Grundzüge der Informatik 1

Vorlesung 16

Überblick Vorlesung

Überblick

- Wiederholung
  - Ein grundlegendes Datenverwaltungsproblem
  - Elementare Datenstrukturen und ihre Analyse
  - Binärbäume
  - Suchbaumeigenschaft
- Binäre Suchbäume
  - Schlüsselsuche, Minimumsuche, Nachfolgersuche
  - Einfügen
  - Löschen
- Rot-Schwarz-Bäume
  - Einführung

Datenstrukturen

Was ist eine Datenstruktur?

- Eine Datenstruktur ist eine Anordnung von Daten im Speicher eines Rechners, die effizienten Zugriff auf die Daten ermöglicht
- Datenstrukturen für viele unterschiedliche Anfragen vorstellbar

Datenstrukturen

Grundlegendes Datenverwaltungsproblem

- Organisiere die Daten im Speicher eines Rechners so, dass folgende Operationen effizient durchgeführt werden können (S die aktuelle Menge der Objekte):
- Suchen(S,k):
  - Es wird ein Zeiger x auf ein Objekt mit Schlüssel k=key[x] zurückgegeben oder NIL, wenn es kein Objekt mit Schlüssel k in S gibt
- Einfügen(S,x):
  - Objekt x wird in S eingefügt
- Löschen(S,x):
  - Objekt x wird aus S entfernt

Datenstrukturen

Drei grundlegende Datenstrukturen

- Feld
- sortiertes Feld
- doppelt verkettete Liste

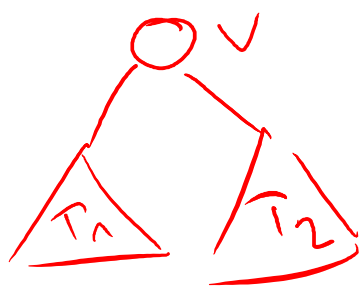
Diskussion

- Alle drei Strukturen haben gewichtige Nachteile
- Zeiger/Referenzen helfen beim Speichermanagement
- Sortierung hilft bei Suche ist aber teuer aufrecht zu erhalten

Datenstrukturen

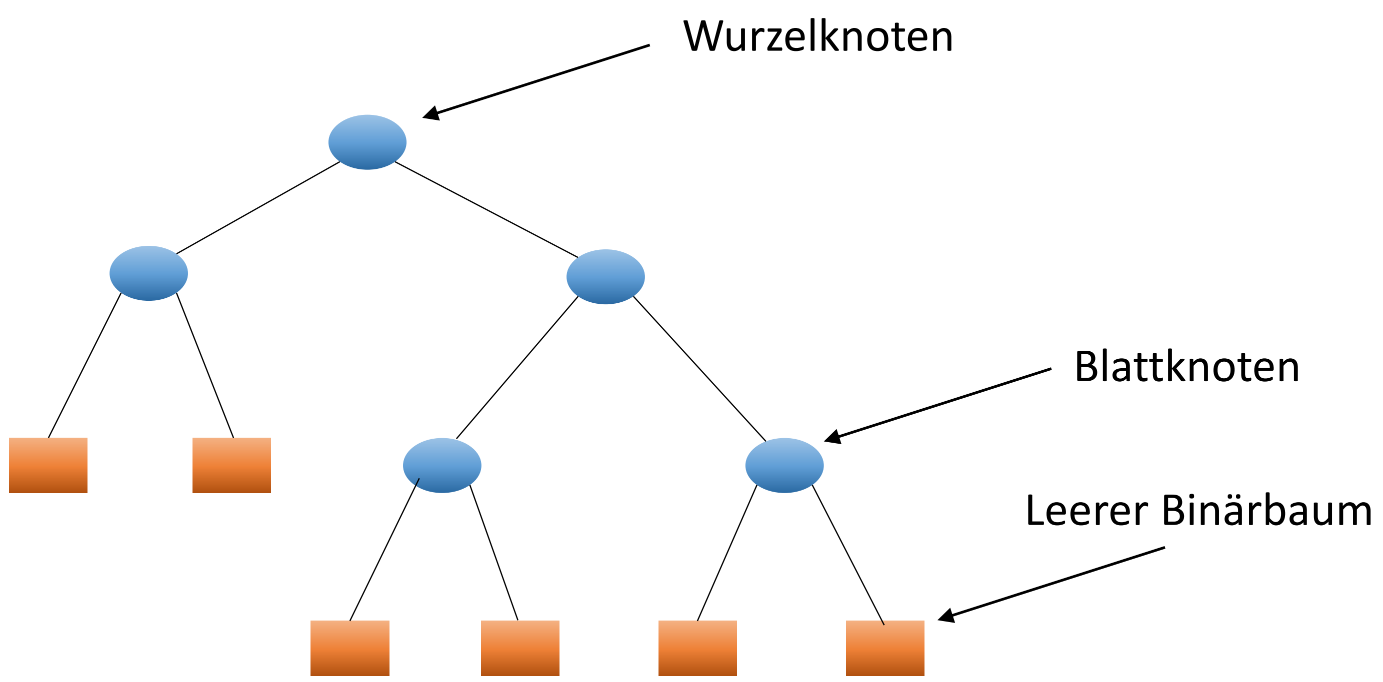
Definition (Binärbaum)

- Ein Binärbaum T ist eine Struktur, die auf einer endlichen Menge definiert ist. Diese Menge nennt man auch die Knotenmenge des Binärbaums.
- Die leere Menge ist ein Binärbaum. Dieser wird auch als leerer Baum bezeichnet.
- Ein Binärbaum ist ein Tripel (v, T_1, T_2), wobei T_1 und T_2 Binärbäume mit disjunkten Knotenmengen V_1 und V_2 sind und $v \notin V_1 \cup V_2$ Wurzelknoten heißt. Die Knotenmenge des Baums ist dann $\{v\} \cup V_1 \cup V_2$.
  T_1 heißt linker Unterbaum von v und T_2 heißt rechter Unterbaum von v.



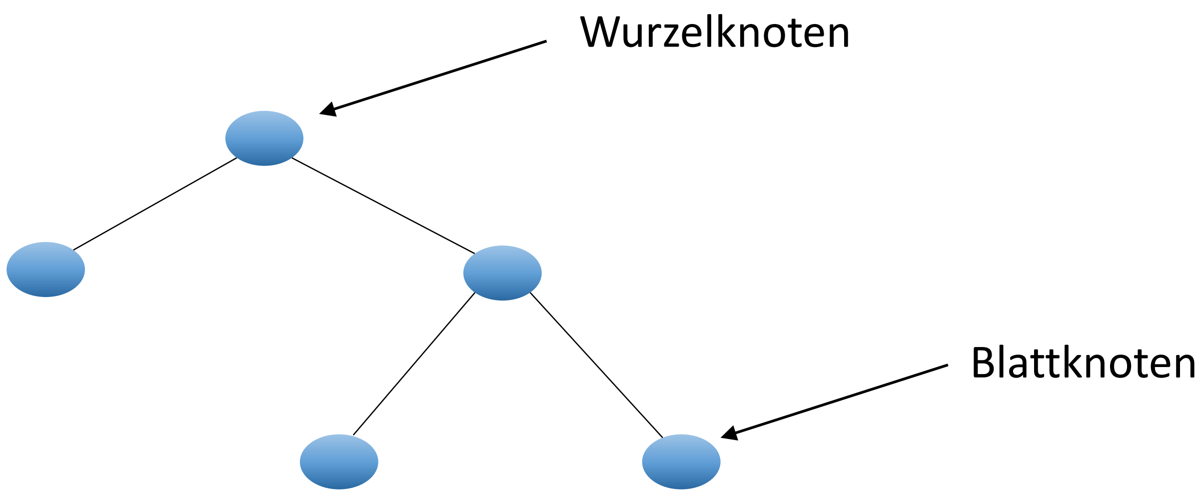
Datenstrukturen

Darstellung von Binärbäumen



Datenstrukturen

Darstellung von Binärbäumen

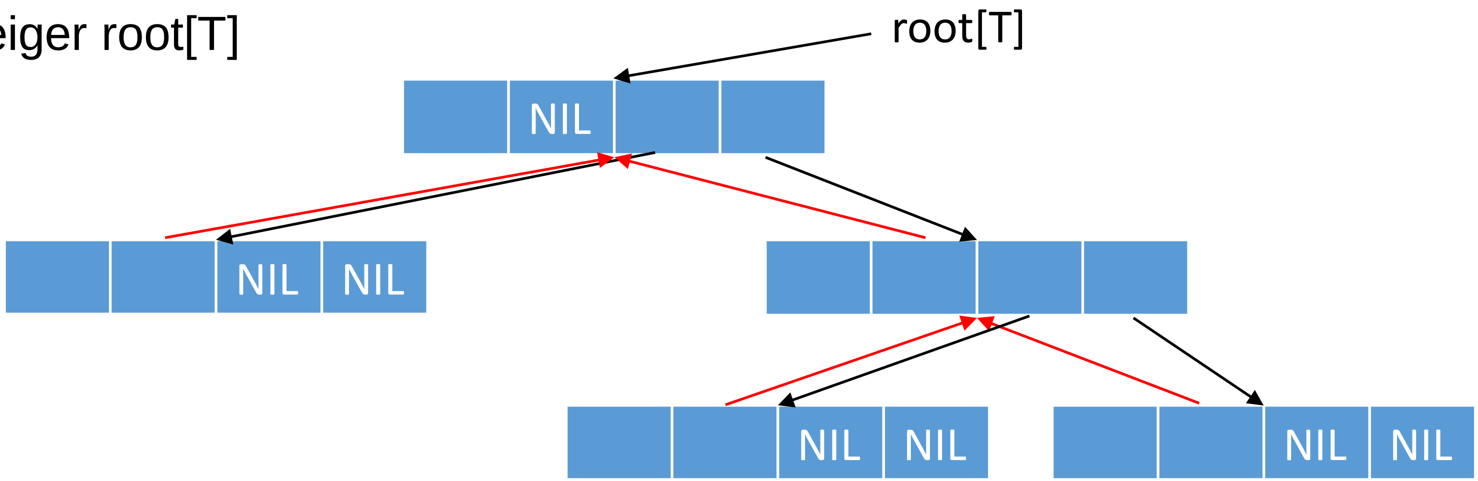


- Häufig lässt man die leeren Bäume in der Darstellung weg

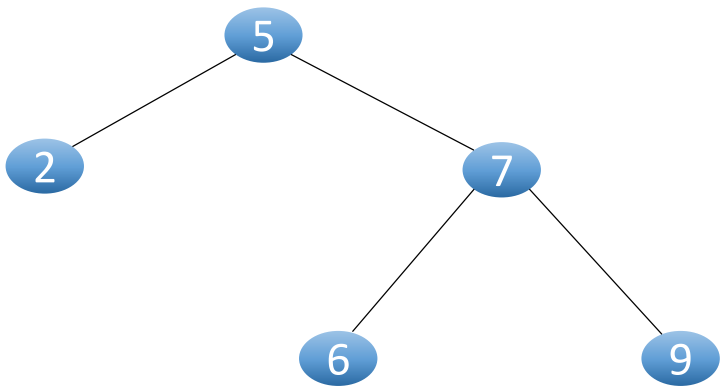
Datenstrukturen

Binärbäume T(Darstellung im Rechner)

- Ein Knoten v ist ein Verbundobjekt bestehend aus
  - Schlüssel key[v] und ggf. weitere Daten
  - Vaterzeiger parent[v] auf Vater von v
  - Zeiger left[v] und right[v] auf linkes bzw. rechtes Kind von v
- Wurzelzeiger root[T]



Datenstrukturen



Binäre Suchbäume

- Verwende Binärbaum
- Speichere Schlüssel „geordnet“

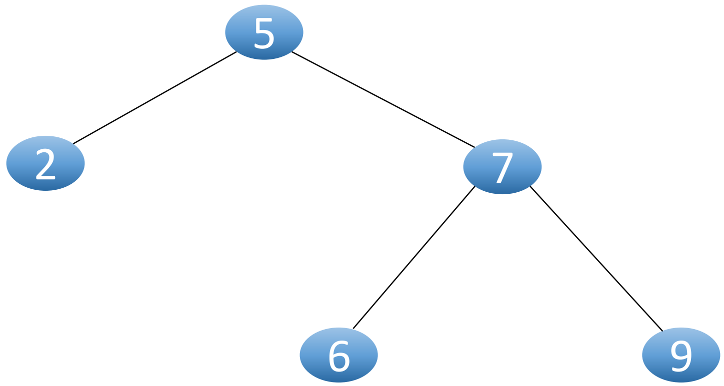
Binäre Suchbaumeigenschaft

- Sei x Knoten im binären Suchbaum
- Ist y Knoten im linken Unterbaum von x, dann gilt $key[y] \leq key[x]$
- Ist y Knoten im rechten Unterbaum von x, dann gilt $key[y] > key[x]$

Datenstrukturen

Suchen in Binärbäumen

- Gegeben ist Schlüssel k
- Gesucht ist ein Knoten mit Schlüssel k

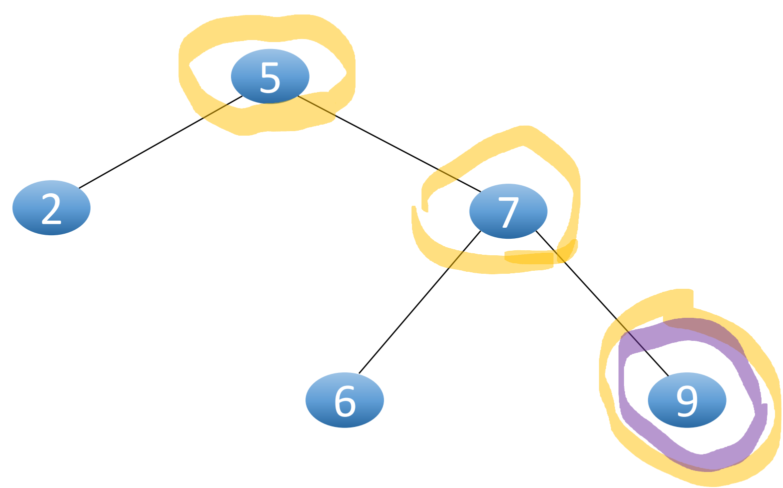


Datenstrukturen

Baumsuche(x,k)

1. **if** x=NIL **or** k=key[x] **then return** x
2. **if** k<key[x] **then return** Baumsuche(left[x],k)
3. **else return** Baumsuche(right[x],k)

Suche nach 9



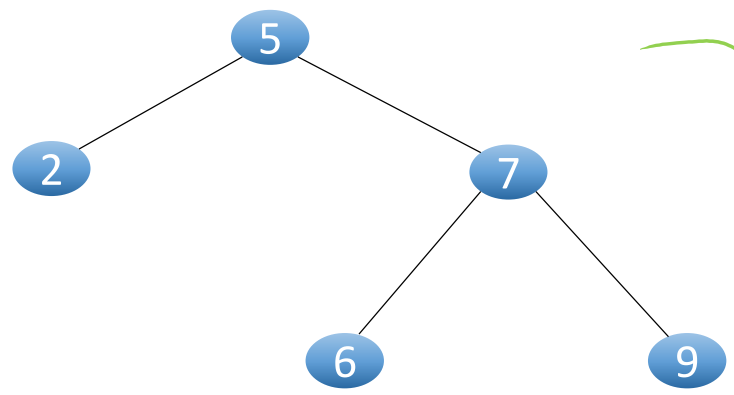
Datenstrukturen

Laufzeit O(h)

Baumsuche(x,k)

1. **if** x=NIL **or** k=key[x] **then return** x
2. **if** k<key[x] **then return** Baumsuche(left[x],k)
3. **else return** Baumsuche(right[x],k)

h: Höhe des Suchbaums



O(h)

Datenstrukturen

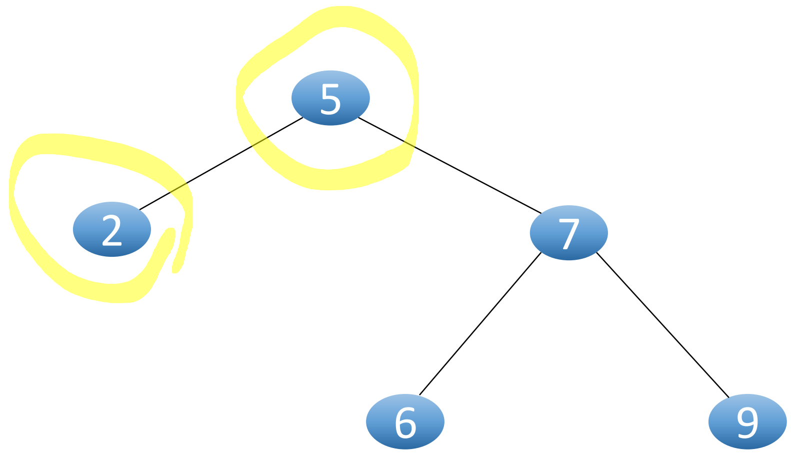
Minimum- und Maximumsuche

- Suchbaumeigenschaft:
  - Alle Schlüssel im rechten Unterbaum eines Knotens x sind größer als key[x]
  - Alle Schlüssel im linken Unterbaum von x sind kleiner oder gleich key[x]

Datenstrukturen

MinimumSuche(x)

1. **while** left[x]≠NIL **do** x = left[x]
2. **return** x

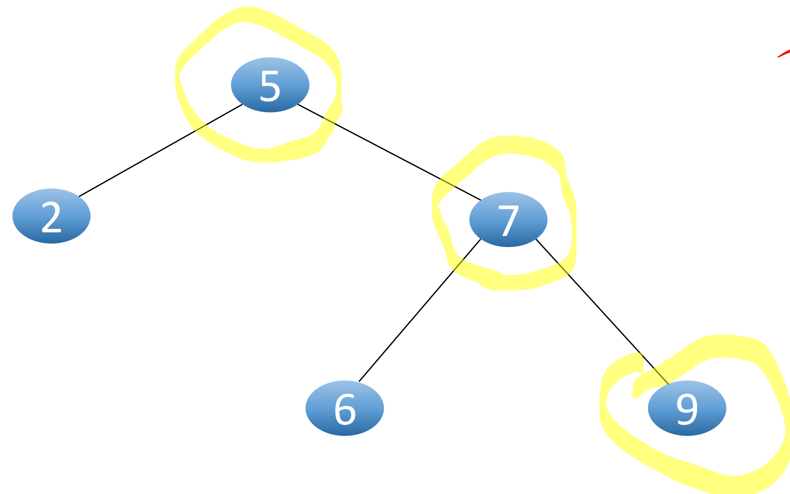


Universität
zu Köln

Datenstrukturen

MaximumSuche(x)

1. **while** right[x]≠NIL **do** x = right[x]
2. **return** x

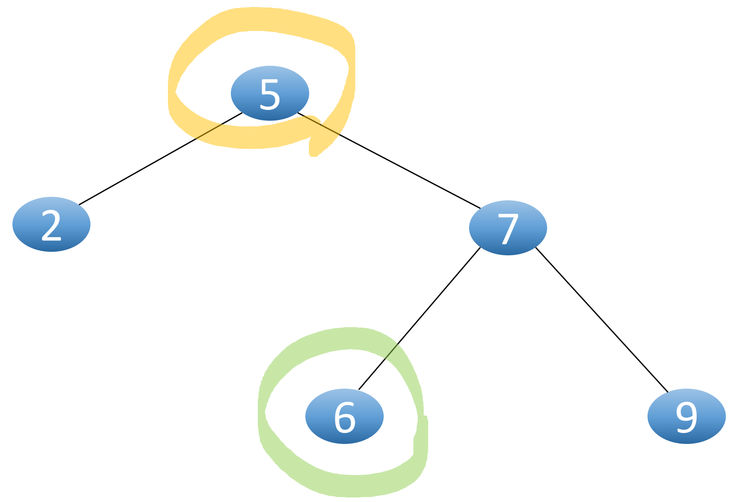


Laufzeit $O(h)$

Datenstrukturen

Nachfolgersuche

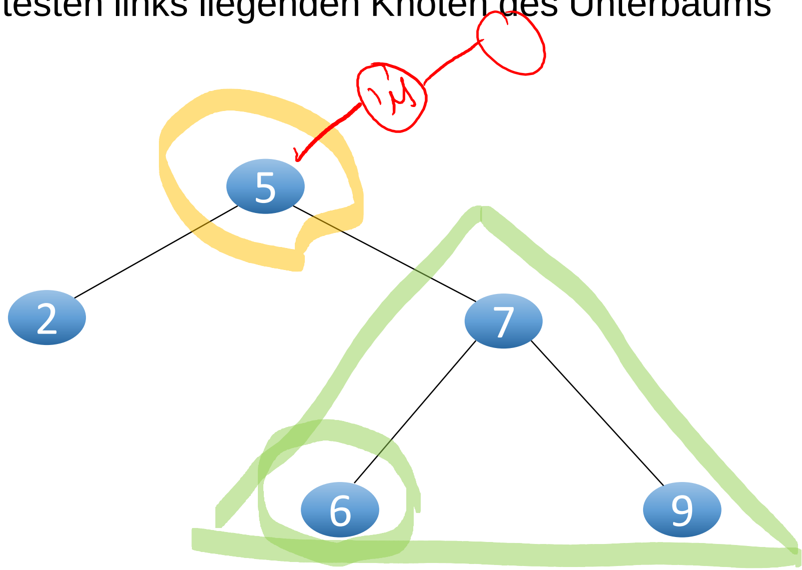
- Nachfolgerknoten bzgl. Inorder-Tree-Walk
- Dieser enthält den nächstgrößeren Schlüssel (wir erlaube kein mehrfaches Vorkommen)



Datenstrukturen

Nachfolgersuche

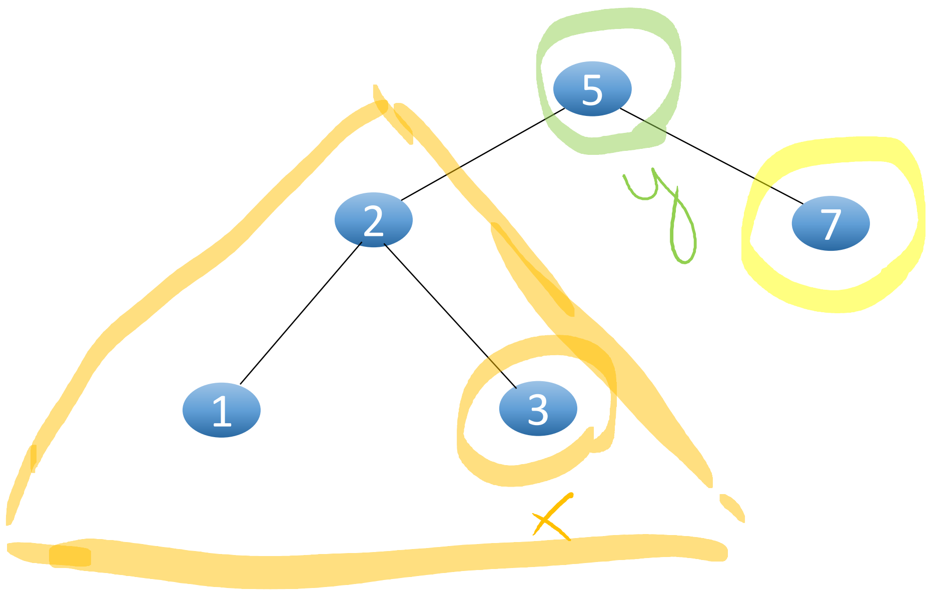
- Fall 1 (rechter Unterbaum von x nichtleer):
  - Dann ist der Knoten mit kleinstem Schlüssel im rechten Unterbaum der Nachfolger von x
  - Dieser ist der am weitesten links liegenden Knoten des Unterbaums



Datenstrukturen

Nachfolgersuche

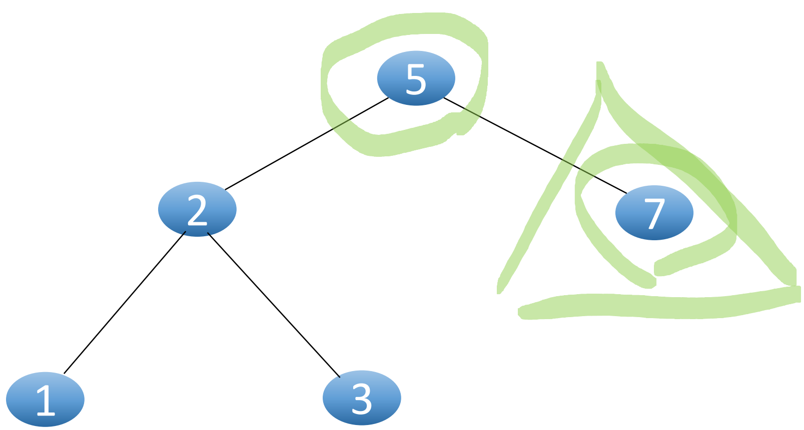
- Fall 2 (rechter Unterbaum von x leer):
  - Dann ist y der erste Knoten auf dem Pfad zur Wurzel, mit größerem Schlüssel als key[x] ist
  - Gibt es keinen solchen Knoten, dann ist key[x] der größte Schlüssel im Baum



Datenstrukturen

Nachfolgersuche(x)

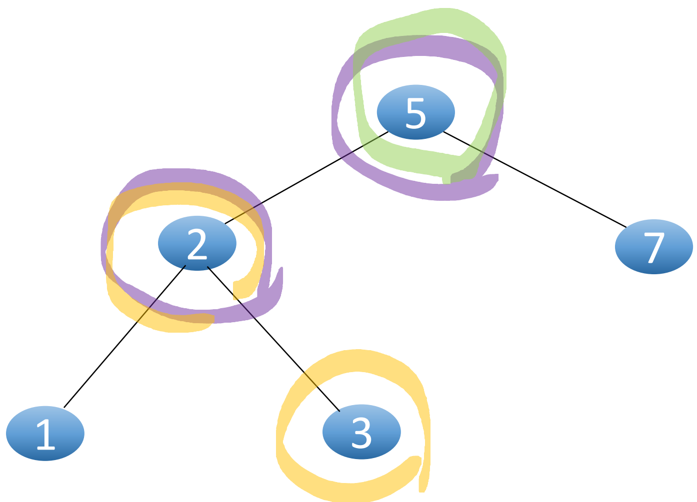
if right[x] $\neq$ NIL **then return** MinimumSuche(right[x])
y = parent[x]
while y $\neq$ NIL and x = right[y] **do**
    x = y
    y = parent[y]
return y



Datenstrukturen

Nachfolgersuche(x)

1. **if** right[x] $\neq$ NIL **then return** MinimumSuche(right[x])
2. y = parent[x]
3. **while** y $\neq$ NIL and x = right[y] **do**
4.   x = y
5.   y = parent[y]
6. **return** y



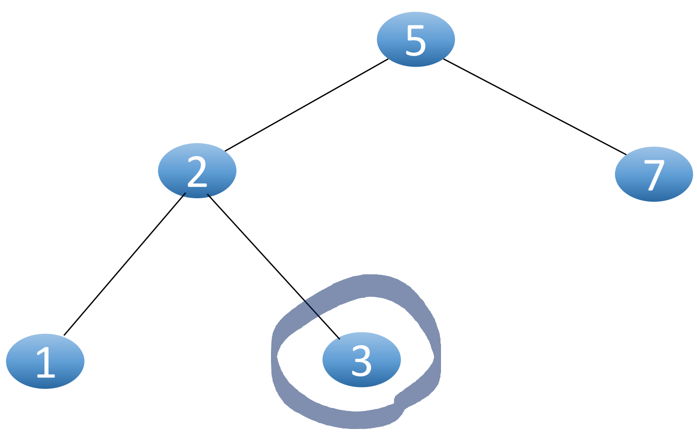
Datenstrukturen

Nachfolgersuche(x)

1. **if** right[x] $\neq$ NIL **then return** MinimumSuche(right[x]) $O(h)$
2. y = parent[x]
3. **while** y $\neq$ NIL and x = right[y] **do** $O(h)$
4.   x = y
5.   y = parent[y]
6. **return** y

$O(1)$

$O(h)$



Datenstrukturen

Vorgängersuche

- Symmetrisch zu Nachfolgersuche
- Daher ebenfalls O(h) Laufzeit

Datenstrukturen

Binäre Suchbäume

- Aufzählen der Elemente mit Inorder-Tree-Walk in O(n) Zeit
- Suche in O(h) Zeit
- Minimum/Maximum in O(h) Zeit
- Vorgänger/Nachfolger in O(h) Zeit

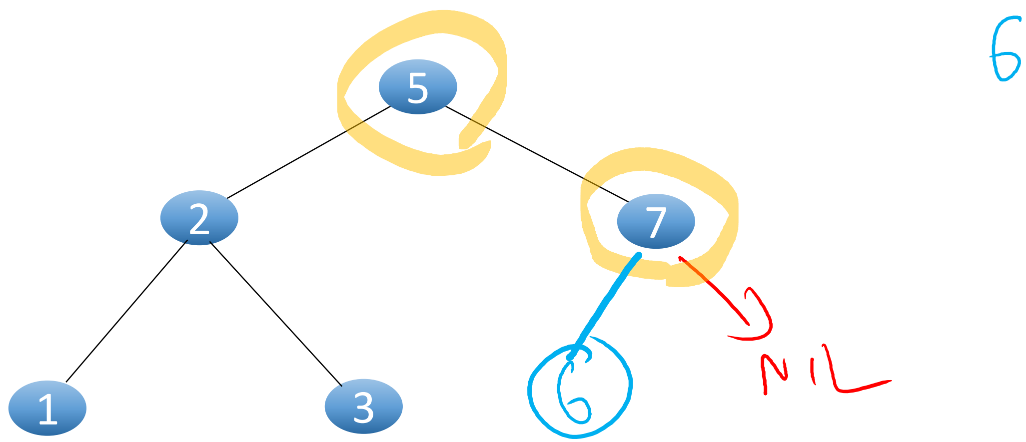
Dynamische Operationen?

- Einfügen und Löschen
- Müssen Suchbaumeigenschaft aufrecht erhalten
- Auswirkung auf Höhe des Baums?

Datenstrukturen

Einfügen

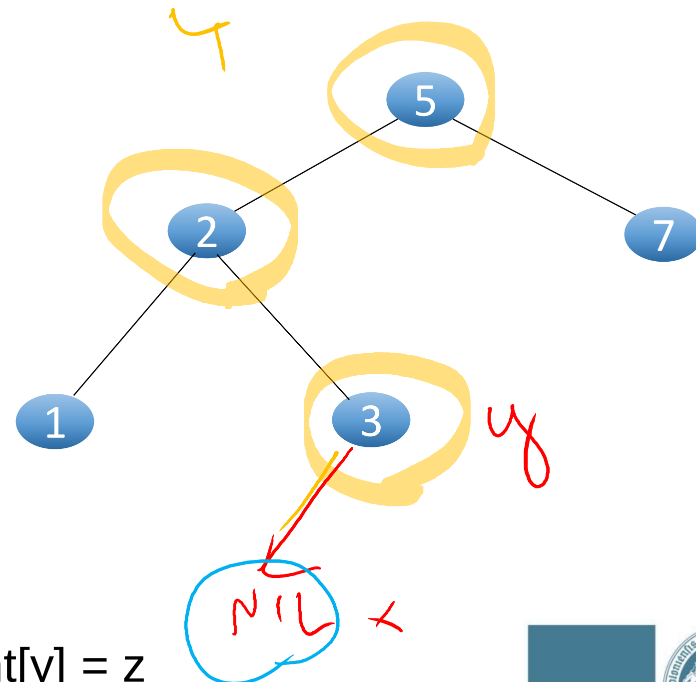
- Ähnlich wie Baumsuche: Finde Blatt, an das neuer Knoten angehängt wird
- Danach wird NIL-Zeiger durch neues Element ersetzt



Datenstrukturen

Einfügen(T,k)

```
z = new node                          \* neues Verbundobjekt für Knoten
key[z] = k; right[z]=NIL; left[z]=NIL
y = NIL; x = root[T]
while x≠NIL do
    y = x
    if k < key[x] then x = left[x]
    else x = right[x]
parent[z] = y
if y=NIL then root[T] = z
else
    if key[z]< key[y] then left[y] = z else right[y] = z
```

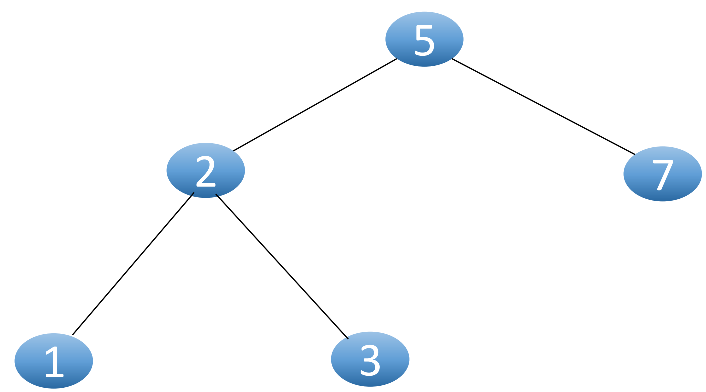


Datenstrukturen

Laufzeit O(h)

Einfügen(T,k)

```
z = new node                                   \* neues Verbundobjekt für Knoten
key[z] = k; right[z]=NIL; left[z]=NIL
y = NIL; x = root[T]
while x≠NIL do
    y = x
    if k < key[x] then x = left[x]
    else x = right[x]
parent[z] = y
if y=NIL then root[T] = z
else
    if key[z]< key[y] then left[y] = z else right[y] = z
```



Datenstrukturen

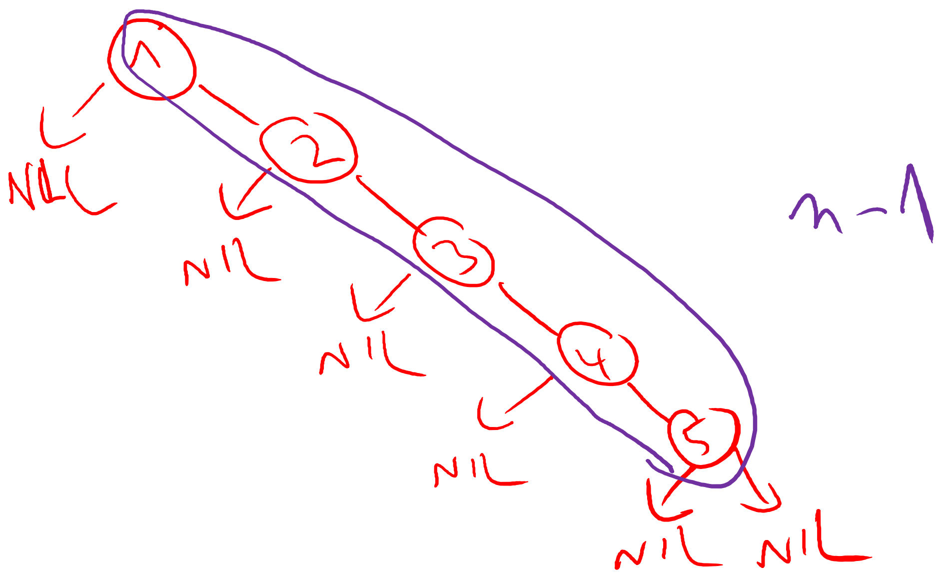
Aufgabe

- Was ist die maximale Höhe des Baums, nachdem n Zahlen nacheinander in einen anfangs leeren binären Suchbaum eingefügt werden?

Datenstrukturen

Aufgabe

- Was ist die maximale Höhe des Baums, nachdem n Zahlen nacheinander in einen anfangs leeren binären Suchbaum eingefügt werden?



Datenstrukturen

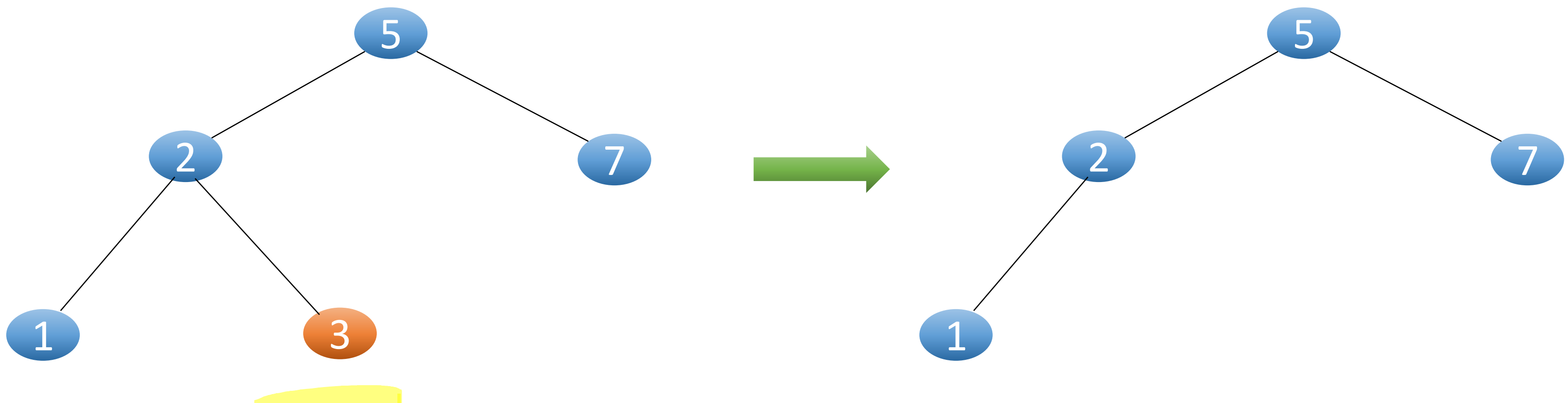
Löschen

- 3 unterschiedliche Fälle
- (a) zu löschender Knoten hat keine Kinder
- (b) zu löschender Knoten hat ein Kind
- (c) zu löschender Knoten hat zwei Kinder

Datenstrukturen

Fall (a)

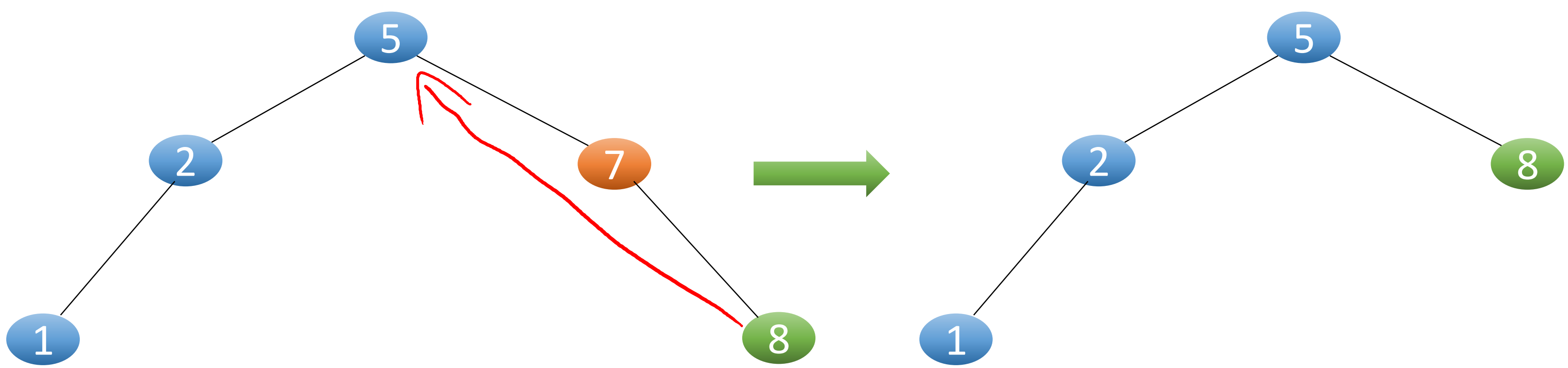
- Zu löschender Knoten hat keine Kinder
- Entferne Knoten aus dem Suchbaum



Datenstrukturen

Fall (b)

- zu löschender Knoten hat ein Kind



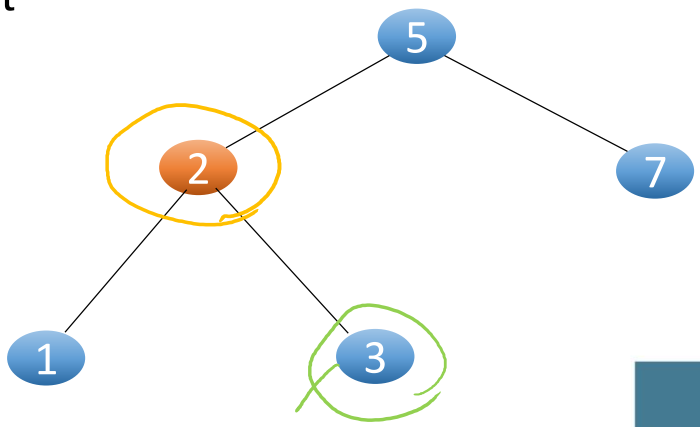
Datenstrukturen

Lemma 16.1

- Sei x ein Knoten in einem binären Suchbaum mit zwei Kindern. Dann hat der Nachfolger von x maximal ein Kind.

Beweis

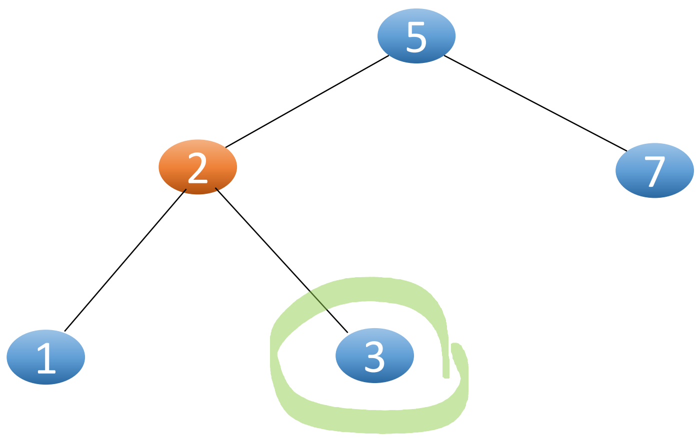
- Der Nachfolger y von x ist das Minimum im rechten Teilbaum von x
- Hätte y ein linkes Kind, so wäre y nicht das Minimum im rechten Teilbaum
- Daher hat y maximal ein Kind



Datenstrukturen

Fall (c)

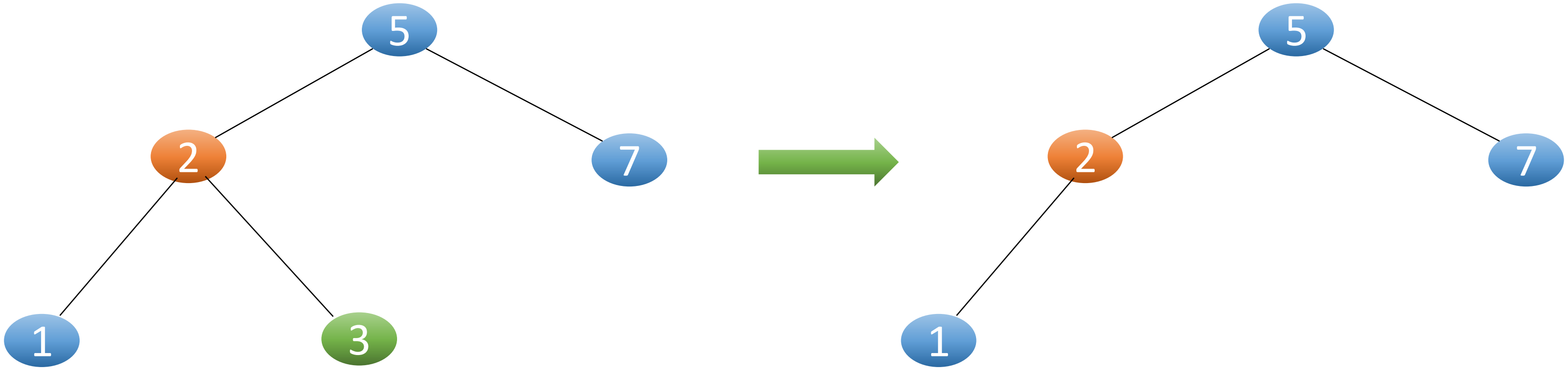
- zu löschender Knoten x hat zwei Kinder
- Schritt 1: Bestimme Nachfolger y von x



Datenstrukturen

Fall (c)

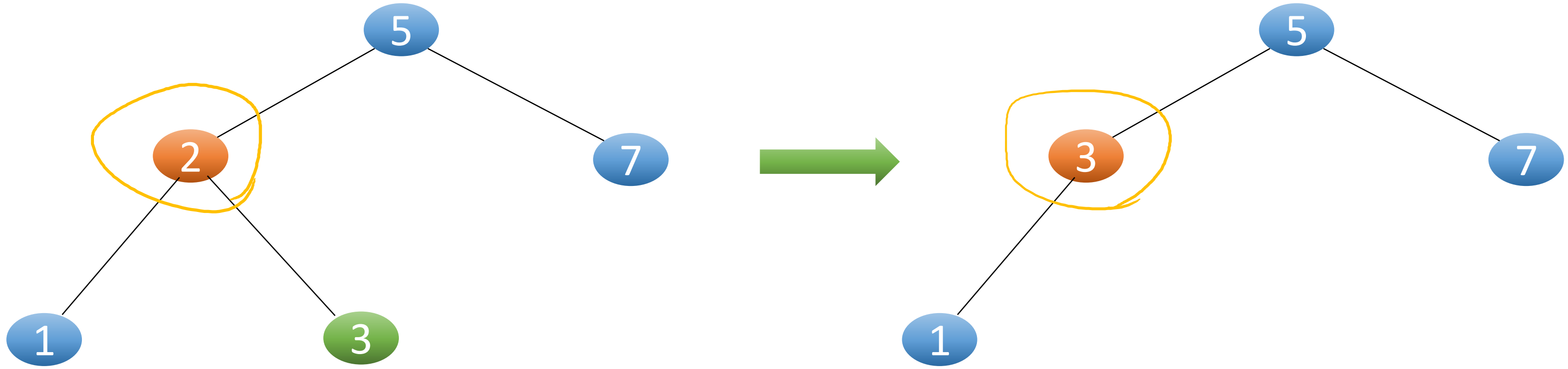
- zu löschender Knoten x hat zwei Kinder
- Schritt 1: Bestimme Nachfolger y von x
- Schritt 2: Lösche y (Fall (a) oder (b))



Datenstrukturen

Fall (c)

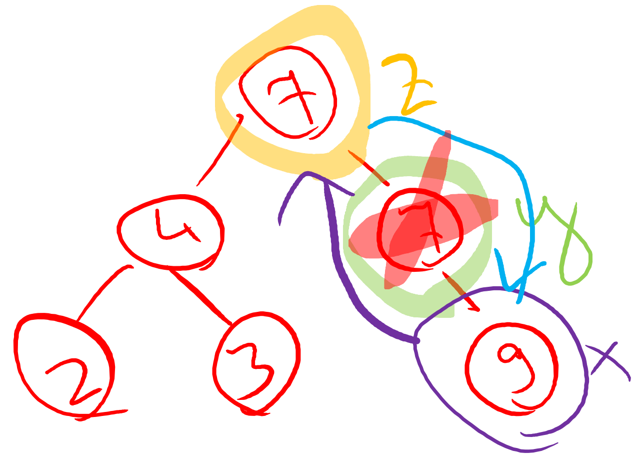
- zu löschender Knoten x hat zwei Kinder
- Schritt 1: Bestimme Nachfolger y von x
- Schritt 2: Lösche y (Fall (a) oder (b))
- Schritt 3: Ersetze key[x] durch key[y]



Datenstrukturen

Löschen(T,z)

1. **if** left[z]=NIL or right[z]=NIL **then** y=z
2. **else** y=NachfolgerSuche(z)
3. **if** left[y]≠NIL **then** x=left[y]
4. **else** x=right[y]
5. **if** x≠NIL **then** parent[x]=parent[y]
6. **if** parent[y]=NIL **then** root[T]=x
7. **else if** y=left[parent[y]] **then** left[parent[y]]=x
8.   **else** right[parent[y]]=x
9. key[z]=key[y]
10. **delete** y



Universität zu Köln

Datenstrukturen

Laufzeit O(h)

Löschen(T,z)

1. **if** left[z]=NIL or right[z]=NIL **then** y=z
2. **else** y=NachfolgerSuche(z)
3. **if** left[y]$\neq$NIL **then** x=left[y]
4. **else** x=right[y]
5. **if** x$\neq$NIL **then** parent[x]=parent[y]
6. **if** parent[y]=NIL **then** root[T]=x
7. **else if** y=left[parent[y]] **then** left[parent[y]]=x
8. **else** right[parent[y]]=x
9. key[z]=key[y]
10. **delete** y

Universität zu Köln

Datenstrukturen

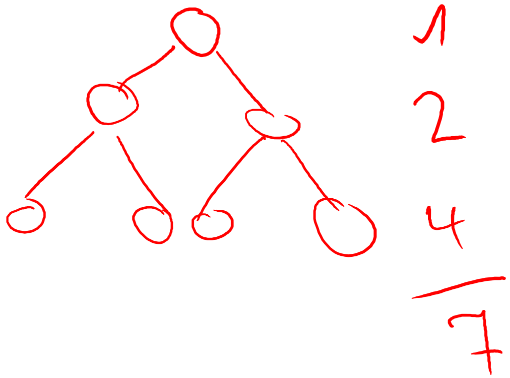
Binäre Suchbäume

- Suchen, Minimum, Nachfolger, etc. in $O(h)$ Zeit
- Einfügen in $O(h)$ Zeit
- Löschen in $O(h)$ Zeit
- h ist Höhe des Baums
- Im schlechtesten Fall ist die Höhe des Suchbaums $\Omega(n)$

Nächstes Ziel

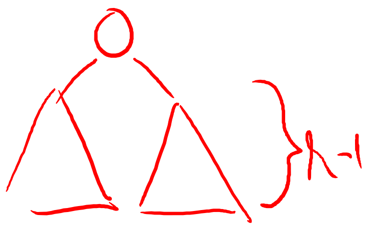
- Verbesserung der Laufzeiten durch „Balanzierung“ des Suchbaums

Datenstrukturen



Lemma 16.2

- Ein Binärbaum der Höhe h hat höchstens 2^{h+1}-1 Knoten.



Beweis

- Induktion über h
- Induktionsanfang: Ein Binärbaum der Höhe 0 hat einen Knoten
- Es gilt $2^1-1 = 1$ ✓
- Induktionsannahme: Die Aussage gilt für Binärbäume der Höhe h-1
- Induktionsschluss: Ein Binärbaum der Höhe h besteht aus einem Wurzelknoten und zwei Binärbäumen der Höhe maximal h-1
- Daher ist die Anzahl seiner Knoten höchstens

$$1+ 2^h-1 + 2^h-1 = 2^{h+1}-1$$ ✓

Universität zu Köln

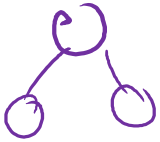
Datenstrukturen

Korollar 16.3

- Ein Binärbaum mit n Knoten hat mindestens Höhe $\lfloor \log n \rfloor$.

Beweis

- Wir wissen aus Lemma 16.2, dass ein Binärbaum der Höhe $\lfloor \log n \rfloor -1$ höchstens $2^{\lfloor \log n \rfloor -1+1} -1<n$ Knoten hat
- Damit muss die Höhe des Baums mindestens $\lfloor \log n \rfloor$ sein



Datenstrukturen

Rot-Schwarz-Bäume

- Balancierter Suchbaum
- Nach Einfügen/Löschen wird die Struktur des Suchbaums so modifiziert, dass eine Höhe von O(log n) garantiert wird
- Rebalancierung nach Einfügen/Löschen wird in O(log n) Zeit möglich sein
- Damit sind Operationen Suchen, Einfügen und Löschen in O(log n) Zeit möglich

Datenstrukturen

Rot-Schwarz-Bäume

- Binäre Suchbäume
- Verbundtyp Knoten enthält color, key, parent, left, right
- NIL-Zeiger werden als Zeiger auf Blätter interpretiert
- Informationen werden nur in den internen Knoten gespeichert
- Erfüllen die Rot-Schwarz-Eigenschaften

Universität zu Köln

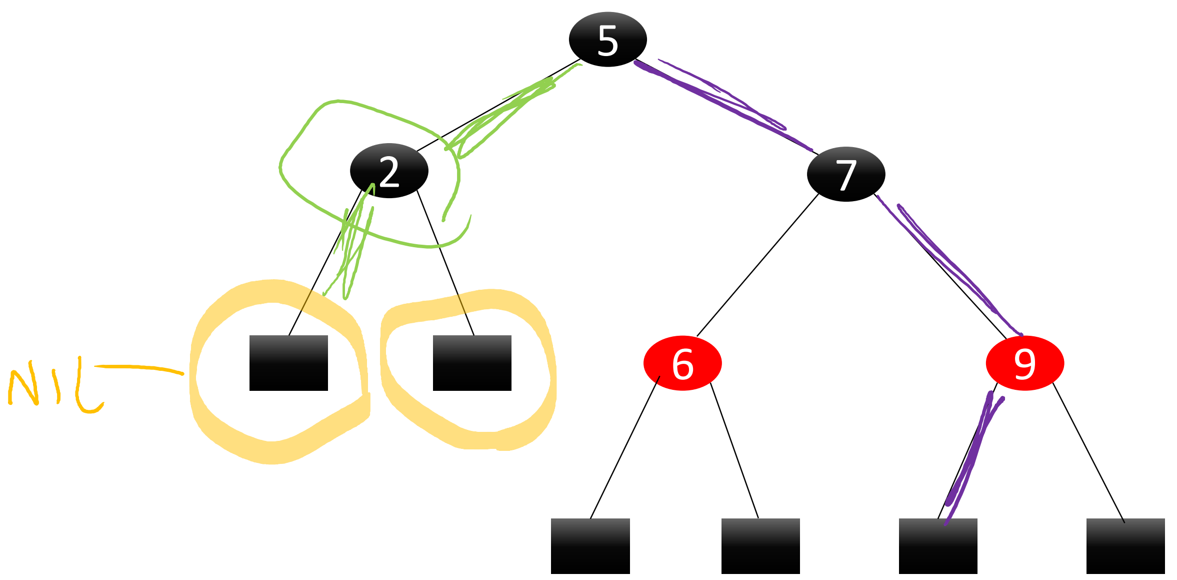
Datenstrukturen

Die Rot-Schwarz-Eigenschaften

- Jeder Knoten ist rot oder schwarz
- Die Wurzel ist schwarz
- Jedes Blatt ist schwarz
- Wenn ein Knoten rot ist, dann sind seine Kinder schwarz
- Für jeden Knoten haben alle Pfade vom Knoten zu den Blättern dieselbe Anzahl schwarzer Knoten

Datenstrukturen

Beispiel Rot-Schwarz-Baum



Zusammenfassung

Zusammenfassung

- Binäre Suchbäume
  - Schlüsselsuche, Minimumsuche, Nachfolgersuche
  - Einfügen
  - Löschen
- Rot-Schwarz-Bäume
  - Einführung

Referenzen

- T. Cormen, C. Leisserson, R. Rivest, C. Stein. Introduction to Algorithms. The MIT press. Second edition, 2001.

Universität zu Köln