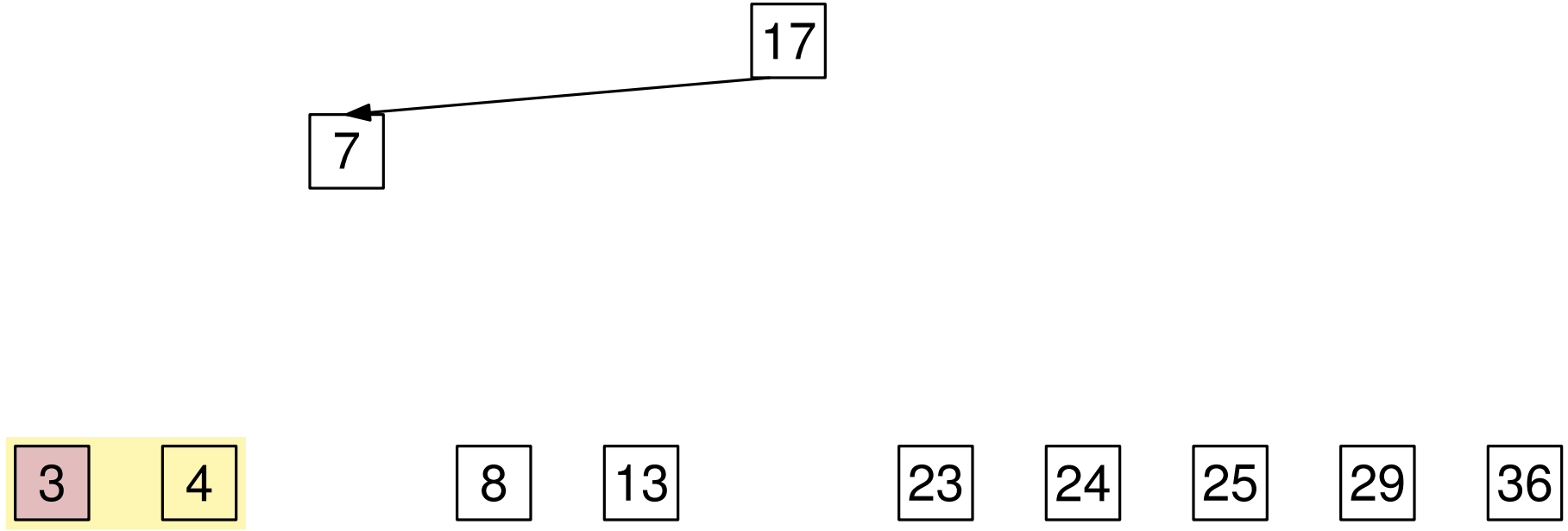
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



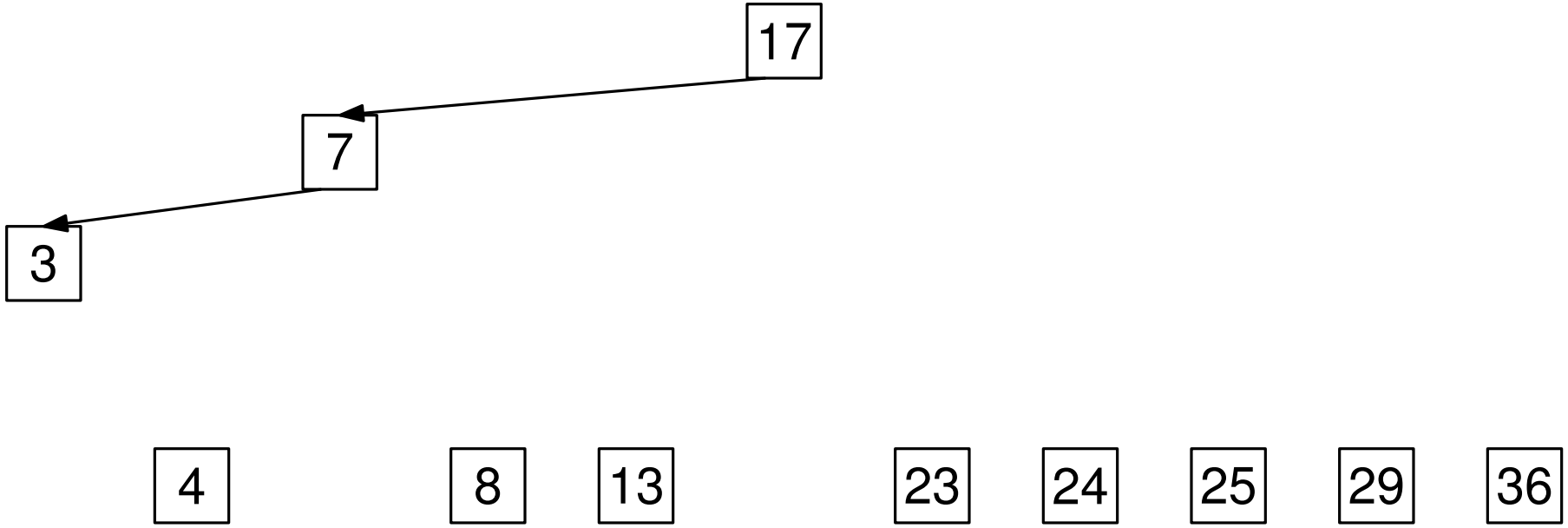
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



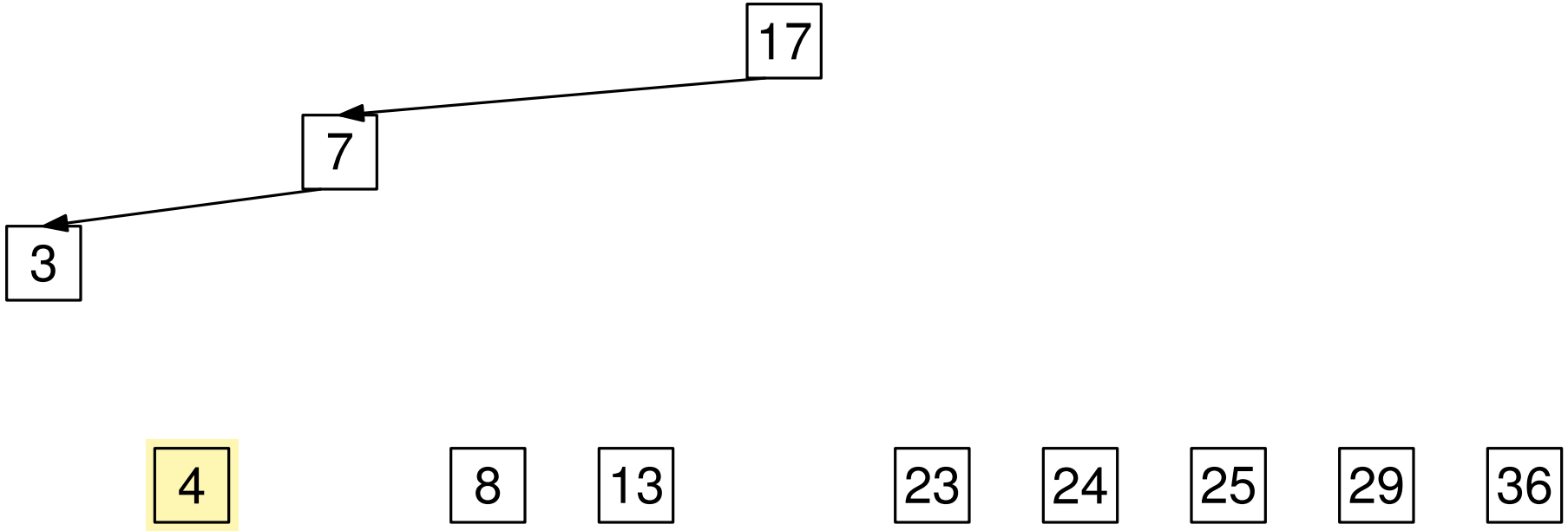
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



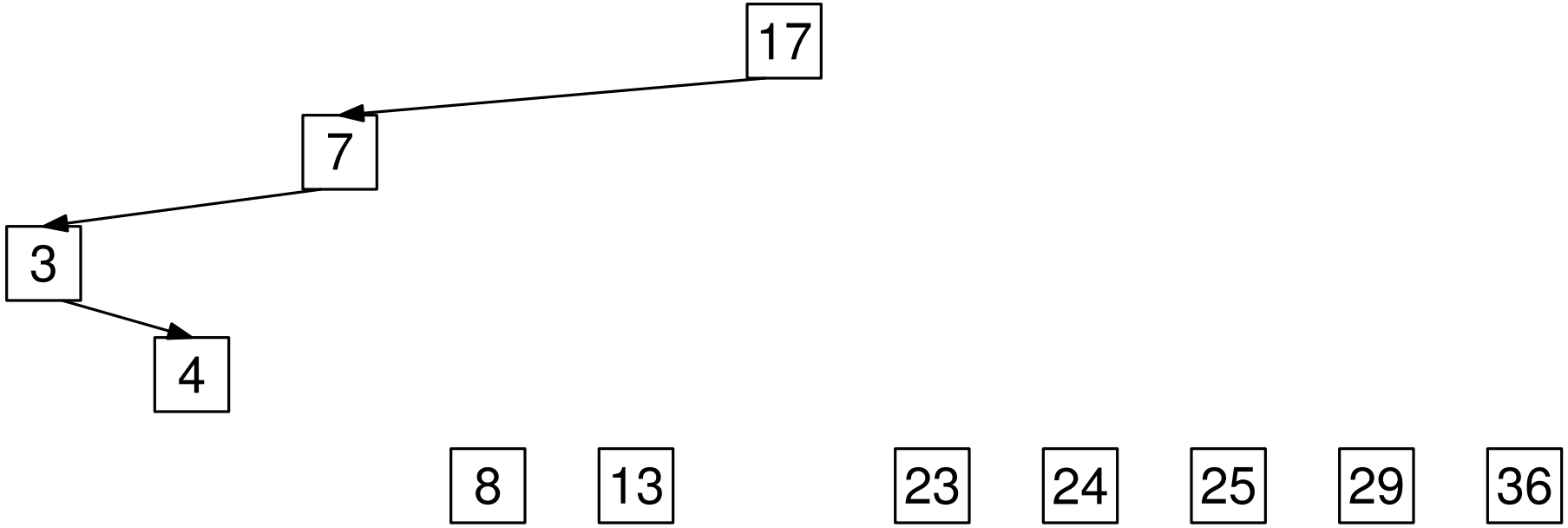
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



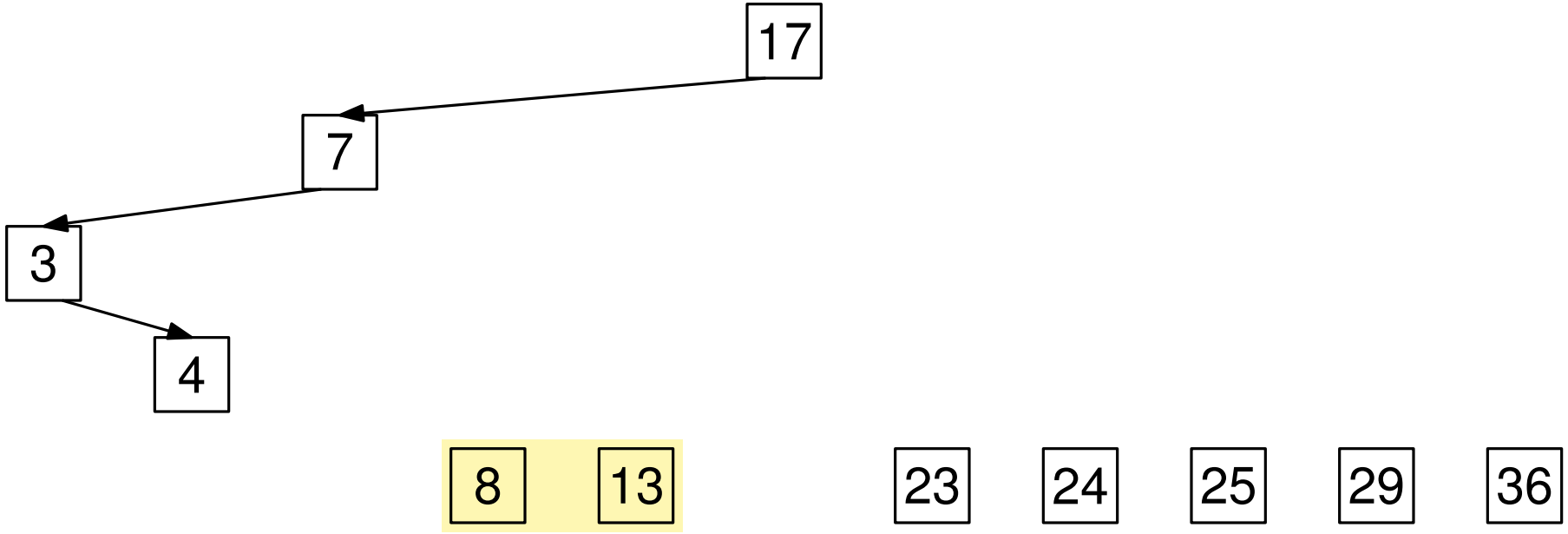
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



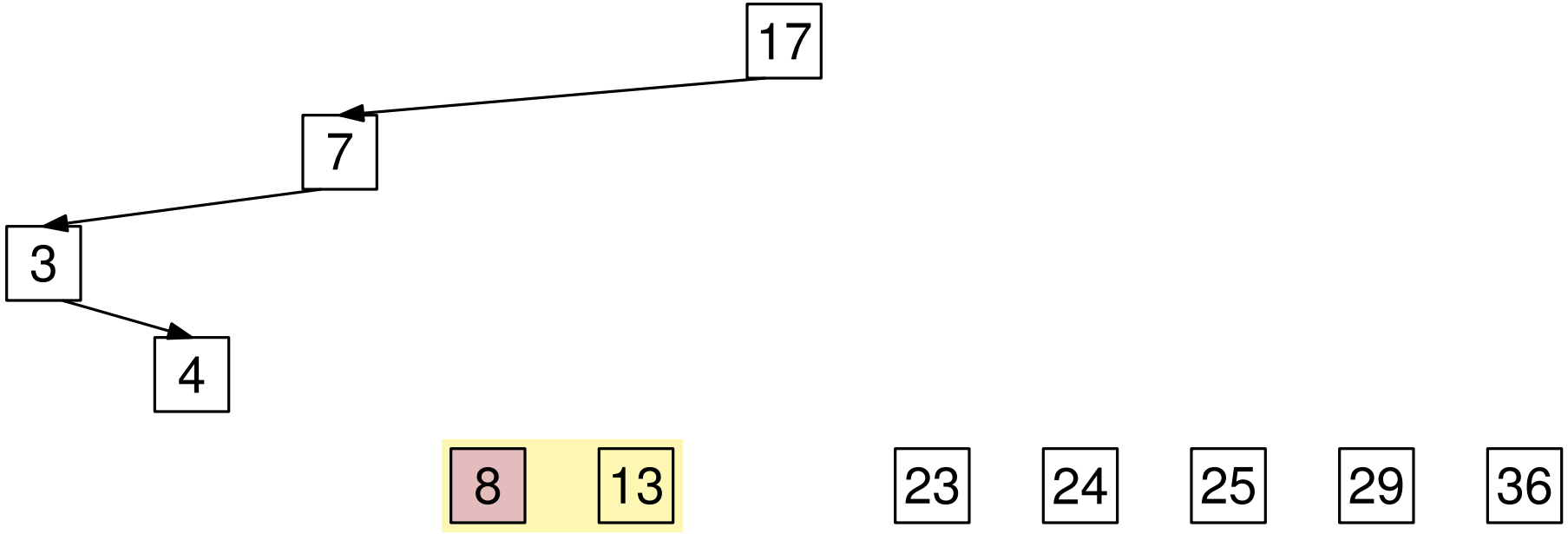
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



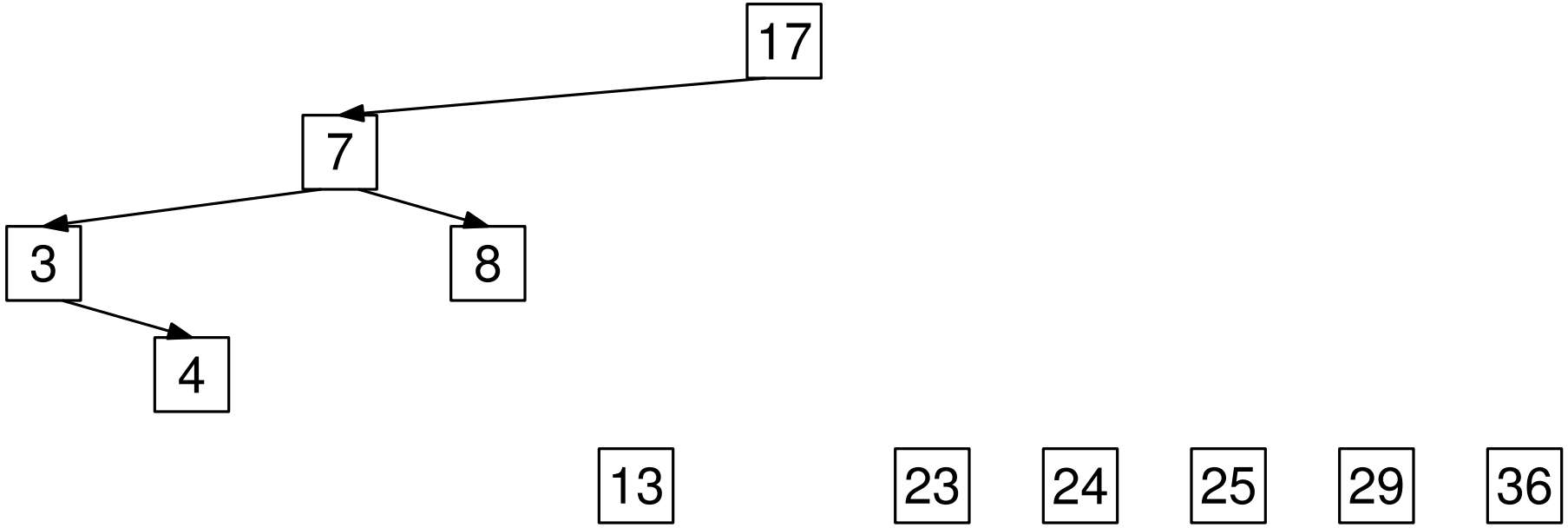
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



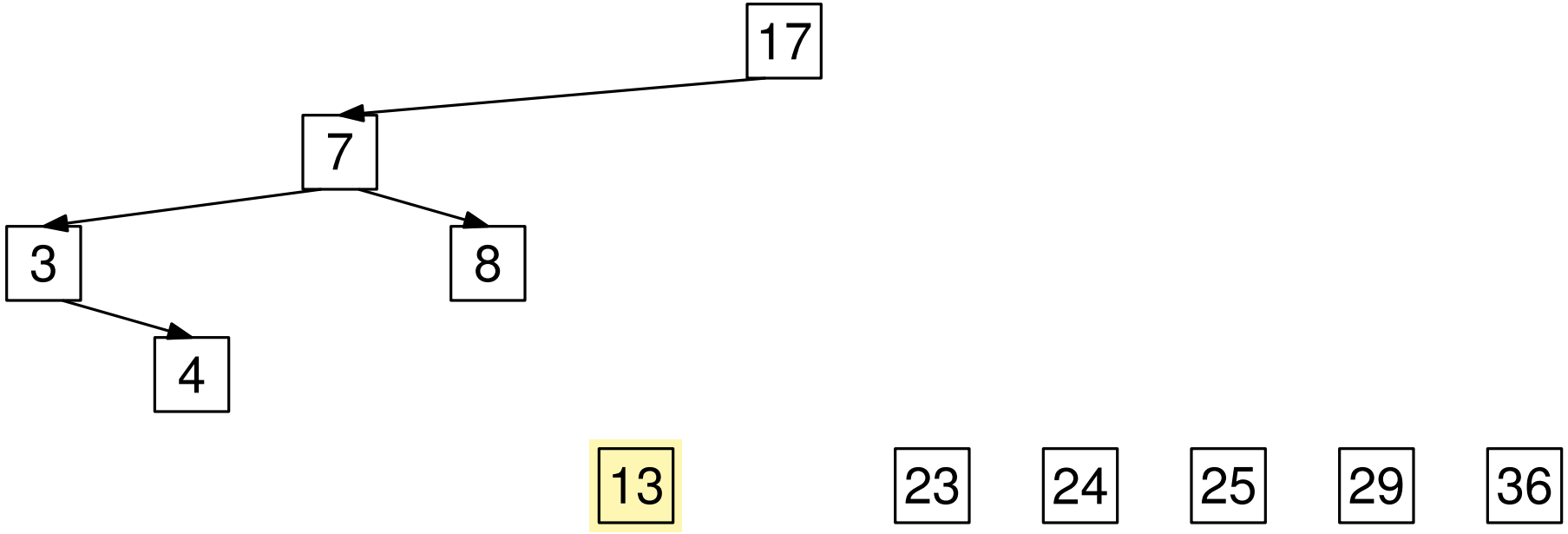
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



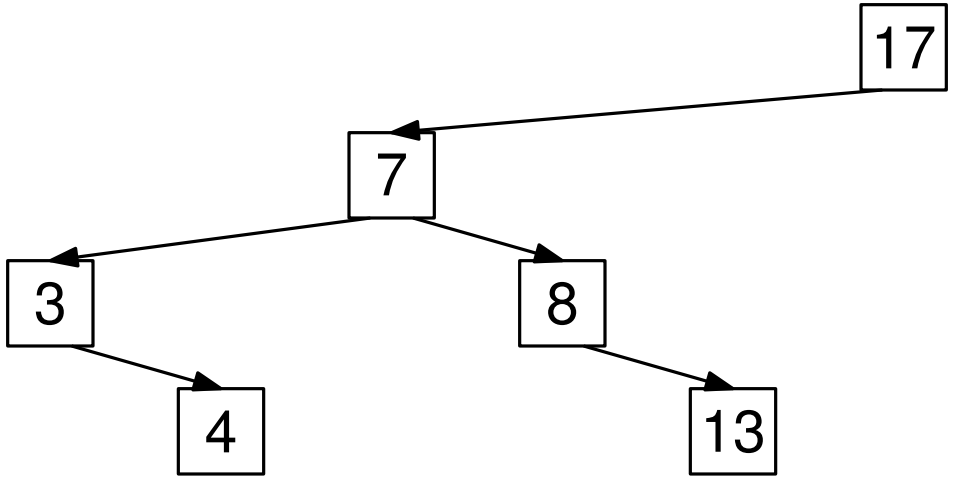
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



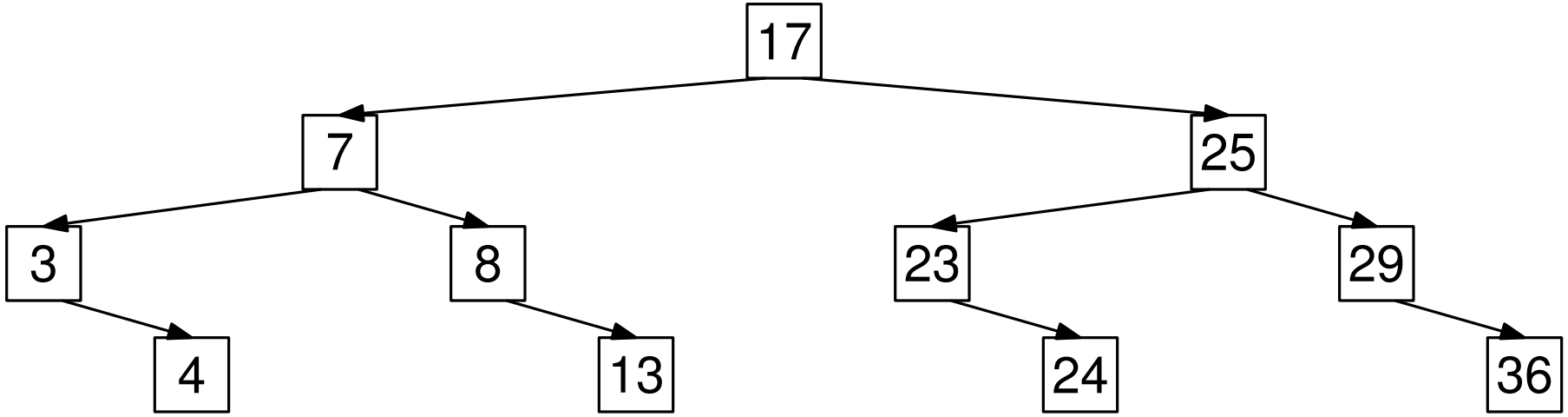
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



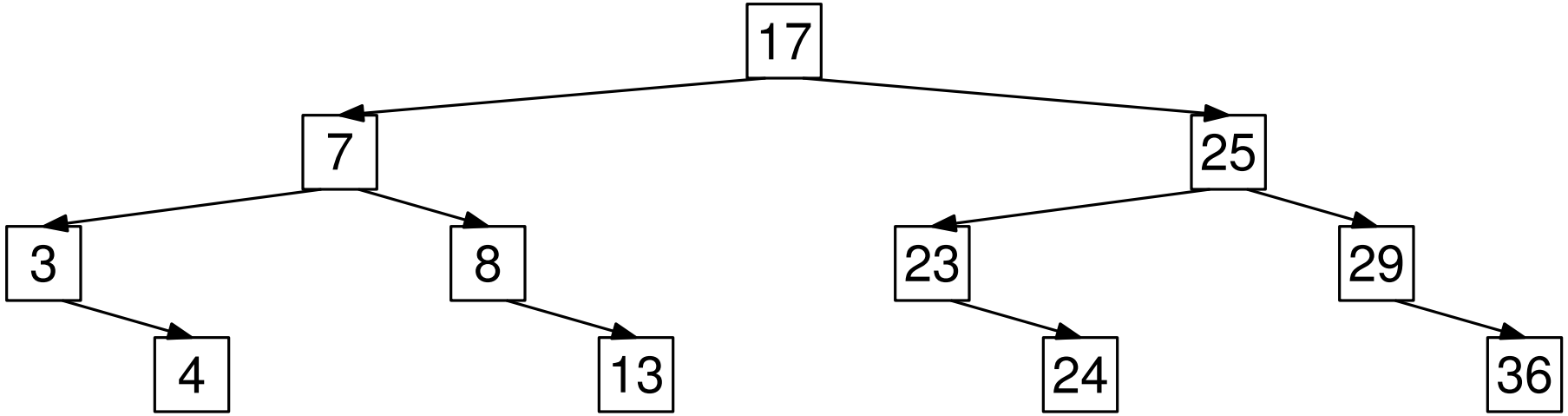
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern



Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

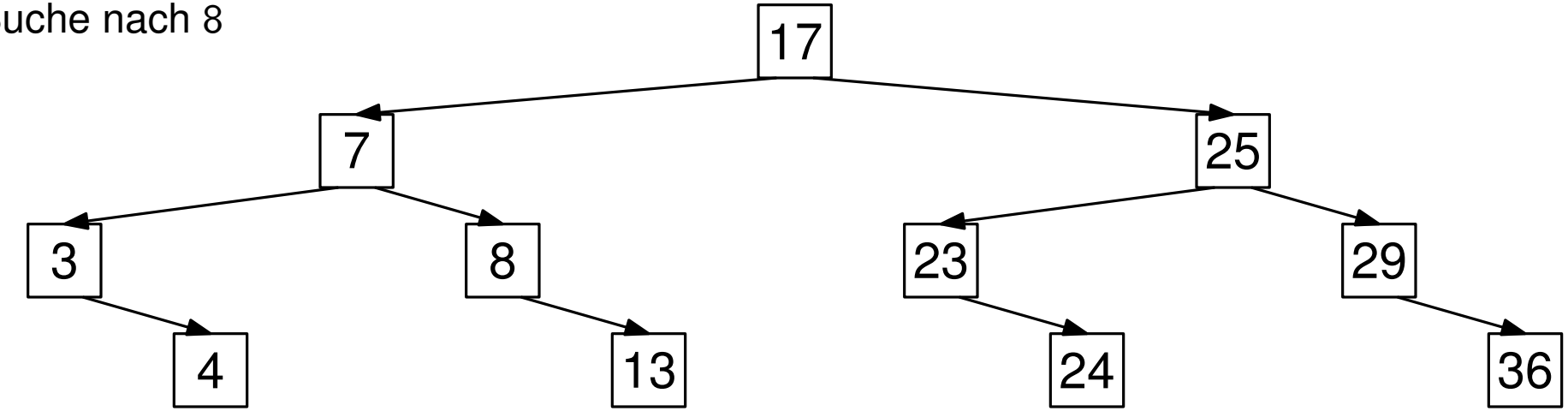


- Suche nach x: Vergleiche x mit Schlüssel k
  - $x < k$: Abstieg in linken Teilbaum
  - $x = k$: Element gefunden
  - $x > k$: Abstieg in rechten Teilbaum

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: Suche nach 8

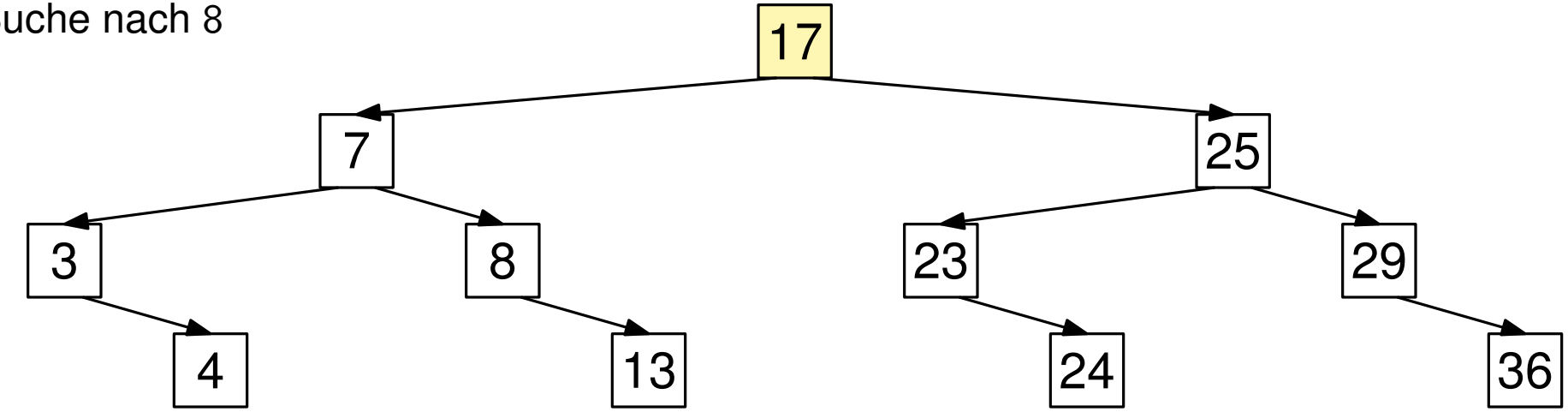


- Suche nach x: Vergleiche x mit Schlüssel k
  - $x < k$: Abstieg in linken Teilbaum
  - $x = k$: Element gefunden
  - $x > k$: Abstieg in rechten Teilbaum

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: Suche nach 8

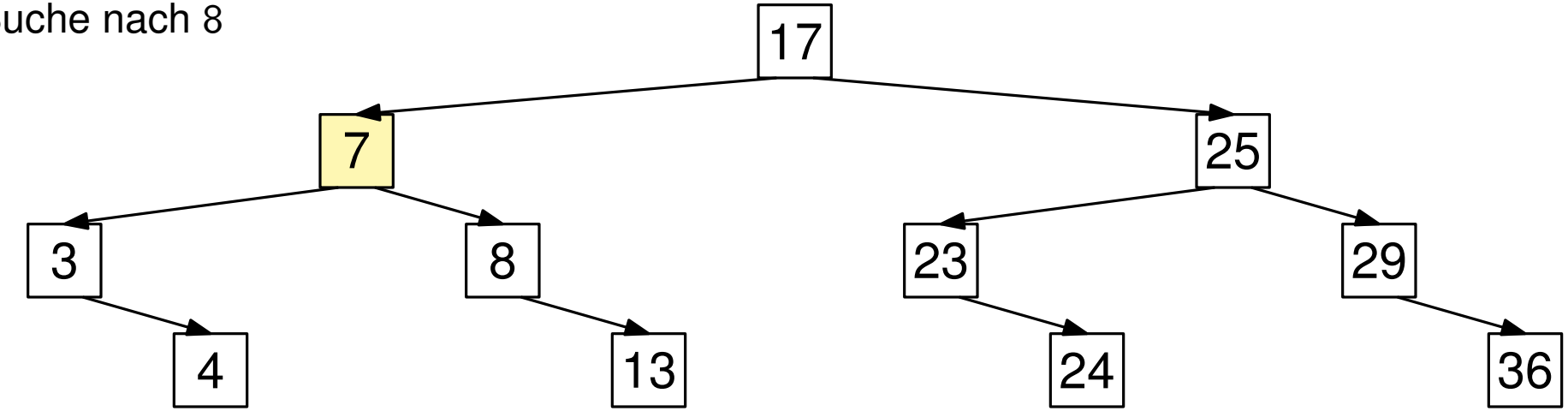


- Suche nach x: Vergleiche x mit Schlüssel k
  - $x < k$: Abstieg in linken Teilbaum
  - $x = k$: Element gefunden
  - $x > k$: Abstieg in rechten Teilbaum

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: Suche nach 8

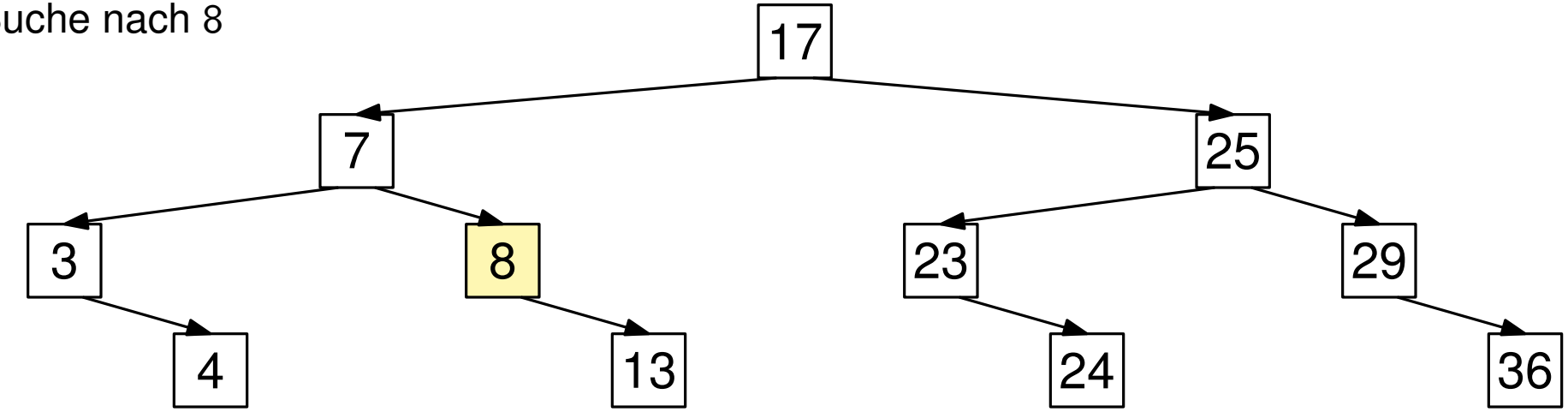


- Suche nach x: Vergleiche x mit Schlüssel k
  - $x < k$: Abstieg in linken Teilbaum
  - $x = k$: Element gefunden
  - $x > k$: Abstieg in rechten Teilbaum

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: Suche nach 8

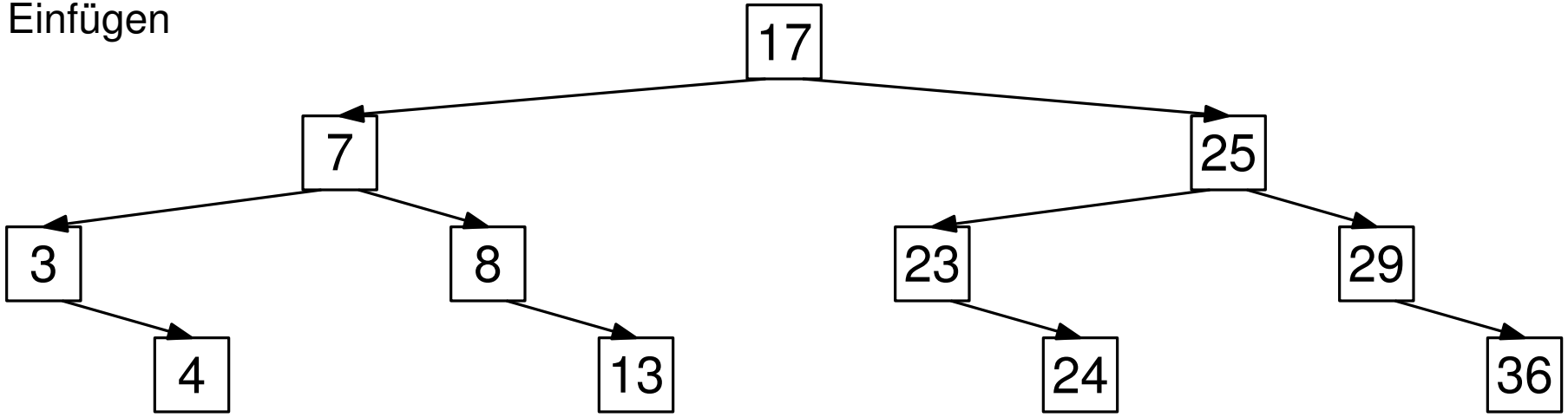


- Suche nach x: Vergleiche x mit Schlüssel k
  - $x < k$: Abstieg in linken Teilbaum
  - $x = k$: Element gefunden
  - $x > k$: Abstieg in rechten Teilbaum

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 50 Einfügen

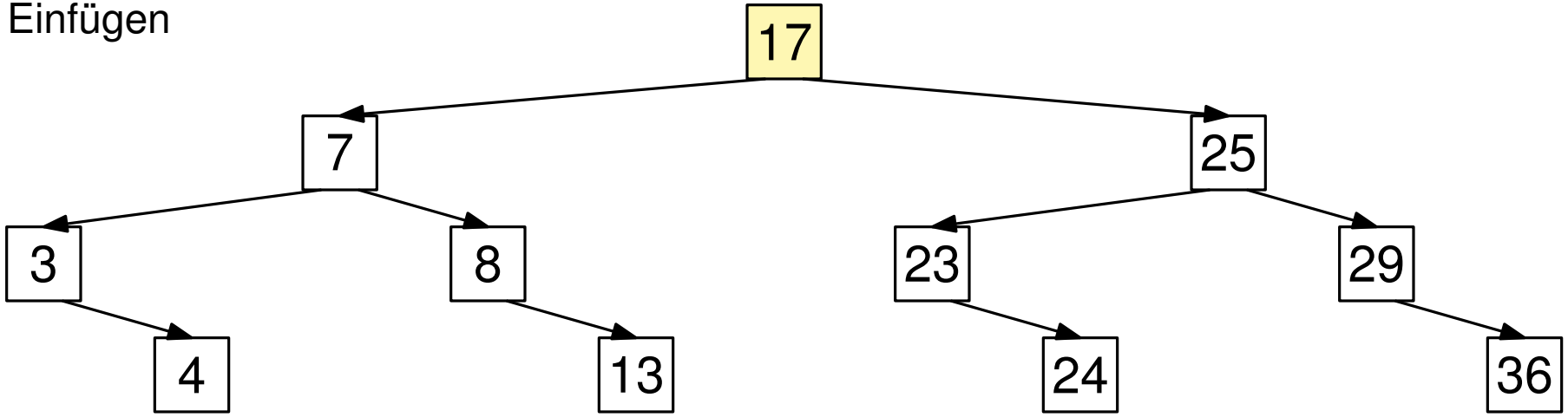


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 50 Einfügen

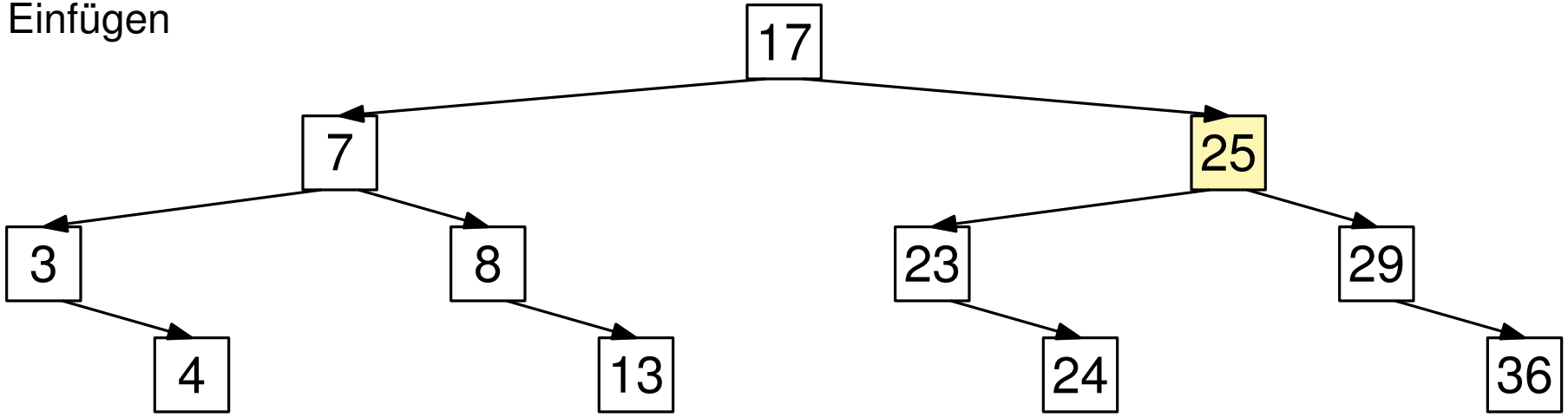


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 50 Einfügen

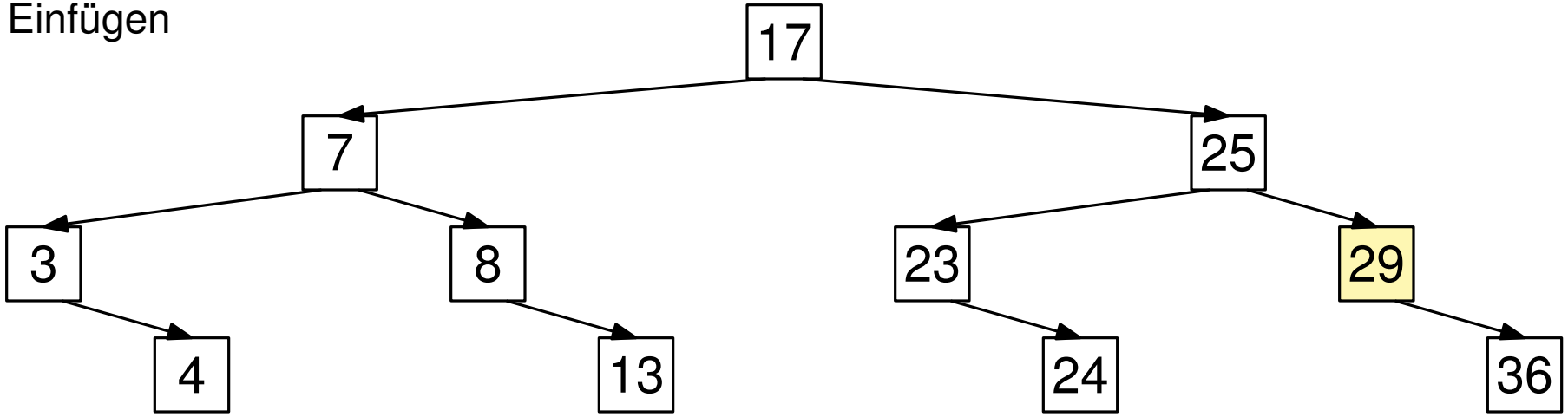


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 50 Einfügen

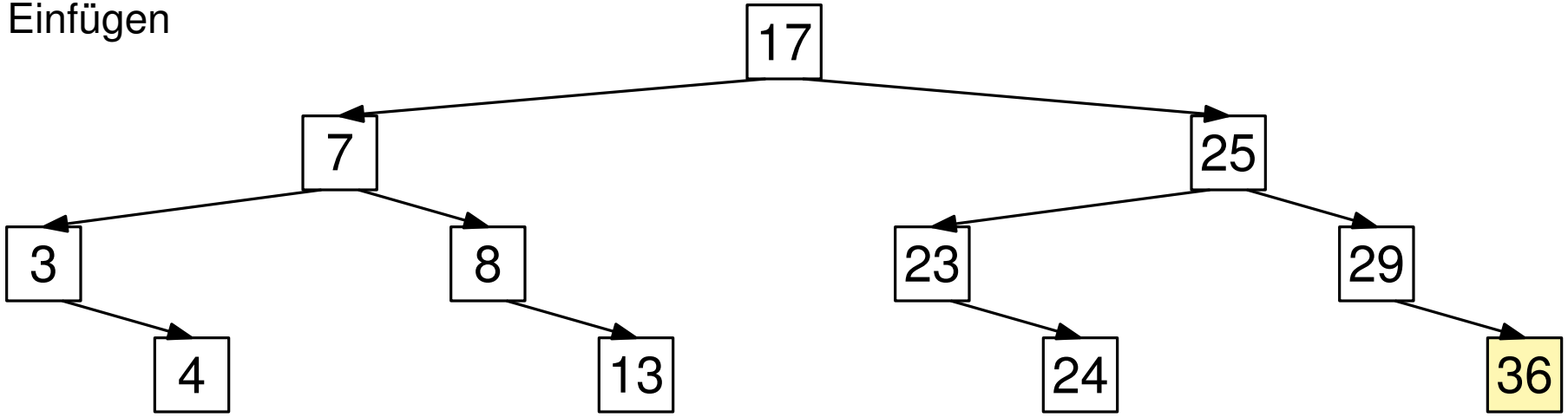


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 50 Einfügen

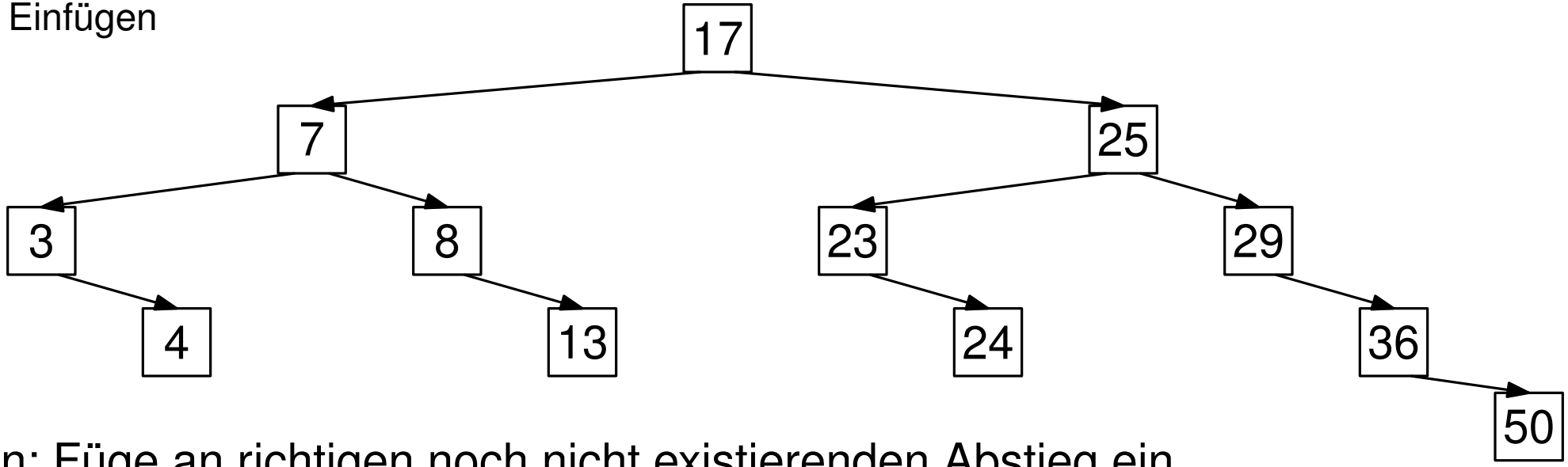


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 50 Einfügen

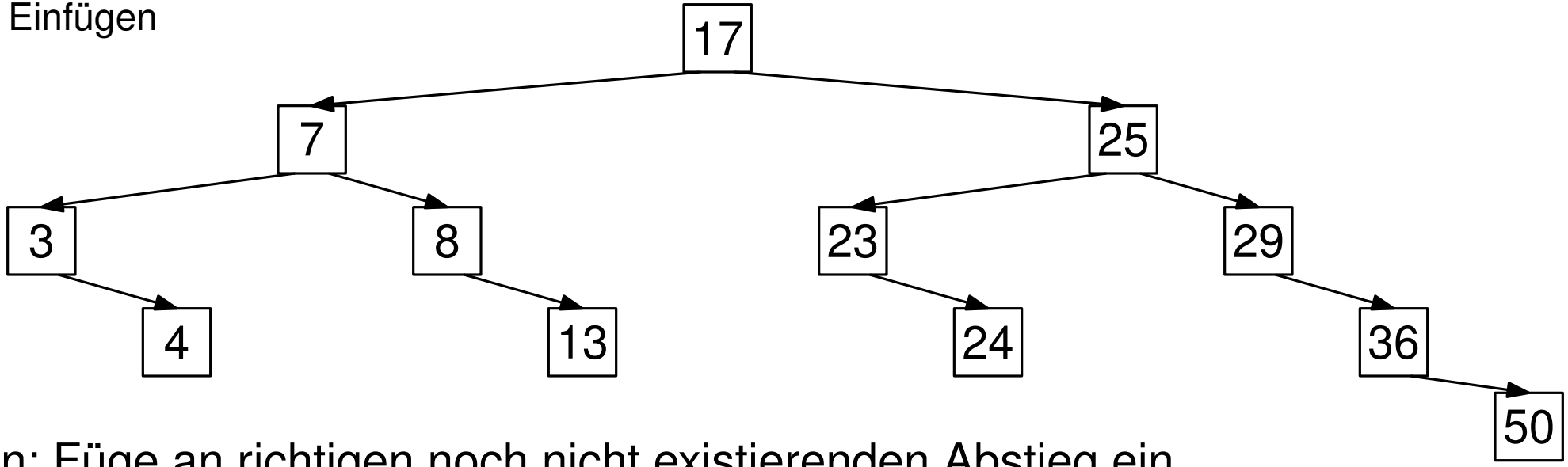


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 37 Einfügen

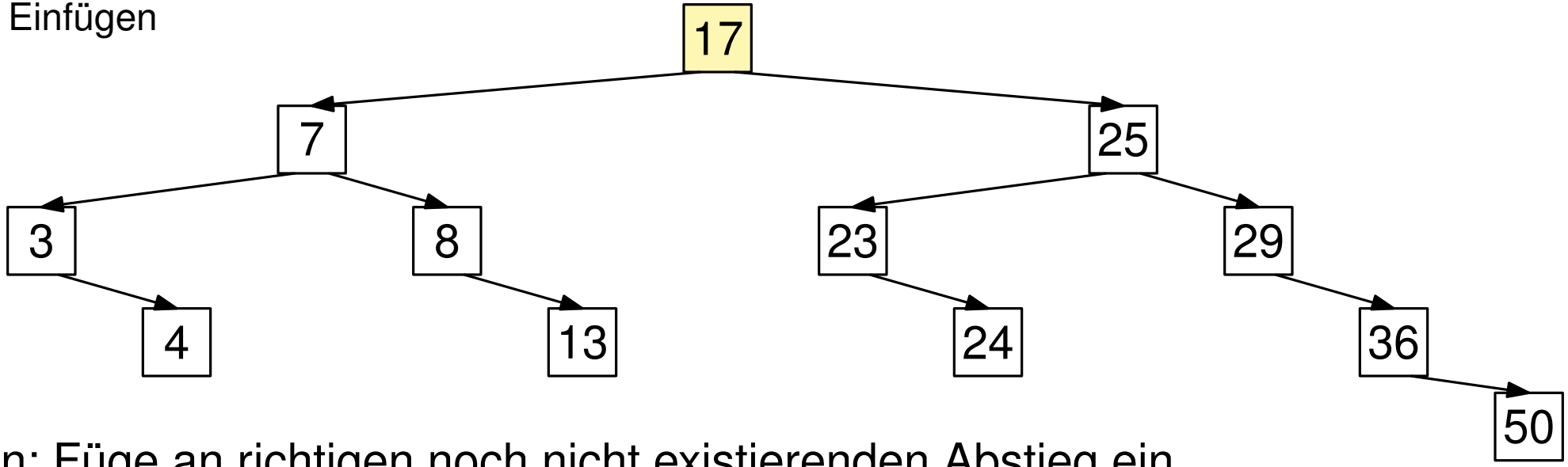


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 37 Einfügen

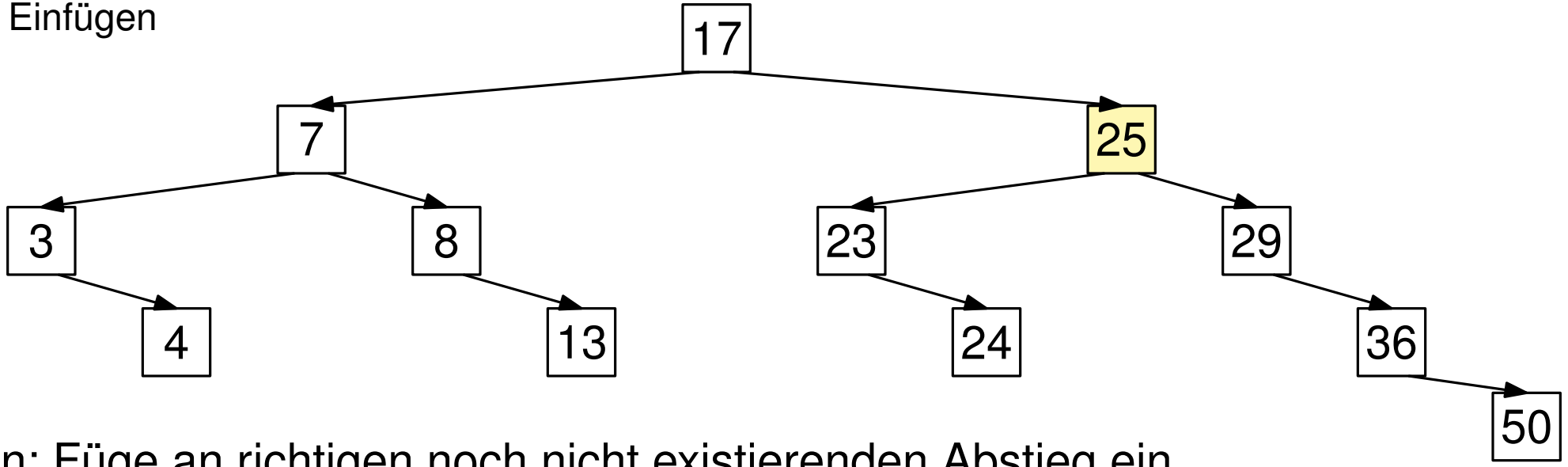


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 37 Einfügen

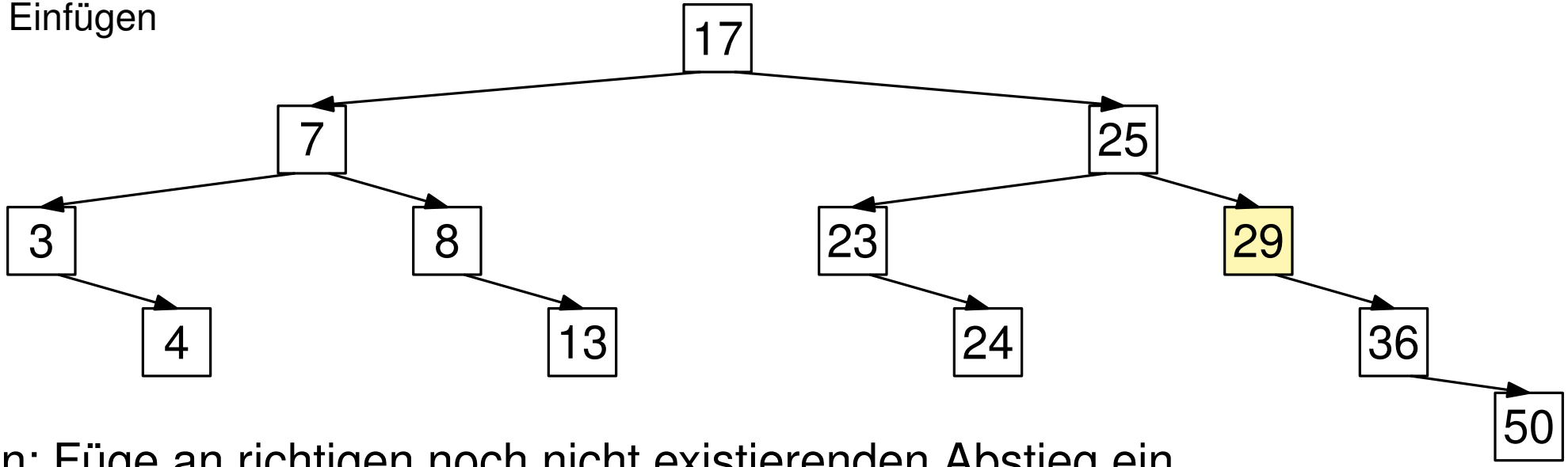


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 37 Einfügen

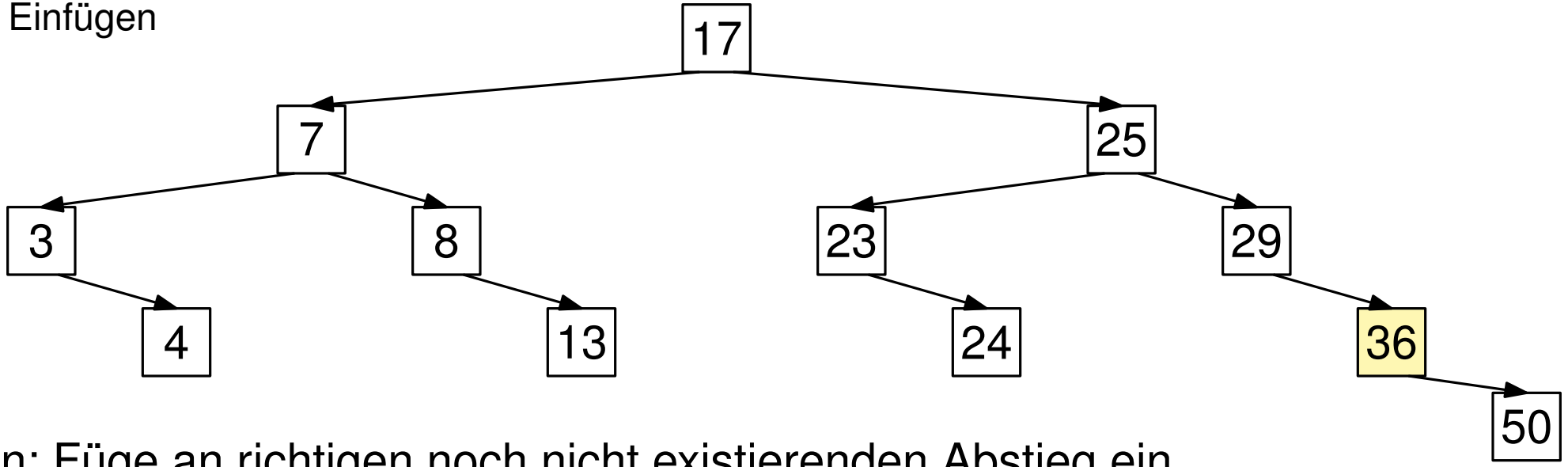


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 37 Einfügen

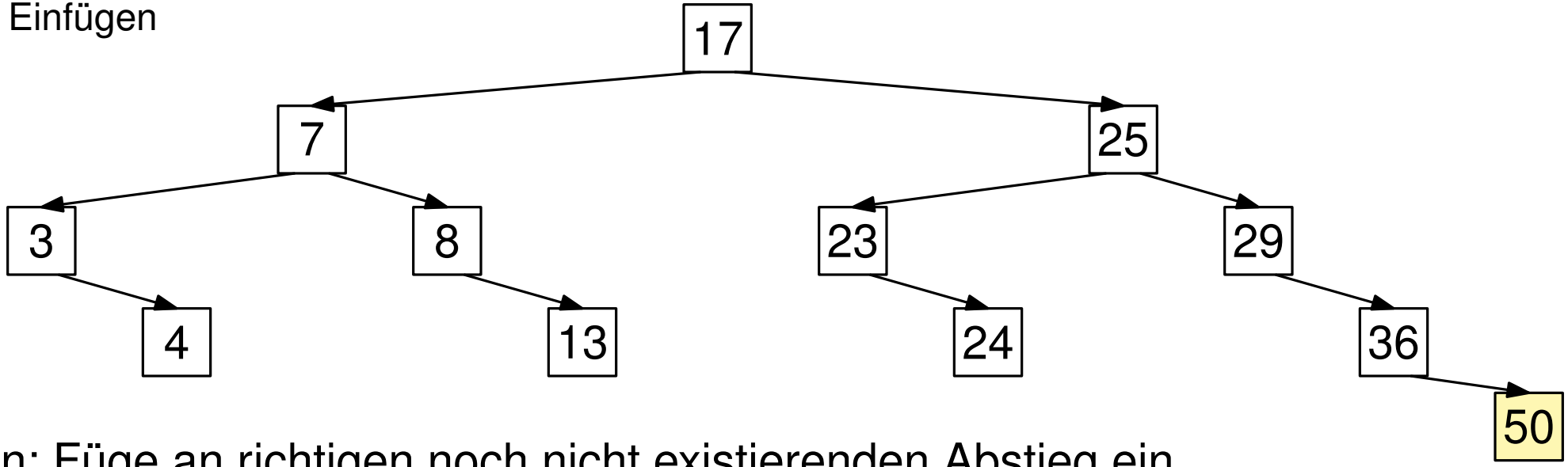


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 37 Einfügen

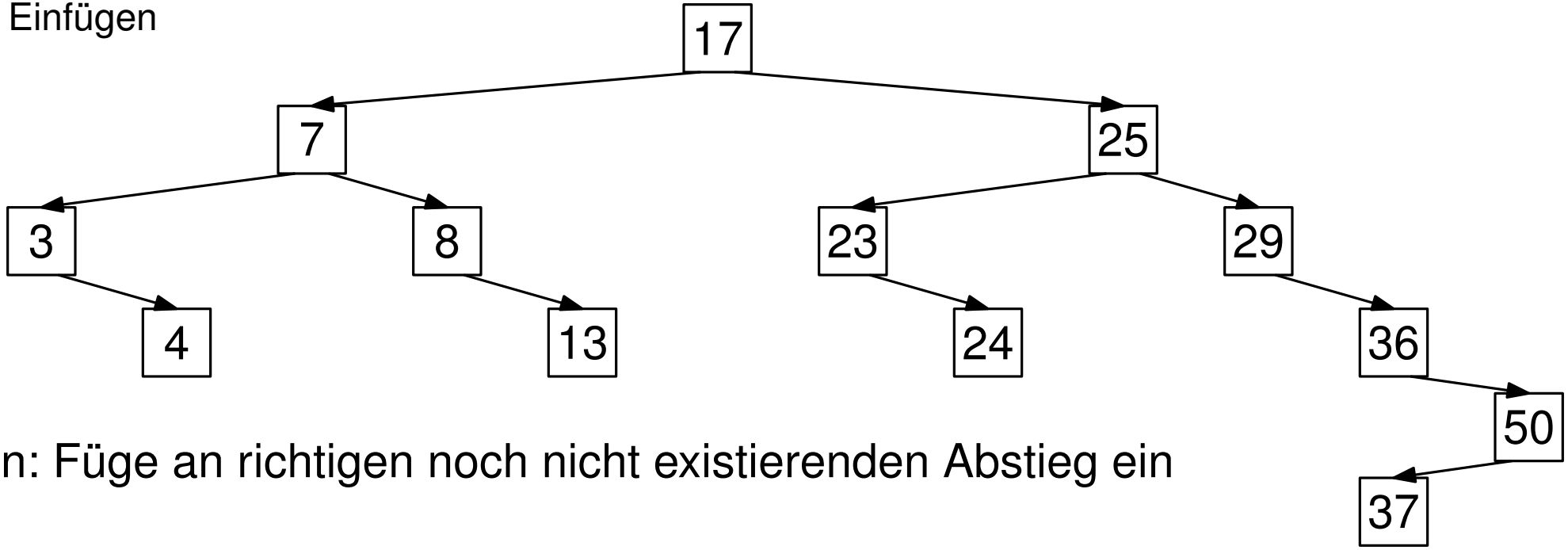


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 37 Einfügen

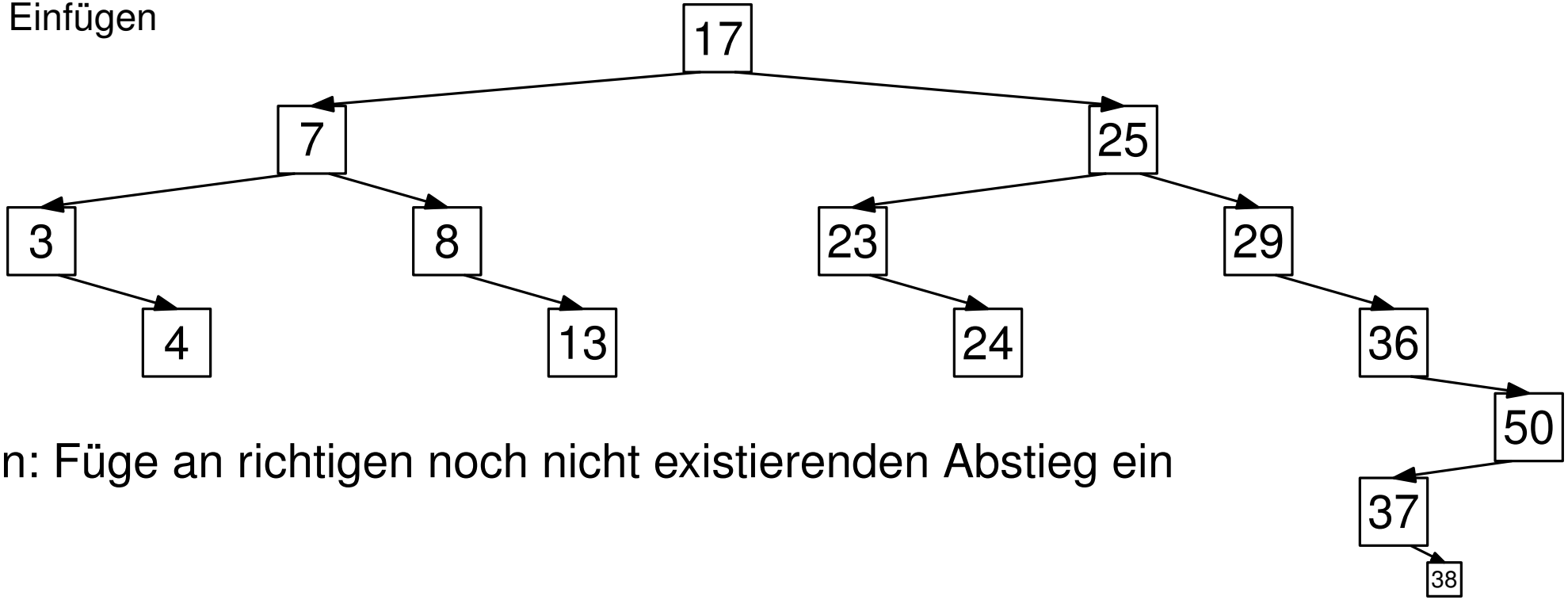


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 38 Einfügen

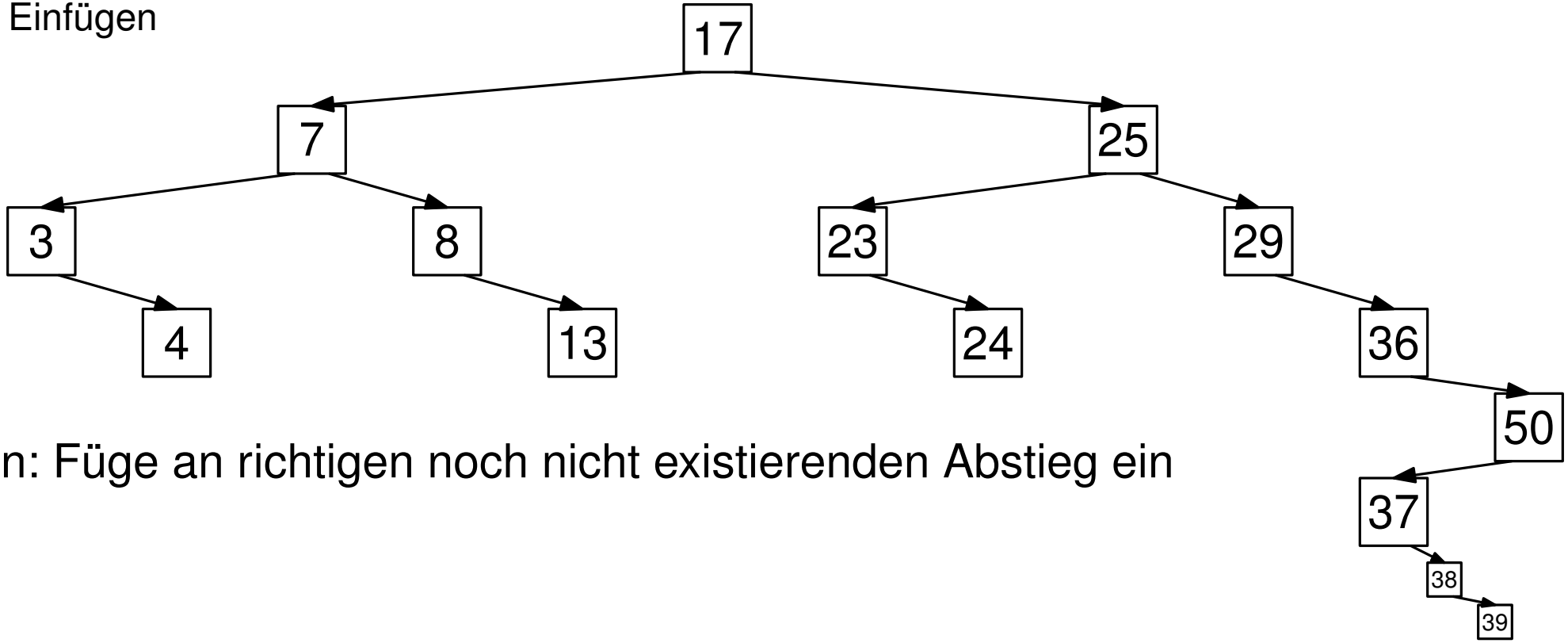


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

Beispiel: 39 Einfügen

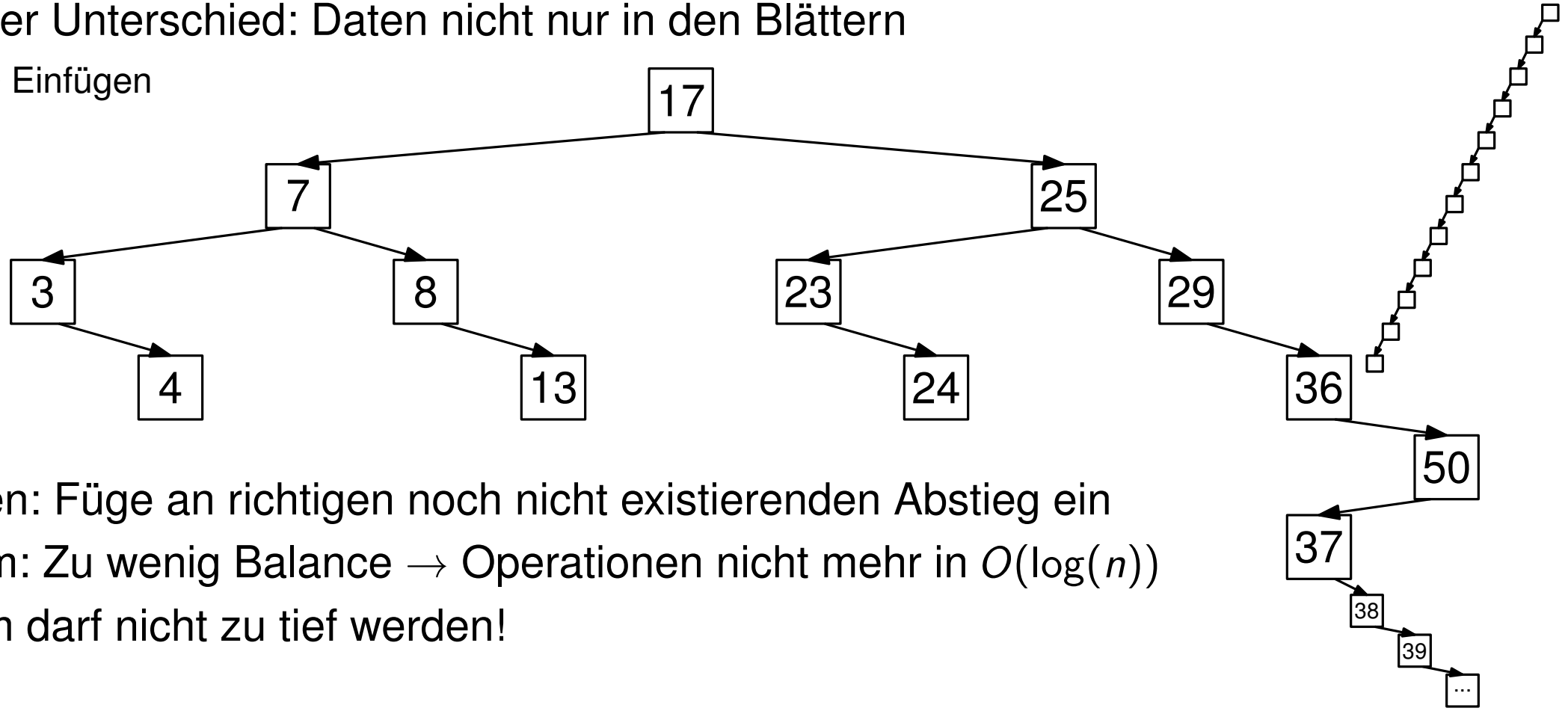


- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Wichtiger Unterschied: Daten nicht nur in den Blättern

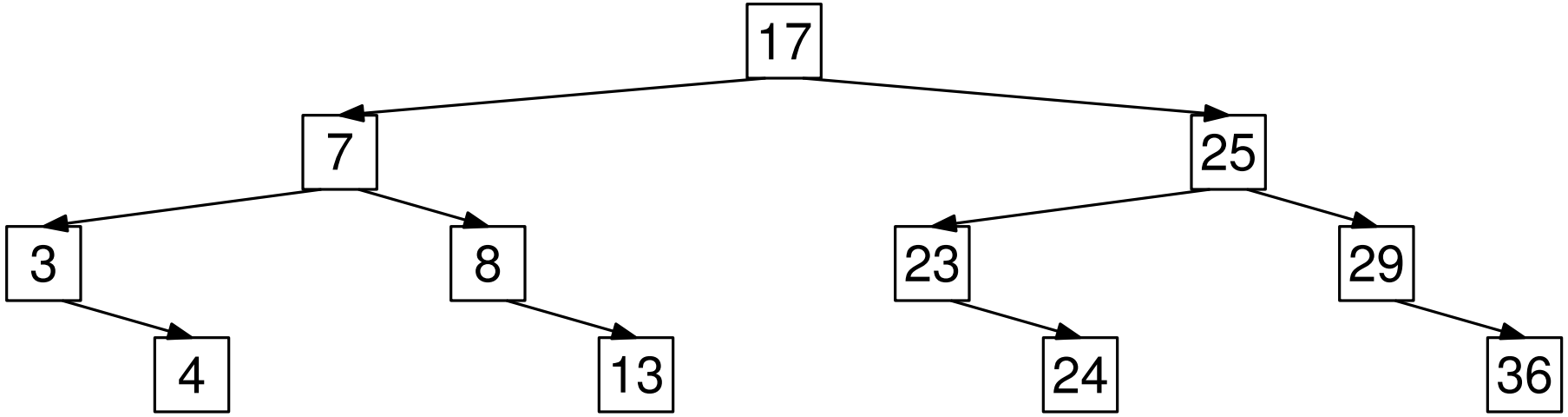
Beispiel: 39 Einfügen



- Einfügen: Füge an richtigen noch nicht existierenden Abstieg ein
- Problem: Zu wenig Balance $\rightarrow$ Operationen nicht mehr in $O(\log(n))$
  - Baum darf nicht zu tief werden!

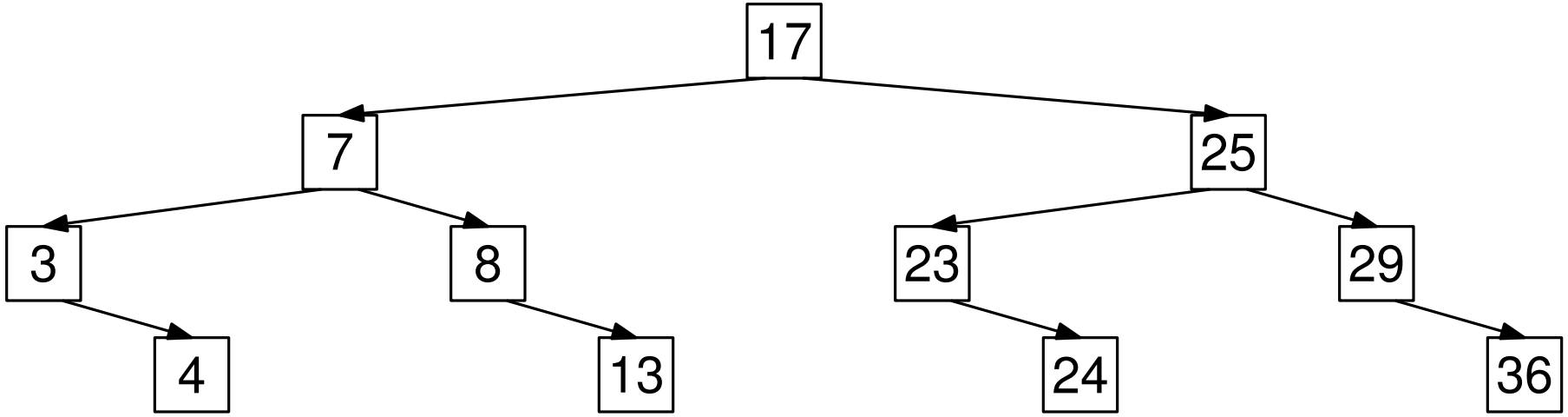
Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.



Rot-Schwarz-Bäume

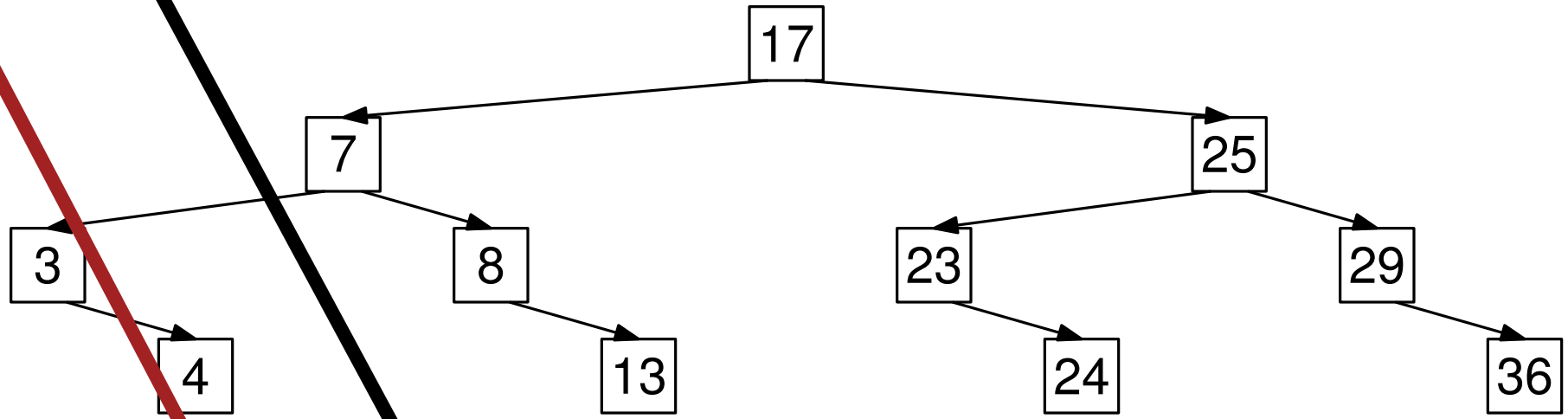
- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.



- Knoten werden rot oder blau gefärbt

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.

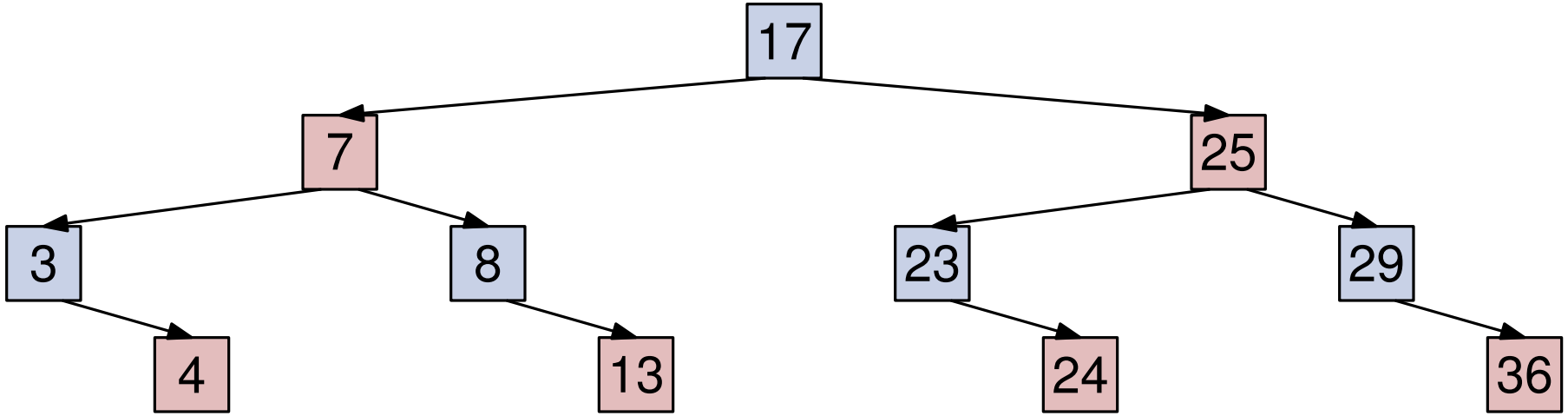


- Knoten werden rot oder blau gefärbt

?

Rot-Schwarz-Bäume

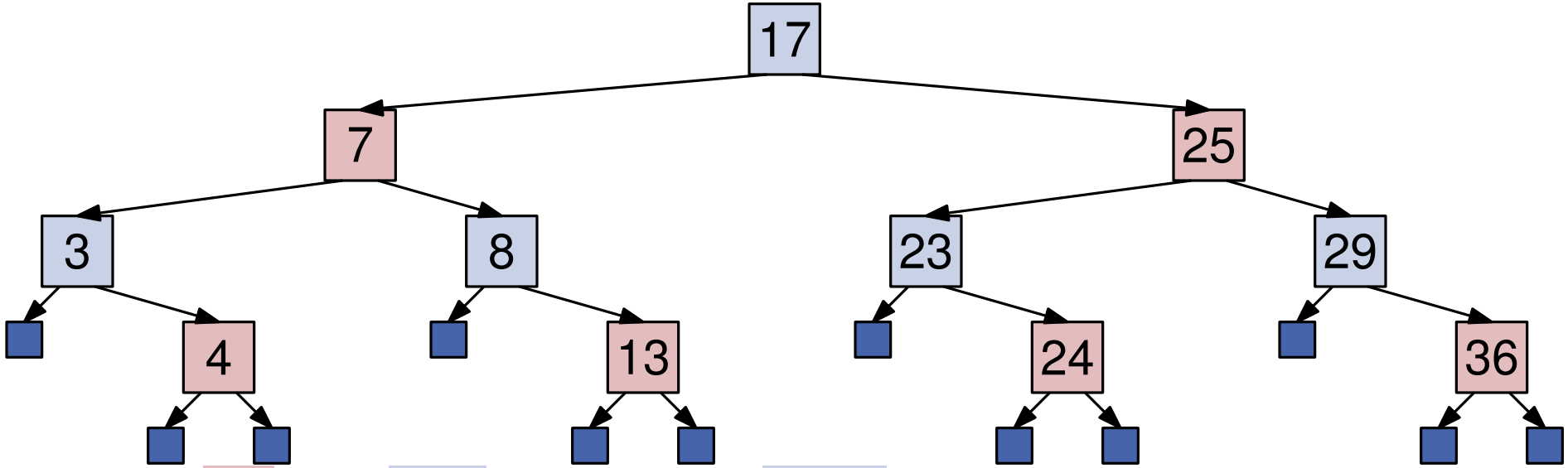
- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.



- Knoten werden rot oder blau gefärbt

Rot-Schwarz-Bäume

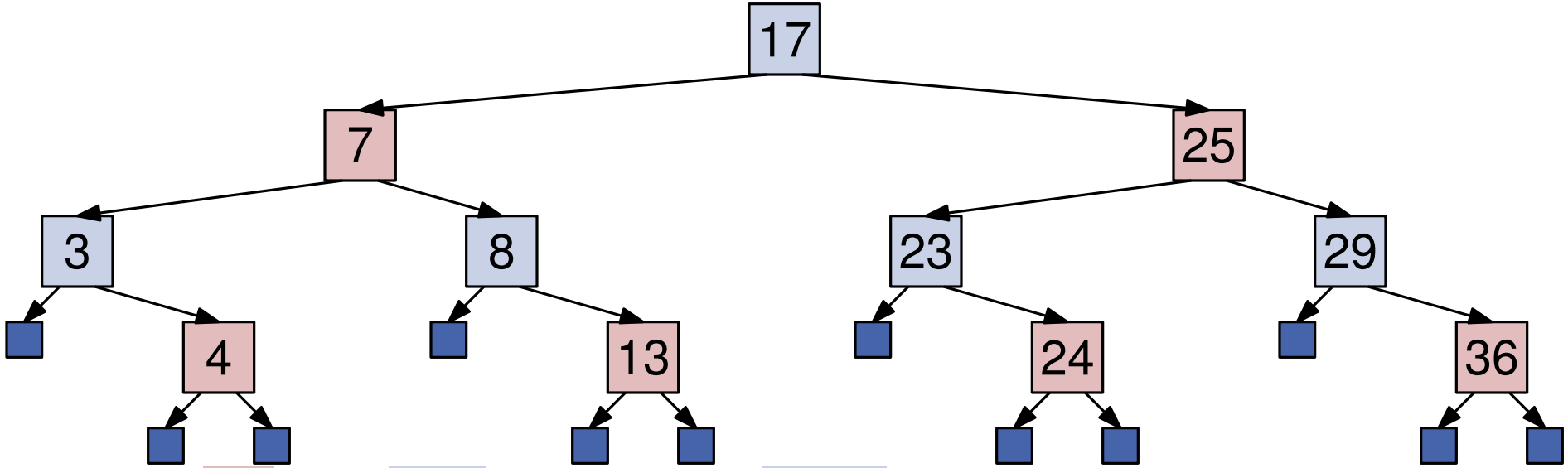
- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.



- Knoten werden rot oder blau gefärbt und blaue Dummy-Blätter eingefügt

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.

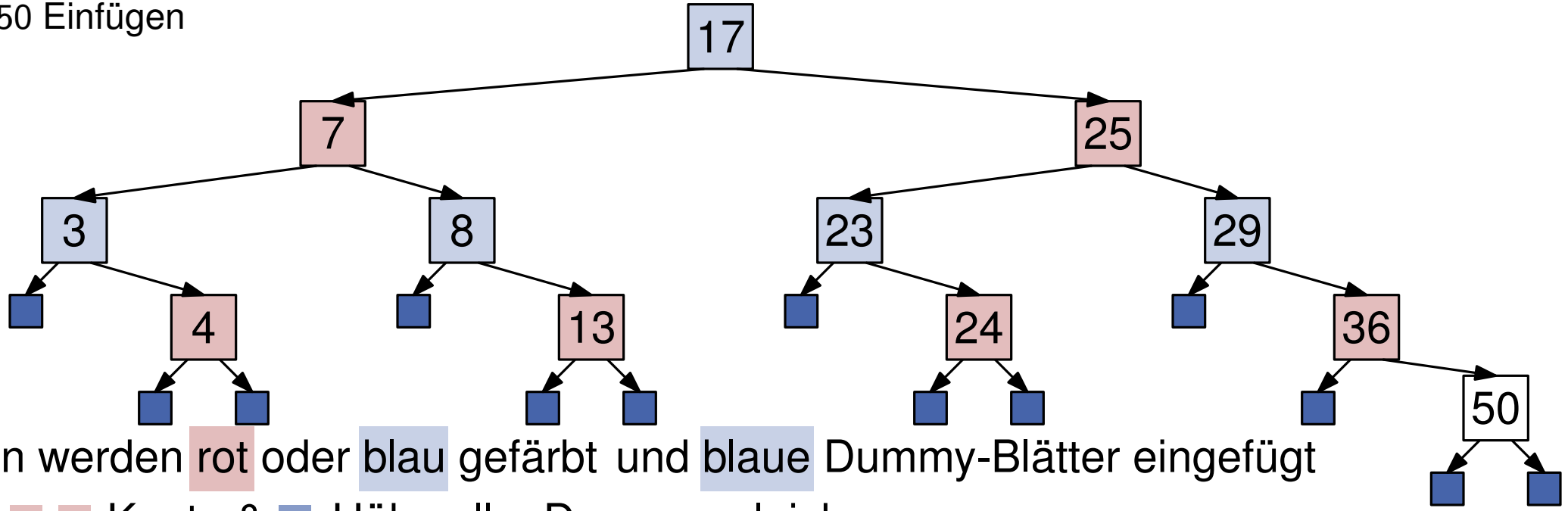


- Knoten werden rot oder blau gefärbt und blaue Dummy-Blätter eingefügt
- Keine – Kante & -Höhe aller Dummys gleich.

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.

Beispiel: 50 Einfügen

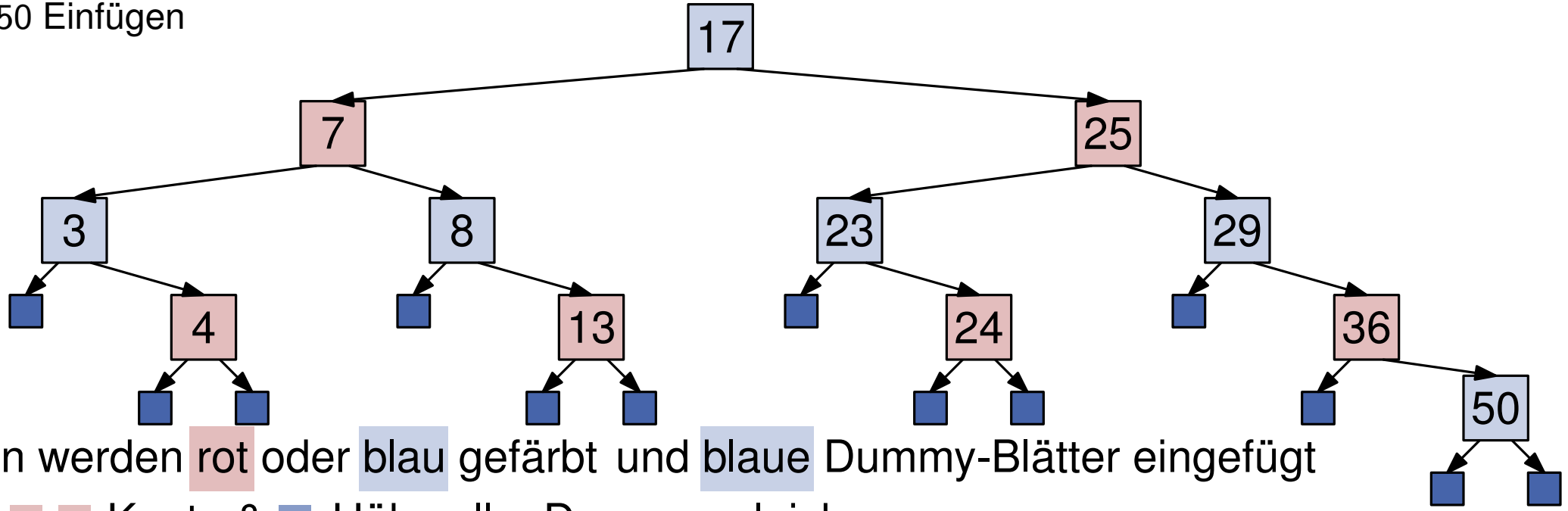


- Knoten werden rot oder blau gefärbt und blaue Dummy-Blätter eingefügt
- Keine – Kante & -Höhe aller Dummys gleich.

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.

Beispiel: 50 Einfügen

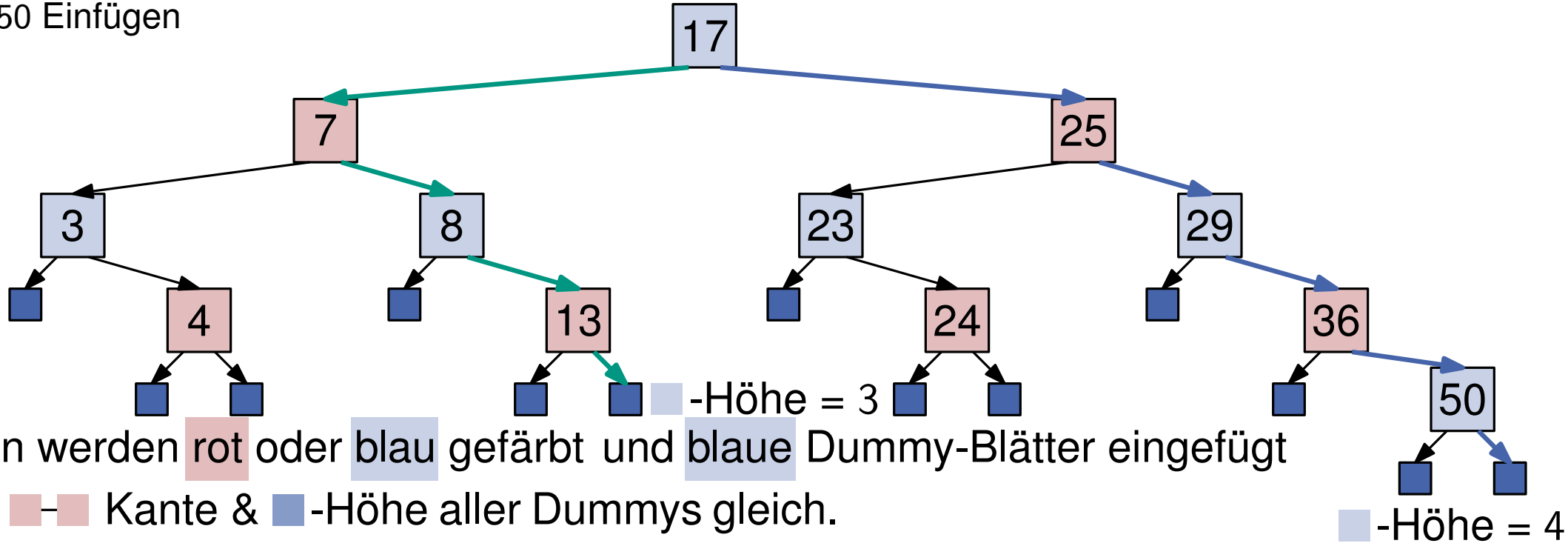


- Knoten werden rot oder blau gefärbt und blaue Dummy-Blätter eingefügt
- Keine – Kante & -Höhe aller Dummys gleich.

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.

Beispiel: 50 Einfügen

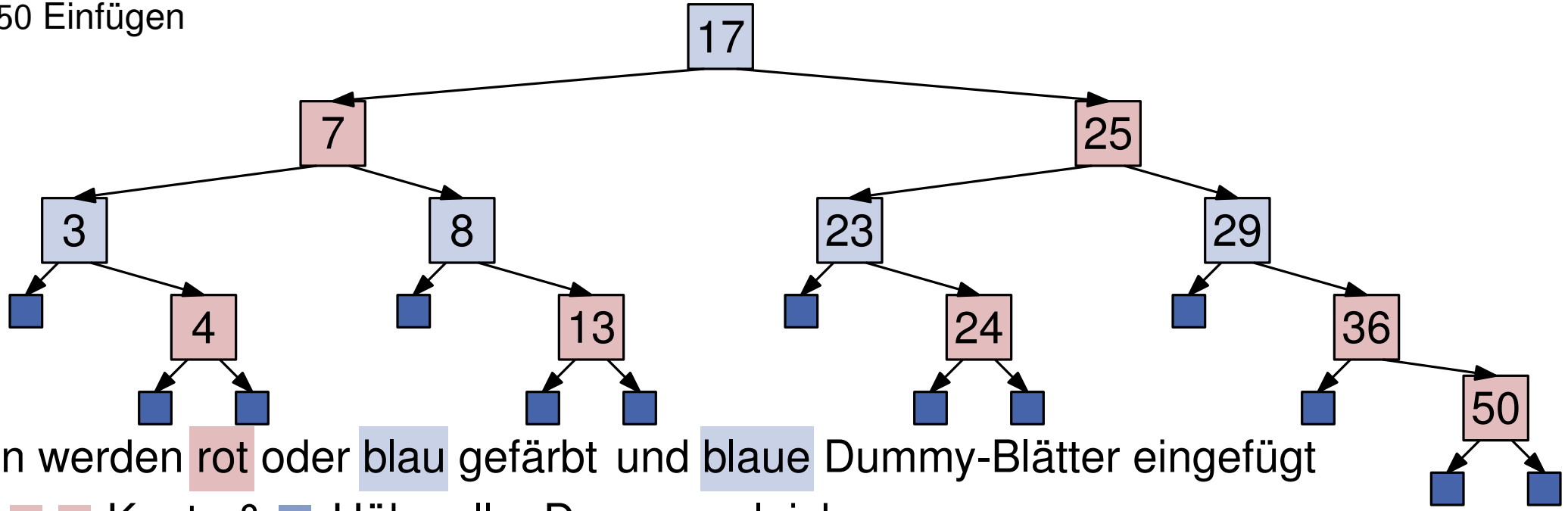


- Knoten werden rot oder blau gefärbt und blaue Dummy-Blätter eingefügt
- Keine – Kante & -Höhe aller Dummys gleich.

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.

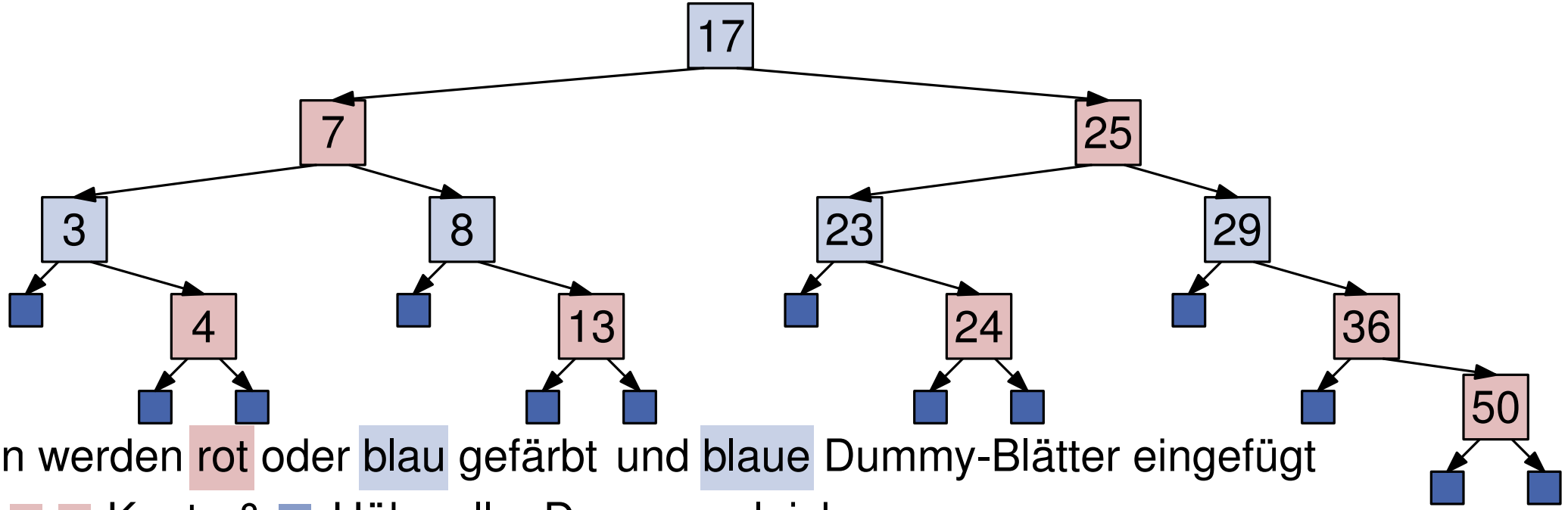
Beispiel: 50 Einfügen



- Knoten werden rot oder blau gefärbt und blaue Dummy-Blätter eingefügt
- Keine – Kante & -Höhe aller Dummys gleich.

Rot-Schwarz-Bäume

- Idee: Suchbaum, bei dem wir weniger Wert auf Balance legen, dafür bitte immer binär.
- Kriterium: Das tiefste Blatt hängt höchstens **doppelt** so tief wie das flachste Blatt.

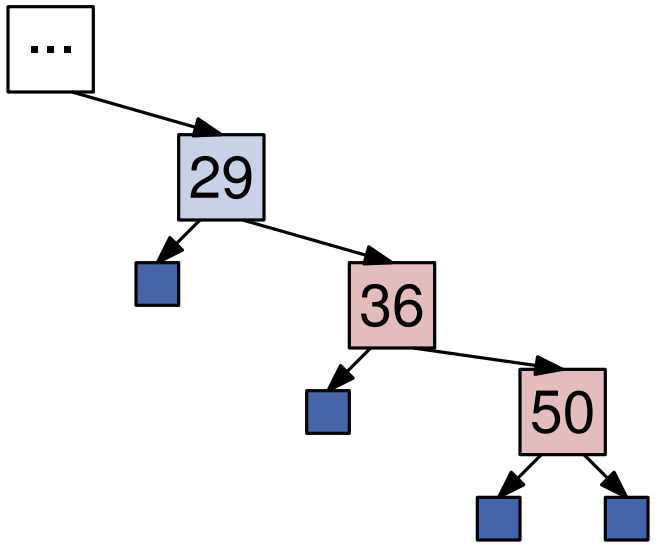


- Knoten werden rot oder blau gefärbt und blaue Dummy-Blätter eingefügt
- Keine – Kante & -Höhe aller Dummys gleich.
- Wir können 50 nicht einfügen ohne die Kriterien zu verletzen
- Es muss ausbalanciert werden!

Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.

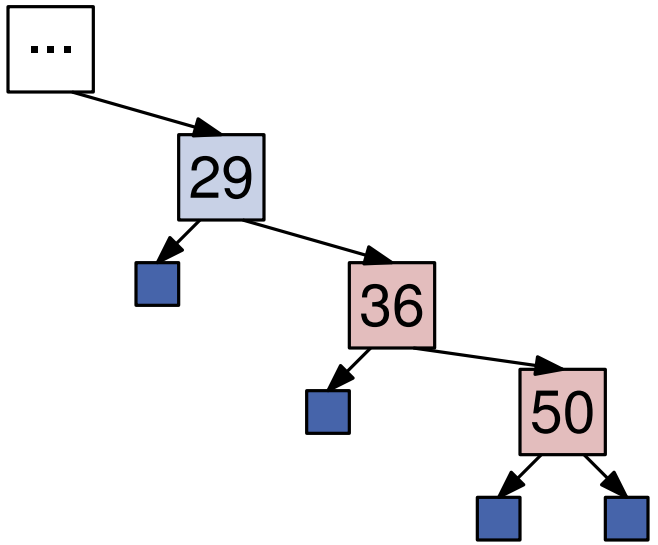
keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

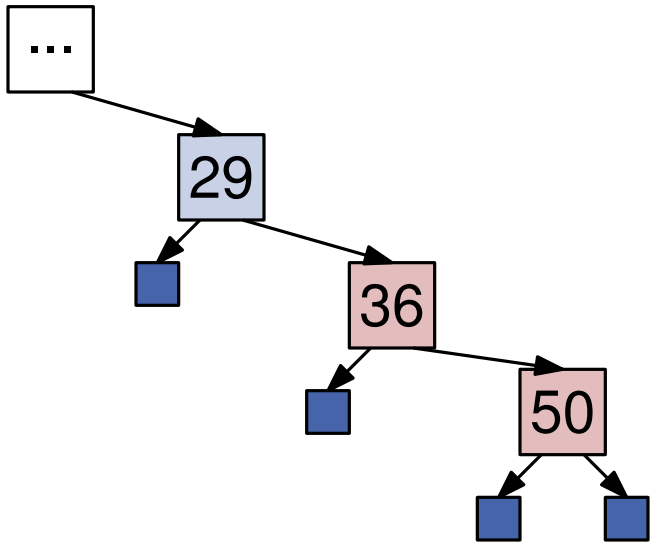


Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    trimagische Umstrukturierung

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

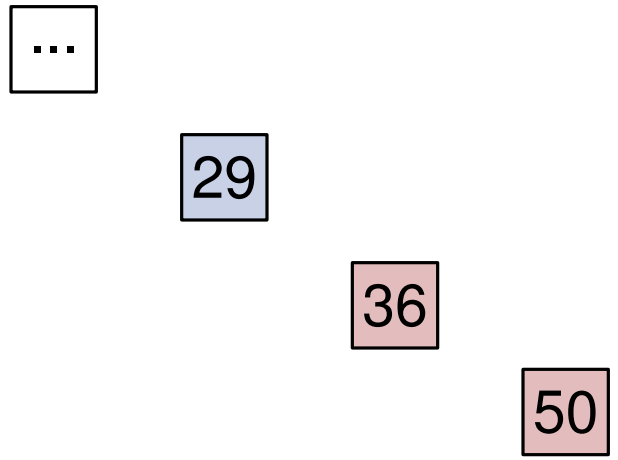


Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

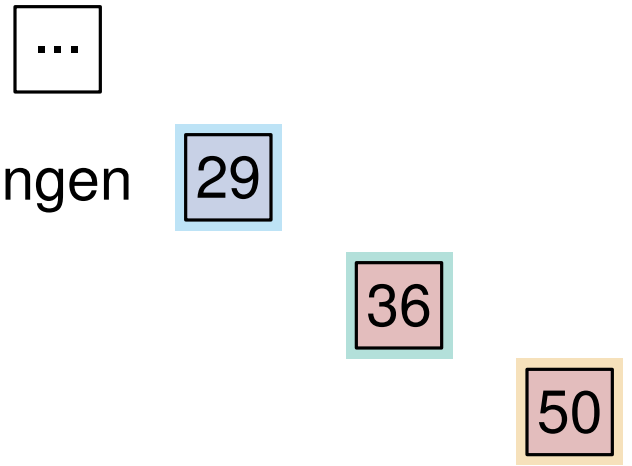


Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

...

29 36 50

Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

...

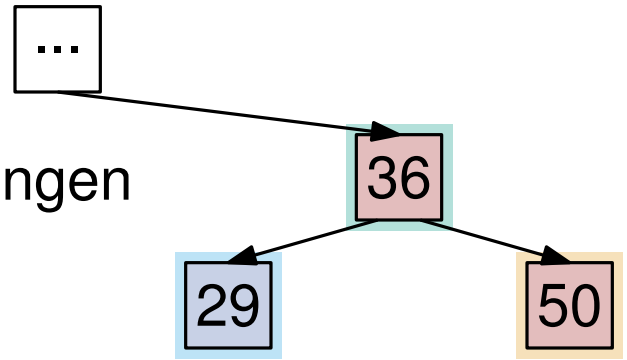
29 36 50

Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

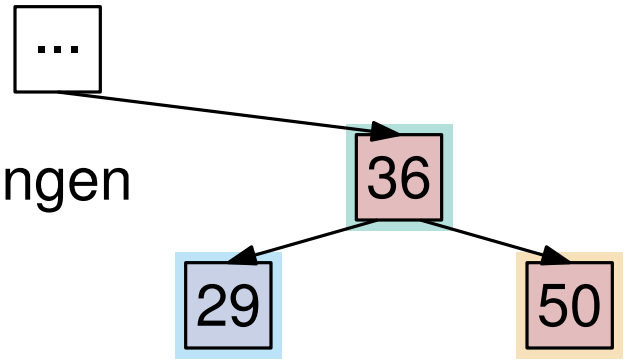


Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

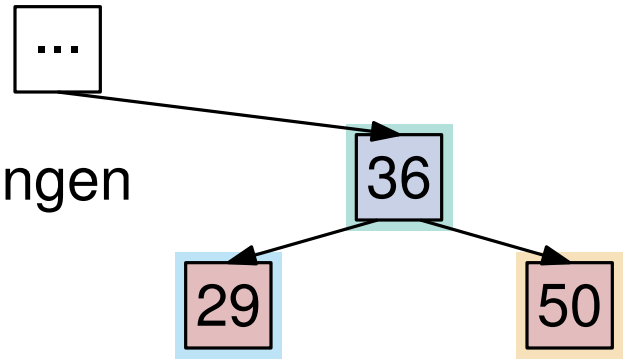


Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

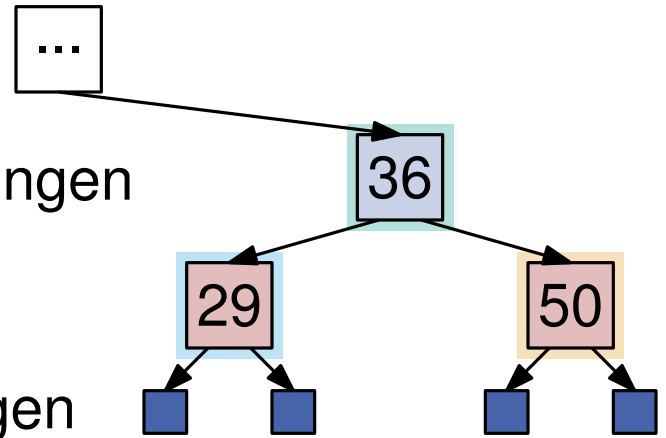


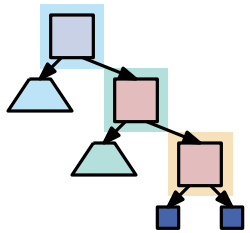
Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



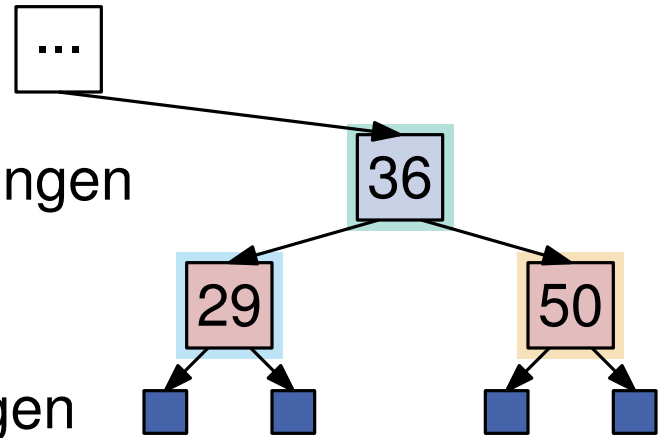


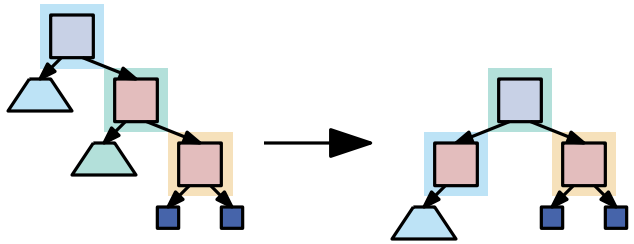
Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



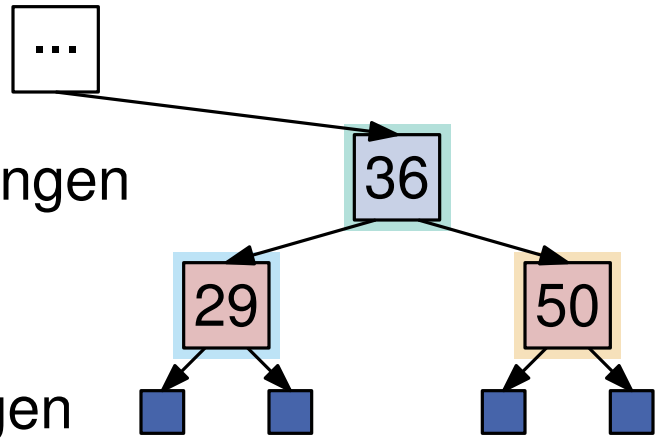


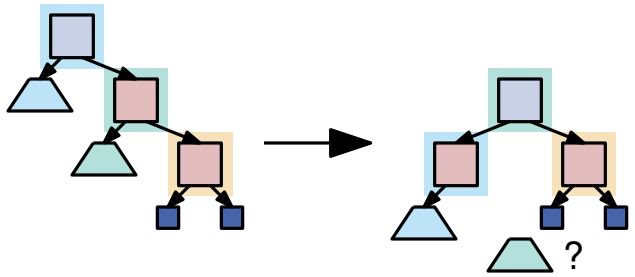
Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



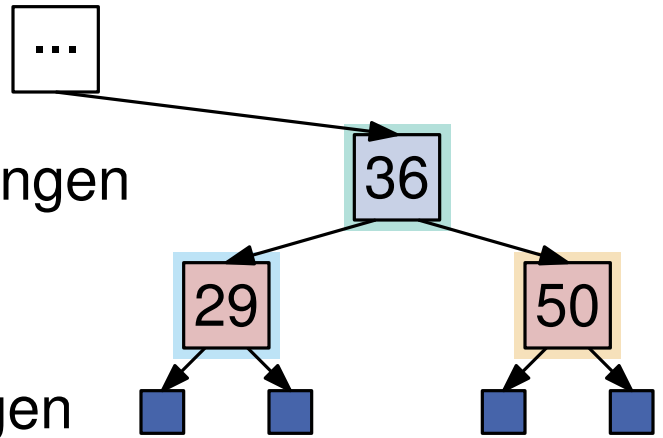


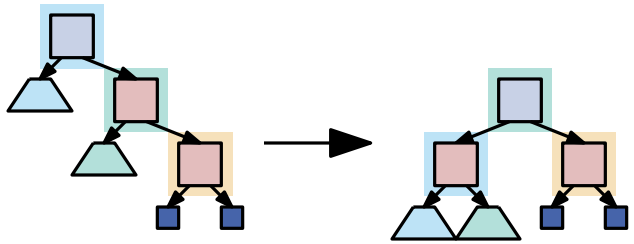
Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



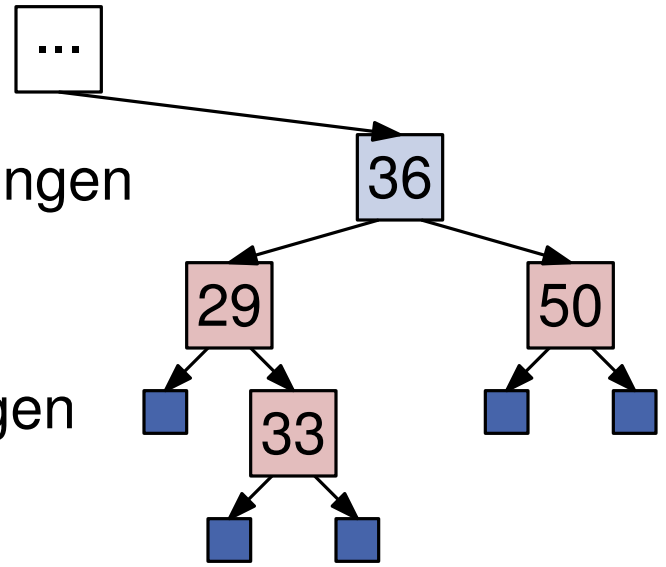


Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen
  - Fall 1b: Eltergeschwister ist rot

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

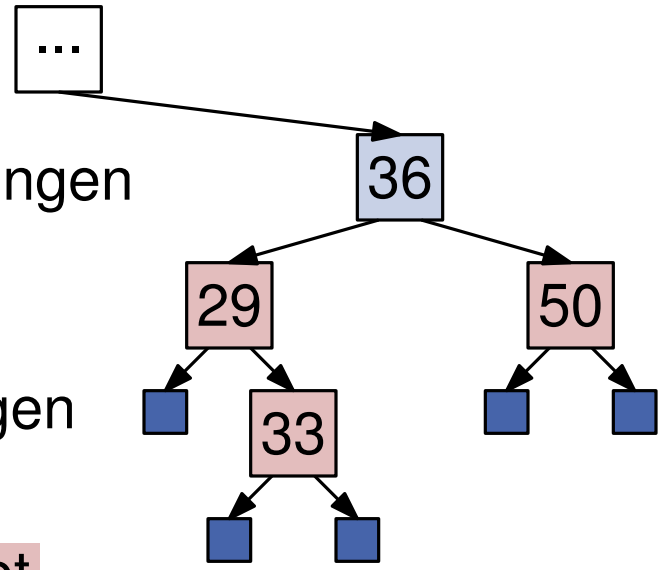


Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen
  - Fall 1b: Eltergeschwister ist rot
    - Umfärben: Elter und Eltergeschwister blau, Großelter rot

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

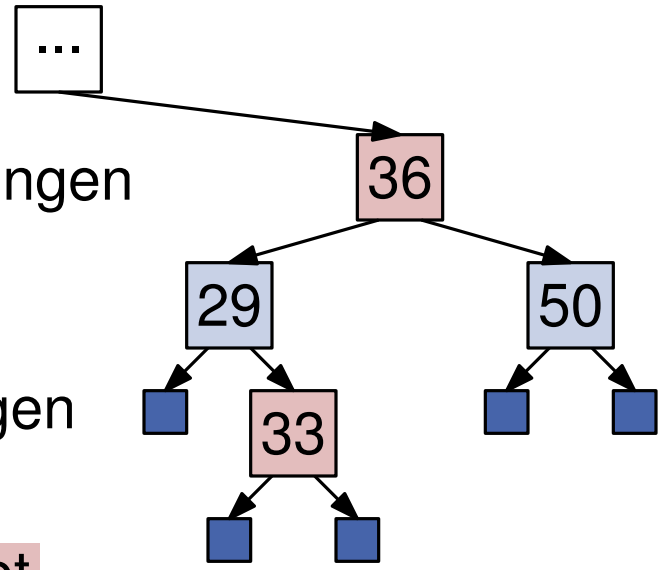


Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen
  - Fall 1b: Eltergeschwister ist rot
    - Umfärben: Elter und Eltergeschwister blau, Großelter rot

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



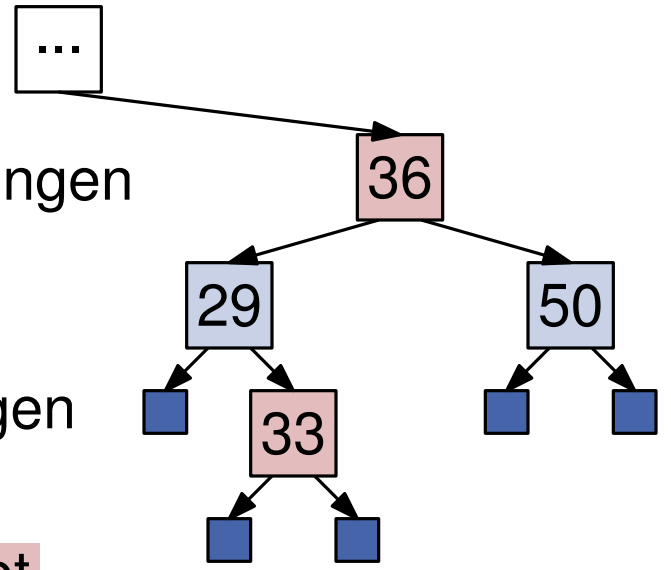
Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen
  - Fall 1b: Eltergeschwister ist rot
    - Umfärben: Elter und Eltergeschwister blau, Großelter rot

    Sind wir jetzt fertig?

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



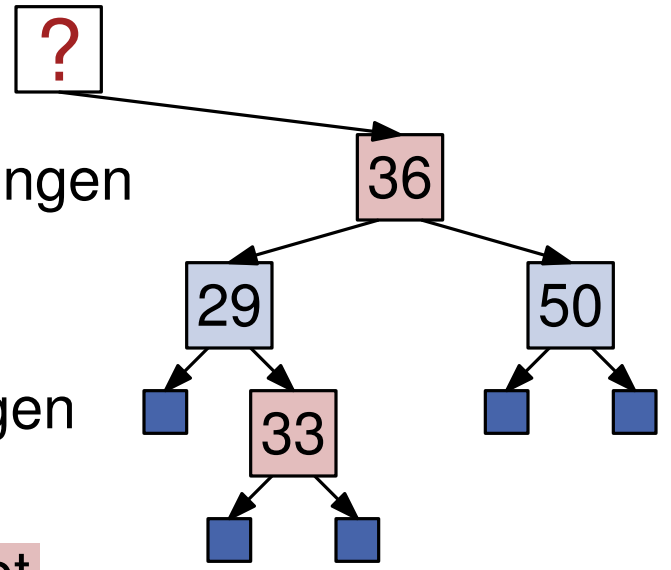
Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen
  - Fall 1b: Eltergeschwister ist rot
    - Umfärben: Elter und Eltergeschwister blau, Großelter rot

Sind wir jetzt fertig? -Höhe und Suchbaumeigenschaft erhalten, ggf. – Kante erzeugt!

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



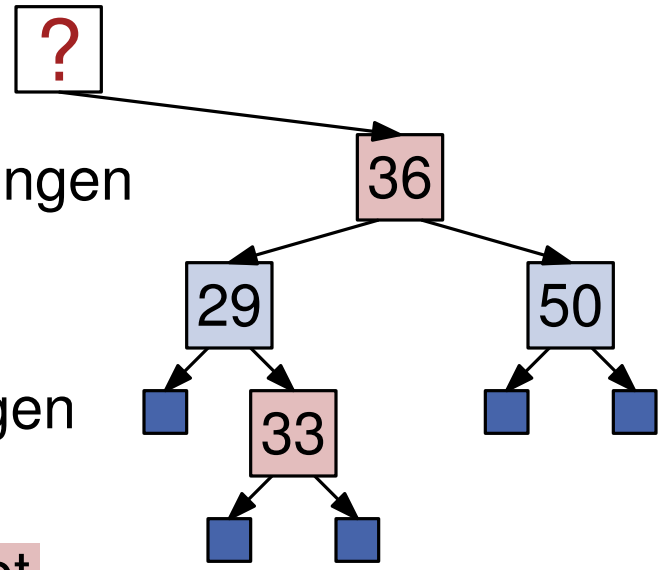
Rot-Schwarz-Bäume – Einfügen

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen
  - Fall 1b: Eltergeschwister ist rot
    - Umfärben: Elter und Eltergeschwister blau, Großelter rot

Sind wir jetzt fertig? -Höhe und Suchbaumeigenschaft erhalten, ggf. – Kante erzeugt! **Rekursion**

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



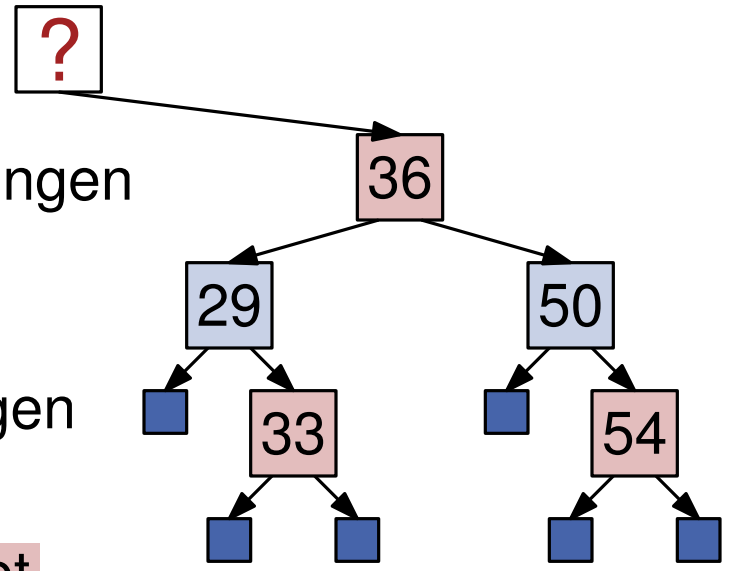
Rot-Schwarz-Bäume – Einfügen

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen
  - Fall 1b: Eltergeschwister ist rot
    - Umfärben: Elter und Eltergeschwister blau, Großelter rot

    Sind wir jetzt fertig? -Höhe und Suchbaumeigenschaft erhalten, ggf. – Kante erzeugt! **Rekursion**
- Fall 2: Elter ist blau



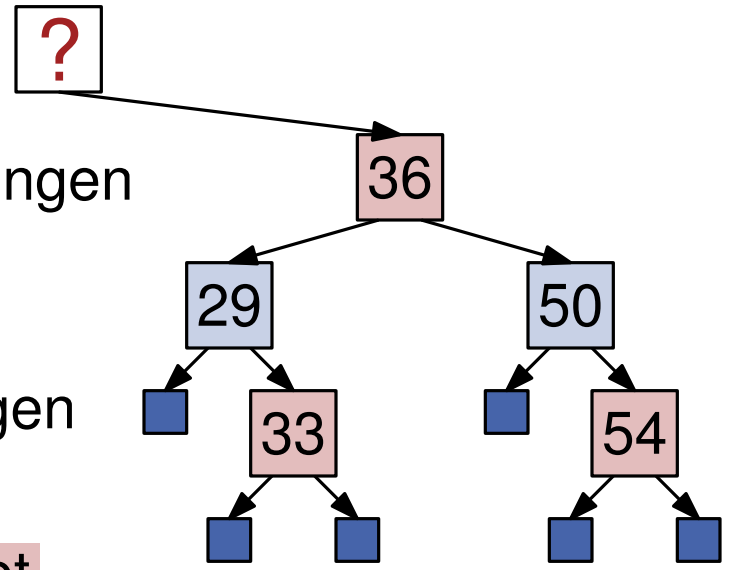
Rot-Schwarz-Bäume – Einfügen

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft

- Idee: Neues Element als rotes Blatt an richtiger Stelle einfügen und gegebenenfalls ausbalancieren.
- Fall 1: Elter ist rot
  - dann ist Großelter blau (Warum?)
  - Fall 1a: Eltergeschwister ist blau

    Drei-Knoten-Umstrukturierung
    - Kind, Elter, und Großelter in die richtige Reihenfolge bringen
    - Mittlerer Knoten wird Elter der anderen beiden
    - Mittlerer Knoten wird blau, die anderen rot
    - Teilbäume die unter den Knoten hingen korrekt umhängen
  - Fall 1b: Eltergeschwister ist rot
    - Umfärben: Elter und Eltergeschwister blau, Großelter rot

    Sind wir jetzt fertig? -Höhe und Suchbaumeigenschaft erhalten, ggf. – Kante erzeugt! **Rekursion**
- Fall 2: Elter ist blau → fertig



Rot-Schwarz-Bäume – Einfügen

- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)

Rot-Schwarz-Bäume – Einfügen

- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)
- Laufzeit
  - Blatt an richtiger Stelle anhängen

Rot-Schwarz-Bäume – Einfügen

- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)
- Laufzeit
  - Blatt an richtiger Stelle anhängen $O(\textit{Höhe des Baumes})$

Rot-Schwarz-Bäume – Einfügen

- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)
- Laufzeit
  - Blatt an richtiger Stelle anhängen
  - Umstrukturieren **oder** Umfärben
    - Umstrukturieren (höchstens 1x)

$O(\textit{Höhe des Baumes})$

Rot-Schwarz-Bäume – Einfügen

- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)
- Laufzeit
  - Blatt an richtiger Stelle anhängen — $O(\textit{Höhe des Baumes})$
  - Umstrukturieren **oder** Umfärben
    - Umstrukturieren (höchstens 1x) — $O(1)$

Rot-Schwarz-Bäume – Einfügen

- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)
- Laufzeit
  - Blatt an richtiger Stelle anhängen — $O(\textit{Höhe des Baumes})$
  - Umstrukturieren **oder** Umfärben
    - Umstrukturieren (höchstens 1x) — $O(1)$
    - Umfärben + Rekursion (höchstens $O(\textit{Höhe des Baumes})$)

Rot-Schwarz-Bäume – Einfügen

- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)
- Laufzeit
  - Blatt an richtiger Stelle anhängen — $O(\textit{Höhe des Baumes})$
  - Umstrukturieren **oder** Umfärben
    - Umstrukturieren (höchstens 1x) — $O(1)$
    - Umfärben + Rekursion (höchstens $O(\textit{Höhe des Baumes})$) — $O(\textit{Höhe des Baumes})$

Rot-Schwarz-Bäume – Einfügen

- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)
- Laufzeit
  - Blatt an richtiger Stelle anhängen — $O(\textit{Höhe des Baumes})$
  - Umstrukturieren **oder** Umfärben
    - Umstrukturieren (höchstens 1x) — $O(1)$
    - Umfärben + Rekursion (höchstens $O(\textit{Höhe des Baumes})$) — $O(\textit{Höhe des Baumes})$

$$\overline{O(\textit{Höhe des Baumes})}$$

Rot-Schwarz-Bäume – Einfügen

- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)
- Laufzeit
  - Blatt an richtiger Stelle anhängen — $O(\textit{Höhe des Baumes})$
  - Umstrukturieren **oder** Umfärben
    - Umstrukturieren (höchstens 1x) — $O(1)$
    - Umfärben + Rekursion (höchstens $O(\textit{Höhe des Baumes})$) — $O(\textit{Höhe des Baumes})$

    $\overline{O(\textit{Höhe des Baumes})}$
- Löschen auch in $O(\textit{Höhe des Baumes})$

Rot-Schwarz-Bäume – Einfügen

- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)
- Laufzeit
  - Blatt an richtiger Stelle anhängen — $O(\textit{Höhe des Baumes})$
  - Umstrukturieren **oder** Umfärben
    - Umstrukturieren (höchstens 1x) — $O(1)$
    - Umfärben + Rekursion (höchstens $O(\textit{Höhe des Baumes})$) — $O(\textit{Höhe des Baumes})$

    $\overline{O(\textit{Höhe des Baumes})}$
- Löschen auch in $O(\textit{Höhe des Baumes})$

noch umständlicher

Rot-Schwarz-Bäume – Einfügen

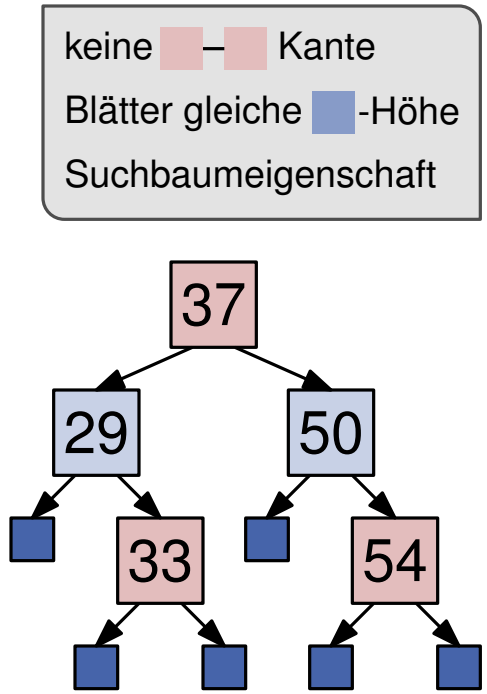
- Diverse Fallunterscheidungen (Welche Eltergeschwister existieren? Wie werden zugehörige Teilbäume umgehangen? etc.)
- Laufzeit
  - Blatt an richtiger Stelle anhängen — $O(\textit{Höhe des Baumes})$
  - Umstrukturieren **oder** Umfärben
    - Umstrukturieren (höchstens 1x) — $O(1)$
    - Umfärben + Rekursion (höchstens $O(\textit{Höhe des Baumes})$) — $O(\textit{Höhe des Baumes})$

    $\overline{O(\textit{Höhe des Baumes})}$
- Löschen auch in $O(\textit{Höhe des Baumes})$

  noch umständlicher
- Aber was ist denn nun die Höhe des Baumes?
- Hat das mit den Farben überhaupt funktioniert?

Rot-Schwarz-Bäume – Höhe

- Wir wollen, dass die Pfade in unserem Baum (abhängig von n) nicht zu lang werden können
- Oder anders herum: wenn es lange Pfade gibt, wollen wir auch viele Knoten haben

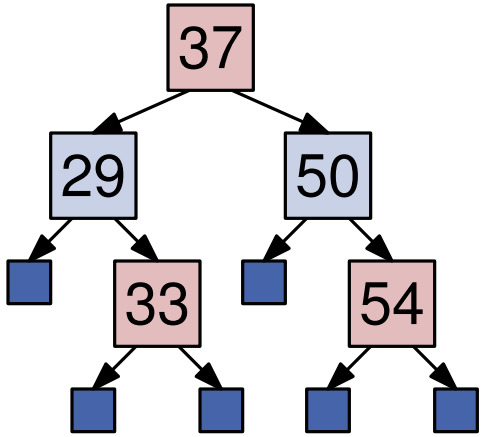


Rot-Schwarz-Bäume – Höhe

- Wir wollen, dass die Pfade in unserem Baum (abhängig von n) nicht zu lang werden können
- Oder anders herum: wenn es lange Pfade gibt, wollen wir auch viele Knoten haben

- Gegeben ein Rot-Schwarz Baum T mit ■-Höhe h.
  Wie viele Knoten hat T mindestens?

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft

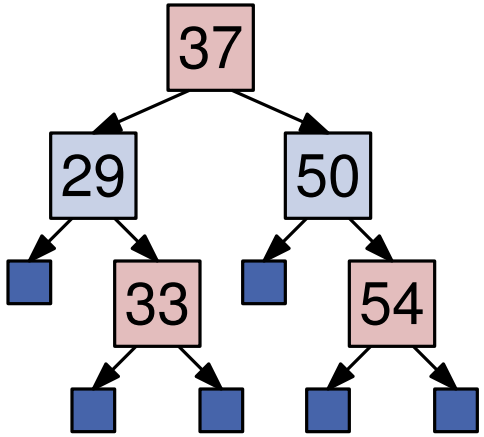


Rot-Schwarz-Bäume – Höhe

- Wir wollen, dass die Pfade in unserem Baum (abhängig von n) nicht zu lang werden können
- Oder anders herum: wenn es lange Pfade gibt, wollen wir auch viele Knoten haben

- Gegeben ein Rot-Schwarz Baum T mit ■-Höhe h. Wie viele Knoten hat T mindestens?
- Wie lang ist der längste Pfad von der Wurzel zu einem Blatt?

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft

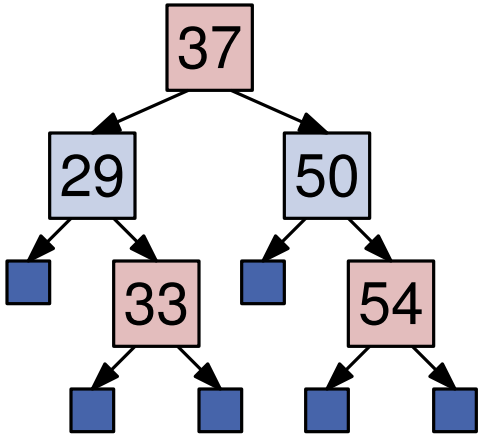


Rot-Schwarz-Bäume – Höhe

- Wir wollen, dass die Pfade in unserem Baum (abhängig von n) nicht zu lang werden können
- Oder anders herum: wenn es lange Pfade gibt, wollen wir auch viele Knoten haben

- Gegeben ein Rot-Schwarz Baum T mit ■-Höhe h. Wie viele Knoten hat T mindestens?
- Wie lang ist der längste Pfad von der Wurzel zu einem Blatt?
- Wie viele Knoten hat T maximal?

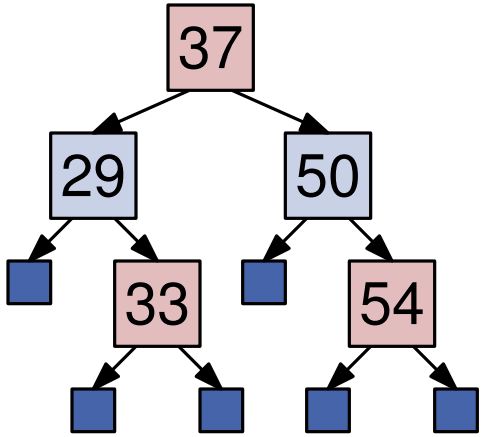
keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



Rot-Schwarz-Bäume – Höhe

- Wir wollen, dass die Pfade in unserem Baum (abhängig von n) nicht zu lang werden können
- Oder anders herum: wenn es lange Pfade gibt, wollen wir auch viele Knoten haben

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft

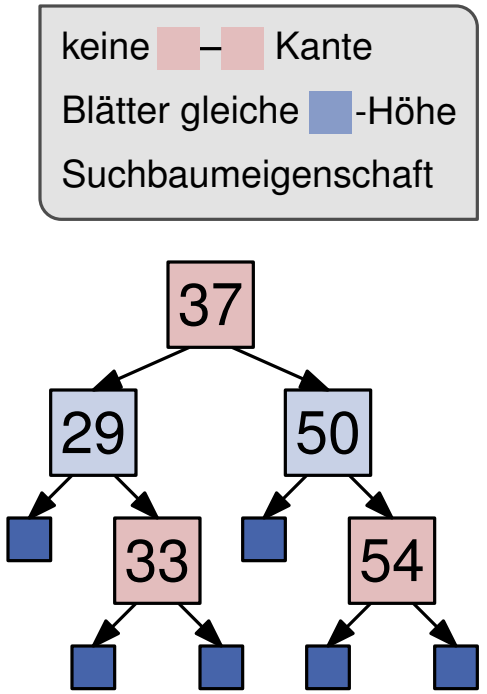


- Gegeben ein Rot-Schwarz Baum T mit ■-Höhe h. Wie viele Knoten hat T mindestens?
- Wie lang ist der längste Pfad von der Wurzel zu einem Blatt?
- Wie viele Knoten hat T maximal?

pwa.klicker.uzh.ch/join/algo1

Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.



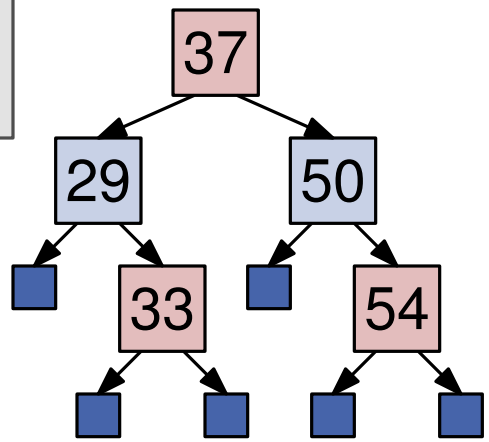
Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit ■-Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



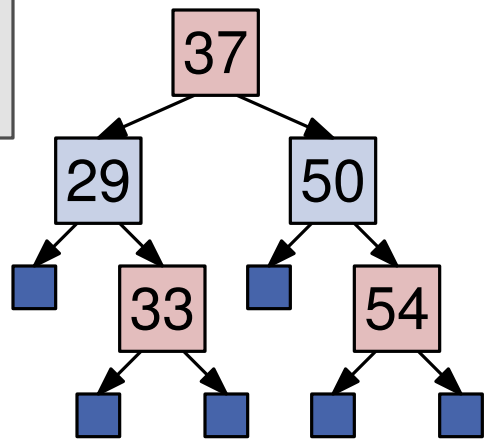
Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit ■-Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



Beweis über Induktion

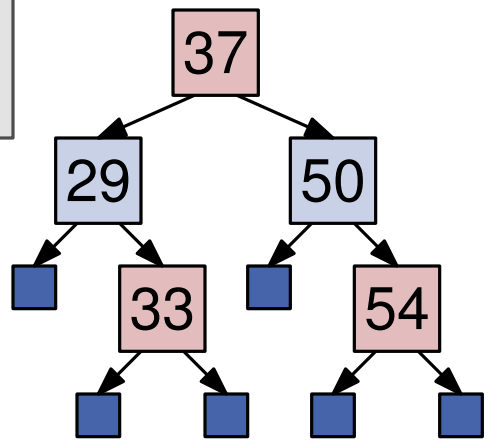
- Anfang: $h = 1$ ■

$$n = 1 \geq 2^1 - 1 = 1$$

Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



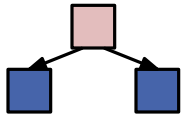
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit -Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

Beweis über Induktion

- Anfang: $h = 1$

$n = 1 \geq 2^1 - 1 = 1$

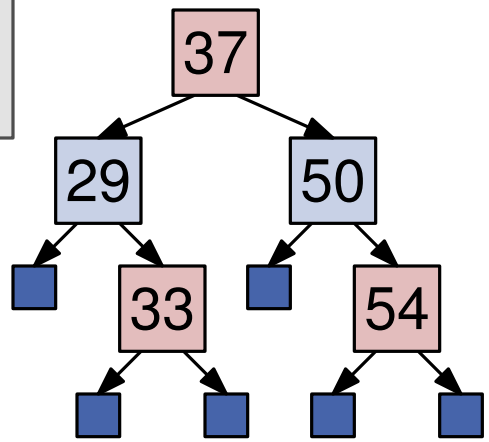


$n = 3 \geq 2^1 - 1 = 1$

Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



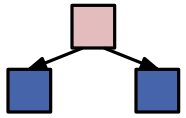
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit -Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

Beweis über Induktion

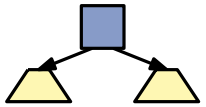
- Anfang: $h = 1$

$n = 1 \geq 2^1 - 1 = 1$



$n = 3 \geq 2^1 - 1 = 1$

- Schritt: $h + 1$

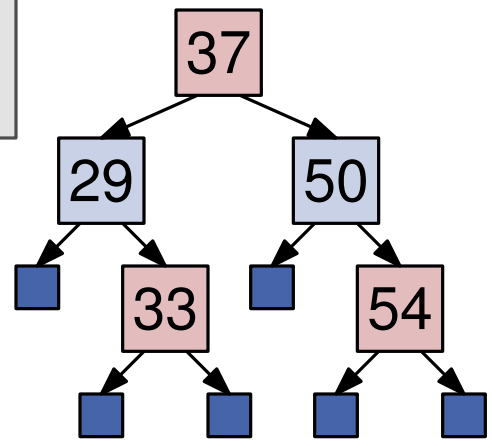


Teilbäume mit -Höhe h

Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



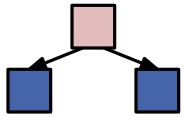
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit -Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

Beweis über Induktion

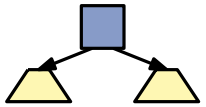
- Anfang: $h = 1$

$n = 1 \geq 2^1 - 1 = 1$



$n = 3 \geq 2^1 - 1 = 1$

- Schritt: $h + 1$



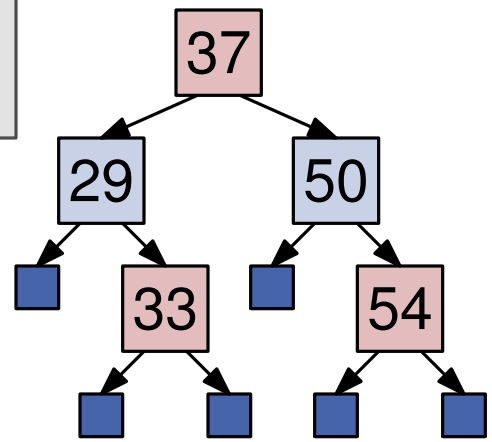
Teilbäume mit -Höhe h

$$n \geq 2 \cdot (2^h - 1) + 1$$

Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

keine ■ – ■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



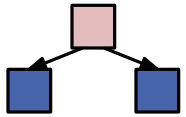
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit ■-Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

Beweis über Induktion

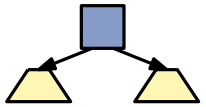
- Anfang: $h = 1$

$$n = 1 \geq 2^1 - 1 = 1$$



$$n = 3 \geq 2^1 - 1 = 1$$

- Schritt: $h + 1$



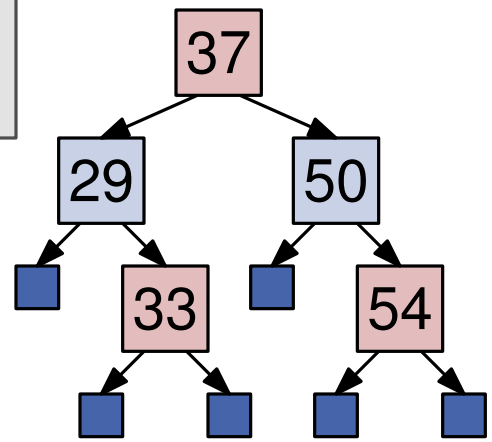
Teilbäume mit ■-Höhe h

$$n \geq 2 \cdot (2^h - 1) + 1$$
$$= 2^{h+1} - 1$$

Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



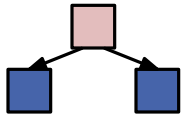
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit -Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

Beweis über Induktion

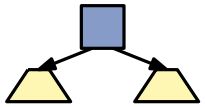
- Anfang: $h = 1$

$n = 1 \geq 2^1 - 1 = 1$

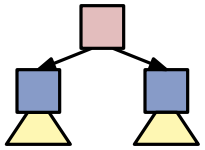


$n = 3 \geq 2^1 - 1 = 1$

- Schritt: $h + 1$



Teilbäume mit -Höhe h

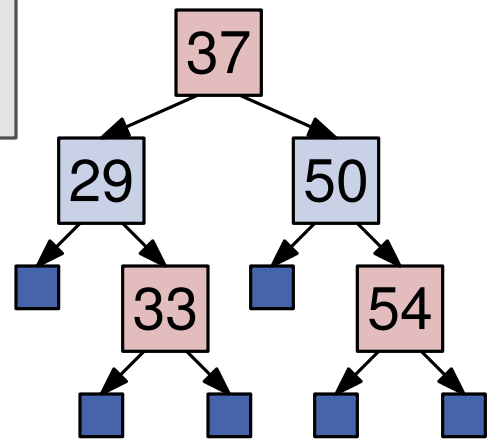


$$n \geq 2 \cdot (2^h - 1) + 1$$
$$= 2^{h+1} - 1$$

Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



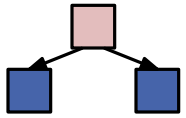
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit -Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

Beweis über Induktion

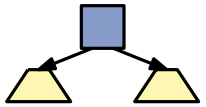
- Anfang: $h = 1$

$n = 1 \geq 2^1 - 1 = 1$

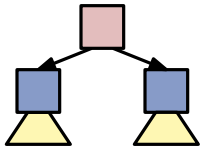


$n = 3 \geq 2^1 - 1 = 1$

- Schritt: $h + 1$



Teilbäume mit -Höhe h



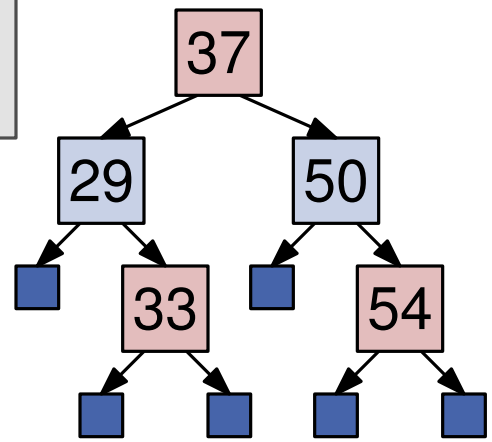
$n \geq 2 \cdot (2^h - 1) + 1$
$= 2^{h+1} - 1$

$n \geq 2 \cdot (2^h - 1) + 2 + 1$

Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



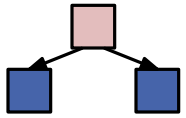
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit -Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

Beweis über Induktion

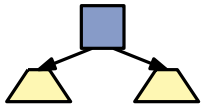
- Anfang: $h = 1$

$n = 1 \geq 2^1 - 1 = 1$

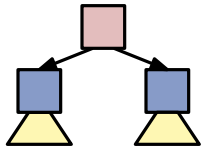


$n = 3 \geq 2^1 - 1 = 1$

- Schritt: $h + 1$



Teilbäume mit -Höhe h



$n \geq 2 \cdot (2^h - 1) + 1$
$= 2^{h+1} - 1$

$n \geq 2 \cdot (2^h - 1) + 2 + 1$
$= 2^{h+1} + 1$

Rot-Schwarz-Bäume – Höhe

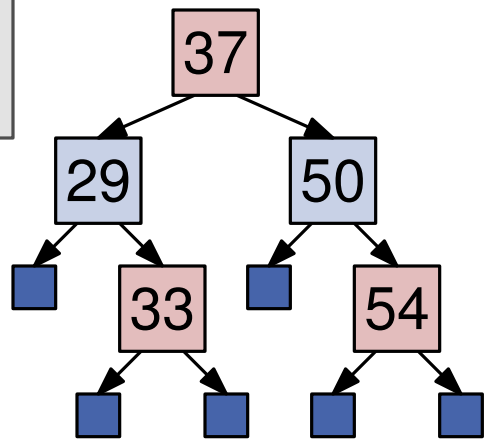
- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit ■-Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

$2^h - 1 \leq n$

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



Rot-Schwarz-Bäume – Höhe

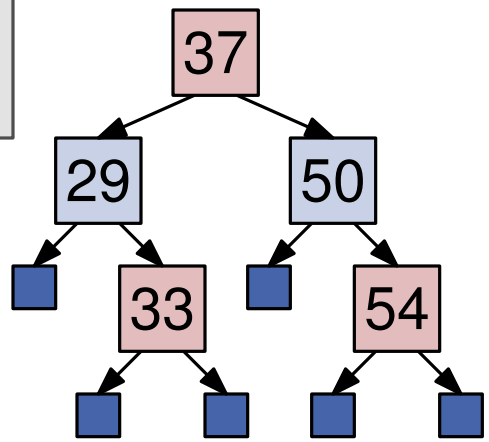
- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit ■-Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

$2^h - 1 \leq n \rightarrow 2^h \leq n + 1$

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



Rot-Schwarz-Bäume – Höhe

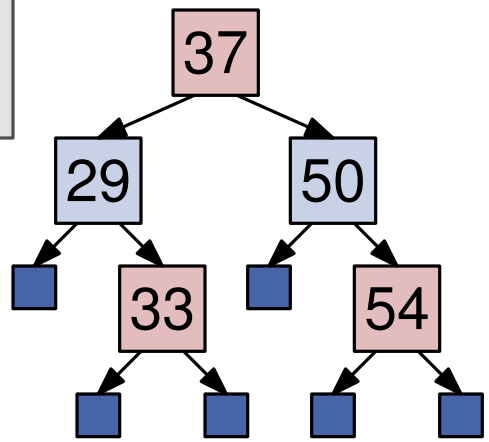
- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit ■-Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

$$2^h - 1 \leq n \rightarrow 2^h \leq n + 1 \rightarrow h \leq \log(n+1)$$

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

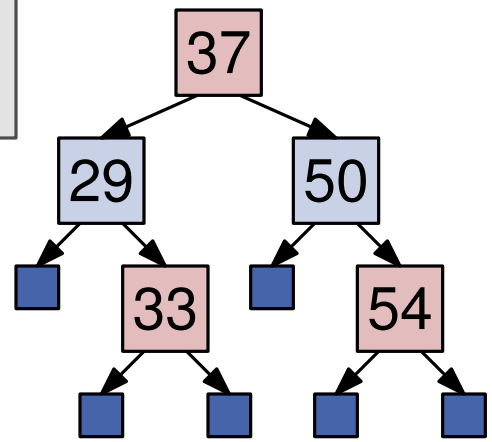
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit ■-Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

$2^h - 1 \leq n \rightarrow 2^h \leq n + 1 \rightarrow h \leq \log(n + 1)$

- Beobachtung 2: Auf jedem Wurzel-Dummy-Pfad ist mindestens die Hälfte der Knoten blau. (Dummy ist blau & keine ■–■ Kante)

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

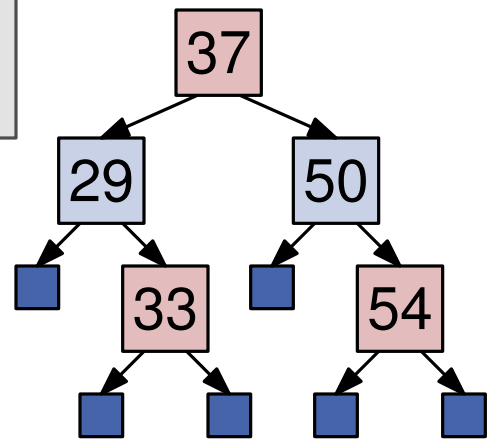
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit ■-Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

$2^h - 1 \leq n \rightarrow 2^h \leq n + 1 \rightarrow h \leq \log(n + 1)$

- Beobachtung 2: Auf jedem Wurzel-Dummy-Pfad ist mindestens die Hälfte der Knoten blau. (Dummy ist blau & keine ■–■ Kante)
  - Für die Höhe H des Baumes gilt: $H \leq 2 \cdot h$

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

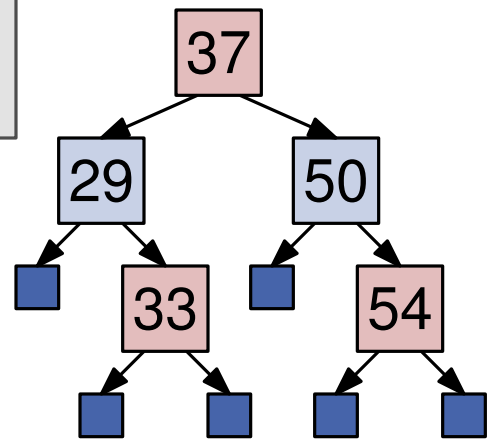
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit ■-Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

$2^h - 1 \leq n \rightarrow 2^h \leq n + 1 \rightarrow h \leq \log(n+1)$

- Beobachtung 2: Auf jedem Wurzel-Dummy-Pfad ist mindestens die Hälfte der Knoten blau. (Dummy ist blau & keine ■–■ Kante)
  - Für die Höhe H des Baumes gilt: $H \leq 2 \cdot h$

keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

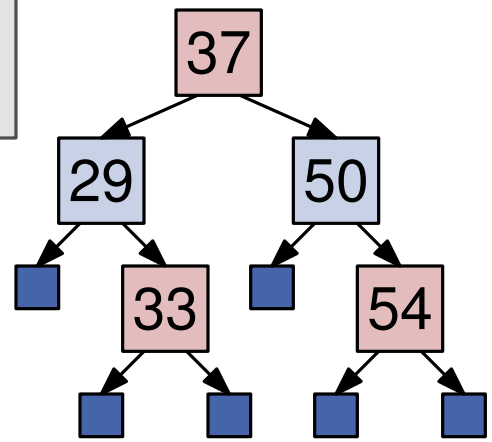
Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit ■-Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

$2^h - 1 \leq n \rightarrow 2^h \leq n + 1 \rightarrow h \leq \log(n+1)$

- Beobachtung 2: Auf jedem Wurzel-Dummy-Pfad ist mindestens die Hälfte der Knoten blau. (Dummy ist blau & keine ■–■ Kante)
  - Für die Höhe H des Baumes gilt: $H \leq 2 \cdot h \leq 2 \cdot \log(n+1)$

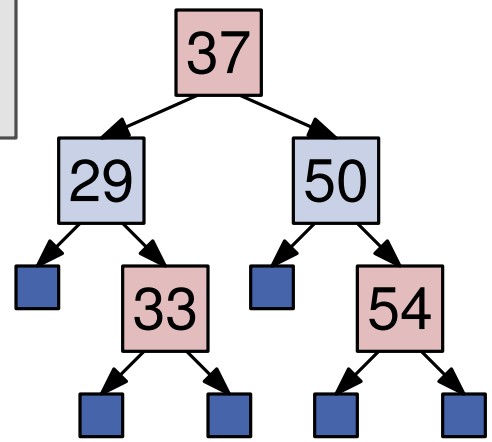
keine ■–■ Kante
Blätter gleiche ■-Höhe
Suchbaumeigenschaft



Rot-Schwarz-Bäume – Höhe

- Beobachtung 1: Der Teilbaum eines Knotens in einem Rot-Schwarz-Baum ist wieder ein Rot-Schwarz-Baum.

keine – Kante
Blätter gleiche -Höhe
Suchbaumeigenschaft



Lemma *Rot-Schwarz-Baum – Anzahl Knoten*

Sei T ein Rot-Schwarz-Baum mit -Höhe h. Dann hat T mindestens $n \geq 2^h - 1$ Knoten.

$2^h - 1 \leq n \rightarrow 2^h \leq n + 1 \rightarrow h \leq \log(n+1)$

- Beobachtung 2: Auf jedem Wurzel-Dummy-Pfad ist mindestens die Hälfte der Knoten blau. (Dummy ist blau & keine – Kante)
  - Für die Höhe H des Baumes gilt: $H \leq 2 \cdot h \leq 2 \cdot \log(n+1)$

Lemma *Rot-Schwarz-Baum – Höhe*

Sei T ein Rot-Schwarz-Baum mit n Knoten. Dann hat T eine Höhe von $H \leq 2\log(n+1)$.

Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume

Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden

Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert

Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert

marvel-filme.fandom.com/de/wiki/Thanos

Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert
  - Rot-Schwarz-Bäume: binär, dafür **ausreichend** balanciert

Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert
  - Rot-Schwarz-Bäume: binär, dafür **ausreichend** balanciert

Class TreeMap<K,V>

java.lang.Object
  java.util.AbstractMap<K,V>
    java.util.TreeMap<K,V>

Type Parameters:

`K` - the type of keys maintained by this map

`V` - the type of mapped values

All Implemented Interfaces:

`Serializable`, `Cloneable`, `Map<K,V>`, `NavigableMap<K,V>`, `SortedMap<K,V>`

```
public class TreeMap<K,V>
extends AbstractMap<K,V>
implements NavigableMap<K,V>, Cloneable, Serializable
```

A Red-Black tree based `NavigableMap` implementation. The map is sorted according to the natural ordering of its keys, or by a `Comparator` provided at map creation time, depending on which constructor is used.

This implementation provides guaranteed log(n) time cost for the `containsKey`, `get`, `put` and `remove` operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest's *Introduction to Algorithms*.

Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert
  - Rot-Schwarz-Bäume: binär, dafür **ausreichend** balanciert

Class TreeMap<K,V>

java.lang.
java.
j

Type Para

K - the

V - the

All Implen

Serializ

public c
extends
implemen

A Red-Bla
Comparat

This impl
adaptatio

std::map

Defined in header <map>

```
template<
    class Key,
    class T,
    class Compare = std::less<Key>,                                    (1)
    class Allocator = std::allocator<std::pair<const Key, T>>
> class map;
```

```
namespace pmr {
    template<
        class Key,
        class T,
        class Compare = std::less<Key>                                 (2)   (since C++17)
    > using map = std::map<Key, T, Compare,
                           std::pmr::polymorphic_allocator<std::pair<const Key, T>>>;
}
```

std::map is a sorted associative container that contains key-value pairs with unique keys. Keys are sorted by using the comparison function Compare. Search, removal, and insertion operations have logarithmic complexity. Maps are usually implemented as Red-black trees.

Suchbäume – Höhe

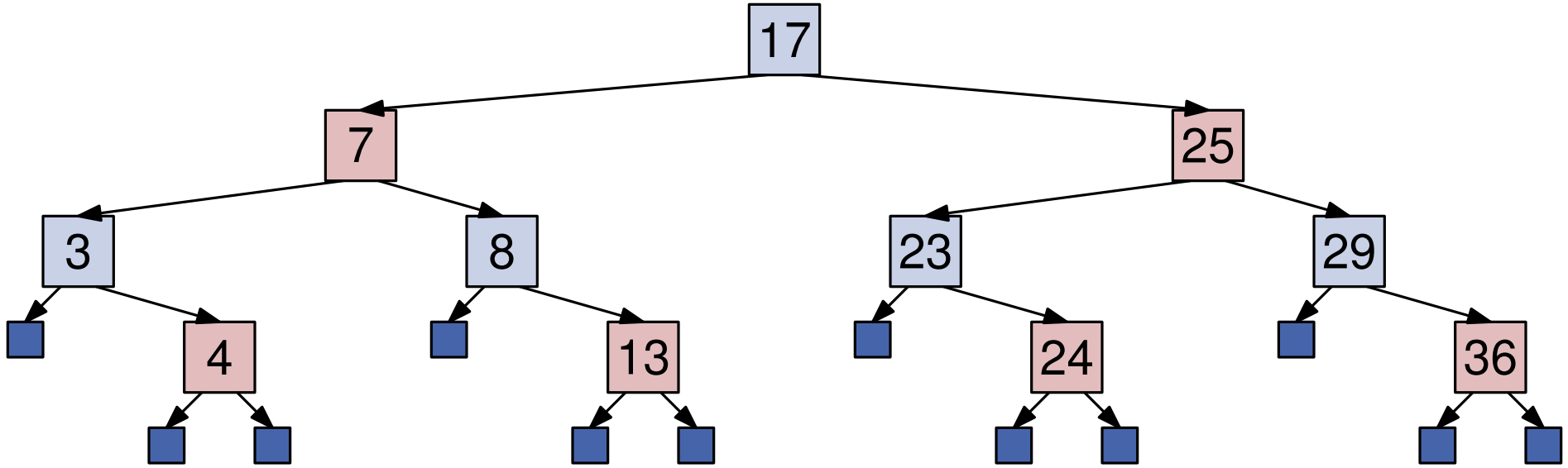
- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert
  - Rot-Schwarz-Bäume: binär, dafür **ausreichend** balanciert
- in beiden Fällen konnten wir logarithmische Höhe erreichen

Zufall?

Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert
  - Rot-Schwarz-Bäume: binär, dafür **ausreichend** balanciert
- in beiden Fällen konnten wir logarithmische Höhe erreichen
- Ziehe im Rot-Schwarz-Baum die roten Knoten zu ihren Eltern

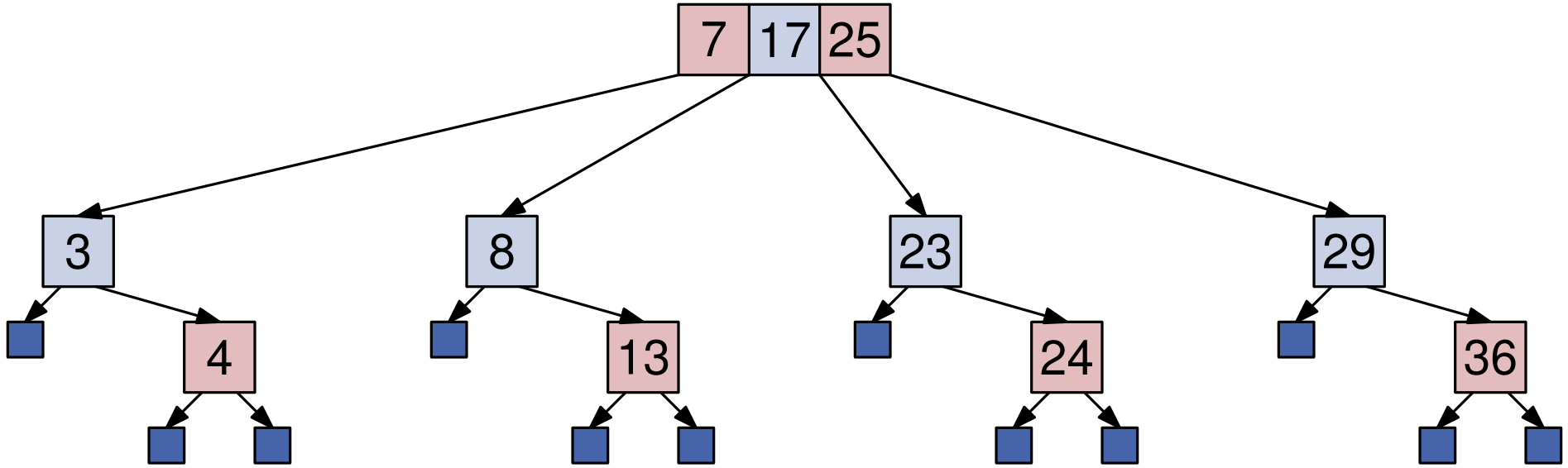
Zufall?



Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert
  - Rot-Schwarz-Bäume: binär, dafür **ausreichend** balanciert
- in beiden Fällen konnten wir logarithmische Höhe erreichen
- Ziehe im Rot-Schwarz-Baum die roten Knoten zu ihren Eltern

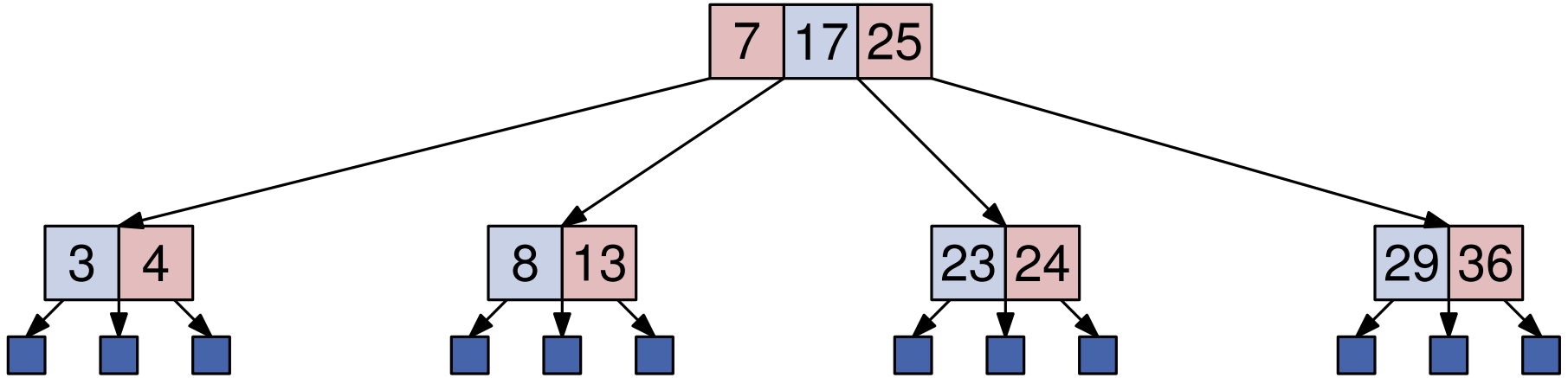
Zufall?



Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert
  - Rot-Schwarz-Bäume: binär, dafür **ausreichend** balanciert
- in beiden Fällen konnten wir logarithmische Höhe erreichen
- Ziehe im Rot-Schwarz-Baum die roten Knoten zu ihren Eltern

Zufall?

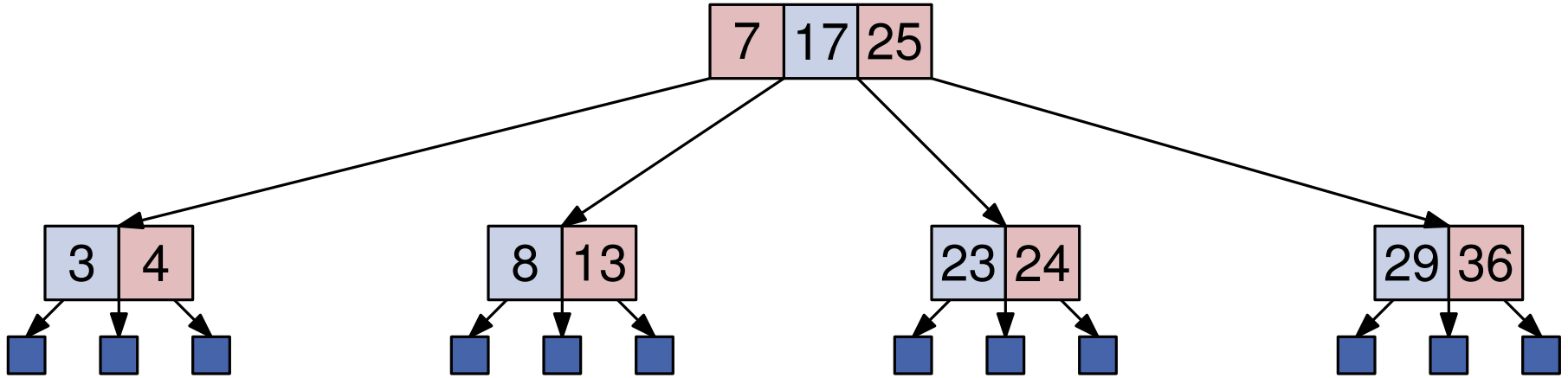


Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert
  - Rot-Schwarz-Bäume: binär, dafür **ausreichend** balanciert
- in beiden Fällen konnten wir logarithmische Höhe erreichen

Zufall?

- Ziehe im Rot-Schwarz-Baum die roten Knoten zu ihren Eltern



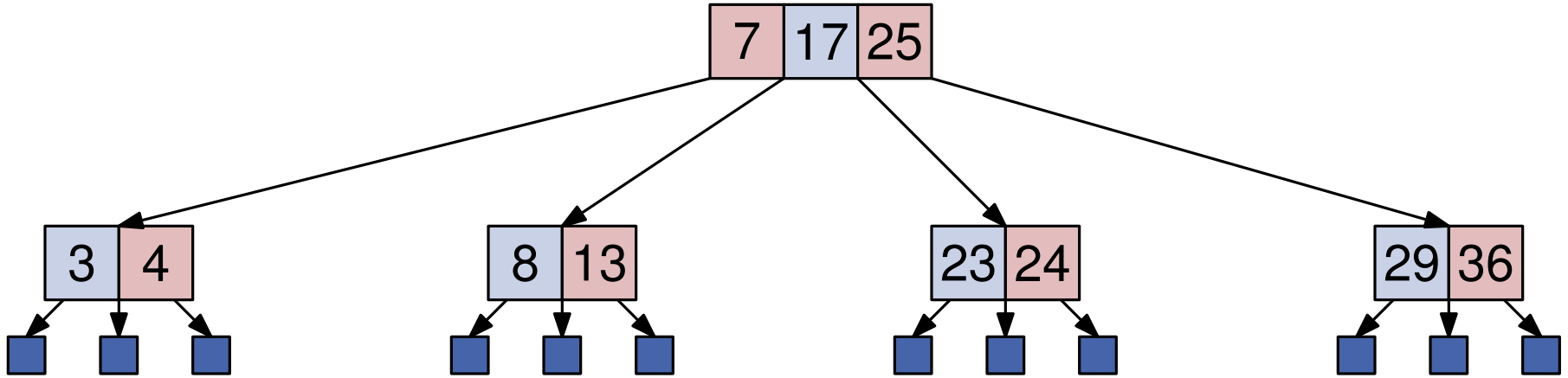
- Rot-Schwarz-Bäume haben quasi die gleiche Struktur wie $(2, 4)$-Bäume!

Suchbäume – Höhe

- Wir wünschen uns binäre und balancierte Suchbäume
- je eins der Kriterien kann etwas aufgeweicht werden
  - (a, b)-Bäume: **fast** binär, dafür perfekt balanciert
  - Rot-Schwarz-Bäume: binär, dafür **ausreichend** balanciert
- in beiden Fällen konnten wir logarithmische Höhe erreichen

Zufall?

- Ziehe im Rot-Schwarz-Baum die roten Knoten zu ihren Eltern



- Rot-Schwarz-Bäume haben quasi die gleiche Struktur wie $(2, 4)$-Bäume!

Aber irgendwie ist uns die Listenstruktur in den Blättern verloren gegangen...

Tiefensuche

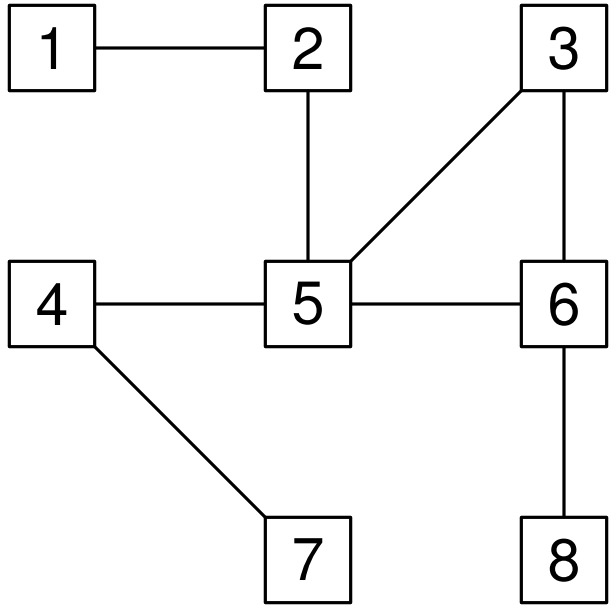
- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger

Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

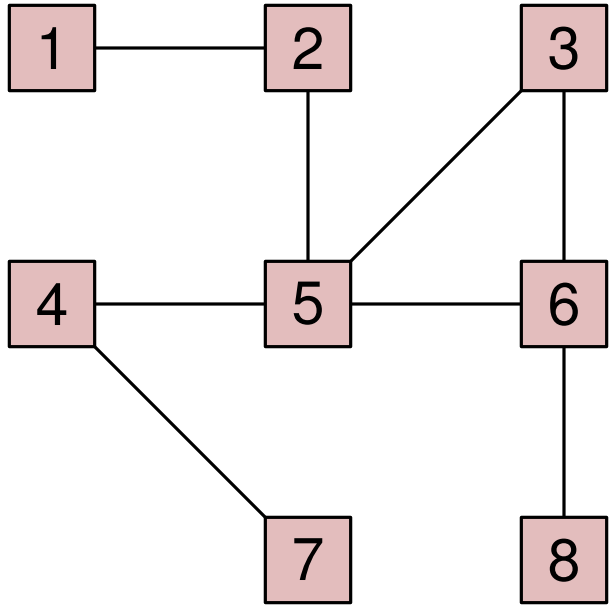
Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig



Tiefensuche

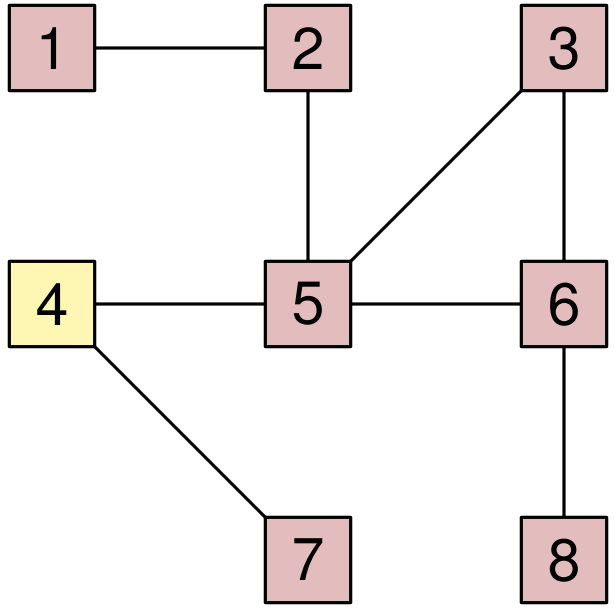
- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

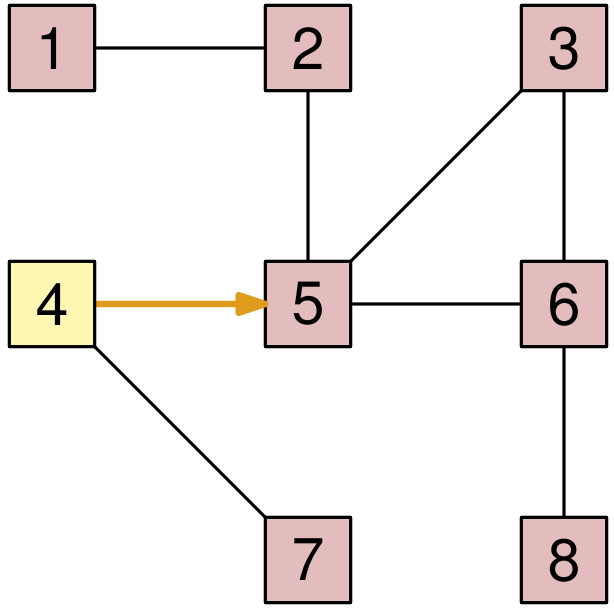
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

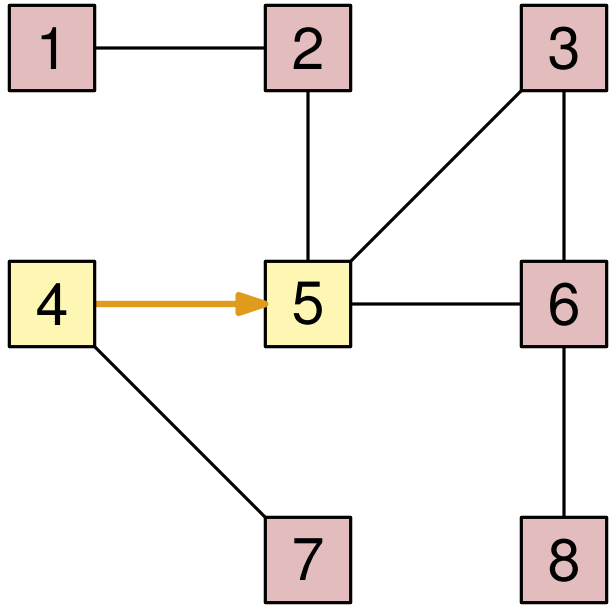
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

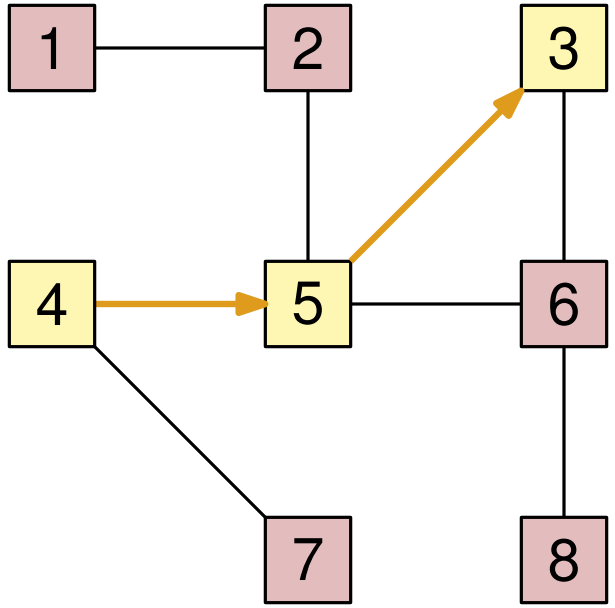
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

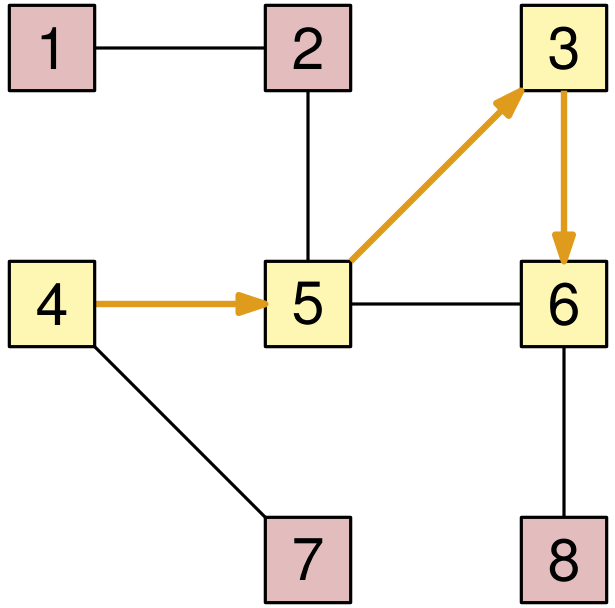
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

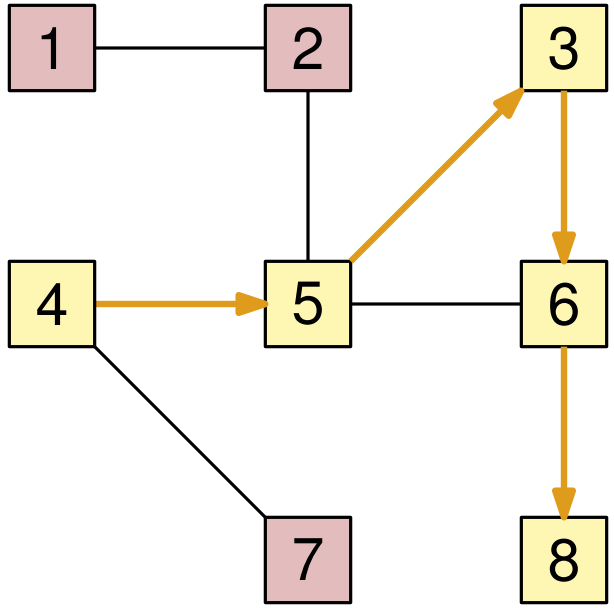
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

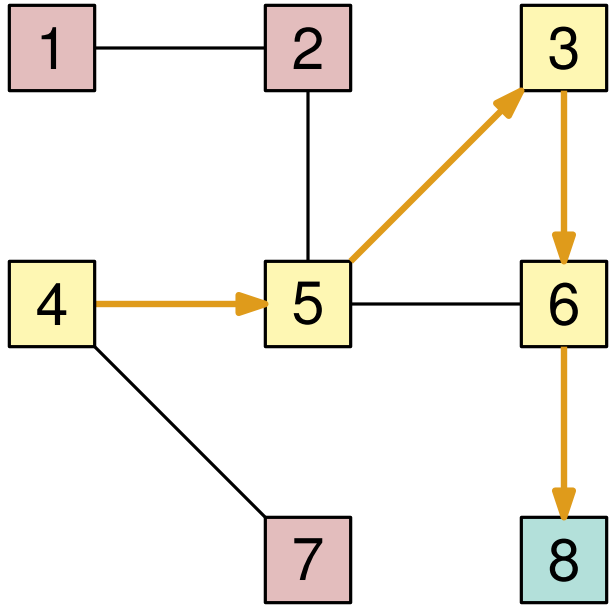
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

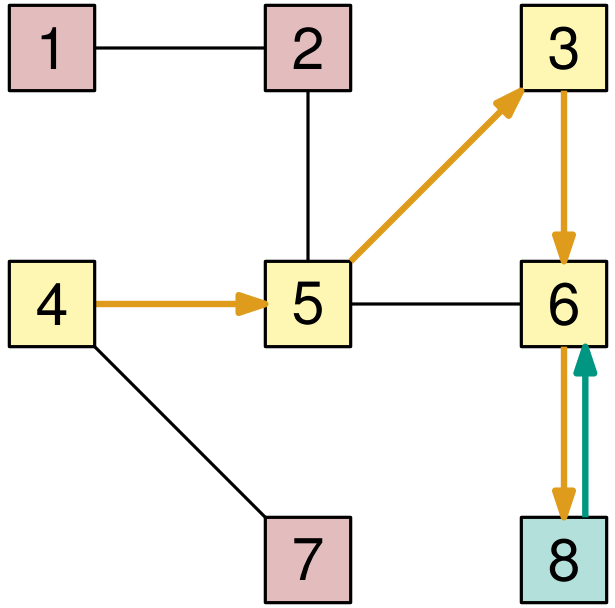
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

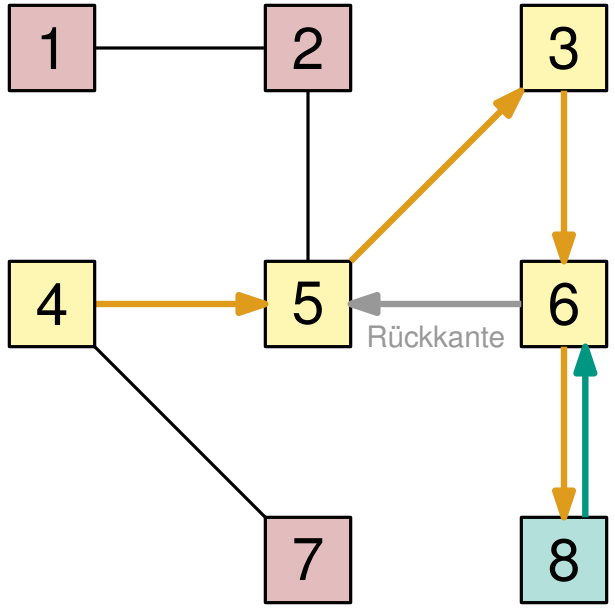
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

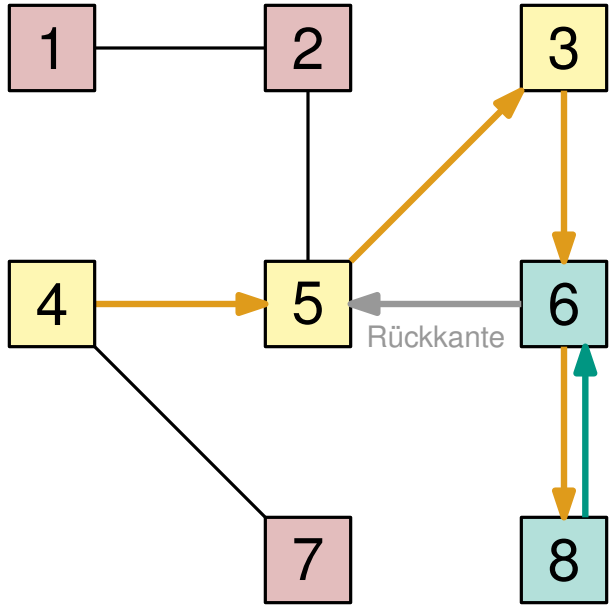
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

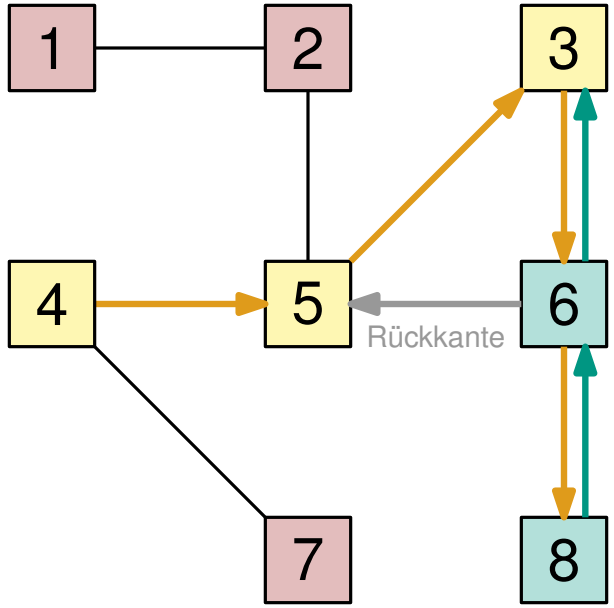
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem ungesehenen neuen Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

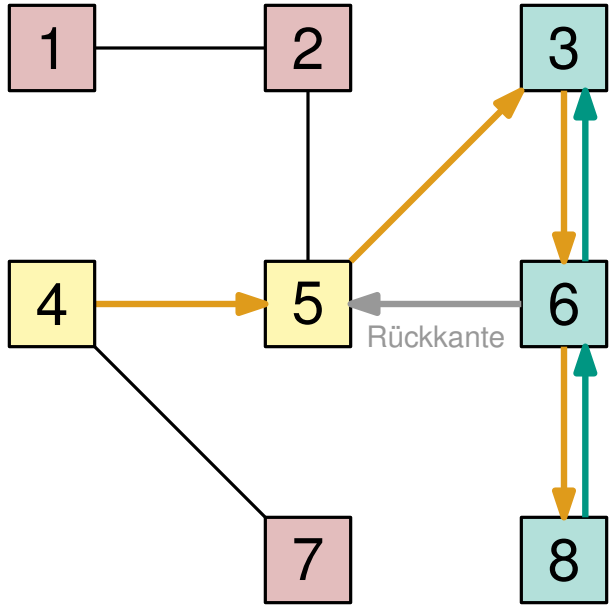
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

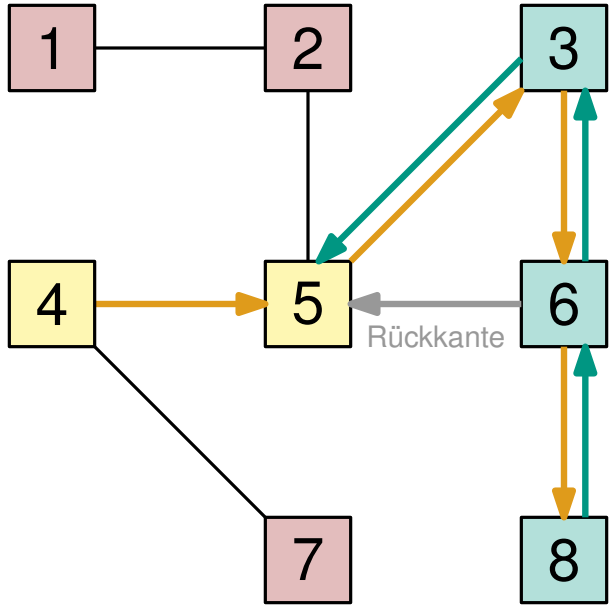
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

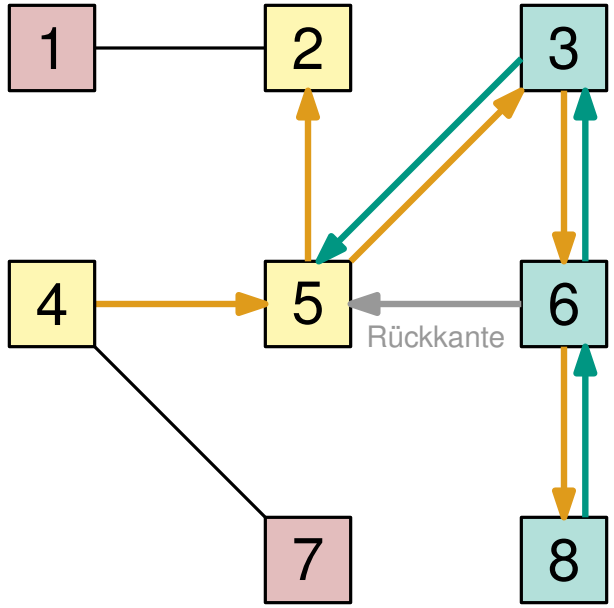
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

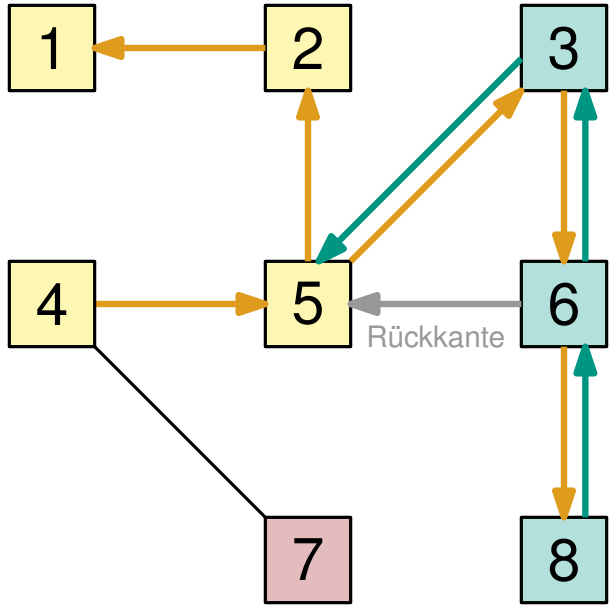
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

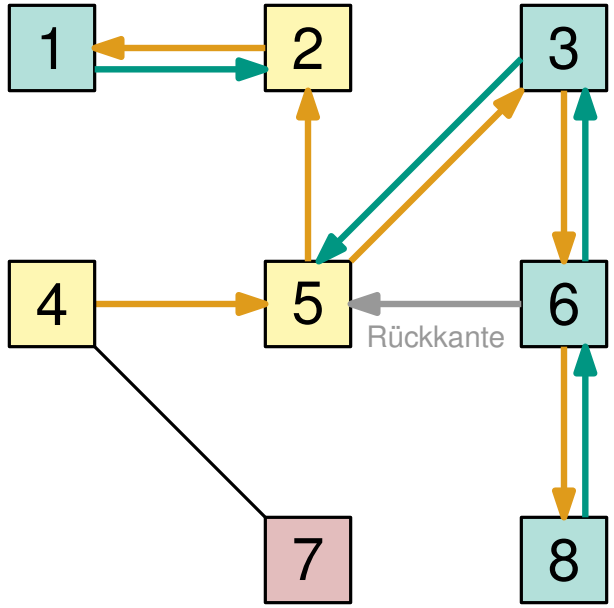
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

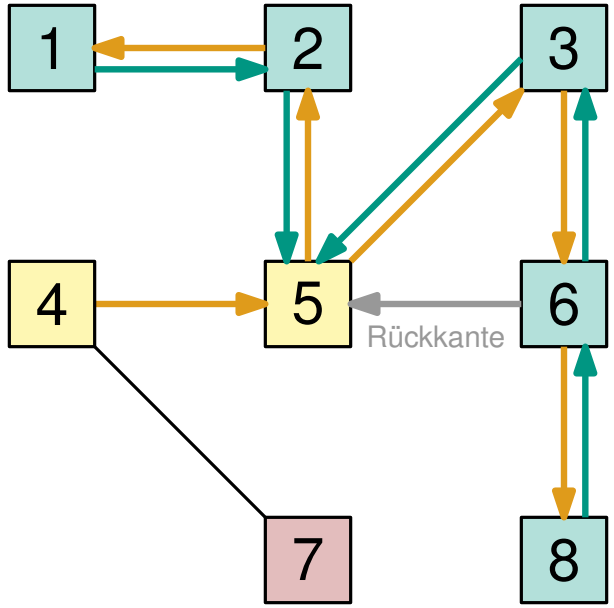
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

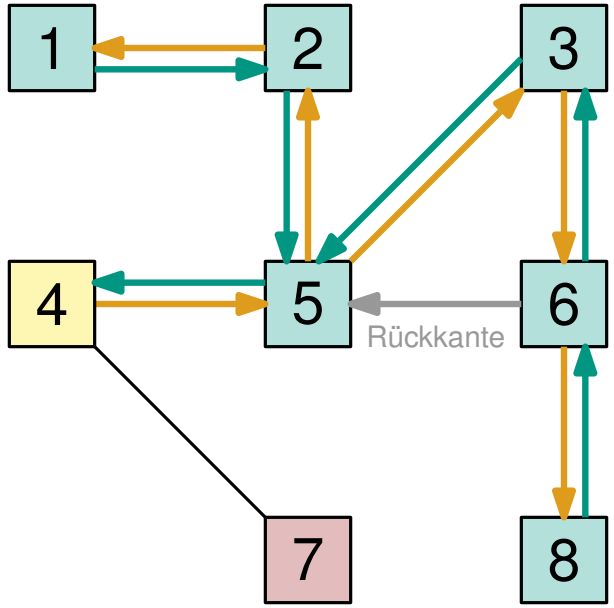
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

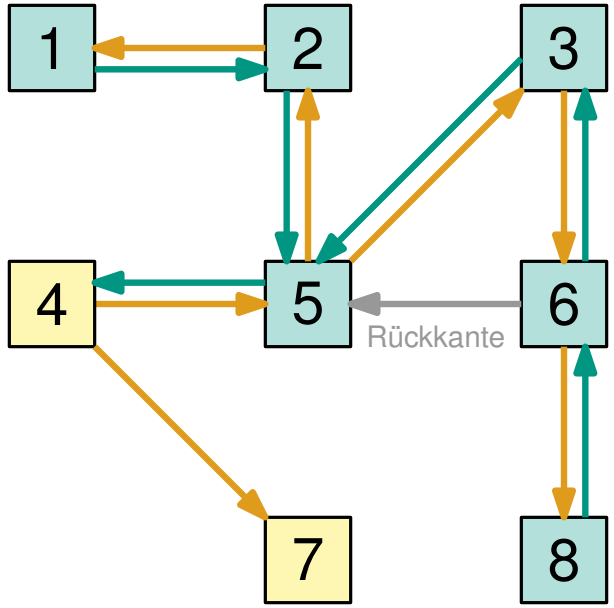
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

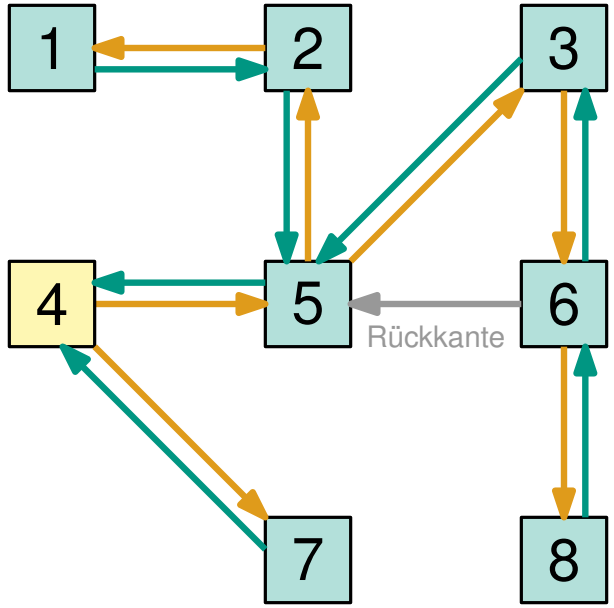
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

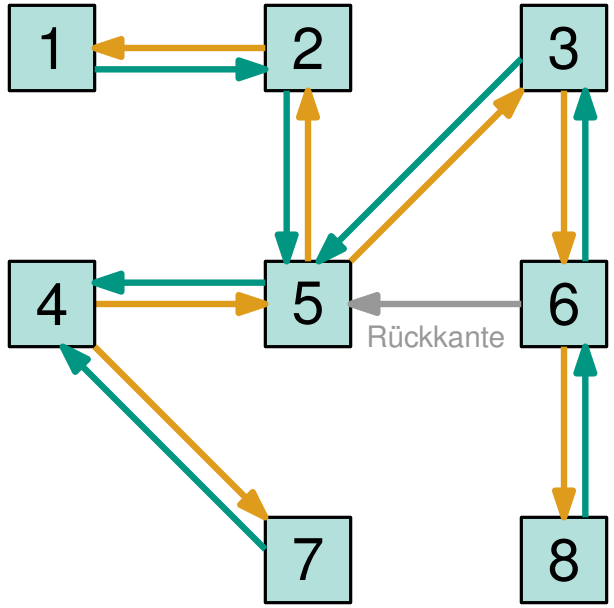
Beispiel: Startknoten 4



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem ungesehenen neuen Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

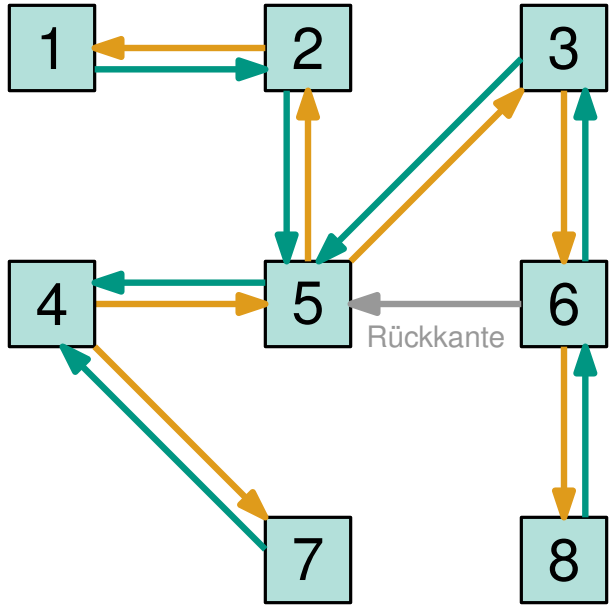
Beispiel: Startknoten 4



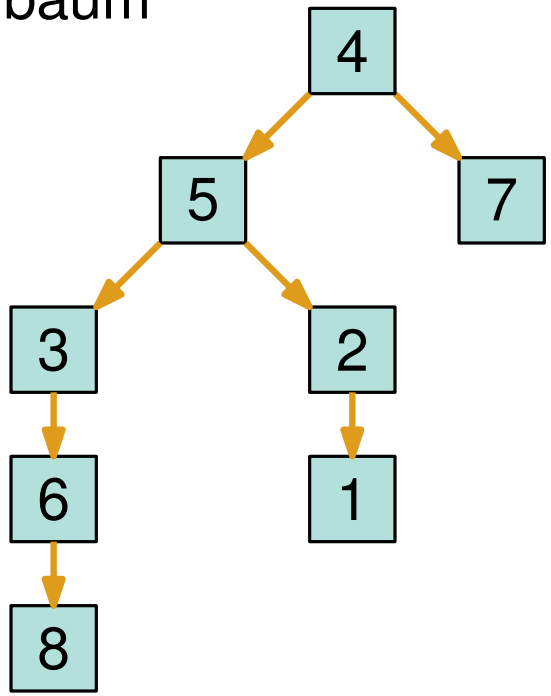
Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem ungesehenen neuen Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

Beispiel: Startknoten 4



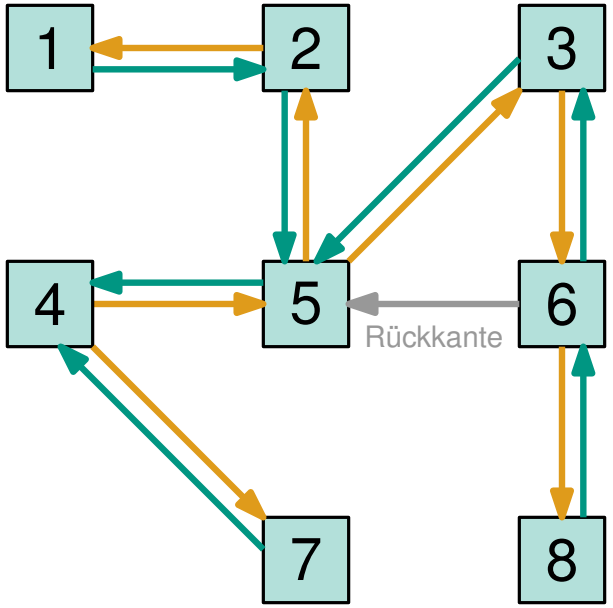
- Tiefensuchbaum



Tiefensuche

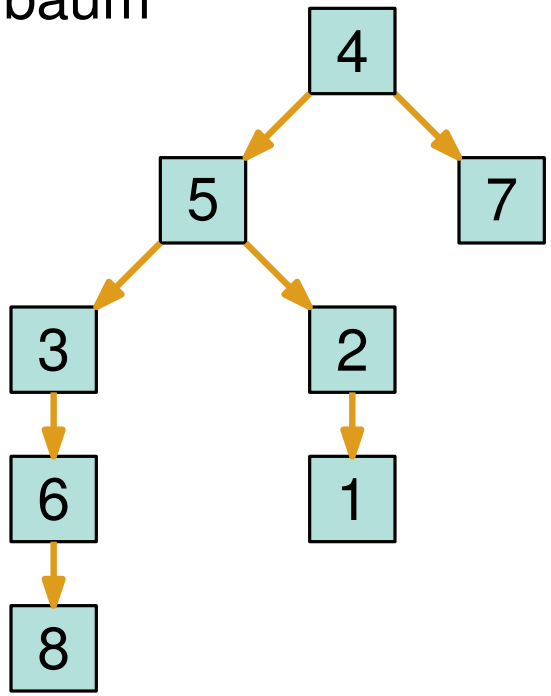
- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem ungesehenen neuen Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

Beispiel: Startknoten 4



- Tiefensuchbaum

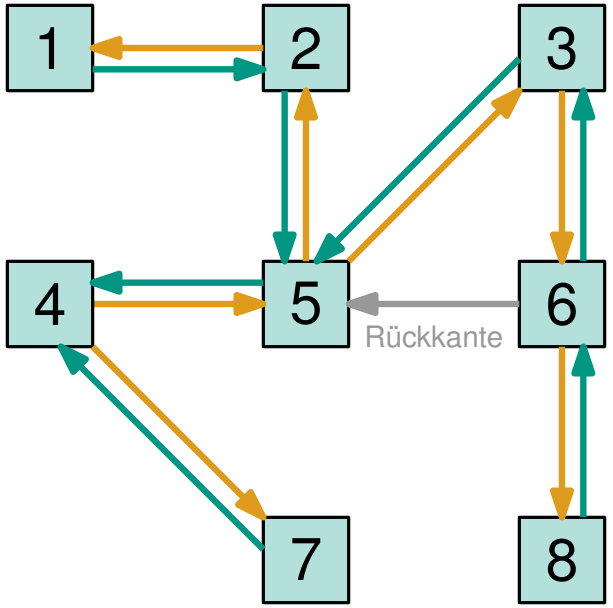
Warum?



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem neuen (ungesehenen) Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

Beispiel: Startknoten 4

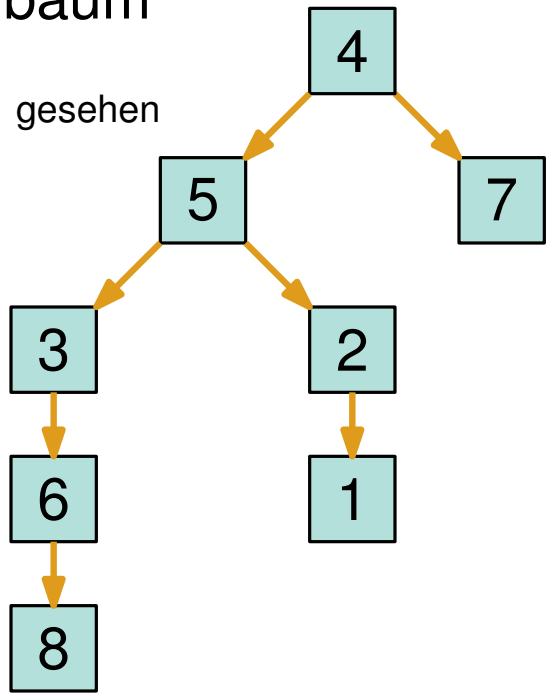


- Tiefensuchbaum

Warum?

- jeder Knoten 1x gesehen (außer Wurzel)

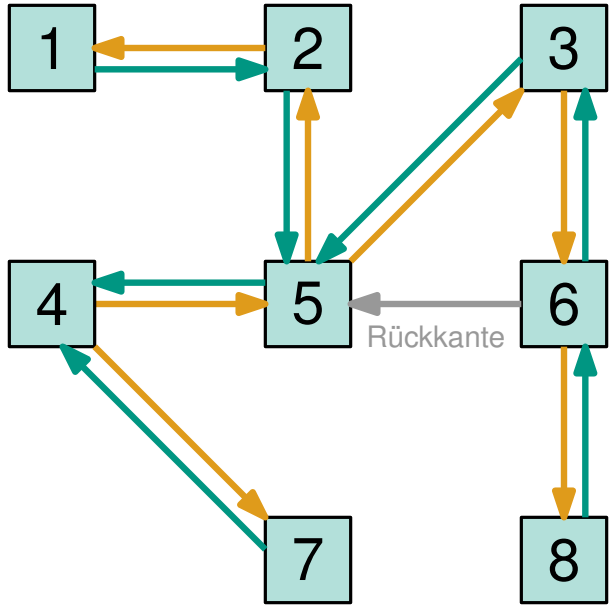
$\rightarrow n - 1$ Kanten



Tiefensuche

- Starte bei einem Knoten s
- Gehe in jedem Schritt zu einem ungesehenen neuen Nachbarn
- Sackgasse: (kein neuer Nachbar) $\rightarrow$ Backtracking: zurück zum Vorgänger
- Während einer Tiefensuche kann sich ein Knoten in 3 Zuständen befinden
  - ungesehen
  - gesehen
  - fertig

Beispiel: Startknoten 4

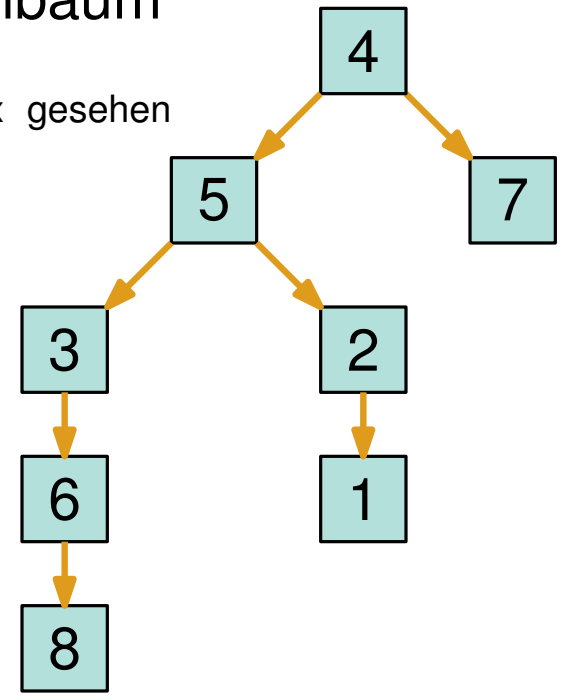


- Tiefensuchbaum

Warum?

- jeder Knoten 1x gesehen (außer Wurzel)

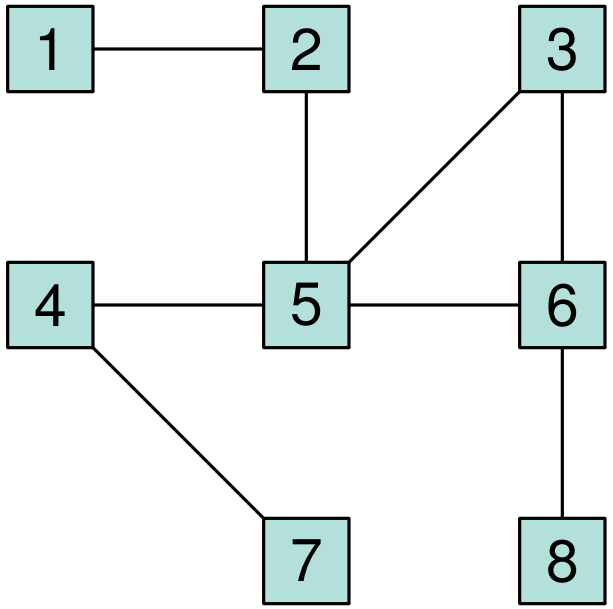
$\rightarrow n - 1$ Kanten



- Freiheitsgrad: Knotenreihenfolge

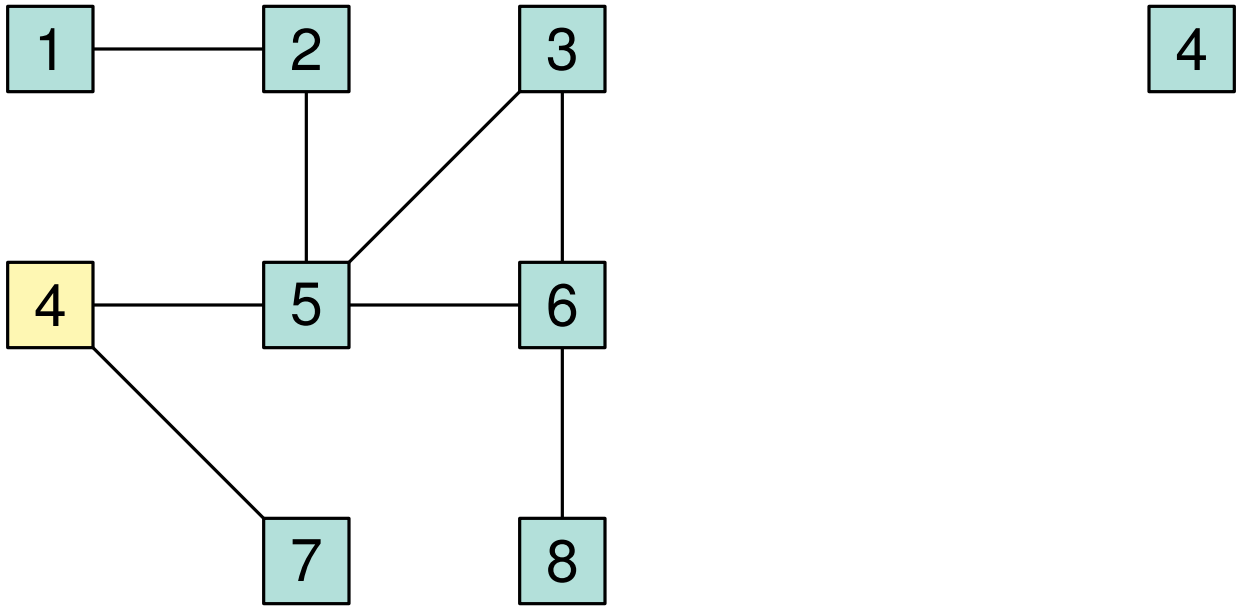
Tiefensuche – Knotenreihenfolge

- Freiheitsgrad: Knotenreihenfolge



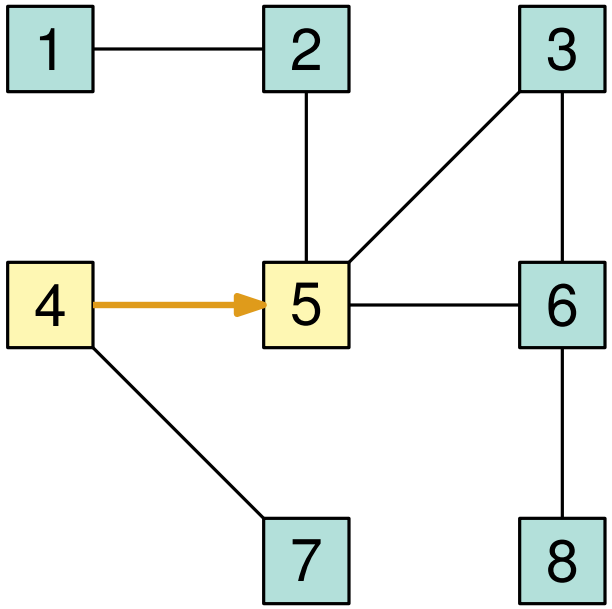
Tiefensuche – Knotenreihenfolge

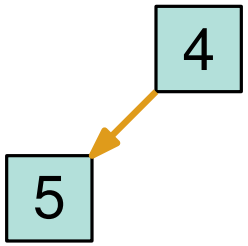
- Freiheitsgrad: Knotenreihenfolge



Tiefensuche – Knotenreihenfolge

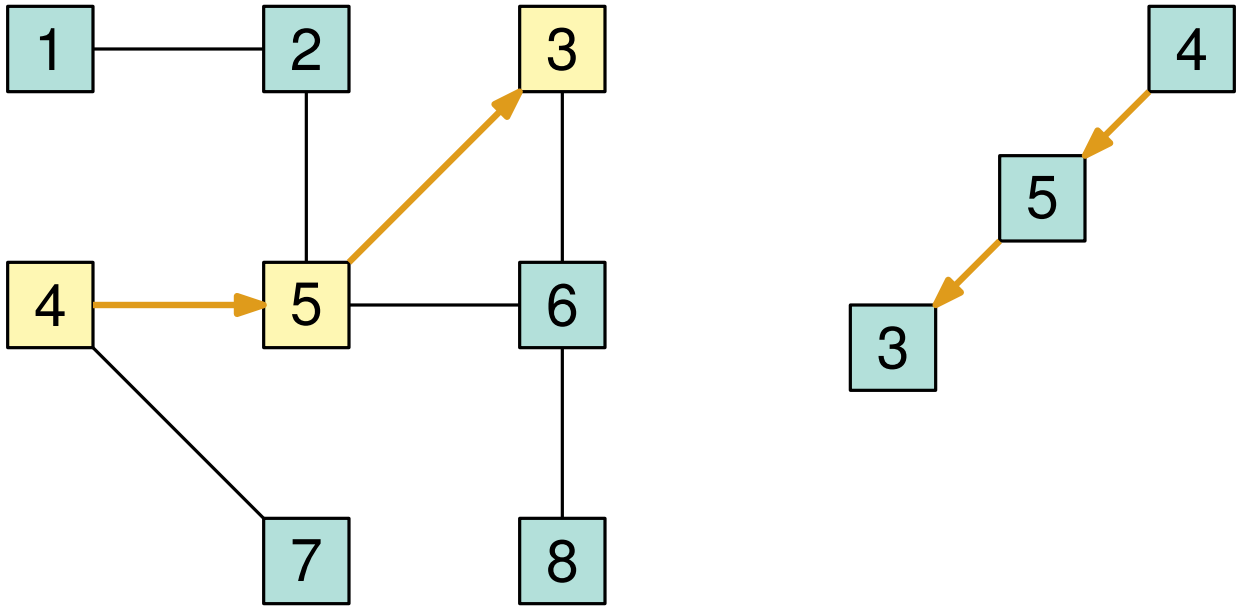
- Freiheitsgrad: Knotenreihenfolge





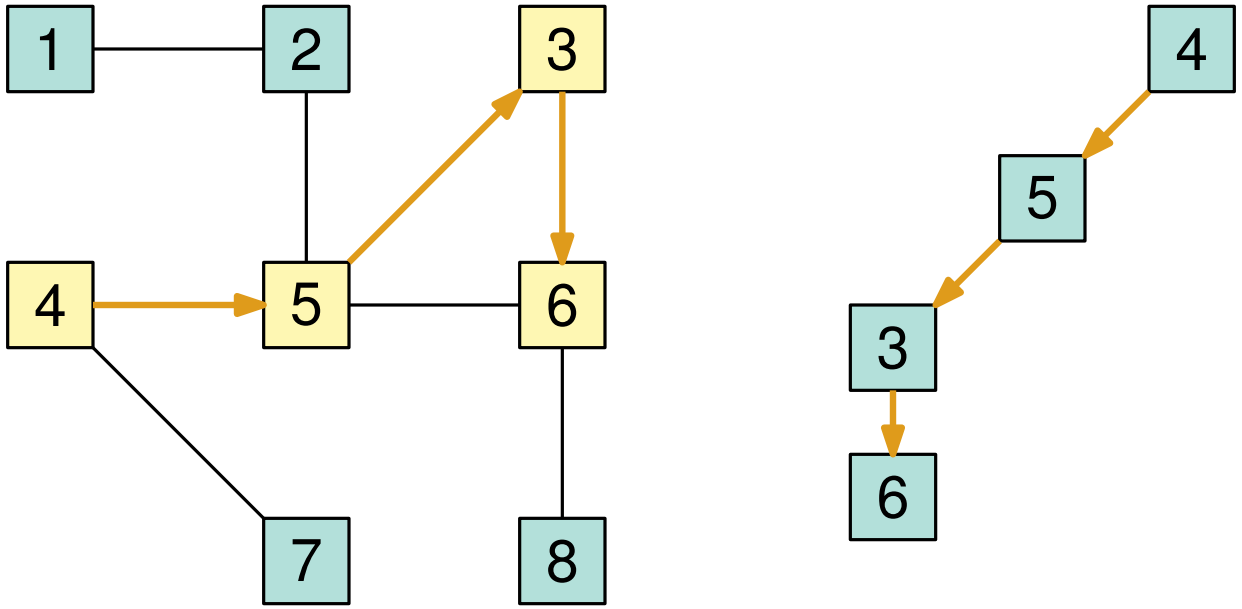
Tiefensuche – Knotenreihenfolge

- Freiheitsgrad: Knotenreihenfolge



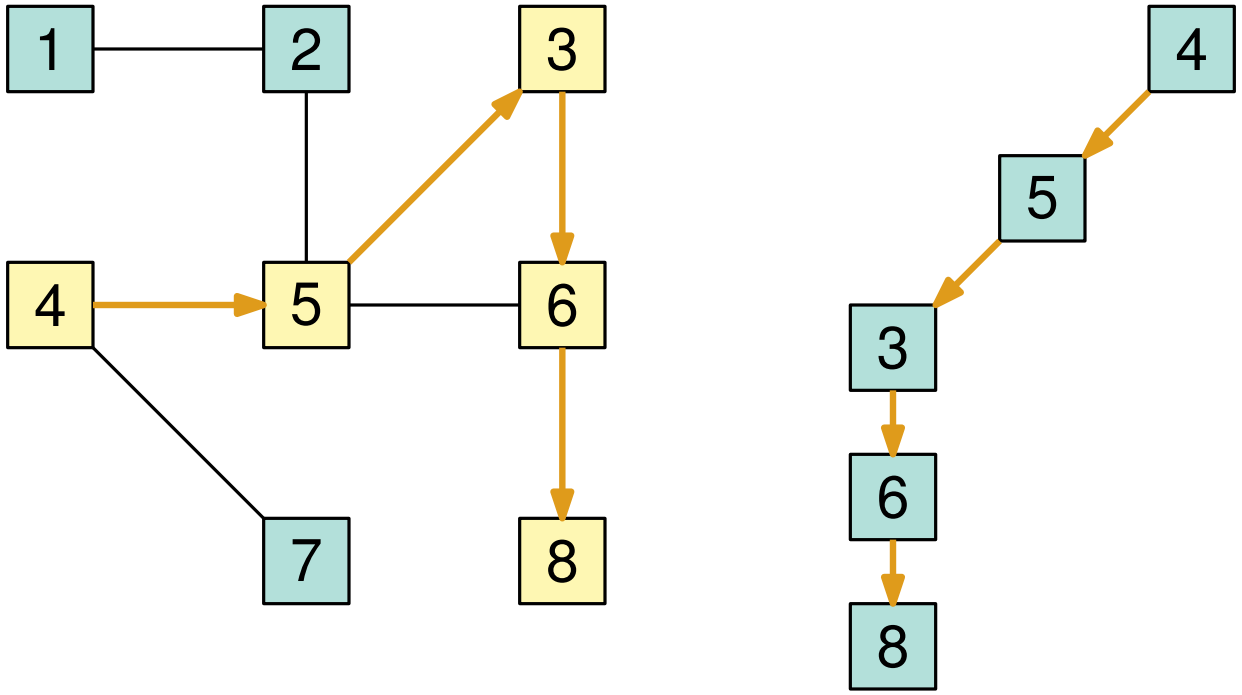
Tiefensuche – Knotenreihenfolge

- Freiheitsgrad: Knotenreihenfolge



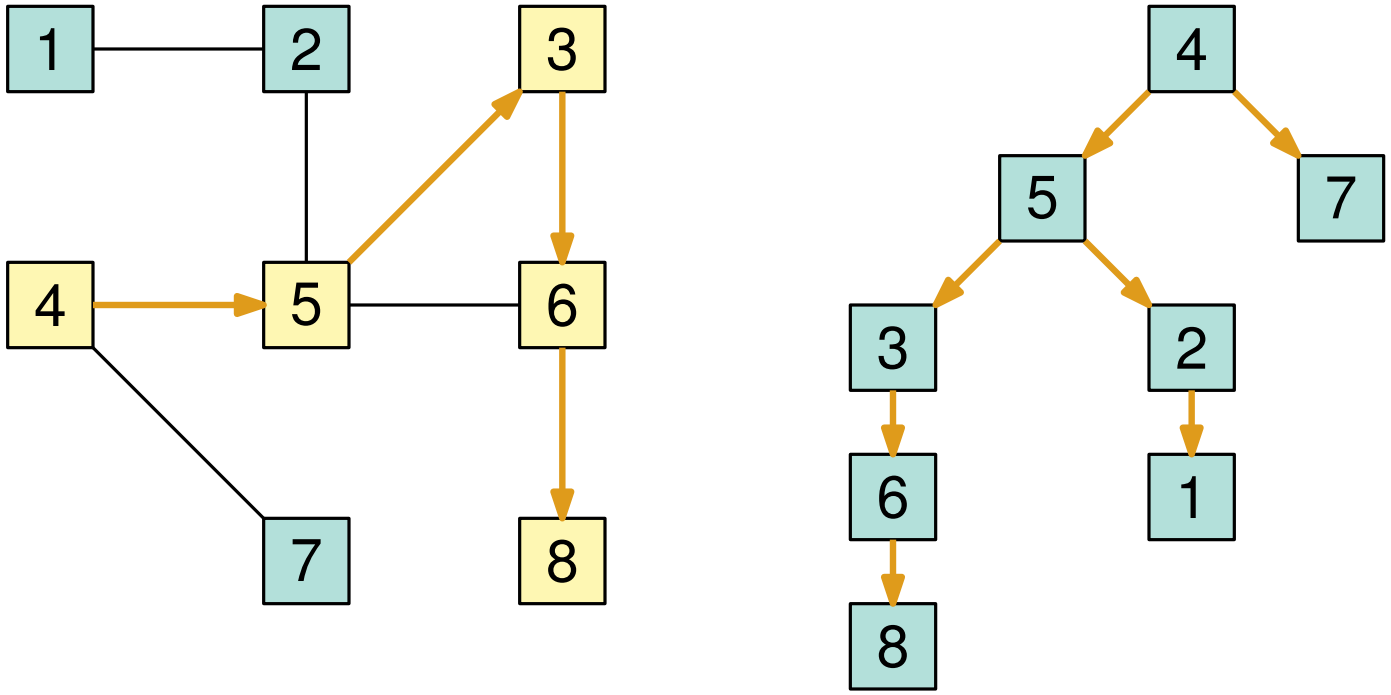
Tiefensuche – Knotenreihenfolge

- Freiheitsgrad: Knotenreihenfolge



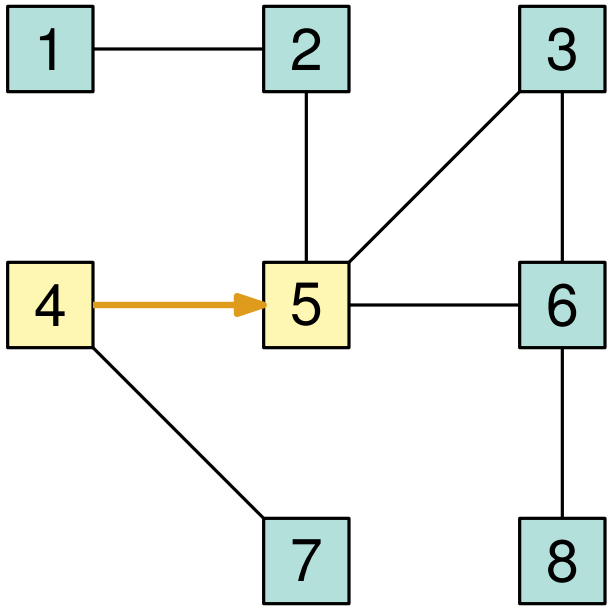
Tiefensuche – Knotenreihenfolge

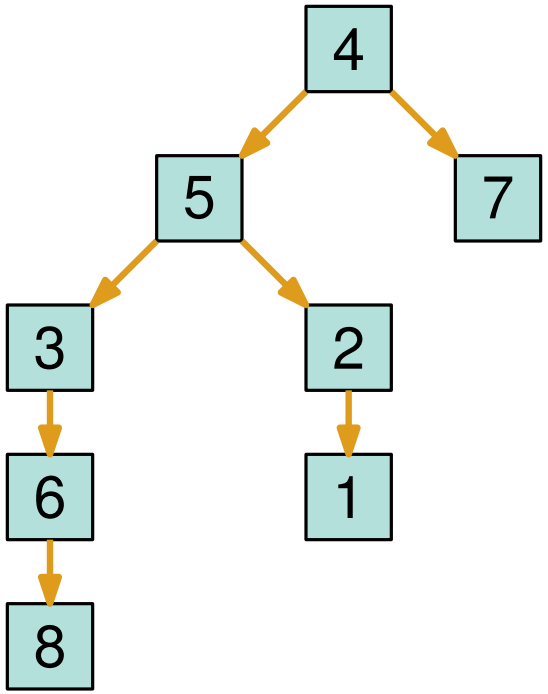
- Freiheitsgrad: Knotenreihenfolge

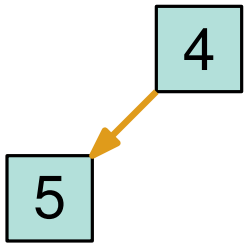


Tiefensuche – Knotenreihenfolge

- Freiheitsgrad: Knotenreihenfolge

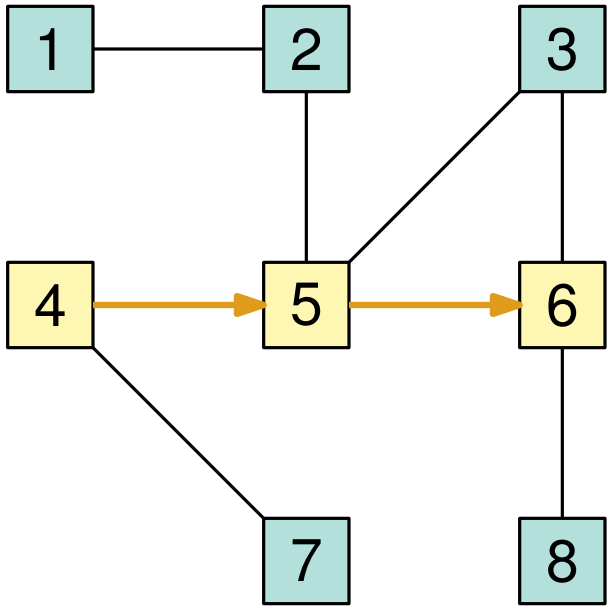


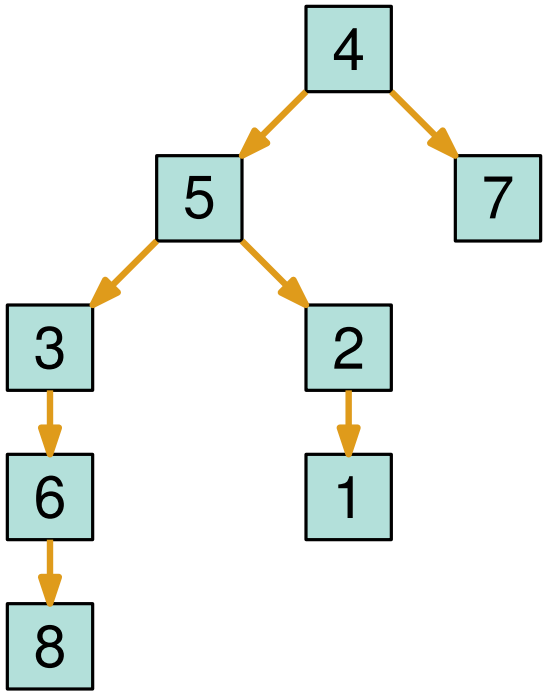


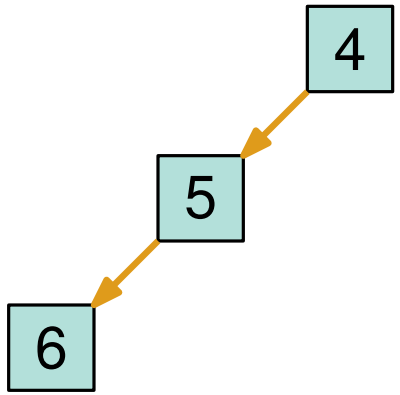


Tiefensuche – Knotenreihenfolge

- Freiheitsgrad: Knotenreihenfolge

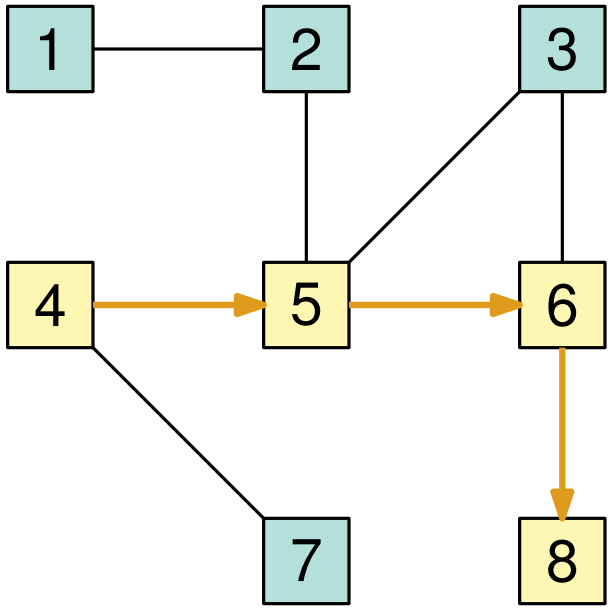


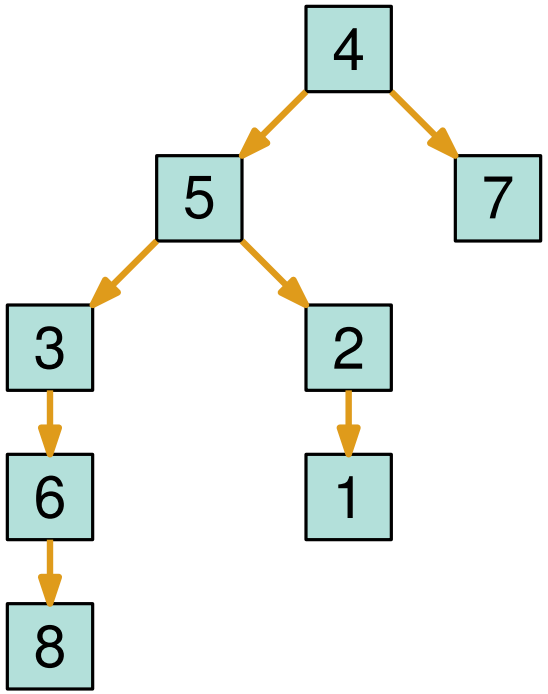


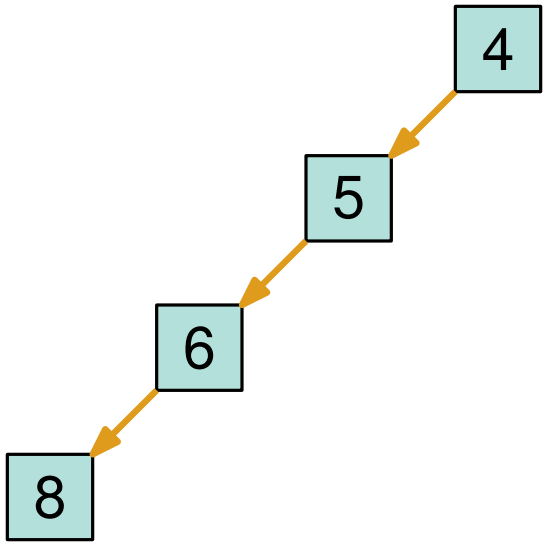


Tiefensuche – Knotenreihenfolge

- Freiheitsgrad: Knotenreihenfolge

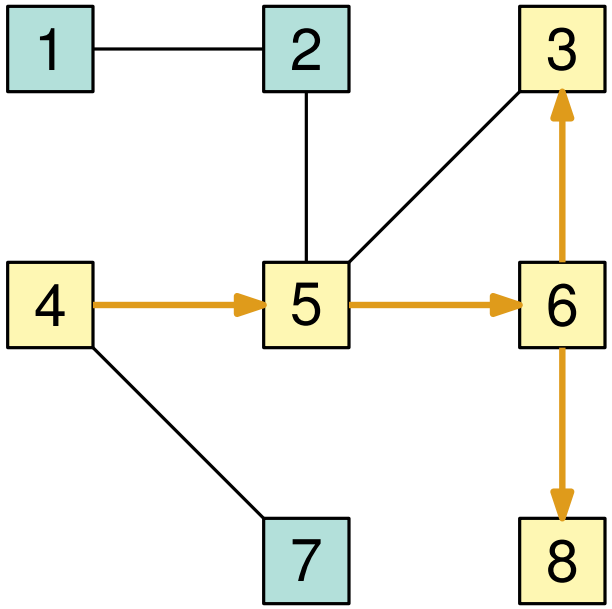


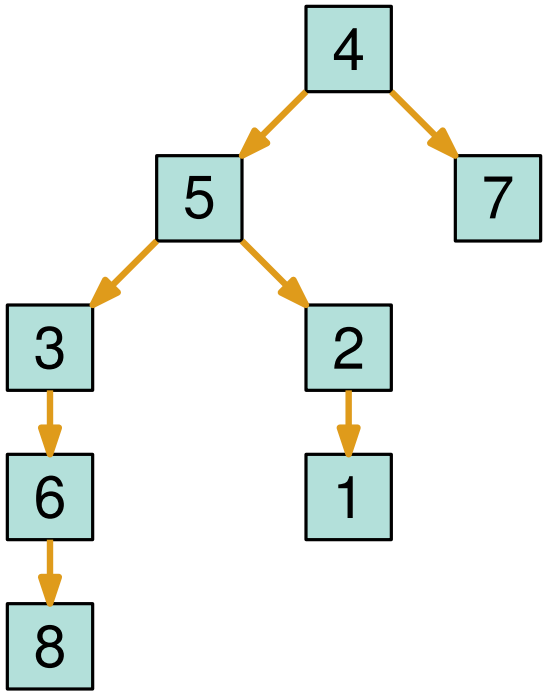


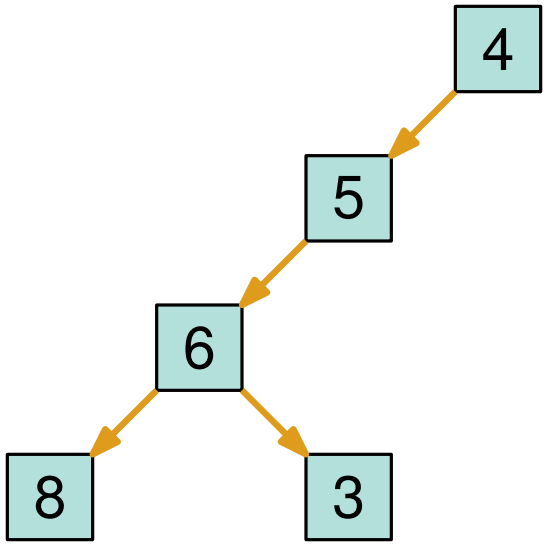


Tiefensuche – Knotenreihenfolge

- Freiheitsgrad: Knotenreihenfolge

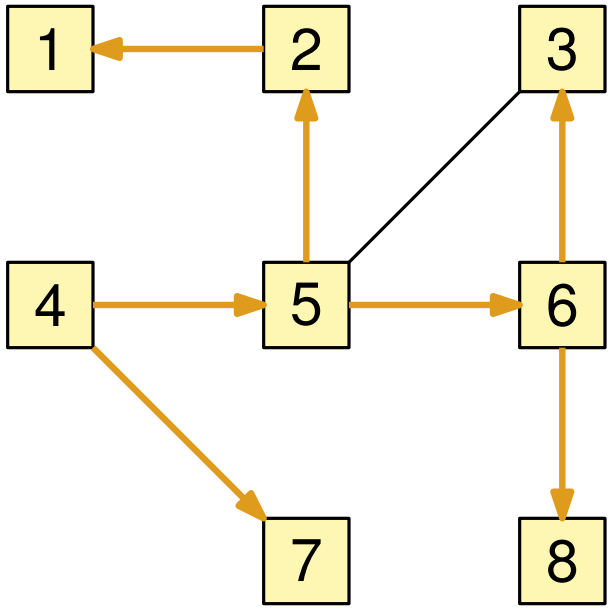


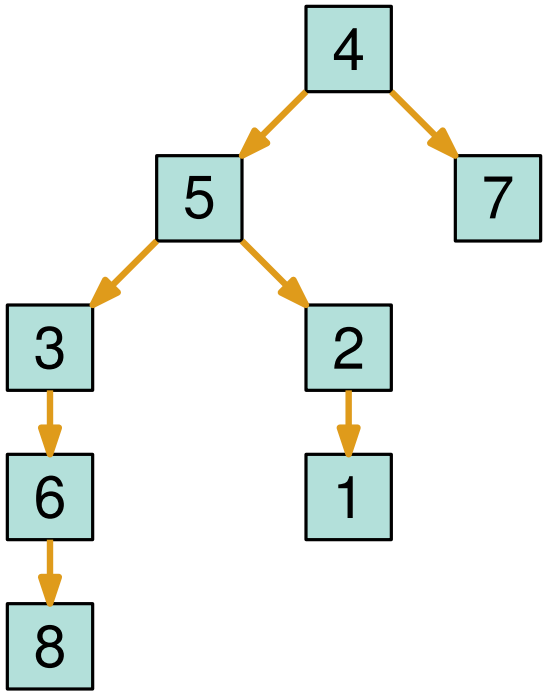


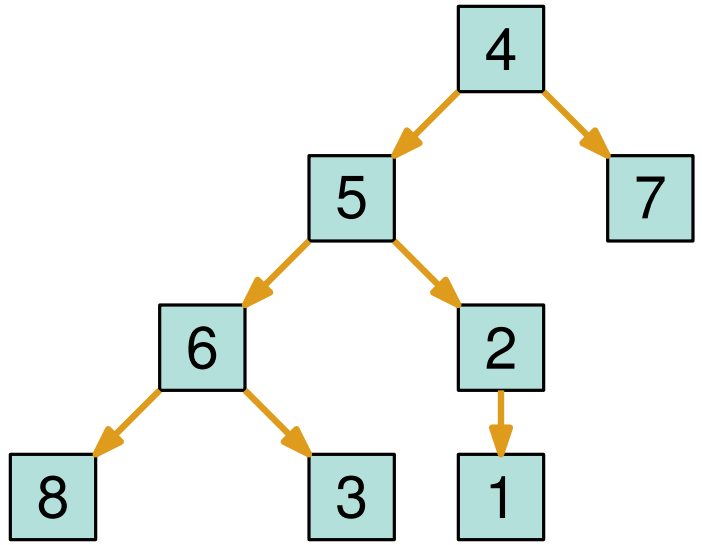


Tiefensuche – Knotenreihenfolge

- Freiheitsgrad: Knotenreihenfolge

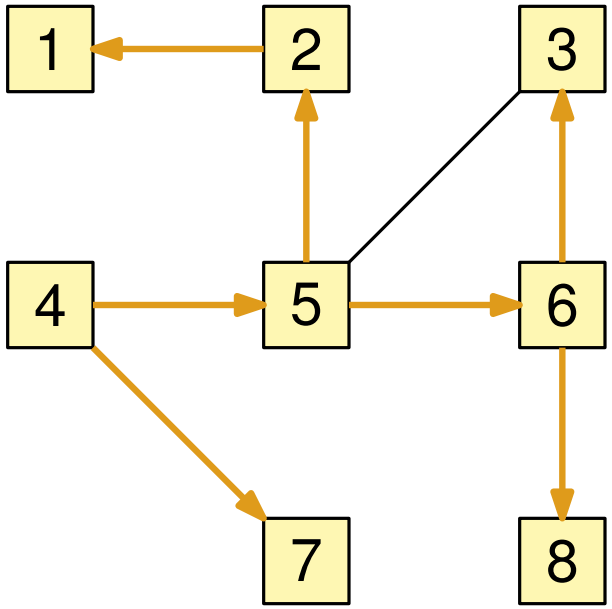


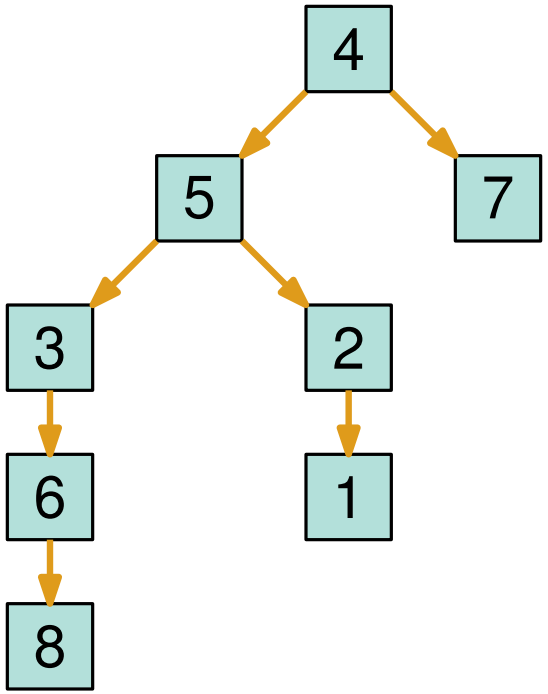


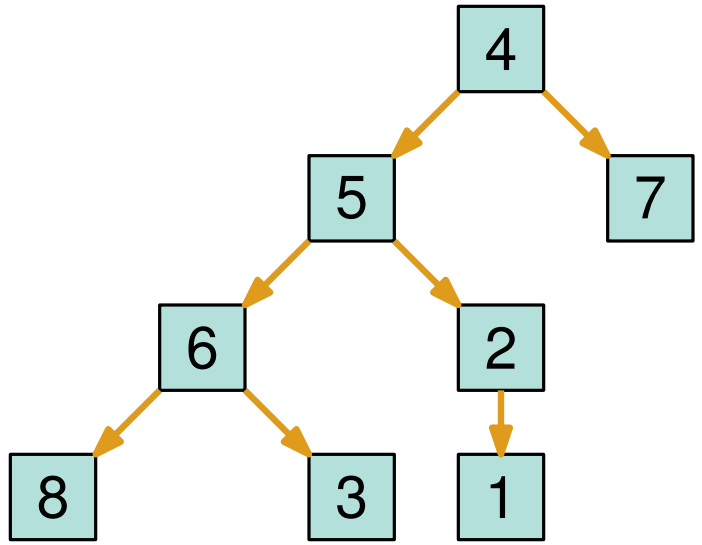


Tiefensuche – Knotenreihenfolge

- Freiheitsgrad: Knotenreihenfolge





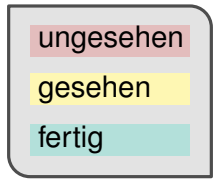


- Die Knotenreihenfolge bestimmt die Struktur des Tiefensuchbaums.
- Wir können uns die Reihenfolge zu Nutze machen...

Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

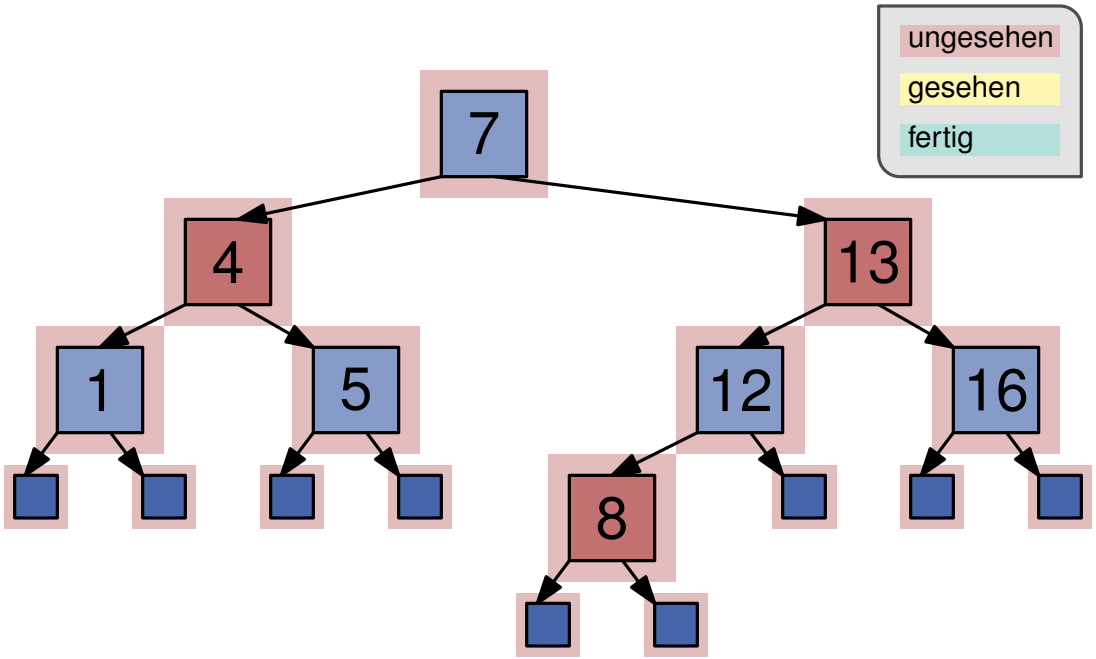
- Knotenreihenfolge: Kleineres Kind zuerst



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

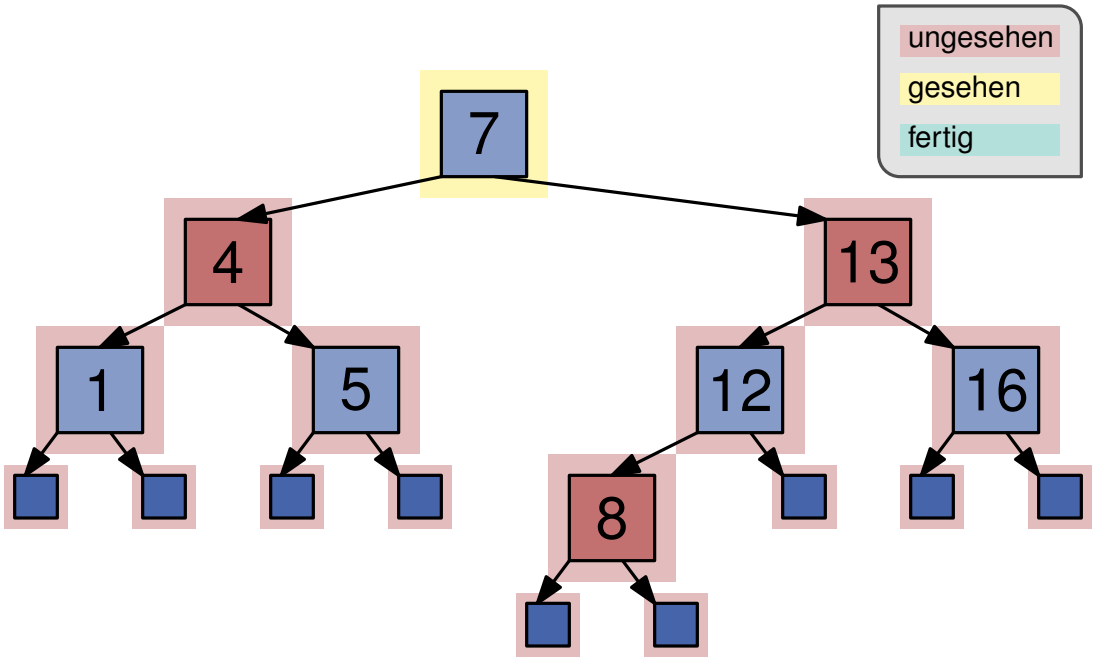
- Knotenreihenfolge: Kleineres Kind zuerst



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

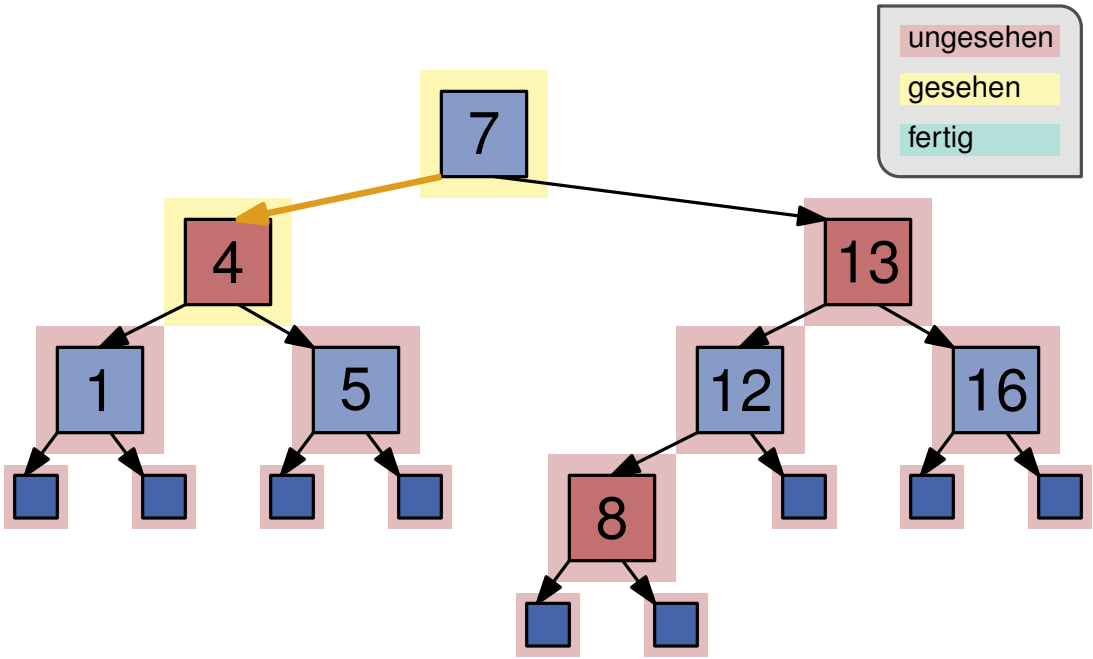
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

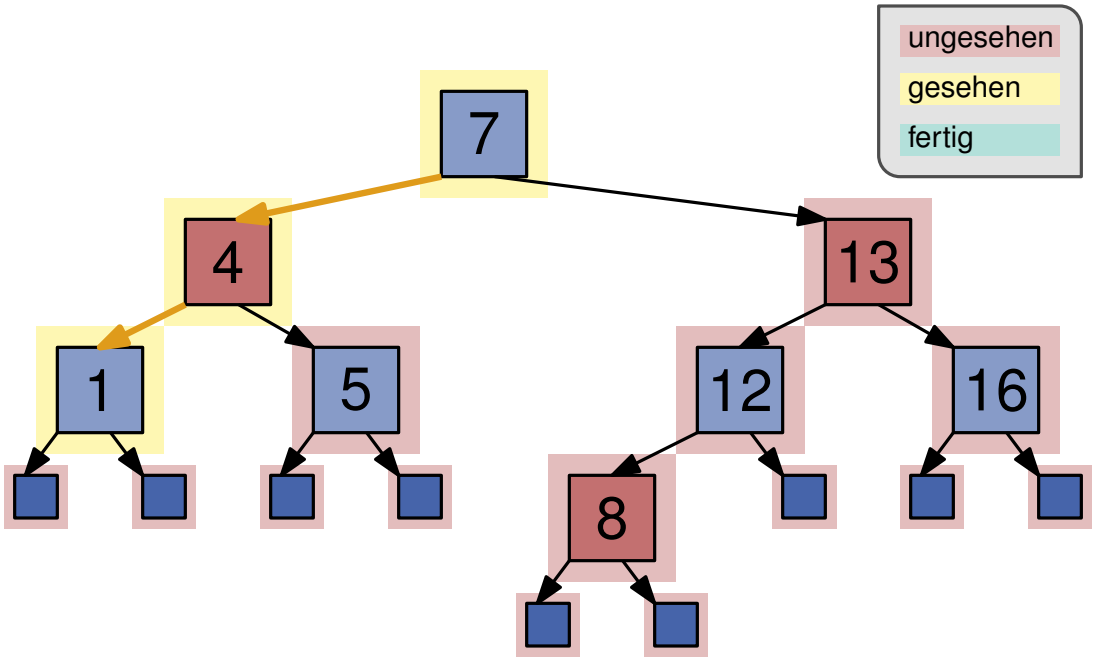
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

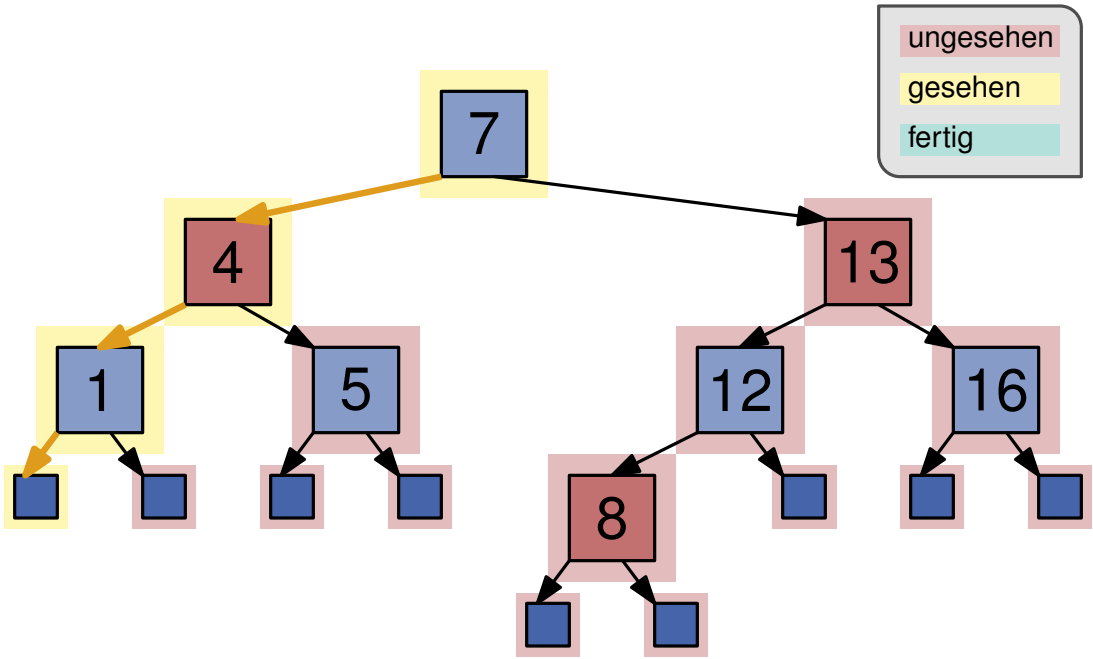
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

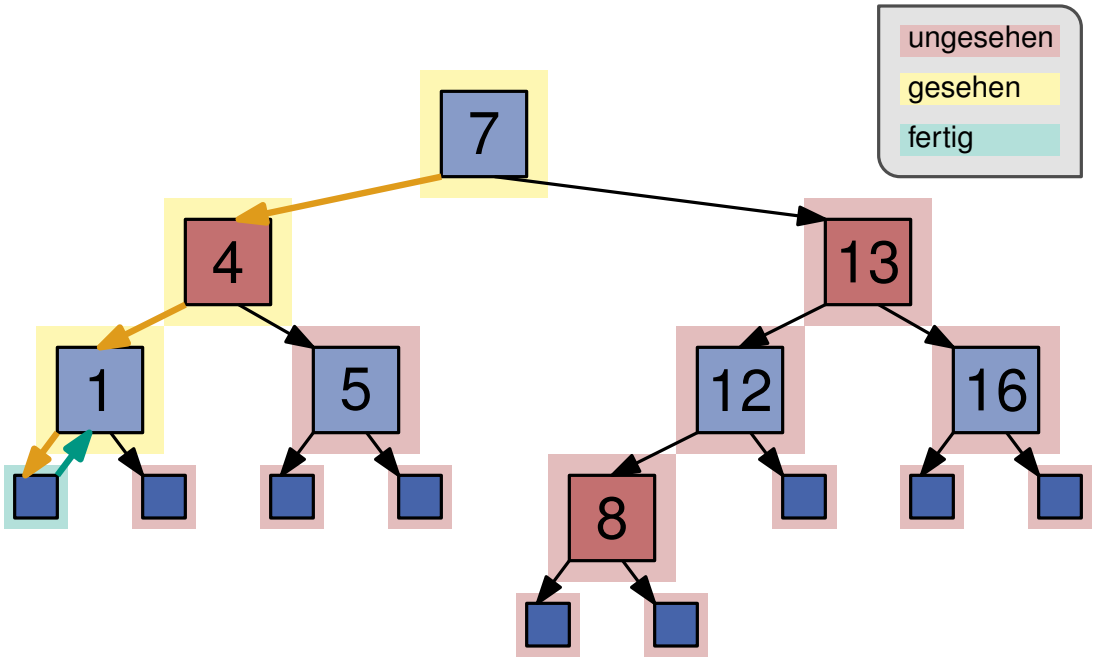
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

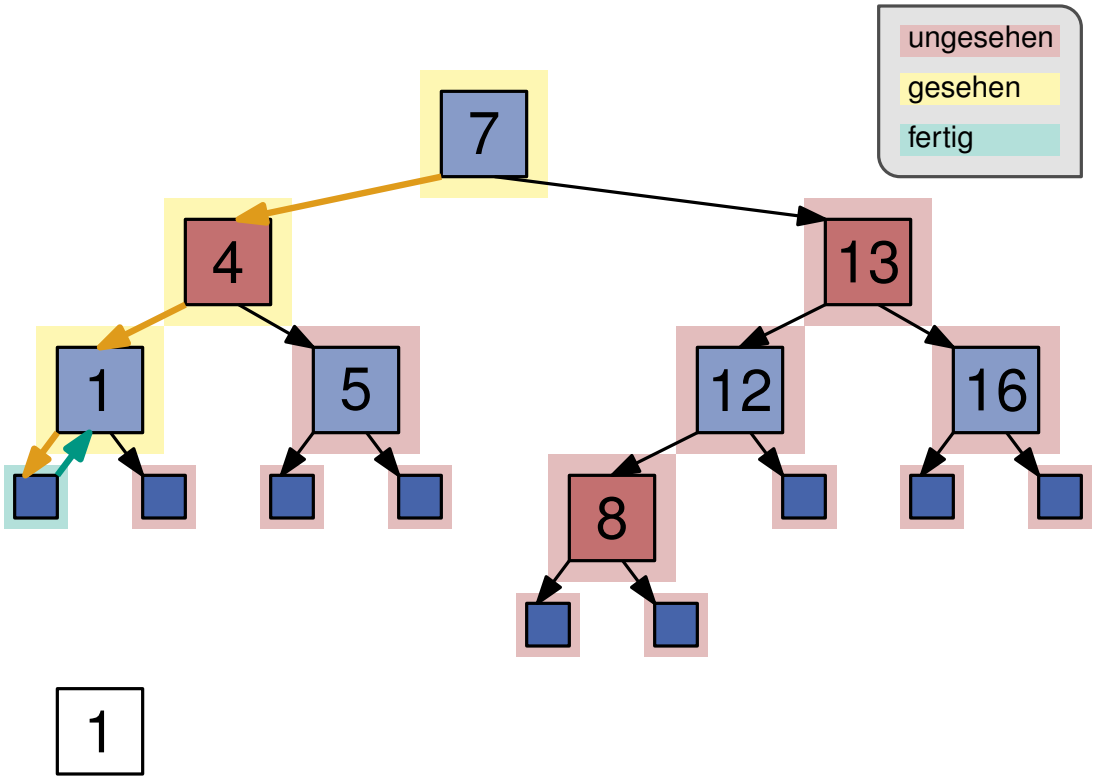
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

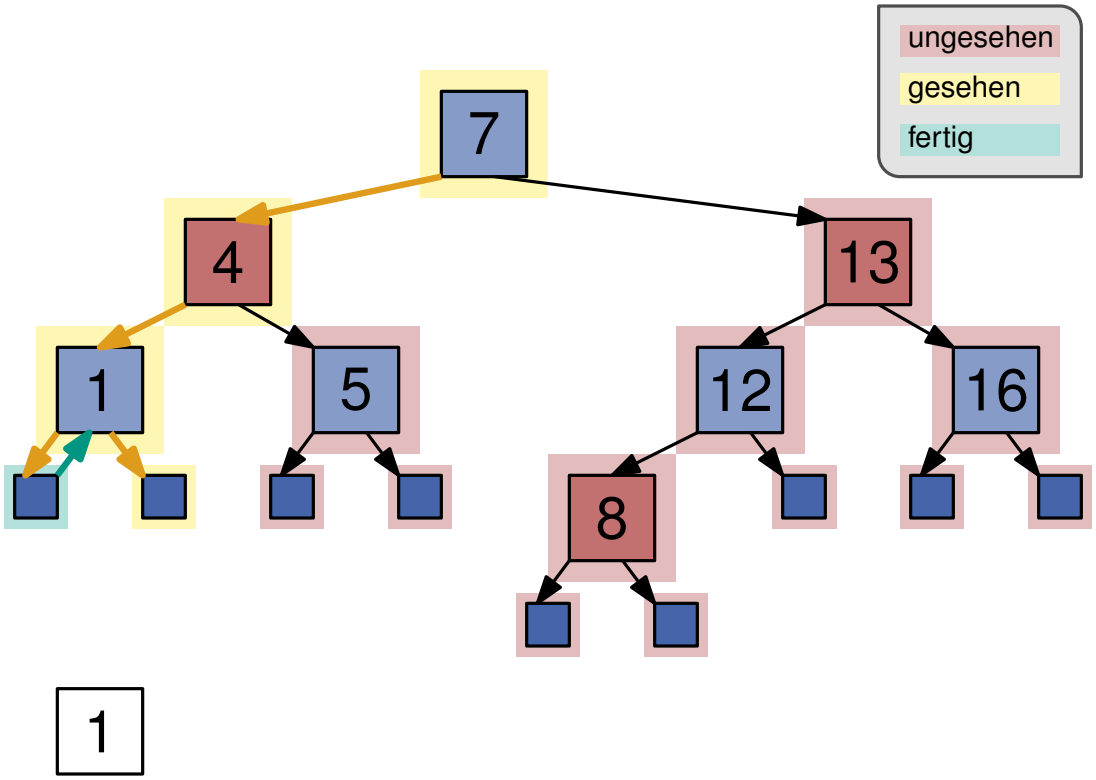
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

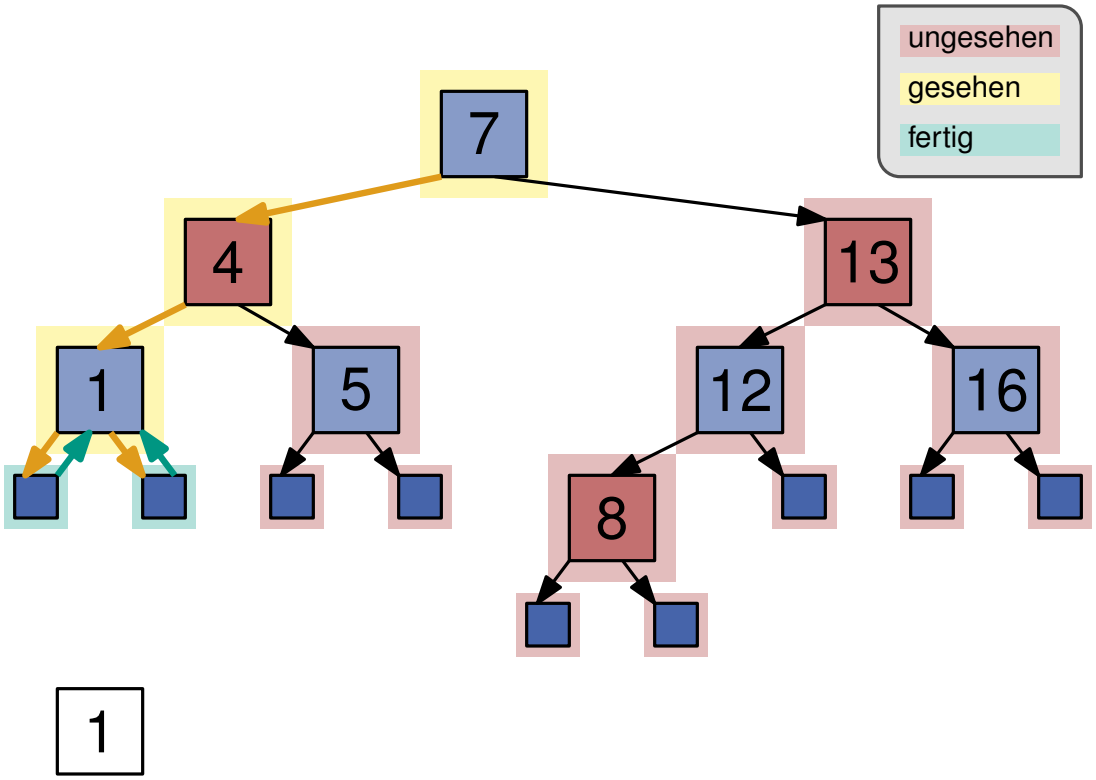
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

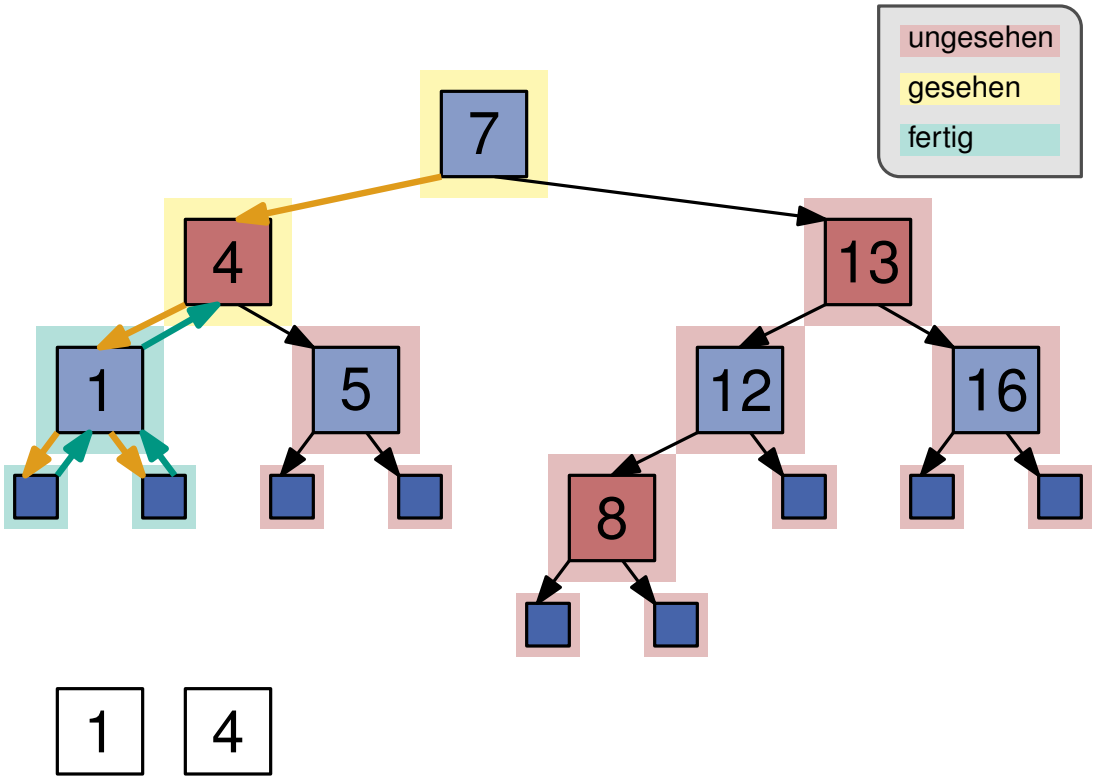
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

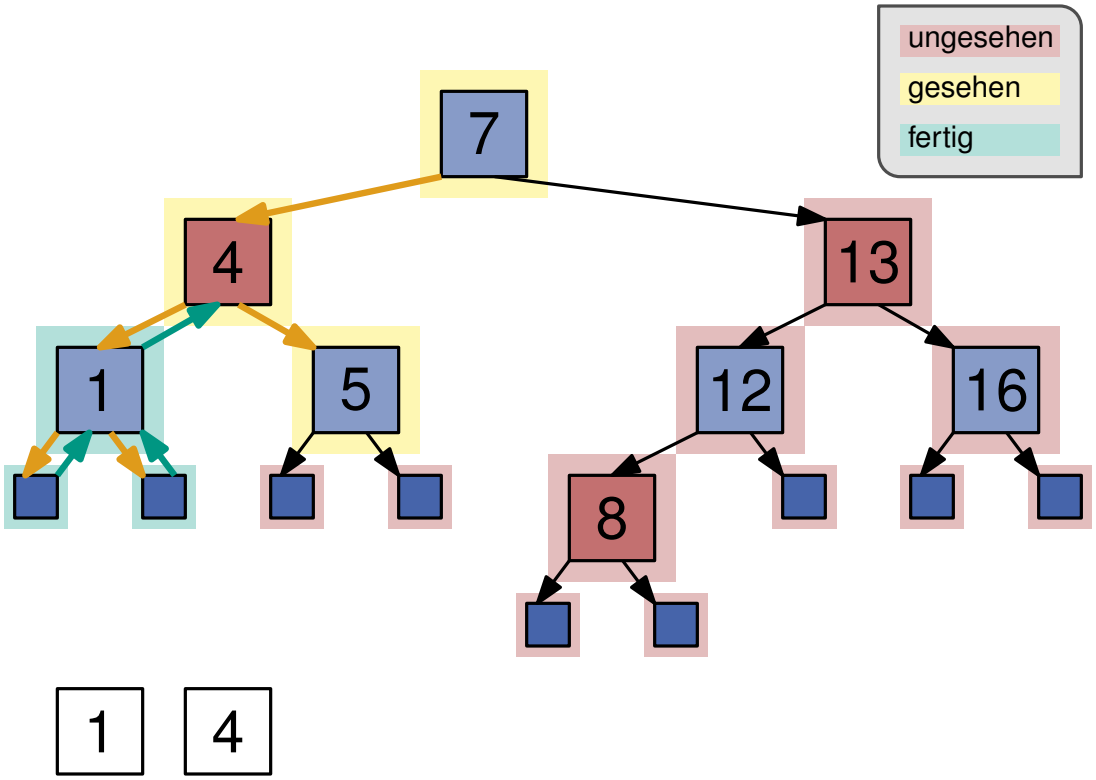
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

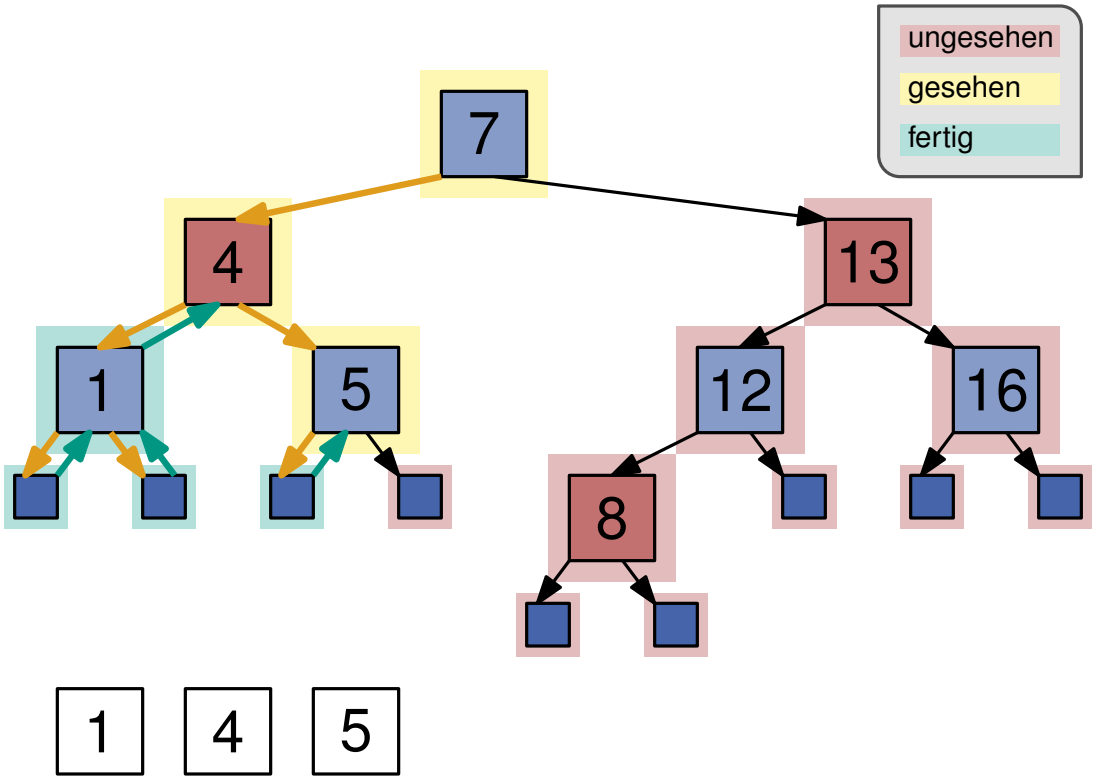
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

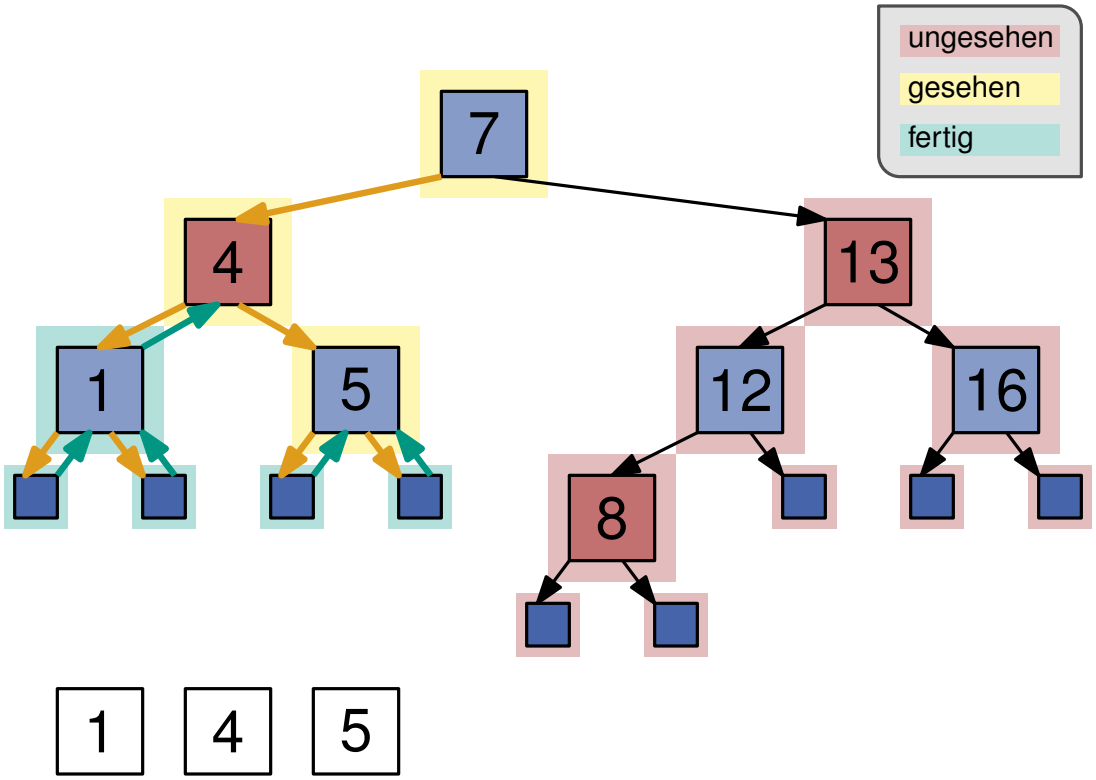
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

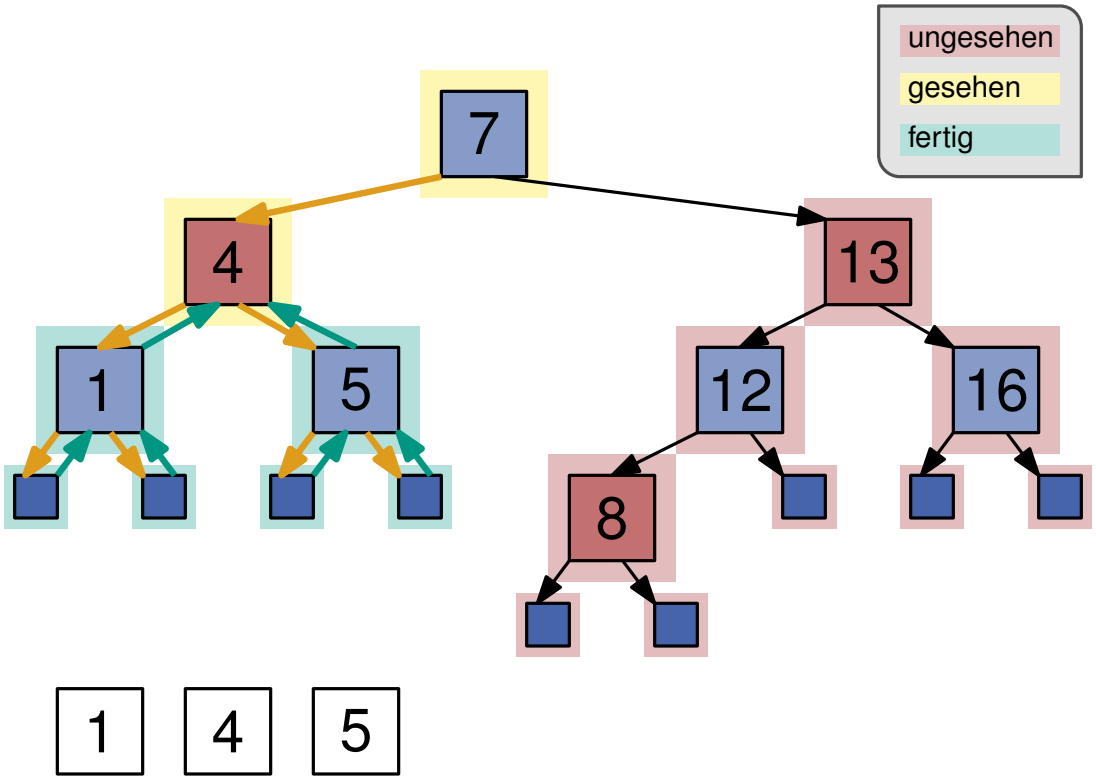
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

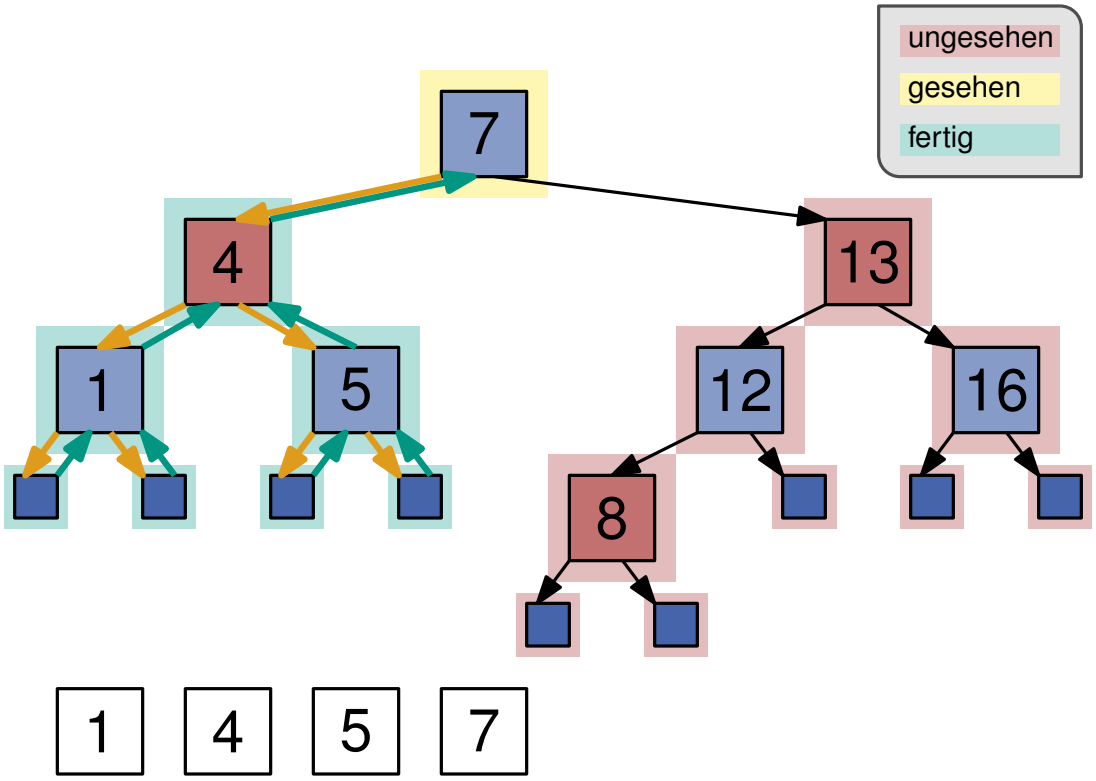
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

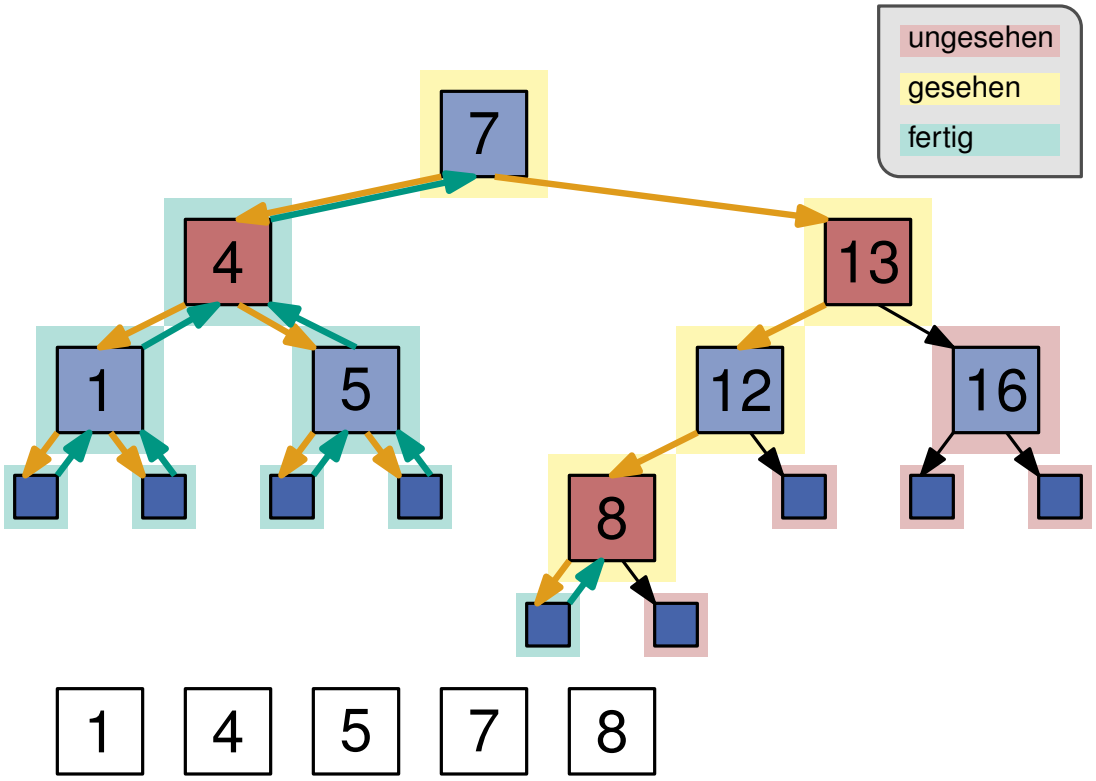
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

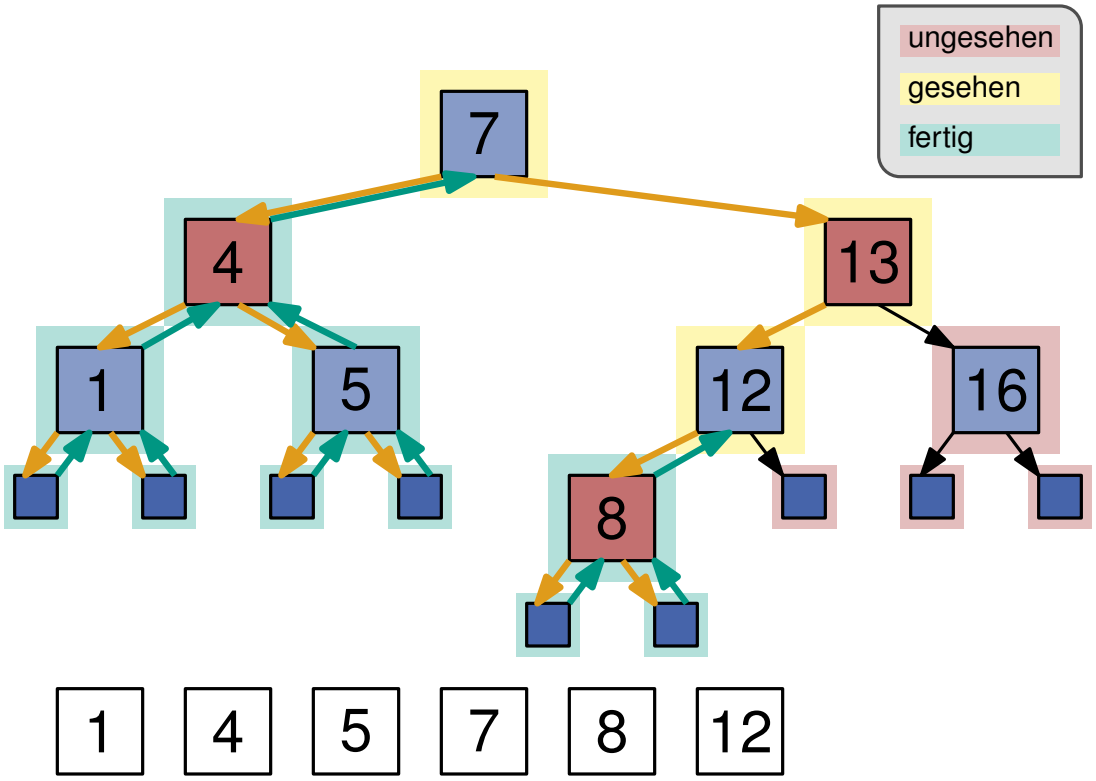
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

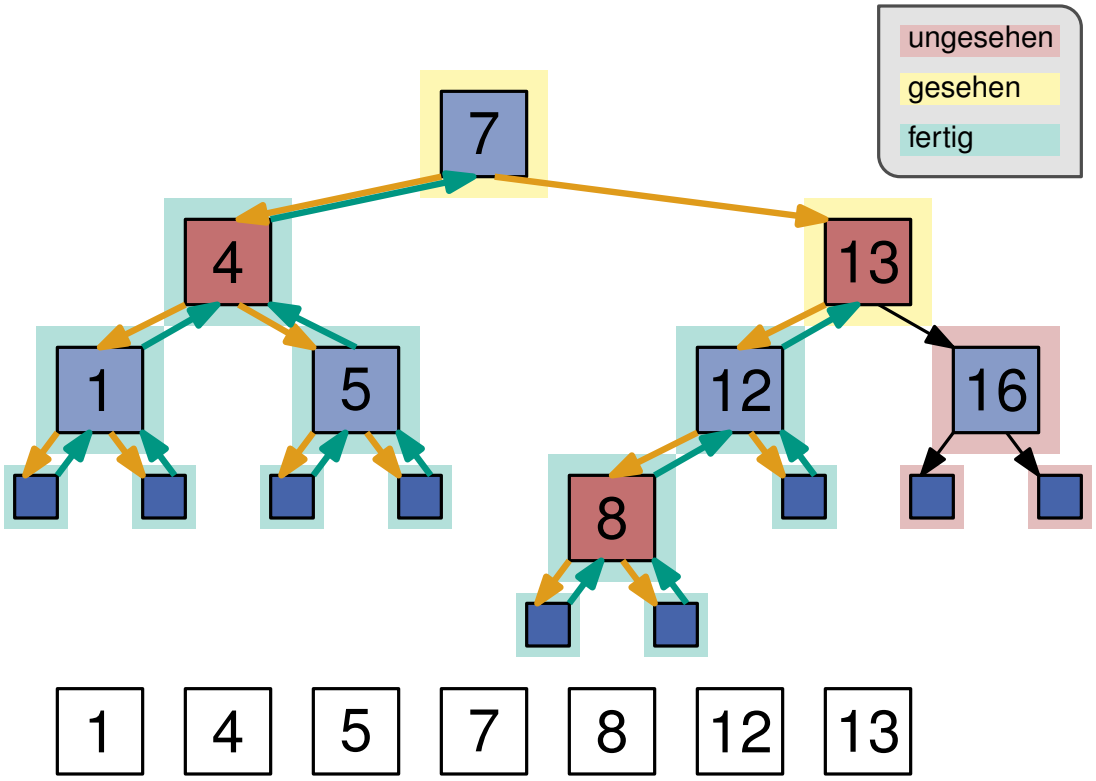
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

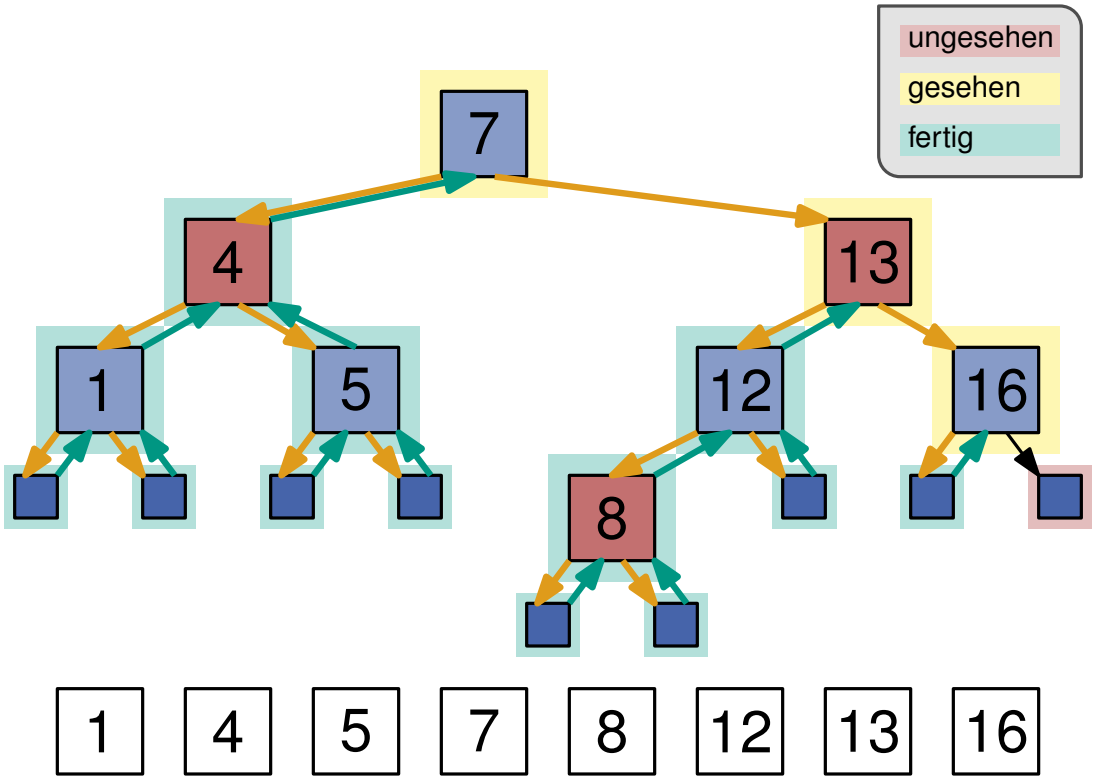
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

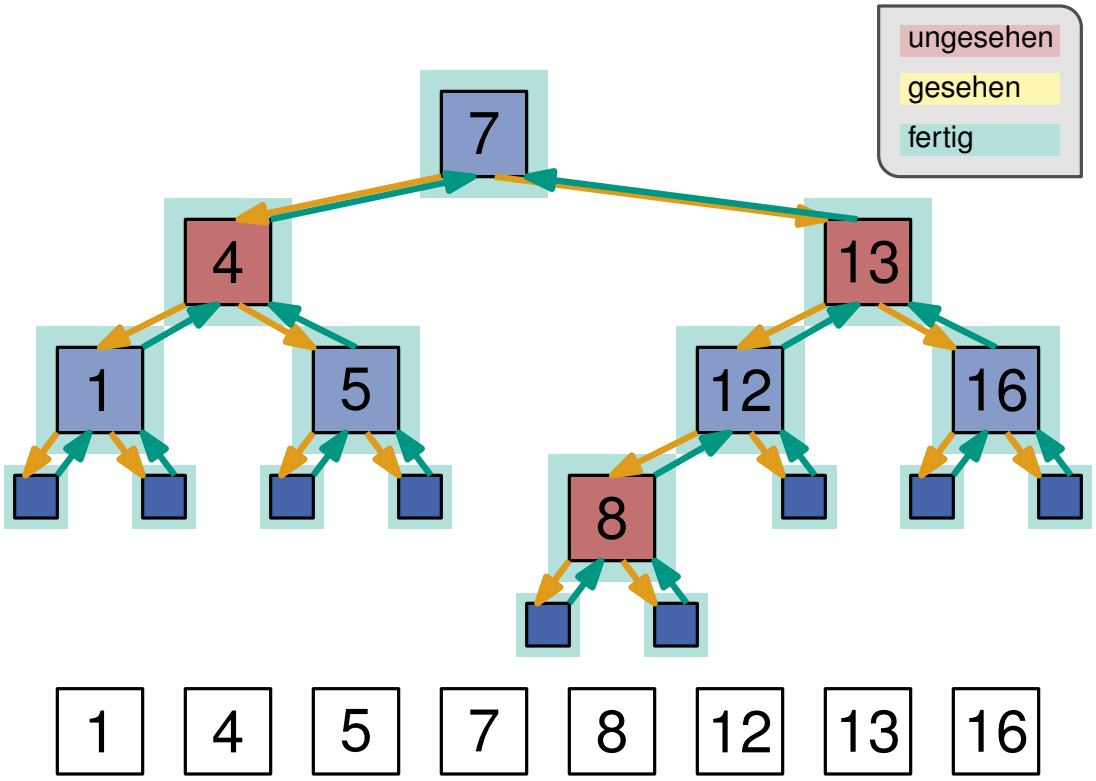
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

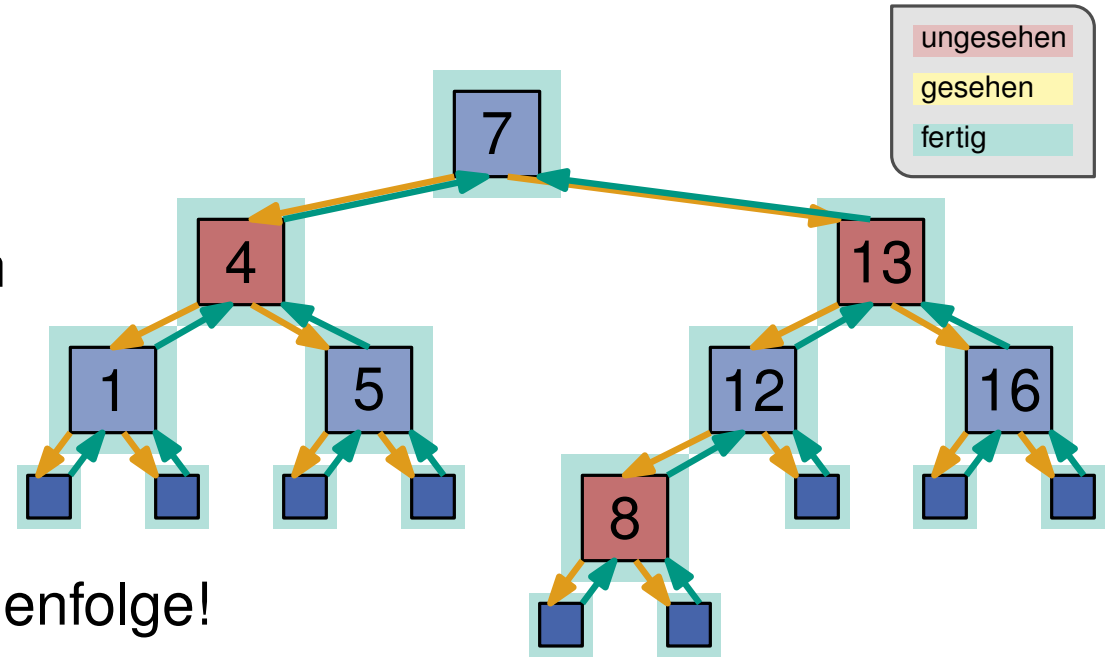
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



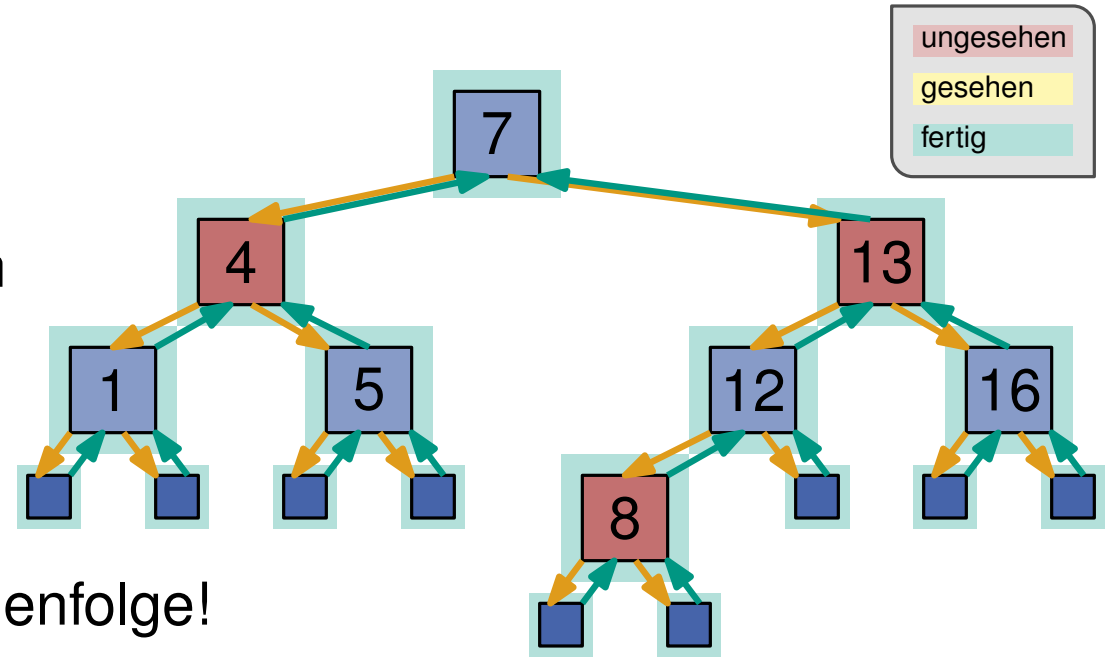
Wir erhalten die Schlüssel in aufsteigender Reihenfolge!

1	4	5	7	8	12	13	16

Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

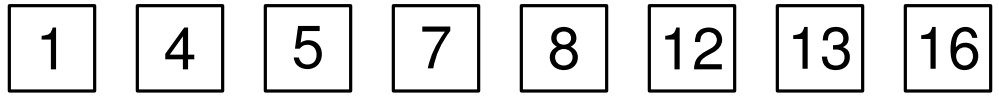
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Wir erhalten die Schlüssel in aufsteigender Reihenfolge!

Laufzeit?

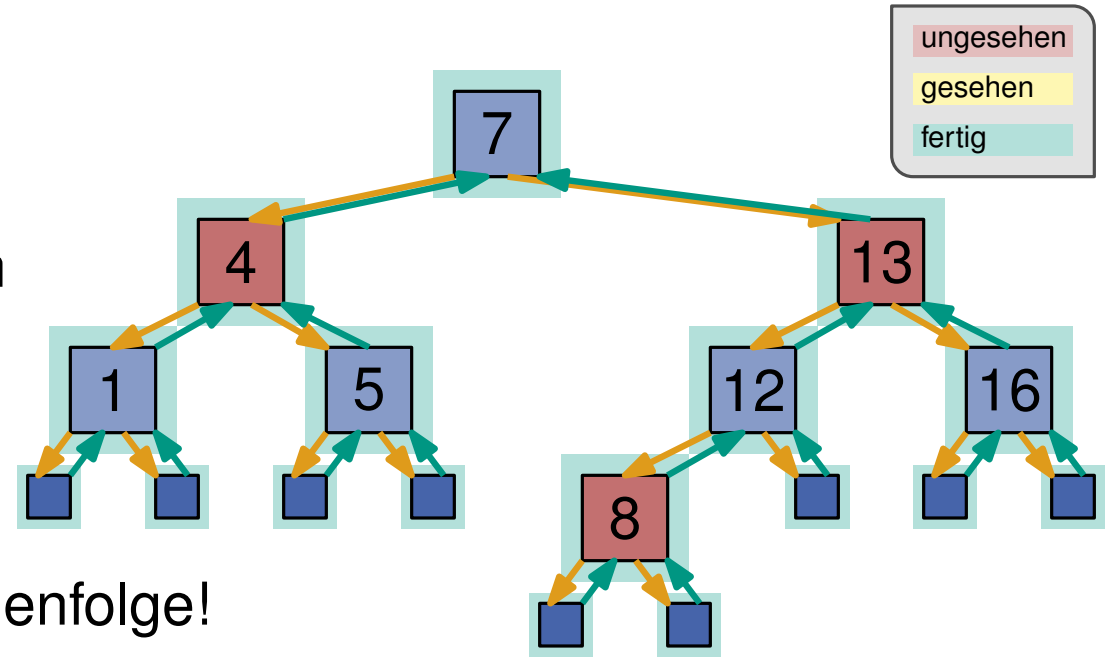
- Jede Kante wird genau zwei Mal abgelaufen



Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

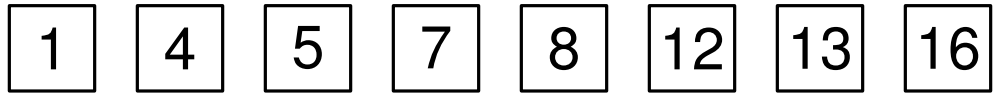
- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Wir erhalten die Schlüssel in aufsteigender Reihenfolge!

Laufzeit?

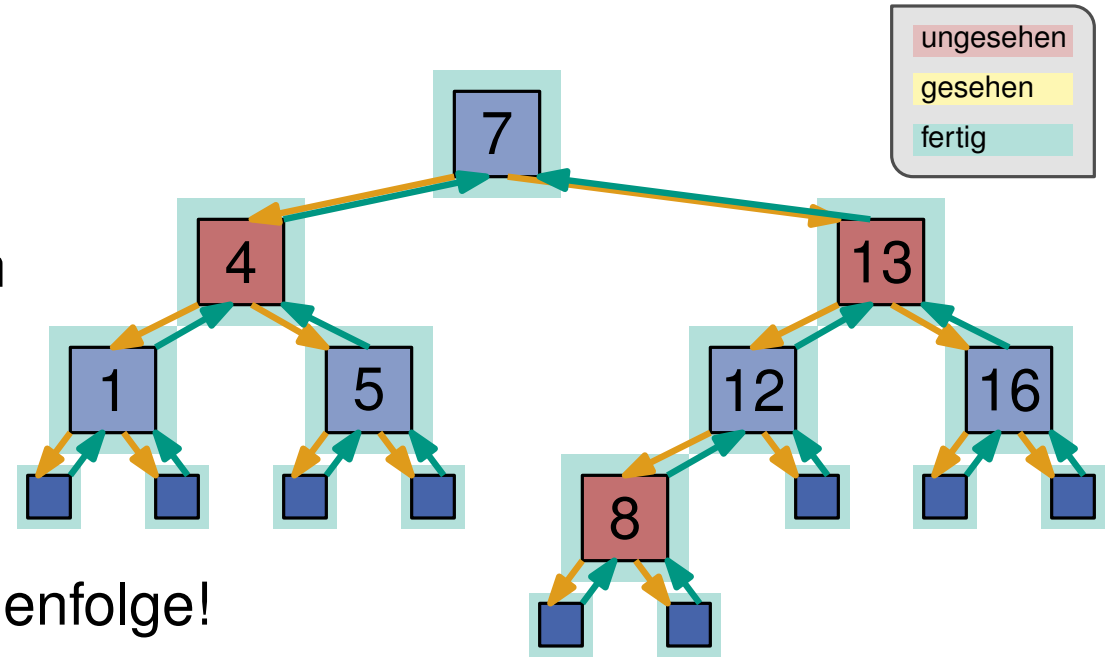
- Jede Kante wird genau zwei Mal abgelaufen
- Der Baum hat $n - 1$ Kanten



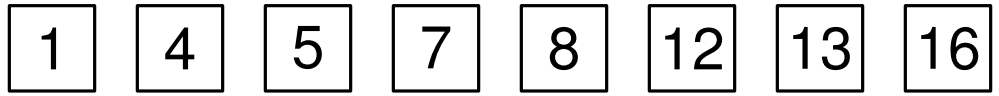
Rot-Schwarz-Bäume – Traversieren

Tiefensuche: **In-Order-Traversierung**

- Knotenreihenfolge: Kleineres Kind zuerst
- Startknoten: Wurzel
- Linker Teilbaum fertig? $\rightarrow$ Schlüssel ausgeben



Wir erhalten die Schlüssel in aufsteigender Reihenfolge!

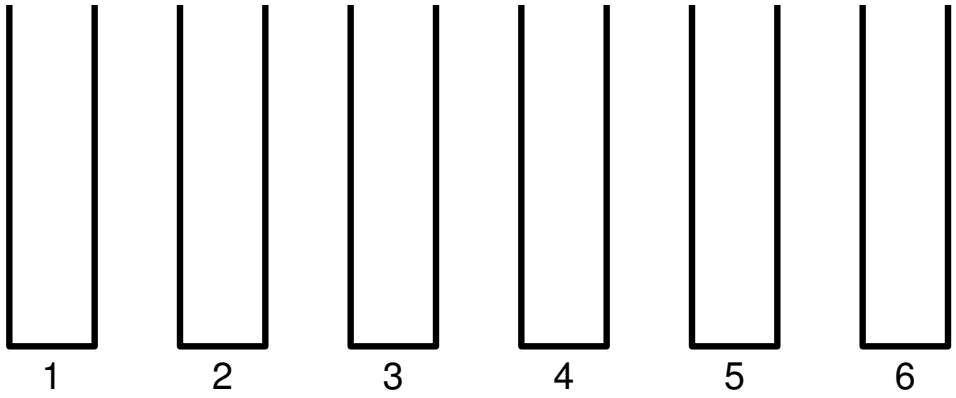


Laufzeit?

- Jede Kante wird genau zwei Mal abgelaufen
- Der Baum hat $n - 1$ Kanten
- Die sortierte Folge kann in Linearzeit aus dem Baum abgelesen werden.

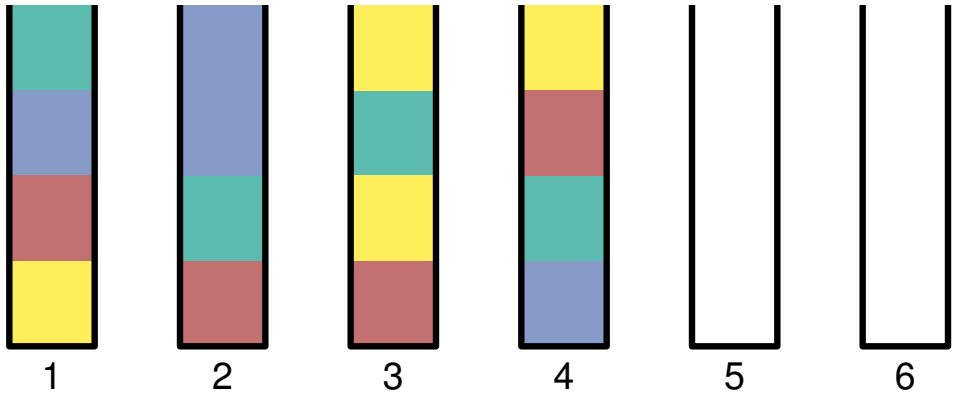
Suche in Graphen – Ein Spiel

- wir haben eine Menge von Gläsern



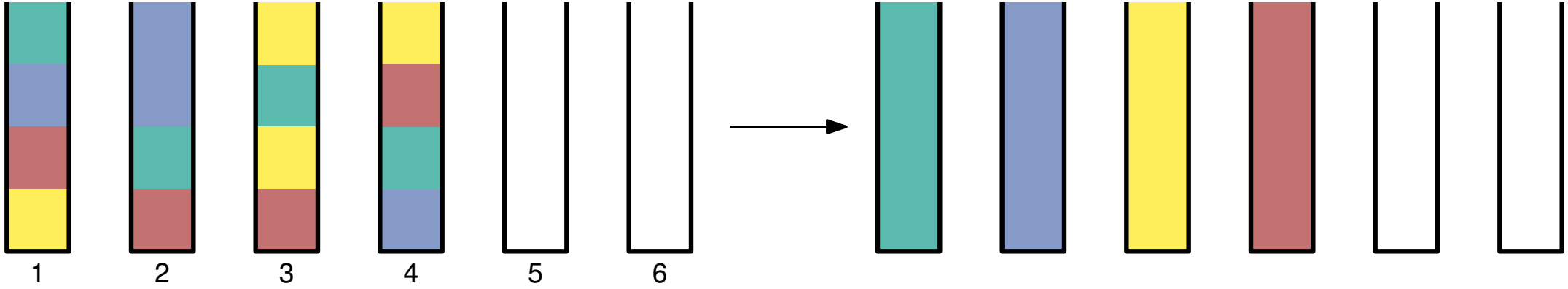
Suche in Graphen – Ein Spiel

- wir haben eine Menge von Gläsern
- darin sind (alkoholfreie) Cocktails bestehend aus verschiedenen Säften



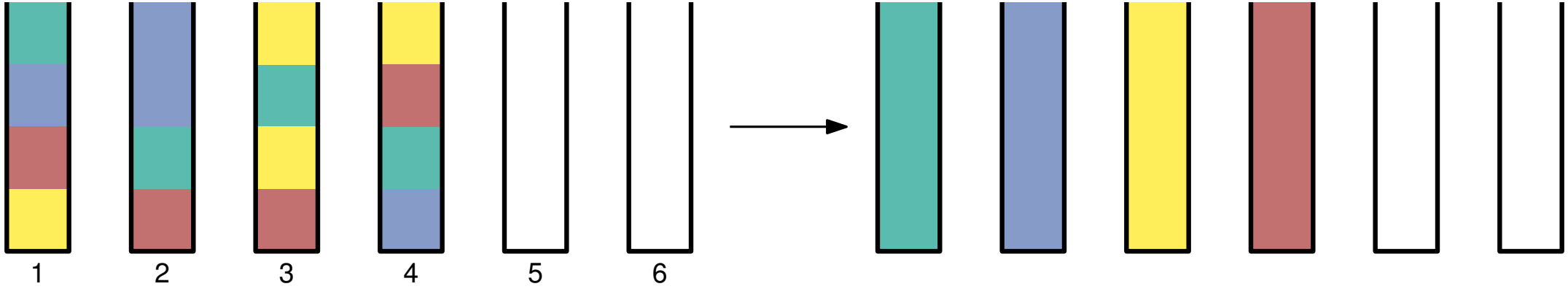
Suche in Graphen – Ein Spiel

- wir haben eine Menge von Gläsern
- darin sind (alkoholfreie) Cocktails bestehend aus verschiedenen Säften
- **Ziel**: Entmischen (jeder Saft für sich in einem Glas)
  - Gesucht: Abfolge von erlaubten Umfüll-Operationen



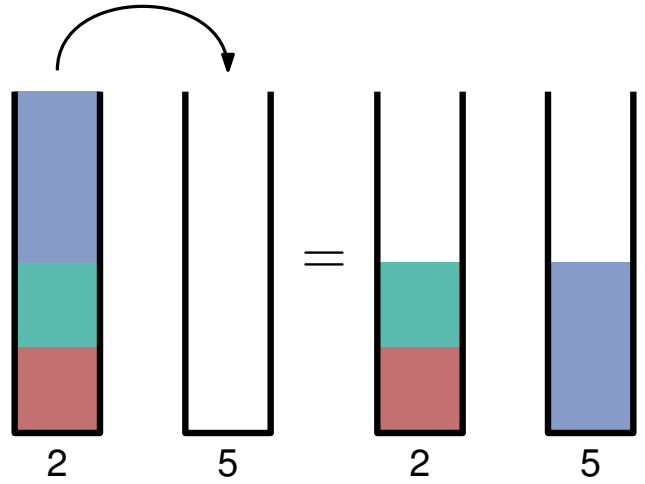
Suche in Graphen – Ein Spiel

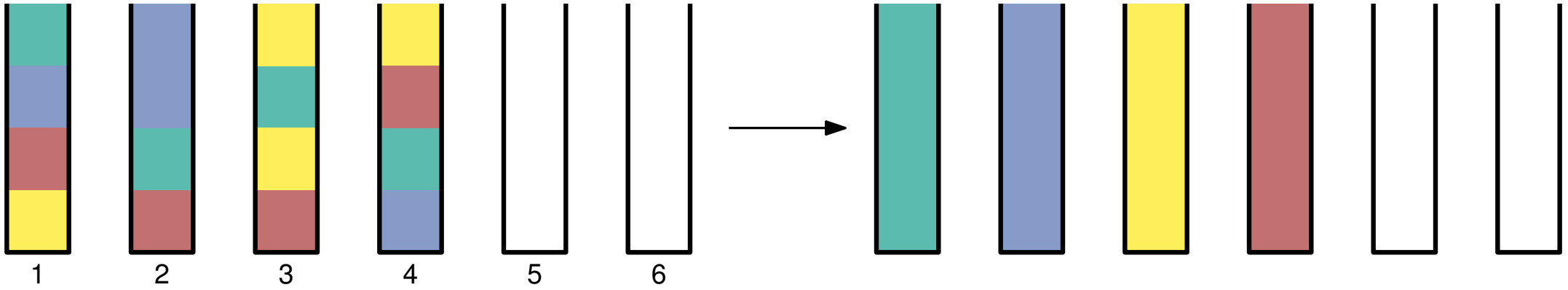
- wir haben eine Menge von Gläsern
- darin sind (alkoholfreie) Cocktails bestehend aus verschiedenen Säften
- **Ziel**: Entmischen (jeder Saft für sich in einem Glas)
  - Gesucht: Abfolge von erlaubten Umfüll-Operationen
- Regeln
  - ein Glas hält ≤ 4 Einheiten, es gibt 4 Einheiten pro Saft



Suche in Graphen – Ein Spiel

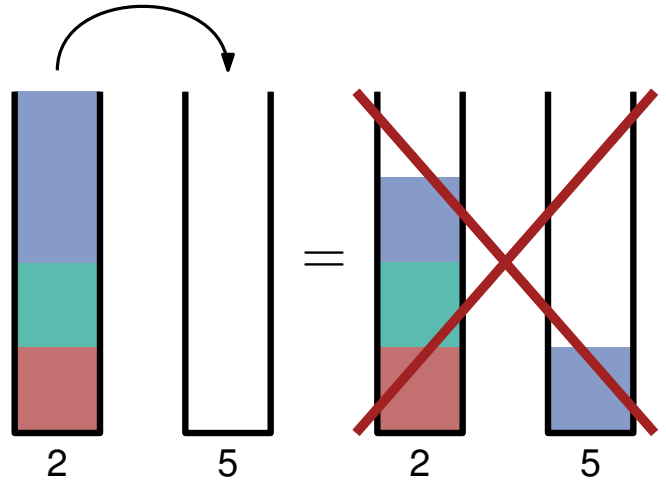
- wir haben eine Menge von Gläsern
- darin sind (alkoholfreie) Cocktails bestehend aus verschiedenen Säften
- **Ziel**: Entmischen (jeder Saft für sich in einem Glas)
  - Gesucht: Abfolge von erlaubten Umfüll-Operationen
- Regeln
  - ein Glas hält ≤ 4 Einheiten, es gibt 4 Einheiten pro Saft
  - ein Saft wird immer in Gänze umgefüllt

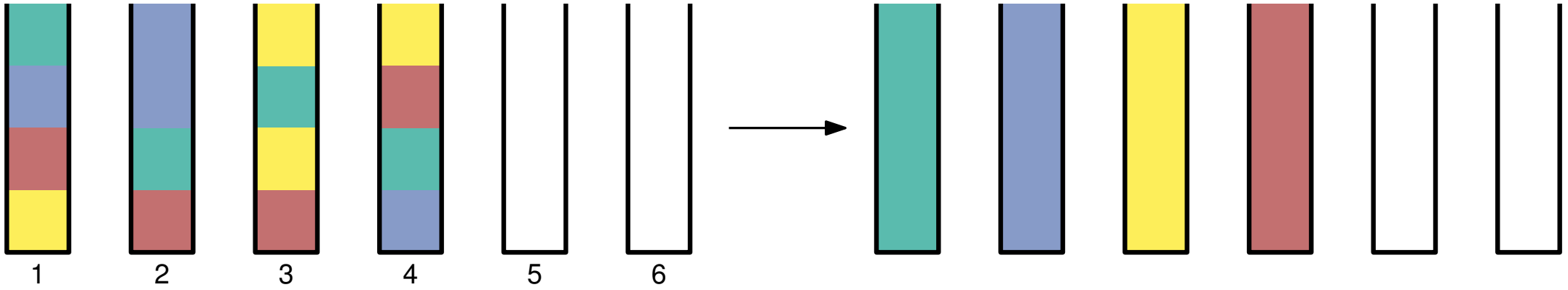




Suche in Graphen – Ein Spiel

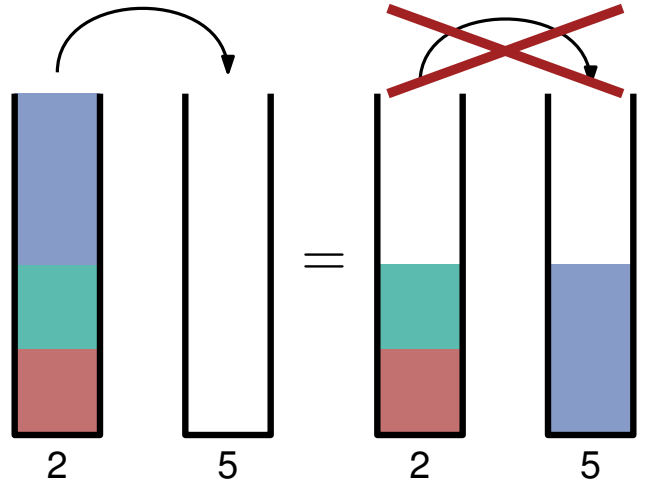
- wir haben eine Menge von Gläsern
- darin sind (alkoholfreie) Cocktails bestehend aus verschiedenen Säften
- **Ziel**: Entmischen (jeder Saft für sich in einem Glas)
  - Gesucht: Abfolge von erlaubten Umfüll-Operationen
- Regeln
  - ein Glas hält ≤ 4 Einheiten, es gibt 4 Einheiten pro Saft
  - ein Saft wird immer in Gänze umgefüllt

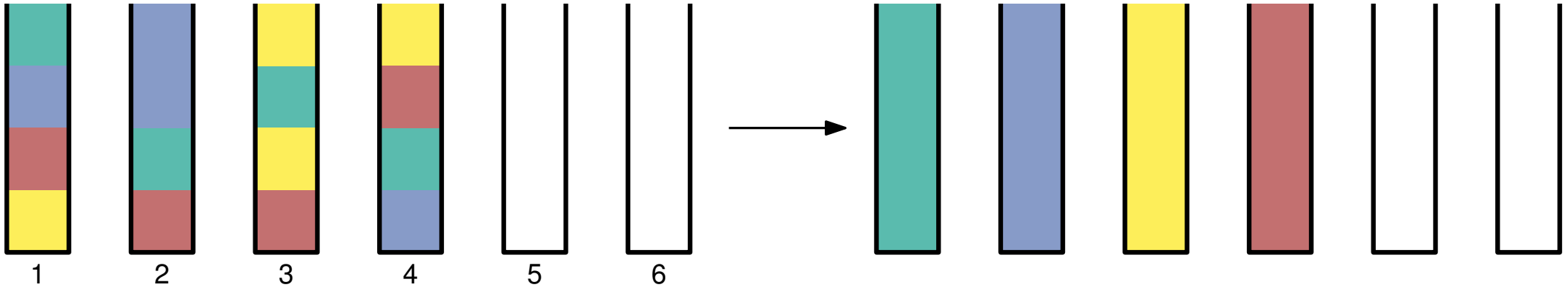




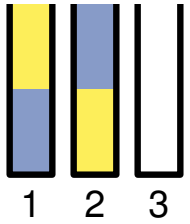
Suche in Graphen – Ein Spiel

- wir haben eine Menge von Gläsern
- darin sind (alkoholfreie) Cocktails bestehend aus verschiedenen Säften
- **Ziel**: Entmischen (jeder Saft für sich in einem Glas)
  - Gesucht: Abfolge von erlaubten Umfüll-Operationen
- Regeln
  - ein Glas hält ≤ 4 Einheiten, es gibt 4 Einheiten pro Saft
  - ein Saft wird immer in Gänze umgefüllt
  - keinen Saft auf einen anderen kippen

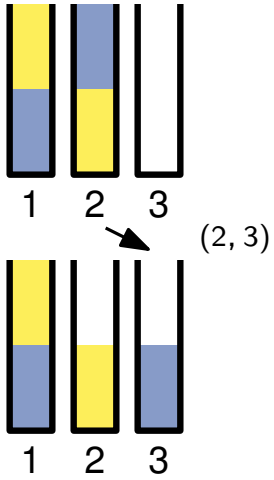




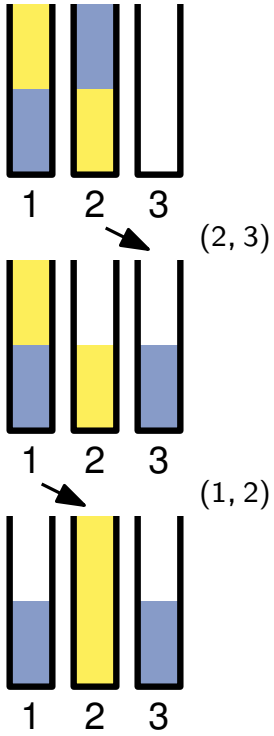
Cocktails – Beispiel & Modellierung



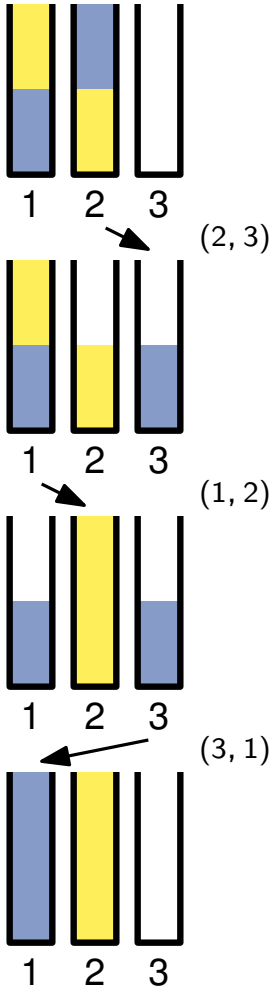
Cocktails – Beispiel & Modellierung



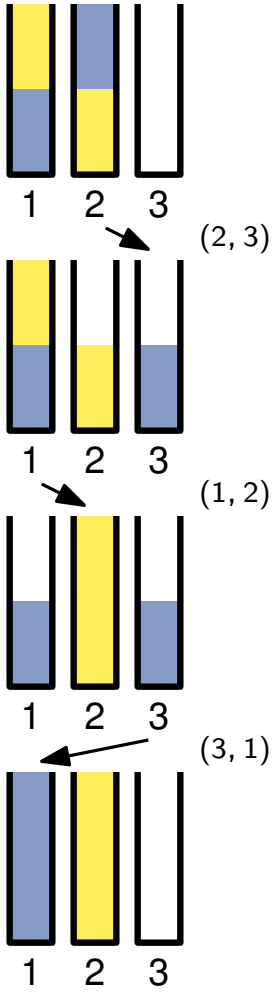
Cocktails – Beispiel & Modellierung



Cocktails – Beispiel & Modellierung

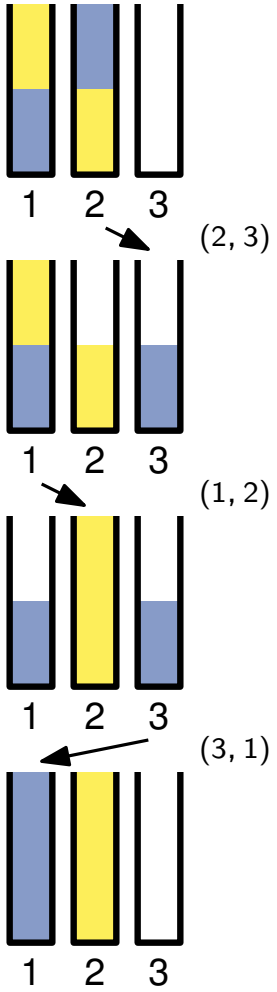


Cocktails – Beispiel & Modellierung



- Wie lösen wir dieses Problem algorithmisch?

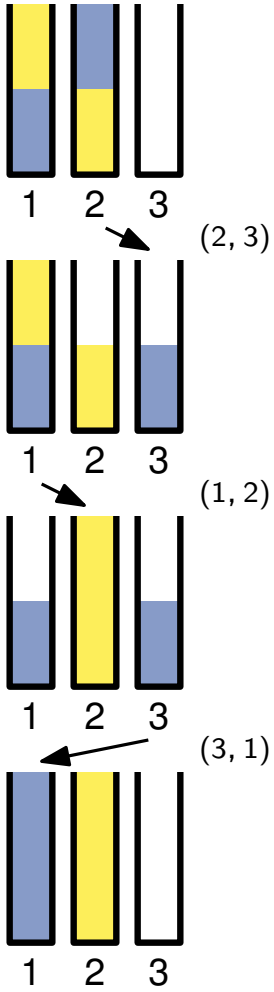
Cocktails – Beispiel & Modellierung



- Wie lösen wir dieses Problem algorithmisch?

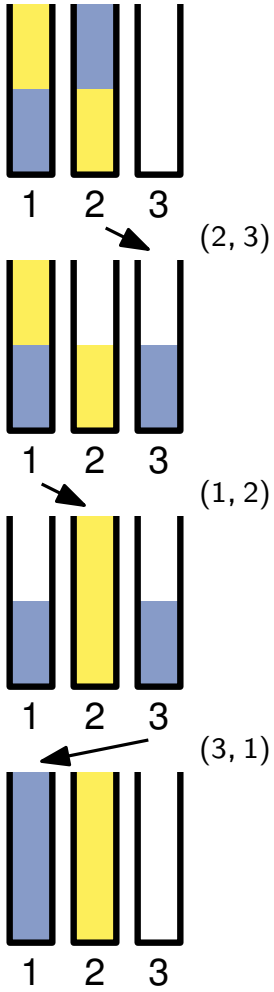
https://imgflip.com/memegenerator

Cocktails – Beispiel & Modellierung



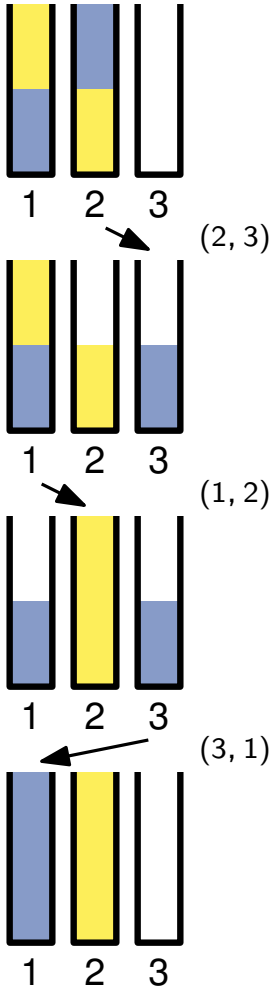
- Wie lösen wir dieses Problem algorithmisch?
  - Beginne mit Startkonfiguration

Cocktails – Beispiel & Modellierung



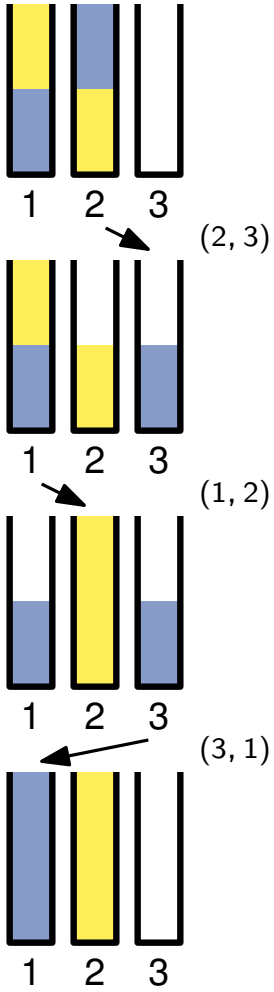
- Wie lösen wir dieses Problem algorithmisch?
  - Beginne mit Startkonfiguration
  - Führe irgendeine erlaubte Umfüll-Operation durch

Cocktails – Beispiel & Modellierung



- Wie lösen wir dieses Problem algorithmisch?
  - Beginne mit Startkonfiguration
  - Führe irgendeine erlaubte Umfüll-Operation durch
  - ... wiederholt ...
  - ... bis entmischte Konfiguration erreicht

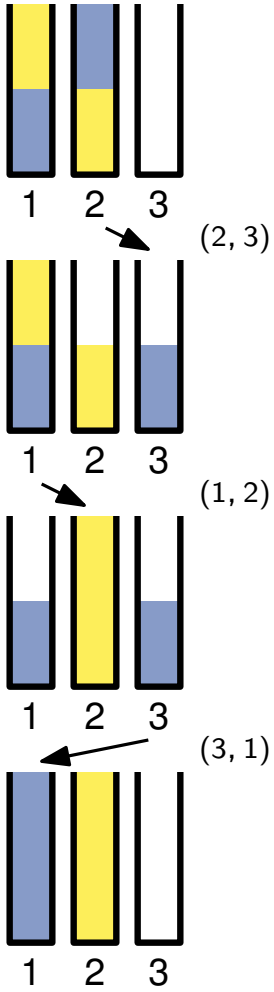
Cocktails – Beispiel & Modellierung



- Wie lösen wir dieses Problem algorithmisch?
  - Beginne mit Startkonfiguration
  - Führe irgendeine erlaubte Umfüll-Operation durch
  - ... wiederholt ...
  - ... bis entmischte Konfiguration erreicht

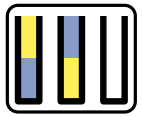
Wie verwaltet man das sinnvoll?

Cocktails – Beispiel & Modellierung

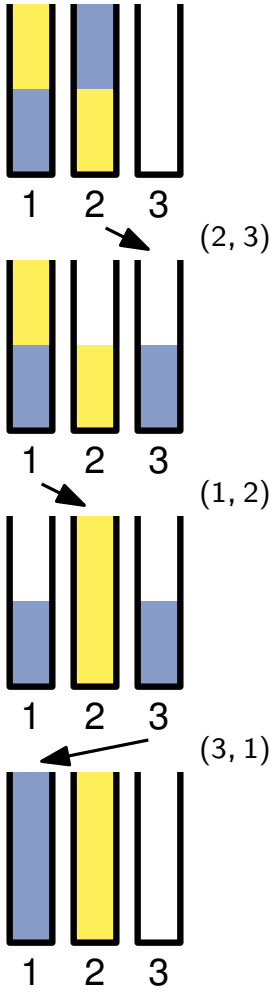


- Wie lösen wir dieses Problem algorithmisch?
  - Beginne mit Startkonfiguration
  - Führe irgendeine erlaubte Umfüll-Operation durch
  - ... wiederholt ...
  - ... bis entmischte Konfiguration erreicht
- Modellierung als Graph
  - eine Konfiguration ist ein Knoten

Wie verwaltet man das sinnvoll?

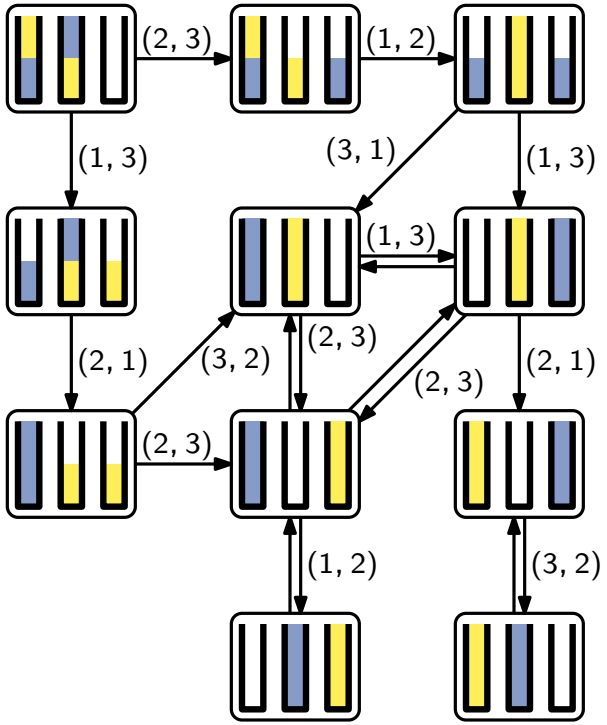


Cocktails – Beispiel & Modellierung

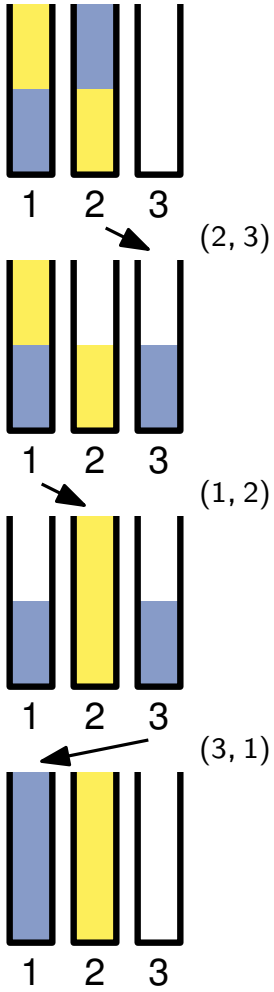


- Wie lösen wir dieses Problem algorithmisch?
  - Beginne mit Startkonfiguration
  - Führe irgendeine erlaubte Umfüll-Operation durch
  - ... wiederholt ...
  - ... bis entmischte Konfiguration erreicht
- Modellierung als Graph
  - eine Konfiguration ist ein Knoten
  - Kanten zwischen Konfigurationen die man mit erlaubten Operationen ineinander überführen kann

Wie verwaltet man das sinnvoll?

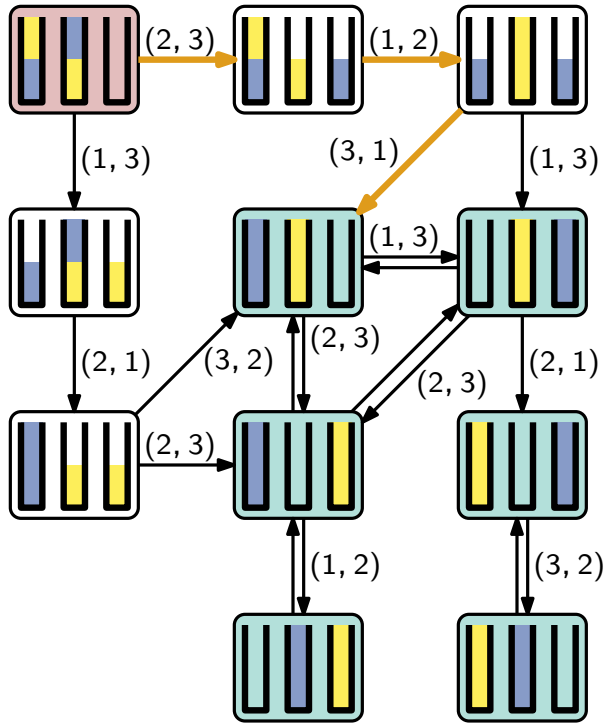


Cocktails – Beispiel & Modellierung

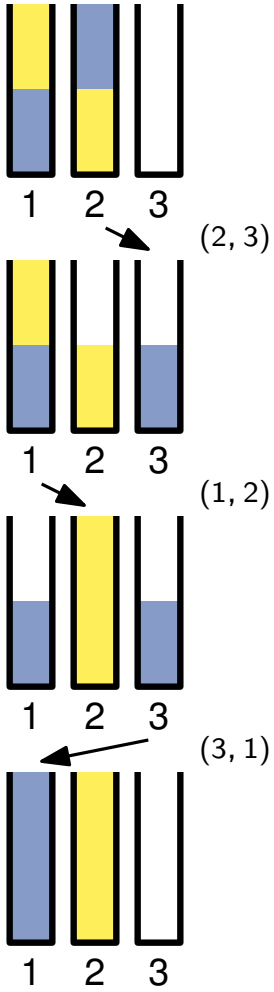


- Wie lösen wir dieses Problem algorithmisch?
  - Beginne mit Startkonfiguration
  - Führe irgendeine erlaubte Umfüll-Operation durch
  - ... wiederholt ...
  - ... bis entmischte Konfiguration erreicht
- Modellierung als Graph
  - eine Konfiguration ist ein Knoten
  - Kanten zwischen Konfigurationen die man mit erlaubten Operationen ineinander überführen kann
  - Eine Lösung ist ein Pfad von der Startkonfiguration zu einer entmischten Zielkonfiguration

Wie verwaltet man das sinnvoll?



Cocktails – Beispiel & Modellierung

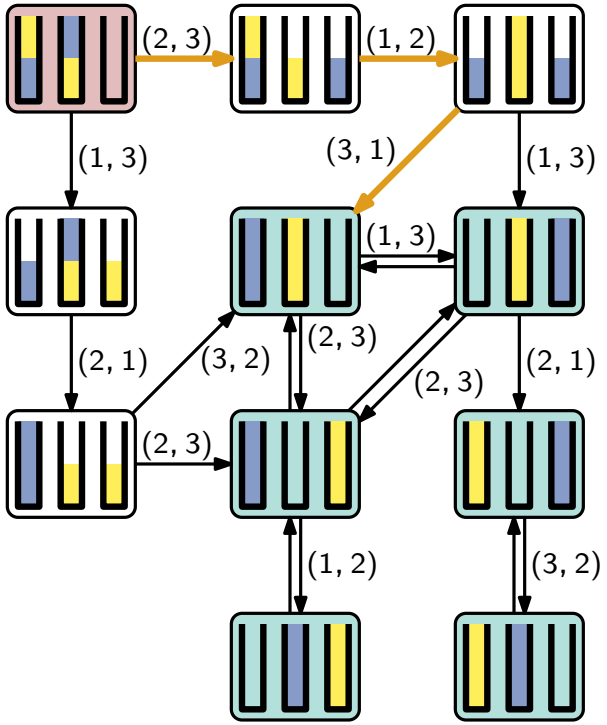


- Wie lösen wir dieses Problem algorithmisch?
  - Beginne mit Startkonfiguration
  - Führe irgendeine erlaubte Umfüll-Operation durch
  - ... wiederholt ...
  - ... bis entmischte Konfiguration erreicht
- Modellierung als Graph
  - eine Konfiguration ist ein Knoten
  - Kanten zwischen Konfigurationen die man mit erlaubten Operationen ineinander überführen kann
  - Eine Lösung ist ein Pfad von der Startkonfiguration zu einer entmischten Zielkonfiguration

Wir wissen schon, wie man Wege in Graphen sucht!

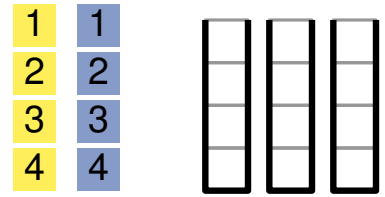
... aber es gibt da ein Problem...

Wie verwaltet man das sinnvoll?



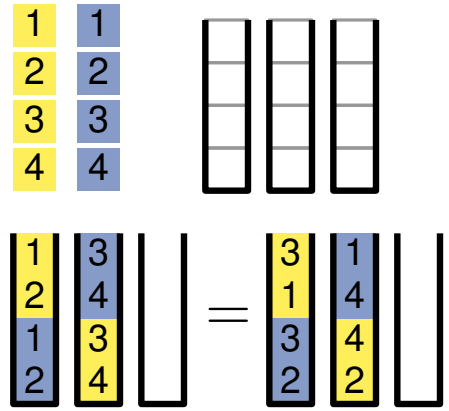
Cocktails – Das Problem

- k Säfte, ℓ Gläser
  - $4k$ Safteinheiten, 4ℓ Glaseinheiten
  - $\binom{4\ell}{4k}$ Möglichkeiten Safteinheiten auf Gläser aufzuteilen



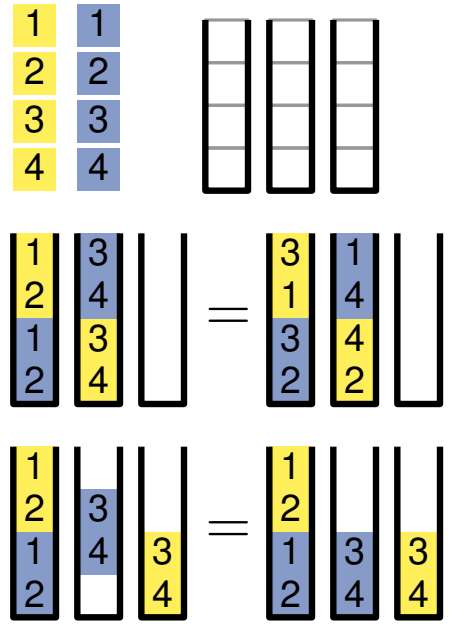
Cocktails – Das Problem

- k Säfte, ℓ Gläser
  - $4k$ Safteinheiten, 4ℓ Glaseinheiten
  - $\binom{4\ell}{4k}$ Möglichkeiten Safteinheiten auf Gläser aufzuteilen
  - $k \cdot 4!$ Möglichkeiten Safteinheiten gleicher Säfte zu vertauschen



Cocktails – Das Problem

- k Säfte, ℓ Gläser
  - $4k$ Safteinheiten, 4ℓ Glaseinheiten
  - $\binom{4\ell}{4k}$ Möglichkeiten Safteinheiten auf Gläser aufzuteilen
  - $k \cdot 4!$ Möglichkeiten Safteinheiten gleicher Säfte zu vertauschen
  - Schwerkraft: Pro Glas $2^4 - 5$ Füllungen mit Luftblasen

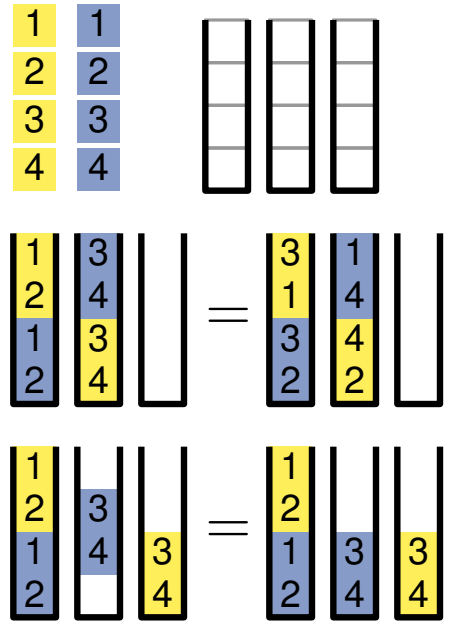


Cocktails – Das Problem

- k Säfte, ℓ Gläser
  - $4k$ Safteinheiten, 4ℓ Glaseinheiten
  - $\binom{4\ell}{4k}$ Möglichkeiten Safteinheiten auf Gläser aufzuteilen
  - $k \cdot 4!$ Möglichkeiten Safteinheiten gleicher Säfte zu vertauschen
  - Schwerkraft: Pro Glas $2^4 - 5$ Füllungen mit Luftblasen

Anzahl Konfigurationen:

$$\approx \frac{\binom{4\ell}{4k}}{k \cdot 4! \cdot k \cdot (2^4 - 5)}$$



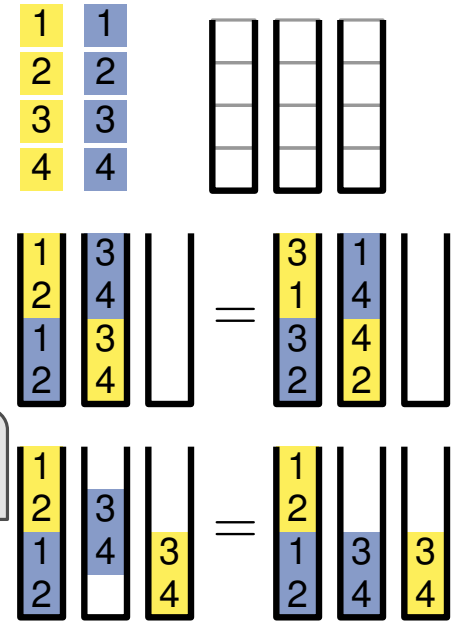
Cocktails – Das Problem

- k Säfte, ℓ Gläser
  - $4k$ Safteinheiten, 4ℓ Glaseinheiten
  - $\binom{4\ell}{4k}$ Möglichkeiten Safteinheiten auf Gläser aufzuteilen
  - $k \cdot 4!$ Möglichkeiten Safteinheiten gleicher Säfte zu vertauschen
  - Schwerkraft: Pro Glas $2^4 - 5$ Füllungen mit Luftblasen

Anzahl Konfigurationen:

$$\approx \frac{\binom{4\ell}{4k}}{k \cdot 4! \cdot k \cdot (2^4 - 5)} \approx \frac{(4\ell)^{4k}}{k^2 \cdot (4k)!}$$

$\binom{\ell}{k} \approx \frac{\ell^k}{k!}$



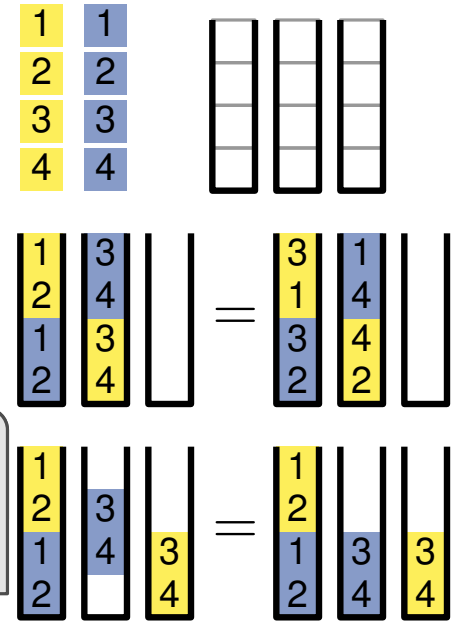
Cocktails – Das Problem

- k Säfte, ℓ Gläser
  - $4k$ Safteinheiten, 4ℓ Glaseinheiten
  - $\binom{4\ell}{4k}$ Möglichkeiten Safteinheiten auf Gläser aufzuteilen
  - $k \cdot 4!$ Möglichkeiten Safteinheiten gleicher Säfte zu vertauschen
  - Schwerkraft: Pro Glas $2^4 - 5$ Füllungen mit Luftblasen

Anzahl Konfigurationen:

$$\approx \frac{\binom{4\ell}{4k}}{k \cdot 4! \cdot k \cdot (2^4 - 5)} \approx \frac{(4\ell)^{4k}}{k^2 \cdot (4k)!} \approx \frac{\left(e\frac{4\ell}{4k}\right)^{4k}}{k^2}$$

$$\binom{\ell}{k} \approx \frac{\ell^k}{k!}$$
$$k! \approx \left(\frac{k}{e}\right)^k$$



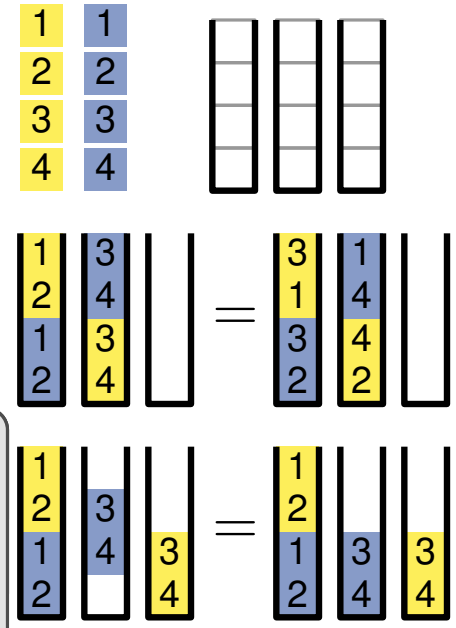
Cocktails – Das Problem

- k Säfte, ℓ Gläser
  - $4k$ Safteinheiten, 4ℓ Glaseinheiten
  - $\binom{4\ell}{4k}$ Möglichkeiten Safteinheiten auf Gläser aufzuteilen
  - $k \cdot 4!$ Möglichkeiten Safteinheiten gleicher Säfte zu vertauschen
  - Schwerkraft: Pro Glas $2^4 - 5$ Füllungen mit Luftblasen

Anzahl Konfigurationen:

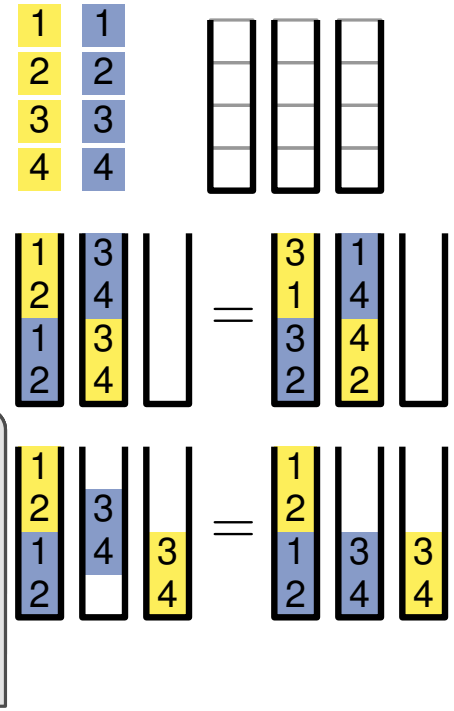
$$\approx \frac{\binom{4\ell}{4k}}{k \cdot 4! \cdot k \cdot (2^4 - 5)} \approx \frac{(4\ell)^{4k}}{k^2 \cdot (4k)!} \approx \frac{\left(e\frac{4\ell}{4k}\right)^{4k}}{k^2} \geq \frac{e^{4k}}{k^2}$$

$$\binom{\ell}{k} \approx \frac{\ell^k}{k!}$$
$$k! \approx \left(\frac{k}{e}\right)^k$$
$$\ell \geq k$$



Cocktails – Das Problem

- k Säfte, ℓ Gläser
  - $4k$ Safteinheiten, 4ℓ Glaseinheiten
  - $\binom{4\ell}{4k}$ Möglichkeiten Safteinheiten auf Gläser aufzuteilen
  - $k \cdot 4!$ Möglichkeiten Safteinheiten gleicher Säfte zu vertauschen
  - Schwerkraft: Pro Glas $2^4 - 5$ Füllungen mit Luftblasen



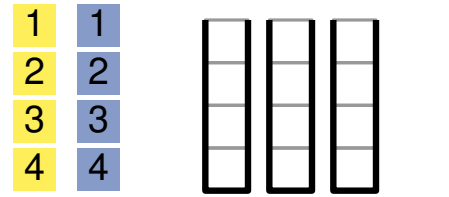
Anzahl Konfigurationen:

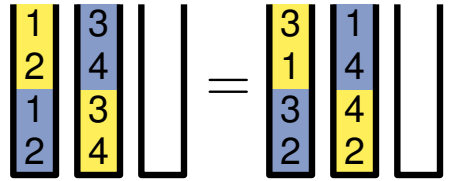
$$\approx \frac{\binom{4\ell}{4k}}{k \cdot 4! \cdot k \cdot (2^4 - 5)} \approx \frac{(4\ell)^{4k}}{k^2 \cdot (4k)!} \approx \frac{\left(e\frac{4\ell}{4k}\right)^{4k}}{k^2} \geq \frac{e^{4k}}{k^2} \geq e^{3k}$$

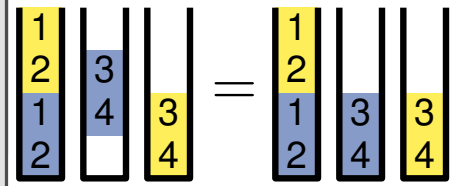
$$\binom{\ell}{k} \approx \frac{\ell^k}{k!}$$
$$k! \approx \left(\frac{k}{e}\right)^k$$
$$\ell \geq k$$
$$e^k \geq k^2$$

Cocktails – Das Problem

- k Säfte, ℓ Gläser
  - $4k$ Safteinheiten, 4ℓ Glaseinheiten
  - $\binom{4\ell}{4k}$ Möglichkeiten Safteinheiten auf Gläser aufzuteilen
  - $k \cdot 4!$ Möglichkeiten Safteinheiten gleicher Säfte zu vertauschen
  - Schwerkraft: Pro Glas $2^4 - 5$ Füllungen mit Luftblasen







Anzahl Konfigurationen:

$$\approx \frac{\binom{4\ell}{4k}}{k \cdot 4! \cdot k \cdot (2^4 - 5)} \approx \frac{(4\ell)^{4k}}{k^2 \cdot (4k)!} \approx \frac{\left(e\frac{4\ell}{4k}\right)^{4k}}{k^2} \geq \frac{e^{4k}}{k^2} \geq e^{3k}$$

$$\binom{\ell}{k} \approx \frac{\ell^k}{k!}$$
$$k! \approx \left(\frac{k}{e}\right)^k$$
$$\ell \geq k$$
$$e^k \geq k^2$$

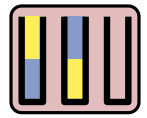
- Die Anzahl möglicher Konfigurationen ist **exponentiell** in der Anzahl Säfte.
  - Anzahl Knoten im Graph viel zu hoch
  - Graph aufbauen benötigt zu viel Zeit / Speicher
  - Und dann müssen wir in dem Graphen auch noch Pfade finden...

Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren

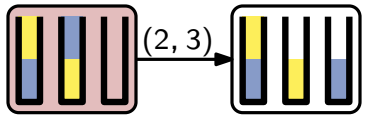
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen



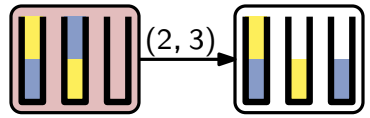
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen



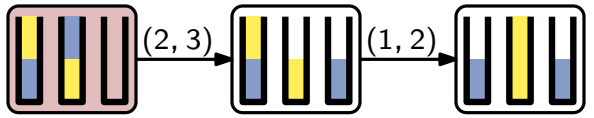
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert



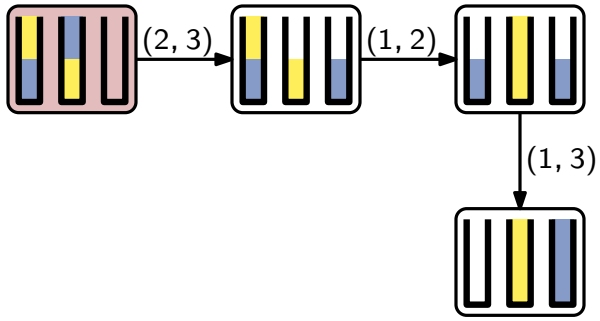
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert



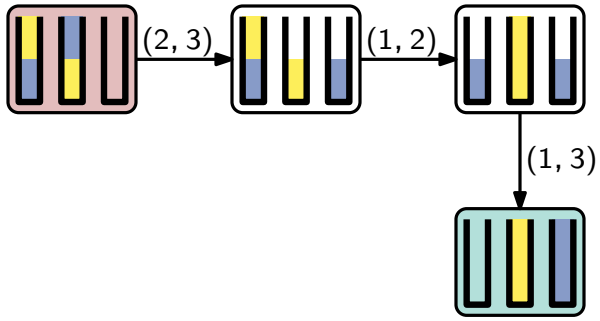
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert



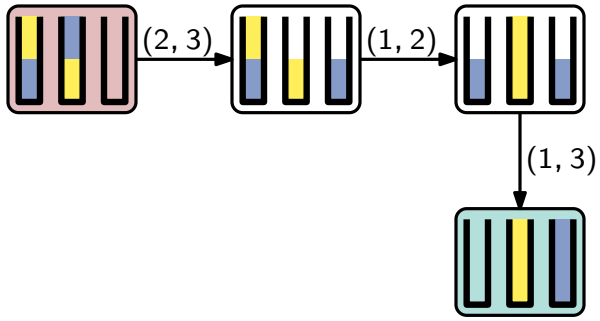
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert



Cocktails – Die Lösung

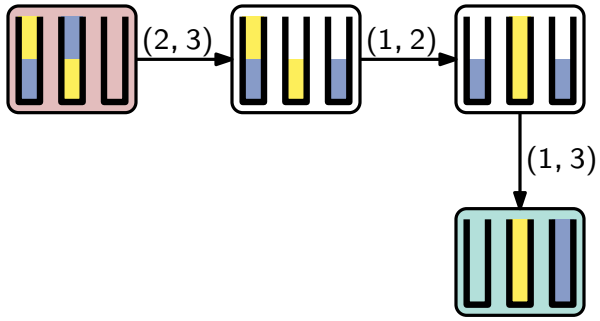
- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert



Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

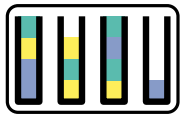
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert



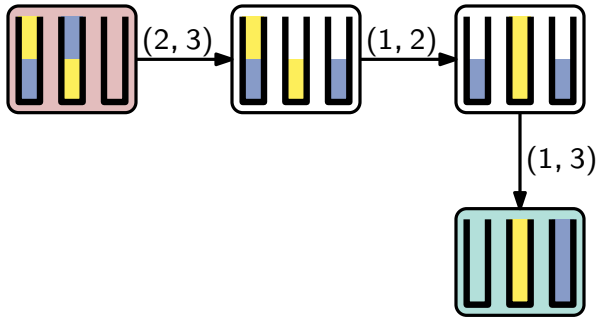
Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden



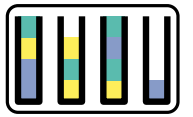
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert



Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

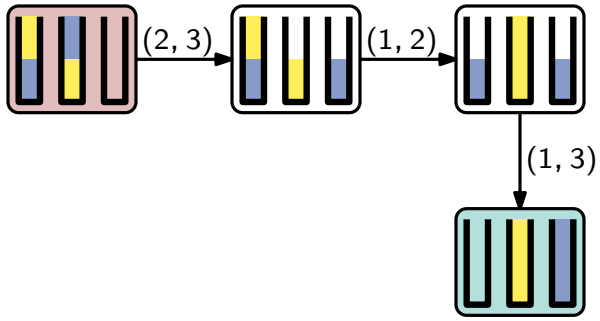
- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden



- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?

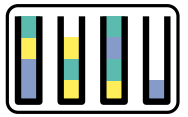
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert

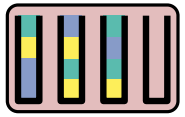


Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden

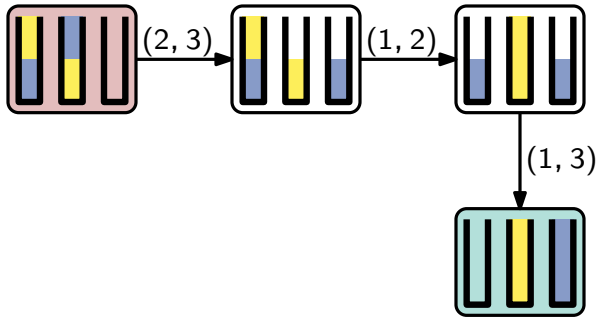


- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?



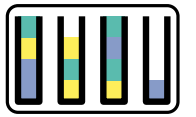
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert

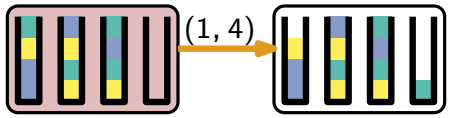


Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden

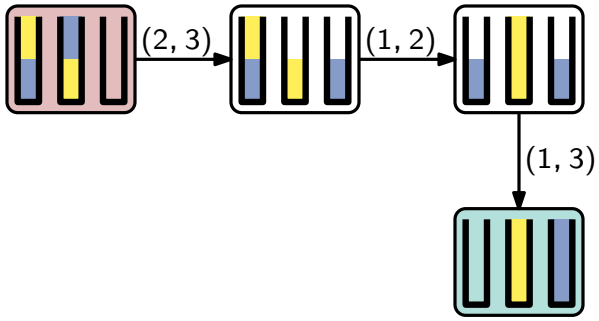


- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?



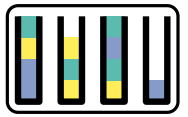
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert

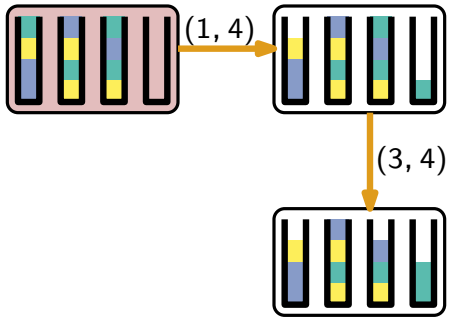


Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden

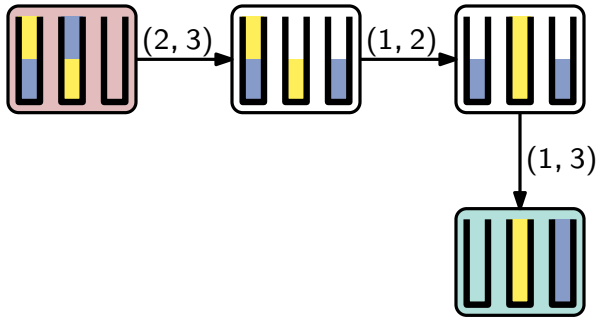


- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?



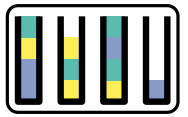
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert

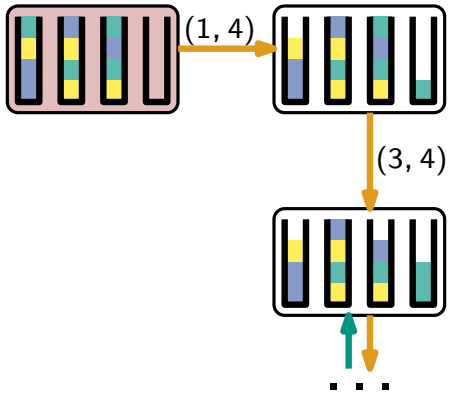


Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden

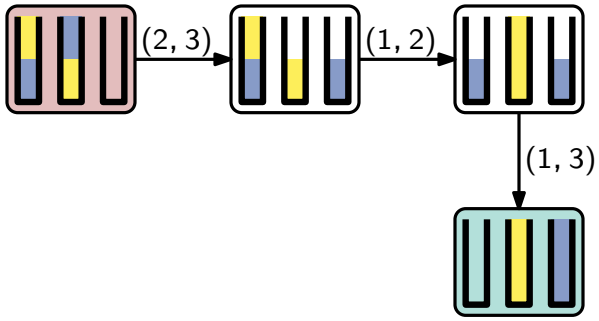


- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?



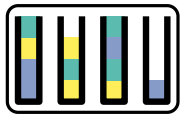
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert

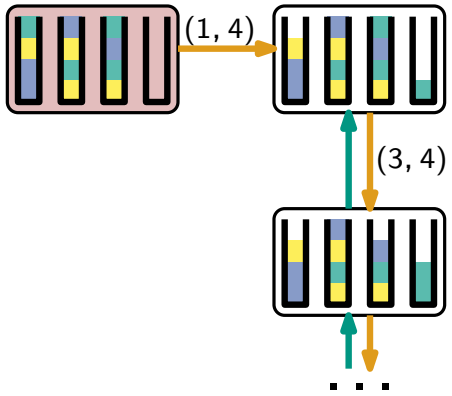


Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden

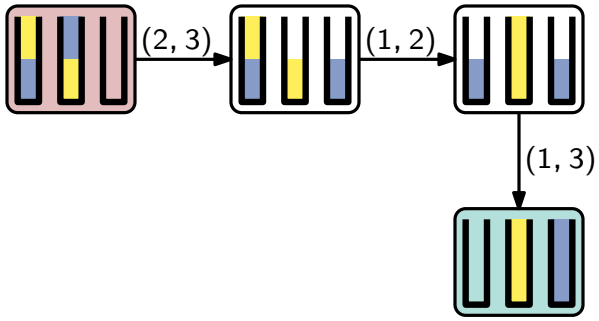


- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?



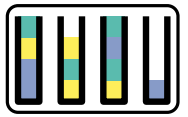
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert

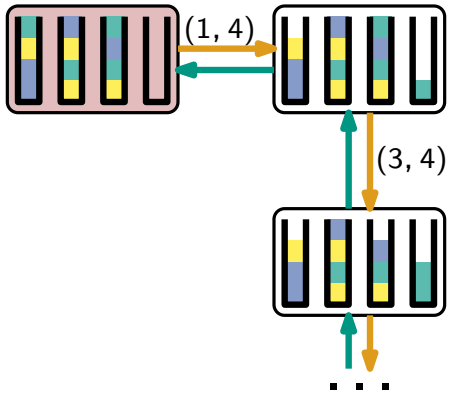


Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden

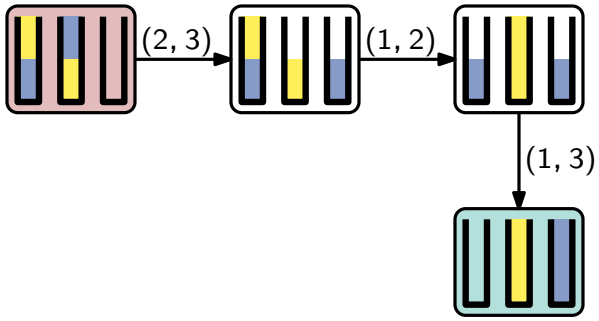


- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?



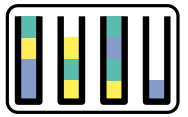
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert

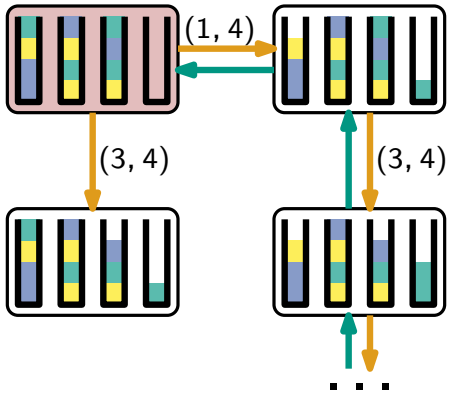


Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden

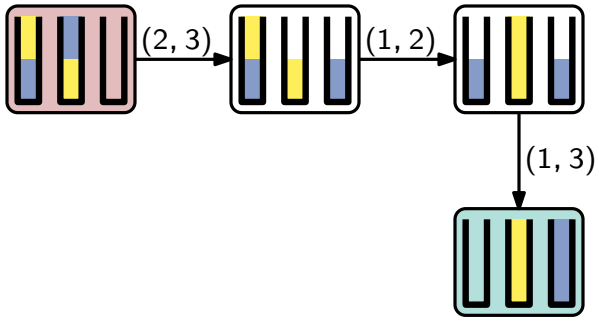


- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?



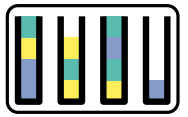
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert

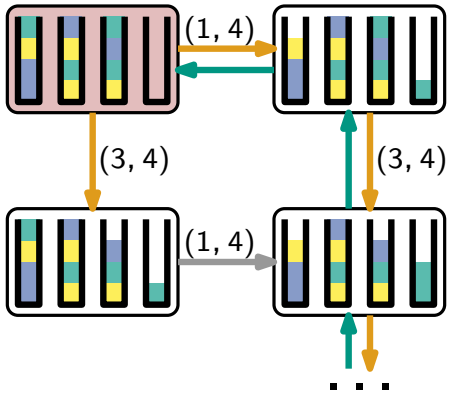


Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden

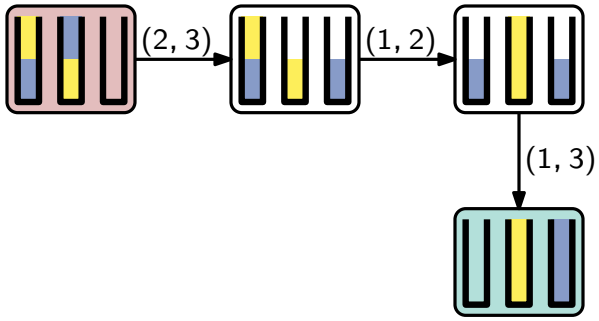


- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?



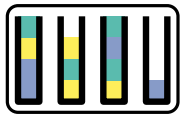
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert

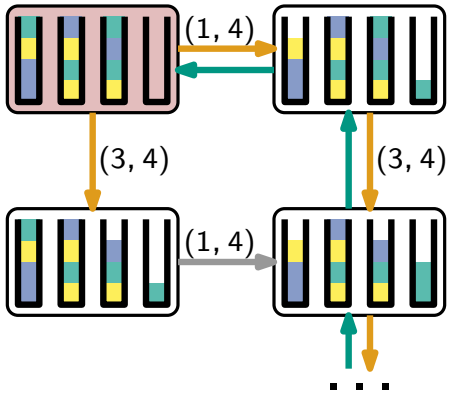


Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden



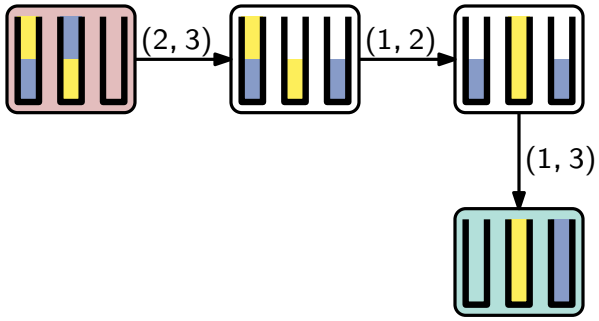
- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?



- Verwalte Hashtabelle von bereits gesehenen Konfigurationen!

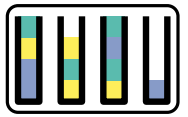
Cocktails – Die Lösung

- Idee: Graph aufbauen während wir ihn explorieren
  - Startknoten erzeugen
  - eine erlaubte Umfüll-Operationen bestimmen und zugehörige Kante / Knoten in den Graph einfügen
  - Testen ob neuer Knoten eine Zielkonfiguration repräsentiert

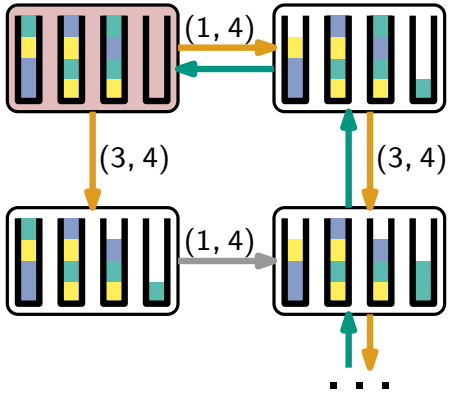


Dieses Verfahren entspricht einer Tiefensuche im Konfigurationsgraphen!

- Sackgasse: In einer Konfiguration kann nicht mehr umgefüllt werden



- Woher wissen wir, wann eine Kante die wir in den Graphen einfügen eine Rückkante ist?



- Verwalte Hashtabelle von bereits gesehenen Konfigurationen!

Breitensuche?

- Wie sieht eine Breitensuche in dem Graphen aus?
- Warum ist eine Breitensuche ggf. nicht so gut geeignet wie eine Tiefensuche?

Zusammenfassung

Suchbäume

- Einfügen, Suchen, Löschen in logarithmischer Zeit
- Beschränkte Höhe des Suchbaums durch Ausgleich von Binarität und Balanciertheit
- $(2, 3)$-Bäume vs. Rot-Schwarz-Bäume

Zusammenfassung

Suchbäume

- Einfügen, Suchen, Löschen in logarithmischer Zeit
- Beschränkte Höhe des Suchbaums durch Ausgleich von Binarität und Balanciertheit
- $(2, 3)$-Bäume vs. Rot-Schwarz-Bäume

Tiefensuche

- Freiheitsgrad: Reihenfolge der Knoten
- In-Order-Traversierung zum Auslesen von Rot-Schwarz-Bäumen

Zusammenfassung

Suchbäume

- Einfügen, Suchen, Löschen in logarithmischer Zeit
- Beschränkte Höhe des Suchbaums durch Ausgleich von Binarität und Balanciertheit
- $(2, 3)$-Bäume vs. Rot-Schwarz-Bäume

Tiefensuche

- Freiheitsgrad: Reihenfolge der Knoten
- In-Order-Traversierung zum Auslesen von Rot-Schwarz-Bäumen

Modellierung als Graphproblem

- Mittels Tiefensuche den Graph möglicher Konfigurationen Schritt für Schritt generieren und durchsuchen
- Das funktioniert auch mit Konfigurationen anderer Spiele
  - Schach, Dame, Sudoku, …
- Graphen als Framework für Spiele-Solver!