

# Kapitel 4.3: Binäre Suchbäume

Professor Dr. Petra Mutzel  
Lehrstuhl für Algorithm Engineering, LS11

10. VO

9. Mai 2006

## Motivation

„Warum soll ich heute hier bleiben?“  
Binäre Suchbäume begegnen Ihnen ständig!

„Warum soll mich das interessieren?“  
Beliebte Klausuraufgaben!

2

## Binäre Suchbäume

- Datenstruktur zur Unterstützung der Operationen des ADT Dictionary:

3

## Operationen des ADT Dictionary

- **INSERT(K k, V v)**
  - Falls k nicht schon in D ist, dann wird ein neuer Schlüssel k mit Wert v in D eingefügt, andernfalls wird der Wert des Schlüssels k auf v geändert.
- **DELETE(K k)**
  - Entfernt Schlüssel k mit Wert aus D (falls k in D)
- **SEARCH(K k): V**
  - Gibt den bei Schlüssel k gespeicherten Wert zurück (falls k in D)

4

## Zusätzliche Operationen der Binären Suchbäume

- **MINIMUM (MAXIMUM)**
  - Sucht den Schlüssel mit kleinstem (größtem) Wert.
- **PREDECESSOR (SUCCESSOR)**
  - Ausgehend von einem gegebenen Element suchen wir das Element mit dem nächstkleineren (nächstgrößeren) Schlüssel.

5

## Überblick

- Bezeichnungen
- Traversierungen für binäre Bäume
- Implementierungen
- Laufzeitanalyse

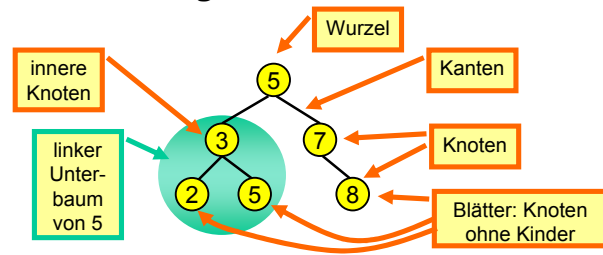
6

## Definitionen

- **Gewurzelter Baum** (rekursiv):
  - entweder leer oder
  - Knoten (Wurzel) mit Verweisen auf mehrere gewurzelte Bäume (Teilbäume)
- **Kind von Knoten  $v$** : Wurzelknoten eines nicht-leeren Teilbaums von  $v$
- **Blatt**: Knoten ohne Kinder

7

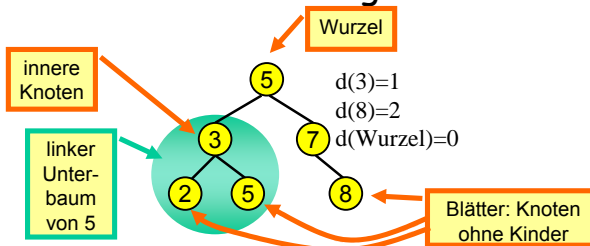
## Bezeichnungen: Gewurzelte Bäume



- Tiefe  $d(v)$ : # der Elter-Knoten bis Wurzel (inkl.)

- 3 ist Wurzel des linken Unterbaums von 5
- 3 ist Elter von 2, 3 ist Elter von 5
- 2 ist linkes Kind von 3, 5 ist rechtes Kind von 3

## Bezeichnungen

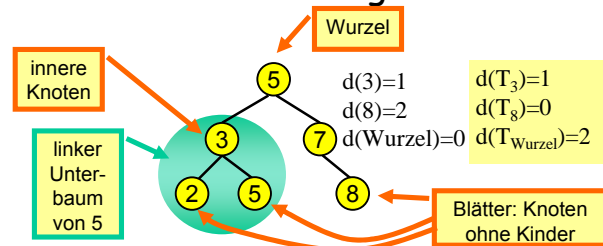


- Tiefe  $d(v)$ : # der Elter-Knoten bis Wurzel (inkl.)

Ebene: Menge aller Knoten mit gleicher Tiefe

9

## Bezeichnungen



Nachfolger( $v$ ): Menge aller Knoten im Unterbaum mit Wurzel  $v$

Höhe  $h(T_r)$  eines (Teil-)baumes  $T_r$  mit Wurzel  $r$ :  
 $\max \{ d(v) : v \text{ ist Knoten in } T_r \}$

10

## Weitere Definitionen

- Ein gewurzelter Baum heißt **geordnet**, wenn die Reihenfolge der Kinder festgelegt ist
- Ein **binärer** gewurzelter Baum ist ein gewurzelter Baum, bei dem jeder Knoten genau zwei Kinder hat.
- Ein **geordneter** gewurzelter binärer Baum unterscheidet also zwischen **linkem und rechtem** Kind bzw. linkem und rechtem Unterbaum.
- Ein binärer Baum heißt **vollständig**, wenn alle Blätter die gleiche Tiefe haben.

11

## Def. Binärer Suchbaum

- Ein binärer Suchbaum ist ein binärer geordneter Baum, dessen Knoten einen Suchschlüssel als Datum enthalten, und der die **Suchbaumeigenschaft** erfüllt:

- **Suchbaumeigenschaft**: Enthält ein Knoten den Schlüssel  $x$ , so sind alle Schlüssel in seinem linken Unterbaum kleiner als  $x$  und alle Schlüssel im rechten Unterbaum größer als  $x$ .

Achtung: alle Schlüssel verschieden

12

## Implementierung binärer Bäume

- Realisierung als verallgemeinerte Listen mit bis zu zwei Nachfolgern:
  - **x.key**: Schlüssel von Knoten x
  - **x.info**: zum Schlüssel zu speichernde Daten
  - **x.parent**: Elter von Knoten x
  - **x.left**: linkes Kind von Knoten x
  - **x.right**: rechtes Kind von Knoten x

- Zugriff auf den Baum erfolgt über seinen Wurzelknoten **root**

13

## Traversierungen für binäre Bäume

- **Inorder-Traversierung**: Durchsuche rekursiv zunächst den linken Unterbaum, dann die Wurzel, durchsuche dann den rechten Unterbaum.
- **Preorder-Traversierung**: Besuche zuerst die Wurzel, durchsuche dann rekursiv den linken Unterbaum, dann den rechten.
- **Postorder-Traversierung**: Durchsuche rekursiv zunächst den linken Unterbaum, durchsuche dann rekursiv den rechten Unterbaum, besuche dann die Wurzel.

14

## Traversierungen für binäre Bäume

- **Level-Order-Traversierung**: Durchsuche zunächst alle Ebenen von links nach rechts, wobei die Ebenen von oben nach unten besucht werden.

- Die wichtigste Traversierungsordnung für binäre Suchbäume ist die **Inorder-Traversierung**, weil dabei die Schlüssel aufsteigend sortiert durchlaufen werden.

15

## Inorder-Traversierung

- (1) Procedure INORDER(p)
- (2) **if** p ≠ NULL {
- (3) INORDER(p.left)
- (4) Ausgabe von p
- (5) INORDER(p.right)
- (6) }

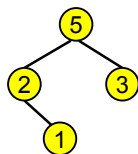
Aufruf: INORDER(root)

Beispiel Inorder-Traversierung (s. Skript)

16

## Beispiel für INORDER-Aufruf

- INORDER(5)
- INORDER(2)
- Ausgabe von „2“
- INORDER(1)
- Ausgabe von „1“
- Ausgabe von „5“
- INORDER(3)
- Ausgabe von „3“



Inorder: 2,1,5,3

17

## Pre order-Traversierung

- (1) Procedure PREORDER(p)
- (2) **if** p ≠ NULL {
- (3) INORDER(p.left)
- (4) PREORDER(p.left)
- (5) PREORDER(p.right)
- (6) }

18

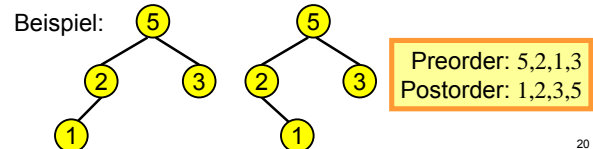
## Post order-Traversierung

- (1) Procedure **POSTORDER(p)**
- (2) **if**  $p \neq \text{NULL}$  {
- (3) **POSTORDER(p.left)**
- (4) **POSTORDER(p.right)**
- (5) **INORDER(p.right)**
- (6) }

19

## Traversierungen für binäre Bäume

- Aus **Inorder-Traversierung** und **Preorder-Traversierungsreihenfolge** kann der binäre Baum eindeutig rekonstruiert werden, wenn er lauter verschiedene Schlüssel enthält.
- Dies gilt nicht, wenn lediglich Preorder- und Postorder-Reihenfolge gegeben sind.



20

## Implementierung von SEARCH(r,s) in binären Suchbäumen

- Finde die Position im Baum mit Wurzel r, an der s gespeichert ist (bzw. sein müßte):
  - Idee: nutze Suchbaumeigenschaft:
1. Vergleiche s mit dem Schlüssel s' an der Wurzel (des Teilbaums)
  2. Falls gefunden: STOP!
  3. Sonst: Falls  $s < s'$ : suche im linken Teilbaum
  4. Sonst: suche im rechten Teilbaum
  5. Gehe zu 1.
  6. Ausgabe: nicht gefunden!

Pseudocode s. Skript bzw. Folien

21

## Implementierung von INSERT(r,q) in binären Suchbäumen

- Einfügen eines Knotens q in den Baum mit Wurzel r
1. Suche nach q.key → Suche endet mit der Position eines leeren Unterbaums
  2. Einfügen von q an diese Position

Pseudocode s. Skript bzw. Folien

22

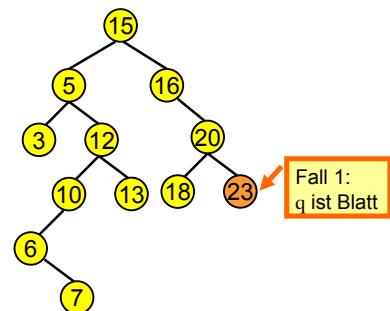
## Implementierung von DELETE(r,q) in binären Suchbäumen

- Entfernt Knoten q im Baum mit Wurzel r
1. Suche nach q.key → Suche endet an Knoten v
  2. Falls  $v \neq \text{NULL}$ , dann
  3. Falls v **keine Kinder** besitzt, dann: streiche v
  4. Falls v **genau ein Kind** hat: ändere 2 Zeiger („herausschneiden“)
  5. Falls v **zwei Kinder** hat, dann hat  $y := \text{Successor}(v)$  kein linkes Kind (warum??): Herausschneiden bzw. Entfernen von y und Ersetzen von q durch y.

Pseudocode s. Skript bzw. Folien

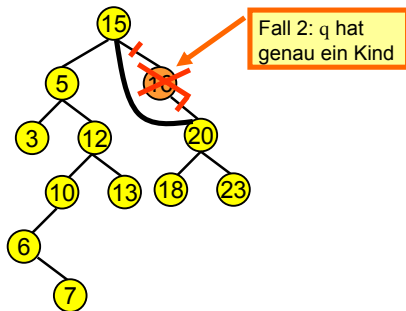
23

## Beispiel für DELETE(r,q)



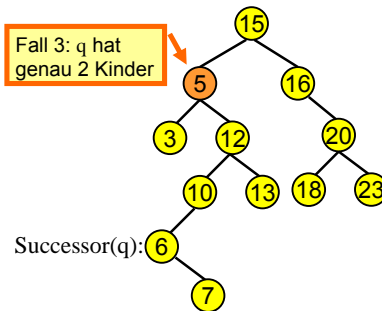
24

## Beispiel für DELETE(r,q)



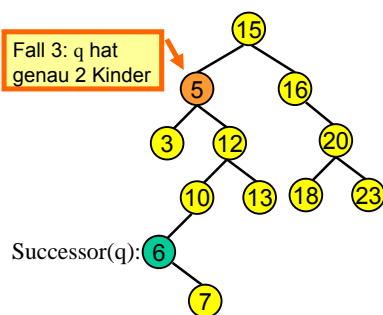
25

## Beispiel für DELETE(r,q)



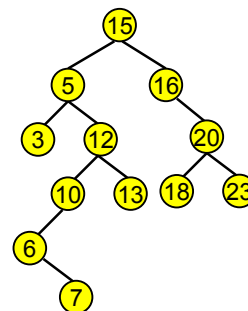
26

## Beispiel für DELETE(r,q)



27

## Successor-Suche



Successor(5) = 6  
Successor(12) = 13  
Successor(13) = 15

28

## Implementierung von SUCCESSOR(r,p)

- Gibt den Nachfolger von Knoten p in der Inorder-Durchmusterungsreihenfolge aus
- Falls p rechtes Kind hat: Return Minimum(p.right)
  - Sonst: Falls p linkes Kind ist: Return(p.parent)
  - Sonst: wandere solange nach oben bis der aktuelle Knoten zum ersten Mal **linkes Kind ist**; dann: Return(p.parent)
  - oder die Wurzel erreicht ist; dann: existiert kein Nachfolger.

Pseudocode s. Skript bzw. Folien

29

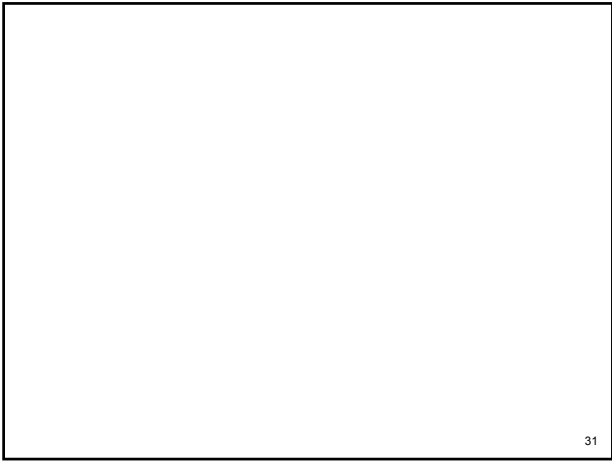
## Analyse der Operationen

- Alle Operationen benötigen eine Laufzeit von  $O(h(T))$  für binäre Suchbäume, wobei  $h(T)$  die Höhe des gegebenen Suchbaumes T ist.

Frage: Wie hoch kann  $h(T)$  für einen binären Suchbaum mit n Knoten sein?

Hausaufgabe bis Donnerstag!

30



31