Lernmodul 7 Algorithmus von Dijkstra

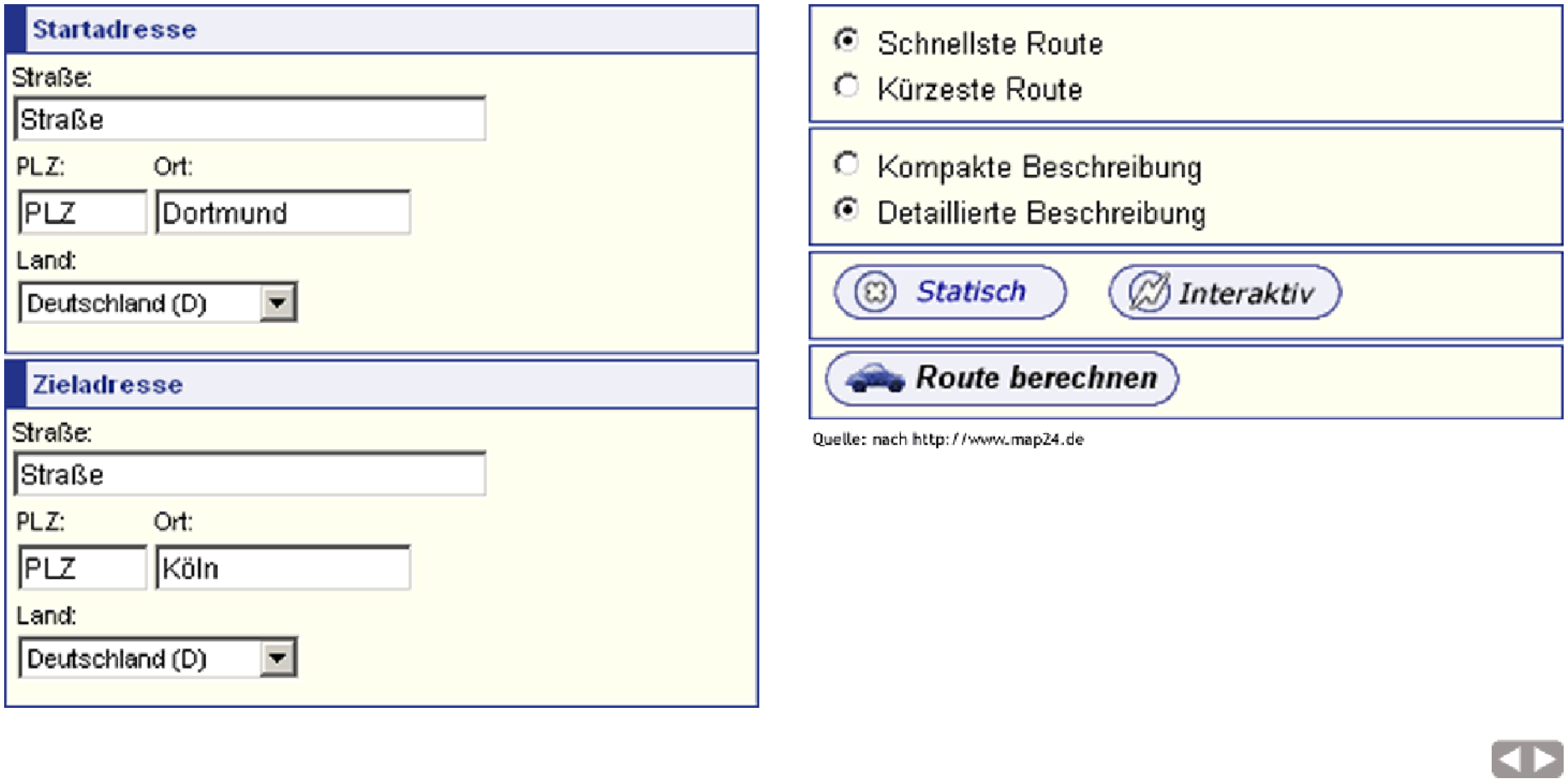
Quelle: http://www.map24.de

Algorithmus von Dijkstra Übersicht

Kürzester Weg von A nach B in einem Graphen

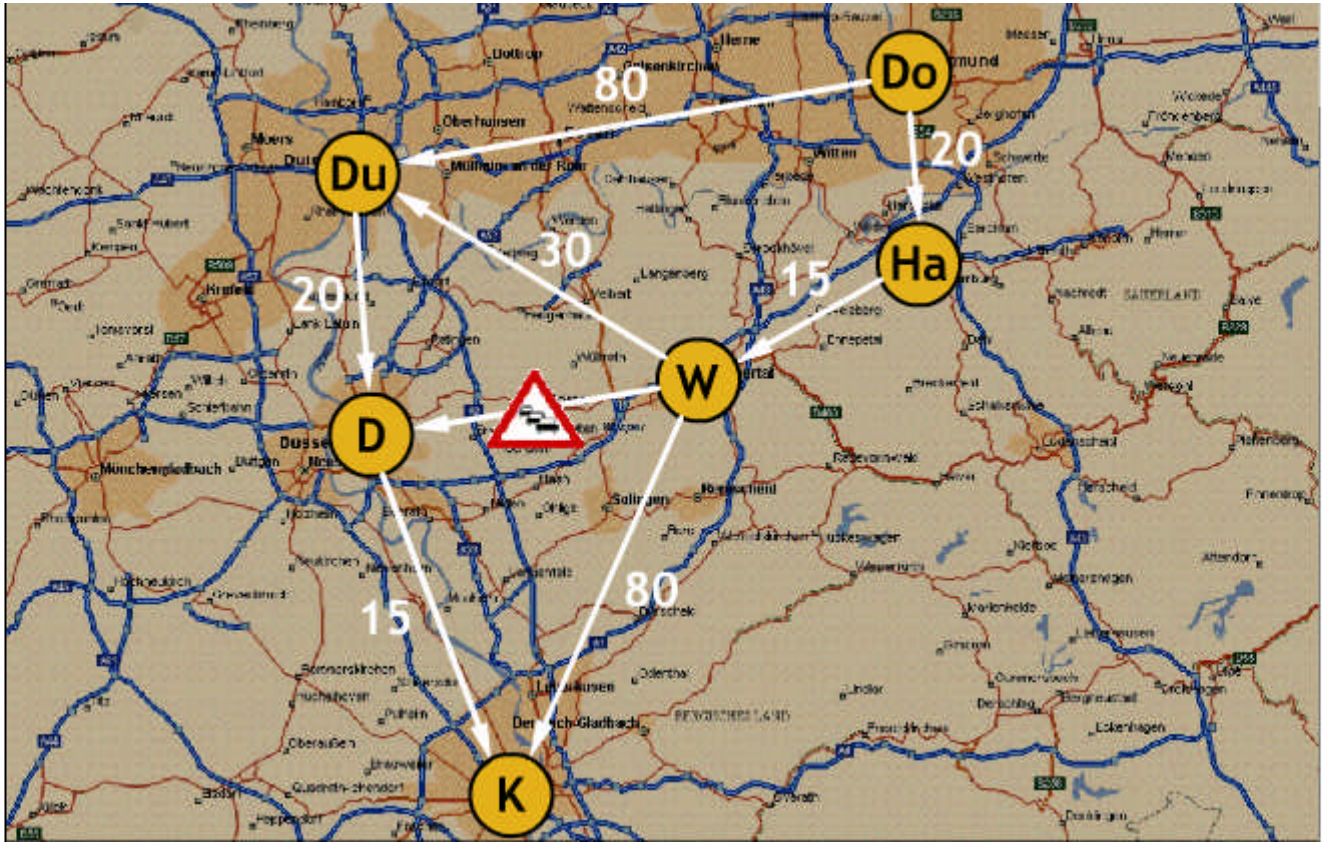
- Problemstellung: Suche einer Route von Ort A nach Ort B
- Formulierung des Problems / einer Lösung
- Animation eines Beispiels für den Algorithmus
- Formulierung des Algorithmus in Pseudocode
- Sätze zur Korrektheit des Algorithmus
- Erforderliche Datenstrukturen
  - Adjazenzmatrix
  - Adjazenzliste
  - Heap

Algorithmus von Dijkstra Problemstellung



Quelle: nach http://www.map24.de

Algorithmus von Dijkstra Beispielgraph

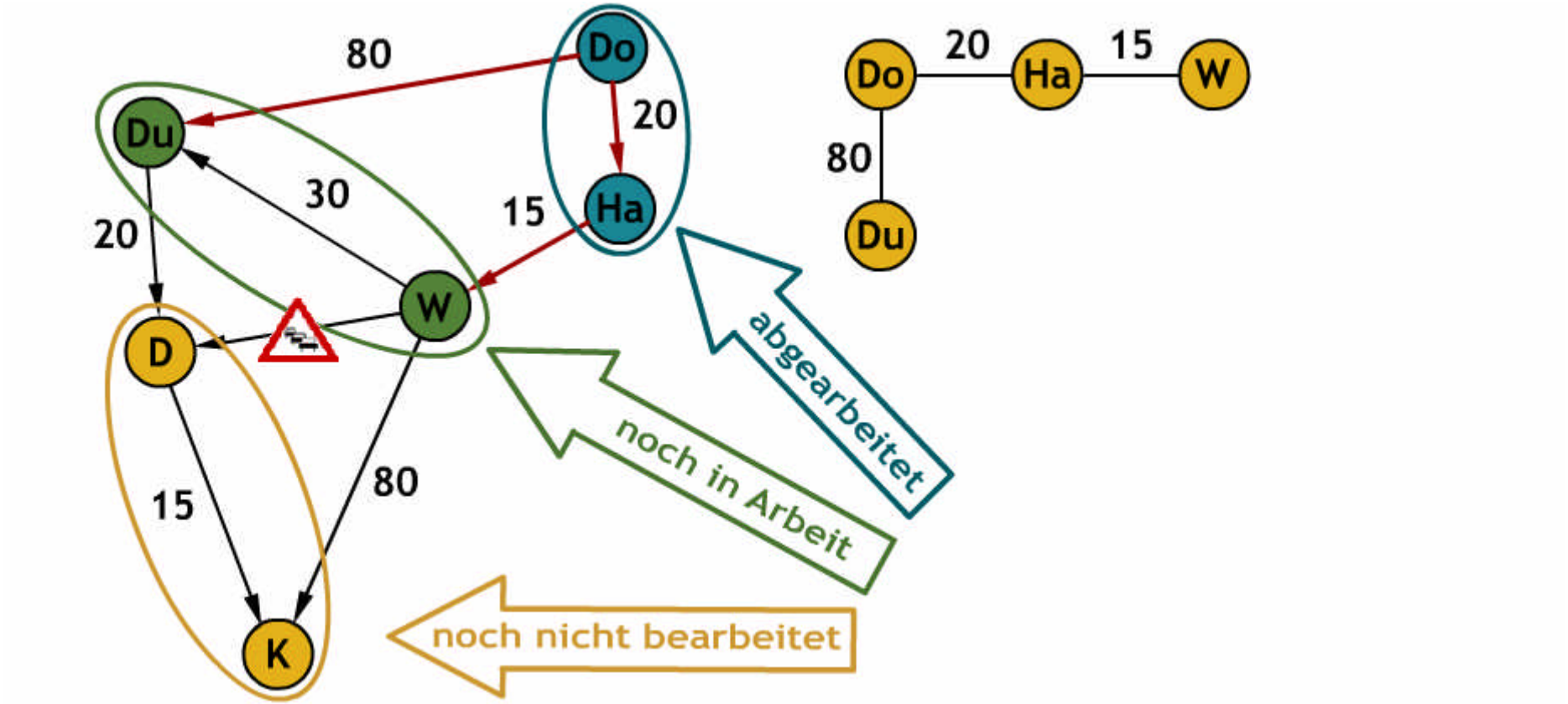


A 5x

Dijkstra Formulierung / Lösung des Problems

- **Gegeben:** Gerichteter Graph G, dessen Kanten mit Zahlen (Kosten, z.B. km oder min.) beschriftet sind
- **Aufgabe:** Berechnung des kürzesten Weges **s** ⟶ **z** von einem Startknoten **s** zu einem Zielknoten **z**
- **Erste Idee:** Berechne alle Wege und wähle den kürzesten
- **Beobachtung:** Wenn der kürzeste Weg von **s** nach **z** über **y** führt, sind die Teilwege **s** ⟶ **y** und **y** ⟶ **z** ebenfalls kürzeste Wege
- **Effiziente Lösung:** Berechne **alle kürzesten** Wege und wähle den von **s** nach **z** aus
- **Algorithmus von Dijkstra:** jeder Schritt sitzt („Greedy"-Algorithmus)
  - Berechnung der kürzesten Wege von einem beliebigen Startknoten zu allen anderen Knoten des Graphen (1 : n)
  - **single source shortest path** - Problem

Dijkstra Beispiel zum Ablauf



32x

Dijkstra Bezeichnungen im Algorithmus

s	Startknoten
v	beliebiger Knoten im Graphen
dist (v)	Abstand des Knotens v vom Startknoten s
BLAU	Menge der abgearbeiteten Knoten
GRÜN	Menge der aktiven Knoten ("in Arbeit")
succ (v)	Menge der Nachfolger(-Nachbarn) von v
∀	für alle Elemente
dist (v,v‘)	Distanz (Zeit) der Kante (v,v‘)

Dijkstra Beschreibung des Algorithmus

Idee: Bildung eines Teilgraphen TG in G, ausgehend vom Startknoten s

- TG beschreibt den erkundeten Teil von G, bestehend aus
  - **BLAUEN** Knoten: Alle Nachfolger wurden betrachtet
  - **GRÜNEN** Knoten: Rand von TG, abgehende Kanten wurden nicht betrachtet
- Zwei Arten von Kanten innerhalb von TG:
  - **rote** Kanten: Baum der kürzesten Wege
  - **grüne** Kanten: "längere Wege"
- In jedem Knoten von TG wird der Abstand zu s verwaltet (dist(K))
- TG wächst, indem in jedem Schritt der Randknoten mit minimalem Abstand von s ins Innere von TG (**BLAU**) übernommen wird
- Nachfolgeknoten übernommener Knoten werden Randknoten (**GRÜN**)
- Kürzeste Pfade zu **GRÜNEN** Knoten, die erneut erreicht werden, sind ggf. zu korrigieren

Quelle: nach Güting/Dieker

Dijkstra Formulierung des Algorithmus

```
algorithm Dijkstra (s)     //berechne alle kürzesten Wege von s aus
BLAU = ∅; GRÜN = {s}; dist(s) = 0;
while( GRÜN != ∅ ) {
   wähle v ∈ GRÜN, so dass ∀ v' ∈ GRÜN: dist(v) <= dist(v');
   färbe v blau;
   for( v_i ∈ succ(v) ) {
      if (v_i ∉ (GRÜN ∪ BLAU) //noch nicht besuchter Knoten
         färbe die Kante (v,v_i) rot;
         färbe v_i grün;
         dist(v_i) = dist(v) + dist(v,v_i); }
      else if (v_i ∈ GRÜN) {      // v_i erneut erreicht
         if(dist(v_i) > dist(v) + dist(v,v_i)) {
            färbe die Kante (v,v_i) rot;
            färbe die bisher rote Kante zu v_i grün;
            dist(v_i) = dist(v) + dist(v,v_i); }
         else {
            färbe (v,v_i) grün; }}
      else { färbe (v,v_i) grün; }}}      // v_i ∈ BLAU
```

Quelle: nach Güting/Dieker

Dijkstra Offene Fragen

- Wie finden wir schnell
  - alle Nachfolger eines Knoten: **for ($v_i \in$ succ(v)) ...**?
  - aktive Knoten: **v ∈ GRÜN, so dass ∀ v‘ ∈ GRÜN: dist(v) <= dist (v‘);**?
- Wir benötigen:
  - Datenstruktur für den Graphen
  - Datenstruktur für die grünen (aktiven) Knoten
- Der schrittweise Entwurf des Algorithmus lässt diese Fragen **zunächst absichtlich offen**...
- ...zugunsten der **Konzentration auf die wesentliche Idee:** Wir wollen zunächst die **Korrektheit des Algorithmus** beweisen.

Dijkstra Korrektheit des Algorithmus I

Satz 1: Für jeden **GRÜNEN** Knoten v gilt: Unter den rotgrünen Wegen zu v ist der rote der kürzeste.

>>Beweis: Induktion über die Folge der **BLAU** gefärbten Knoten

Quelle: nach Güting/Dieker

Dijkstra Beweis Satz 1 - Teil 1

Induktionsanfang:

Die Behauptung gilt für s, da noch keine Kante gefärbt ist.

Induktionsschluß:

Annahme: Die Aussage gilt für **BLAU** und **GRÜN**.

$v \in$ **GRÜN** wird blau gefärbt. Seien $v_1, \ldots, v_m$ die Nachfolger von v. Für die v_i sind folgende Fälle zu unterscheiden:

>>Fortsetzung

Quelle: Güting/Dieker

Dijkstra Beweis Satz 1 - Teil 2

1. v_i wurde zum erstenmal erreicht, war also bisher ungefärbt und wird jetzt grün. Der einzige grünrote Weg zu v_i hat die Form $s \Rightarrow v \rightarrow v_i$ ("$\Rightarrow$" bezeichne einen Weg, "$\rightarrow$" eine einzelne Kante). $s \Rightarrow v$ ist nach Induktionsannahme minimal. Also ist $s \Rightarrow v_i$ minimal.
2. v_i ist grün und wurde erneut erreicht.
   Der bislang rote Weg zu v_i war $s \Rightarrow x \rightarrow v_i$, der neue rote Weg zu v_i ist der kürzeste Weg unter den Wegen
   a. $s \Rightarrow x \rightarrow v_i$ und
   b. $s \Rightarrow v \rightarrow v_i$

   Der Weg (1) ist minimal unter allen grünroten Wegen, die nicht über v führen, (2) ist minimal unter allen, die über v führen (wie in (a)), also ist der neue Weg minimal unter allen grünroten Wegen.

Quelle: Güting/Dieker

Dijkstra Korrektheit des Algorithmus II

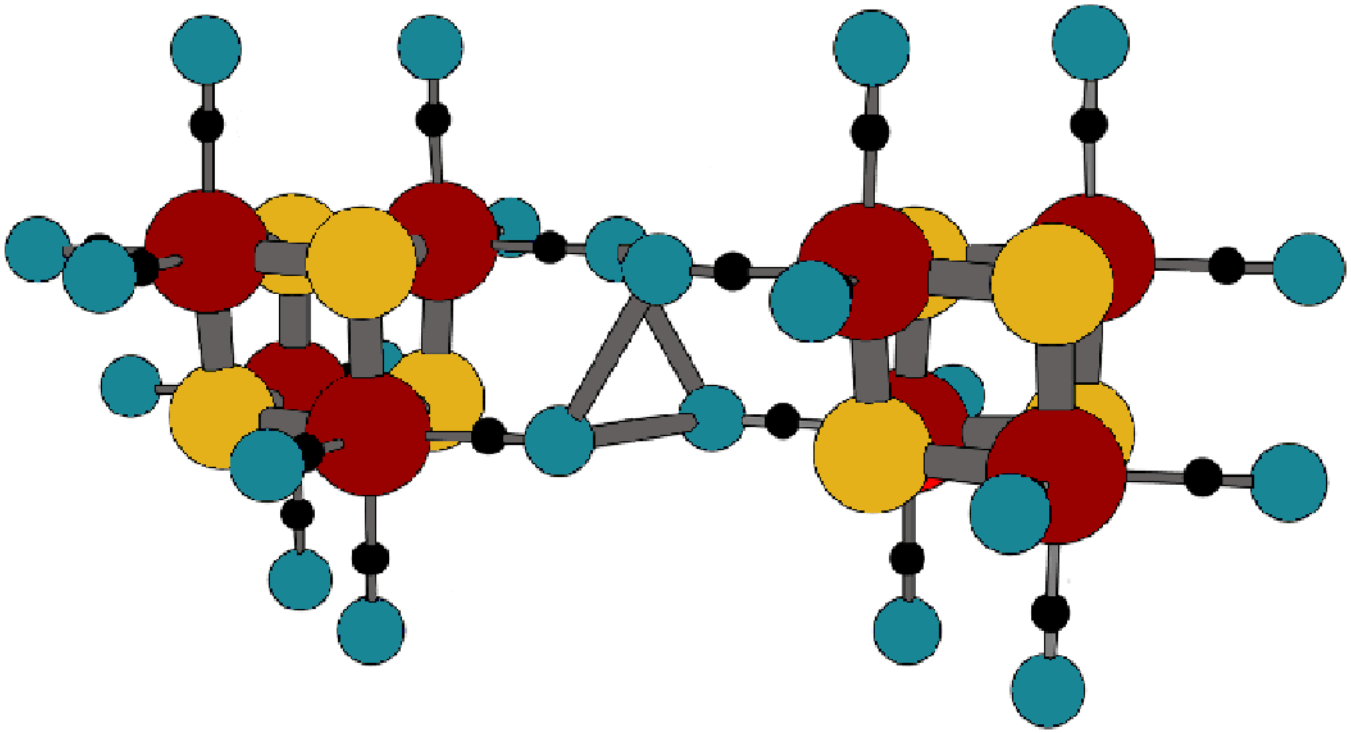
Satz 2: Für jeden **BLAUEN** Knoten v gilt: Unter allen Wegen zu v ist der **rote** der kürzeste.
Beweis: Induktion über die Folge der **BLAU** gefärbten Knoten

Satz 3 (Hauptsatz): Der Algorithmus von Dijkstra berechnet die kürzesten Wege zu allen von s erreichbaren Knoten.
Beweis: Nach Ablauf des Algorithmus sind alle erreichbaren Knoten **BLAU** gefärbt. Die Behauptung folgt dann aus Satz 2.

- Übung:
  - Vollziehen Sie den Gedankengang von Beweis 1 am Beispiel nach.
  - Versuchen Sie einen Beweis für Satz 2 (Hinweis: Finden Sie einen Widerspruch zur Induktionsannahme)

Quelle: Güting/Dieker

Datenstrukturen Organisation der Knoten



Quelle: nach http://www.hmi.de

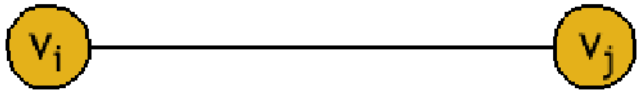
Datenstrukturen Übersicht

- Datenstruktur für den **Graphen** (mit Kosten)
  - Variante 1: **Adjazenzmatrix**
  - Variante 2: **Adjazenzliste**

- Datenstruktur für die **Menge der aktiven** (**GRÜNEN**) **Knoten**: **Heap**

Datenstrukturen Adjazenzmatrix

Definition: Knoten, die durch eine Kante verbunden sind, heißen benachbart oder **adjazent**.



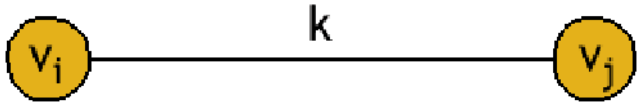
Definition: Die n × n Matrix $A = (a_{ij})$ mit

$$a_{ij} = \begin{cases} true & \text{falls } v_i \text{ und } v_j \text{ adjazent} \\ false & \text{sonst} \end{cases}$$

heißt **Adjazenzmatrix** des Graphen.

Datenstrukturen Adjazenzmatrix mit Kosten

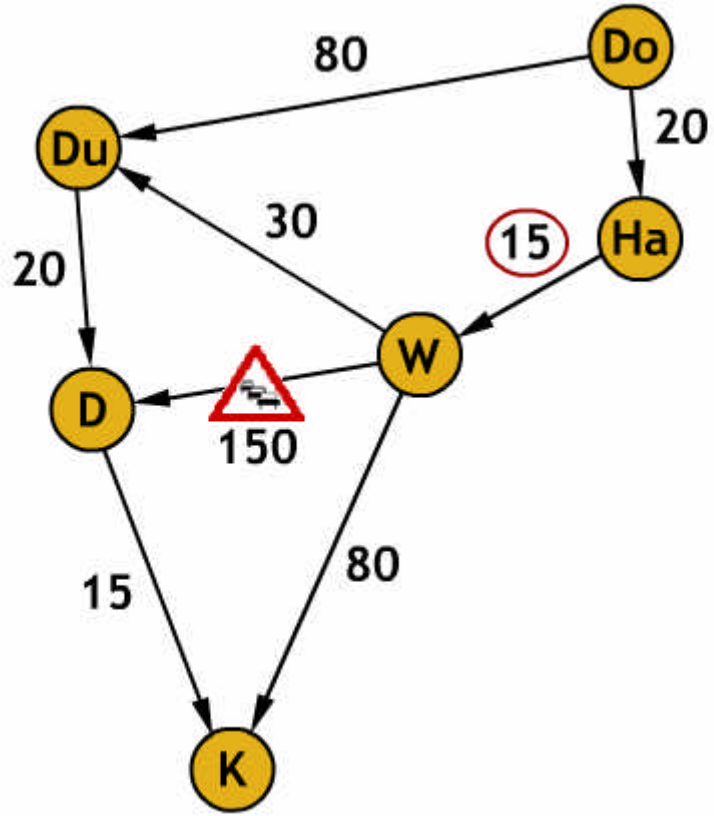
Bei bewerteten Graphen lassen sich direkt die Kosten in die Adjazenzmatrix eintragen:



$$a_{ij} = \begin{cases} k & \text{falls k die Kosten der Kante von } v_i \text{ nach } v_j \\ \infty & \text{sonst} \end{cases}$$

Beachte: alle Diagonalelemente sind 0

Datenstrukturen Beispiel zur Adjazenzmatrix



	Do	Du	Ha	W	D	K
Do	0	80	20	∞	∞	∞
Du	∞	0	∞	∞	20	∞
Ha	∞	∞	0	15	∞	∞
W	∞	30	∞	0	150	80
D	∞	∞	∞	∞	0	15
K	∞	∞	∞	∞	∞	0

A 6x

Datenstrukturen Adjazenzmatrix ja/nein?

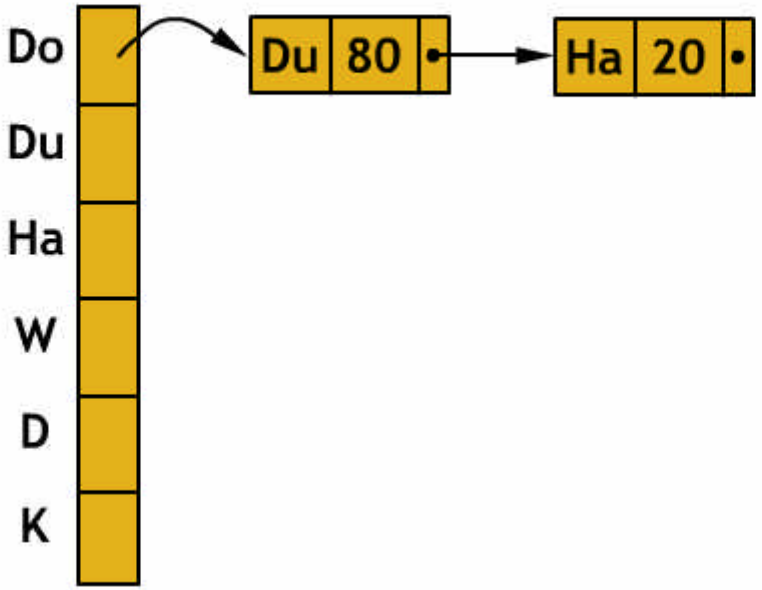
- **Vorteil:** Möglichkeit, in einer Laufzeit von O(1) festzustellen, ob eine Kante von v_i nach v_j existiert.
- **Nachteil:** hoher Platzbedarf: $O(n^2)$

Datenstrukturen Adjazenzliste

- Für jeden Knoten wird eine **Liste** seiner (Nachfolger-) Nachbarknoten abgespeichert.
- Der Zugriff auf die Listen erfolgt über ein **Array** der Länge n (n = Anzahl der Knoten).

Datenstrukturen Beispiel zur Adjazenzliste

	Do	Du	Ha	W	D	K
Do	0	80	20	∞	∞	∞
Du	∞	0	∞	∞	20	∞
Ha	∞	∞	0	15	∞	∞
W	∞	30	∞	0	150	80
D	∞	∞	∞	∞	0	15
K	∞	∞	∞	∞	∞	0



3x

Datenstrukturen Adjazenzliste ja/nein?

- **Vorteil:**
  - geringer Platzbedarf: O(n+e) (e = Anzahl der Kanten)
  - alle m Nachfolger eines Knotens sind in der Zeit O(m) erreichbar
- **Nachteil:**
  - Um zu prüfen, ob v_i und v_j benachbart sind, muß die Adjazenzliste von v_i durchlaufen und nach v_j durchsucht werden.
  - **Übung:** Wie hoch ist der Aufwand zum Durchlaufen der Adjazenzliste von v_i?
- **aber:** für Dijkstra ist eine Adjazenzliste **ideal**

Datenstr. Repräsentation der GRÜNEN Knoten

- Der Algorithmus benötigt folgende **Operationen**:
  - Einfügen der Nachfolger des betrachteten Knoten
  - Selektion und Entfernen des kleinsten Elements
- **Ziel:**
  - Einfügen eines Knotens in O(log n)
  - Finden und Entfernen des kleinsten Knotens in O(log n)
- **Variante A:** AVL-Baum
- **Variante B** (spezialisierter auf diese Anwendung): **Heap**

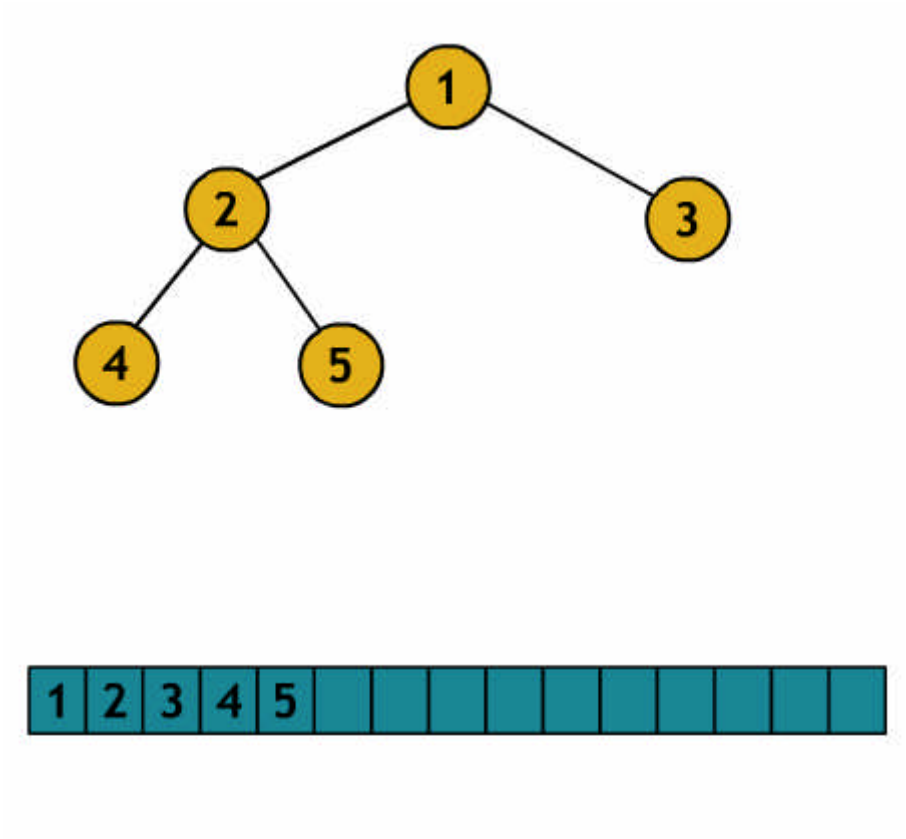
Datenstrukturen Definition des Heaps

- Ein zu jeder Zeit möglichst **vollständiger binärer Baum T**: Alle Ebenen bis auf die letzte sind voll besetzt.
- Der Baum besitzt eine **partielle Ordnung**: Für jeden Teilbaum T‘ mit Wurzel w gilt

  info(v) >= info(w)

  für alle Knoten v von T‘. Dabei bezeichnet "info" den Eintrag des Knotens. In der Wurzel steht also das Minimum des Teilbaums.
- **Darstellung** (Einbettung) eines vollständigen Baums erfolgt in einem Array
- **>>Beispiel:** Eingabe der sortierten Folge von Zahlen {1 .. 15}

Datenst. Einbettung des Heaps in ein Array



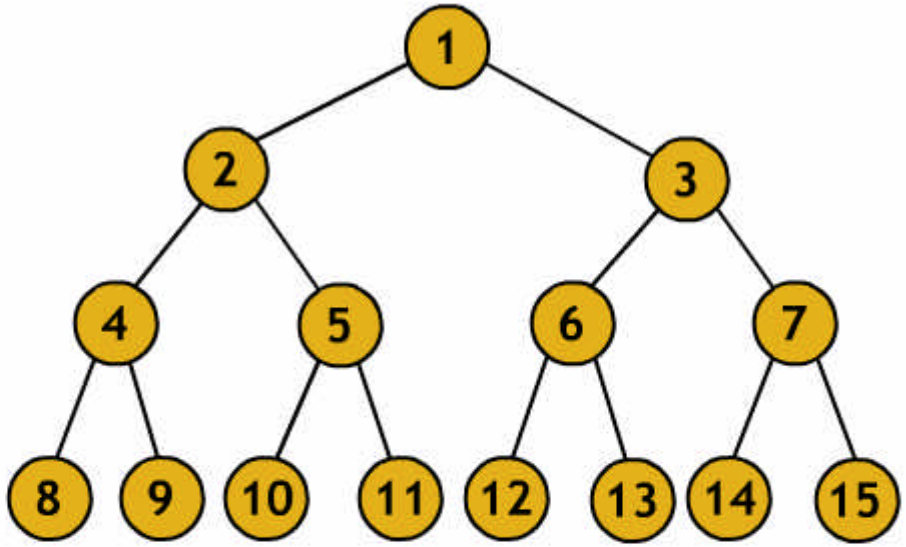
- **Problem:** Abbildung der Kanten auf Indizes
  - Index des Vaters?
  - Indizes der beiden Söhne?
- **Vorgehen** (i = Nummer des Knotens in der Abbildung):
  Jeder Knoten v mit Nummer i wird im Feld i des Arrays dargestellt.
  - index(v) = i
  - index (v.linkerSohn) = 2 × i
  - index (v.rechterSohn) = 2 × i + 1
  - index (v.vater) = [i/2]

A 10x

Datenstr. Entfernen des kleinsten Heap-Elements

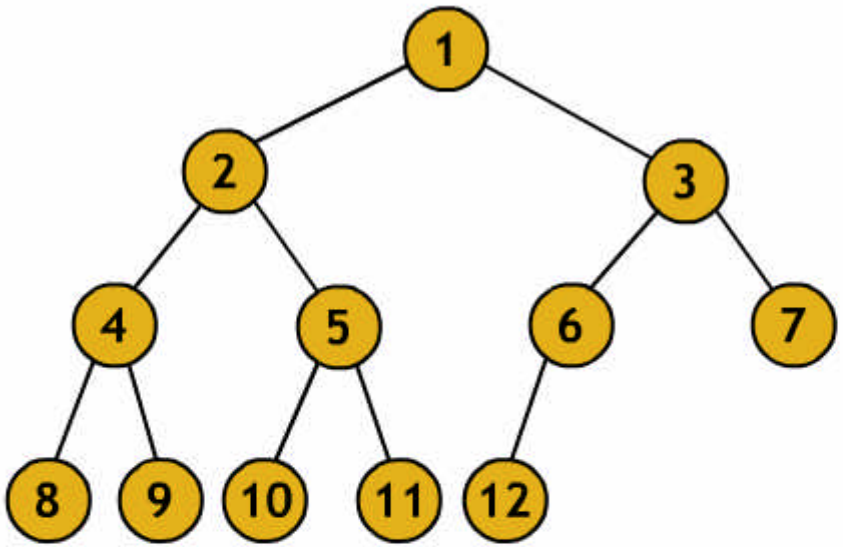
```
algorithm deletemin (H)

/*lösche das minimale Element des Heaps H
  und gib es aus */

  gib den Eintrag der Wurzel aus;
  lösche die Wurzel und
  ersetze sie durch die letzte
  Position im Baum (lösche diesen Knoten)
  sei w die Wurzel mit den Söhnen p und q;

while p oder q existieren und
 ((info (w) > info (p)) oder
  (info (w) > info (q))) do
  vertausche p mit dem kleineren der
  beiden Söhne
  benenne w, p, q um
```

Quelle Pseudocode: Güting/Dieker



22x

Datenstr. Einfügen eines neuen Heap-Elements




```
algorithm insert (H,e)
/* füge Element e in H ein */
  füge einen neuen Knoten q mit
  Eintrag e an der ersten freien
  Position der untersten Ebene
  des Baumes ein, eröffne ggf.
  neue Ebene;
  p sei der Vater von q;
while p existiert und
  (info(q) < info (p)) do
   vertausche p und q
   benenne p und q um
```

Quelle Pseudocode: Güting/Dieker

23x

Dijkstra Zwei Erweiterungen im Überblick

- **Dijkstra:** Finden kürzester Wege in Graphen
- **Reale Netze** stellen besondere Anforderungen
  - **Größe** des Netzwerkes (Effizienz)
    - Dijkstra-Erweiterung "Dijkstra mit Geometrie"
  - Straßenverkehrsordnung (**Abbiege- und Wendeverbote**)
    - Abbildung realer Straßennetze auf Graphen
    - Ansätze:
      - Modifikation des Graphen
      - Modifikation von Dijkstra

Dijkstra Literatur

Güting, Ralf, Stefan Dieker: Datenstrukturen und Algorithmen. 2. Auflage - B.G. Teubner, Stuttgart, Leipzig, Wiesbaden, 2003