

Algorithmen & Datenstrukturen

Herbst 2022

Vorlesung 8

Datenstrukturen, Wörterbuch, AVL-Bäume

Vorlesung gestartet:	Algorithmus	Datensstrukturen
Entwurf	✓	heute
Analyse	✓	heute

## Datensstrukturen für abstrakte Datentypen (ADTs)

ADT: Objekte  
Operationen

Beispiel: Objekte = Schlüssel  $\in \mathbb{N}$

Beispiel: Studenten definieren  
Schlüssel = Matr. Nummer

Datenstruktur =

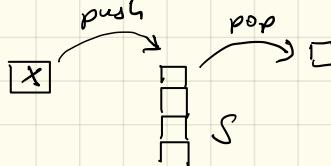
Implementierung eines ADTs

Ziel: Effizienz

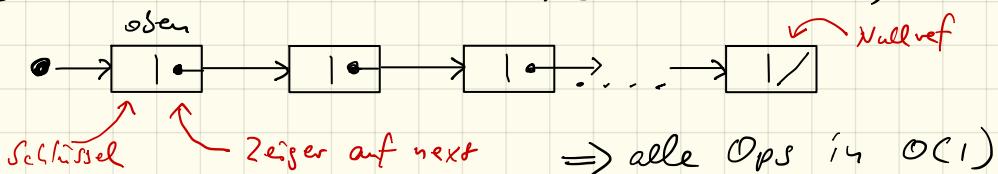
### 1. ADT Stapel (Stack)

push ( $x, S$ ) legt  $x$  auf Stapel  $S$   
 pop ( $S$ ) entfernt (und liefert) oberstes Element  
 top ( $S$ ) liefert oberstes Element  
 (isempty ( $S$ ), emptystack  $\rightarrow$  leerer Stapel)

Visualisierung:



Datenstruktur: verdeckte Liste (linked list)



Binary geht auch, aber man muss max. lange Kette

## 2. ADT Schlange (Queue)

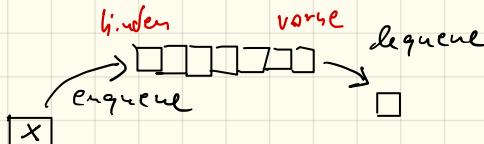
enqueue( $x, S$ )

füge  $x$  hinter an

dequeue( $S$ )

entferne (und liefern) vorletztes Element

Visualisierung:



Datenstruktur: z.B. doubly linked list  
+ Zeiger auf letztes Element  
 $\Rightarrow$  beide Ops in  $O(1)$

## 3. ADT Prioritätsschlange (Priority Queue)

insert( $x, P$ )

füge  $x$  ein

extract-max( $P$ )

lösle (und liefe) Maximum

Datenstruktur: Heap

Beispiel: heap sort  
 $\Rightarrow$  beide Ops in  $O(\log n)$

## 4. ADT Wörterbuch (Dictionary)

search( $x, w$ )

ist  $x$  in  $w$ ?

insert( $x, w$ )

füge  $x$  in  $w$  ein (wenn es schon da ist)

remove( $x, w$ )

entferne  $x$  aus  $w$

Datenstruktur?

## Sortiertes Array

[---|7|12|17|81|---]

Suche:  $O(\log n)$ , Einfügen/entfernen:  $O(n)$

## Unsortiertes Array

Alle Ops  $O(n)$

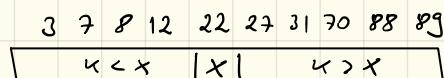
Linear List Suche/Einfügen  $O(n)$  (egal ob sortiert oder unsortiert)

Heaps Suche ist  $O(n)$

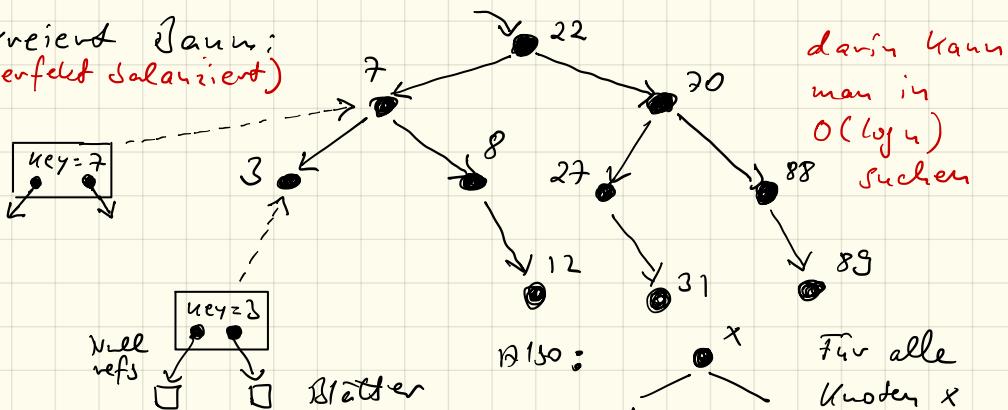
Ziel: alle Ops in  $O(\log n)$

Idee: verwende Baum

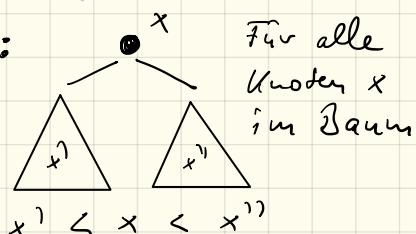
## Suchbaum

Binäre Suche: 

Kreisend Baum;  
(perfekt balanciert)



also:



"Suchbaum definierung"

Suche ( $x, p$ ) (search)

if  $p = \text{null}$ : Misserfolg

else if  $p.\text{key} = x$ : Erfolg

else

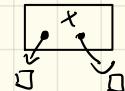
if  $x < p.\text{key}$ : Suche ( $x, p.\text{left}$ )

else: Suche ( $x, p.\text{right}$ )

p:	
key	
left	right

Einfügen ( $x, p$ ): Suche ( $x, p$ )  
(invers)

Erscheint nicht durch



Suche Ops in  $O(h)$ ,  $h = \text{Höhe}(\text{Bau})$

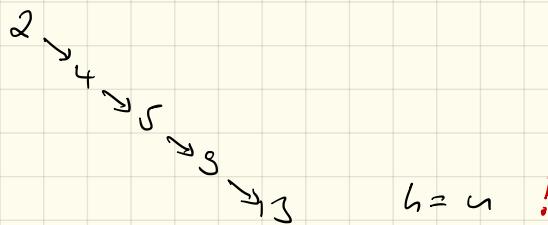
$\log_2(n) \leq h \leq n$  ( $n = \text{Anzahl Schlüssel}$ )

Problem: Baum kann sehr unbalanciert werden

Beispiel: Eingabe einer sordierten Reihe

2, 4, 5, 3, 18, ...

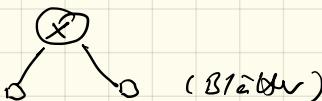
ergebt



Wie halten wir  $h$  klein?  $\rightarrow$  später

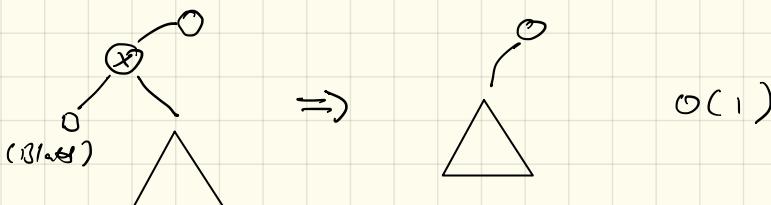
Entferne ( $x, p$ ): zuerst Suche ( $x, p$ ) in  $O(4)$   
 (remove)

Fall 1:

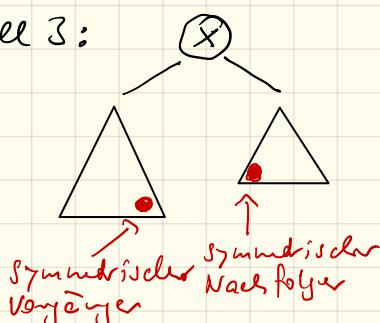


$\Rightarrow$  einfach löschen  $O(1)$

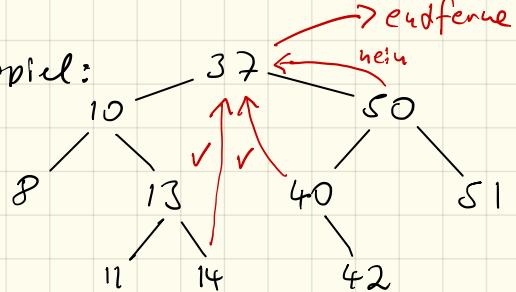
Fall 2:



Fall 3:



Beispiel:



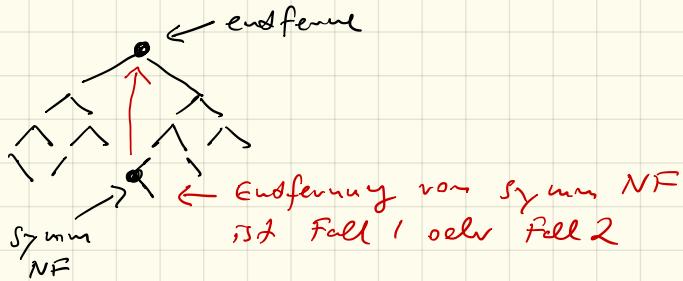
womit ersetzen wir 37 um die  
 Suchbaumseigenschaft zu erhalten?

Symmetrischer Nachfolgr.; nächst größeres Element  
 im Baum

$\Rightarrow$  Suche  $x$  und symm. NF

(symm NF: gehe von  $x$  einstell nach rechts  
 und dann immer nach links  
 bis zu einem Blatt) Blatt = Nullref

Visuell:



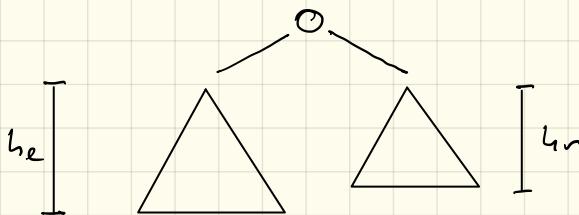
$\Rightarrow$  Endfeme ist  $O(h)$ , also alle Ops  $O(h)$

Verbleibendes Problem:  $h = \Theta(\log n)$  erhalten

Idee 1: erreiche perfekte Balancierung  
ist schwierig



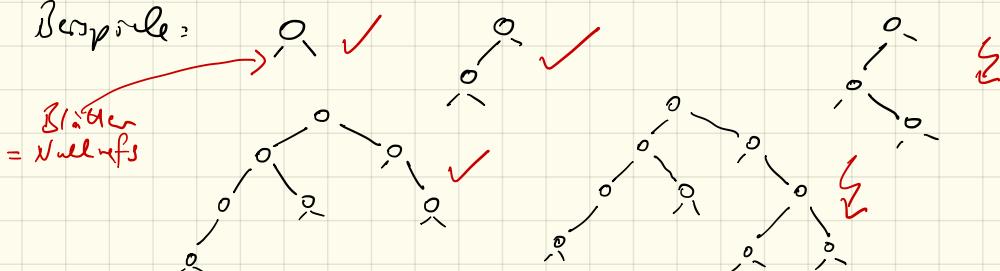
Idee 2: relaxiere und verlange nur die  
folgende Strukturbedingung



AVL-Bäume  
(Adel'son-Velsky,  
Landis)

$|h_e - h_r| \leq 1$  für alle Knoten im Baum

Beispiele:



- 1.) Wie ist die Höhe  $h$ ? (näher an  $\log_2(n)$  oder  $n^2$ ?)  
 2.) Wie erreicht man die AVL-Balanzierung?

z- 1.) Idee: Bestimmen eine untere Schranke für die Anzahl der Blätter und verwenden:

Satz: Ein Binärbaum der  $n$  Schlüssel (innere Knoten) speichert höchstens  $n+1$  Blätter. (Beweis: Induktion  $\rightarrow$ )

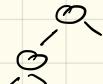
Also:  $MIS(h) = \text{Mindestanzahl eines AVL-Baums der Höhe } h$  ( $\Rightarrow n \geq MIS(h)-1$ )  
 ↑ Anzahl Knoten

$$MIS(1) = 2$$



$$= MIS(3)$$

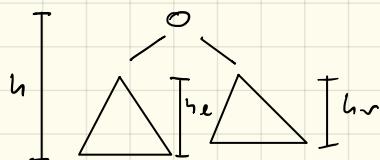
$$MIS(2) = 3$$



$$= MIS(4)$$

$$MIS(h) = MIS(h-1) + MIS(h-2)$$

$$= MIS(h+2)$$



$h_L, h_R$ : eine  $= h-1$   
 andere  $\geq h-2$

$$\Rightarrow MIS(h) = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^h\right), \text{ d.h. } n \geq \sum \left(\left(\frac{1+\sqrt{5}}{2}\right)^h\right)$$

genauer:  $n \geq 1.6 \dots^h$

$$\approx 1/\log_2((1+\sqrt{5})/2)$$

$$\Rightarrow h \leq 1.44 \log_2(n)$$

also Höchstanzahl 44% höher als perfekt balancierte

$$\Rightarrow O(h) = O(\log n)$$

Also: einfügen und löschen:

1.) einfügen

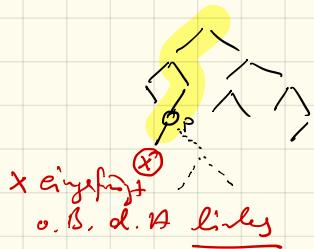
2.) evtl. rebalancieren (AVL Bedingung wiederherstellen)

dazu genügt es alle Voraussetzungen des eingefüllten Elements anzusehen ( $O(h)$  viele)

Beispiel:



Rebalancieren:



ges.: muss schlimmstenfalls  
geknotet + rebalanciert werden

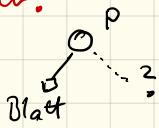
wenn  $O(1)$  pro Knoten  
 $\Rightarrow O(h)$

Wir merken uns in jedem Knoten  $p$   
 $\text{Sal}(p) = h_r - h_e$

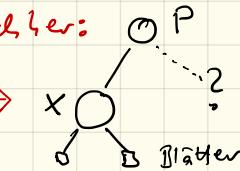
- 1: linker Teilbaum höher
- 0: beide gleich hoch
- 1: rechter Teilbaum höher

Einfügen von  $x$  (Annahme: links, rechts ist analog):

vorher:



nachher:



$p$  steht  
irgendwo  
im Baum

Fälle:  
*(vorher)* 1.)  $\text{Sal}(p) = -1$

nicht möglich

2.)  $\text{Sal}(p) = 0$   
 $\text{Sal}(x) = 0$

d.h. vorher



Höhe Teilbaum

$\text{Sal}(p) = -1$

$p$  ist gewachsen!  $\text{upin}(p)$

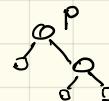
( $p$  after insertion)  
nötig für:

implementieren  
wir später

- updates von Sal
- in Vorgängern
- evtl. Rebalanzierung

3.)  $\text{Sal}(p) = +1$

d.h. vorher



Höhe TB  $p$

$\text{Sal}(p) = 0$

nicht gewachsen

fertig

In allen 3 Fällen  
ist TB  $p$  AVL!

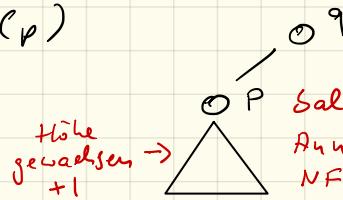
Der Aufruf  $\text{upin}(p)$  gilt invariante

- $\text{Sal}(p) \neq 0$
- Höhe TB  $p$  ist gewachsen
- $p$  hat Vorgänger (sonst ist Aufruf unnötig)

Bild dazu: nächste Seite

Beschreibung  $\text{upin}(p)$

Situation:



$$\text{Sal}(p) \neq 0$$

Annahme:  $p$  ist Linker  
NF von  $q$ . Rechter NF  
geht analog.

Fälle: 1.)  $\text{Sal}(q) = +1$

$$\text{Sal}(q) = 0$$

TB  $q$  ist AVL

festig

2.)  $\text{Sal}(q) = 0$

höhe TB  $q$

gewachsen

$$\text{Sal}(q) = -1$$

$\text{upin}(q)$

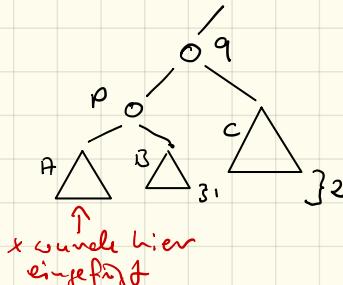
TB  $q$  ist AVL

3.)  $\text{Sal}(q) = -1$

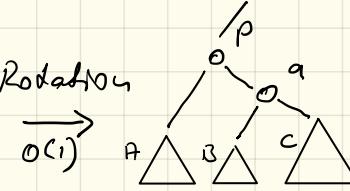
$$\text{Sal}(p) = -1 \text{ oder } +1$$

TB  $q$  ist nicht mehr AVL  
→ Anseit nötig

3a.)  $\text{Sal}(p) = -1$



Rotation  
 $OC1 \rightarrow$



$\text{Sal}(p) = 0$   
 $\text{Sal}(q) = 0$   
festig

TB  $p$  hat gleiche  
höhe wie TB  $q$  von einfügen  
⇒  $\text{upin}$  nicht nötig

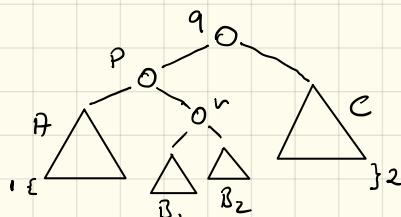
A: füg 1 nach oben

B: gleich

C: 1 nach unten

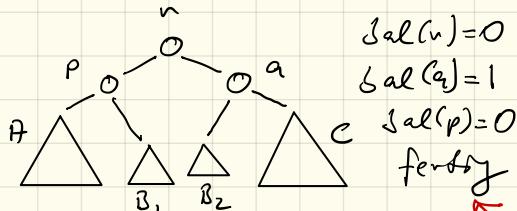
Suchbaumbedingung  
erfüllt!

$$35.) \text{ bal}(p) = +1$$



$x$  wurde in  $B_1$ ,  
oder  $B_2$  eingefügt  
(hier:  $B_1$ )

Doppel-  
notation  
 $\xrightarrow{\quad}$   
 $O(1)$



$\text{bal}(n)=0$   
 $\text{bal}(q)=1$   
 $\text{bal}(p)=0$   
fehl

TB r hat gleiche Höhe wie  
TB q vor einfügen  
 $\Rightarrow$  upsh nicht nötig

A steht in Höhe  
B1 geht 1 nach oben  
B2 geht 1 nach oben  
C geht 1 nach unten

Suchbaumbedingung  
erfüllt!

also: Einfügen ist  $O(\log n)$

Entfernen in AVL Bäumen: geht ähnlich in  $O(\log n)$