# Algorithmen und Datenstrukturen 1

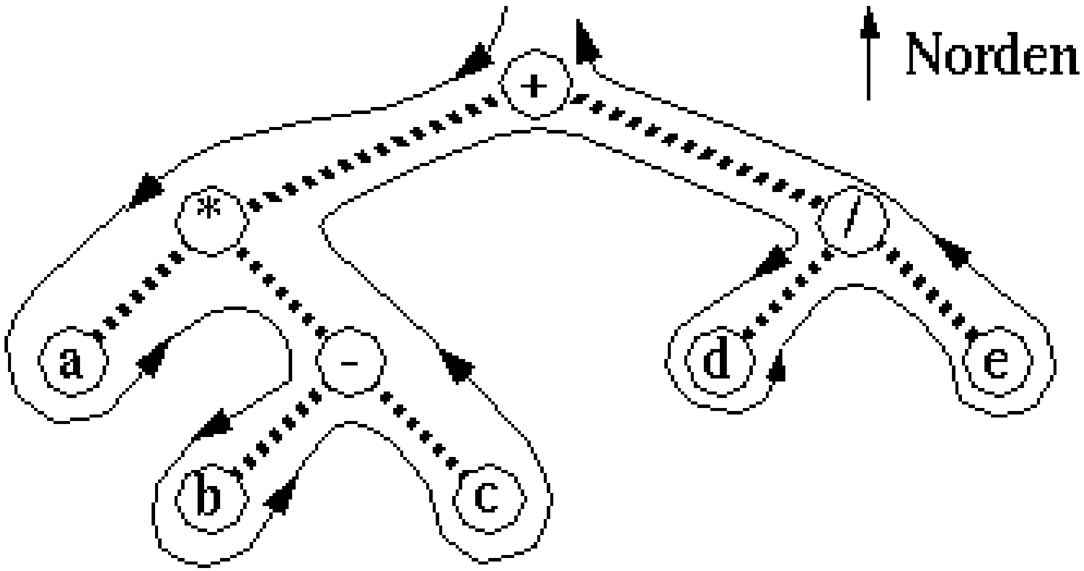
## 8. Vorlesung

*Peter F. Stadler*

Universität Leipzig
Institut für Informatik
studla@*bioinf.uni-leipzig.de*

# Wdhlg.: Durchlaufmöglichkeiten

**Anschauliche Darstellung**




Ausgabe bei Passage des Turms (Knoten)

WLR: in Richtung Süden

LWR: an seiner Südseite

LRW: in Richtung Norden

# Gefädelte Binärbäume I

Weitere Verbesserung von iterativen Durchlaufalgorithmen

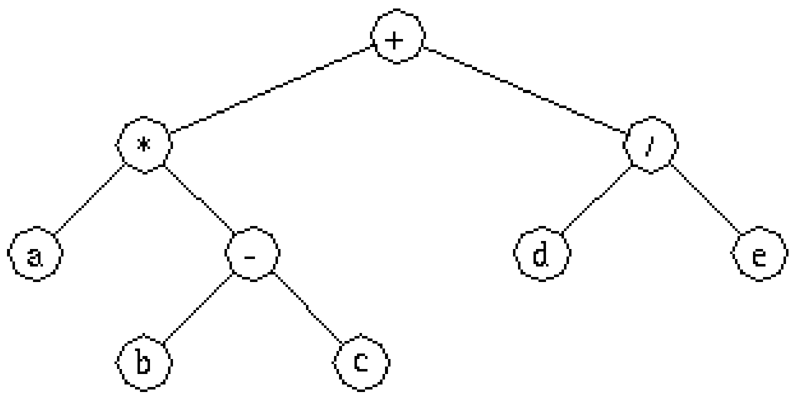
Methode benutzt einen "Faden", der die Baumknoten in der Folge der Durchlaufordnung verknüpft.

Zwei Typen von Fäden:

- *Rechtsfaden* verbindet jeden Knoten mit seinem Nachfolgerknoten in Durchlaufordnung
- *Linksfaden* stellt Verbindung zum Vorgängerknoten in Durchlaufordnung her

Lösung 1. Explizite Speicherung von 2 Fäden

Beispiel: Zwischenordnung



# Gefädelte Binärbäume II

Lösung 2. Vermeidung von Redundanz

Eine zweite Art der Fädelung kommt ohne zusätzliche Zeiger aus und erfordert daher geringeren Speicherplatzaufwand. Die Wartungs- und Durchlauf-Algorithmen werden lediglich geringfügig komplexer.

Beobachtung 1: Binärbaum mit n Knoten hat n+1 freie Zeiger (null)

Beobachtung 2: für die Zwischenordnung können Fadenzeiger in inneren Knoten durch Folgen von Baumzeigern ersetzt werden

Idee: Benutze freie Zeiger und Baumzeiger für Fädelung

- pro Knoten zusätzliche Variablen Lfaden, Rfaden statt Lchild, Rchild
- zeigen auf linken bzw. rechten Nachbarn in Durchlaufreihenfolge.
- Achtung: Normale Baumzeiger müssen von Fädelzeigern unterschieden werden.

# Gefädelte Binärbäume III

## Algorithmus für die Traversierung

```
Start bei Wurzelknoten
Schleife bis der Knoten rechts außen erreicht ist:
   Solange wie möglich nach links verzweigen
   Knoten ausgeben
   Falls Knoten Rfaden hat:
      Rfaden einen Schritt folgen
      Knoten ausgeben
   Sonst falls Knoten rechten Sohn hat:
      einen Schritt nach rechts verzweigen
```

# Zusammenfassung

**Definitionen**

- Baum, orientierter Baum (Wurzel-Baum), geordneter Baum, Binärbaum
- vollständiger, fast vollständiger, strikter, ausgeglichener, ähnlicher, äquivalenter Binärbaum
- Höhe, Grad, Stufe / Pfadlänge, Gewicht

**Speicherung von Binärbäumen**

- verkettete Speicherung
- Feldbaum-Realisierung
- sequentielle Speicherung

**Baum-Traversierung**

- Preorder (WLR): Vorordnung
- Inorder (LWR): Zwischenordnung
- Postorder (LRW): Nachordnung

**Gefädelte Binärbäume**: Unterstützung der (iterativen) Baum-Traversierung durch Links/Rechts-Zeiger auf Vorgänger/Nachfolger in Traversierungsreihenfolge

# 6. Binäre Suchbäume

**Natürliche binäre Suchbäume**

- Begriffe und Definitionen
- Grundoperationen: Einfügen, sequentielle Suche, direkte Suche, Löschen
- Bestimmung der mittleren Zugriffskosten

**Balancierte Binärbäume**

**AVL-Baum**

- Einfügen mit Rotationstypen
- Löschen mit Rotationstypen
- Höhe von AVL-Bäumen

**Gewichtsbalancierte Binärbäume**

**Positionssuche mit balancierten Bäumen (Lösung des Auswahlproblems)**

# Binäre Suchbäume

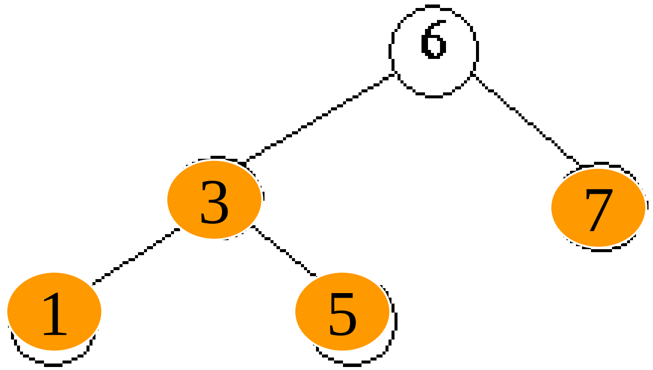
Def.: Ein natürlicher binärer Suchbaum B ist ein Binärbaum; Er ist entweder leer oder jeder Knoten in B enthält einen Schlüssel und:

(1) alle Schlüssel im linken Unterbaum von B sind kleiner als der Schlüssel in der Wurzel von B

(2) alle Schlüssel im rechten Unterbaum von B sind größer als der Schlüssel in der Wurzel von B

(3) die linken und rechten Unterbäume von B sind auch binäre Suchbäume.

Beispiel: Binärbaum aus 1, 3, 5, 6, 7



# 4 Grundoperationen

- Einfügen
- direkte Suche
- sequentielles Durchlaufen
- Löschen

# Suche in Binärbäumen

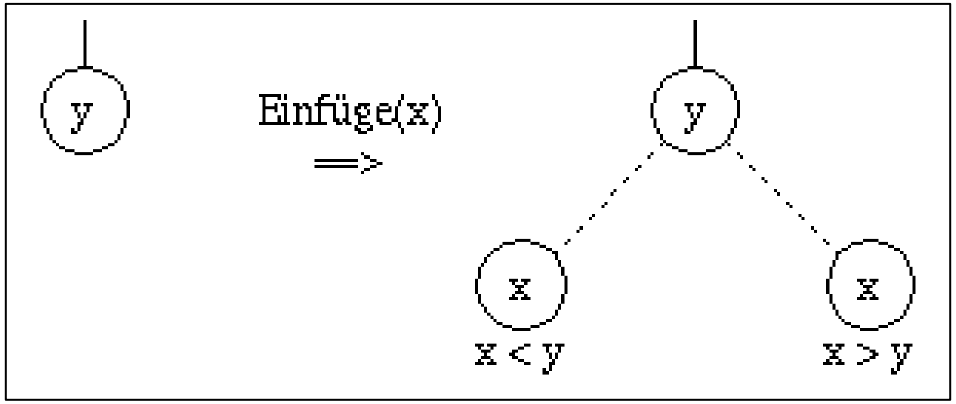
## Direkte Suche:

- Die Suche nach einem Schlüssel x in einem Baum (Teilbaum) läuft nach folgendem rekursiven Schema ab:
- Man inspiziere den Wurzelknoten des Baumes.
- Falls x = Schlüssel des inspizierten Knotens: Suche beendet. Sonst:
- Falls x < Schlüssel des inspizierten Knotens: Setze Suche im linken Teilbaum fort.
- Falls x > Schlüssel des inspizierten Knotens: Setze Suche im rechten Teilbaum fort.
- Maximale Anzahl inspizierter Knoten: Tiefe des Baumes.

## Sequentielle Suche:

- Einsatz eines Durchlauf-Algorithmus (Zwischenordnung)

# Einfügen in binären Suchbäumen




Neue Knoten werden immer als
Blätter eingefügt

Suche der Einfügeposition:

Aussehen des Baumes wird durch die
Folge der Einfügungen bestimmt:
reihenfolgeabhängige Struktur

n Schlüssel erlauben n! verschiedene Schlüsselfolgen

# Beispiel Einfügereihenfolge I

**Einfügereihenfolge 1:**

KANT, LEIBNIZ, HEGEL, HUME, LOCKE, SOCRATES, SPINOZA, DESCARTES, CARNAP, FREGE, PLATON

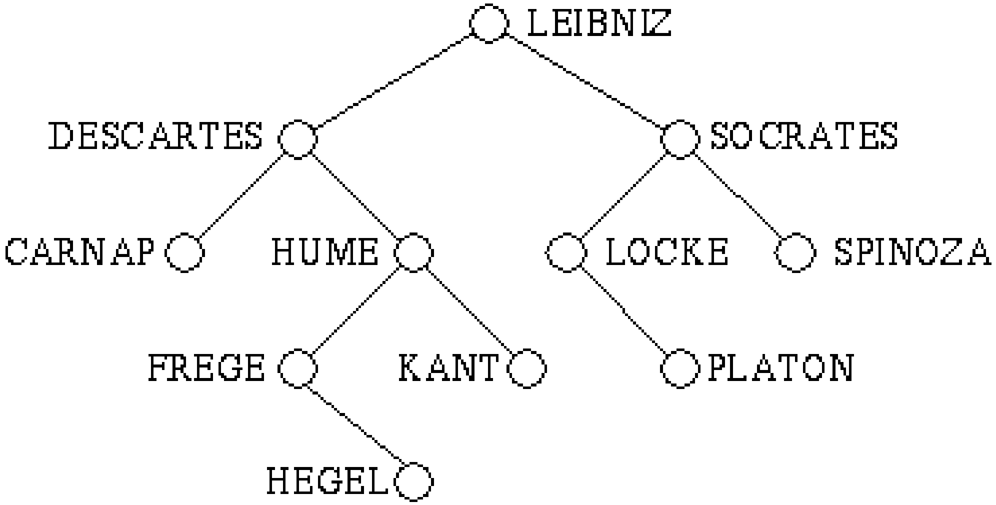
# Beispiel Einfügereihenfolge II

**Einfügereihenfolge 1:**

KANT, LEIBNIZ, HEGEL, HUME, LOCKE, SOCRATES, SPINOZA, DESCARTES, CARNAP, FREGE, PLATON

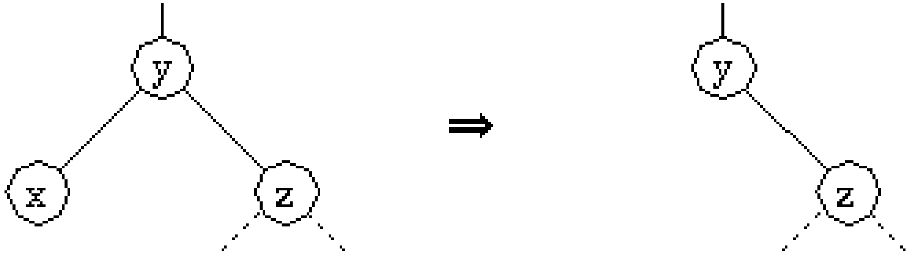

**Einfügereihenfolge 2:**

LEIBNIZ, DESCARTES, CARNAP, HUME, SOCRATES, FREGE, LOCKE, KANT, HEGEL, PLATON, SPINOZA

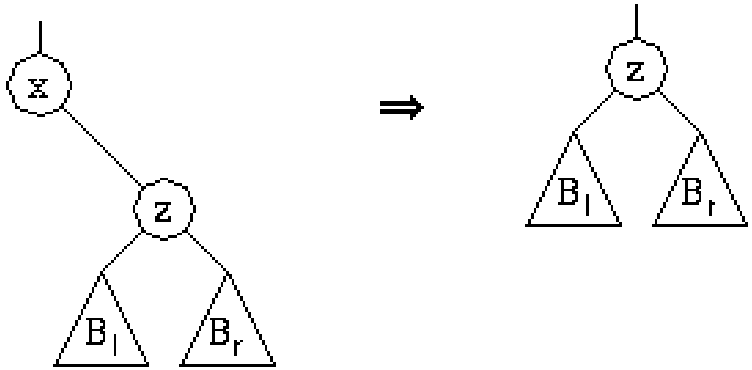


# Löschen in binären Suchbäumen

**Fall 1: x ist Blatt**

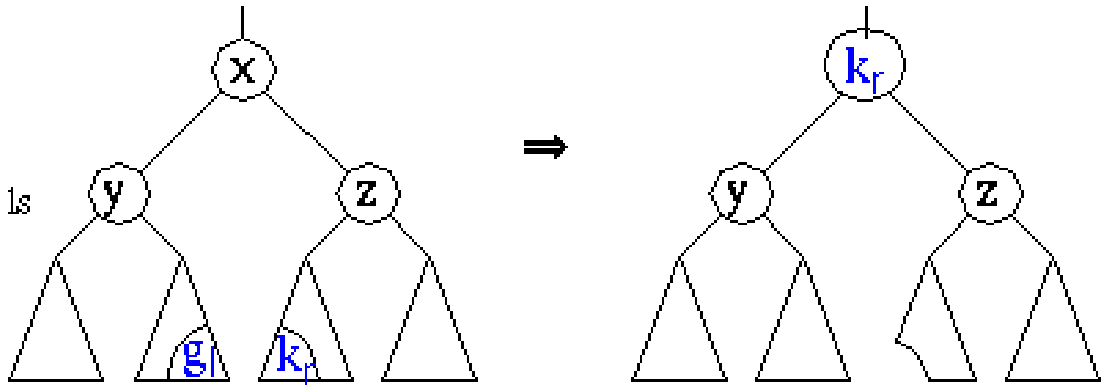



**Fall 2/3: x hat leeren linken/rechten Unterbaum**

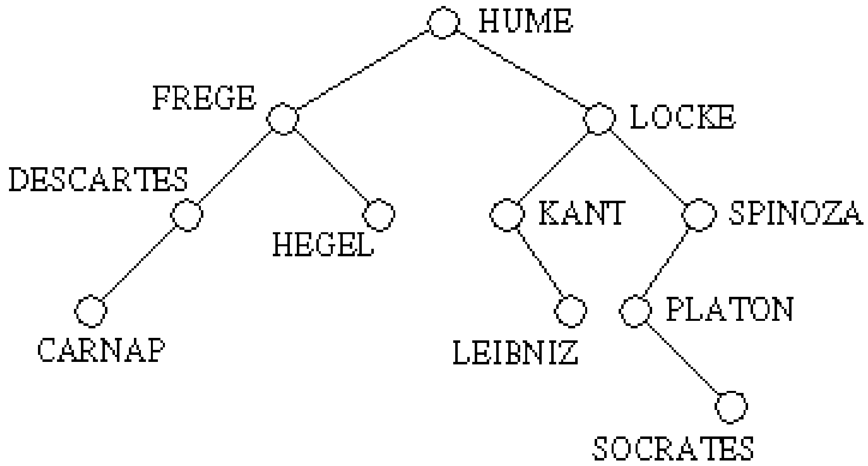



**Fall 4: x hat zwei nicht-leere Unterbäume**

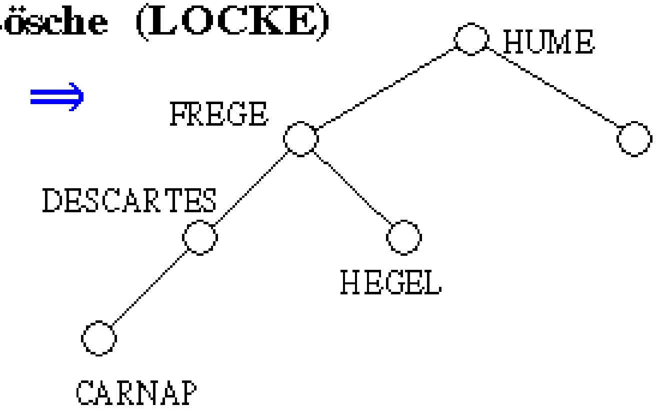
Heranziehen des größten Schlüssels im linken Unterbaum ($g_l$) oder des kleinsten Schlüssels im rechten Unterbaum ($k_r$)



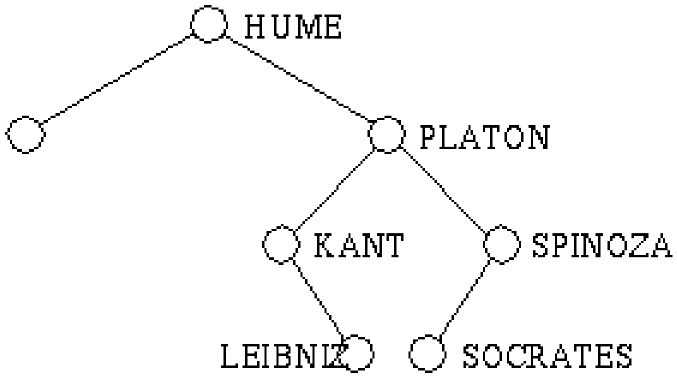
# Beispiele: Löschen



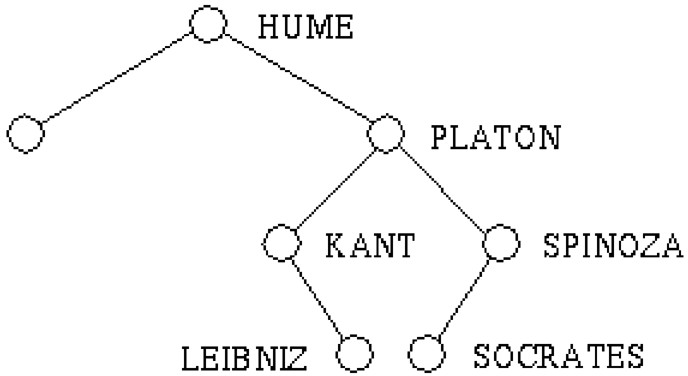

**Lösche (LOCKE)**




**Lösche (DESCARTES)**




**Lösche (FREGE)**



# Markieren statt Löschen

Alternative: Jeder zu löschende Knoten wird speziell markiert; bei Such- und Einfügevorgängen wird er gesondert behandelt.

Vorgehen:

- Bei der Suche den gelöschten Knoten nur zur Entscheidung benutzen, ob links oder rechts weitergesucht wird.
- Bei erneutem Einfügen den Wert neu als Blatt einfügen oder Löschmarkierung entfernen.

Achtung: Beim Löschen wird kein Speicher frei!

# Binäre Suchbäume: Zugriffskosten

Kostenmaß: Anzahl der aufgesuchten Knoten bzw. Anzahl der benötigten Suchschritte oder Schlüsselvergleiche.

Kosten der Grundoperationen für

- sequentielle Suche
- Einfügen, Löschen, direkte Suche

Bestimmung der mittleren Zugriffskosten $\bar{z}$ (direkte Suchkosten)

- Zunächst Berechnung seiner gesamten Pfadlänge PL als Summe der Längen der Pfade von der Wurzel bis zu jedem Knoten $K_i$.

$$PL(B) = \sum_{i=1}^{n} \text{Stufe}(K_i)$$

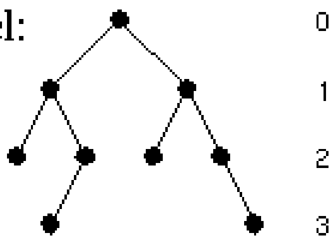
- Mit $n_i$ = Zahl der Knoten auf Stufe i gilt

$$PL(B) = \sum_{i=0}^{h-1} i \cdot n_i \quad \text{und} \quad \sum_{i=0}^{h-1} n_i = n = \text{gesamte Knotenzahl}$$

- Die mittlere Pfadlänge ergibt sich zu $p = PL / n$
- Da bei jedem Zugriff noch auf die Wurzel zugegriffen werden muß, erhält man

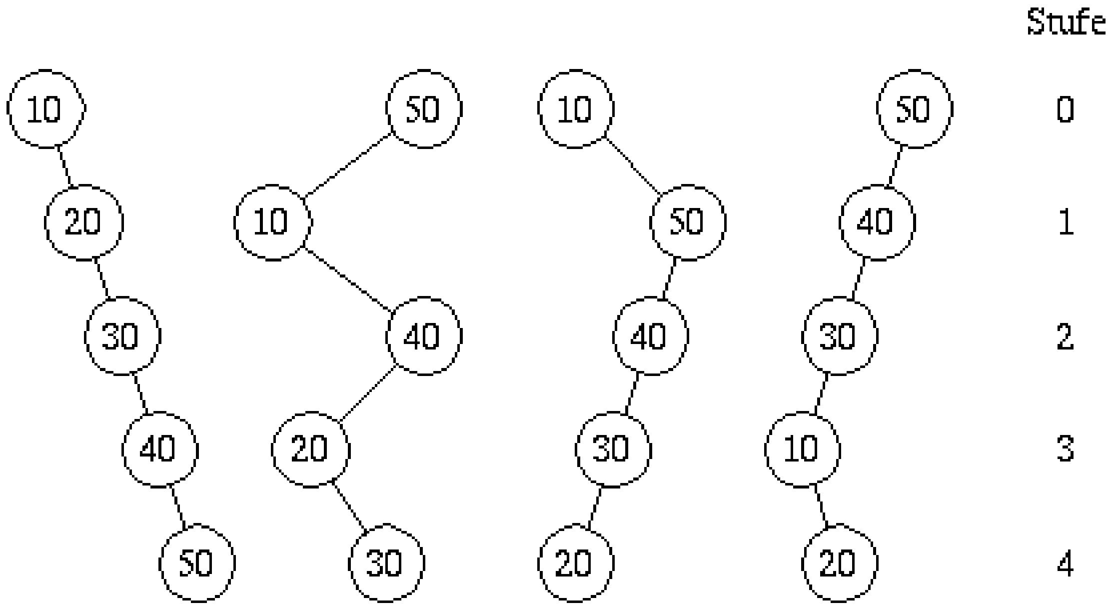
$$\bar{z} = p + 1 = \frac{1}{n} \cdot \sum_{i=0}^{h-1} (i+1) \cdot n_i$$

Beispiel:



# Maximale Zugriffskosten

Die längsten Suchpfade und damit die maximalen Zugriffskosten ergeben sich, wenn der binäre Suchbaum zu einer linearen Liste entartet




- Höhe: $h = l_{max} + 1 = n$

- Max. mittl. Zugriffskosten:

$$\bar{z}_{max} = \frac{1}{n} \cdot \sum_{i=0}^{n-1} (i+1) \cdot 1 = n \cdot \frac{(n+1)}{2n} = \frac{(n+1)}{2} = O(n)$$

# Minimale (mittlere) Zugriffskosten

Minimale (mittlere) Zugriffskosten: können in einer fast vollständigen oder ausgeglichenen Baumstruktur erwartet werden

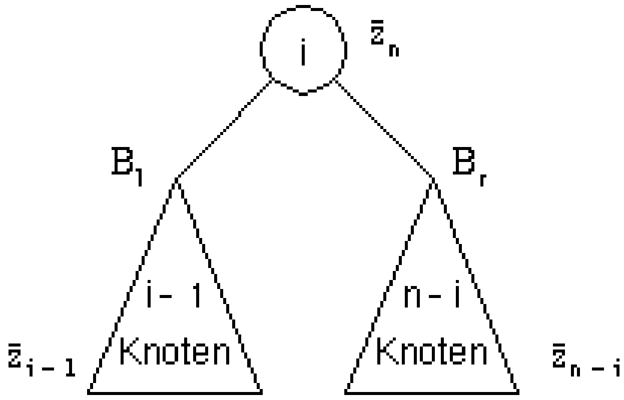
- Gesamtzahl der Knoten: $2^{h-1}-1 < n < 2^h$

- Höhe: $h = \lfloor \log_2 n \rfloor + 1$

- Minimale mittlere Zugriffskosten: $\bar{z}_{min} \approx \log_2 n - 1$ (Vollständiger Baum)

Differenz der mittleren zu den minimalen Zugriffskosten ist ein Maß für Dringlichkeit von Balancierungstechniken

# Bestimmung der mittleren Zugriffskosten




- n verschiedene Schlüssel mit den Werten 1, 2, ..., n seien in zufälliger Reihenfolge gegeben. Die Wahrscheinlichkeit, daß der erste Schlüssel den Wert i besitzt, ist 1/n (Annahme: gleiche Zugriffswahrscheinlichkeit auf alle Knoten)

- Für den Baum mit i als Wurzel erhalten wir

$$\bar{z}_n(i) = \frac{1}{n} \cdot ((\bar{z}_{i-1} + 1) \cdot (i-1) + 1 + (\bar{z}_{n-i} + 1) \cdot (n-i))$$

- Die Rekursionsgleichung läßt sich in nicht-rekursiver, geschlossener Form mit Hilfe der harmonischen Funktion $H_n = \sum_{i=1}^{n} \frac{1}{i}$ darstellen.

- Es ergibt sich $\bar{z}_n = 2 \cdot \frac{(n+1)}{n} \cdot H_n - 3$ = 2 ln(n) - c.

- Relative Mehrkosten: $\frac{\bar{z}_n}{\bar{z}_{min}} = \frac{2\ln(n) - c}{\log_2(n) - 1} \sim \frac{2\ln(n) - c}{\log_2(n)} = 2\ln(2) = 1{,}386\ldots$