

# Algorithmen I

## Tutorium 1 - 6. Sitzung

Dennis Felsing

`dennis.felsing@student.kit.edu`  
`www.stud.uni-karlsruhe.de/~ubcqr/algo`

2011-05-23



# Überblick

- 1 Binäre Suchbäume
- 2 Rot-Schwarz-Bäume

# Binäre Suchbäume

- 1 Binäre Suchbäume**
  - Allgemeines
  - Operationen
  - Einfügen und Löschen
  - Kreativaufgabe
  
- 2 Rot-Schwarz-Bäume**

# Binäre Suchbäume

## Idee

Suchen in  $O(\log n)$

## Definition

Ein **Binärer Suchbaum** ist ein Binärer Baum, für den die **Binäre-Suchbaum-Eigenschaft** gilt.

# Binäre Suchbäume

## Binäre-Suchbaum-Eigenschaft

Sei  $x$  ein Knoten in einem Binären Suchbaum.

Falls  $y$  ein Knoten im linken Unterbaum von  $x$  ist, dann gilt

$y.\text{schlüssel} \leq x.\text{schlüssel}$ .

Falls  $y$  ein Knoten im rechten Unterbaum von  $x$  ist, dann gilt

$y.\text{schlüssel} \geq x.\text{schlüssel}$ .

# Binäre Suchbäume

## Binäre-Suchbaum-Eigenschaft

Sei  $x$  ein Knoten in einem Binären Suchbaum.

Falls  $y$  ein Knoten im linken Unterbaum von  $x$  ist, dann gilt  
 $y.\text{schlüssel} \leq x.\text{schlüssel}$ .

Falls  $y$  ein Knoten im rechten Unterbaum von  $x$  ist, dann gilt  
 $y.\text{schlüssel} \geq x.\text{schlüssel}$ .

## Erinnerung: Heapeigenschaft

Bei Max-Heaps:  $A[\text{PARENT}(i)] \geq A[i]$

Unterschiede?

# Binäre Suchbäume

## Binäre-Suchbaum-Eigenschaft

Sei  $x$  ein Knoten in einem Binären Suchbaum.

Falls  $y$  ein Knoten im linken Unterbaum von  $x$  ist, dann gilt  
 $y.\text{schlüssel} \leq x.\text{schlüssel}$ .

Falls  $y$  ein Knoten im rechten Unterbaum von  $x$  ist, dann gilt  
 $y.\text{schlüssel} \geq x.\text{schlüssel}$ .

## Erinnerung: Heapeigenschaft

Bei Max-Heaps:  $A[\text{PARENT}(i)] \geq A[i]$

Unterschiede? Bei Binären Suchbäumen ist sortierte Ausgabe einfacher.

# Suchen

TREE-SEARCH( $x, k$ )

```
1  if  $x == \text{NIL}$  oder  $k == x.\text{schlüssel}$ 
2      return  $x$ 
3  if  $k < x.\text{schlüssel}$ 
4      return TREE-SEARCH( $x.\text{links}, k$ )
5  else return TREE-SEARCH( $x.\text{rechts}, k$ )
```

Ohne Rekursion?



# Suchen

TREE-SEARCH( $x, k$ )

```
1  if  $x == \text{NIL}$  oder  $k == x.\text{schlüssel}$ 
2      return  $x$ 
3  if  $k < x.\text{schlüssel}$ 
4      return TREE-SEARCH( $x.\text{links}, k$ )
5  else return TREE-SEARCH( $x.\text{rechts}, k$ )
```

Ohne Rekursion?

ITERATIVE-TREE-SEARCH( $x, k$ )

```
1  while  $x \neq \text{NIL}$  und  $k \neq x.\text{schlüssel}$ 
2      if  $k < x.\text{schlüssel}$ 
3           $x = x.\text{links}$ 
4      else  $x = x.\text{rechts}$ 
5  return  $x$ 
```

# Einfache Operationen

TREE-MINIMUM( $x$ )

# Einfache Operationen

TREE-MINIMUM( $x$ )

```
1  while  $x.links \neq \text{NIL}$ 
2       $x = x.links$ 
3  return  $x$ 
```

TREE-MAXIMUM( $x$ )

# Einfache Operationen

TREE-MINIMUM( $x$ )

```
1  while  $x.links \neq \text{NIL}$ 
2       $x = x.links$ 
3  return  $x$ 
```

TREE-MAXIMUM( $x$ )

```
1  while  $x.rechts \neq \text{NIL}$ 
2       $x = x.rechts$ 
3  return  $x$ 
```

# Einfügen

```
TREE-INSERT( $T, z$ )
1   $y = \text{NIL}$ 
2   $x = T.wurzel$ 
3  while  $x \neq \text{NIL}$ 
4       $y = x$ 
5      if  $z.schlüssel < x.schlüssel$ 
6           $x = x.links$ 
7      else  $x = x.rechts$ 
8   $z.vater = y$ 
9  if  $y == \text{NIL}$ 
10      $T.wurzel = z$ 
11 elseif  $z.schlüssel < y.schlüssel$ 
12      $y.links = z$ 
13 else  $y.rechts = z$ 
```

# Löschen

TRANSPLANT( $T, u, v$ )

```
1  if  $u.vater == \text{NIL}$ 
2       $T.wurzel = v$ 
3  elseif  $u == u.vater.links$ 
4       $u.vater.links = v$ 
5  else  $u.vater.rechts = v$ 
6  if  $v \neq \text{NIL}$ 
7       $v.vater = u.vater$ 
```

# Löschen

TREE-DELETE( $T, z$ )

# Löschen

TREE-DELETE( $T, z$ )

- 1 **if**  $z.links == \text{NIL}$  // Fall a)
- 2     TRANSPLANT( $T, z, z.rechts$ )



# Löschen

TREE-DELETE( $T, z$ )

- 1 **if**  $z.links == \text{NIL}$  // Fall a)
- 2     TRANSPLANT( $T, z, z.rechts$ )
- 3 **elseif**  $z.rechts == \text{NIL}$  // Fall b)
- 4     TRANSPLANT( $T, z, z.links$ )

# Löschen

TREE-DELETE( $T, z$ )

```
1  if  $z.links == NIL$  // Fall a)
2      TRANSPLANT( $T, z, z.rechts$ )
3  elseif  $z.rechts == NIL$  // Fall b)
4      TRANSPLANT( $T, z, z.links$ )
5  else  $y = TREE-MINIMUM(z.rechts)$  // Fall c)

10     TRANSPLANT( $T, z, y$ )
11      $y.links = z.links$ 
12      $y.links.vater = y$ 
```

# Löschen

TREE-DELETE( $T, z$ )

```
1  if  $z.links == NIL$  // Fall a)
2      TRANSPLANT( $T, z, z.rechts$ )
3  elseif  $z.rechts == NIL$  // Fall b)
4      TRANSPLANT( $T, z, z.links$ )
5  else  $y = TREE-MINIMUM(z.rechts)$  // Fall c) und d)
6      if  $y.vater \neq z$  // Fall d)
7          TRANSPLANT( $T, y, y.rechts$ )
8           $y.rechts = z.rechts$ 
9           $y.rechts.vater = y$ 
10     TRANSPLANT( $T, z, y$ )
11      $y.links = z.links$ 
12      $y.links.vater = y$ 
```

# Kreativaufgabe

## Aufgabe

Schreibe eine Prozedur  $\text{INORDER-TREE-WALK}(w)$ , so dass  $\text{INORDER-TREE-WALK}(T.\text{wurzel})$  alle Elemente eines Binären Suchbaumes  $T$  in sortierter Reihenfolge ausgibt.

# Kreativaufgabe

## Aufgabe

Schreibe eine Prozedur  $\text{INORDER-TREE-WALK}(w)$ , so dass  $\text{INORDER-TREE-WALK}(T.\text{wurzel})$  alle Elemente eines Binären Suchbaumes  $T$  in sortierter Reihenfolge ausgibt.

- Einfach: Rekursiv

# Kreativaufgabe

## Aufgabe

Schreibe eine Prozedur  $\text{INORDER-TREE-WALK}(w)$ , so dass  $\text{INORDER-TREE-WALK}(T.\text{wurzel})$  alle Elemente eines Binären Suchbaumes  $T$  in sortierter Reihenfolge ausgibt.

- Einfach: Rekursiv
- Mittel: Nichtrekursiv mit Stack

# Kreativaufgabe

## Aufgabe

Schreibe eine Prozedur  $\text{INORDER-TREE-WALK}(w)$ , so dass  $\text{INORDER-TREE-WALK}(T.\text{wurzel})$  alle Elemente eines Binären Suchbaumes  $T$  in sortierter Reihenfolge ausgibt.

- Einfach: Rekursiv
- Mittel: Nichtrekursiv mit Stack
- Schwer: Nichtrekursiv ohne zusätzliche Datenstrukturen

# Kreativaufgabe

## Aufgabe

Schreibe eine Prozedur  $\text{INORDER-TREE-WALK}(w)$ , so dass  $\text{INORDER-TREE-WALK}(T.wurzel)$  alle Elemente eines Binären Suchbaumes  $T$  in sortierter Reihenfolge ausgibt.

- Einfach: Rekursiv
- Mittel: Nichtrekursiv mit Stack
- Schwer: Nichtrekursiv ohne zusätzliche Datenstrukturen

Welchen Zeit- und Platzaufwand hat dein Algorithmus?



# Kreativaufgabe

## Aufgabe

Schreibe eine Prozedur  $\text{INORDER-TREE-WALK}(w)$ , so dass  $\text{INORDER-TREE-WALK}(T.wurzel)$  alle Elemente eines Binären Suchbaumes  $T$  in sortierter Reihenfolge ausgibt.

- Einfach: Rekursiv
- Mittel: Nichtrekursiv mit Stack
- Schwer: Nichtrekursiv ohne zusätzliche Datenstrukturen

Welchen Zeit- und Platzaufwand hat dein Algorithmus?

Kommt dein Algorithmus ohne Vater-Zeiger aus?

# Probleme

Schnelle Laufzeiten bei balancierten Suchbäumen. Balance kann aber nicht garantiert werden.

Abhilfe schaffen **Rot-Schwarz-Bäume**.

# Rot-Schwarz-Bäume

## 1 Binäre Suchbäume

## 2 Rot-Schwarz-Bäume

- Eigenschaften
- Drehungen
- Einfügen

# Rot-Schwarz-Bäume

## Definition

Ein **Rot-Schwarz-Baum** ist ein Binärer Suchbaum mit einem zusätzlichen Knotenattribut **Farbe**, das entweder den Wert ROT oder SCHWARZ hat.

## Rot-Schwarz-Baum-Eigenschaften

- 1 Jeder Knoten ist entweder rot oder schwarz
- 2 Die Wurzel ist schwarz
- 3 Jedes Blatt (NIL) ist schwarz
- 4 Falls ein Knoten rot ist, sind seine Kinder schwarz
- 5 Für jeden Knoten enthalten alle Pfade vom Knoten zu einem Blatt die selbe Anzahl schwarzer Knoten

# Höhe

## Definition

Die **Schwarz-Höhe eines Knotens**  $x$ ,  $bh(x)$ , ist die Anzahl der schwarzen Knoten zwischen  $x$  und einem Blatt, ohne  $x$  selbst. Die **Schwarz-Höhe eines Baumes** ist die Schwarz-Höhe der Wurzel.

# Höhe

## Definition

Die **Schwarz-Höhe eines Knotens**  $x$ ,  $bh(x)$ , ist die Anzahl der schwarzen Knoten zwischen  $x$  und einem Blatt, ohne  $x$  selbst. Die **Schwarz-Höhe eines Baumes** ist die Schwarz-Höhe der Wurzel.

## Höhe

Ein Rot-Schwarz-Baum mit  $n$  inneren Knoten hat höchstens die Höhe  $2 \lg(n + 1)$ .

Was heißt das?

# Höhe

## Definition

Die **Schwarz-Höhe eines Knotens**  $x$ ,  $bh(x)$ , ist die Anzahl der schwarzen Knoten zwischen  $x$  und einem Blatt, ohne  $x$  selbst. Die **Schwarz-Höhe eines Baumes** ist die Schwarz-Höhe der Wurzel.

## Höhe

Ein Rot-Schwarz-Baum mit  $n$  inneren Knoten hat höchstens die Höhe  $2 \lg(n + 1)$ .

Was heißt das? Jeder Pfad von der Wurzel zu einem Blatt ist maximal doppelt so lang wie jeder andere.

# Höhe

## Definition

Die **Schwarz-Höhe eines Knotens**  $x$ ,  $bh(x)$ , ist die Anzahl der schwarzen Knoten zwischen  $x$  und einem Blatt, ohne  $x$  selbst. Die **Schwarz-Höhe eines Baumes** ist die Schwarz-Höhe der Wurzel.

## Höhe

Ein Rot-Schwarz-Baum mit  $n$  inneren Knoten hat höchstens die Höhe  $2 \lg(n + 1)$ .

Was heißt das? Jeder Pfad von der Wurzel zu einem Blatt ist maximal doppelt so lang wie jeder andere. Operationen TREE-SEARCH, TREE-MINIMUM, TREE-MAXIMUM lassen sich in  $O(\lg n)$  ausführen.



# Höhe

## Definition

Die **Schwarz-Höhe eines Knotens**  $x$ ,  $bh(x)$ , ist die Anzahl der schwarzen Knoten zwischen  $x$  und einem Blatt, ohne  $x$  selbst. Die **Schwarz-Höhe eines Baumes** ist die Schwarz-Höhe der Wurzel.

## Höhe

Ein Rot-Schwarz-Baum mit  $n$  inneren Knoten hat höchstens die Höhe  $2 \lg(n + 1)$ .

Was heißt das? Jeder Pfad von der Wurzel zu einem Blatt ist maximal doppelt so lang wie jeder andere. Operationen `TREE-SEARCH`, `TREE-MINIMUM`, `TREE-MAXIMUM` lassen sich in  $O(\lg n)$  ausführen. `TREE-INSERT` und `TREE-DELETE` können wir nicht direkt verwenden, da Rot-Schwarz-Baum-Eigenschaften nicht erhalten bleiben.

# Rot-Schwarz-Bäume

## Aufgabe

Zeichne einen Binären Suchbaum der Höhe 3 mit den Schlüsseln  $\{1, 2, \dots, 15\}$ . Füge die NIL-Blätter hinzu und färbe die Knoten auf drei verschiedene Arten, so dass die Schwarz-Höhe der resultierenden Rot-Schwarz-Bäume 2, 3 und 4 ist.

# Drehungen

LEFT-ROTATE( $T, x$ )

```
1   $y = x.rechts$ 
2   $x.rechts = y.links$ 
3  if  $y.links \neq T.nil$ 
4       $y.left.vater = x$ 
5   $y.vater = x.vater$ 
6  if  $x.vater == T.nil$ 
7       $T.wurzel = y$ 
8  elseif  $x == x.vater.links$ 
9       $x.vater.links = y$ 
10 else  $x.vater.rechts = y$ 
11  $y.links = x$ 
12  $x.vater = y$ 
```

Drehungen erhalten die Binärer-Suchbaum-Eigenschaft.

# Einfügen

## Idee

- 1 Füge Knoten  $z$  an passender Stelle ein
- 2 Färbe  $z$  rot
- 3 Behebe entstandene Verstöße gegen Rot-Schwarz-Baum-Eigenschaften

# Einfügen

## Idee

- 1 Füge Knoten  $z$  an passender Stelle ein
- 2 Färbe  $z$  rot
- 3 Behebe entstandene Verstöße gegen Rot-Schwarz-Baum-Eigenschaften

## Verstöße

- 1 Onkel von  $z$  ist rot
- 2 Onkel von  $z$  ist schwarz und  $z$  rechtes Kind
- 3 Onkel von  $z$  ist schwarz und  $z$  linkes Kind

# Übersicht

- 1 **Binäre Suchbäume**
  - Allgemeines
  - Operationen
  - Einfügen und Löschen
  - Kreativaufgabe
  
- 2 **Rot-Schwarz-Bäume**
  - Eigenschaften
  - Drehungen
  - Einfügen



calamitiesofnature.com © 2010 Tony Piro