INFORMATIK Oberstufe

Die Datenstruktur Graph

Ulli Freiberger

Hinweis

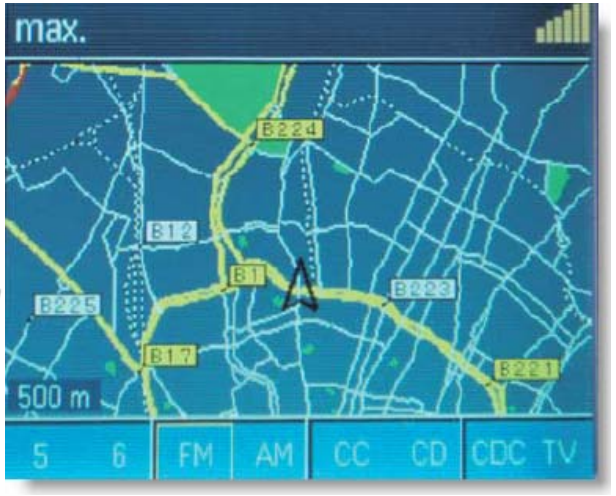
Die Abbildungen sind dem Buch Informatik Oberstufe 1 entnommen und unterliegen dem Copyright des Oldenbourg Schulbuchverlags!

Kennzeichen von Graphen

Themenzentrierter Ansatz

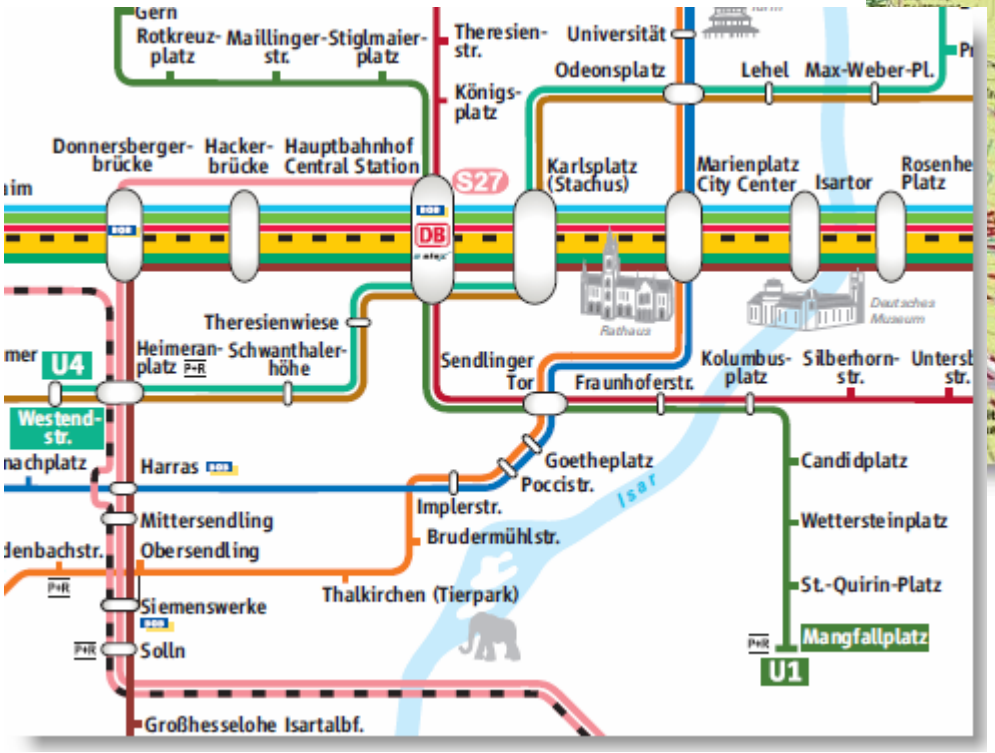
Leitbeispiel Navigationssystem - Autobahnnetz

Schrittweise Einführung in die Algorithmen auf der Datenstruktur Graph durch praxisnahe, immer tiefergehende Aufgabenstellungen



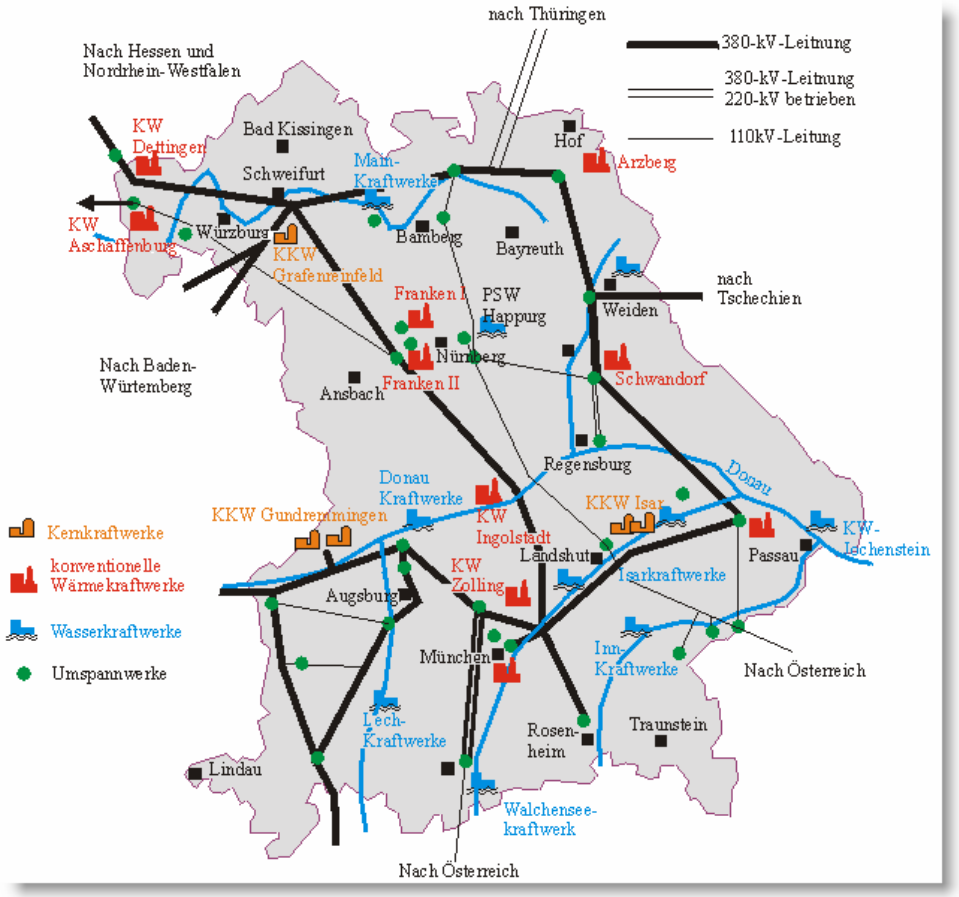
Kennzeichen von Graphen

- Streckennetze: Straßen-, U-, S-Bahnen;
  Zug-, Flugverbindungen; Wander-, Radwege
- Daten-, Telefonkabelnetze
- Abwasserkanäle einer Stadt
- Stromleitungen ...

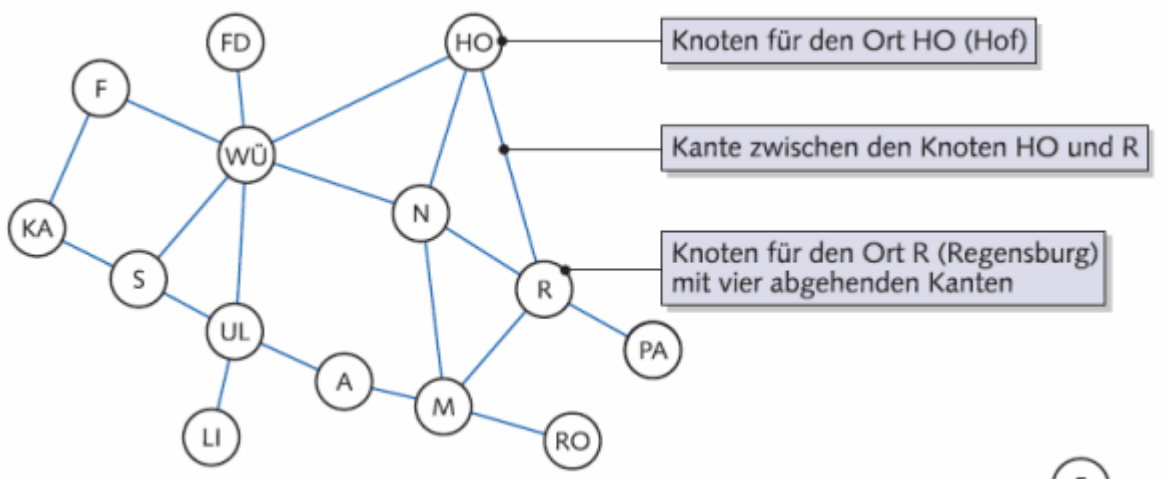


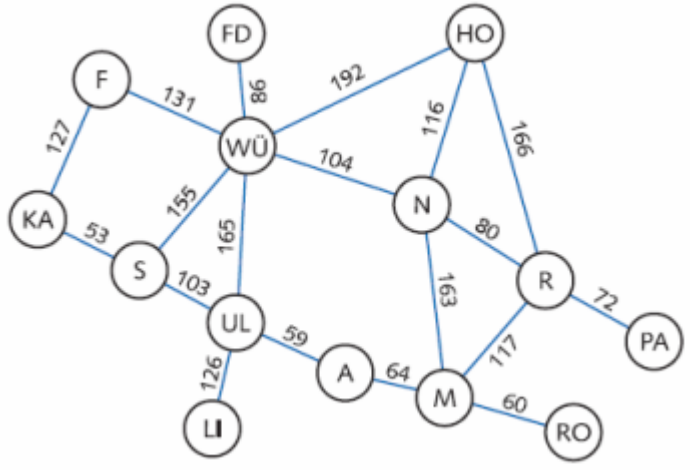
Kennzeichen von Graphen

- Streckennetze: Straßen-, U-, S-Bahnen;
  Zug-, Flugverbindungen; Wander-, Radwege
- Daten-, Telefonkabelnetze
- Abwasserkanäle einer Stadt
- Stromleitungen ...



Kennzeichen von Graphen

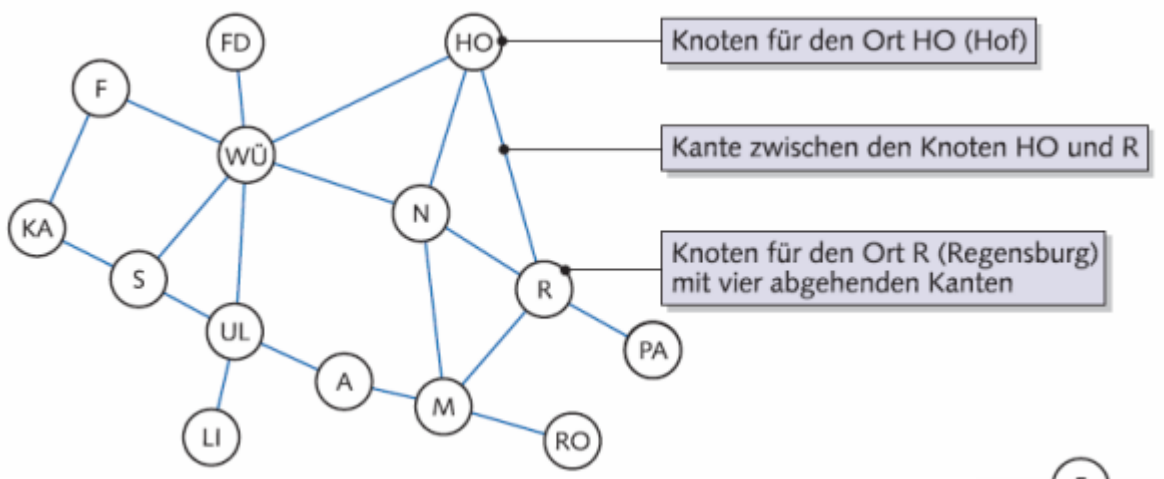


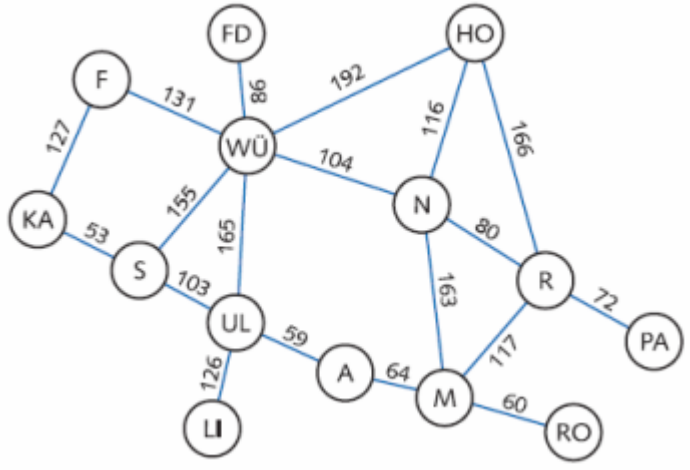


Begriffe am Leitbeispiel erarbeiten:

- endliche Menge von **Knoten**
- endliche Menge von **Kanten**
- eine Kante verbindet genau zwei Knoten

Kennzeichen von Graphen

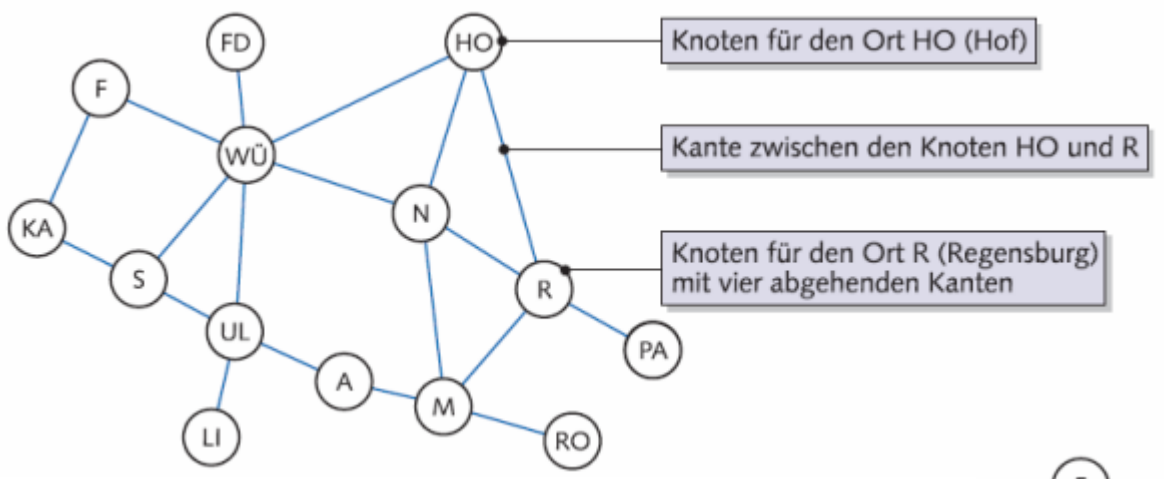




Begriffe am Leitbeispiel erarbeiten:

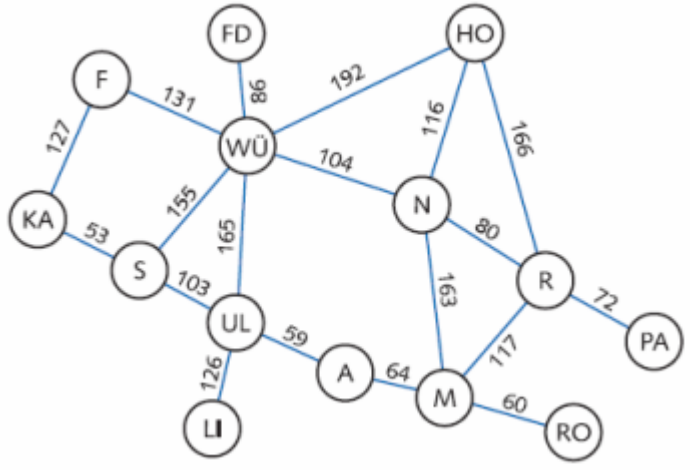
- **Pfad (Weg)**
- **einfacher Pfad**
- **Zyklus**
- **zusammenhängender Graph**

Kennzeichen von Graphen

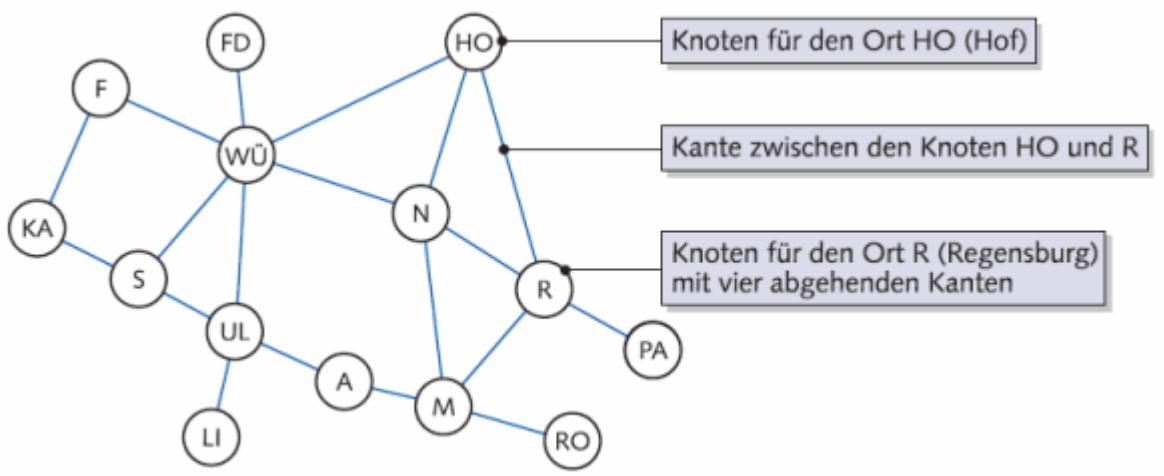


Begriffe am Leitbeispiel erarbeiten:

- Kanten mit **Gewicht** (Bewertung)
- **gewichteter Graph**
- **Länge** eines Wegs

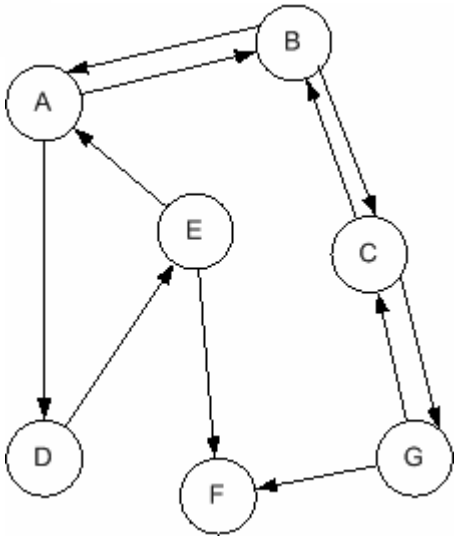


Kennzeichen von Graphen



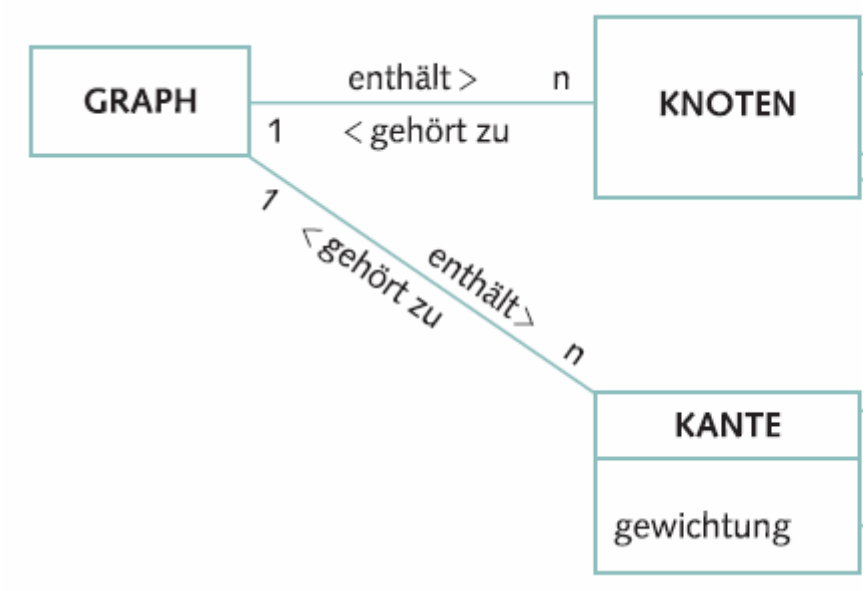
Begriffe am Leitbeispiel erarbeiten:

- **gerichteter Graph**
- **Vorgängerknoten - Nachfolgerknoten**



Kennzeichen von Graphen

Klassendiagramm



Analyse der Struktur

Kennzeichen von Graphen

Praxisnahe Aufgaben: Information, Analyse und grafische Darstellung

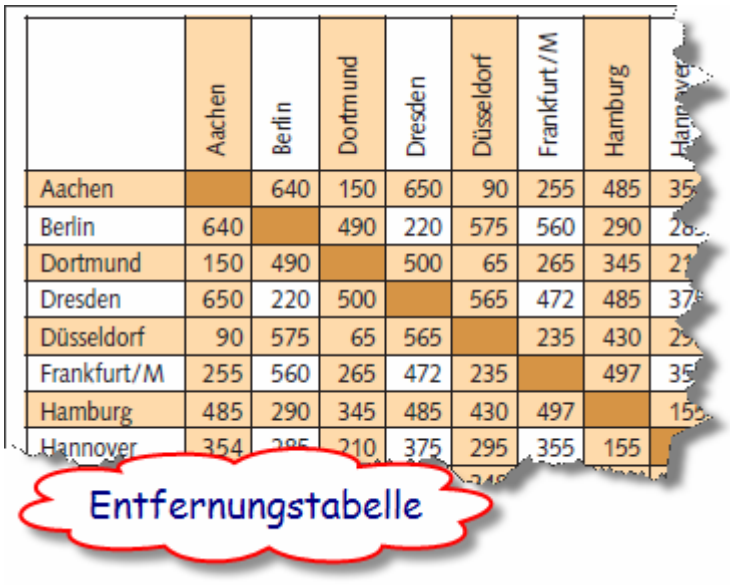
1 Fluglinie
Informieren Sie sich im Internet über die Flugrouten einer (nicht zu kleinen) Fluggesellschaft. Im Folgenden werden nur ihre internationalen Flugverbindungen und die nationalen Routen zwischen den Abflugorten ins Ausland betrachtet.
a Erstellen Sie einen ungerichteten, unbewerteten Graphen für die Flugrouten. A

2 ICE-Verbindungen
Erkundigen Sie sich im Internet nach den ICE-Verbindungen innerhalb Deutschlands.
a Erstellen Sie einen gewichteten, ungerichteten Graphen für die ICE-Verbindungen. Als Knotenbezeichner verwenden Sie die Kfz-Kennzeichen der Bahnhofsor-

3 Autobahnen
Nehmen Sie eine Fernstraßenkarte von Deutschland zur Hand. Erstellen Sie für einen Ausschnitt daraus (circa 30 Knoten) einen gewichteten, ungerichteten Graphen für die Autobahnverbindungen. Als Knotenbezeichner verwenden Sie die Kfz-Kennzeichen

4 S- und U-Bahnen im Verkehrsverbund München
Informieren Sie sich über den Streckenverlauf der S- und U-Bahnen und die angefahrenen Bahnhöfe.
Hinweis: Diese Aufgabe wird im Projekt „Verbindungsauskunft“ in Kapitel 18 wei-

Darstellung von Graphen



	Aachen	Berlin	Dortmund	Dresden	Düsseldorf	Frankfurt/M	Hamburg	Hannover
Aachen		640	150	650	90	255	485	35
Berlin	640		490	220	575	560	290	28
Dortmund	150	490		500	65	265	345	21
Dresden	650	220	500		565	472	485	37
Düsseldorf	90	575	65	565		235	430	29
Frankfurt/M	255	560	265	472	235		497	35
Hamburg	485	290	345	485	430	497		15
Hannover	354	28	210	375	295	355	155	

Entfernungstabelle

	A	F	FD	HO	KA	LI	M	N	PA	R
A	0						64			
F		0			127					
FD			0							
HO				0				116		166
KA		127			0					
LI						0				
M	64						0	163		117
N				116			163	0		80
PA									0	72
R				166			117	80	72	0

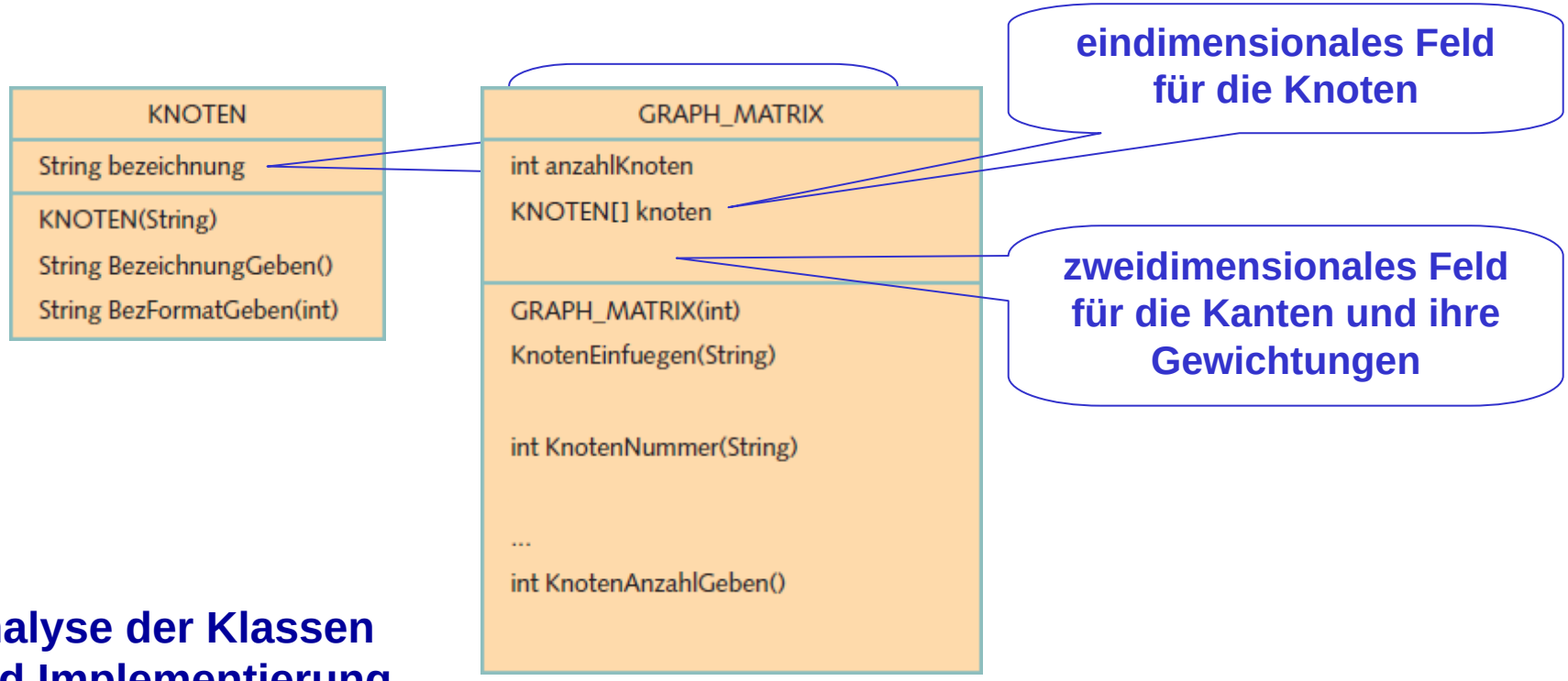
Adjazenzmatrix

Adjazenzmatrix

Darstellungsform (Repräsentation) für den Graphen

Darstellung von Graphen

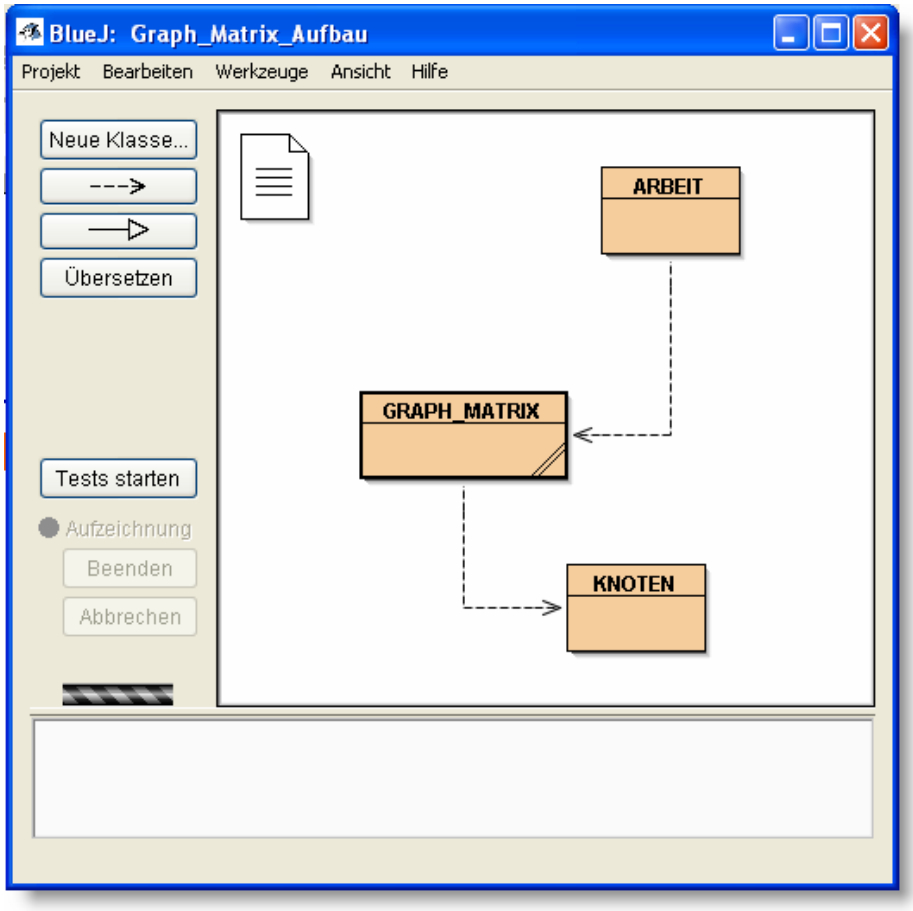
Klassen für die Implementierung mit Adjazenzmatrix



Analyse der Klassen und Implementierung

Darstellung von Graphen

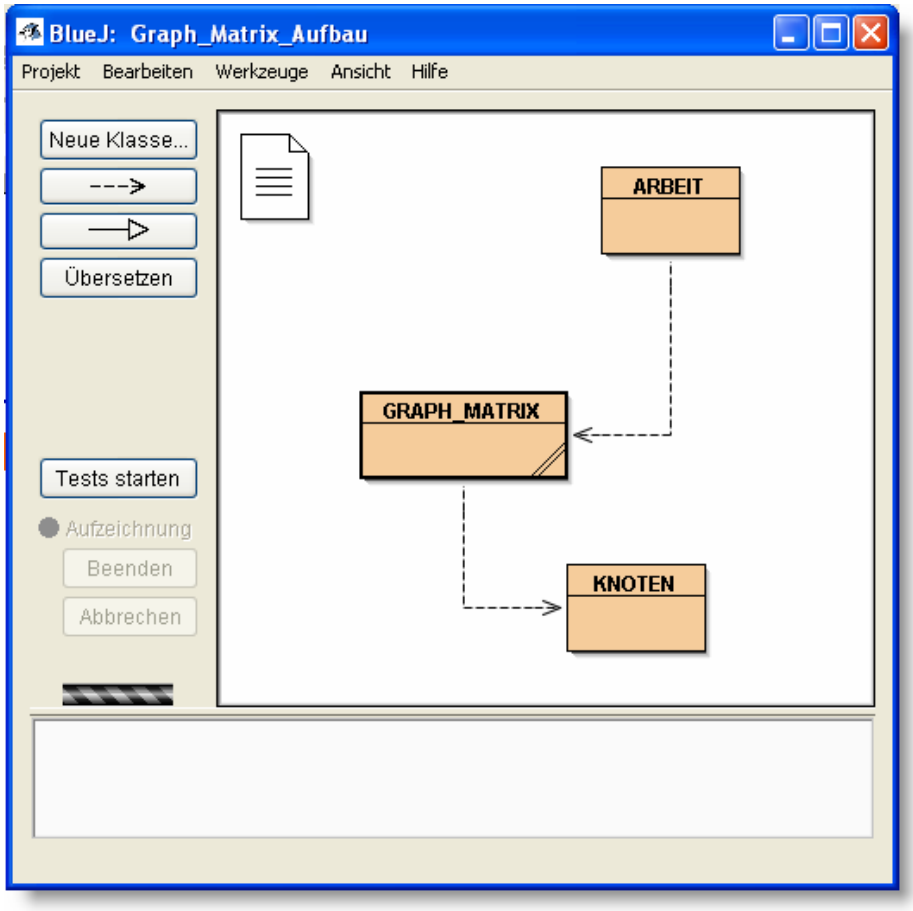
BlueJ - Implementierung mit Adjazenzmatrix




```
public class KNOTEN
{
    private String bezeichnung;

    /**
     * Konstruktor für Objekte der Klasse KNOTEN
     */
    public KNOTEN(String neuerWert)
    {
        bezeichnung = neuerWert;
    }

    /**
     * Gibt den Bezeichner des Knotenobjekts zurück
     *
     * @return  Bezeichner
     *
     */
    public String BezeichnungGeben()
    {
        return bezeichnung;
    }
```

Darstellung von Graphen

BlueJ - Implementierung mit Adjazenzmatrix




```
public class GRAPH_MATRIX
{
    // aktuelle Knotenanzahl
    private int anzahlKnoten;

    // Feld der Knoten des Graphen
    private KNOTEN[] knoten;

    // 2-dim Feld der Adjazenzmatrix
    private int[][] matrix;

    /**
     * Konstruktor für Objekte der Klasse GRAPH_MATRIX
     * Die maximale Anzahl der Knoten wird dabei festgelegt
     *
     * @param   maximaleKnoten   Anzahl der maximal möglichen Knoten
     *
     */
    public GRAPH_MATRIX(int maximaleKnoten)
    {
        anzahlKnoten = 0;
        knoten = new KNOTEN[maximaleKnoten];
        matrix = new int[maximaleKnoten][maximaleKnoten];
    }
```

Darstellung von Graphen

BlueJ - Implementierung mit Adjazenzmatrix

Klasse GRAPH_MATRIX
Methode KnotenEinfuegen(String bezeichner)

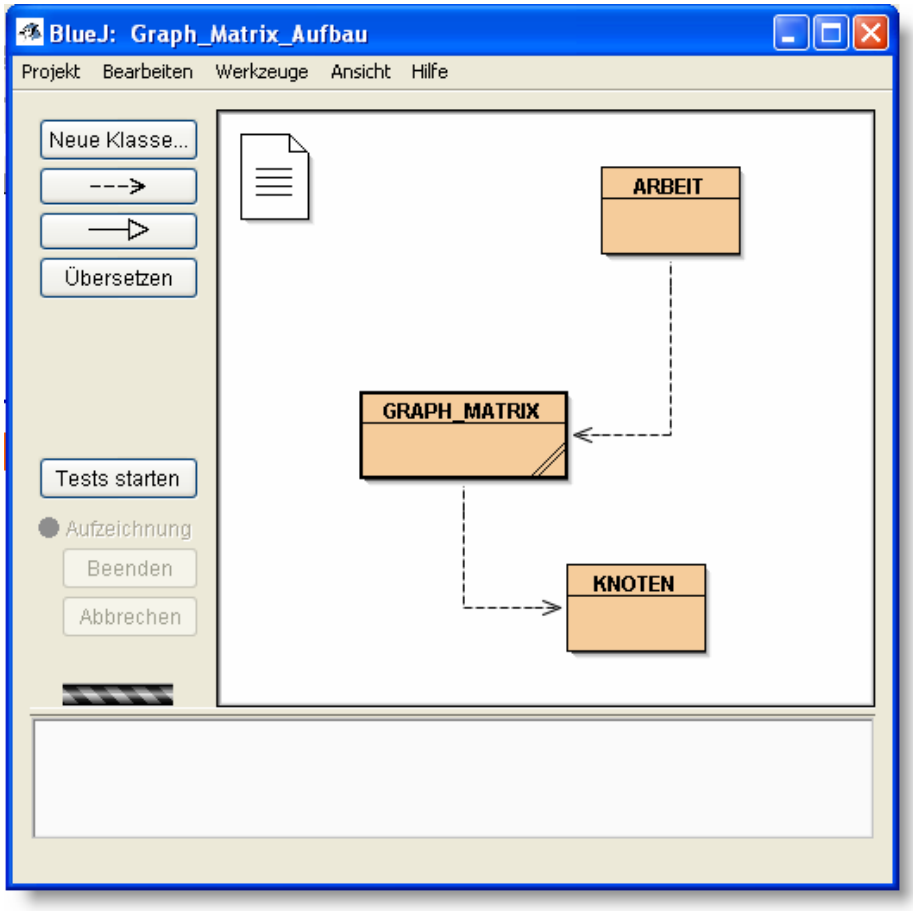
```
wenn anzahlKnoten < knoten.Feldlänge
dann
   knoten[anzahlKnoten] = new KNOTEN(bezeichner)
   matrix[anzahlKnoten][anzahlKnoten] = 0
   zähle i von 0 bis anzahlKnoten-1
      //
      mat
      mat
   endezä
   anzahl
endewenn
```

Klasse GRAPH_MATRIX
Methode KanteEinfuegen(String von, String nach, int gewichtung)

```
int vonNummer, nachNummer

vonNummer = KnotenNummer(von)
nachNummer = KnotenNummer(nach)
wenn (vonNummer != -1) und (nachNummer != -1) und (vonNummer != nachNummer)
dann
   matrix[vonNummer][nachNummer] = gewichtung
   matrix[nachNummer][vonNummer] = gewichtung
endewenn
```

Darstellung von Graphen

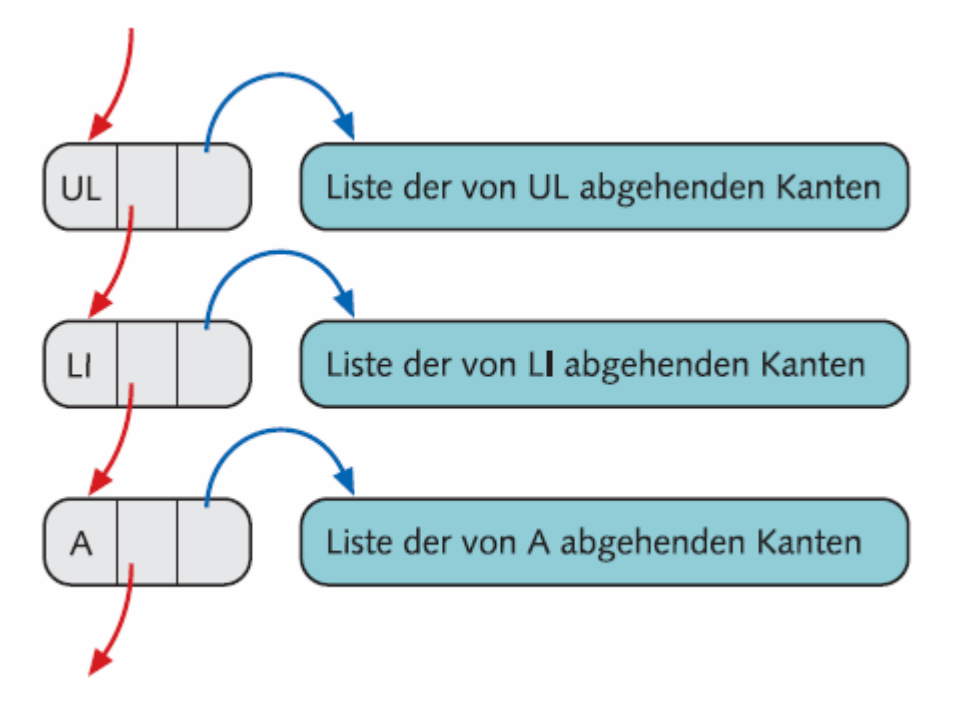
BlueJ - Implementierung mit Adjazenzmatrix




```
public class ARBEIT
{

    GRAPH_MATRIX g;

    public ARBEIT()
    {
    }

    public void AusfuehrenAutobahn()
    {
        // Erzeugen eines Graphenobjekts g für 14 Knoten
        GRAPH_MATRIX g = new GRAPH_MATRIX(14);

        // Anlegen der Knoten
        g.KnotenEinfuegen("A");
        g.KnotenEinfuegen("F");
        g.KnotenEinfuegen("FD");
```

```
        // Anlegen der Kanten
        g.KanteEinfuegen("KA","F",127);
        g.KanteEinfuegen("F","WÜ",131);
        KanteEinfuegen("WÜ","N",104);
```

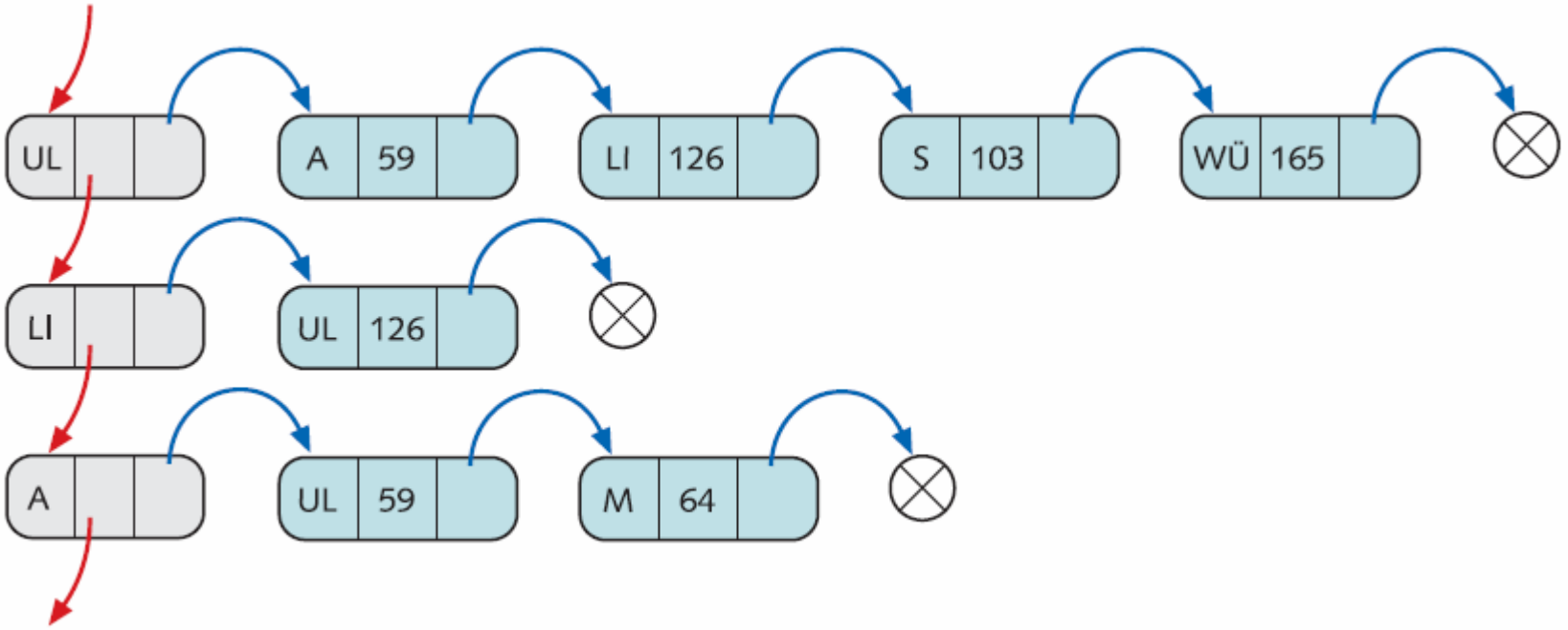
Darstellung von Graphen



Adjazenlisten

Darstellungsform
(Repräsentation) für den Graphen

Darstellung von Graphen



Adjazenlisten

Darstellungsform
(Repräsentation) für den Graphen

Graphendurchlauf

Tiefensuche

Der Hersteller eines Navigationssystems will den Graphen der Straßenverbindungen, der seiner Routenbestimmung zugrunde liegt, überprüfen lassen. Ein Tester soll die Strecken so abfahren, dass er

- jeden Ort mindestens einmal besucht und
- jede Kante mindestens einmal abfährt.

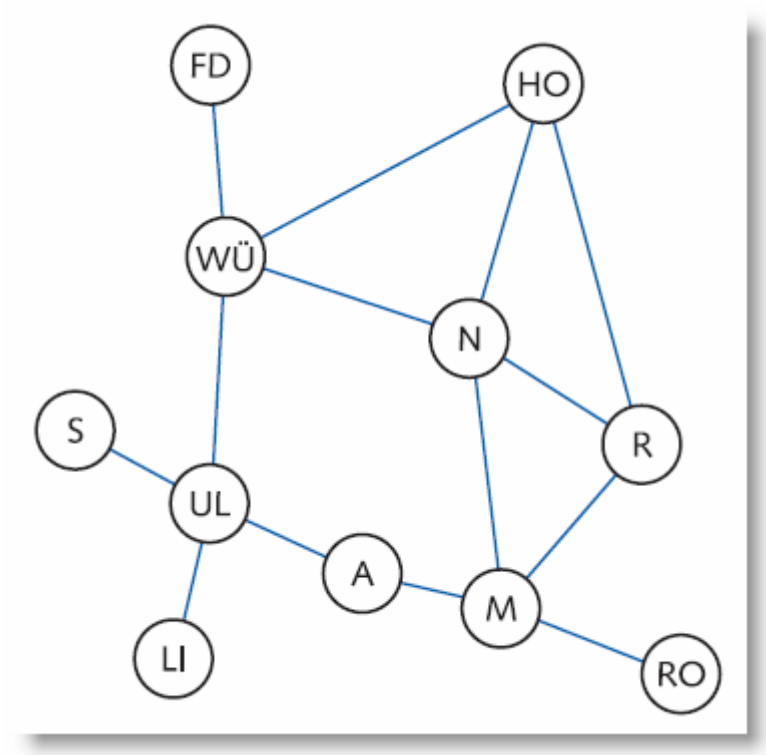
Hierzu ein effizientes Verfahren zu entwickeln, ist nicht einfach, denn schnell ist der Überprüfer im Kreis gefahren oder er muss eine lange Strecke zurückfahren, um einen übergangenen Knoten zu besuchen, oder er hat eine Kante ganz ausgelassen oder er fährt eine Kante unnötigerweise mehrmals ab ...

praxisorientierte Problemstellung

In einer Vorstudie soll ein Algorithmus entwickelt werden, der alle Knoten des Graphen systematisch besucht, dabei aber nicht alle Kanten befährt. Auch auf den kürzesten Weg für die Kontrollfahrt wird nicht geachtet.

Graphendurchlauf

Tiefensuche



erste Strategie

Start M

Wenn mehrere Wege möglich sind, dann werden die Orte immer dem Alphabet nach besucht.

M -> A

Jeder besuchte Ort wird markiert und Wege zu bereits markierten Orten werden bei der Auswahl der Verbindungen nicht berücksichtigt.

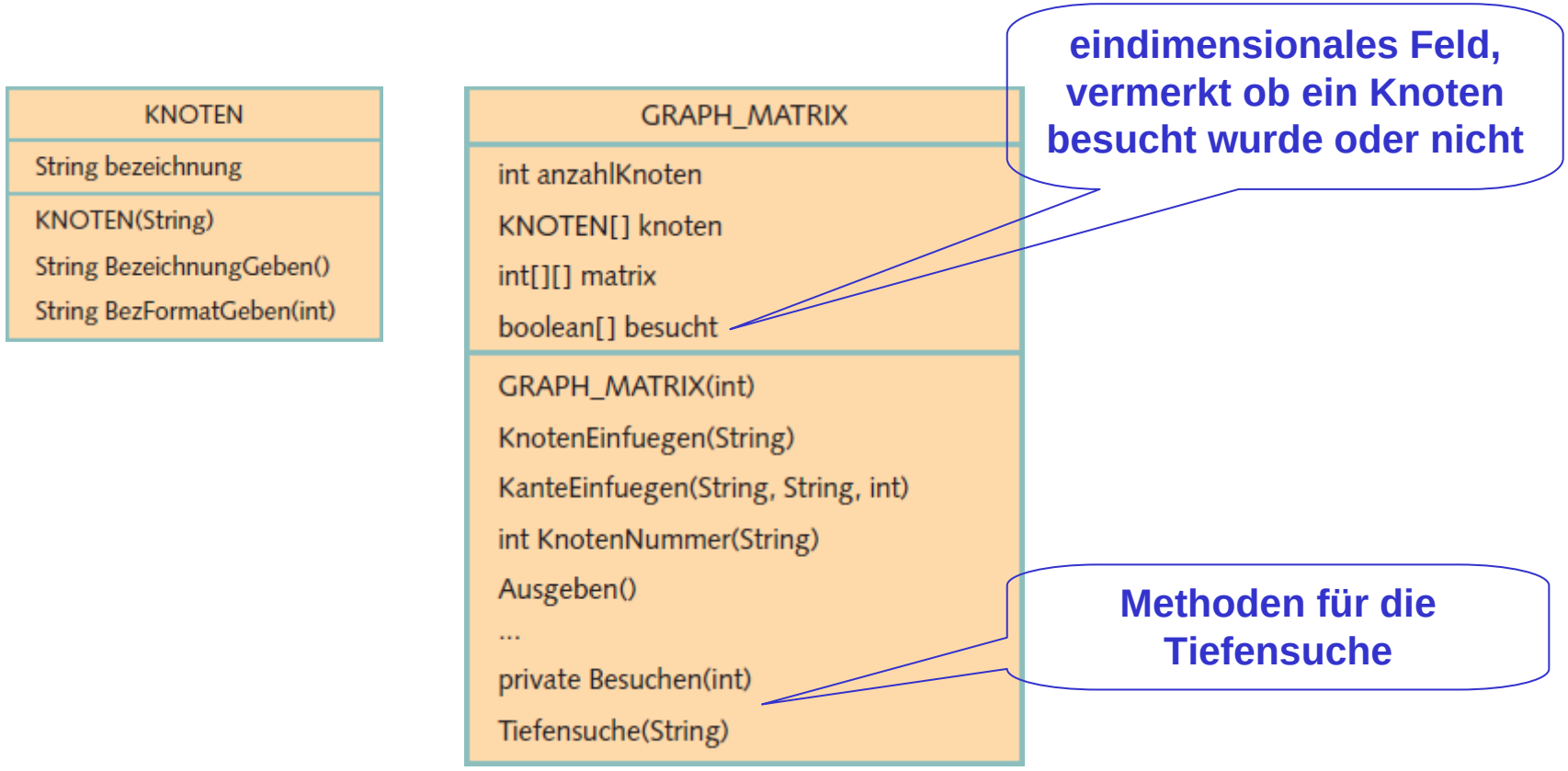
M -> A -> Ul -> Li

Wenn es an einem Ort nicht weitergeht, dann kehre zu dem Nachbarort zurück, von dem aus dieser Ort besucht wurde.

M -> A -> Ul -> Li -> Ul -> S -> Ul -> WÜ -> ...

Graphendurchlauf

Ergänzung der Klasse GRAPH_MATRIX für die Tiefensuche



Graphendurchlauf

Klasse GRAPH_MATRIX
Methode Besuchen(int knotenNummer)

```
// aktiven Knoten auf besucht setzen und in der Konsole ausgeben
besucht[knotenNummer] = true
In Konsole ausgeben: knoten[knotenNummer].BezeichnungGeben() + "; "

// in der Matrix die Zeile des aktiven Knotens nach Kanten durchforsten
zähle abzweigNummer von 0 bis anzahlKnoten-1

   // es gibt eine Kante und deren Zielknoten ist noch nicht besucht
   wenn (matrix[knotenNummer][abzweigNummer] > 0) und
                (nicht besucht[abzweigNummer])
   dann
      // in die Tiefe gehen und den Abzweigknoten besuchen
      Besuchen(abzweigNummer)
   endewenn

endezähle

// Der aktive Knoten mit der knotenNummer ist fertig bearbeitet
In Konsole ausgeben mit Zeilenvorschub:
         knoten[knotenNummer].BezeichnungGeben() + "(fertig); "
```

Graphendurchlauf

Klasse GRAPH_MATRIX
Methode Tiefensuche(String startKnoten)

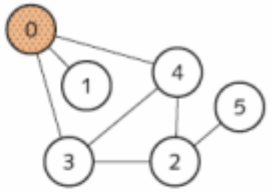
```
int startNummer

startNummer = KnotenNummer(startKnoten)
wenn (startNummer != -1)
dann
    zähle i von 0 bis anzahlKnoten-1
       besucht[i] = false
    endezähle

    Besuchen(startNummer)
endewenn
```

Graphendurchlauf

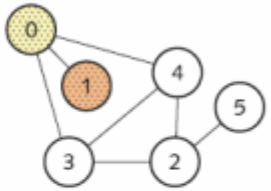
Tiefensuche

```
Besuchen(0)
```

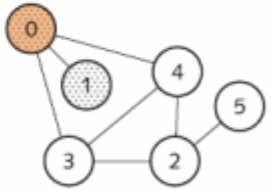
```
knoten[0] besucht setzen
// zähle abzweigNummer von 0 bis 5
...
// für abzweigNummer == 1
(matrix[0][1] > 0) ist wahr
(knoten[1] unbesucht) ist wahr
besuchen(1)
```




```
   knoten[1] besucht setzen
   // zähle abzweigNummer von 0 bis 5
   (matrix[1][abzweigNummer] > 0 und
     knoten[abzweigNummer] unbesucht) immer falsch
   // fertig 1
```




```
// für abzweigNummer == 3
(matrix[0][3] > 0) ist wahr
(knoten[3] unbesucht) ist wahr
besuchen(3)
```




```
   knoten[3] besucht setzen
   // zähle abzweigNummer von 0 bis 5
   ...
   (matrix[3][2] > 0) ist wahr
   (knoten[2] unbesucht) ist wahr
   besuchen(2)
```

Veranschaulichung des Algorithmus durch Bildsequenzen

Graphendurchlauf - Vertiefung

Zu den wesentlichen Aufgaben eines Navigationssystems gehört es, zu einem gegebenen Paar aus Start- und Zielort den kürzesten Verbindungsweg und seinen Verlauf zu bestimmen. Bei den Lösungsansätzen darf man sich nicht von einer landkartenartigen Darstellung des Graphen irritieren lassen. Sie nimmt schon Einiges an Information über die kürzesten Wege vorweg, das Navigationssystem kann diese Darstellung aber nicht auswerten; sein Programm sieht immer nur von einem zum nächsten Knoten und es weiß nicht schon vorab, welche der Abzweigungen zum Ziel führen.

praxisorientierte Problemstellung

Schritt 1:

Alle möglichen Wege zwischen zwei gegebenen Knoten suchen => WegeSuchen()

Schritt 2:

Den kürzesten Weg zwischen zwei gegebenen Knoten suchen => KuerzesterWeg()

(Algorithmus von Dijkstra)

Graphendurchlauf - Vertiefung

Alle Wege suchen zwischen zwei Knoten

Klasse GRAPH_MATRIX
Methode WegeSuchen(String startKnoten, String zielKnoten)

```
int startNummer = KnotenNummer(startKnoten)
int zielNummer = KnotenNummer(zielKnoten)

wenn (startNummer != -1) und (zielNummer != -1) und (startNummer != zielNummer)
dann
   zähle i von 0 bis anzahlKnoten-1
      besucht[i] = false
   endezähle

   Ablaufen(startNummer, zielNummer, startKnoten, 0)
endewenn
```

Graphendurchlauf - Vertiefung

Klasse GRAPH_MATRIX
Methode Ablaufen(int knotenNummer, int zielKnotenNummer, String pfad, int laenge)

```
int neueLaenge
String neuerPfad

besucht[knotenNummer] = true

// Wenn der Zielknoten erreicht wird, dann abbrechen und Pfad ausgeben
wenn (knotenNummer == zielKnotenNummer)
dann
   Konsole ausgeben mit Zeilenvorschub: pfad + "; Länge " + laenge
sonst
   zähle abzweigNummer von 0 bis anzahlKnoten-1

      wenn (matrix[knotenNummer][abzweigNummer] > 0) und
                (nicht besucht[abzweigNummer])
      dann
         // der Weg wird in Richtung der Abzweigung verlängert
         neuerPfad  = pfad + "/" + knoten[abzweigNummer].BezeichnungGeben()
         neueLaenge = laenge + matrix[knotenNummer][abzweigNummer]

         Ablaufen(abzweigNummer, zielKnotenNummer, neuerPfad, neueLaenge)
      endewenn
   endezähle
endewenn
// Knoten im Gegensatz zur Tiefensuche wieder freigeben
besucht[knotenNummer] = false
```

Datenstruktur Graph

Aufgaben siehe Arbeitsblatt.

Viel Erfolg!