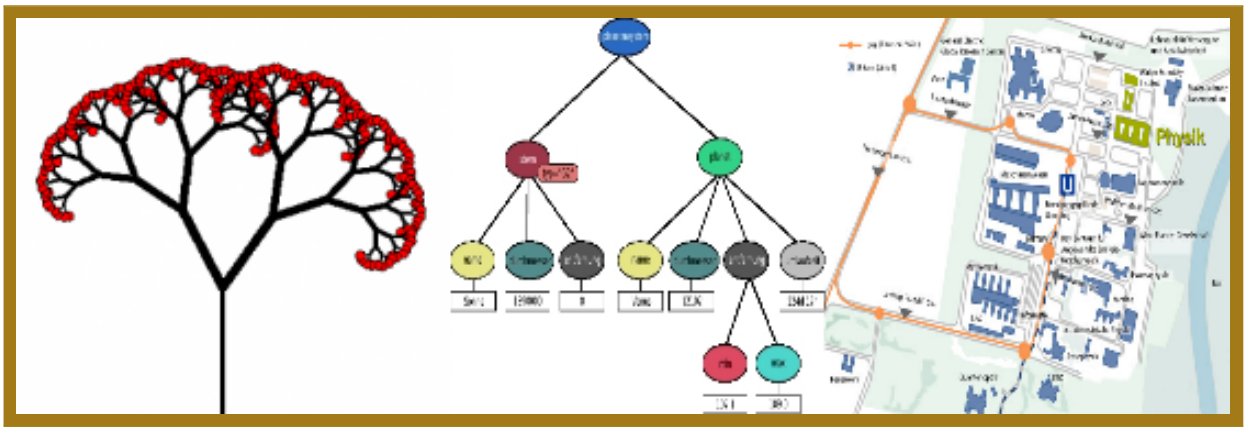
Bäume als spezielle Graphen

M. Jakob

Gymnasium Pegnitz

3. Februar 2019



Inhaltsverzeichnis

Die rekursive Datenstruktur `Baum`
- Listen umstrukturieren

Geordnete Binärbäume
- Deklaration und Initialisierung
- Deklaration `Suchen`
- Einfügen
- Baum in perfekter Komposition
- Baumdurchläufe

Die Datenstruktrur Graph
- Kennzeichen von Graphen
- Darstellung von Graphen
- Implementation
- Graphendurchlauf

In diesem Abschnitt

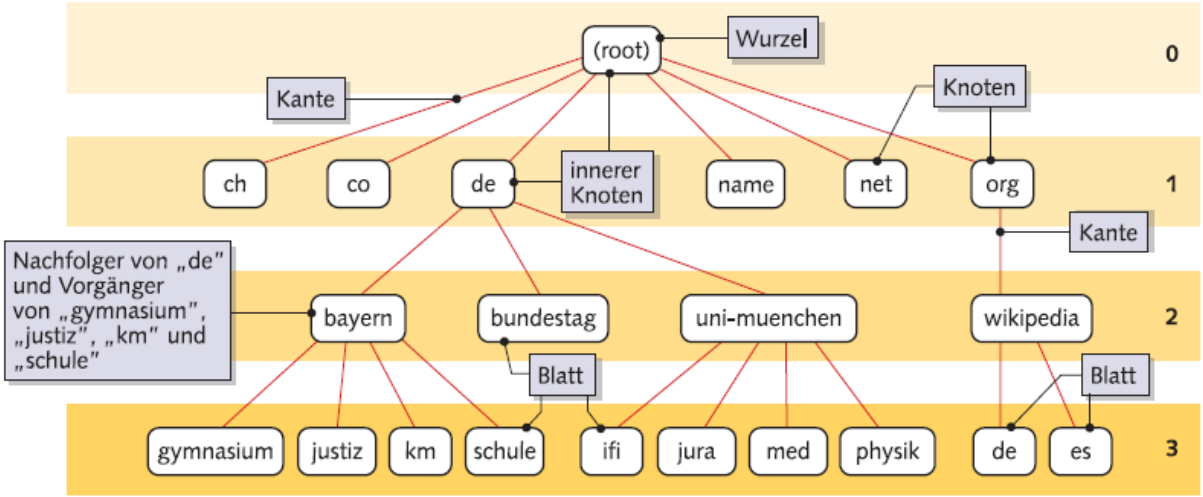
Die rekursive Datenstruktur `Baum`
Listen umstrukturieren

Definition Baum

Definition `Baum`

Bestandteile der Datenstruktur `Baum` sind Knoten und Kanten. Der erste Knoten, der keinen Vorgänger hat, heißt Wurzel. Bei Knoten unterscheidet man zwischen Blättern (Knoten ohne Nachfolger) und inneren Knoten. Die Tiefe oder Ebene eines Knotens ist die Anzahl der Kanten des Weges von der Wurzel bis hin zu diesem Knoten. Die Tiefe n des Baums ist gleich der maximalen Knotentiefe, die Ordnung k des Baums ist gleich der Maximum an Kanten, die von einem Knoten ausgehen.
In die n-te Ebene eines Baumes der Ordnung k passen maximal k^n Elemente, ein Baum der Tiefe n kann maximal $2^{n+1} - 1$ aufnehmen.

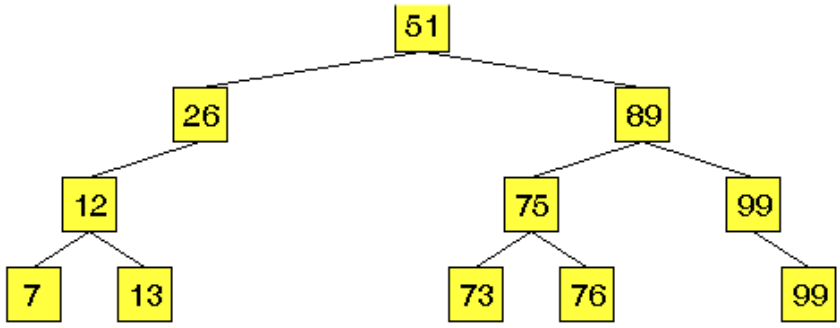
Definition Baum



Defintion

Definition geordneter Binärbaum

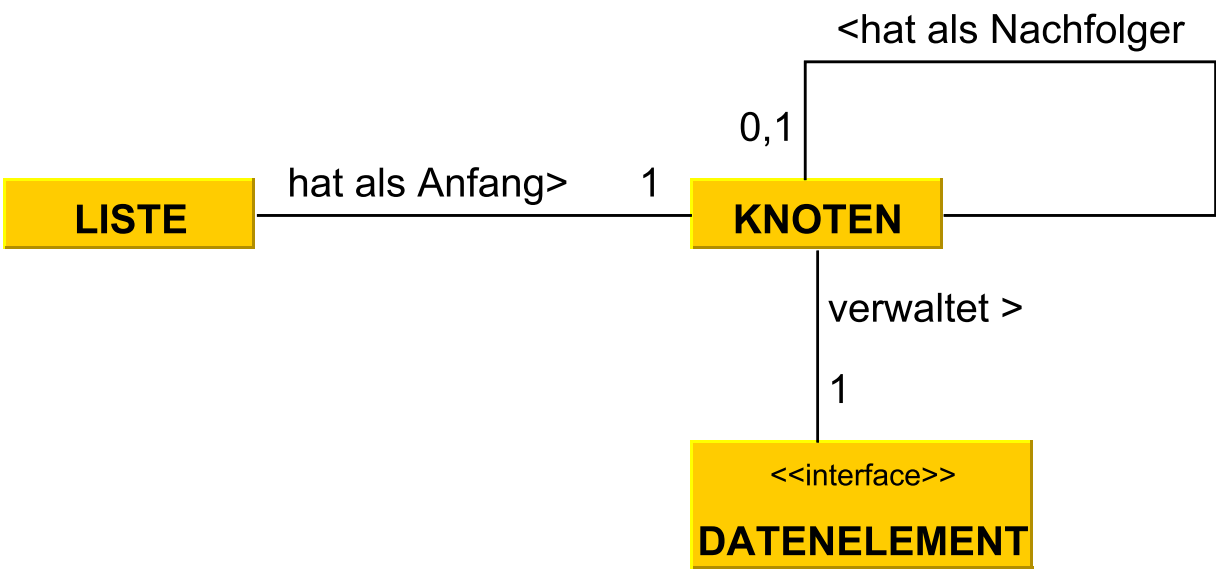
Ein geordneter Binärbaum ist ein Binärbaum bei dem für jeden Knoten K die Werte in allen Knoten des linken Teilbaums kleiner sind als der Wert dieses Knotens K, die Werte in allen Knoten des rechten Teilbaums sind größer (oder gleich) als der Wert dieses Knotens K.



Übung

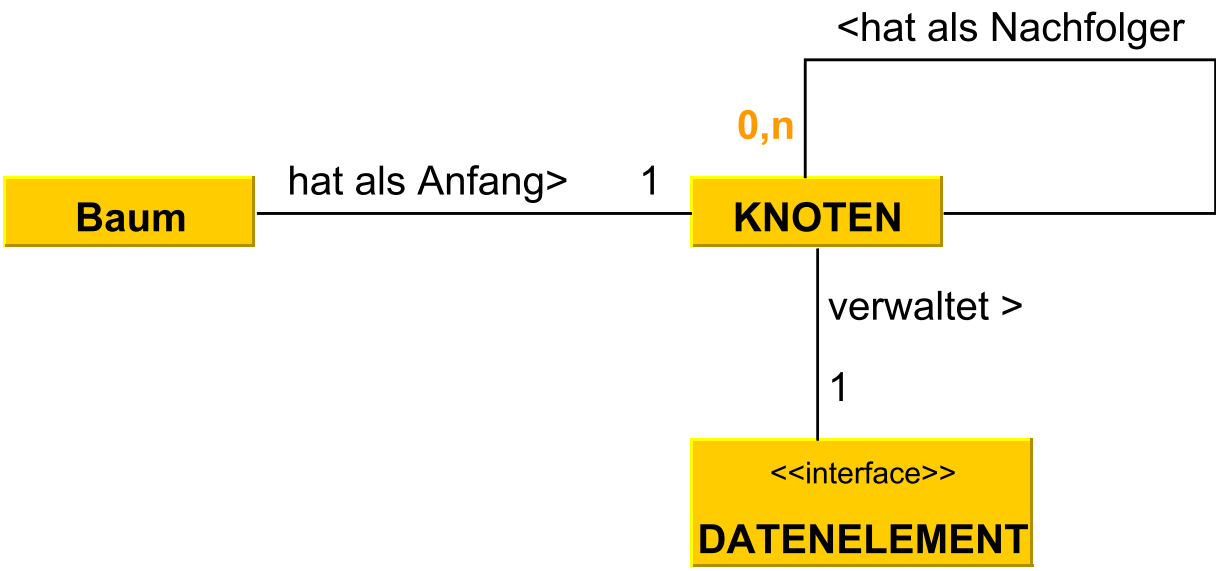
- Ü 1.1: Matheprisma: Binärbäume
  `http://www.matheprisma.uni-wuppertal.de/Module/BinSuch/index.htm`
- Ü 1.2: S. 56 / 2, 3
- Ü 1.3: S. 57 / 5, 6, 8, 9
- Ü 1.4: S. 58 / 11
- Ü 1.5: S. 67/1

Klassendiagramm Listen



- Ü6: KD allg Baum, binBaum erstellen
  `run:Arbeitsmaterial/KDListen.graphml`

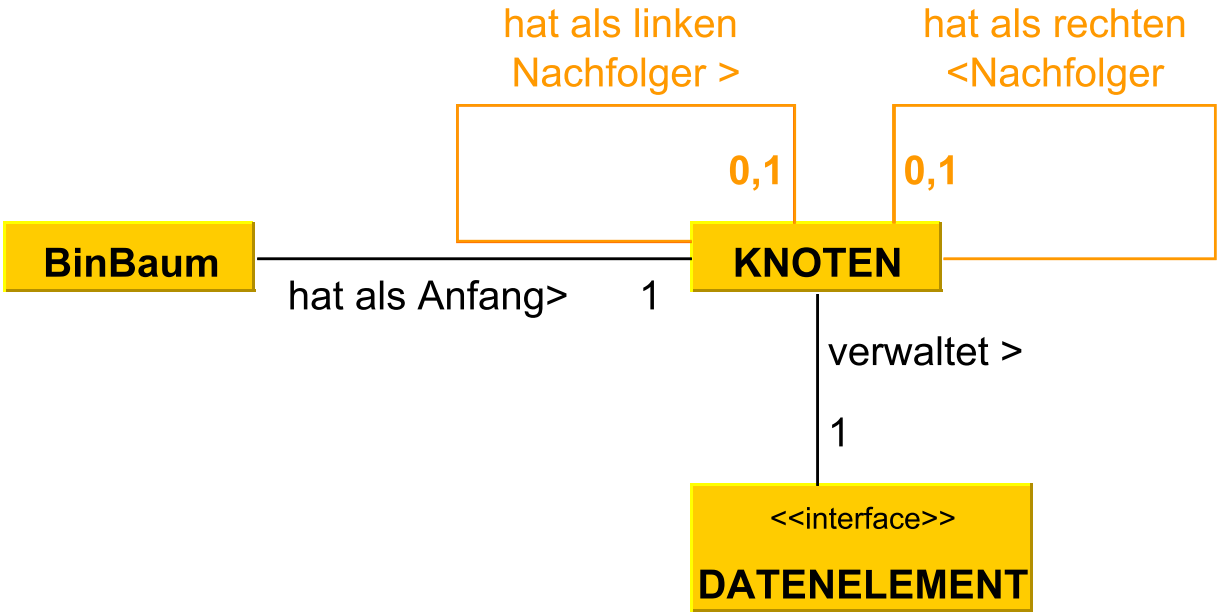
Klassendiagramm Baum



In diesem Abschnitt

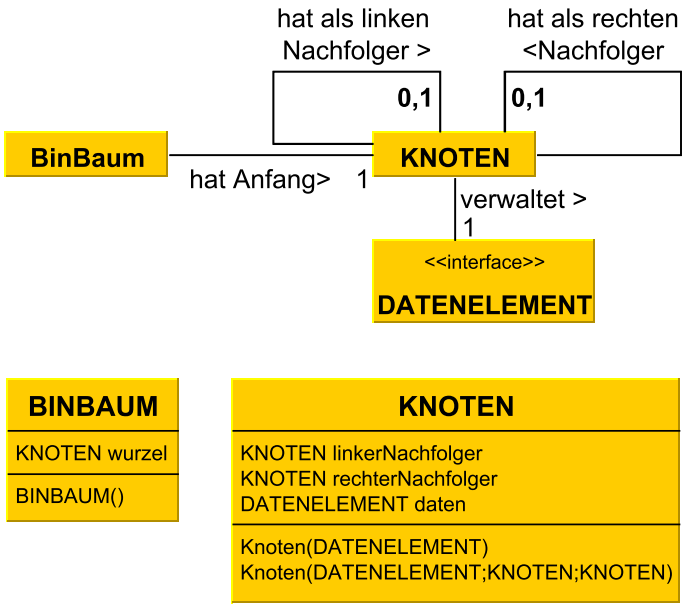
Geordnete Binärbäume
Deklaration und Initialisierung
Deklaration Suchen
Einfügen
Baum in perfekter Komposition
Baumdurchläufe

Klassendiagramm Binärer Baum



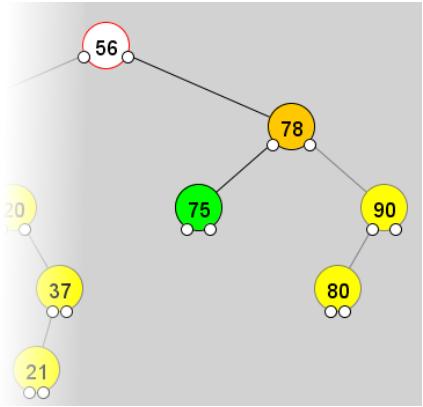
Die Trennung in linken und rechten Nachfolger macht deutlich, dass in Suchbäumen zwingend links und rechts unterschieden werden muss.

Klassenkarten



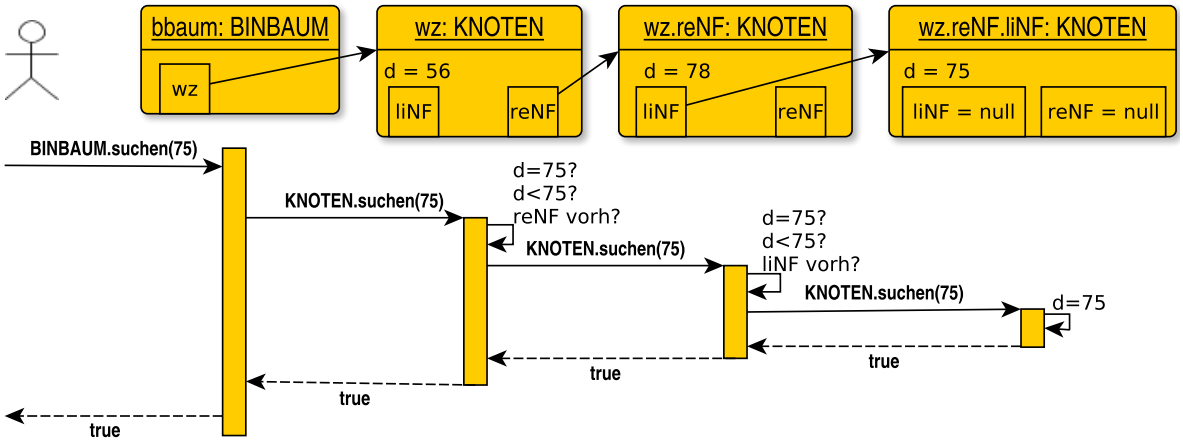
Ablaufprotokoll Suche

- bvt
  `http://www.bvt.ct-labs.de/`



S0: wurzel.suchen(75)
    Schlüssel gleich? nein
    Wo weitersuchen? rechts
    Nachfolger vorhanden? ja
    rechts weitersuchen
    S1: wurzel.reNF.suchen(75)
        Schlüssel gleich? nein
        Wo weitersuchen? links
        Nachfolger vorhanden? ja
        links weitersuchen
        S2: wurzel.reNF.liNF.suchen(75)
            Schlüssel gleich? ja
        return true
    true
true

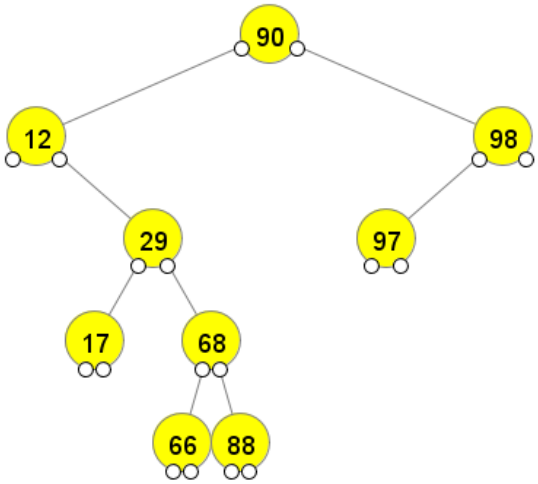
Sequenzdiagramm



Übungen

- Ü 2.1: Ablaufprotokoll und Sequenzdiagramm erstellen

Erstelle zu dem abgebildeten Baum jeweils das Ablaufprotokoll und das Squenzdiagramm für die Suche der Zahlen 29, 98 und 100



Übungen

- Ü 2.2: S. 68/2
- Ü 2.3: S. 68/3
- Ü 2.4: S. 68/4
- Ü 2.5: Deklariere die Klassen `BINBAUM` und `KNOTEN`...

KNOTEN — Deklaration und Initialisierung

```
1  class KNOTEN{
2     private KNOTEN linkerNachfolger;
3     private KNOTEN rechterNachfolger;
4     private DATENELEMENT daten;
5     KNOTEN(DATENELEMENT d){
6        linkerNachfolger = null;
7        rechterNachfolger = null;
8        daten = d;
9     }
10    KNOTEN(DATENELEMENT d, KNOTEN lnf, KNOTEN
         rnf){
11       linkerNachfolger = lnf;
12       rechterNachfolger = rnf;
13       daten = d;
14    }
15 }
```

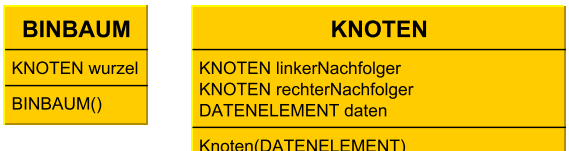

BINBAUM — Deklaration und Initialisierung

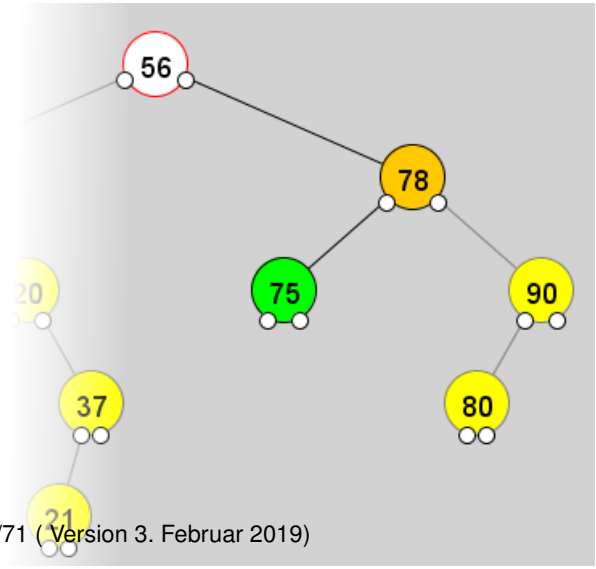
```
1  class BINBAUM{
2      private KNOTEN wurzel;
3      BINBAUM(){
4          wurzel = null;
5      }
6  }
```

In diesem Abschnitt

Geordnete Binärbäume
- Deklaration und Initialisierung
- Deklaration Suchen
- Einfügen
- Baum in perfekter Komposition
- Baumdurchläufe

BINBAUM — Was bisher geschah



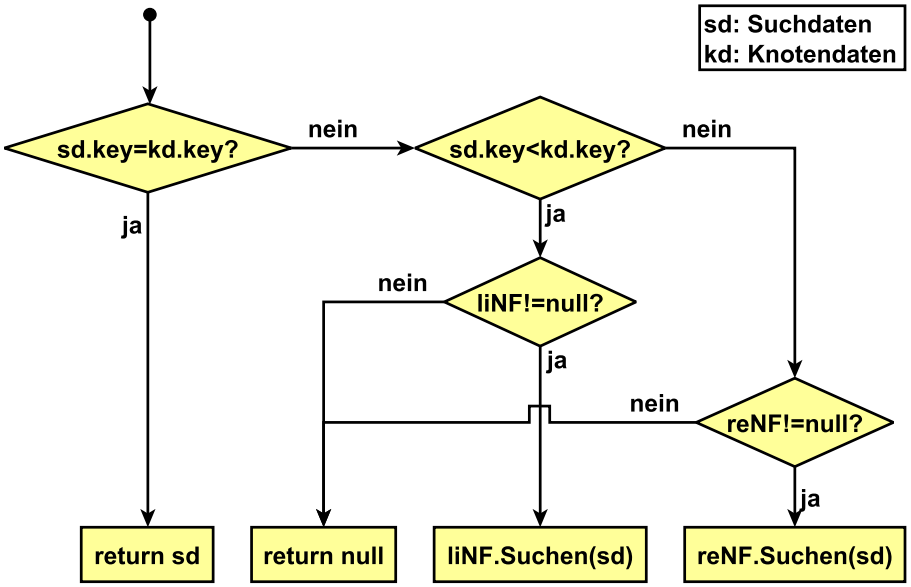


S0: wurzel.suchen(75)
- Schlüssel gleich?nein
- Wo weitersuchen?rechts
- Nachfolger vorhanden?ja
- rechts weitersuchen
- S1: wurzel.reNF.suchen(75)
  - Schlüssel gleich? nein
  - Wo weitersuchen? links
  - Nachfolger vorhanden? ja
  - links weitersuchen
  - S2: wurzel.reNF.liNF.suchen(75)
    - Schlüssel gleich? ja
  - return true
- true

true

KNOTEN — Datenflussdiagramm Suchen

- DFD entwickeln



KNOTEN — Methode Suchen in Pseudocode

```
if (sd.key=kd.key) {return sd}
else {
   if (sd.key<kd.key) {
      if (liNF!=null) {liNF.Suchen(sd)}
      else {return null}
   }
   else { //sd.key<=kd.key
      if (reNF!=null) {reNF.Suchen(sd)}
      else {return null}
   }
}
```

Übungen

- Ü 2.6: BaumcodeAnalysieren

Starte das Paket `Baum1Test` und stelle den Objektzustand der Testklasse `BAUMAUFBAU` (Rechtsklick) wieder her.

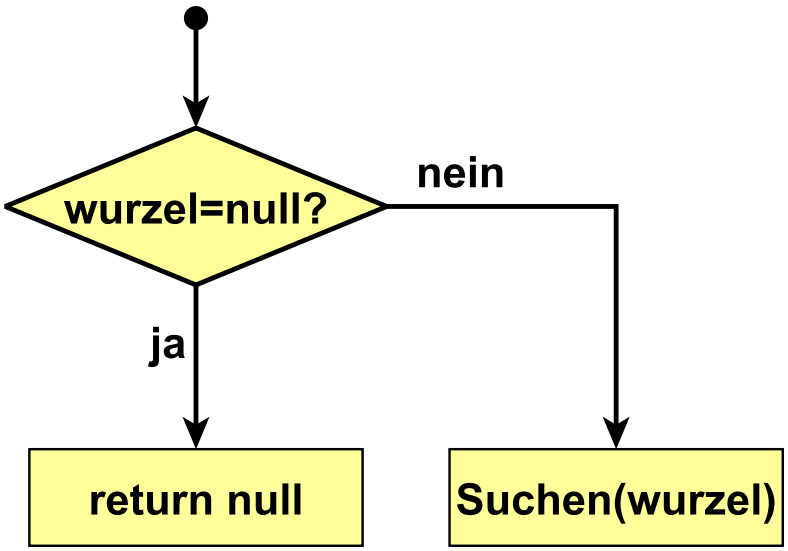
(a) Inspiziere das Objekt `baum`, erforsche so die Baumstruktur und zeichne den Baum mit yEd.

(b) Es soll in der BlueJ-Umgebung getestet werden, ob die Datenelemente 7 und 31 in dem Baum vorkommen. Was ist zu tun? In welcher Form erhält man die gewünschte Information?

(c) Vergleiche den Pseudocode für die Methode `KNOTEN.Suchen` mit dem Code, der in dem Paket `Baum1Test` implementiert ist. Wo liegen die Unterschiede? Schreibe den Code in `Baum1Test` so um, dass er genau zu dem Pseudocode passt und teste das Paket erneut.

(d) Erläutere den Quellcode `BAUM.Suchen`.

BINBAUM — Datenflussdiagramm Suchen



In diesem Abschnitt

Geordnete Binärbäume
- Deklaration und Initialisierung
- Deklaration `Suchen`
- Einfügen
- Baum in perfekter Komposition
- Baumdurchläufe

Die Methode `Einfuegen`

- Ü 2.7: Einfuegen
`run:Arbeitsmaterial/Uebungen/Baumcode/Einfuegen.pdf`

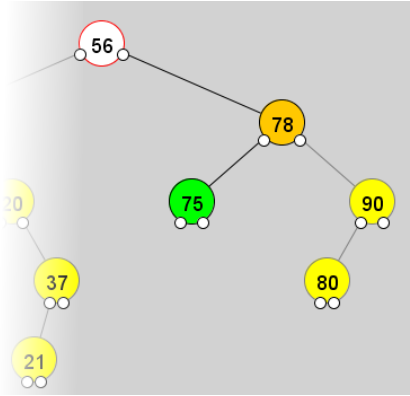
In diesem Abschnitt

Geordnete Binärbäume
- Deklaration und Initialisierung
- Deklaration Suchen
- Einfügen
- Baum in perfekter Komposition
- Baumdurchläufe

Ausgangspunkt: Unübersichtlicher Programmcode „Suchen“

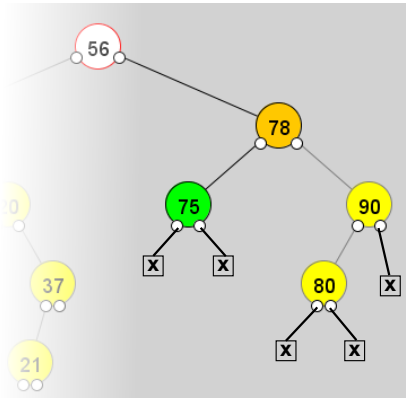
```
DATENELEMENT Suchen (DATENELEMENT wert)
{
    if (daten. Vergleichen (wert) == 0)
    {
        return daten;
    }
    else
    {
        if (daten. Vergleichen (wert) < 0)
        {
            if (rechterNachfolger == null)
            {
                return null;
            }
            else
            {
                return rechterNachfolger.
                    Suchen (wert);
            }
        }
        else
        {
            if (linkerNachfolger == null)
            {
                return null;
            }
            else
            {
                return linkerNachfolger.
                    Suchen (wert);
```

Ursache der Fallunterscheidungen



Knoten müssen keine Nachfolger haben

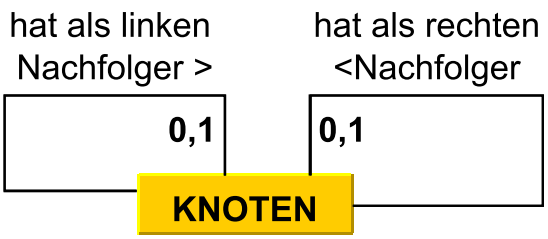
Idee: Nachfolger erzwingen



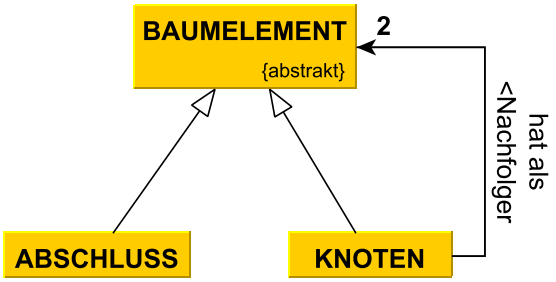
Problem:
Die Baumelemente sind keine Konten mehr sondern Knoten oder Abschlusskappen

Die Klasse „Baumelement“

Bisher

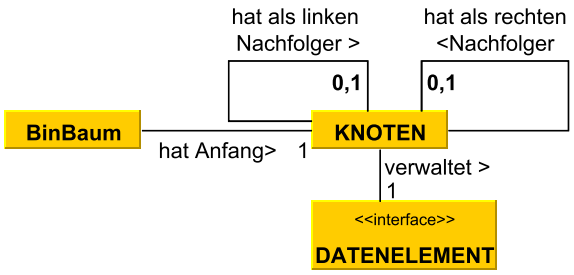


Jetzt

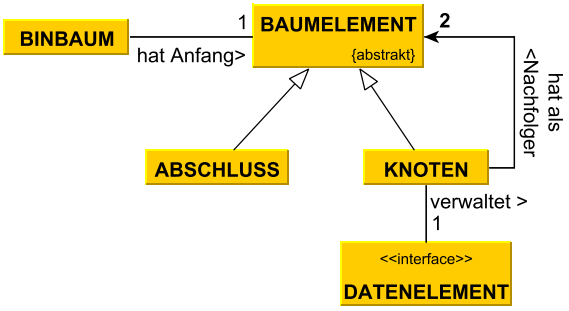


Binbaum in Kompositum

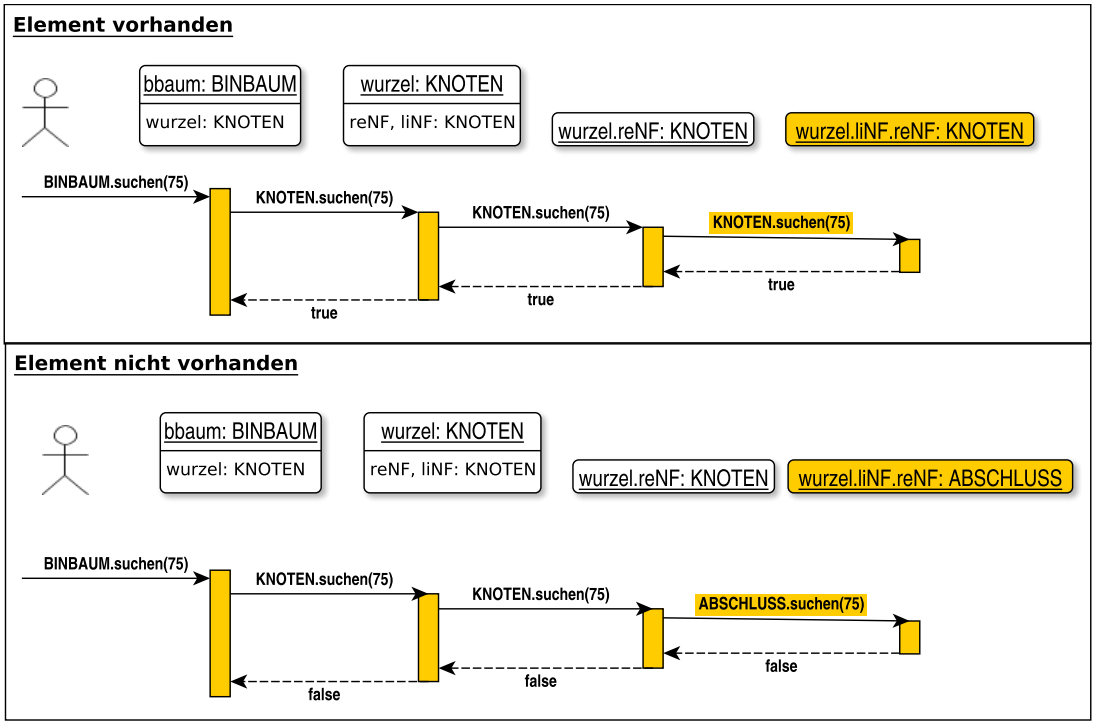
Bisher



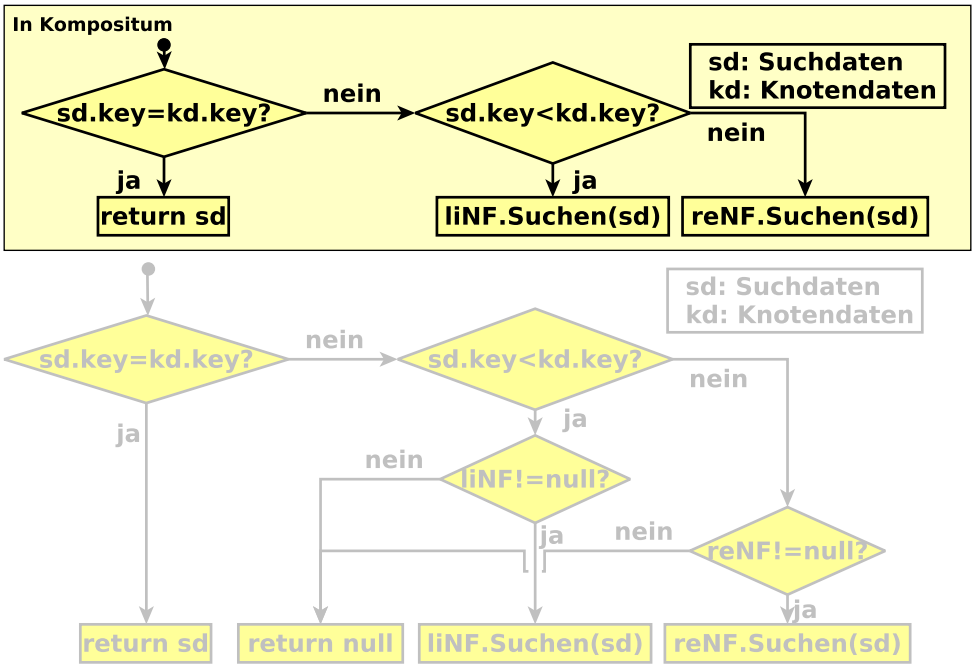
Jetzt



Suchen in Kompositum — Sequenzdiagramm



KNOTEN.Suchen in Kompositum — DFD



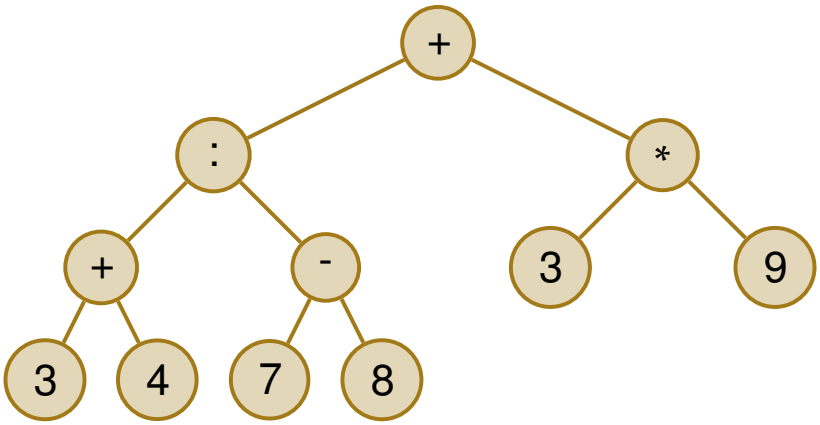
Übung

- Ü 2.8: Kompositum
  `run:Arbeitsmaterial/Uebungen/Kompositum/Kompositum.pdf`

In diesem Abschnitt

Geordnete Binärbäume
- Deklaration und Initialisierung
- Deklaration `Suchen`
- Einfügen
- Baum in perfekter Komposition
- Baumdurchläufe

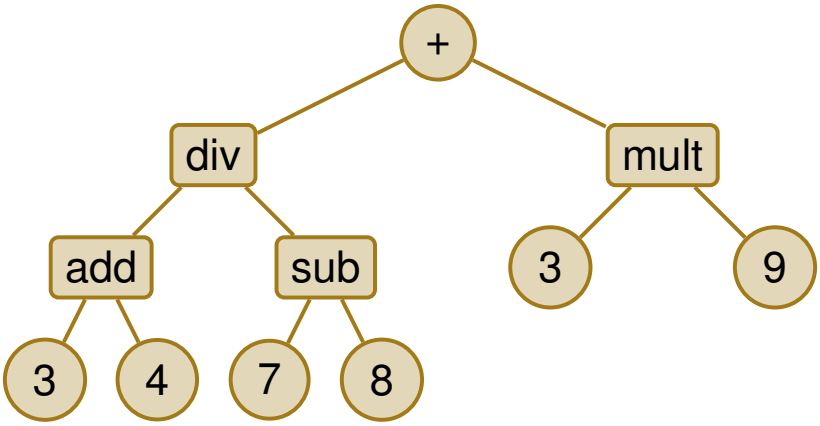
Einführung—Rechenbäume



$$[(3+4):(7-8)]+(3\cdot 9)$$

rekursive Struktur inorder: linker Term — Operand — rechter Term

Einführung—Rechenbäume funktional betrachtet



$$\mathrm{add}(\mathrm{div}(\mathrm{add}(3,4),\mathrm{sub}(7,8)),\mathrm{mult}(3,9))$$

kürzer add div add 3 4 sub 7 8 mult 3 9

rekursive Struktur preorder: Operand — linker Term — rechter Term

Baumdurchlauf

Der Baumdurchlauf dient dazu, alle Elemente des Baums systematisch zu besuchen. Es gibt drei möglichen Durchlaufstrategien.

preorder	Datenel.	<T-Baum li>	<T-Baum re>
inorder	<T-Baum li>	Datenel.	<T-Baum re>
postorder	<T-Baum li>	<T-Baum re>	Datenel.

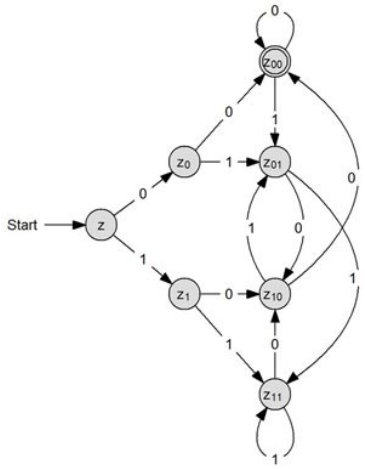
- Ü 2.9: Baumdurchläufe
`run:Arbeitsmaterial/Uebungen/Baumdurchlauf/Baumdurchlauf.pdf`

In diesem Abschnitt

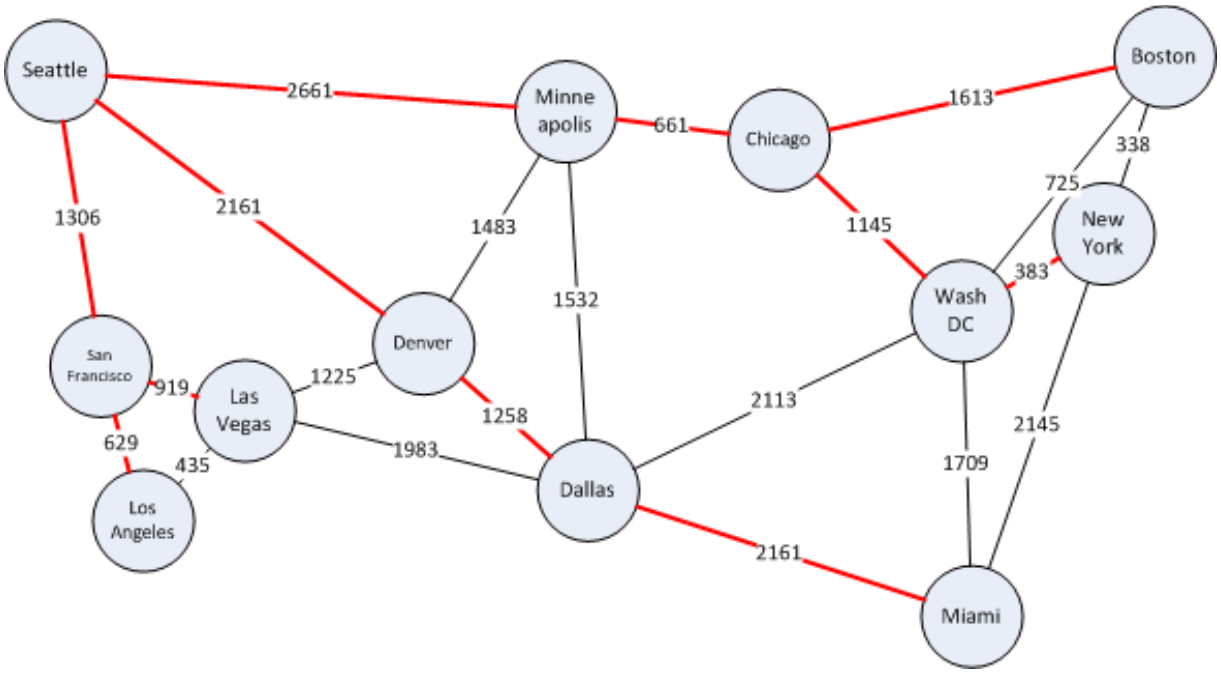
Die Datenstruktrur Graph
Kennzeichen von Graphen
Darstellung von Graphen
Implementation
Graphendurchlauf

Anwendungen und Definition

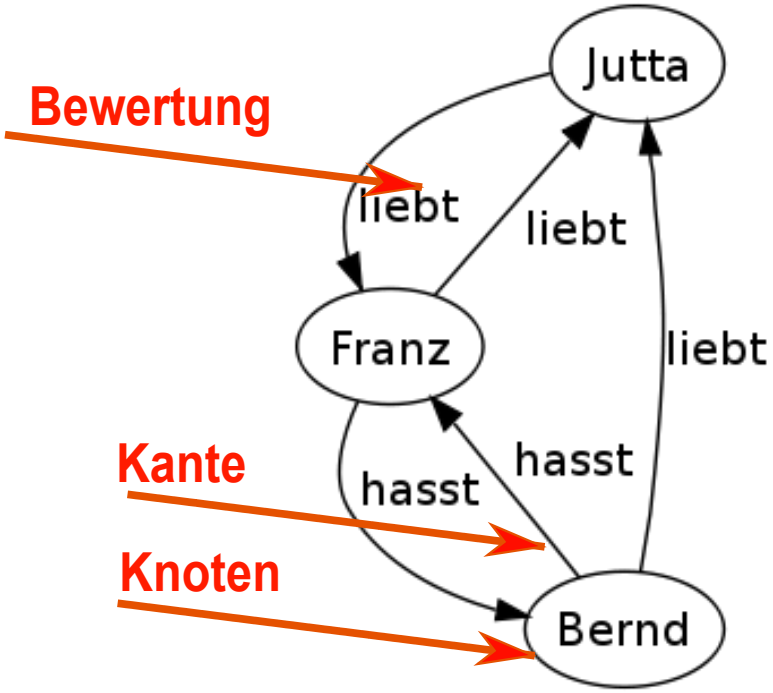
Listen und Bäumen modellieren hierarchische Strukturen. Jetzt sollen netzartige Objekte modelliert werden.



Bewerteter Graph



Gerichteter Graph



Definition

Graph

Ein Graph besteht aus einer endlichen Menge von Knoten und Kanten. Jede Kante verbindet genau zwei Knoten. Die Kanten können

- gerichtet oder ungerichtet sein,
- eine Bewertung haben oder nicht.

Typische Fragestellungen

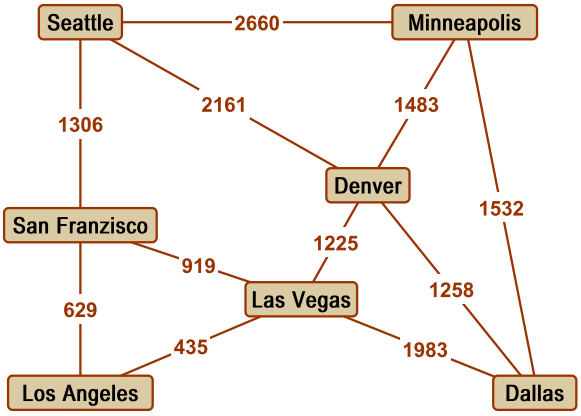
- Wie kann eine Netzstruktur modelliert werden?
- Wie ist es möglich, alle Knoten eines Graphen systematisch zu besuchen?
- Welche Wege gibt es von einem vorgegebenen Startknoten zu einem Zielknoten?
- Wie kann ein bestimmter Knoten im Netz gefunden werden?

• Ü 3.1: S.100/5,6

In diesem Abschnitt

Die Datenstruktrur Graph
Kennzeichen von Graphen
Darstellung von Graphen
Implementation
Graphendurchlauf

Adjazenzmatrix



↗	Da	De	LV	LA	Mi	SF	Se
Da	0	1258	1983		1532		
De	1258	0	1225		1483		2161
LV	1983	1225	0	435		919	
LA			435	0		629	
Mi	1532	1483			0		2660
SF			919	629		0	1306
Se		2161			2660	1306	0

Leerstellen müssen mit unverwechselbaren Werten belegt werden, z.B. -1 oder ∞.

Adjazenzmatrix

Adjazenzmatrix

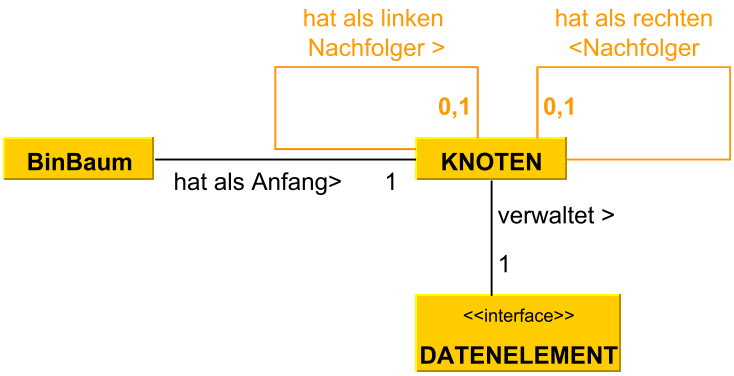
Die Adjazenzmatrix eines Graphen ist eine quadratische Matrix (Tabelle), deren Anzahl an Spalten bzw. Zeilen mit der Anzahl der Knoten des Graphen übereinstimmt. Die Einträge in den Zellen der Matrix geben wieder, ob zwischen zwei Knoten eine Kante existiert und welche Bewertung diese gegebenenfalls hat.

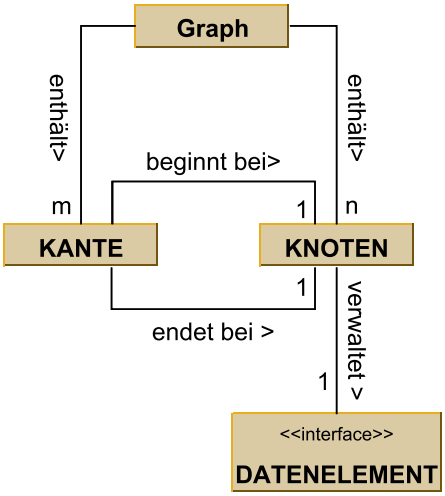
Bemerkungen

- Für unbewertete Graphen reichen boolsche Werte in der Adjazenzmatrix.
- Für einen ungerichteten Graphen ist die Adjazenzmatrix symmetrisch.

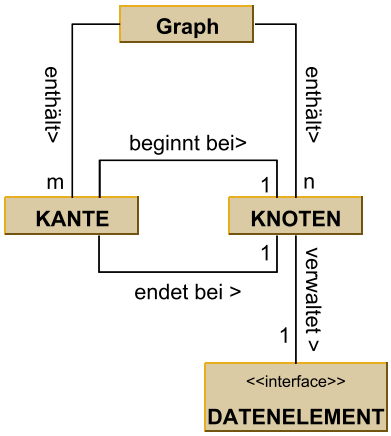
• Ü 3.2: S. 108/2,3

Klassendiagramm





Klassenkarten eines Graphen



Graph
int knotenzahl Konten[ ] knotenliste int[ ][ ] adjazenzmatrix
Graph(maxKnotenzahl

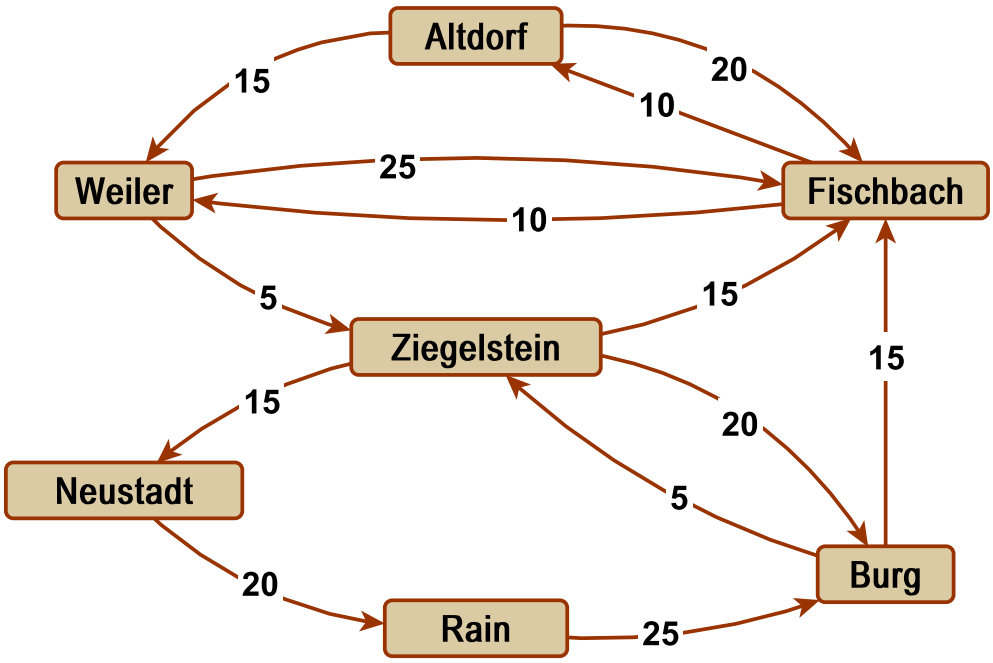
Knoten
Datenelement daten
KNOTEN(Datenelement inhalt)

Datenelement
ortsname uebernachtungsplaetze sehenswuerdigkeiten
Datenelement(ortsname, uebernachtungsplaetze, sehenswuerdigkeiten)

In diesem Abschnitt

Die Datenstruktrur Graph
Kennzeichen von Graphen
Darstellung von Graphen
Implementation
Graphendurchlauf

Fahrradwegeplan



Radwegeplan—Knoten indizieren

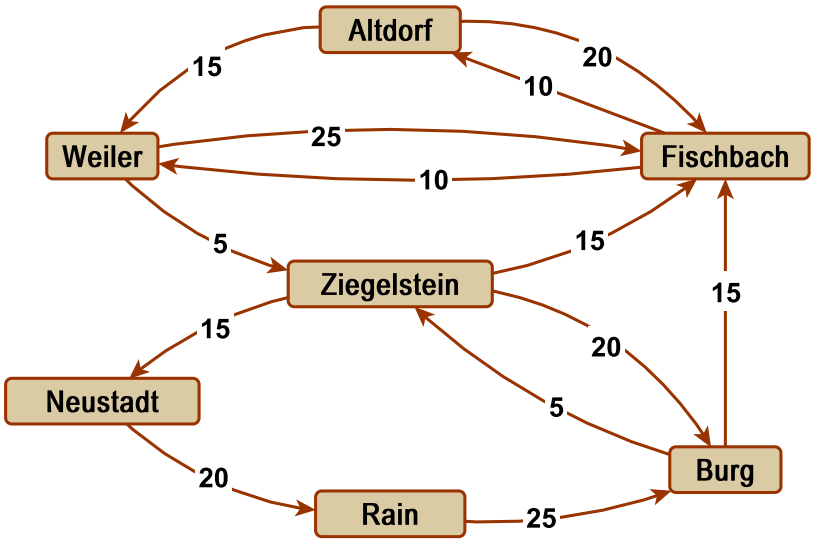


Tabelle:
Knotenindizierung

Altdorf	0
Fischbach	1
Weiler	2
Ziegelstein	3
Burg	4
Neustadt	5
Rain	6

Radwegeplan — Adjazenzmatrix erstellen

		Altdorf	Fischbach	Weiler	Ziegelstein	Burg	Neustadt	Rain
	↗	0	1	2	3	4	5	6
Altdorf	0	0	20	15	-1	-1	-1	-1
Fischbach	1	10	0	10	-1	-1	-1	-1
Weiler	2	-1	25	0	5	-1	-1	-1
Ziegelstein	3	-1	15	-1	0	20	15	-1
Burg	4	-1	15	-1	5	0	-1	-1
Neustadt	5	-1	-1	-1	-1	-1	0	20
Rain	6	-1	-1	-1	-1	25	-1	0

Radwegplan — lästige Basisimplementationen Datenelement—Attribute

```
class DATENELEMENT
{
    private String ortsname;
    private int uebernachtungsplaetze;
    private String sehenswuerdigkeiten;
```


Radwegplan — lästige Basisimplementationen Datenelement—Konstruktor

```
    DATENELEMENT(String ortsname, int
       uebernachtungsplaetze, String
       sehenswuerdigkeiten )
    {
        this.ortsname = ortsname;
        this.uebernachtungsplaetze =
           uebernachtungsplaetze;
        this.sehenswuerdigkeiten =
           sehenswuerdigkeiten;
    }
```

Radwegplan — lästige Basisimplementationen Datenelement—Ausgabe

```
1     void ausgeben()
2     {
3         System.out.println( " Im Ort " +
              ortsname + " gibt es " +
              uebernachtungsplaetze + " Betten.
               ");
4         System.out.println( " Sie können
              hier folgendes erleben: " +
              sehenswuerdigkeiten );
5     }
```


Radwegplan — lästige Basisimplementationen Knoten

```
1  class KNOTEN
2  {
3      private DATENELEMENT daten;
4
5      KNOTEN(DATENELEMENT inhalt)
6      {
7          daten = inhalt;
8      }
9
10     DATENELEMENT datenLiefern()
11     {
12         return daten;
13     }
14 }
```

Radwegplan — Klasse Graph.Attribute

```
1 class GRAPH
2 {
3     private KNOTEN[] knotenfeld;
4     private int[][] adjazenzmatrix;
5     private int anzahlKnoten = 0;
```


Radwegplan — Klasse Graph.Konstruktor

```
1     GRAPH(int maxAnzahlKnoten)
2     {
3         knotenfeld = new KNOTEN[
              maxAnzahlKnoten];
4         adjazenzmatrix = new int[
              maxAnzahlKnoten][maxAnzahlKnoten
              ];
```

Radwegplan — Graph.Konstruktor.Adjazenzmatrix

```
1   for(int i=0; i < maxAnzahlKnoten; i=i+1)
2   {
3       for(int j=0; j < maxAnzahlKnoten; j=j+1)
4       {
5           if(i == j)
6           {
7               adjazenzmatrix[i][j] = 0;
8           }
9           else
10          {
11              adjazenzmatrix[i][j] = -1;
12          }
```


Übung

- Ü 3.3: Graphenimplementation

`run: Arbeitsmaterial/Uebungen/GraphenImplementation/GraphenImplementation.pdf`

In diesem Abschnitt

Die Datenstruktrur Graph
- Kennzeichen von Graphen
- Darstellung von Graphen
- Implementation
- Graphendurchlauf

Typische Fragestellungen

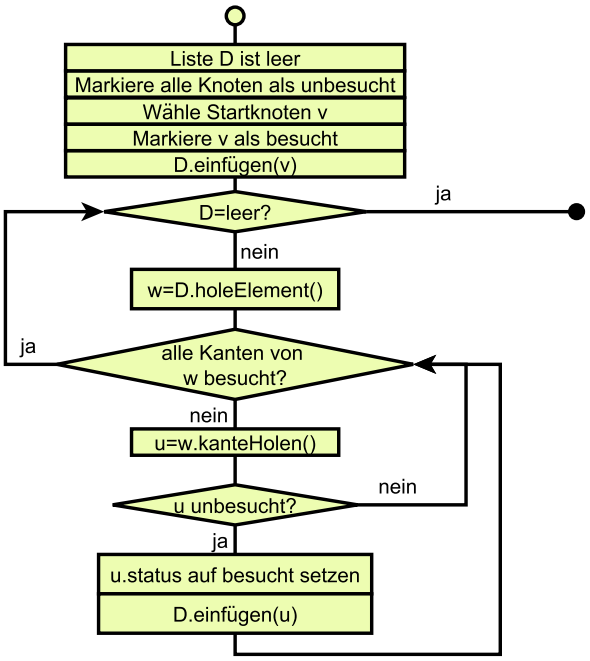
Gesucht ist ein Weg von A nach B der

- möglichst kurz ist,
- möglichst wenige Zwischenknoten enthält,
- jeden anderen Knoten mindestens einmal besucht.

Breiten- und Tiefensuche

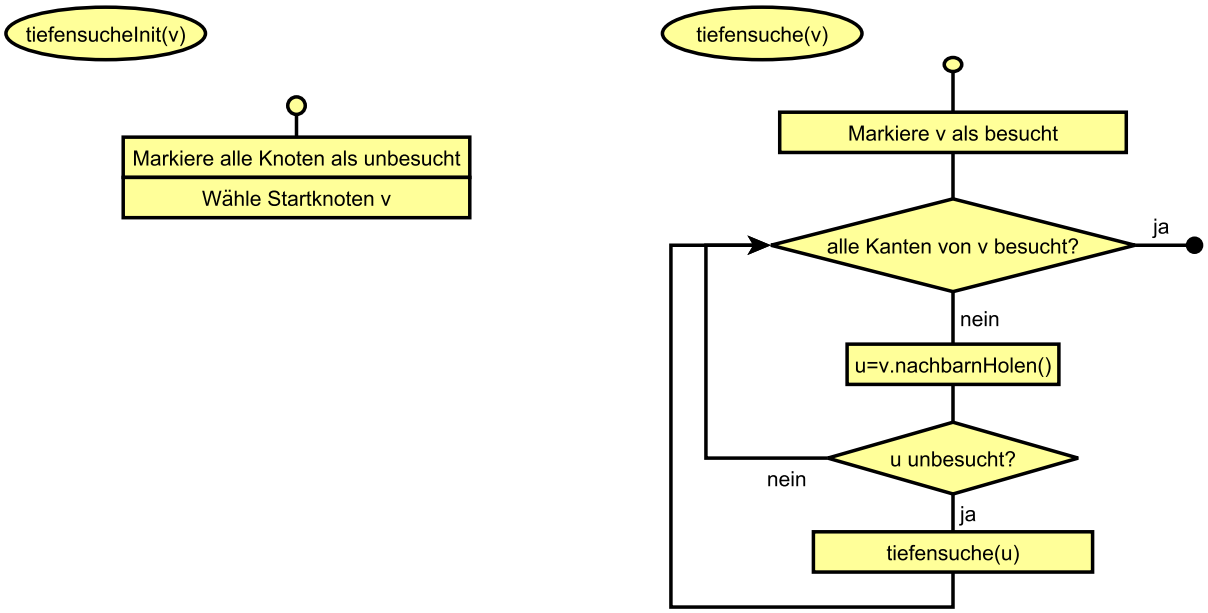
- Lernprogramm Pirsma S. 1-6
`http://www.matheprisma.uni-wuppertal.de/Module/Graphen/index.htm`
- Ü 3.4: Graphendurchläufe
`run:Arbeitsmaterial/Uebungen/Graphendurchlauf/Graphendurchlauf.pdf`

Suchalgorithmen implementieren (MathePrisma-Alogrithmus)



Abhängig davon, ob die Liste `D` als FIFO- oder FILO-Speicher abgearbeitet wird erhält man einen Tiefen- bzw. Breitensuchalgorithmus.

Tiefensuche rekursiv implementieren



Methode `tiefensucheInit` — Pseudocode

```
Methode tiefensucheStarten(startKnotenNr)
  wiederhole für alle Knoten
    setze Markierung des Knotens auf den
       Wert für unbesucht
  endeWiederhole
  wenn startKnotenNr >=0 und startKnotenNr <
      anzahlKnoten dann
    tiefensucheDurchfuehren(startKnotenNr)
  endeWenn
```

Methode `tiefensuche` — Hauptteil(Pseudocode)

```
Methode tiefensucheDurchfuehren(knotenNr)
  Markiere den Knoten mit der Nummer knotenNr als besucht
  //Gib evtl. Informationen über den Knoten aus
  //Tiefensuche für den nächsten direkt erreichbaren unbesuchten Knoten aufrufen
  wiederhole für j = 0 bis j < anzahlKnoten
    wenn Knoten mit der Nummer j direkt erreichbarer und nicht besucht ist dann
      tiefensucheDurchfuehren(j) // rekursiver Aufruf
    endeWenn
  endeWiederhole
```


Übung

- Ü 3.5: Tiefensuche implementieren
  `run:Arbeitsmaterial/Uebungen/GraphendurchlaufImplementation/GraphendurchlaufImplementation.pdf`