Datenstrukturen und Algorithmen

Übung 6

FS 2019

Programm von heute

1. Feedback letzte Übung
2. Wiederholung Theorie

Nachbesprechung

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

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

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

Coocoo 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, __, __, __
```

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: ...
```

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

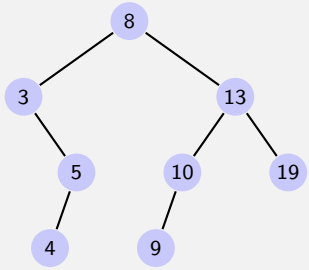
Finden eines Sub-Arrays

```
// 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;
}
```

2. Wiederholung Theorie

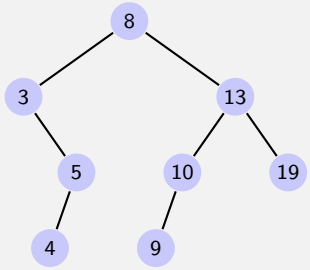
Traversierungsarten

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.



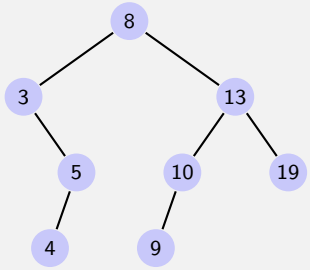
Traversierungsarten

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



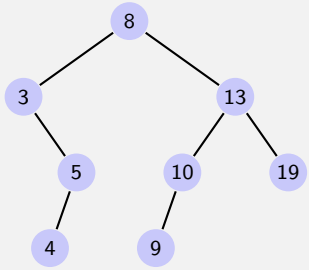
Traversierungsarten

- 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.



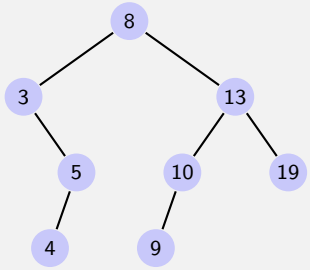
Traversierungsarten

- 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



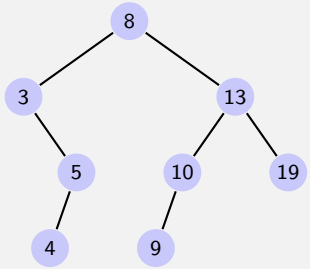
Traversierungsarten

- 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 $T_{\text{right}}(v)$.



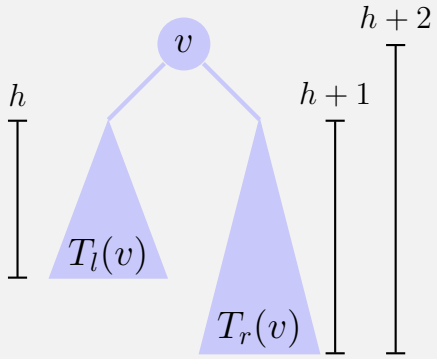
Traversierungsarten

- 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

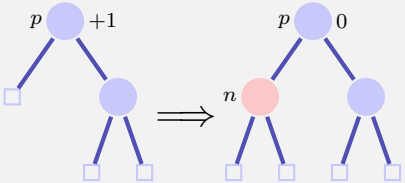


AVL Bedingung

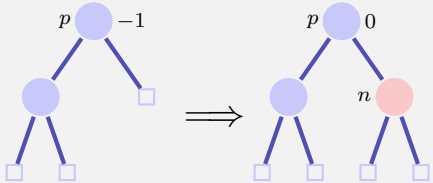
AVL Bedingung: für jeden Knoten v eines Baumes gilt $\mathrm{bal}(v) \in \{-1, 0, 1\}$



Balance am Einfügeort



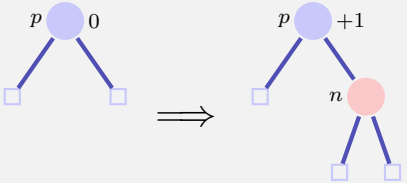
Fall 1: $\mathrm{bal}(p) = +1$



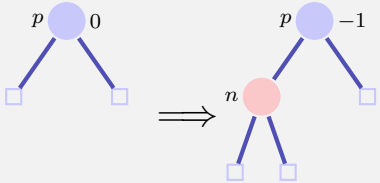
Fall 2: $\mathrm{bal}(p) = -1$

Fertig in beiden Fällen, denn der Teilbaum ist nicht gewachsen.

Balance am Einfügeort



Fall 3.1: $\mathrm{bal}(p) = 0$ rechts



Fall 3.2: $\mathrm{bal}(p) = 0$, links

In beiden Fällen noch nicht fertig. Aufruf von `upin(p)`.

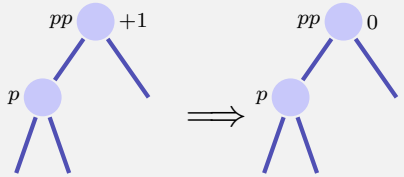
upin(p) - Invariante

Beim Aufruf von `upin(p)` gilt, dass

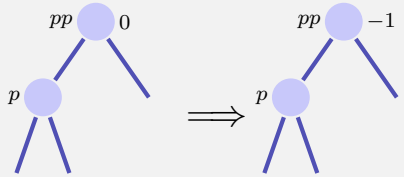
- der Teilbaum ab p gewachsen ist und
- $\mathrm{bal}(p) \in \{-1, +1\}$

upin(p)

Annahme: p ist linker Sohn von pp[1]



Fall 1: $\mathrm{bal}(pp) = +1$, fertig.



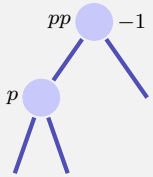
Fall 2: $\mathrm{bal}(pp) = 0$, `upin(pp)`

In beiden Fällen gilt nach der Operation die AVL-Bedingung für den Teilbaum ab pp

[1] Ist p rechter Sohn: symmetrische Fälle unter Vertauschung von $+1$ und -1

upin(p)

Annahme: p ist linker Sohn von pp



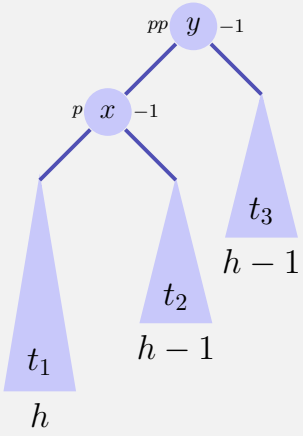
Fall 3: $\mathrm{bal}(pp) = -1$,

Dieser Fall ist problematisch: das Hinzufügen von n im Teilbaum ab pp hat die AVL-Bedingung verletzt. Rebalancieren!

Zwei Fälle $\mathrm{bal}(p) = -1$, $\mathrm{bal}(p) = +1$

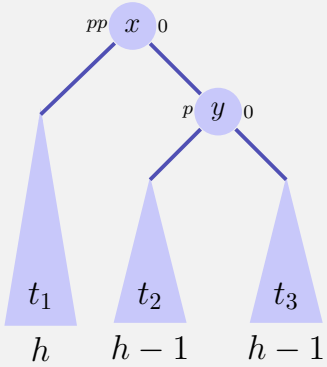
Rotationen

Fall 1.1 $\mathrm{bal}(p) = -1$. [2]



$\Longrightarrow$

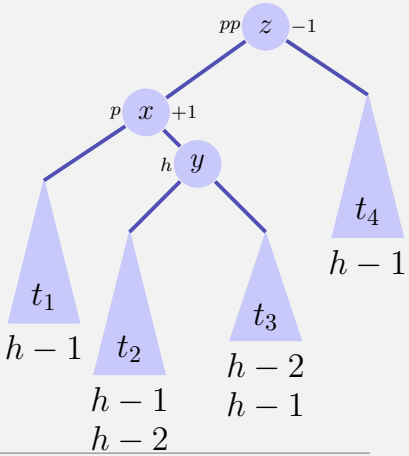
Rotation nach rechts



[2] p rechter Sohn: $\mathrm{bal}(pp) = \mathrm{bal}(p) = +1$, Linksrotation

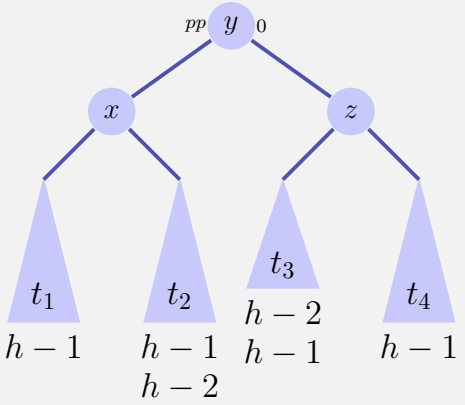
Rotationen

Fall 1.2 $\mathrm{bal}(p) = +1$. [3]



$\Longrightarrow$

Doppel-
rotation
links-
rechts



[3] p rechter Sohn: $\mathrm{bal}(pp) = +1$, $\mathrm{bal}(p) = -1$, Doppelrotation rechts links

Fragen?