Algorithmen und Datenstrukturen (für ESE) WS 2010 / 2011

Vorlesung 8, Montag, 13. Dezember 2010
(Balancierte Suchbäume)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

- Organisatorisches
  - Zum Ü6 diese Woche (Prioritätswarteschlangen) gibt es keine Übungsgruppe sondern Fragetermin auf Anfrage
  - Ihre Erfahrungen mit dem 7. Übungsblatt
- Balancierte Suchbäume
  - Im Ü7 haben wir gesehen, dass unser BinarySearchTree Tiefe O(log n) haben kann aber nicht muss
  - Also schauen wir heute, dass er immer Tiefe O(log n) hat

Ihre Erfahrungen mit dem 7. Ü-Blatt

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback
  - Im Prinzip ok
  - Aber ziemlich "frickelig"
  - Anspruchsvoller als das letzte Übungsblatt (priority queues)
  - Keine Lust auf Bäume mehr

Balancierte Bäume

UNI FREIBURG

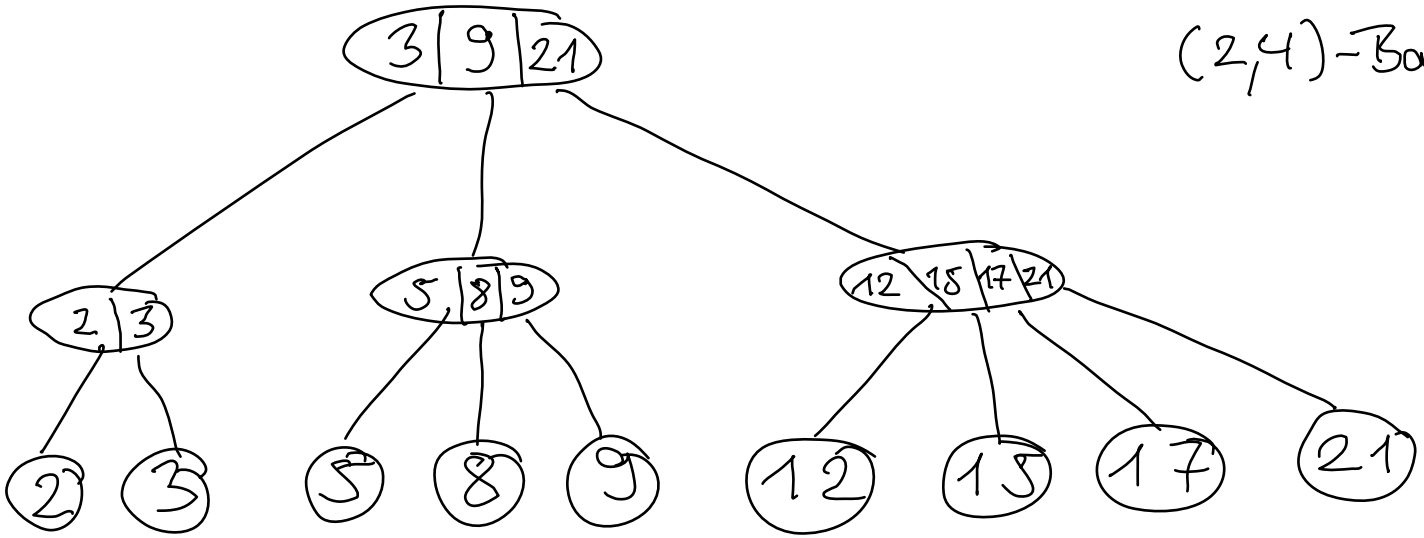
- Motivation
  - Mit BinarySearchTree hatten wir lookup und insert in Zeit O(depth), wobei depth = Tiefe des Baumes
  - Wenn es gut läuft ist depth = O(log n)
    - z.B. wenn die Schlüssel zufällig gewählt sind
  - Wenn es schlecht läuft ist depth = Θ(n)
    - z.B. wenn der Reihe nach 1, 2, 3, ... eingefügt werden
  - Wir wollen uns aber nicht auf eine bestimmte Eigenschaft der Schlüsselmenge verlassen müssen
  - Und werden uns heute deswegen explizit darum kümmern, dass der Baum immer Tiefe O(log n) hat

(a,b)-Bäume

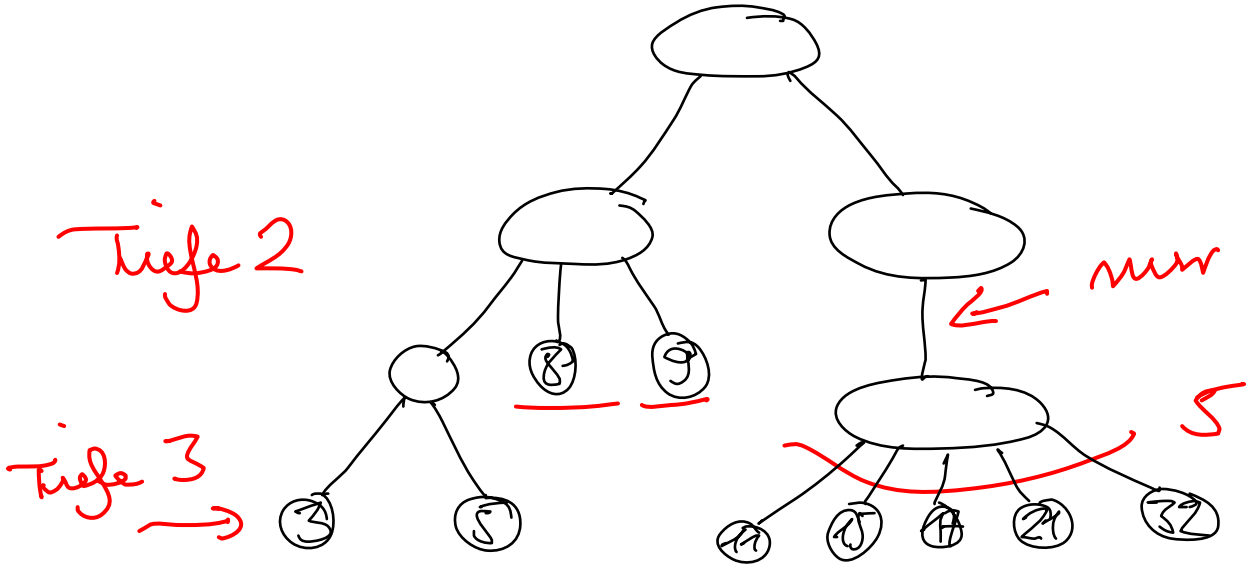
UNI FREIBURG

- Wie erreicht man immer Tiefe $O(\log n)$?
  - Es gibt Dutzende verschiedener Verfahren dafür
  - Wie schauen uns heute (a,b)-Bäume an
  - Die sind intuitiv, einfach und praktisch
- Definition (a,b)-Baum
  - Die Elemente / Schlüssel stehen nur in den Blättern
  - Alle Blätter haben die gleiche Tiefe
  - Jeder innere Knoten hat $\geq a$ und $\leq b$ Kinder
    (nur die Wurzel darf weniger Kinder haben)
  - Wir verlangen $a \geq 2$ und $b \geq 2a - 1$ Begründung folgt
  - An den inneren Knoten steht für jedes Kind der größte Schlüssel in dem Unterbaum dieses Kindes

(a,b)-Bäume — Beispiele



(2,4)-Baum



kein (2,4)-Baum

nur 1 Kind

5 Kinder

(a,b)-Bäume — Lookup

UNI FREIBURG

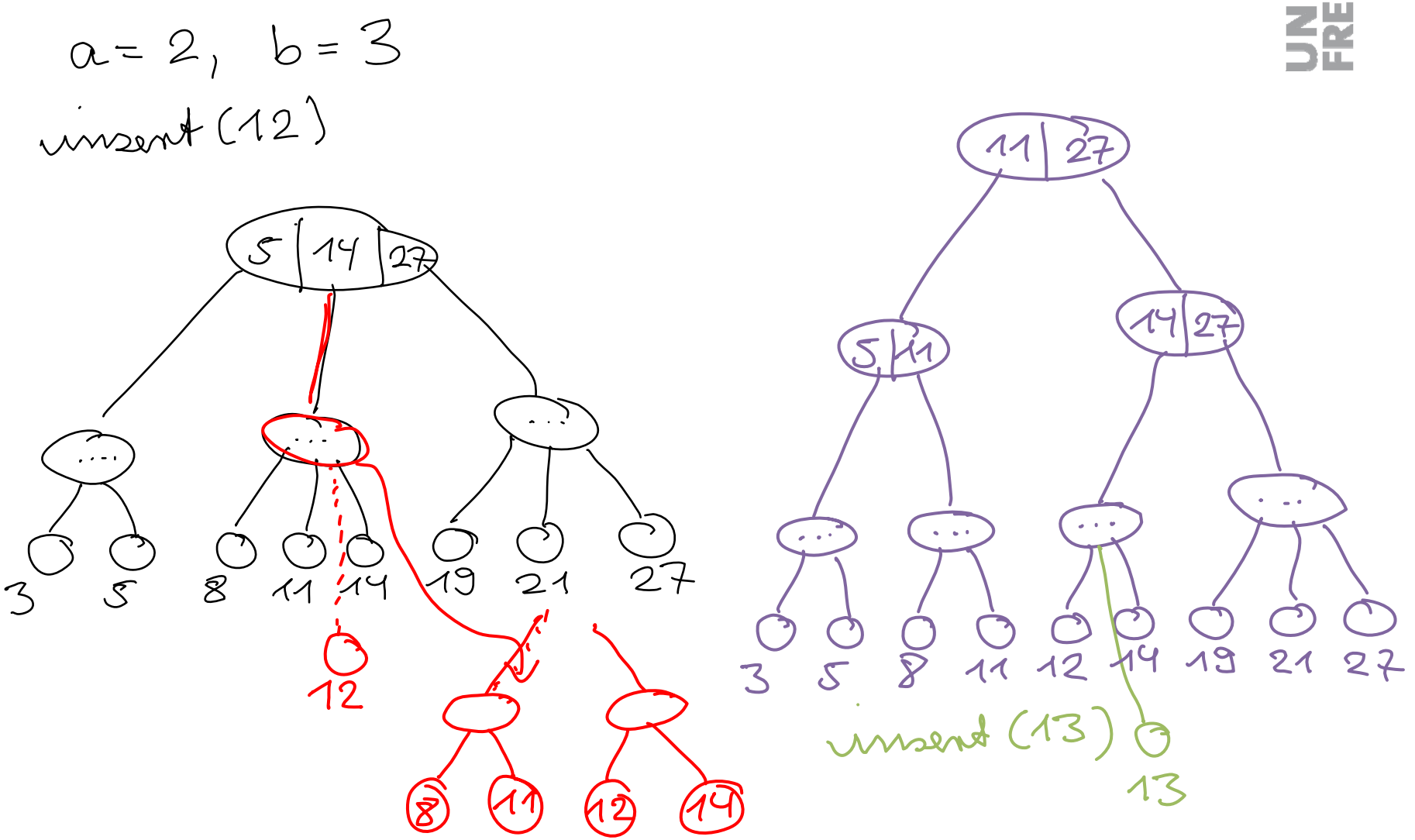
- Im Prinzip genauso wie beim BinarySearchTree
  - Suche von der Wurzel abwärts
  - Die Schlüssel an den inneren Knoten weisen den Weg

(a,b)-Bäume — Insert

UNI FREIBURG

- Einfügen eines Elementes / Schlüssels
  - Finde die Stelle, wo der neue Schlüssel einzufügen ist
  - Und füge dort ein neues Blatt ein
  - Achtung: der Elternknoten kann jetzt b+1 Knoten haben!
  - Dann spalten wir den Elternknoten einfach auf in zwei Knoten mit ceil(b/2) und floor(b/2) + 1 Kinder
    - für $b \geq 2a-1$ ist $ceil(b/2) \geq a$ und $floor(b/2) + 1 \geq a$
  - Der Großelternknoten kann jetzt b+1 Kinder haben
  - Dann spalten wir den auf dieselbe Weise auf ... usw.
  - Wenn das bis zur Wurzel geht, spalten wir auch die auf und erzeugen einen neuen Wurzelknoten → Baum wird 1 tiefer

(a,b)-Bäume — Insert

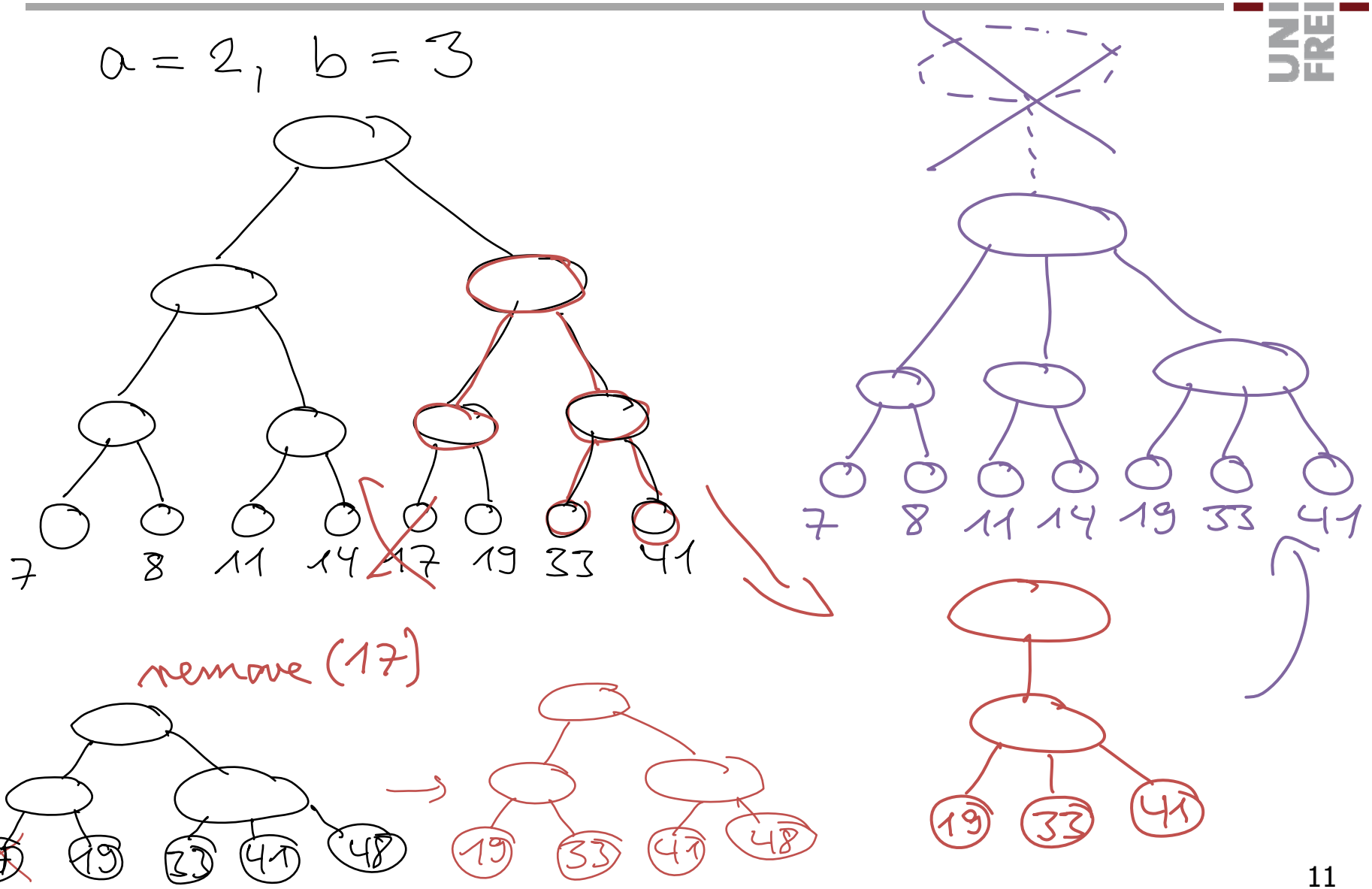


(a,b)-Bäume — Remove

UNI FREIBURG

- Entfernen eines Elementes / Schlüssels
  - Finde das zu entfernende Element
  - Und lösche das entsprechende Blatt
  - Achtung: der Elternknoten kann jetzt a-1 Kinder haben!
  - Falls eines der anderen Kinder des Elternknoten > a Kinder hat, nehmen wir eins von da weg und sind fertig
  - Sonst verschmelzen wir den Elternknoten mit einem seiner Geschwister
  - Der Großelternknoten hat jetzt ein Kind weniger und kann jetzt a-1 Kinder haben → weiter evtl. bis zur Wurzel
  - Wenn die Wurzel am Ende nur noch ein Kind hat, mache dieses Kind zur neuen Wurzel → Baum wird 1 weniger tief

(a,b)-Bäume — Remove



(a,b)-Bäume — Komplexität

$n = \#\text{Knoten}$, $d = \text{Tiefe}$

$n \geq \underbrace{a \cdot a \cdot \ldots \cdot a}_{d-1 \text{ mal}} = a^{d-1}$

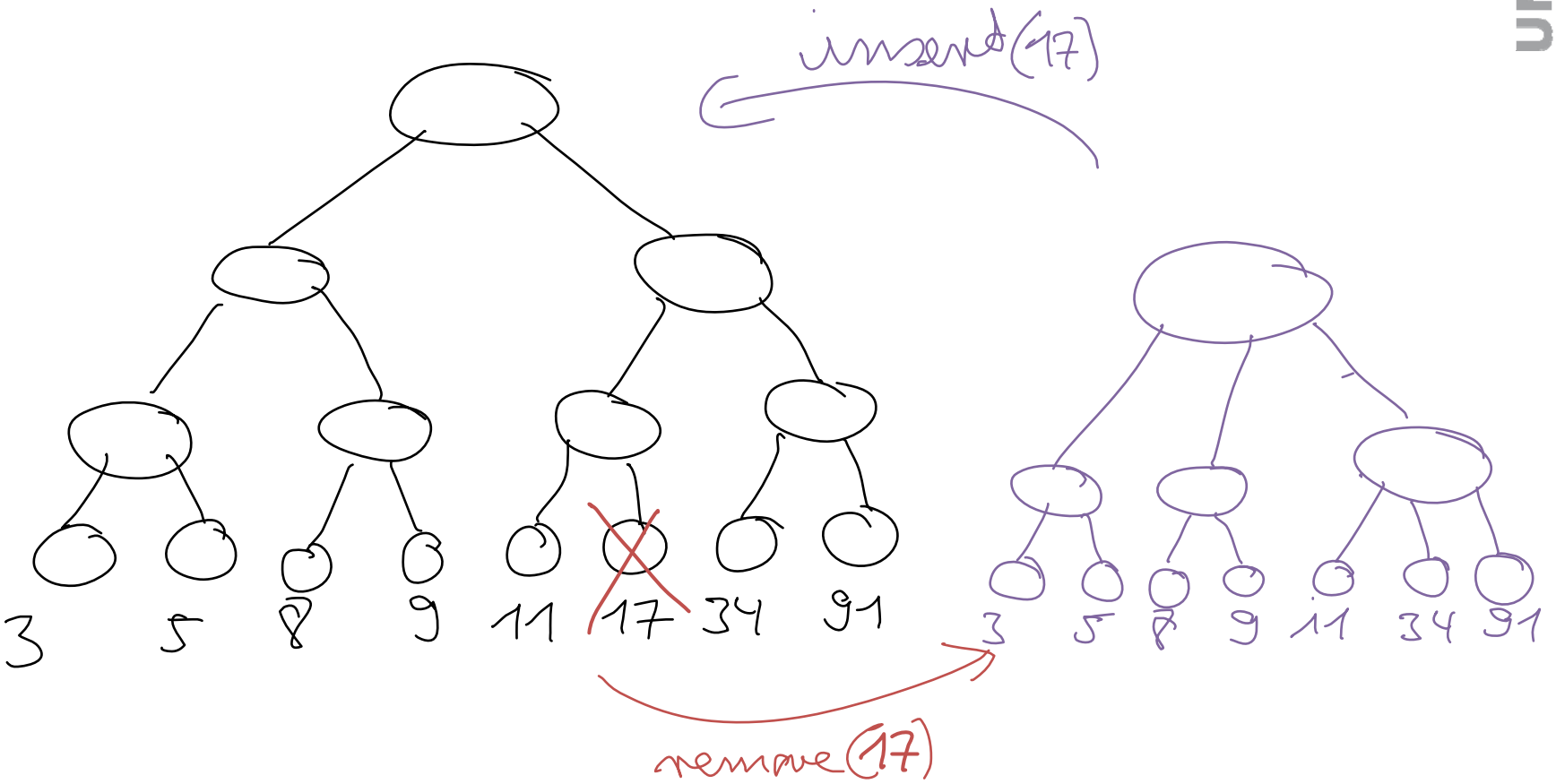
$\Rightarrow d - 1 \leq \log_a n$

$d \leq \log_a n + 1$

- Kosten von lookup, insert, remove ...
  - ... sind alle $O(b \cdot \text{depth})$, wobei depth = Baumtiefe
  - Weil jeder Knoten, außer evtl. der Wurzel, mindestens a Kinder hat, ist $\text{depth} = O(\log_a n)$
  - Bei genauerem Hinsehen fällt auf
    - Die Operation lookup braucht immer Zeit ~ depth
    - Aber insert und remove scheinen oft in $O(1)$ zu gehen

      (nur im schlechtesten Fall müssen alle Knoten auf dem Weg zur Wurzel geteilt / verschmolzen werden)
  - Das wollen wir jetzt genauer analysieren
  - Dafür reicht $b \geq 2a - 1$ allerdings nicht, wir brauchen $b \geq 2a$
  - Auf der nächsten Folie ein Gegenbeispiel für $a = 2$ und $b = 3$
  - Dann die Analyse für $a = 2$ und $b = 4$

(2,3)-Bäume — Gegenbeispiel



Man kann zeigen:
Wenn $b = 2a - 1$, dann gibt es eine Folge von n Operationen mit Kosten $\Omega(n \cdot \log n)$.

(2,4)-Bäume — Analyse

UNI FREIBURG

- Intuition
  - Wenn alle Knoten im Baum **2** Kinder haben, müssen wir nach einem remove alle Knoten bis zur Wurzel verschmelzen
  - Wenn alle Knoten im Baum **4** Kinder haben, müssen wir nach einem insert alle Knoten bis zur Wurzel aufspalten
  - Wenn alle Knoten im Baum **3** Kinder haben, dauert es lange bis wir in eine dieser beiden Situationen kommen
  - Idee für die Analyse: wir müssen formalisieren, dass der Baum nach einer teuren Operation in einem Zustand ist, so dass es lange dauert, bis wieder eine Operation teuer wird
    - das ist ähnlich wie bei den dynamischen Feldern: Reallokation ist teuer, aber danach dauert es, bis wieder realloziert werden muss, und wenn man es richtig macht, sind die Kosten im Durchschnitt konstant

(2,4)-Bäume — Analyse 1/3

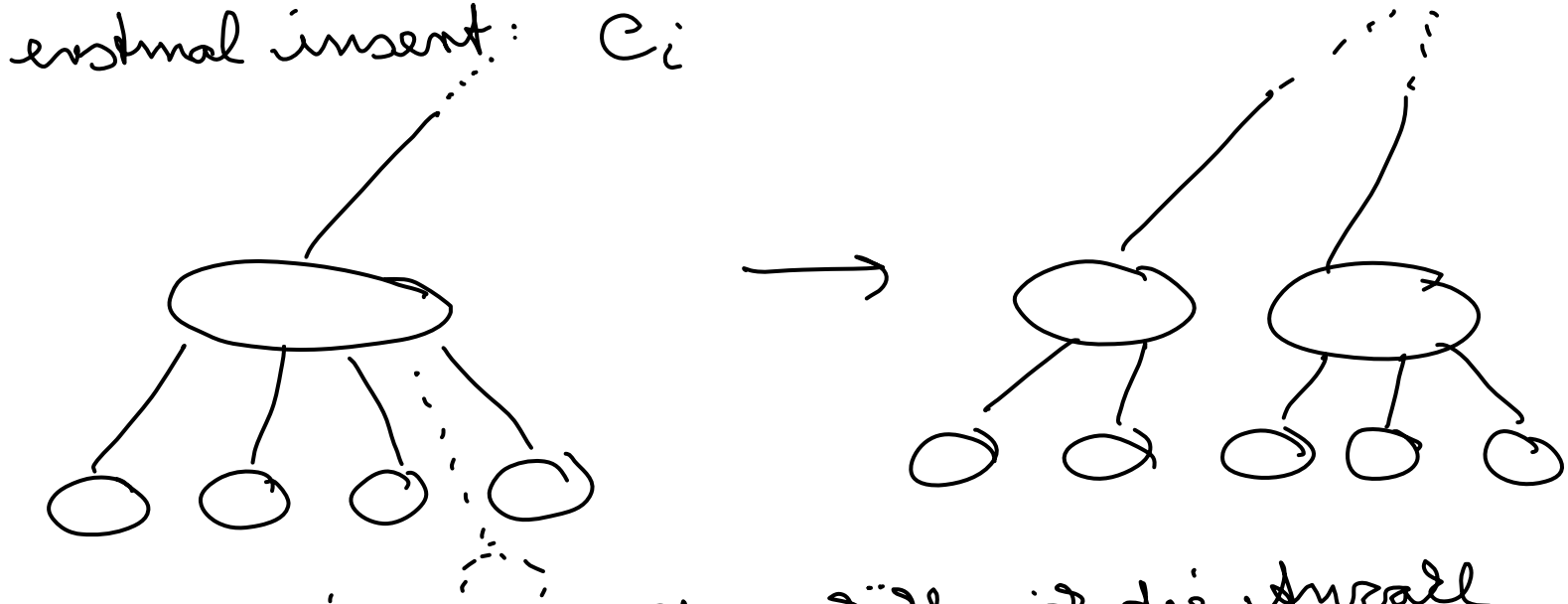
- Terminologie
  - Wir betrachten eine Folge von n Operationen
  - Seien c_i die Kosten = Laufzeit der i-ten Operation
  - Sei Φ_i das Potential des Baumes nach der i-ten Operation
    = die Anzahl der Knoten mit Grad genau 3
    - Φ_0 = Potential am Anfang = 0

Beachte: $\Phi_n \leq n$

- Lemma
  - Es gilt $c_i \leq A \cdot (\Phi_i - \Phi_{i-1}) + B$ für irgendwelche $A, B > 0$
  - Daraus folgt dann $\Sigma c_i = O(n)$

$$\sum_{i=1}^{n} c_i = c_1 + c_2 + \cdots + c_n$$
$$\leq A \cdot (\Phi_1 - \Phi_0) + A \cdot (\Phi_2 - \Phi_1) + \cdots + A(\Phi_n - \Phi_{n-1}) + B \cdot n$$
$$= A \cdot (\Phi_n - \Phi_0) + B \cdot n = A \cdot \Phi_n + B \cdot n = O(n)$$

(2,4)-Bäume — Analyse 2/3



Also: Für jedes Aufspalten erhöht sich die Anzahl der Knoten vom Grad 3 um eins.

Sei s_i = # Aufspalte-Operationen

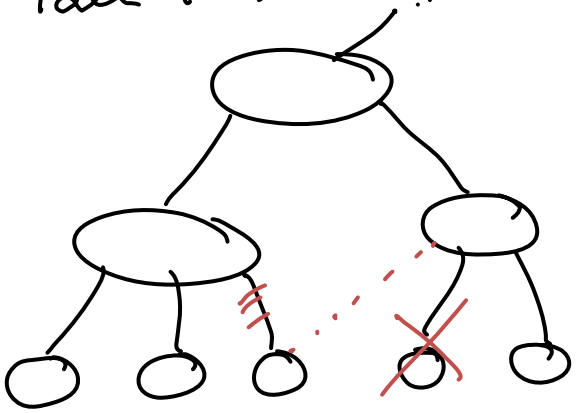
$\Rightarrow \Phi_i = \Phi_{i-1} + s_i$; $c_i \leq A \cdot s_i + B$

$= A \cdot (\Phi_i - \Phi_{i-1}) + B$

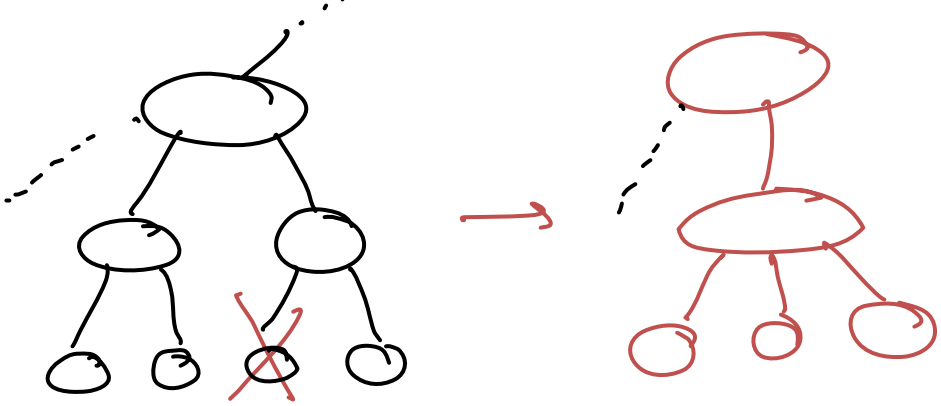
(2,4)-Bäume — Analyse 3/3

c_i = remove.

Fall 1: Klauen



Fall 2: verschmelzen



$\Rightarrow \Phi_i \geq \Phi_{i-1} - 1$

Sei m die Anzahl der Verschmelzungen

$\Rightarrow \Phi_i \geq \Phi_{i-1} + m - 1$; $c_i \leq A \cdot m + B$

$\Rightarrow c_i \leq A \cdot (\Phi_i - \Phi_{i-1}) + A + B$

Literatur / Links

- (a,b)-Bäume
  - In Mehlhorn/Sanders:

    7 Sorted Sequences [Kapitel 7.2 und 7.4]
  - In Cormen/Leiserson/Rivest

    14 Red-Black Trees [die sind ähnlich, aber anders]
  - In Wikipedia

    http://cs.wikipedia.org/wiki/(a,b)-strom (Tschechisch)

    http://en.wikipedia.org/wiki/(a,b)-tree