Verkettete Datenstrukturen: Bäume

Graphen

Gerichteter Graph:

Menge von „Knoten“ (= Elementen)
+ Menge von „Kanten“.

Kante:

Verbindung zwischen zwei Knoten $k_1 \longrightarrow k_2$
= Paar von Knoten (k_1, k_2).

Menge aller Kanten = binäre Relation auf der Knotenmenge.

Graphen

Wir nehmen hier an, daß die Knotenmenge endlich ist.
(Damit ist auch die Kantenmenge endlich.)

Man benutzt gelegentlich aber auch unendliche Knotenmengen.

Graphen

Weg:

Folge von Knoten, so daß je zwei aufeinanderfolgende durch eine Kante verbunden sind:
$k_1 \longrightarrow k_2 \longrightarrow \ldots \longrightarrow k_{n+1}$.

Länge des Weges: Anzahl der durchlaufenen Kanten (also n).

Sonderfall: Leerer Weg = Weg der Länge 0
(von einem Knoten zu sich selbst).

Zyklus:

Nicht-leerer Weg von einem Knoten zu sich selbst:
$k_1 \longrightarrow k_2 \longrightarrow \ldots \longrightarrow k_n \longrightarrow k_1$.

Bäume

Baum:

Gerichteter Graph, der die folgenden drei Bedingungen erfüllt:

Es gibt einen Knoten, der nicht Endknoten einer Kante ist.
(Dieser Knoten heißt Wurzel des Baums.)

Jeder andere Knoten ist Endknoten genau einer Kante.

Der Graph ist zusammenhängend:
Zu jedem Knoten existiert ein Weg von der Wurzel aus.

Bäume sind zyklenfrei (d. h., in einem Baum gibt es keine Zyklen).
(Warum?)

Bäume

Manchmal, je nach Anwendung, läßt man auch den „leeren Baum“ zu, also einen Baum mit null Knoten.

Wald:

> Menge von Bäumen (disjunkt, also ohne gemeinsame Knoten).

Bäume

Sei $k_1 \longrightarrow k_2$ eine Kante in einem Baum.

k_1 bezeichnet man als Elternknoten von k_2.

k_2 bezeichnet man als Kindknoten von k_1.

Grad eines Baums

(Ausgangs-)Grad eines Knotens = Anzahl seiner Kinder.

Knoten mit Grad 0 heißen Blätter.

Knoten mit Grad ≥ 1 heißen innere Knoten.

Der Grad eines Baums ist der maximale Grad seiner Knoten.
Wichtiger Spezialfall: Binärbaum = Baum mit Grad 2.

Was ist ein Baum mit Grad 1?

Tiefe eines Baums

In einem Baum existiert von einem beliebigen Knoten k_0 zu einem beliebigen Knoten k_1 höchstens ein Weg.

Von der Wurzel bis zu einem beliebigen Knoten k_1 existiert genau ein Weg.

Tiefe des Knotens k_1 = Länge des Wegs von der Wurzel bis k_1.

Tiefe eines Baums = maximale Tiefe seiner Knoten.

Unterbäume

Wenn k ein Knoten in einem Baum ist, dann ist die Menge aller Knoten, zu denen von k aus ein Weg existiert, mit den Kanten dazwischen wieder ein Baum.

Dieser Baum heißt *Unterbaum ab* k; seine Wurzel ist k.

Rekursive Definition für Bäume:
Ein Baum besteht aus einem Wurzelknoten und einer (möglicherweise leeren) Menge von direkten Unterbäumen, zu denen je eine Kante von der Wurzel führt.

Bäume: Beispiele

Firmenhierarchie

Knotenmenge: Beschäftigte der Firma.

Kante $k_1 \longrightarrow k_2$: k_1 ist Vorgesetzter von k_2.

Bäume: Beispiele

Java-Klassenhierarchie

Knotenmenge: Java-Klassen

Kante $k_1 \longrightarrow k_2$: k_1 ist Oberklasse von k_2.

Bäume: Beispiele

Internet

Knotenmenge: Domain-Adressen (`x1.studcs.uni-sb.de`).

Kante $k_1 \longrightarrow k_2$: k_2 ist Subdomain von k_1.

(Baum oder Wald?)

Bäume: Beispiele

Arithmetische Ausdrücke

Knotenmenge: arithmetische Operatoren ($+$, $-$),
Variablennamen (x, y),
Zahlen.

Kante $k_1 \longrightarrow k_2$: Der Term, der durch den Unterbaum ab k_2 repräsentiert wird, ist Argument des Operators k_1.

Vorsicht: Hier gibt es Probleme.

Zwei kleine Erweiterungen

Wenn wir arithmetische Ausdrücke als Bäume repräsentieren wollen, dann sind zwei Erweiterungen des bisherigen Konzepts notwendig:

Wir müssen annehmen, daß nicht die Zahlen und Operatoren selbst die Knoten sind, sondern daß die Knoten mit den Zahlen oder Operatoren nur markiert sind.
D. h., es kann mehrere (verschiedene!) Knoten geben, die mit dem gleichen Wert (z.B.: „+“) markiert sind.

Wir müssen für jeden Knoten die Reihenfolge seiner Kindknoten festlegen, zum Beispiel, indem wir die Kanten durchnumerieren ($\leadsto$ Kantenmarkierung).
Ein solcher Baum heißt geordneter Baum.

Bäume: Beispiele

Abstrakte Syntaxgraphen

Knotenmenge: wie bei arithmetischen Ausdrücken, außerdem z. B.: `while (...) {...}`.

Kante $k_1 \longrightarrow k_2$: Das syntaktische Element, das durch den Unterbaum ab k_2 repräsentiert wird, ist Argument des Operators k_1.

Bäume: Beispiele

Unix-Dateisystem

Knotenmenge: Dateien und Verzeichnisse.

Kante $k_1 \longrightarrow k_2$: Das Verzeichnis k_1 enthält die Datei oder das Verzeichnis k_2.

Bäume: Beispiele

Suchbäume

Knotenmenge: irgendeine total geordnete Menge
(z. B.: Zahlen, Strings, ...).

Falls k_2 linkes Kind von k_1 ist, dann enthält der Unterbaum ab k_2 nur Knoten, die kleiner als k_1 sind.

Falls k_2 rechtes Kind von k_1 ist, dann enthält der Unterbaum ab k_2 nur Knoten, die größer als k_1 sind.

Bäume: Beispiele

Ein Nicht-Beispiel: Familienstammbaum

> In beide Richtungen kein Baum, sondern nur ein DAG
> (directed acyclic graph = gerichteter azyklischer Graph):
> Knoten können Endknoten mehrerer Kanten sein.

Auch das Unix-Dateisystem ist eigentlich kein Baum,
sondern ein DAG:
mehrere Links auf eine Datei sind möglich.

Implementierung

Die erste Frage bei der Implementierung:
In welcher Richtung werden die Kanten operational benötigt?

(1) von oben nach unten (Domainadressen),

(2) von unten nach oben (Java-Klassenhierarchie),

(3) in beide Richtungen (Unix-Dateisystem).

Fall (2) ist ein Sonderfall:

jeder Knoten benötigt nur eine Referenz
(auf seinen Elternknoten, falls vorhanden)
$\leadsto$ Implementierung wie einfachverkettete Listen (mit Sharing).

Implementierung

Für (1) und (3) brauchen wir eine andere Datenstruktur.

Wir betrachten Fall (1); Fall (3) geht im wesentlichen wie (1),
nur daß noch eine Referenz auf den Elternknoten hinzukommt.

```
public class Knoten {
    Knoten[] kinder; // jedes kinder[i] ist Referenz auf
                     // einen Kindknoten
    int wert;
    // oder z.B.: Object wert
}
```

Implementierung

Alternative:

Manchmal (Beispiel: Firmenhierarchie) ist es erwünscht, daß zur Laufzeit eine a priori unbekannte Zahl von Kindknoten hinzugefügt werden soll. Die Anzahl der Kinderknoten ist also veränderlich. Dann besser:

```
public class Knoten {
    Vector kinder;
    // Vector ist in java.util vordefiniert,
    // alternativ geht auch eine Liste
    // (siehe letzte Woche).
    int wert;
    // oder z.B.: Object wert
}
```

Implementierung

Alternative:

In anderen Anwendungsgebieten ist im Gegensatz dazu der Grad des Baumes (!) klein und bereits zur Compilezeit bekannt (Beispiel: Suchbäume oder andere binäre Bäume). Dann geht auch:

```
public class Knoten {
    Knoten kindLinks;
    Knoten kindRechts;
    int wert;
    // oder z.B.: Object wert
}
```

Implementierung

Elementsicht:

`kinder[i]` bzw. `(Knoten) kinder.elementAt(i)`
bzw. `kindLinks` und `kindRechts` sind Referenzen
auf Kindknoten.

Baumsicht:

`kinder[i]` bzw. `(Knoten) kinder.elementAt(i)`
bzw. `kindLinks` und `kindRechts` sind Referenzen
auf direkte Unterbäume.

Algorithmen auf Bäumen

Generell: meist rekursiv

entweder: rufe Funktion rekursiv für alle Kinder
(bzw. für alle direkten Unterbäume) auf,

oder: rufe Funktion rekursiv für eines der Kinder auf
(falls vorhanden).

Algorithmen auf Bäumen

Beispiel: Ausdrucken („Linearisieren“) eines Baums:

präfix-Ausgabe:

drucke den Wert des Knotens,

rufe die Prozedur rekursiv für alle Kindknoten auf.

postfix-Ausgabe:

rufe die Prozedur rekursiv für alle Kindknoten auf,

drucke den Wert des Knotens.

Algorithmen auf Bäumen

Beispiel: Suchen einer Zahl n in einem Suchbaum:

falls n gleich dem Wert des Knotens ist: return true.

falls n kleiner als der Wert des Knotens ist:

falls es keinen linken Kindknoten gibt: return false.

sonst rufe Prozedur rekursiv für linken Kindknoten auf.

falls n größer als der Wert des Knotens ist:

falls es keinen rechten Kindknoten gibt: return false.

sonst rufe Prozedur rekursiv für rechten Kindknoten auf.

Algorithmen auf Bäumen

Beispiel: Einfügen einer Zahl n in einen Suchbaum:

falls n gleich dem Wert des Knotens ist: return.

falls n kleiner als der Wert des Knotens ist:

falls es keinen linken Kindknoten gibt:
erzeuge linken Kindknoten mit Wert n; return.

sonst rufe Prozedur rekursiv für linken Kindknoten auf.

falls n größer als der Wert des Knotens ist:

falls es keinen rechten Kindknoten gibt:
erzeuge rechten Kindknoten mit Wert n; return.

sonst rufe Prozedur rekursiv für rechten Kindknoten auf.

Algorithmen auf Bäumen

Schwieriger: Löschen einer Zahl n in einem Suchbaum:

falls n in dem Suchbaum nicht vorkommt: return.

falls n in einem Blatt des Suchbaums vorkommt:
lösche das Blatt.

falls n in einem inneren Knoten des Suchbaums vorkommt:
???