ADS: Algorithmen und Datenstrukturen

Teil VII

Uwe Quasthoff

Institut für Informatik
Abteilung Automatische Sprachverarbeitung
Universität Leipzig

28.11.2017

[Letzte Aktualisierung: 12/12/2017, 14:02]

Bäume als Graphen

- Ein *Graph* ist ein Paar (V, E). Hierbei ist V eine Menge (=Knotenmenge) und E eine Menge von ungeordneten Paaren (=Kanten) aus V.

Ein Baum ist

- ein azyklischer, einfach zusammenhängender Graph
- d. h.
  - er enthält keine Schleifen: Für keine Kante fallen Startknoten und Endknoten zusammen.
  - er ist zusammenhängend: Für jeden Startknoten und jeden Zielknoten gibt es (mindestens) einen Weg vom Startknoten zum Zielknoten.
  - er enthält keine Zyklen: Für jeden Startknoten und jeden Zielknoten gibt es höchstens einen Weg vom Startknoten zum Zielknoten.

Orientierte Bäume

- Anschauliche Beschreibung: Verallgemeinerung von Listen. Die Kanten sind zusätzlich gerichtet und führen von der Wurzel in Richtung der Blätter. Anders als in einer Liste hat jedes Element (Knoten) möglicherweise mehrere Nachfolger (Kinder).
  - Genau 1 Knoten ohne Vorgänger: Wurzel
  - Knoten *mit* Nachfolger: innere Knoten
  - Knoten *ohne* Nachfolger: Blätter
- Häufig verwendete Datenstruktur: Entscheidungsbäume, Syntaxbäume, Ableitungsbäume, Suchbäume, ...
- Hier besonders interessant: Verwendung von Bäumen zur Speicherung von Schlüsseln und Realisierung der Wörterbuchoperationen (Suchen, Einfügen, Entfernen) in Binärbäumen.

Orientierte Bäume

Eine Menge B ist ein orientierter (Wurzel-) Baum, falls

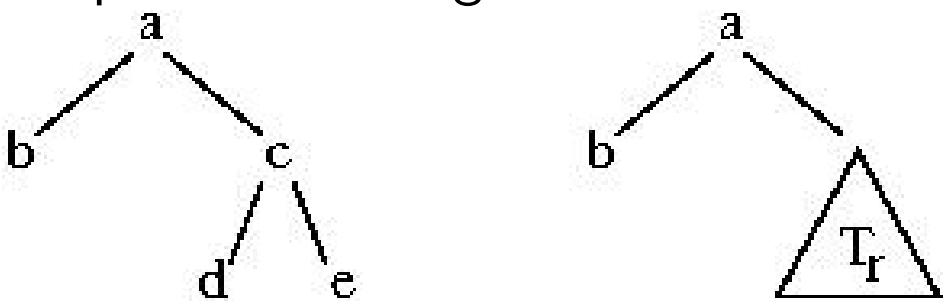
1. in B ein ausgezeichnetes Element w – die Wurzel von B – existiert
2. die Elemente in $B \setminus \{w\}$ disjunkt zerlegt werden können in $B_1, B_2, \ldots, B_m$, wobei jedes B_i ebenfalls ein orientierter Baum ist.

Anschaulicher:

- Die Kanten des Graphen sind gerichtet.
- Von der Wurzel gehen nur Kanten aus, keine treffen ein.
- Jeder Knoten ist von der Wurzel aus auf genau einem Weg entlang der gerichteten Kanten erreichbar.
- Für jeden Nicht-Wurzelknoten gibt es Knoten, die nicht entlang der gerichteten Kanten erreichbar sind.

Darstellungsarten für orientierte Bäume

1. Graphendarstellung



2. Mengendarstellung
   $\{\{a, b, c, d, e\}, \{b\}, \{c, d, e\}, \{d\}, \{e\}\}$
3. Klammerdarstellung
   $(a, (b), (c, (d), (e)))$
4. Rekursives Einrücken

   a

   b

   c

   d

   e

Eigenschaften I

Baum B **heißt geordnet**, wenn Nachfolger jedes Knotens geordnet sind (1., 2., 3. etc.; linker, rechter). Bei einem geordneten Baum bilden die Unterbäume B_i jedes Knotens eine geordnete Menge. (Beispiel: Arithmetischer Ausdruck)

Eine geordnete Menge von geordneten Bäumen heißt geordneter Wald.

Ordnung von B: maximale Anzahl von Nachfolgern eines Knotens

Eigenschaften II

Tiefe eines Knotens: Abstand zur Wurzel, d.h. Anzahl der Kanten auf dem Pfad von diesem Knoten zur Wurzel. Die Knoten auf der **Stufe** i sind alle Knoten mit Tiefe i.

Höhe eines Baums: maximale Tiefe eines Knotes $\underline{+\ 1}$. Der leere Baum hat Höhe 0.

Ein Baum der Ordnung n heißt **vollständig**, wenn alle Blätter dieselbe Tiefe haben und auf jeder Stufe die maximale Anzahl von Knoten vorhanden ist.

Binärbäume

Ein Binärbaum ist eine endliche Menge von Elementen, die entweder leer ist oder ein ausgezeichnetes Element - die Wurzel des Baumes - besitzt und folgende Eigenschaften aufweist:

- Die verbleibenden Elemente sind in zwei disjunkte Untermengen zerlegt.
- Jede Untermenge ist selbst wieder ein Binärbaum und heißt linker bzw. rechter Unterbaum des ursprünglichen Baumes

Ein Binärbäum ist also ein geordneter Baum, in dem jeder Knoten höchstens zwei Kinder besitzt (Ordnung 2).

Formale ADT-Spezifikation: BINTREE

Datentyp BINTREE, Basistyp ELEM

Operationen:

CREATE:		$\rightarrow$ BINTREE
EMPTY:	BINTREE	$\rightarrow \{\text{TRUE, FALSE}\}$
BUILD:	BINTREE$\times$ELEM$\times$BINTREE	$\rightarrow$ BINTREE
LEFT:	BINTREE$\setminus\{0\}$	$\rightarrow$ BINTREE
ROOT:	BINTREE$\setminus\{0\}$	$\rightarrow$ ELEM
RIGHT:	BINTREE$\setminus\{0\}$	$\rightarrow$ BINTREE

Axiome

1. CREATE() $= 0$;
2. EMPTY (CREATE()) $=$ TRUE;
3. $\forall l, r \in$ BINTREE, $\forall d \in$ ELEM:
   EMPTY (BUILD (l, d, r)) $=$ FALSE
   LEFT (BUILD (l, d, r)) $=$ l;
   ROOT (BUILD (l, d, r)) $=$ d;
   RIGHT(BUILD (l, d, r)) $=$ r;

Beispiele für Konstruktion

Welche Binärbäume entstehen durch

- BUILD (BUILD (0, b, BUILD (0, d, 0)), a, BUILD (0, c, 0))
- BUILD (BUILD (BUILD (0, d, 0), b, 0), a, BUILD (0, c, 0))

Eigenschaften von Binärbäumen I

Satz: Die maximale Anzahl von Knoten eines Binärbaumes

1. auf Stufe i ist 2^i, $i \geq 0$
2. der Höhe h ist 2^h-1, $h \geq 0$ (Der leere Baum hat Höhe 0)

Definition: Ein *vollständiger* Binärbaum der Höhe $k+1$ hat folgende Eigenschaften:

- Jeder Knoten der Stufe k ist ein Blatt.
- Jeder Knoten auf einer Stufe $< k$ hat nicht-leere linke und rechte Unterbäume.

Definition: In einem *strikten* Binärbaum besitzt jeder innere Knoten nicht-leere linke und rechte Unterbäume

Eigenschaften von Binärbäumen II

Definition: Ein *ausgeglichener* Binärbaum der Höhe $k+1$ ist ein Binärbaum, so dass gilt:

1. Jedes Blatt im Baum ist auf Stufe k oder $k-1$ (falls $k \geq 1$).
2. Jeder Knoten auf Stufe $< k-1$ hat nicht-leere linke und rechte Teilbäume

Definition: Ein *fast vollständiger* Binärbaum ist ein ausgeglichener Binärbaum, bei dem die Blätter auf Stufe k möglichst weit links stehen. Letzteres heisst formal: für jeden inneren Knoten dessen rechter Teilbaum mindestens ein Blatt auf Stufe k enthält, ist sein linker Teilbaum nicht leer und hat alle Blätter auf Stufe k, ist also vollständig.

Eigenschaften von Binärbäumen III

- **Definition**: Zwei Binärbäume werden als *ähnlich* bezeichnet, wenn sie dieselbe Struktur besitzen.
- **Definition**: Sie heißen *äquivalent*, wenn sie ähnlich sind und dieselbe Information enthalten.

Abgeleitete Eigenschaften von Binärbäumen

1. Für zwei beliebige Knoten in einem Baum existiert genau ein Pfad, der sie verbindet.
2. Ein Binärbaum mit N Knoten hat $N - 1$ Kanten.
3. Ein strikter binärer Baum mit N inneren Knoten hat $N + 1$ äußere Knoten.
4. Die Höhe eines vollständigen binären Baumes mit N Knoten beträgt $\log_2(N + 1)$.

Speicherung in Binärbäumen

Vereinbarung:

- In jedem Knoten des Baumes wird genau ein Schlüssel gespeichert. Der Baum enthält also so viele Knoten wie Schlüssel.

Anordnung der Schlüssel in binären Suchbäumen (kommen später)

- für jeden Knoten gilt: Die Schlüssel im linken Teilbaum sind sämtlich kleiner als der in der Wurzel und dieser ist wiederum kleiner als die Schlüssel im rechten Teilbaum.

Speicherung von Binärbäumen

(1) Verkettete Speicherung: Freispeicherverwaltung der Struktur wird von der Speicherverwaltung des Programmiersystems übernommen.

(2) Feldbaum-Realisierung: Simulation einer dynamischen Struktur in einem statischen Feld
Eigenschaften:
- statische Speicherplatzzuordnung
- explizite Freispeicherverwaltung

Sequenzielle Speicherung von Binärbäumen

(3) Sequentielle Speicherung: Methode kommt ohne explizite Verweise aus. für fast vollständige oder zumindest ausgeglichene Binärbäume bietet sie eine sehr elegante und effiziente Darstellungsform an.

Ein fast vollständiger Baum mit n Knoten wird sequentiell nach folgendem Numerierungsschema gespeichert. für jeden Knoten mit Index i, $1 \leq i \leq n$, gilt:

- Vater(i) hat Nummer $\lfloor i/2 \rfloor$ für $i > 1$
- Lsohn(i) hat Nummer $2i$ für $2i \leq n$
- Rsohn(i) hat Nummer $2i + 1$ für $2i + 1 \leq n$

Durchlaufen eines Binärbaums

Baumdurchlauf (Traversierung) = Verarbeitung aller gespeicherten Schlüssel in einer Reihenfolge, die durch die Baumstruktur gegeben ist.

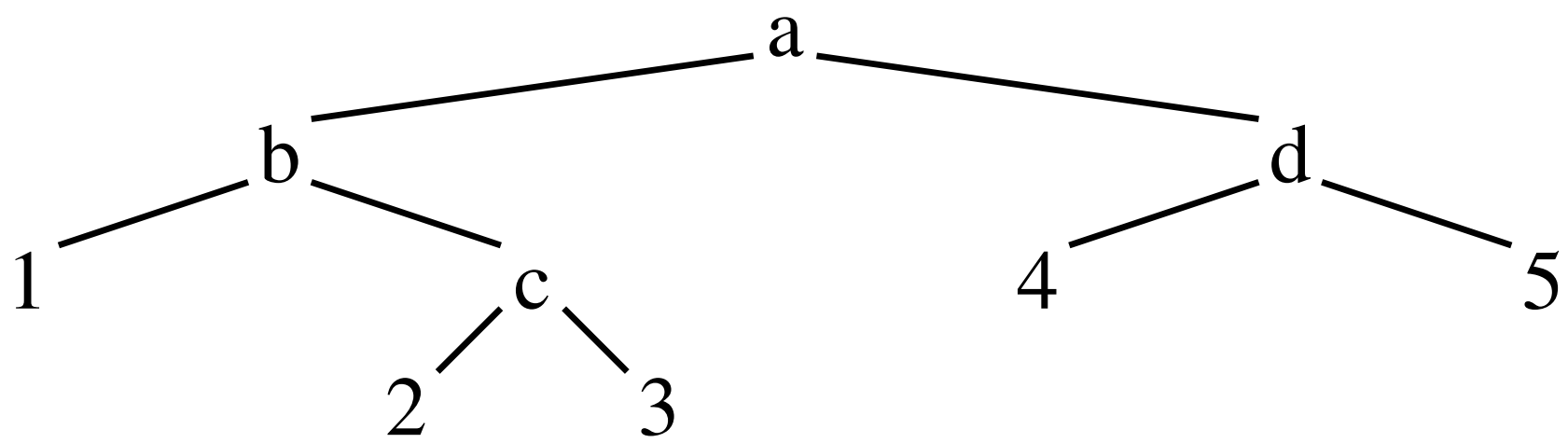
Rekursiv anzuwendende Schritte

- Verarbeite Wurzel: W
- Durchlaufe linken UB: L
- Durchlaufe rechten UB: R

Durchlaufprinzip impliziert sequentielle, lineare Ordnung auf der Menge der Knoten
Es gibt 6 Möglichkeiten, W, L und R anzuordnen, aber Konvention: linker UB vor rechtem UB
Verbleibende 3 Möglichkeiten:

- Vorordnung (preorder): WLR
- Zwischenordnung (inorder): LWR
- Nachordnung (postorder): LRW

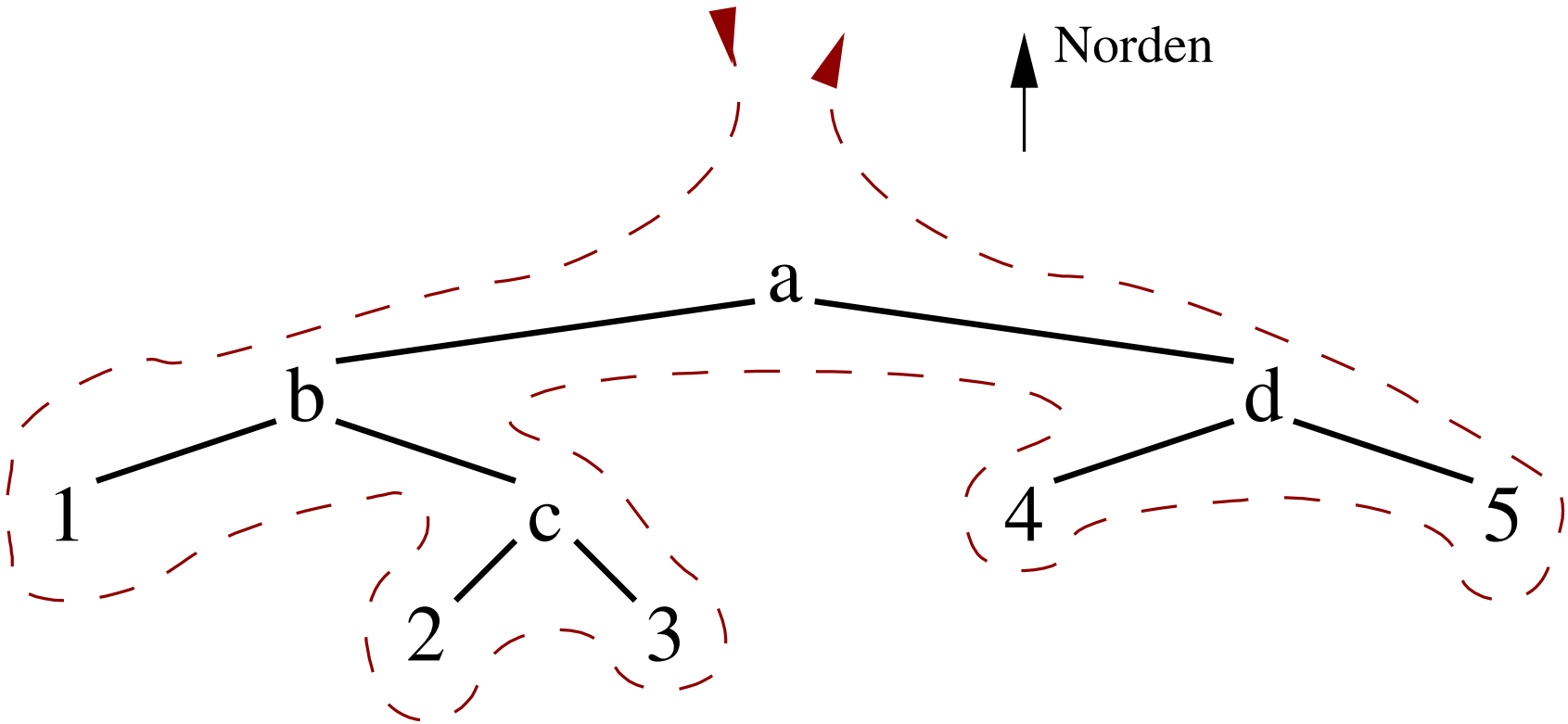
Durchlaufmöglichkeiten I



Vorordnung, preorder, WLR:	a	b	1	c	2	3	d	4	5
Zwischenordnung, inorder, LWR:	1	b	2	c	3	a	4	d	5
Nachordnung postorder, LRW:	1	2	3	c	b	4	5	d	a

Durchlauf: Anschauliche Darstellung

"Wanderung" um den Baum, Knoten werden aufgerufen bei bestimmter relativer Bewegung:



bei Passage Richtung Süden, WLR:	a	b	1	c	2	3	d	4	5
an Südseite, inorder, LWR:	1	b	2	c	3	a	4	d	5
bei Passage Richtung Norden, LRW:	1	2	3	c	b	4	5	d	a

Rekursive Realisierung

Rekursive Version für Inorder-Traversierung (LWR)

```
DurchlaufInOrder(Baum)
  Falls Baum leer, dann fertig
  sonst {
    DurchlaufInOrder(LinkeTochter(Baum))
    Drucke Inhalt des aktuellen Knotens
    DurchlaufInOrder(RechteTochter(Baum)) }
```

Iterative Realisierung

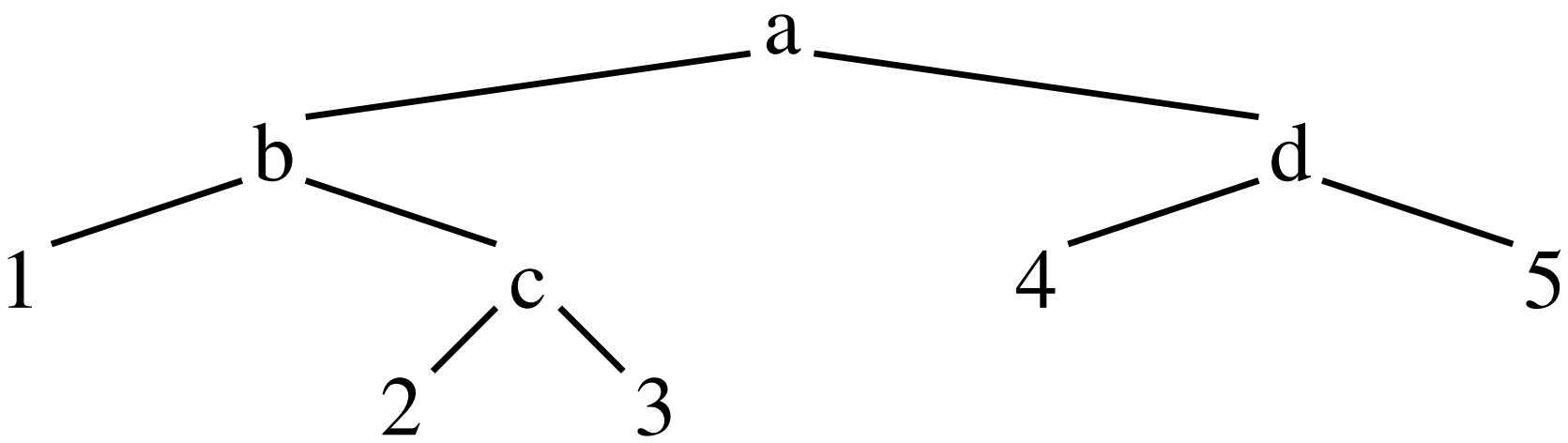
Iterative Version von LWR (inorder)

Ziel: effizientere Ausführung durch eigene Stapelverwaltung

Vorgehensweise:

- Nimm, solange wie möglich, linke Abzweigung und speichere die Knoteninhalte des zurückgelegten Weges auf einem Stapel (Aktion 1).
- Wenn es links nicht mehr weitergeht, wird der oberste Knoten des Stapels ausgegeben und vom Stapel entfernt. Der Durchlauf wird mit dem rechten Unterbaum des entfernten Knotens fortgesetzt (Aktion 2).

Beispiel für iteratives LWR



Gefädelte Binärbäume I

Ziel: Weitere Verbesserung von iterativen Durchlaufalgorithmen

Problem: Es gibt in vielen Fällen keinen Zeiger auf den Nachfolger oder Vorgänger einer Traversierung im Baum.

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

Gefädelte Binärbäume II

Lösung 2: Vermeidung von Redundanz

Eine zweite Art der Fädelung kommt ohne zusätzliche Zeiger aus und erfordert daher geringeren Speicherplatzaufwand. Die 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 Inorder-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), Binärbaum
- vollständiger, fast vollständiger, strikter, ausgeglichener Binärbaum
- Höhe, Grad, Stufe / Pfadlänge

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.