Grundzüge der Informatik 1

Vorlesung 15

Überblick Vorlesung

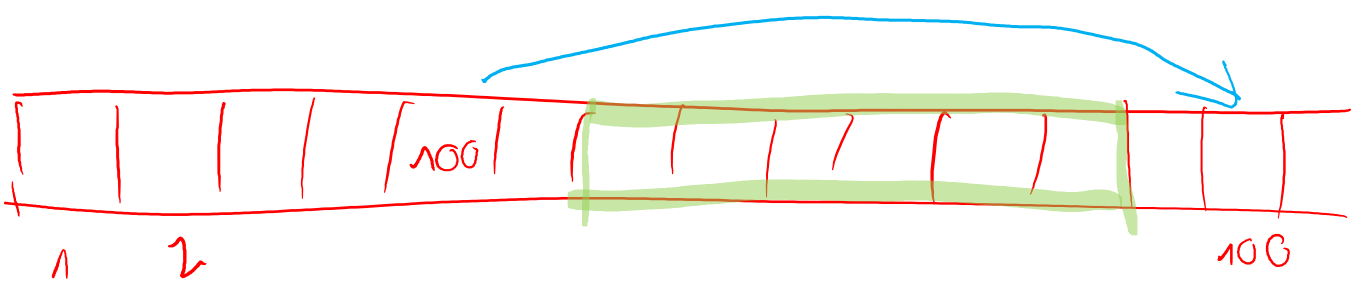
Überblick

- Ein grundlegendes Datenverwaltungsproblem
- Elementare Datenstrukturen und ihre Analyse
- Binärbäume
- Suchbaumeigenschaft

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

Ein grundlegendes Datenverwaltungsproblem

- Speicherung von Datensätzen

Beispiel

- Kundendaten (Name, Adresse, Wohnort, Kundennummer, offene Rechnungen, offene Bestellungen,…)

Anforderungen

- Schneller Zugriff
- Einfügen neuer Datensätze
- Löschen bestehender Datensätze

Datenstrukturen

Zugriff auf Daten

- Jedes Objekt x hat einen Schlüssel key[x]
- Eingabe des Schlüssels liefert Datensatz
- Schlüssel sind vergleichbar (es gibt totale Ordnung der Schlüssel)

Beispiel

- Kundendaten (Name, Adresse, Kundennummer)
- Schlüssel: Name
- Totale Ordnung: Lexikographische Ordnung

Datenstrukturen

Zugriff auf Daten

- Jedes Objekt x hat einen Schlüssel key[x]
- Eingabe des Schlüssels liefert Datensatz
- Schlüssel sind vergleichbar (es gibt totale Ordnung der Schlüssel)

Beispiel

- Kundendaten (Name, Adresse, Kundennummer)
- Schlüssel: Kundennummer
- Totale Ordnung: ‚$\leq$'

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

Vereinfachung

- Schlüssel sind natürliche Zahlen
- Schlüssel sind eindeutig
- Eingabe nur aus Schlüsseln

Analyse von Datenstrukturen

- Platzbedarf in O-Notation
- Laufzeit der Operationen in O-Notation

Datenstrukturen

Aufgabe

- Welche einfachen Datenstrukturen kennen Sie, mit denen man das Datenverwaltungsproblem lösen kann?
- Was sind die Laufzeiten für Suchen, Einfügen und Löschen? (n=|S| bezeichne die Anzahl der Objekte in der Datenstruktur)

Datenstrukturen

Einfaches Feld

- Feld A[1,…,max]
- Integer n, $1 \leq n \leq max$
- n bezeichnet Anzahl Elemente in Datenstruktur

Datenstrukturen

Einfügen(x**)**

```
if n=max then Ausgabe „Fehler: Kein Platz in Datenstruktur“
else
    n = n+1
    A[n] = x
```

3	13	4	1	NIL	NIL	NIL
			n			max

Datenstrukturen

Suchen(x)

```
for i=1 to n do
    if A[i] = x then return i
return nil
```

3	13	4	1	NIL	NIL	NIL
			n			max

Datenstrukturen

Löschen(i) * *i ist Index des zu löschenden Objekts im Feld*

1. A[i] = A[n]
2. A[n] = **nil**
3. n = n-1

3	13	4	1	NIL	NIL	NIL
			n			max

- Löschen(2)

Datenstrukturen

Löschen(i) ** i ist Index des zu löschenden Objekts im Feld*

1. A[i] = A[n]
2. A[n] = **nil**
3. n = n-1

3	**1**	**4**	**1**	**NIL**	**NIL**	**NIL**
			n			max

- Löschen(2)

Datenstrukturen

Löschen(i) * *i ist Index des zu löschenden Objekts im Feld*

1. A[i] = A[n]
2. A[n] = **nil**
3. n = n-1

3	1	4	NIL	NIL	NIL	NIL
			n			max

- Löschen(2)

Datenstrukturen

Löschen(i) * i ist Index des zu löschenden Objekts im Feld

1. A[i] = A[n]
2. A[n] = **nil**
3. n = n-1

3	1	4	NIL	NIL	NIL	NIL
		n				max

- Löschen(2)

Datenstrukturen

Datenstruktur „einfaches Feld“

- Platzbedarf O(max)
- Laufzeit Suchen: O(n)
- Laufzeit Einfügen/Löschen: O(1)

Vorteile

- Schnelles Einfügen und Löschen

Nachteile

- Speicherbedarf abhängig von max (nicht vorhersagbar)
- Hohe Laufzeit für Suchen

Datenstrukturen

Datenstruktur „sortiertes Feld“

- Sortiertes Feld A[1,…,max]
- Integer n, 1≤ n ≤ max
- n bezeichnet Anzahl Elemente in Datenstruktur

2	5	8	12	NIL	NIL	NIL
			n			max

Datenstrukturen

Einfügen(x**)**

1. **if** n=max **then** Ausgabe „Fehler: Kein Platz in Datenstruktur“
2.   n = n + 1
3.   i = n
4.   **while** x < A[i-1] **do**
5.   A[i] = A[i-1]
6.   i = i - 1
7.   A[i] = x

2	**5**	**8**	**12**	**NIL**	**NIL**	**NIL**
			n			max
				n		
				i		

Universität zu Köln

Datenstrukturen

Löschen(i) * *i ist Index des zu löschenden Objekts im Feld*

1. **for** j = i **to** n-1 **do**
2. A[j] = A[j+1]
3. A[n] = nil
4. n = n - 1

2	5	8	12	NIL	NIL	NIL
			n			max

Datenstrukturen

Suchen(x**)**

- Binäre Suche
- Laufzeit O(log n)

2	5	8	12	NIL	NIL	NIL
			n			max

Datenstrukturen

Datenstruktur sortiertes Feld

- Platzbedarf O(max)
- Laufzeit Suche: O(log n)
- Laufzeit Einfügen/Löschen: O(n)

Vorteile

- Schnelles Suchen

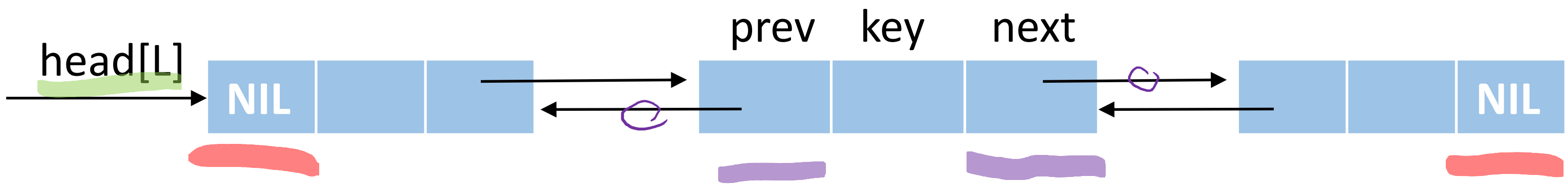
Nachteile

- Speicherbedarf abhängig von max (nicht vorhersagbar)
- Hohe Laufzeit für Einfügen/Löschen

Datenstrukturen

Doppelt verkettete Listen

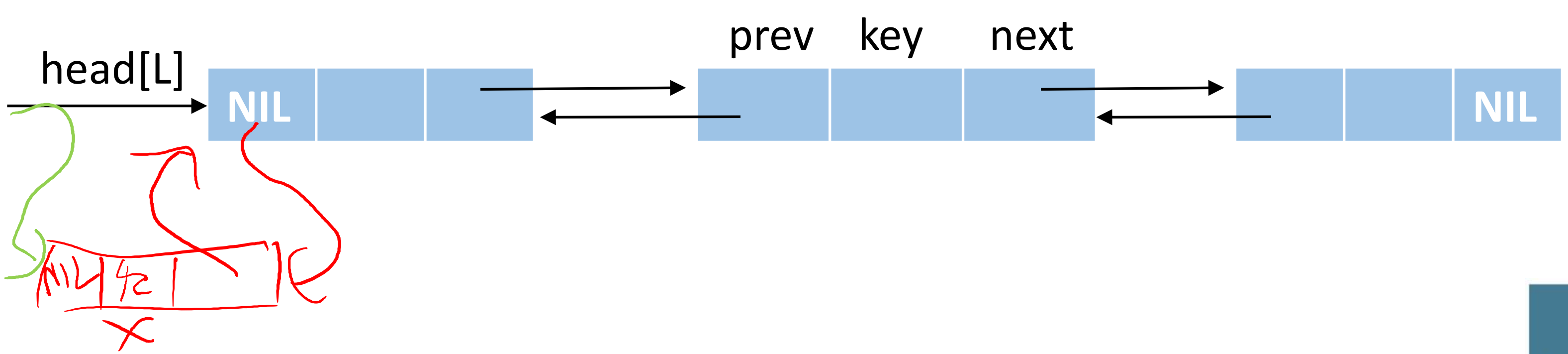
- Listenelement x ist Verbundobjekt bestehend aus Schlüssel key und zwei Zeigern prev und next
- next verweist auf Nachfolger von x
- prev verweist auf Vorgänger von x
- prev/next sind **nil**, wenn Vorgänger/Nachfolger nicht existiert
- head[L] zeigt auf das erste Element



Datenstrukturen

Einfügen(L,k**)**

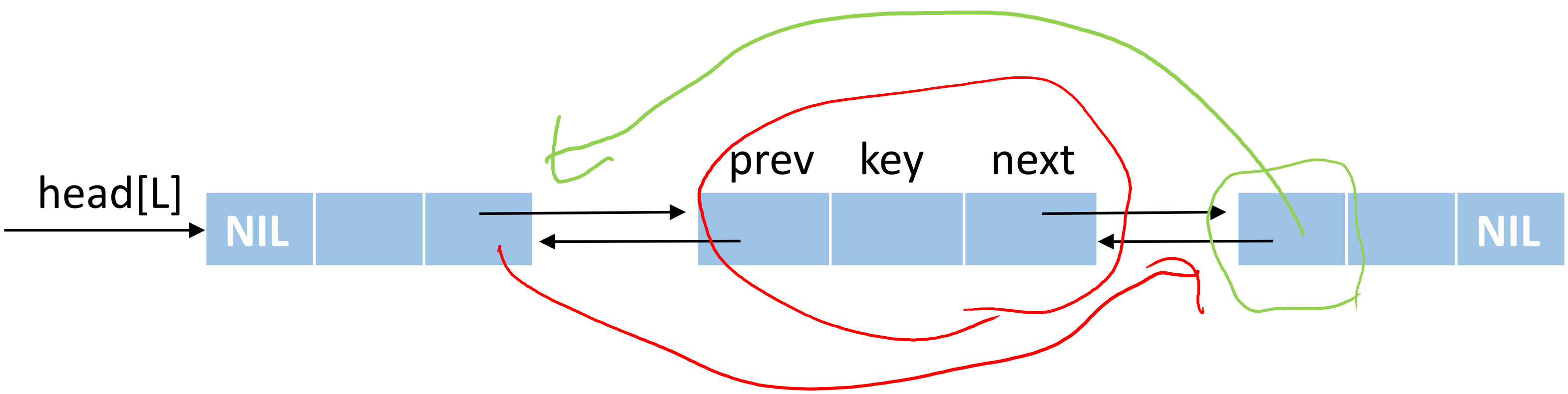
1. x = new Listenelement
2. next[x] = head[L]
3. key[x] = k
4. **if** head[L] ≠ **nil then** prev[head[L]] = x
5. head[L] = x
6. prev[x] = **nil**



Datenstrukturen

Löschen(L,x**)**

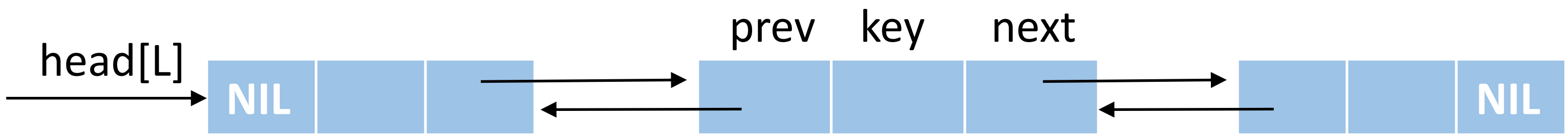
1. **if** prev[x] ≠ **nil then** next[prev[x]] = next[x]
2. **else** head[L] = next[x]
3. **if** next[x] ≠ **nil then** prev[next[x]] = prev[x]
4. **delete** x



Datenstrukturen

Suche(L,k)

1. x = head[L]
2. **while** x≠nil and key[x]≠k **do**
3.   x = next[x]
4. **return** x



Universität zu Köln

Datenstrukturen

Datenstruktur Liste:

- Platzbedarf: O(n)
- Laufzeit Suche: O(n)
- Laufzeit Einfügen/Löschen: O(1)

Vorteile

- Schnelles Einfügen/Löschen
- O(n) Speicherbedarf

Nachteile

- Hohe Laufzeit für Suche

Datenstrukturen

Drei grundlegende Datenstrukturen

- Feld
- sortiertes Feld
- doppelt verkettete Liste

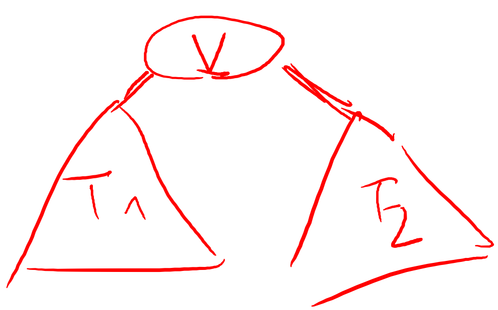
Diskussion

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

Datenstrukturen

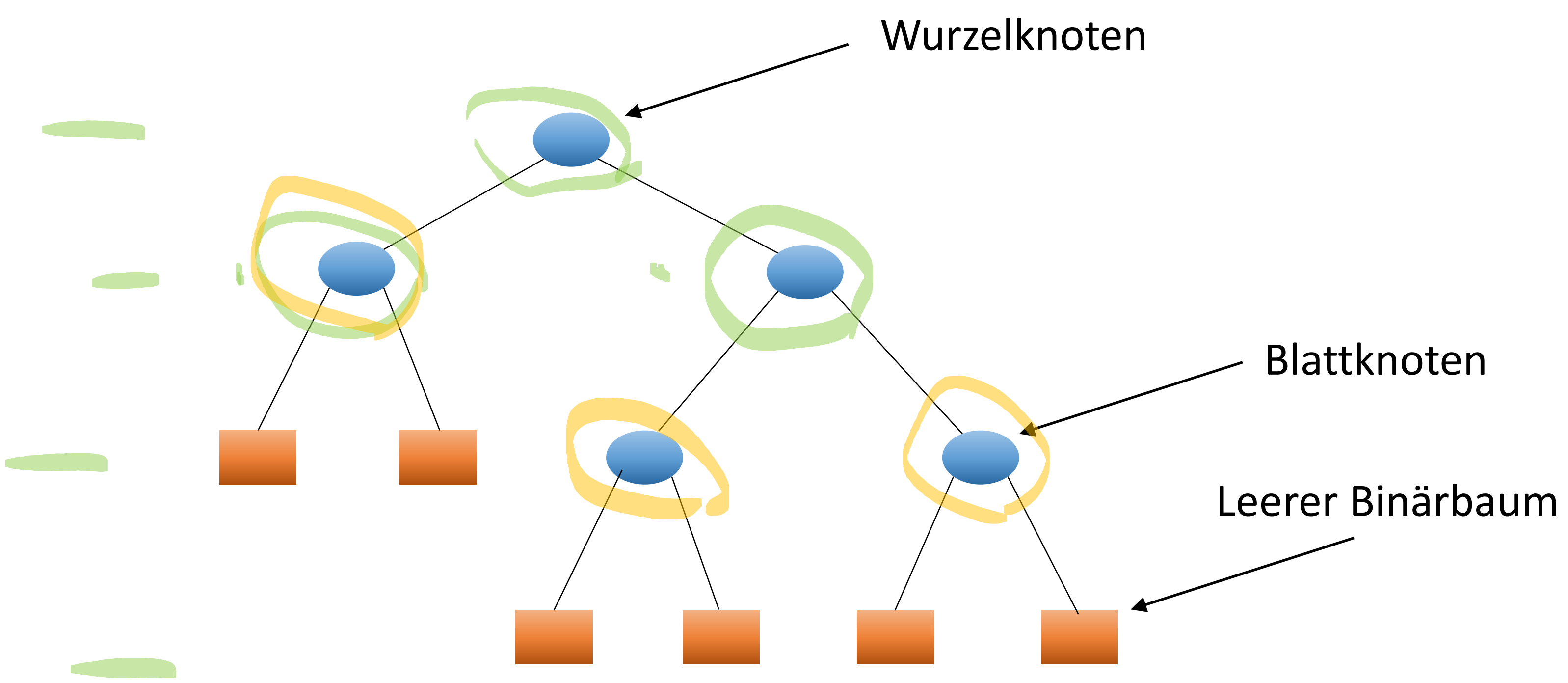
Definition (Binärbaum)

- 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

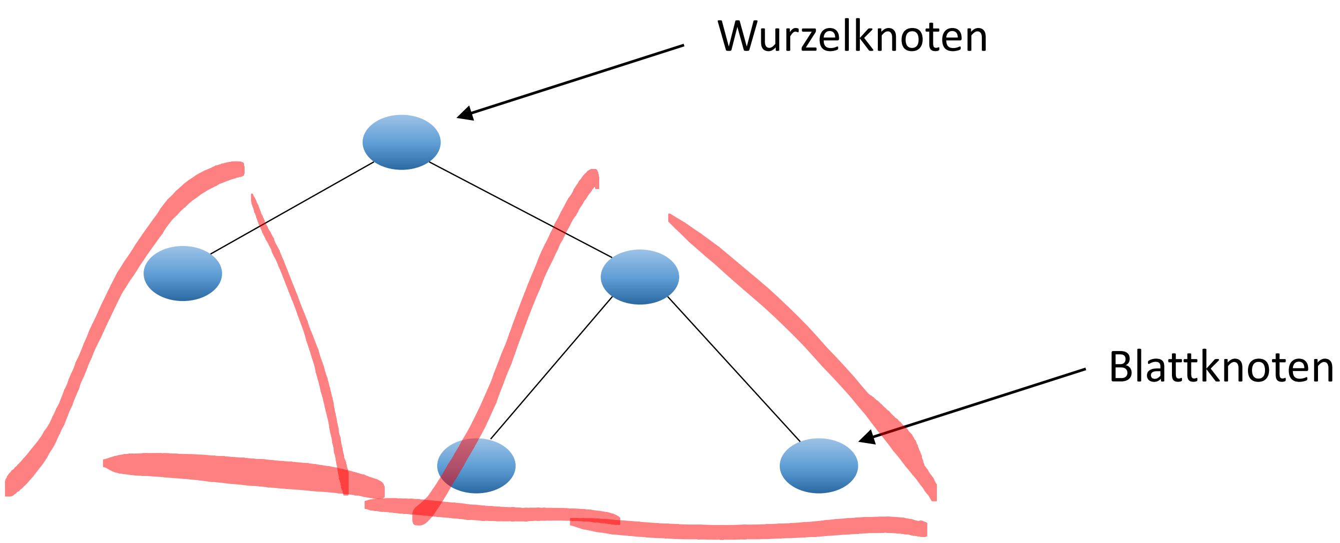
Darstellung von Binärbäumen



Universität zu Köln

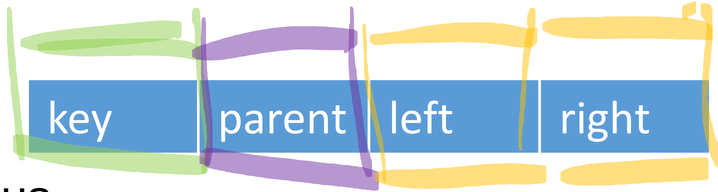
Datenstrukturen

Darstellung von Binärbäumen



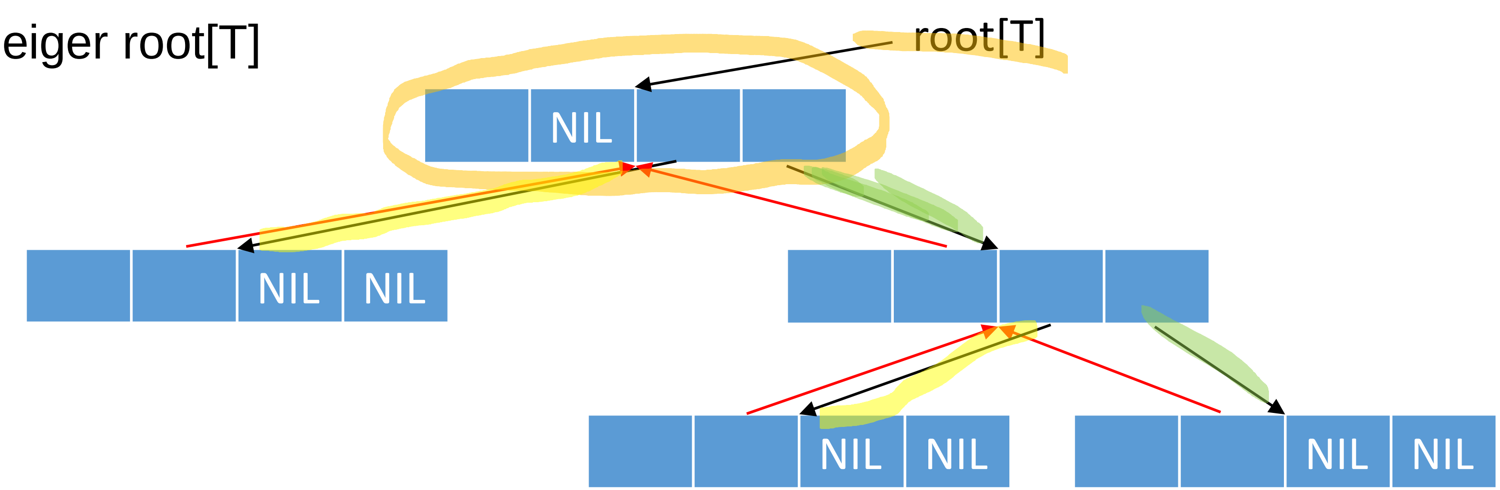
- Häufig lässt man die leeren Bäume in der Darstellung weg

Datenstrukturen



Binärbäume T(Darstellung im Rechner)

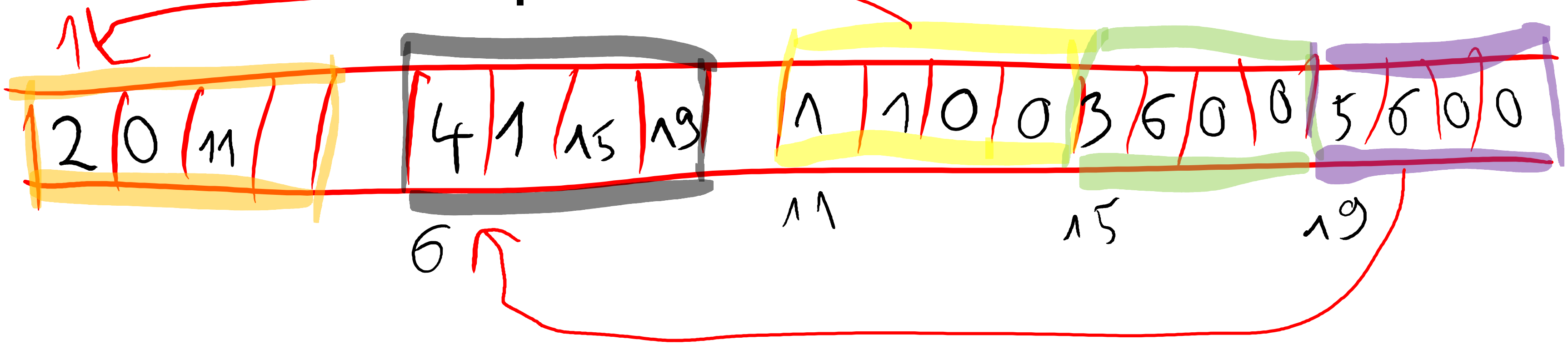
- Ein Knoten v ist ein Verbundobjekt bestehend aus
  - Schlüssel key[v] und ggf. weitere Daten
  - Vaterzeiger parent[v] auf Vater von v
  - Zeiger left[v] und right[v] auf linkes bzw. rechtes Kind von v
- Wurzelzeiger root[T]

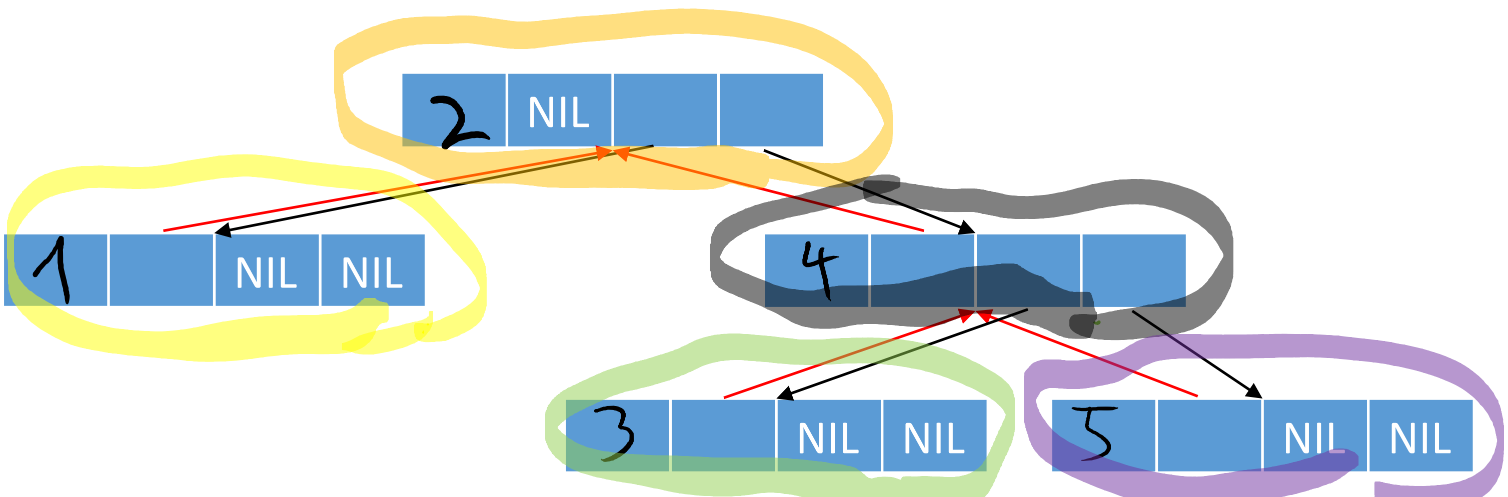


Datenstrukturen

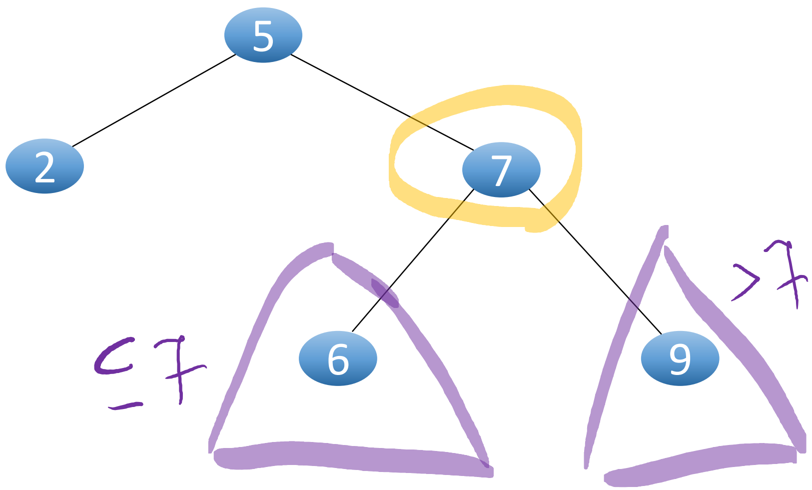
NIL=0

- **Binärbaum im Speicher**





Datenstrukturen



Binäre Suchbäume

- Verwende Binärbaum
- Speichere Schlüssel „geordnet“

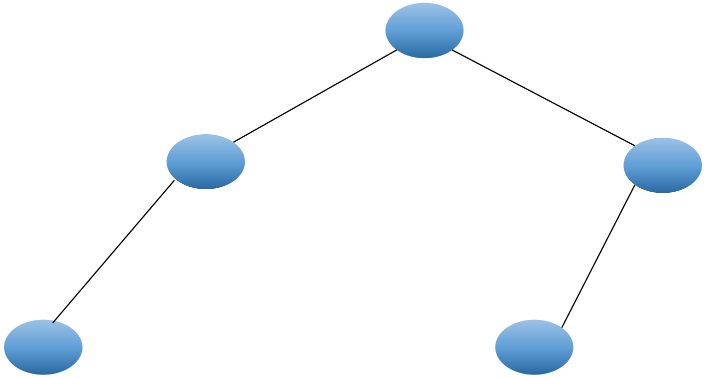
Binäre Suchbaumeigenschaft

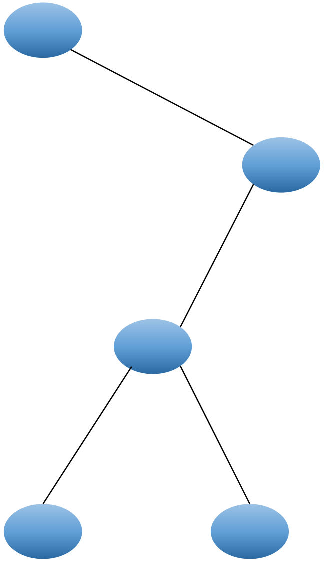
- Sei x Knoten im binären Suchbaum
- Ist y Knoten im linken Unterbaum von x, dann gilt key[y]≤key[x]
- Ist y Knoten im rechten Unterbaum von x, dann gilt key[y]>key[x]

Datenstrukturen

Unterschiedliche Suchbäume

- Schlüsselmenge {2,5,6,7,9}
- Aufgabe: Ordnen Sie die Schlüssel den Knoten der Bäume so zu, dass die Suchbaumeigenschaft erfüllt ist

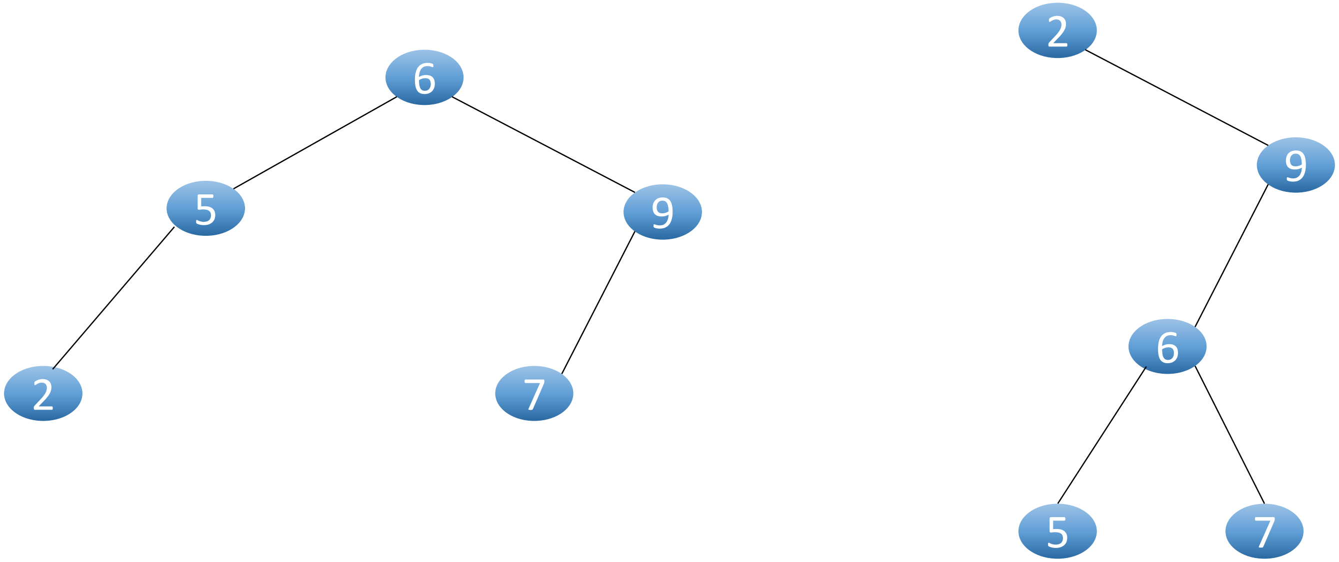




Datenstrukturen

Unterschiedliche Suchbäume

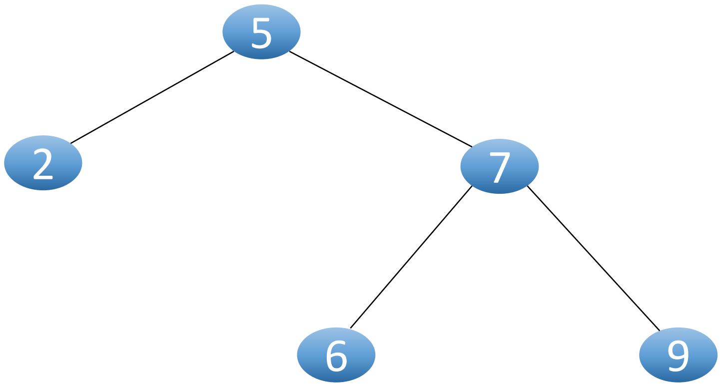
- Schlüsselmenge {2,5,6,7,9}
- Aufgabe: Ordnen Sie die Schlüssel den Knoten der Bäume so zu, dass die Suchbaumeigenschaft erfüllt ist



Datenstrukturen

Ausgabe aller Schlüssel

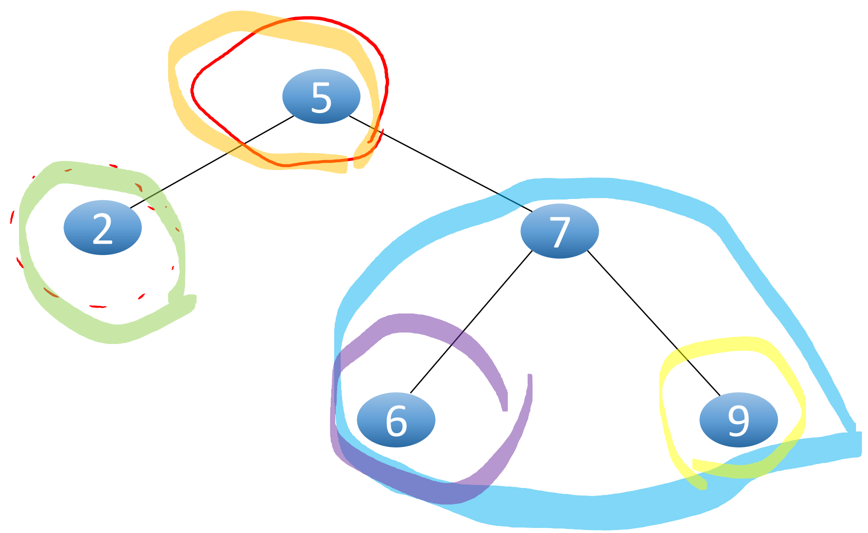
- Gegeben binärer Suchbaum
- Wie kann man alle Schlüssel aufsteigend sortiert in O(n) Zeit ausgeben?



Datenstrukturen

Inorder-Tree-Walk(x)

1. **if** x≠NIL **then**
2. Inorder-Tree-Walk(left[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(right[x])



Universität zu Köln

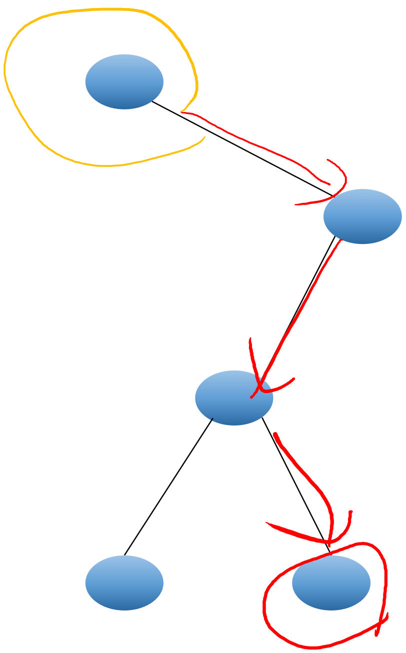
Datenstrukturen

Definition

- Die Höhe eines Binärbaums mit Wurzel v ist die Länge (Anzahl Kanten) des längsten einfachen Weges (keine mehrfach vorkommenden Knoten) von der Wurzel zu einem Blatt.

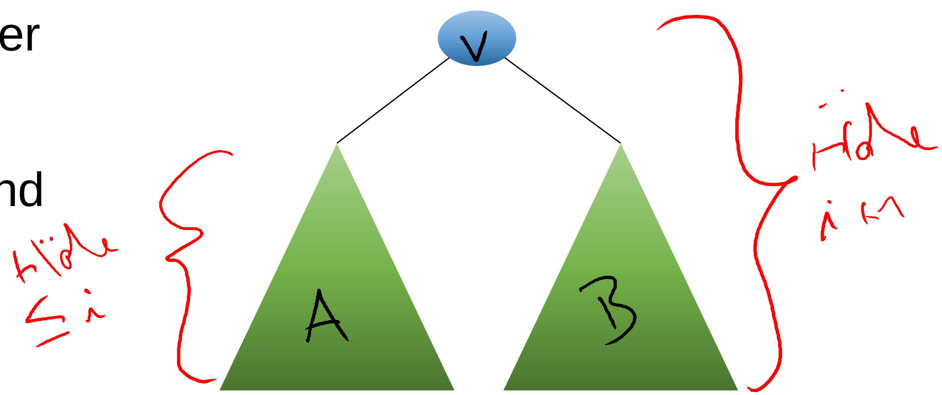
Beispiel

- Baum der Höhe 3



Induktion über die Höhe von Binärbäumen

- Wir wollen Aussage A(i) durch Induktion über die Höhe von Bäumen zeigen
- (a) Zeige die Aussage für leere Bäume (Annahme: leere Bäume haben Höhe -1)
- (b) Zeige: Gilt die Aussage für Bäume der Höhe i, so gilt sie auch für Bäume der Höhe i+1
- Dabei können wir immer annehmen, dass ein Baum der Höhe i+1 aus einer Wurzel v und zwei Teilbäumen A,B besteht, so dass
  (1) A und B Höhe maximal i haben und
  (2) A oder B Höhe i hat



Datenstrukturen

```
Inorder-Tree-Walk(x)
1.  if x≠NIL then
2.      Inorder-Tree-Walk(left[x])
3.      Ausgabe key[x]
4.      Inorder-Tree-Walk(right[x])
```

Lemma

- Inorder-Tree-Walk gibt die Schlüssel eines binären Suchbaums in aufsteigender Reihenfolge aus.

Beweis

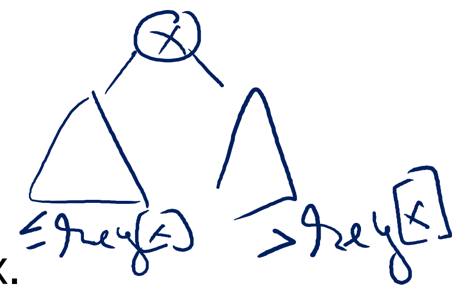
- Induktionsanfang (leerer Suchbaum):
- Für einen leeren Baum wird nichts ausgegeben. Dies ist korrekt. ✓
- Induktionsannahme: Das Lemma gilt für Suchbäume der Höhe i.
- Induktionsschluss:
- z.z.: Lemma gilt auch für Suchbäume der Höhe $i+1\geq 0$.

Datenstrukturen

```
Inorder-Tree-Walk(x)
1.  if x≠NIL then
2.      Inorder-Tree-Walk(left[x])
3.      Ausgabe key[x]
4.      Inorder-Tree-Walk(right[x])
```

Lemma

- Inorder-Tree-Walk gibt die Schlüssel eines binären Suchbaums in aufsteigender Reihenfolge aus.



Beweis

- Betrachte Inorder-Tree-Walk auf solchem Baum mit Wurzel x.
- Nach Suchbaumeigenschaft sind alle Schlüssel im linken Teilbaum kleiner oder gleich Schlüssel von x
- Zeile 2: Aufruf für linken Teilbaum der Höhe ≤i. Nach Induktionsannahme werden Schlüssel in aufsteigender Reihenfolge ausgegeben
- Zeile 3: key[x] wird ausgegeben

Datenstrukturen

```
Inorder-Tree-Walk(x)
1.  if x≠NIL then
2.      Inorder-Tree-Walk(left[x])
3.      Ausgabe key[x]
4.      Inorder-Tree-Walk(right[x])
```

Lemma

- Inorder-Tree-Walk gibt die Schlüssel eines binären Suchbaums in aufsteigender Reihenfolge aus.

Beweis

- Alle Schlüssel im rechten Teilbaum sind größer als Schlüssel von x (Suchbaumeigenschaft)
- Zeile 4: Aufruf für rechten Teilbaum der Höhe ≤i. Nach Induktionsannahme: Schlüssel aus rechtem Teilbaum werden in aufsteigender Reihenfolge ausgegeben
- Insgesamt:
- Schlüssel aus linkem Teilbaum aufsteigend, Schlüssel von x, Schlüssel aus rechtem Teilbaum aufsteigend
- Nach Suchbaumeigenschaft ist dies aufsteigend sortierte Folge

Überblick gesamte Vorlesung

Zusammenfassung

- Ein grundlegendes Datenverwaltungsproblem
- Elementare Datenstrukturen und ihre Analyse
  - Einfaches Feld
  - Sortiertes Feld
  - Doppelt verkettete Liste
- Binärbäume
- Suchbaumeigenschaft

Referenzen

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

Universität zu Köln