

ADS: Algorithmen und Datenstrukturen

Teil VIII

Peter F. Stadler & Konstantin Klemm

Bioinformatics Group, Dept. of Computer Science & Interdisciplinary Center for
Bioinformatics, **University of Leipzig**

15. Dezember 2010

Binäre Suchbäume — Übersicht

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

Definition: Ein natürlicher binärer Suchbaum B ist ein Binärbaum. B 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.

Grundoperationen

- Einfügen
- direkte Suche
- sequentielles Durchlaufen
- Löschen

http:

`//webpages.ull.es/users/jriera/Docencia/AVL/AVLtreeapplet.htm`

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.

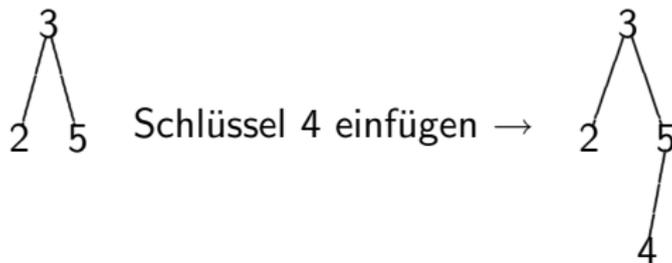
Sequentielle Suche

- Einsatz eines Durchlauf-Algorithmus (Zwischenordnung)

Einfügen in binären Suchbäumen

Einfügen von Schlüssel x :

- Suche nach x . Falls vorhanden, tue nichts.
- Sonst: Füge Knoten mit Schlüssel x als Blatt ein an dem Knoten, wo Suche nach x endete.



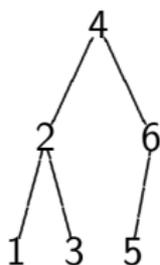
Löschen in binären Suchbäumen

Lösche Knoten (=Schlüssel) x aus dem Baum. Drei Fälle:

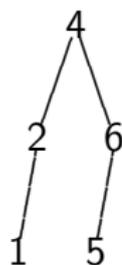
- 1 x ist ein Blatt. Dann entferne x aus dem Baum.
- 2 x hat genau einen nicht-leeren Unterbaum B . Entferne x und verbinde B direkt mit der Wurzel von x .
- 3 x hat zwei nicht-leere Unterbäume. Dann ist der direkte Nachfolger y von x ein Knoten mit leerem linkem Unterbaum. Lösche den Knoten y und ersetze x durch y .

Beispiel: Löschen

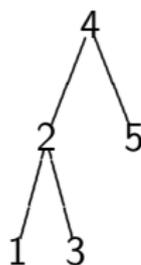
Gegebener Baum



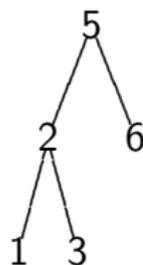
3 gelöscht



6 gelöscht



4 gelöscht



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: Bei dieser Art des Löschens wird kein Speicher frei!

Binäre Suchbäume: Zugriffskosten

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

- Erfolgreiche Suche nach einem Schlüssel in Knoten auf Stufe k braucht $k + 1$ Schritte. (z.B.: Wurzel auf Stufe 0 braucht 1 Schritt.)
- Gesamte Schritte, um jeden Knoten im Baum B mit Höhe $h(B)$ genau ein Mal zu finden:

$$S(B) = \sum_{i=1}^n (k_i + 1) = \sum_{k=0}^{h(B)-1} (k + 1)n_k$$

- k_i = Stufe des i -ten Knotens
- n_k = Anzahl Knoten auf Stufe k .
- mittlere Kosten pro erfolgreicher Suche (Zugriff)

$$z(B) = n^{-1}S(B)$$

Minimale mittlere Zugriffskosten

Minimale Zugriffskosten können in einer fast vollständigen oder ausgeglichenen Baumstruktur erwartet werden. Berechnung für *vollständigen Baum* der Höhe h :

- Knotenzahl gesamt $n = 2^h - 1$.
- Knotenzahl auf Stufe k ist $n_k = 2^k$
- Gesamte Schritte, um jeden Knoten im Baum B genau ein Mal zu finden:

$$S(B) = \sum_{k=0}^{h-1} 2^k(k+1) = (h-1)2^h + 1$$

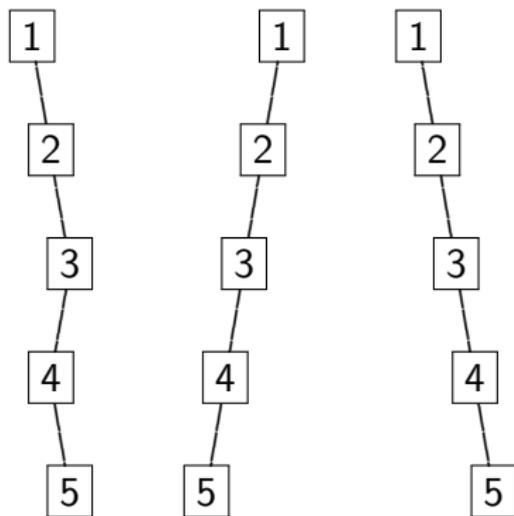
- Mittlere Zugriffskosten

$$z(B) = n^{-1}S(B) = \frac{(h-1)2^h + 1}{2^h - 1} \in \Theta(h) = \Theta(\log n)$$

Maximale Zugriffskosten

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

- Genau ein Knoten auf jeder Stufe: $n_k = 1$ für $k = 0, \dots, n - 1$.



Maximale Zugriffskosten

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

- Genau ein Knoten auf jeder Stufe: $n_k = 1$ für $k = 0, \dots, n-1$.
- Gesamte Schritte, um jeden Knoten im Baum B genau ein Mal zu finden:

$$S(B) = \sum_{k=0}^{n-1} (k+1) = \frac{n(n+1)}{2}$$

- mittlere Zugriffskosten

$$z(B) = n^{-1}S(B) = \frac{n+1}{2}$$

- Kosten wachsen linear (statt logarithmisch) mit n .

Mittlere Zugriffskosten in zufälligen Bäumen (1)

- n verschiedene Schlüssel mit den Werten $1, 2, \dots, n$ seien in zufälliger Reihenfolge gegeben, also alle $n!$ Anordnungen gleich wahrscheinlich.
- Baum, in den die Schlüssel in dieser Reihenfolge eingefügt wurden, lässt sich zerlegen in Wurzel i , linken Teilbaum mit $i - 1$ Schlüsseln und rechten Teilbaum mit $n - i$ Schlüsseln.
- Für die Teilbäume sind die Schlüsselreihenfolgen auch wieder vollkommen zufällig.
- $S_n =$ Erwartungswert der Stufensumme in einem solchen zufälligen Baum mit n Knoten.
- Rekursion

$$S_n = 1 + n^{-1} \sum_{i=1}^n [S_{i-1} + (i-1) + S_{n-i} + (n-i)]$$

= gleich gewichtetes Mittel über alle Schlüssel i in der Wurzel

Mittlere Zugriffskosten in zufälligen Bäumen (2)

Ausrechnen, um Rekursionsgleichung handlicher zu schreiben:

$$\begin{aligned} S_n &= 1 + n^{-1} \sum_{i=1}^n [S_{i-1} + (i-1) + S_{n-i} + (n-i)] \\ &= 1 + \frac{n(n-1)}{n} + n^{-1} \sum_{i=1}^n S_{i-1} + S_{n-i} \\ &= n + \frac{2}{n} \sum_{i=0}^{n-1} S_i \end{aligned}$$

Mittlere Zugriffskosten in zufälligen Bäumen (3)

Rekursionsgleichung bis hierher:

$$S_n = n + \frac{2}{n} \sum_{i=0}^{n-1} S_i$$

Schwierigkeit: S_n hängt nicht nur vom direkten Vorgänger S_{n-1} ab sondern von allen. Aber

$$\sum_{i=0}^{n-2} S_i = \frac{n-1}{2} (S_{n-1} - (n-1))$$

= Rekursionsgleichung für $n-1$ nach dem Summenterm aufgelöst.
Einsetzen in die Rekursion für S_n :

$$S_n = n + \frac{2}{n} \left[S_{n-1} + \frac{n-1}{2} (S_{n-1} - (n-1)) \right] = 2 - \frac{1}{n} + \left(1 + \frac{1}{n} \right) S_{n-1}$$

Mittlere Zugriffskosten in zufälligen Bäumen (4)

Rekursionsgleichung nunmehr:

$$S_n = 2 - \frac{1}{n} + \left(1 + \frac{1}{n}\right) S_{n-1}$$

bzw. als Differenzengleichung

$$S_n - S_{n-1} = 2 - \frac{1}{n} + \frac{1}{n} S_{n-1}$$

Approximation als Differentialgleichung

$$\frac{dS}{dn} = 2 - \frac{1}{n} + \frac{1}{n} S$$

Lösung

$$S_n = 2n \ln n + 1$$

erfüllt Anfangsbedingung $S_1 = 1$ (fein!).

Vergleich: zufällige mit vollständigen Bäumen

Bäume mit n Knoten, mittlere Zugriffskosten z_n

- zufälliger Baum

$$z_n^{\text{zufällig}} = n^{-1} S_n = 2 \ln n + n^{-1}$$

- vollständiger Baum

$$z_n^{\text{vollst}} = \frac{(h-1)2^h + 1}{n} = (\log_2(n+1) - 1) \frac{n+2}{n}$$

- Verhältnis der Kosten asymptotisch

$$\lim_{n \rightarrow \infty} \frac{z_n^{\text{zufällig}}}{z_n^{\text{vollst}}} = 2 \ln 2 \approx 1.39$$

Balancierte Binärbäume

- Der ausgeglichene binäre Suchbaum verursacht für alle Grundoperationen die geringsten Kosten
- Perfekte Balancierung zu jeder Zeit ist jedoch sehr teuer.
- In welchem Maße sollen Strukturabweichungen bei Einfügungen und Löschungen toleriert werden ?

l -balancierter Binärbaum

Seien $B_l(x)$ und $B_r(x)$ die linken und rechten Unterbäume eines Knotens x . Weiterhin sei $h(B)$ die Höhe eines Baumes B .

Definition: Ein l -balancierter Binärbaum ist entweder leer oder es ist ein Baum, bei dem für jeden Knoten x gilt:

$$|h(B_l(x)) - h(B_r(x))| \leq l .$$

l läßt sich als Maß für die zulässige Entartung im Vergleich zur ausgeglichenen Baumstruktur auffassen.

AVL-Bäume = 1-balancierte Binärbäume (nächste Vorlesung).