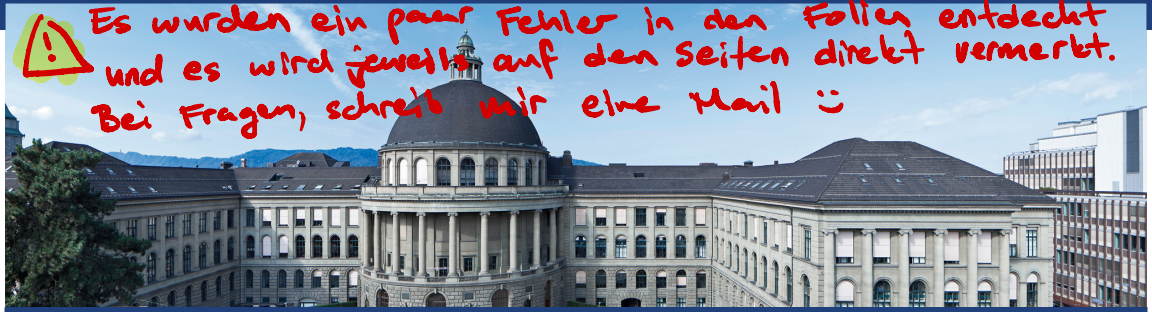
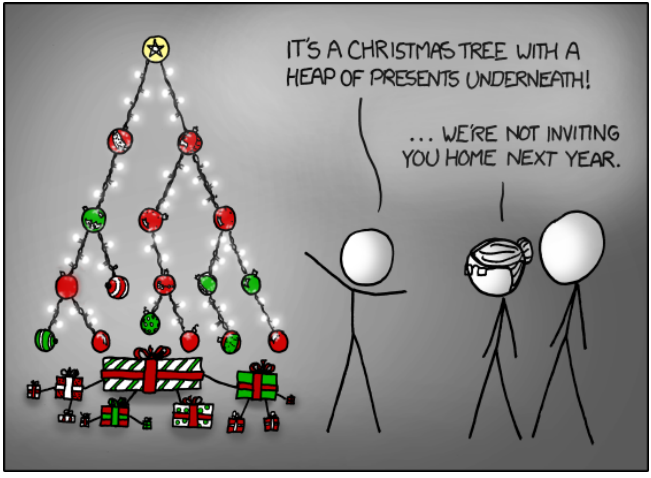
ETH zürich



D&A - Übungsstunde 6

Diese Folien basieren auf denjenigen der Vorlesung, wurden aber durch den Assistenten Adel Gavranović adaptiert und erweitert

Comic der Woche



xkcd

Übersicht

Heutiges Programm

Intro
Follow-up
Feedback zu [code]expert
Master Method Recap
Wiederholung Theorie
- Binäre Bäume und Heaps
- Binäre Bäume
- 2-3-Bäume
- Rot-Schwarz-Bäume

Tipps zu [code]expert
Coding-Aufgabe
Outro

n.ethz.ch/~agavranovic

- Link zum Material für die Übungsstunden
- Webseite des Assistenten
- Mail an Assistenten

1. Intro

Intro

Intro

- Keine Übungsstunde nächste Woche!

2. Follow-up

Follow-up aus vorherigen Übungsstunden

Kann man aus einem instabilen Sortieralgorithmus immer einen stabilen machen, indem man einfach die {grösser, kleiner}-gleich durch strikt-{grösser, kleiner} ersetzt?

Follow-up aus vorherigen Übungsstunden

Kann man aus einem instabilen Sortieralgorithmus immer einen stabilen machen, indem man einfach die {grösser, kleiner}-gleich durch strikt-{grösser, kleiner} ersetzt?

- Alleine schon bei `QuickSort` lassen sich Arrays konstruiren, bei dem das nicht funktionieren würde; daher **nein**

Follow-up aus vorherigen Übungsstunden

Wie geht man bei der Landau-Notation damit um, wenn die Anzahl `std::swaps` *einfach* 0 *ist?*

Follow-up aus vorherigen Übungsstunden

Wie geht man bei der Landau-Notation damit um, wenn die Anzahl `std::swaps` *einfach* 0 *ist?*

- Einfach so wie bei der Musterlösung: 0 hinschreiben

Follow-up aus vorherigen Übungsstunden

Wie geht man bei der Landau-Notation damit um, wenn die Anzahl `std::swaps` *einfach* 0 *ist?*

- Einfach so wie bei der Musterlösung: 0 hinschreiben
- Ausserdem: per Definition d. Landau-Notation $C \neq 0$

3. Feedback zu [code]expert

Allgemeines zu [code]expert

Allgemeines zu [code]expert

- ChatGPT wird an der Prüfung nicht zur Verfügung stehen...

Task "The Master Method"

Task "The Master Method"

- Wiederholung zwingend nötig

Task "Comparing Sorting Algorithms"

Task "Comparing Sorting Algorithms"

- wenn Code gegeben ist, dann ist dieser gemeint, d.h. nicht das "theoretische Ideal" eines Algorithmus, sondern *genau dieser* Code

Task "Amortized Analysis: Dynamic Array"

Task "Amortized Analysis: Dynamic Array"

- Es bringt euch nicht viel, einfach die Musterlösung grob zu reformulieren und abzugeben

Task "Matrices"

Task "Matrices"

- Schwierig...

Task "Matrices"

- Schwierig...
- die meisten haben die Theoriefrage nicht beantwortet

Fragen zu [code]expert eurerseits?

Master Methode

$$\frac{2n}{3} = \frac{n}{b} \Leftrightarrow b = \frac{3}{2}$$

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \quad (a \geq 1, b > 1)$$

a : Anzahl Unterprobleme
$1/b$: Aufteilungsquotient[1]
$f(n)$: Divisions- und Summierungskosten

[1] Teil des Originalproblems, welches wiederum durch Unterprobleme repräsentiert wird

"Meine" Master Methode

1. Forme Rekursionsgleichung ins Schema um (wenn möglich)
2. Berechne $K := \log_b a$
3. Fallunterscheidung machen ($\varepsilon > 0$):

$$f \in \begin{cases} \mathcal{O}\left(n^{K-\varepsilon}\right) \\ \Theta\left(n^{K}\right) \\ \Omega\left(n^{K+\varepsilon}\right) \wedge \left(af\left(\frac{n}{b}\right) \leq cf(n),\ 0 < c < 1\right) \end{cases} \begin{array}{l} \implies T(n) \in \Theta\left(n^K\right) \\ \implies T(n) \in \Theta\left(n^K \log(n)\right) \\ \implies T(n) \in \Theta(f(n)) \end{array}$$

4. Richtig aufschreiben

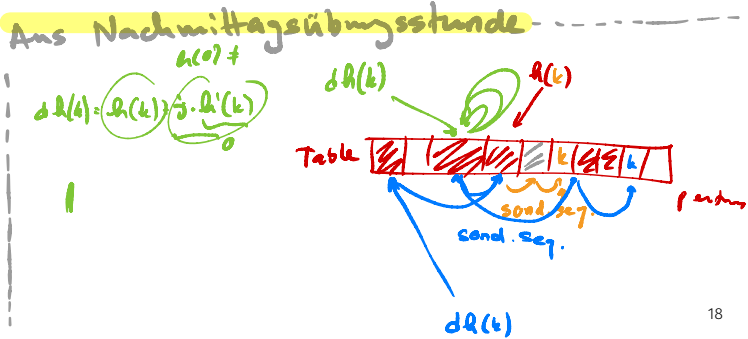
Fragen/Unklarheiten?

Nachbesprechung

$dh: \; h(k) + j \cdot h'(k) \qquad j \in \{0, 1, \ldots\}$

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q$



Nachbesprechung

Open hashing:

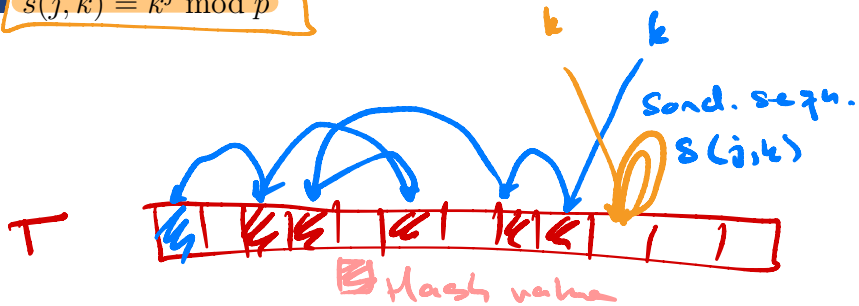
- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ nicht passend: $(k = 0) \mapsto 0$

Nachbesprechung

$H(j,k) = h(k) + j \cdot h'(k)$

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ nicht passend: $(k = 0) \mapsto 0$
- $s(j,k) = k^j \bmod p$



Nachbesprechung

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ nicht passend: $(k = 0) \mapsto 0$
- $s(j,k) = k^j \bmod p \rightarrow$ nicht passend: $(k = 0) \mapsto 0$, $(k = 1) \mapsto 1$

Nachbesprechung

Open hashing:

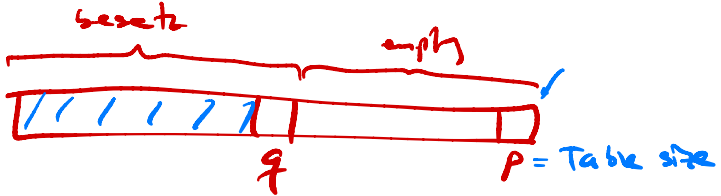
- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ nicht passend: $(k = 0) \mapsto 0$
- $s(j,k) = k^j \bmod p \rightarrow$ nicht passend: $(k = 0) \mapsto 0, (k = 1) \mapsto 1$
- $s(j,k) = ((k \cdot j) \bmod q) + 1$

Nachbesprechung

↳ mehr Info zur Notation gibt's in der nächsten Übungsstunde

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ nicht passend: $(k = 0) \mapsto 0$
- $s(j,k) = k^j \bmod p \rightarrow$ nicht passend: $(k = 0) \mapsto 0, (k = 1) \mapsto 1$
- $s(j,k) = ((k \cdot j) \bmod q) + 1 \rightarrow$ nicht passend: 1 wenn k Vielfaches von q, und Bereich $p - q$ nicht abgedeckt.



Nachbesprechung

Cuckoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 27, 2, 32

	0	1	2	3	4
T_1:	__,	__,	27,	__,	__

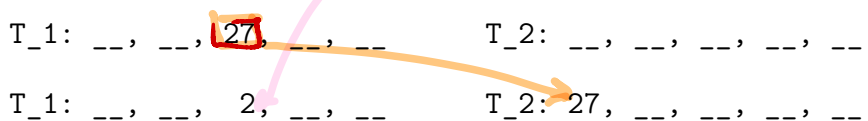
T_2:	__,	__,	__,	__,	__

Nachbesprechung

$h_1(2) = 2$

Cuckoo hashing

- $h_1(k) = k \bmod 5, h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 27, 2, 32



Nachbesprechung

Cuckoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 27, 2, 32

```
T_1: __, __, 27, __, __      T_2: __, __, __, __, __

T_1: __, __,  2, __, __      T_2: 27, __, __, __, __

T_1: __, __, 27, __, __      T_2:  2, 32, __, __, __
```

nach insert (32)

Nachbesprechung

Coocoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 7: Endlosschleife

Nachbesprechung

Coocoo hashing

$Q_n(7) = 2$

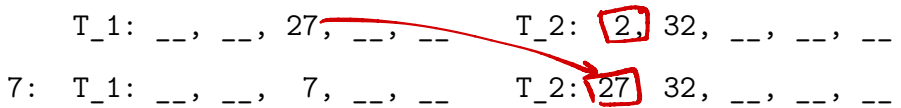
- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 7: Endlosschleife

```
T_1: __, __, 27, __, __     T_2:  2, 32, __, __, __
```

Nachbesprechung

Coocoo hashing

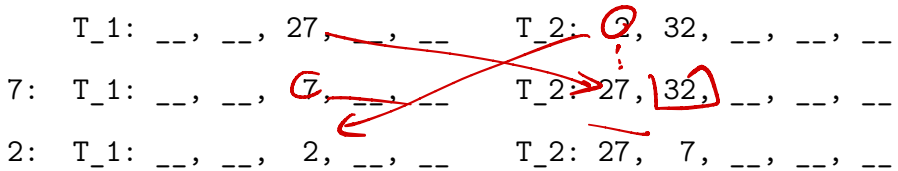
- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 7: Endlosschleife



Nachbesprechung

Coocoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 7: Endlosschleife



Nachbesprechung

Coocoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 7: Endlosschleife

```
     T_1: __, __, 27, __, __    T_2:  2, 32, __, __, __
 7:  T_1: __, __,  7, __, __    T_2: 27, 32, __, __, __
 2:  T_1: __, __,  2, __, __    T_2: 27,  7, __, __, __
32:  T_1: __, __, 32, __, __    T_2:  2,  7, __, __, __
```

Nachbesprechung

Coocoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 7: Endlosschleife

```
     T_1: __, __, 27, __, __    T_2:  2, 32, __, __, __
 7:  T_1: __, __,  7, __, __    T_2: 27, 32, __, __, __
 2:  T_1: __, __,  2, __, __    T_2: 27,  7, __, __, __
32:  T_1: __, __, 32, __, __    T_2:  2,  7, __, __, __
27:  T_1: __, __, 27, __, __    T_2:  2, 32, __, __, __
```

Nachbesprechung

Coocoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 7: Endlosschleife

```
     T_1: __, __, 27, __, __    T_2:  2, 32, __, __, __
 7:  T_1: __, __,  7, __, __    T_2: 27, 32, __, __, __
 2:  T_1: __, __,  2, __, __    T_2: 27,  7, __, __, __
32:  T_1: __, __, 32, __, __    T_2:  2,  7, __, __, __
27:  T_1: __, __, 27, __, __    T_2:  2, 32, __, __, __
 7:  ...
```

Fragen/Unklarheiten?

Nachbesprechung

Finden eines Sub-Arrays

```
// calculating hash_a, hash_b, c_to_k
It1 window_end = from;
for(It2 current = begin; current != end;
    ++current, ++window_end) {
  if(window_end == to) return to;
  hash_b = (C * hash_b % M + *current) % M;
  hash_a = (C * hash_a % M + *window_end) % M;
  c_to_k = c_to_k * C % M;
}
```

Nachbesprechung

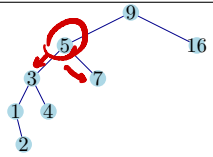
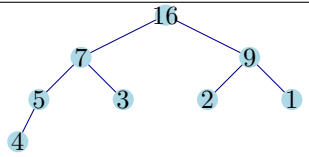
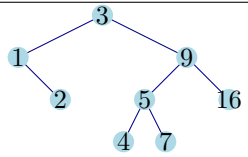
Finden eines Sub-Arrays

```
// looking for b and updating hash_a
for(It1 window_begin = from; ;
    ++window_begin, ++window_end) {
  if(hash_a == hash_b)
    if(std::equal(window_begin, window_end, begin, end))
      return window_begin;
  if(window_end == to) return to;
  hash_a = (C * hash_a % M + *window_end
          + (M - c_to_k) * *window_begin % M) % M;
}
```

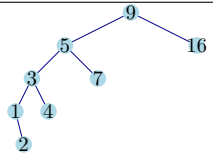
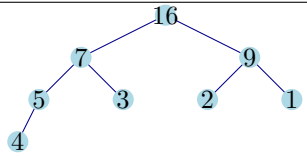
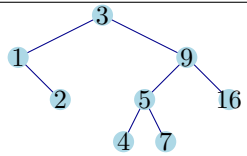
Fragen/Unklarheiten?

5. Wiederholung Theorie

Vergleich binärer Bäume

	Suchbäume	**Heaps** Min- / Max- Heap	**Balancierte Bäume** AVL, red-black tree
in C++:		`std::make_heap`	`std::map`
			
Einfügen	$\Theta(h(T))$	$\Theta(\log n)$	$\Theta(\log n)$
Suchen	$\Theta(h(T))$	$\Theta(n)$ (!!)	$\Theta(\log n)$
Löschen	$\Theta(h(T))$	Suchen + $\Theta(\log n)$	$\Theta(\log n)$
Min/Max	$\Theta(h(T))$	$\Theta(1)$/Suchen	$\Theta(\log n)$

Vergleich binärer Bäume

	Suchbäume	**Heaps** Min- / Max- Heap	**Balancierte Bäume** AVL, red-black tree
in C++:		`std::make_heap`	`std::map`
			
Einfügen	$\Theta(h(T))$	$\Theta(\log n)$	$\Theta(\log n)$
Suchen	$\Theta(h(T))$	$\Theta(n)$ (!!)	$\Theta(\log n)$
Löschen	$\Theta(h(T))$	Suchen + $\Theta(\log n)$	$\Theta(\log n)$
Min/Max	$\Theta(h(T))$	$\Theta(1)$/Suchen	$\Theta(\log n)$

Bemerke: $\Theta(\log n) \leq \Theta(h(T)) \leq \Theta(n)$

Wiederholung: Binäre Bäume, Schlüssel Einfügen

Binäre Suchbäume

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.

MinHeap

- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

Wiederholung: Binäre Bäume, Schlüssel Einfügen

Binäre Suchbäume

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.

MinHeap

- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

kein typo!

Aufgabe: Einfügen von $4, 8, 16, 1, 6, 7$ in leeren Baum/Heap.

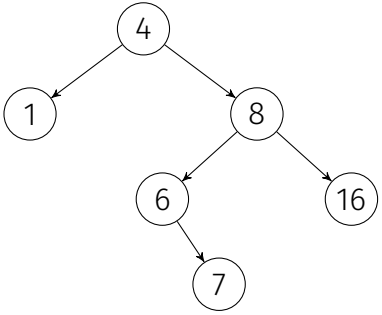
→ übungsaufgabe

≙ gratis Punkte an der Prüfung :)

Wiederholung: Binäre Bäume, Schlüssel Einfügen

Binäre Suchbäume

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.



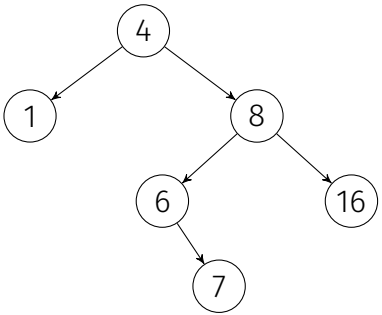
MinHeap

- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

Wiederholung: Binäre Bäume, Schlüssel Einfügen

Binäre Suchbäume

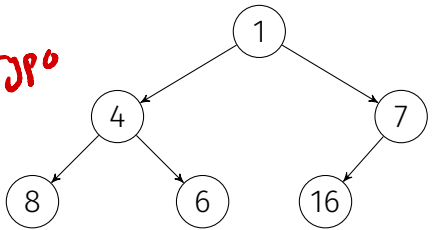
- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.



MinHeap

- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

kein typo



Wiederholung: Binäre Bäume, Schlüssel Löschen

Binäre Suchbäume

- Schlüssel k durch symm. Nachfolger n ersetzen.
- Achtung: Wohin mit rechtem Kind von n?

MinHeap

- Schlüssel durch hinterstes Arrayelement ersetzen.
- Heap-Bedingung wiederherstellen: `siftDown` *or* `siftUp`.

Wiederholung: Binäre Bäume, Schlüssel Löschen

Binäre Suchbäume

- Schlüssel k durch symm. Nachfolger n ersetzen.
- Achtung: Wohin mit rechtem Kind von n?

MinHeap

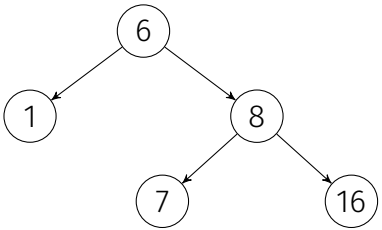
- Schlüssel durch hinterstes Arrayelement ersetzen.
- Heap-Bedingung wiederherstellen: `siftDown` *or* `siftUp`.

Aufgabe: Löschen von 4 in Beispiel-Baum/Heap.

Wiederholung: Binäre Bäume, Schlüssel Löschen

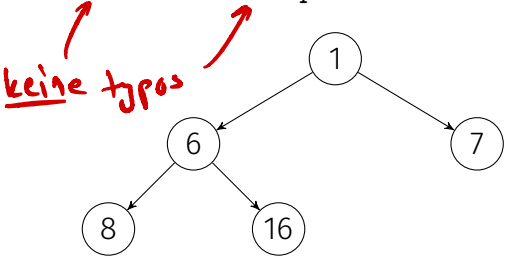
Binäre Suchbäume

- Schlüssel k durch symm. Nachfolger n ersetzen.
- Achtung: Wohin mit rechtem Kind von n?



MinHeap

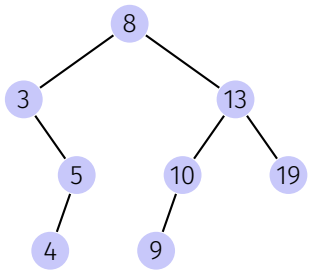
- Schlüssel durch hinterstes Arrayelement ersetzen.
- Heap-Bedingung wiederherstellen: `siftDown` *or* `siftUp`.



Fragen/Unklarheiten?

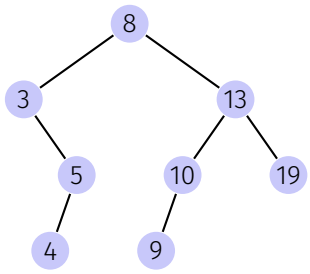
Traversierungsarten

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
- Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
- Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.



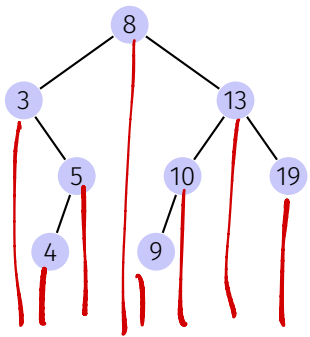
Traversierungsarten

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
- Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.



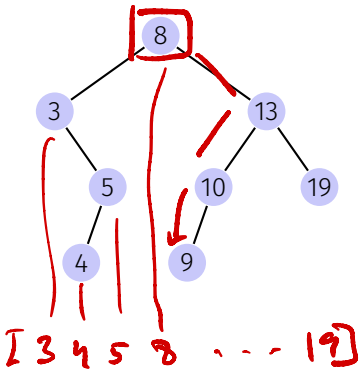
Traversierungsarten

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
  4, 5, 3, 9, 10, 19, 13, 8
- Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.



Traversierungsarten

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
  4, 5, 3, 9, 10, 19, 13, 8
- Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.
  3, 4, 5, 8, 9, 10, 13, 19



Fragen/Unklarheiten?

Quiz

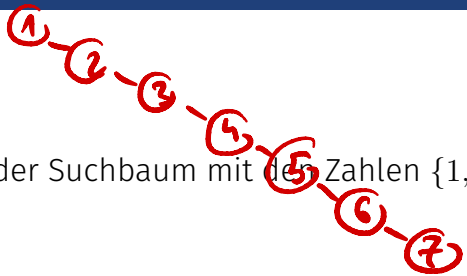
1. Zeichnen Sie jeweils einen binären Suchbaum, der die folgenden Traversierungen erzeugt. Ist der Baum eindeutig? (= unique)

Symmetrische Reihenfolge (inorder)	1 2 3 4 5 6 7 8
Hauptreihenfolge (preorder)	4 3 1 2 8 6 5 7
Nebenreihenfolge (postorder)	1 3 2 5 6 8 7 4

2. Geben Sie zu jeder Reihenfolge eine Zahlensequenz aus $\{1, \ldots, 4\}$, die nicht aus einem gültigen binären Suchbaum stammen kann.

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
  4, 5, 3, 9, 10, 19, 13, 8
- Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.
  3, 4, 5, 8, 9, 10, 13, 19

Antworten



Symmetrische Reihenfolge: jeder Suchbaum mit den Zahlen $\{1, \ldots, 8\}$ ist gültig.

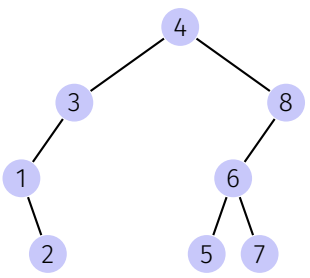
- Der Baum ist nicht eindeutig
- Es gibt keinen Suchbaum, welcher nicht die aufsteigend sortierte Sequenz ausgeben würde. Gegenbeispiel 1 2 4 3

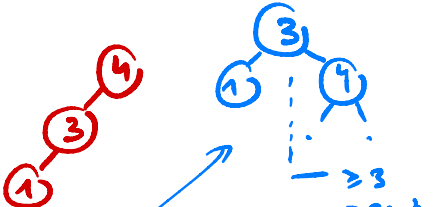
Antworten

? sind alle Traversierungssequenzen für BST unique?

↳ ! Ja, aus validen Haupt- (und Neben-)reihenfolgen, lassen sich die traversierten binären Suchbäume uniquely rekonstruieren.

Hauptreihenfolge (preorder) 4 3 1 2 8 6 5 7



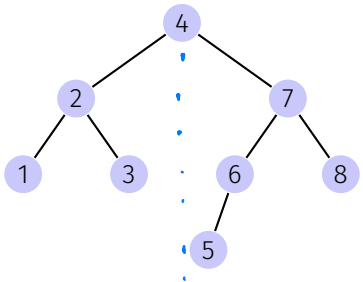


Der Baum ist eindeutig

Es muss rekursiv gelten, dass zuerst eine Gruppe Zahlen grösser und danach kleiner als der erste Wert kommen. Gegenbeispiel: 3 1 4 2

Antworten

Nebenreihenfolge (postorder) 1 3 2 5 6 8 7 4



Der Baum ist eindeutig

Konstruktion hier: `https://www.techiedelight.com/build-binary-search-tree-from-postorder-sequence/`, Ähnliches Argument wie vorher, nur von hinten nach vorne. Gegenbeispiel 4 2 1 3

Quiz

Wahr oder falsch:

1. Die Preorder-Reihenfolge ist die umgekehrte Postorder-Reihenfolge
2. Der erste Knoten in der Preorder ist immer die Wurzel.
3. Der erste Knoten in der Inorder ist nie die Wurzel.
4. Wenn die Knoten in Reihenfolge der Preorder in einen leeren Baum eingefügt werden, erhält man den gleichen Baum.
5. Wenn die Knoten in Reihenfolge der Postorder in einen leeren Baum eingefügt werden, erhält man den gleichen Baum.
6. Wenn die Knoten in Reihenfolge der Inorder in einen leeren Baum eingefügt werden, erhält man den gleichen Baum.

Quiz: Lösung

Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
8, 3, 1, 4, 13, 10, 9, 19

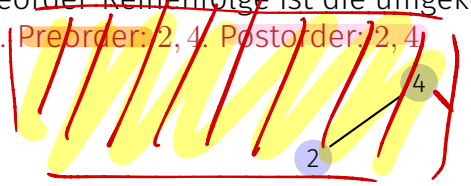
Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
4, 5, 3, 9, 10, 19, 13, 8

Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.
3, 4, 5, 8, 9, 10, 13, 19

Fehler! siehe nächste seite

Wahr oder falsch:

1. Die Preorder-Reihenfolge ist die umgekehrte Postorder-Reihenfolge
   Falsch. Preorder: 2, 4. Postorder: 2, 4.



pre: 4 2 5
post: 2 5 4

2. Der erste Knoten in der Preorder ist immer die Wurzel.
   Wahr (so definiert!)
3. Der erste Knoten in der Inorder ist nie die Wurzel.
   Falsch! Immer wenn der linke Teilbaum leer ist, ist die Wurzel der erste Knoten in inorder.

Edgecases!

Quiz: Lösung **Korrekturfolie**

Wahr oder falsch:

1. Die Preorder-Reihenfolge ist die umgekehrte Postorder-Reihenfolge
   Falsch. Preorder: $4, 2, 5$. Postorder: $2, 5, 4$.

2. Der erste Knoten in der Preorder ist immer die Wurzel.
   Wahr (so definiert!)
3. Der erste Knoten in der Inorder ist nie die Wurzel.
   Falsch! Immer wenn der linke Teilbaum leer ist, ist die Wurzel der erste Knoten in inorder.

Quiz: Lösung

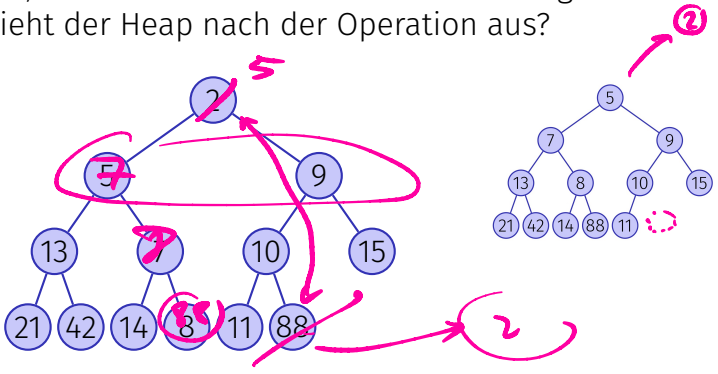
Wahr oder falsch:

4. Wenn die Knoten in Reihenfolge der Preorder in einen leeren Baum eingefügt werden, erhält man den gleichen Baum.
   Wahr! Da beim Einfügen immer zuerst die Wurzel eingefügt wird und danach die Kinder, ergibt das den gleichen Baum!
5. Wenn die Knoten in Reihenfolge der Postorder in einen leeren Baum eingefügt werden, erhält man den gleichen Baum.
   Falsch! Aber es stimmt für die umgekehrte Postorder!
6. Wenn die Knoten in Reihenfolge der Inorder in einen leeren Baum eingefügt werden, erhält man den gleichen Baum.
   Falsch! Es gibt viele verschiedene Bäume mit der gleichen Inorder.

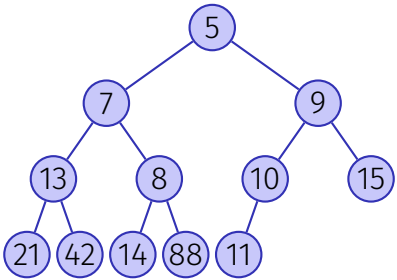
Fragen/Unklarheiten?

Heap

Führen Sie auf folgendem Min-Heap eine Extract-Min Operation aus, wie in der Vorlesung vorgestellt, einschliesslich der Wiederherstellung der Heap-Bedingung. Wie sieht der Heap nach der Operation aus?



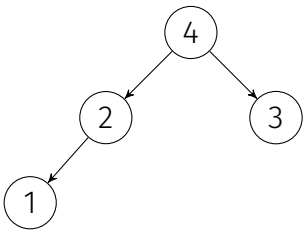
Lösung

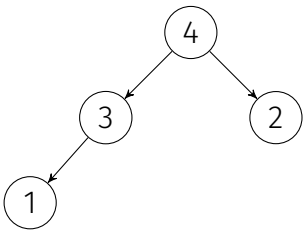


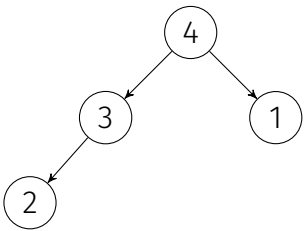
Fragen/Unklarheiten?

Quiz: Anzahl MaxHeaps mit n Schlüsseln

Sei $N(n)$ die Anzahl verschiedener MaxHeaps, welche aus allen Schlüsseln $1, 2, \ldots, n$ gebildet werden können. Beispielsweise ist $N(1) = 1,\ N(2) = 1,\ N(3) = 2,\ N(4) = 3$ und $N(5) = 8$.
Finde die Werte $N(6)$ und $N(7)$.

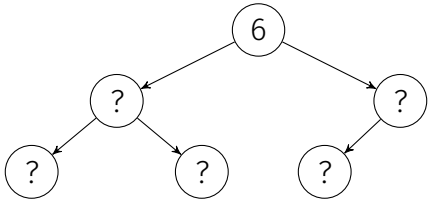






Anzahl MaxHeaps mit n verschiedenen Schlüsseln

Ein die Elemente $1, 2, 3, 4, 5, 6$ enthaltender MaxHeap sieht so aus:



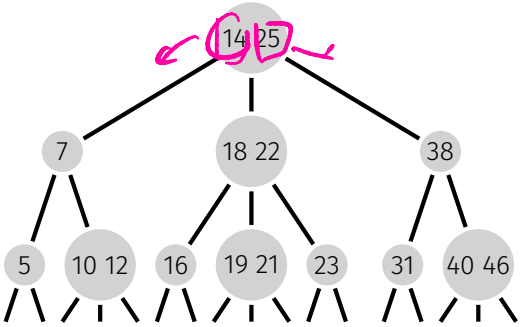
Möglichkeiten, Elemente des linken Teilbaums zu wählen: $\binom{5}{3}$.

$$\Rightarrow N(6) = \binom{5}{3} \cdot N(3) \cdot N(2) = 10 \cdot 2 \cdot 1 = 20.$$

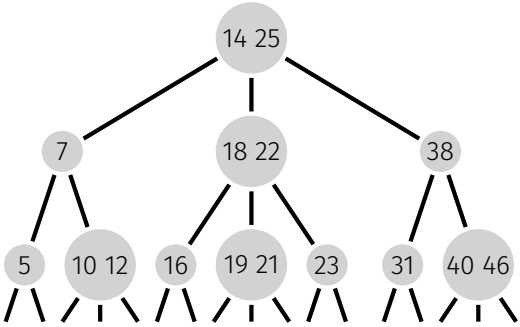
$$\text{und } N(7) = \binom{6}{3} \cdot N(3) \cdot N(3) = 20 \cdot 2 \cdot 2 = 80.$$

Fragen/Unklarheiten?

Suchen

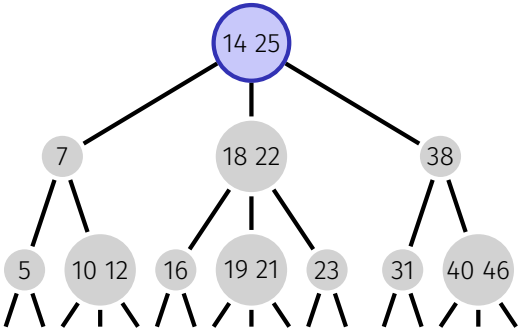


Suchen



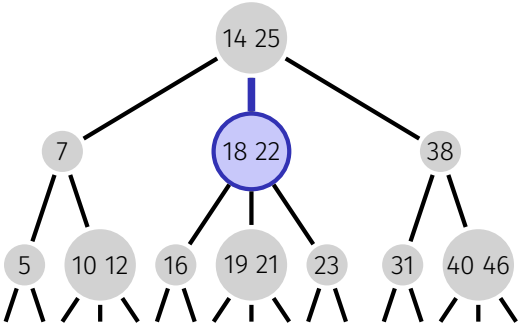
`search`(23)

Suchen



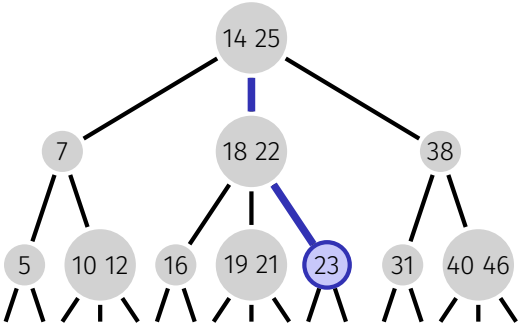
`search`(23)

Suchen



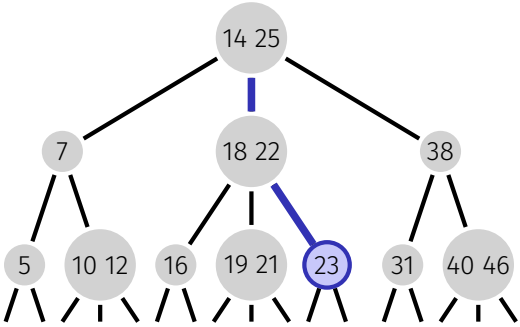
`search`(23)

Suchen



`search`(23)

Suchen



$\texttt{search}(23) \rightarrow \texttt{found}$

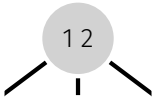
2-3-Baum: Einfügen

Fügen Sie die Schlüssel $1, \ldots, 7$ der Reihe nach in einen (initial leeren) 2-3-Baum ein und zeichnen Sie den resultierenden Baum nach jedem Schritt (`split`/`propagate`, `join`, ...).

2-3-Baum: Einfügen

`insert(1)`:
neuer Knoten

2-3-Baum: Einfügen

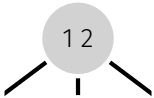


`insert(1)`:
neuer Knoten

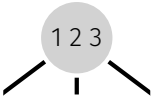
`insert(2)`:
`join`

2-3-Baum: Einfügen

`insert(1)`:
neuer Knoten



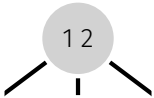
`insert(2)`:
`join`



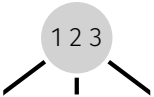
`insert(3)`:
4-Knoten

2-3-Baum: Einfügen

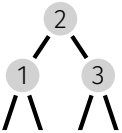
`insert(1)`:
neuer Knoten



`insert(2)`:
`join`



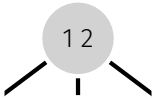
`insert(3)`:
4-Knoten



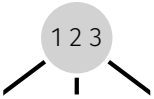
`insert(3)`:
`split/propagate`

2-3-Baum: Einfügen

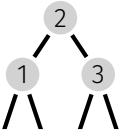
insert(1):
neuer Knoten



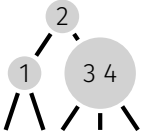
insert(2):
`join`



insert(3):
4-Knoten



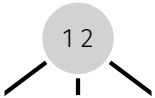
insert(3):
`split/propagate`



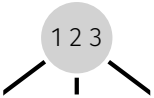
insert(4):
`join`

2-3-Baum: Einfügen

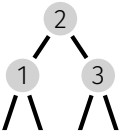
insert(1):
neuer Knoten



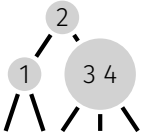
insert(2):
`join`



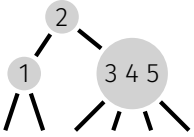
insert(3):
4-Knoten



insert(3):
`split/propagate`



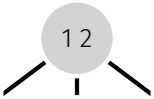
insert(4):
`join`



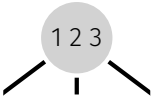
insert(5):
4-Knoten

2-3-Baum: Einfügen

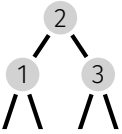
insert(1):
neuer Knoten



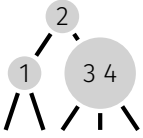
insert(2):
`join`



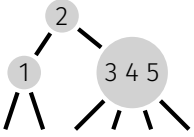
insert(3):
4-Knoten



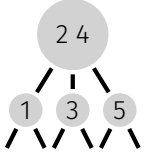
insert(3):
`split/propagate`



insert(4):
`join`



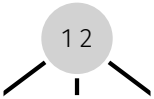
insert(5):
4-Knoten



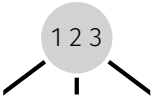
insert(5):
`split/propagate`

2-3-Baum: Einfügen

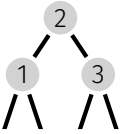
insert(1):
neuer Knoten



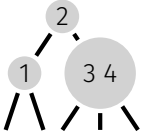
insert(2):
join



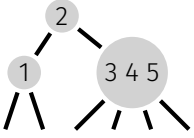
insert(3):
4-Knoten



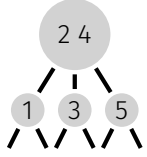
insert(3):
split/propagate



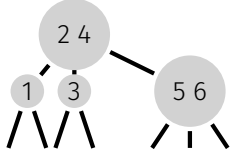
insert(4):
join



insert(5):
4-Knoten

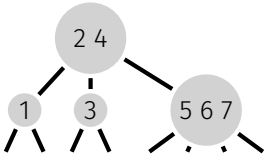


insert(5):
split/propagate



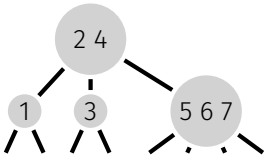
insert(6):
join

2-3-Baum: Einfügen

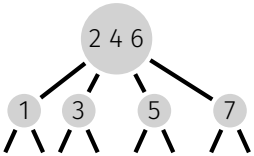


`insert(7)`:
4-Knoten

2-3-Baum: Einfügen

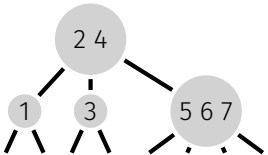


insert(7):
4-Knoten

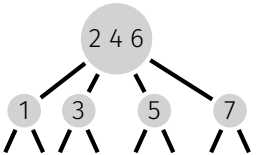


insert(7):
split/propagate

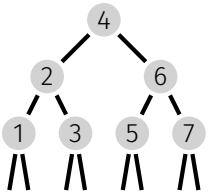
2-3-Baum: Einfügen



insert(7):
4-Knoten

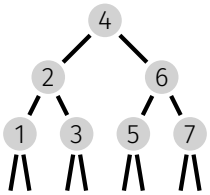


insert(7):
split/propagate



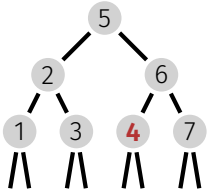
insert(7):
split/propagate

2-4-Baum: Löschen

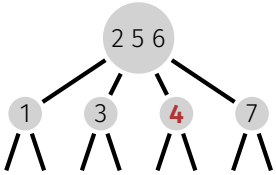


Löschen Sie vom resultierenden Baum nun den Schlüssel 4.

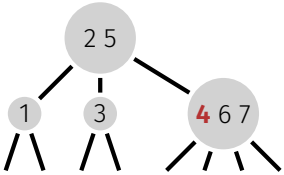
2-4-Baum: Löschen



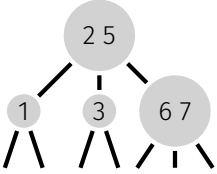
1. Tauschen



2. kreiere 4-Knoten bei Wurzel



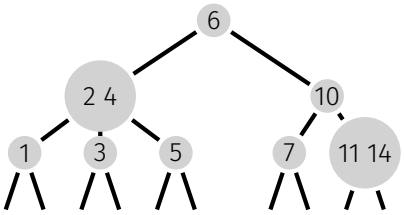
3. mit Geschwister kombinieren



4. Schlüssel löschen

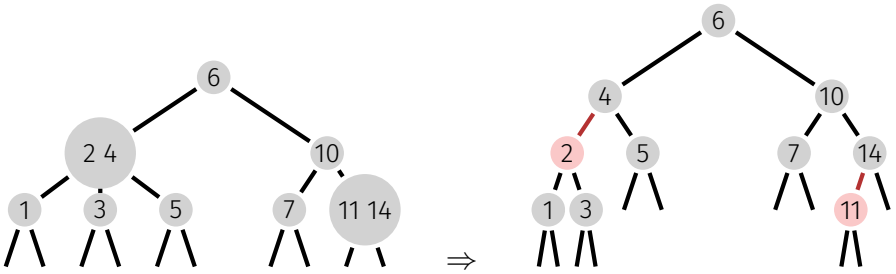
Rot-Schwarz-Bäume

Zeichnen Sie den folgenden 2-3-Baum als Rot-Schwarz-Baum.



Rot-Schwarz-Bäume

Zeichnen Sie den folgenden 2-3-Baum als Rot-Schwarz-Baum.



Rot-Schwarz-Bäume: Wahr oder falsch?

1. Rechter Spine (Pfad, der von Wurzel immer rechts geht) hat Länge $\lceil \log_2(n+1) \rceil$.

Rot-Schwarz-Bäume: Wahr oder falsch?

1. Rechter Spine (Pfad, der von Wurzel immer rechts geht) hat Länge $\lceil \log_2(n+1) \rceil$.
   Korrekt, da keine right-leaning roten Kanten und perfekte schwarze Balancierung.

Rot-Schwarz-Bäume: Wahr oder falsch?

1. Rechter Spine (Pfad, der von Wurzel immer rechts geht) hat Länge $\lceil \log_2(n+1) \rceil$.
   Korrekt, da keine right-leaning roten Kanten und perfekte schwarze Balancierung.
2. Die Anzahl roter Kanten ist höchstens die Anzahl schwarzer Kanten.

Rot-Schwarz-Bäume: Wahr oder falsch?

1. Rechter Spine (Pfad, der von Wurzel immer rechts geht) hat Länge $\lceil \log_2(n+1) \rceil$.
   Korrekt, da keine right-leaning roten Kanten und perfekte schwarze Balancierung.
2. Die Anzahl roter Kanten ist höchstens die Anzahl schwarzer Kanten.
   Falsch, ein Baum mit 2 Knoten und einer Kante hat eine rote aber keine schwarze Kante.

Rot-Schwarz-Bäume: Wahr oder falsch?

1. Rechter Spine (Pfad, der von Wurzel immer rechts geht) hat Länge $\lceil \log_2(n+1) \rceil$.
   Korrekt, da keine right-leaning roten Kanten und perfekte schwarze Balancierung.
2. Die Anzahl roter Kanten ist höchstens die Anzahl schwarzer Kanten.
   Falsch, ein Baum mit 2 Knoten und einer Kante hat eine rote aber keine schwarze Kante.
3. Alle Knoten im linken Teilbaum eines Knotens sind kleiner als der Knoten.

Rot-Schwarz-Bäume: Wahr oder falsch?

1. Rechter Spine (Pfad, der von Wurzel immer rechts geht) hat Länge $\lceil \log_2(n+1) \rceil$.
   Korrekt, da keine right-leaning roten Kanten und perfekte schwarze Balancierung.
2. Die Anzahl roter Kanten ist höchstens die Anzahl schwarzer Kanten.
   Falsch, ein Baum mit 2 Knoten und einer Kante hat eine rote aber keine schwarze Kante.
3. Alle Knoten im linken Teilbaum eines Knotens sind kleiner als der Knoten.
   Korrekt, da ein Rot-Schwarz-Baum ein Suchbaum ist.

Rot-Schwarz-Bäume: Einfügen

Fügen Sie die Zahlen $1, \ldots, 7$ der Reihe nach in einen (initial leeren) Rot-Schwarz-Baum ein und zeichnen Sie den Baum nach jedem Schritt.

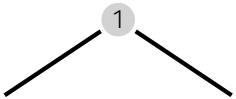
Rot-Schwarz-Bäume: Einfügen

Fügen Sie die Zahlen $1, \ldots, 7$ der Reihe nach in einen (initial leeren) Rot-Schwarz-Baum ein und zeichnen Sie den Baum nach jedem Schritt.

Vergleichen Sie die Schritte mit Ihrem Resultat für den 2-3-Baum vorher.

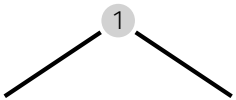
Rot-Schwarz-Bäume: Einfügen

(Korrekturfolien umrahmt)

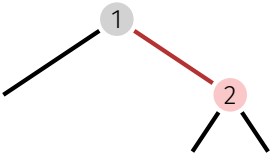


`insert(1)`

Rot-Schwarz-Bäume: Einfügen

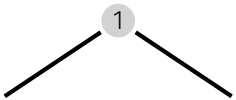


`insert(1)`

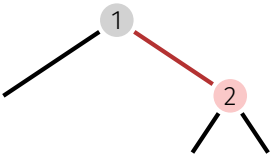


`insert(2): add`

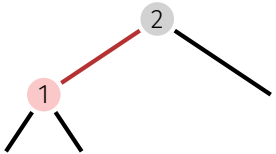
Rot-Schwarz-Bäume: Einfügen



`insert(1)`

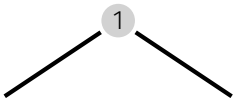


`insert(2): add`

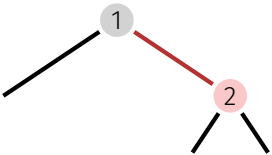


`insert(2): rotate_left`
(da rechtes Kind rot)

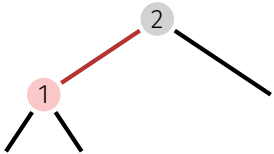
Rot-Schwarz-Bäume: Einfügen



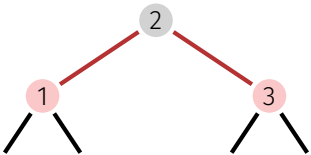
`insert(1)`



`insert(2): add`

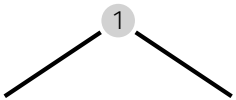


`insert(2): rotate_left`
(da rechtes Kind rot)

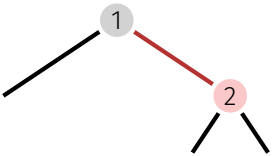


`insert(3): add`

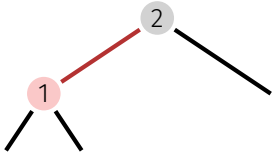
Rot-Schwarz-Bäume: Einfügen



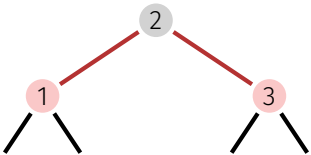
`insert(1)`



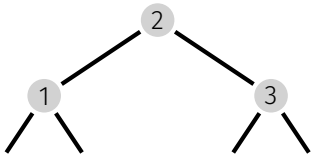
`insert(2): add`



`insert(2): rotate_left`
(da rechtes Kind rot)

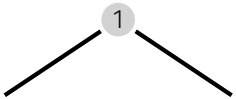


`insert(3): add`

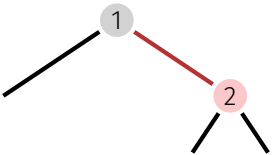


`insert(3): push_up`
(da zwei rote Kinder)

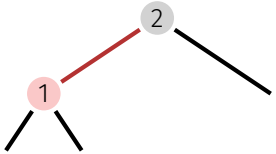
Rot-Schwarz-Bäume: Einfügen



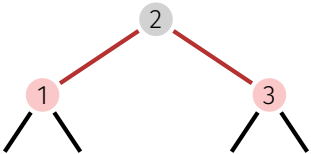
`insert(1)`



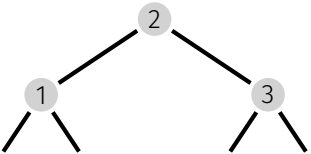
`insert(2): add`



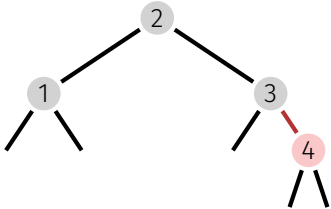
`insert(2): rotate_left`
(da rechtes Kind rot)



`insert(3): add`

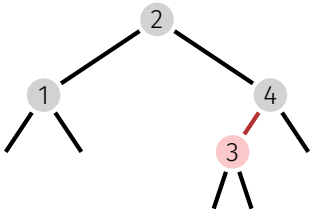


`insert(3): push_up`
(da zwei rote Kinder)



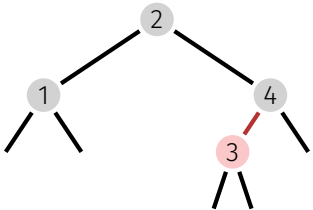
`insert(4): add`

Rot-Schwarz-Bäume: Einfügen

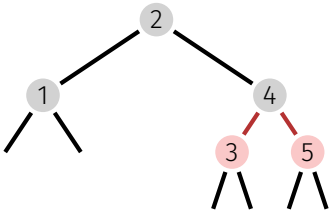


`insert(4): rotate_left`
(da rechtes Kind rot)

Rot-Schwarz-Bäume: Einfügen

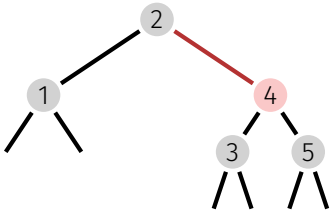


`insert(4): rotate_left`
(da rechtes Kind rot)



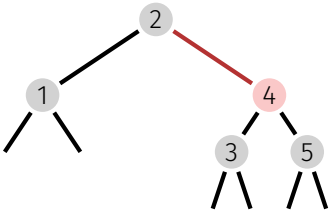
`insert(5): add`

Rot-Schwarz-Bäume: Einfügen

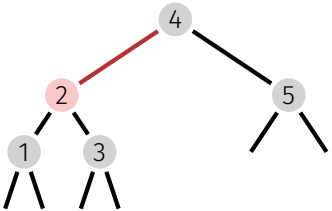


`insert(5): push_up`
(da zwei Kinder rot)

Rot-Schwarz-Bäume: Einfügen

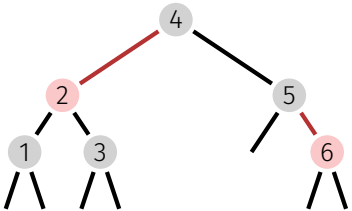


`insert(5): push_up`
(da zwei Kinder rot)



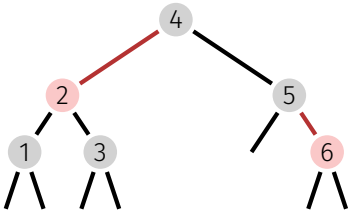
`insert(5): rotate_left`
(da rechtes Kind rot)

Rot-Schwarz-Bäume: Einfügen

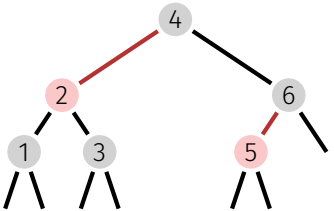



```
insert(6): add
```

Rot-Schwarz-Bäume: Einfügen

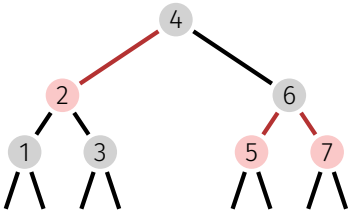


`insert(6): add`



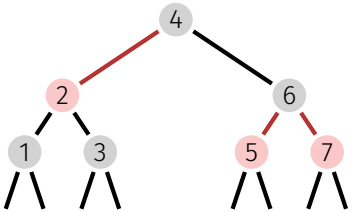
`insert(6): rotate_left`
(da rechtes Kind rot)

Rot-Schwarz-Bäume: Einfügen

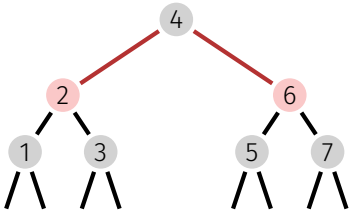



```
insert(7): add
```

Rot-Schwarz-Bäume: Einfügen

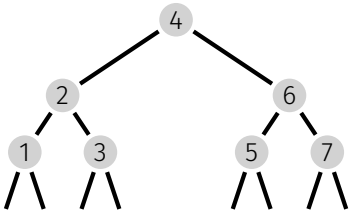


`insert(7): add`



`insert(7): push_up`
(da zwei Kinder rot)

Rot-Schwarz-Bäume: Einfügen



`insert(7): push_up`
(da beide Kinder rot)

6. Tipps zu [code]expert

Tipps für nächste [code]expert -Aufgaben

Allgemein

Tipps für nächste [code]expert -Aufgaben

Allgemein

- Macht euch Skizzen (für den Überblick)

Tipps für nächste [code]expert -Aufgaben

Allgemein

- Macht euch Skizzen (für den Überblick)
- Geht stumpf paar Algorithmen manuell durch (für die Intuition)

Tipps für nächste [code]expert -Aufgaben

Allgemein

- Macht euch Skizzen (für den Überblick)
- Geht stumpf paar Algorithmen manuell durch (für die Intuition)

Aufgabe "Binary Search Tree"

Tipps für nächste [code]expert -Aufgaben

Allgemein

- Macht euch Skizzen (für den Überblick)
- Geht stumpf paar Algorithmen manuell durch (für die Intuition)

Aufgabe "Binary Search Tree"

- Repetiert Klassen und Pointer

Tipps für nächste [code]expert -Aufgaben

Allgemein

- Macht euch Skizzen (für den Überblick)
- Geht stumpf paar Algorithmen manuell durch (für die Intuition)

Aufgabe "Binary Search Tree"

- Repetiert Klassen und Pointer

Tipps für nächste [code]expert -Aufgaben

Aufgabe "k-smallest element"

Tipps für nächste [code]expert -Aufgaben

Aufgabe "k-smallest element"

- Viele Ansätze möglich

Aufgabe "Red-Black Trees - Student Attempt"

Tipps für nächste [code]expert -Aufgaben

Aufgabe "k-smallest element"

- Viele Ansätze möglich

Aufgabe "Red-Black Trees - Student Attempt"

- Vergesst die Edge-Cases nicht

7. Coding-Aufgabe

Code-Beispiel

Exercise class 06: Binary Trees auf Code-Expert

- Binary Tree: Einfache Aufgaben
- Augmenting a Binary Search Tree

8. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schönes Wochenende!

Nächste Woche keine Übungsstunde