DHBW Lörrach
Duale Hochschule Baden-Württemberg

Algorithmen und Komplexität
TIF 21A/B
Dr. Bruno Becker

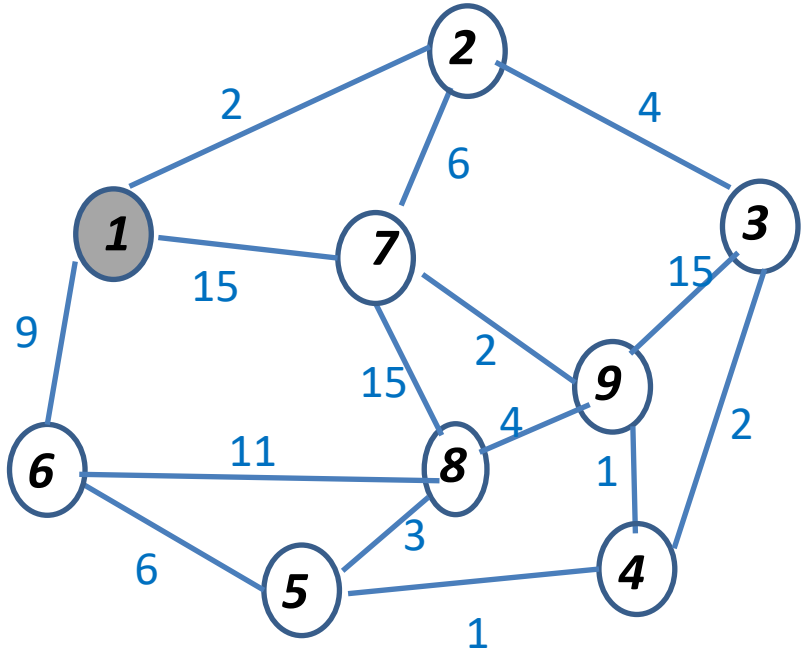
9. Optimierungsprobleme für Graphen
9.2. Kürzeste Wege

Kürzeste Wege

- **Problemstellung**
- Algorithmus von Dijkstra
- Gerichtete Graphen
- Algorithmen in gerichteten Graphen

Problemstellung – kürzeste Wege

- Beispiel: Kürzester Weg von Startknoten (1) zu einem, mehreren oder allen Zielnoten



Definition kürzeste Pfade

- Kantengewichtete Graphen:
  - **Gewicht (Länge) eines Pfades = Summe seiner Kantengewichte**
- **Kürzester Pfad:**
  - Gegeben Knoten *s* und *t* in einem kantengewichteten Graphen
  - Ein ***kürzester Pfad*** von *s* nach *t* ist ein Pfad von *s* nach *t,* sodass kein anderer Pfad von *s* nach *t* ein niedrigeres Gewicht hat
  - Notation $\delta(s,t)$ = Gewicht eines kürzesten Pfades von *s* nach t = *Entfernung* von *s* nach *t*
- **Ein kürzester Pfad von *s* nach *t* existiert genau dann, wenn**
  - Es einen Pfad von *s* nach *t* gibt
  - Kein Pfad zwischen *s* und *t hat* einen Zyklus mit negativen Kantengewichten

Eigenschaften kürzester Pfade

- Nicht immer eindeutig
- Zyklenfrei (unter der Annahme, dass Kantengewichte positiv)
- **Jeder Teilpfad eines kürzesten Pfades ist ein kürzester Pfad**
  - Wieso? – Beweis durch Widerspruch
  - Angenommen
    1. *s,… p_i,,…v* Kürzester Weg von s nach v und
    2. *s,… p_i,* **Kein** Kürzester Weg von s nach p_i
  - Dann könnte man den kürzesten Pfad von s nach p_i, in den kürzesten Pfad von s nach v einbauen und würde diesen Pfad verkürzen
  - **Widerspuch zur Annahme 1 → Annahme 2 stimmt nicht**
- Es gibt Baum von s zu allen erreichbaren Knoten mit kürzesten Pfaden

Datenstruktur für kürzeste Pfade

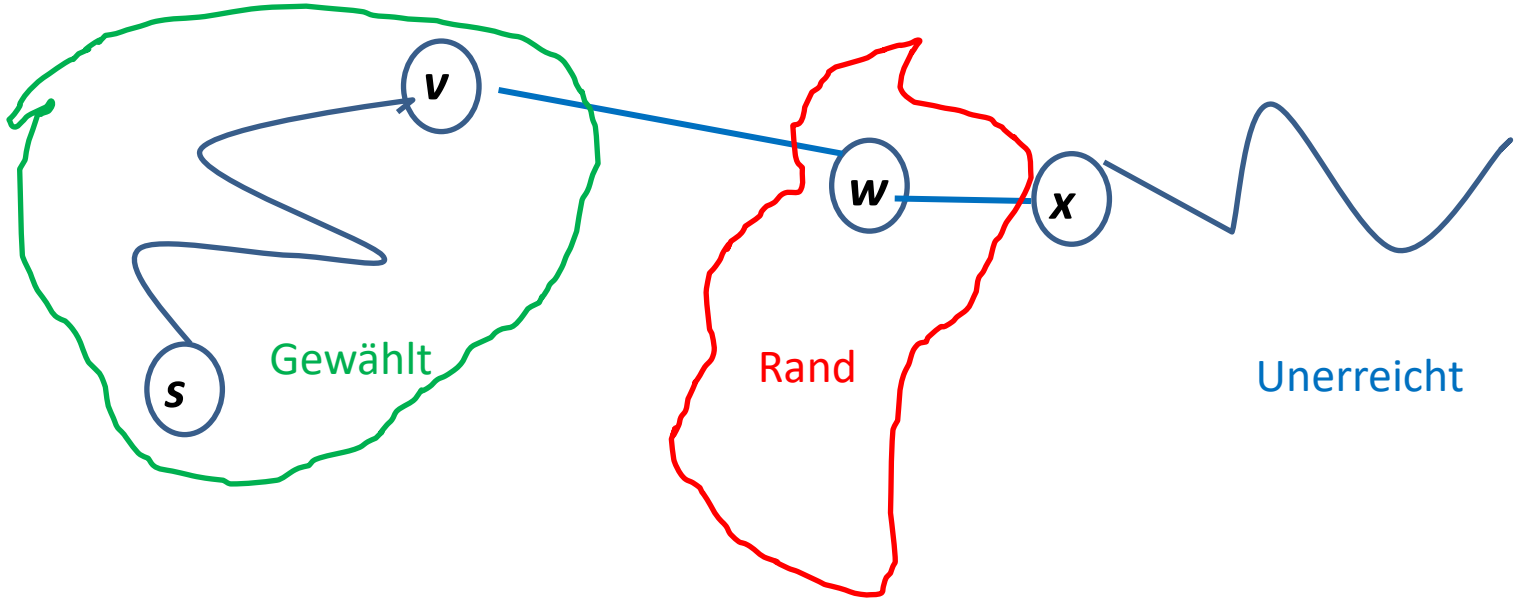
- Speichern für jeden Knoten v:
  - *v.dist* = Gewicht des kürzesten ***bisher gefundenen*** Pfades von *s* nach *v*
  - *v.vorg* = Direkter Vorgänger von v im Pfad
- Ergibt Suchbaum von *s* zu allen erreichbaren Knoten

Kürzeste Wege

- Problemstellung
- **Algorithmus von Dijkstra**
- Gerichtete Graphen
- Algorithmen in gerichteten Graphen

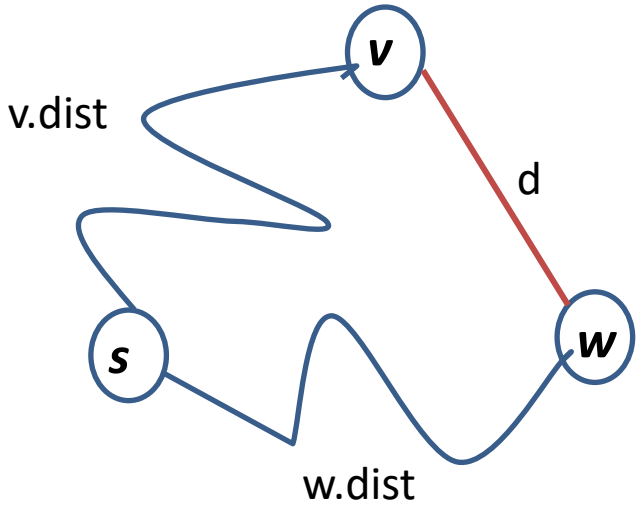
Algorithmus von Dijkstra: Grundidee

- Jeder Knoten ist entweder
  - gewählter Knoten: Dann ist kürzester Weg von *s* zu dem Knoten bekannt
  - Randknoten: Es gibt einen vorläufig kürzesten Weg von *s* zum Knoten
  - Unerreichter Knoten: Es gibt noch keinen Weg von von *s* zum Knoten



Basisoperation: Relaxieren einer Kante

- Verbessere Wege zu Knoten *w* im Rand durch Relaxieren:
  - Betrachte Kante *(v,w)*
  - Lässt sich der kürzeste bisher gefundene Pfad von s nach w über v abkürzen?; *d = c (<v,w>)*
  - *Wenn v.dist + d < w.dist ? Dann: w.dist = v.dist + d; w.vorg = v;*



Algorithmus von Dijkstra

Suche kürzeste Wege in *G=(V,E,c) mit c E→* R_0^+ von *s* zu allen anderen Knoten

1. Für alle *v* außer *s* // *Anfangs sind alle Knoten außer s unerreicht*
   - *v.dist = ∞;*
   - *v.vorg = nil;*
   - *v.gewaehlt = false;*
2. *s.vorg = s; s.dist = 0; s.gewaehlt = true; R= {} // Starte mit s*
3. *Ergänze R bei s; // Alle zu s benachbarten Knoten zum Rand R*
4. *Solange R <> {} tue // wähle nächstgelegenen Randknoten*
   - *Wähle v ε R mit v.dist minimal und entferne v aus R;*
   - *v.gewaehlt = true;*
   - *Ergänze R bei v; // Hinzunahme unerreichter Knoten zum Rand, Entfernungen anpassen*

Algorithmus von Dijkstra (2)

Ergänze Rand R bei v

1. *Für alle <v,w> aus E bezogen auf alle Nachbarn w von v*

   1.1 *Falls (w.gewaehlt) = false) und (v.dist + c(<v,w>) < w.dist) dann*
   // w ist kürzer über v erreichbar → Kante relaxieren

   - *w.vorg = v;*
   - *w.dist = v.dist + c(<v,w>);*
   - *Vermerke w in R;*

Algorithmus von Dijkstra: Aufwandsanalyse

- Initialisieren - Schritt 1-3: $O(|V|)$
- Schleife (Schritt 4): $O(|V|)$-mal wird Schleife durchlaufen
  - Operationen innerhalb der Schleife:
    - Einfügen, Minimum Entfernen, Kante relaxieren – Entfernung anpassen
  - Datenstruktur für PQ:
    1. Array: Einfügen, Kante relaxieren O(1) aber Minimum Entfernen $O(|V|)$
    2. Heap: In den Heap kommen $O(|E|)$ Kanten Alle Einzel-Operationen $O(\log |V|)$ d.h. für alle Kanten $O((|E| \log (|E|)) = O((|E| \log (|V|))$ Aufwand

➔ **Gesamtaufwand für *G(V,E)* mit Array $O(|V^2|)$**

➔ Gut für dichte Graphen (d.h. sehr viele Kanten)

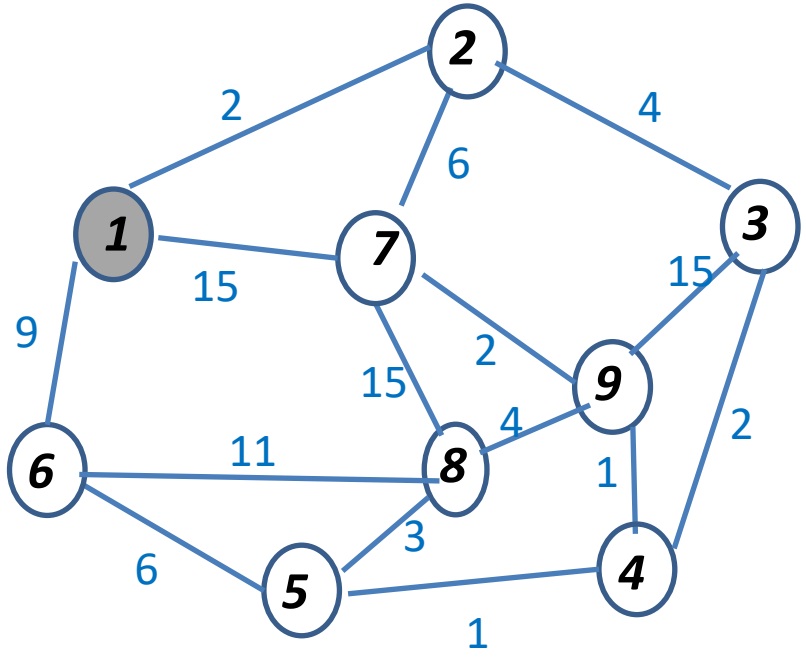
➔ **Gesamtaufwand für *G(V,E)* mit Heap $O((|E| \log (|V|))$**

➔ Gut für dünn besetzte Graphen (d.h. sehr wenige Kanten)

➔ Es geht noch besser mit *Fibonacci-Heaps:* **$O((|E| + |V| \log (|V|))$**

Algorithmus von Dijkstra – Beispiel

- Beispiel: Kürzester Weg von Startknoten (1) zu allen Zielnoten



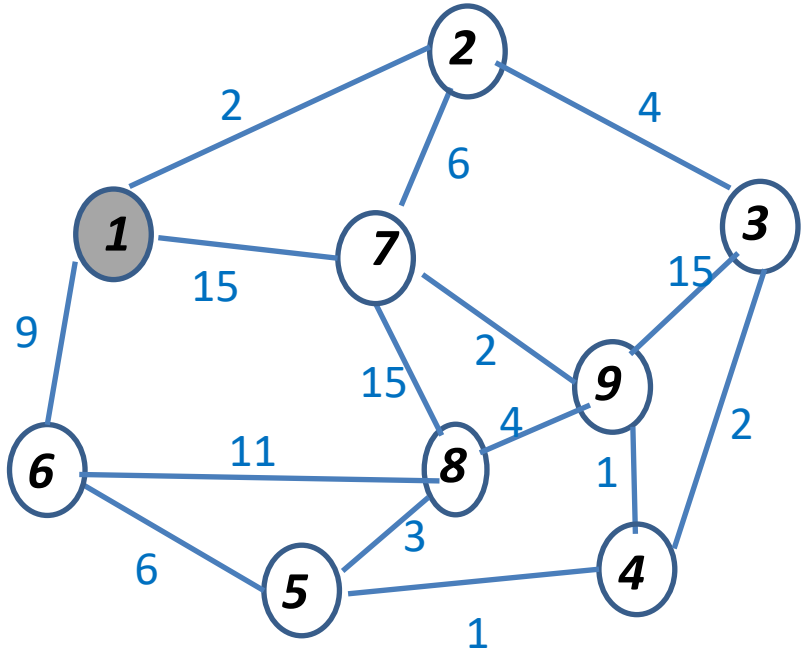
	1	2	3	4	5	6	7	8	9
Vorg.	--	1	2	3	4	1	2 ~~1~~	7	4 ~~3~~
DIST	0	2∝	6∝	8 ∝	9∝	9∝	8 ~~15~~∝	23∝	9 ~~21~~∝
Gewählt	x	x	x	x			x		

Init: Nachbarn von 1 in Rand R:2,6,7

1. Minimal 2: Wähle 2, Nachbarn von 2 in R — R:,6,7, 3
   Relaxiere 7, denn 1-2-7 = 8 < 15
2. Minimal 3: Wähle 3, Nachbarn von 3 in R — R:,6,7, 4, 9
3. Minimal 4 (oder 7): Nachbarn von 4 in R — R:,6,7, 9, 5
   Relaxiere 9, denn 1-2-3-4-9 = 9 < 21
4. Minimal 7: Nachbarn von 7 in R — R:,6 9, 5, 8

Algorithmus von Dijkstra – Beispiel

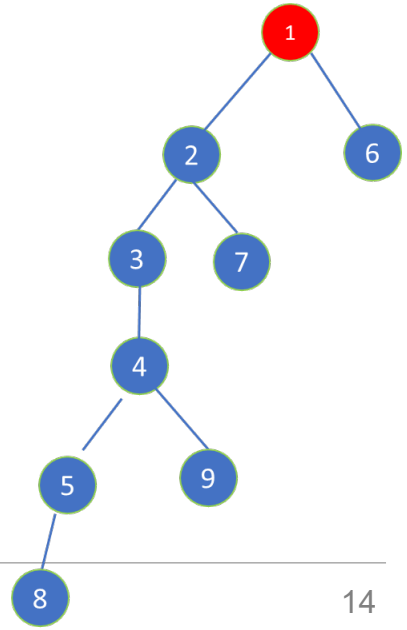
- Beispiel: Kürzester Weg von Startknoten (1) zu allen Zielnoten



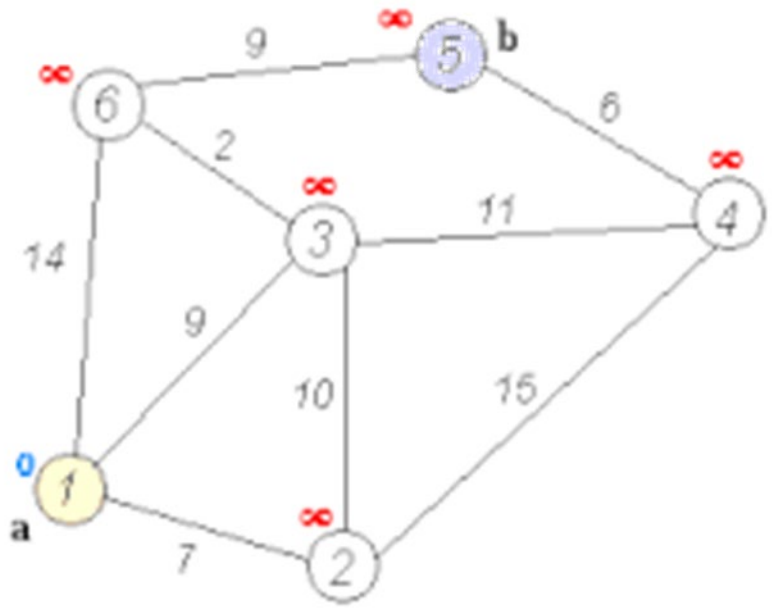
	1	2	3	4	5	6	7	8	9
Vorg.	--	1	2	3	4	1	2	5 ~~7~~	4
DIST	0	2	6	8	9	9	8	12 ~~23~~	9
Gewählt	x	x	x	x	x	x	x	x	x

R:,6 9, 5, 8

5. Minimal 5 (oder 6 oder 9): R:,6, 9,8,
   Nachbarn von 5 in R – keine neuen
   Relaxiere 8: 1-2-3-4-5-8 = 12 < 23
6. Minimal 6 – keine neuen Nachbarn R: 9,8,
7. Minimal 9 - keine neuen Nachbarn R: 8
8. Minimal 8 – Rand leer - fertig



Übung Dijkstra



Gesucht: Kürzester Weg von 1 nach 5

	1	2	3	4	5	6
Vorg.	--	1	1	3	6	3
DIST	0	7∝	9∝	20∝	20∝	11∝
Gewählt	x	x	x		x	x

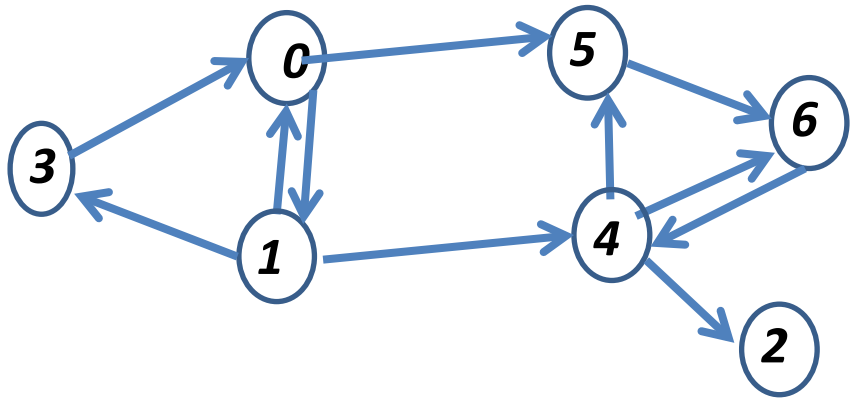
Kürzester Weg von 1 nach 5:
1->3->6->5 Kosten 20

Kürzeste Wege

- Problemstellung
- Algorithmus von Dijkstra
- **Gerichtete Graphen**
- Algorithmen in gerichteten Graphen

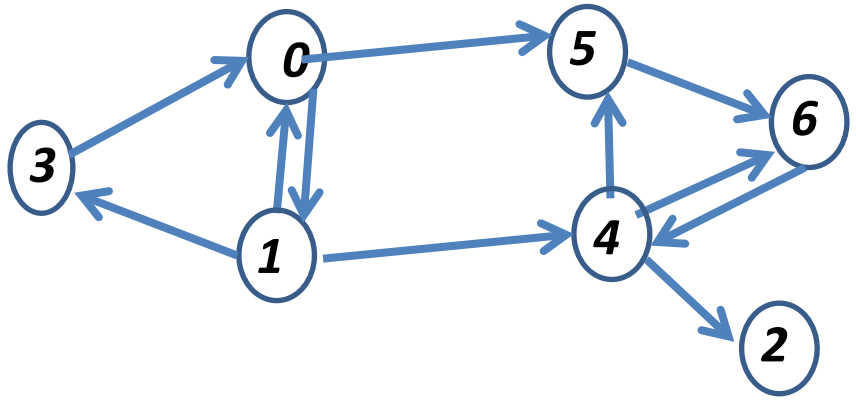
Gerichtete Graphen

- **Gerichteter Graph (*Digraph, directed graph)*** *G(V,E)* ist Graph mit gerichteten Kanten, d.h. die Kanten können nur in eine Richtung durchlaufen werden.
  - Darstellung **mit Pfeilen;** zwischen zwei Knoten maximal 2 Kanten
- **Gerichteter Pfad** von *v* nach w durchläuft Kanten in der richtigen Richtung
- **Zyklus –** Gerichteter Pfad von *v* nach v.
- **Eingangsgrad/Ausgangsgrad –** Anzahl der eingehenden/ausgehenden Kanten
- **Vorgänger** *u* eines Knotens v – Es gibt eingehende Kante <*u,v*> zu *v*
- **Nachfolger** *w* eines Knotens v – Es gibt ausgehende Kante <*v,w*> von *v*



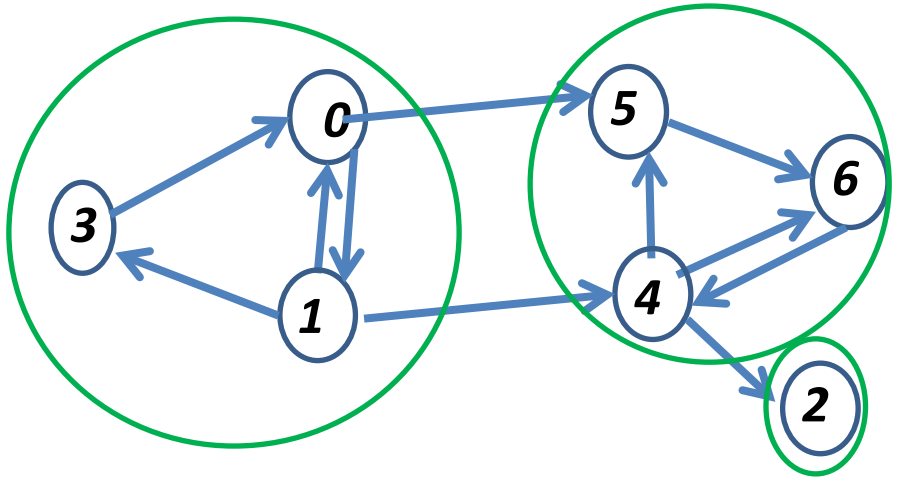
Datenstrukturen für gerichtete Graphen

- Analog zu ungerichteten Graphen, aber…
  - In Adjazenzmatrix $a_{ij} = 1$, falls $\langle i,j \rangle \in E$ (d.h. Kante von *i* nach *j*)
  - a_{ii} kann auch 1 sein
- Adjazenzliste
  - In der Liste zu jedem Knoten werden seine *Nachfolger* erfasst
  - Kanten sind nur einmal in der gesamten Adjazenzliste abgespeichert



Zusammenhangskomponenten im gerichteten Graph

- Ein Graph *G=(V,E)* heißt **zusammenhängend hinsichtlich *s***, wenn jeder Knoten in $V \setminus \{s\}$ von *s* aus erreichbar ist.
- Zwei Knoten *v* und *w* liegen in der selben **starken Zusammenhangskomponente**, wenn es einen gerichteten Pfad von v nach w und von w nach v gibt.
  - Gerichteter Graph wird in starke Zusammenhangskomponenten zerlegt (*partitioniert)*

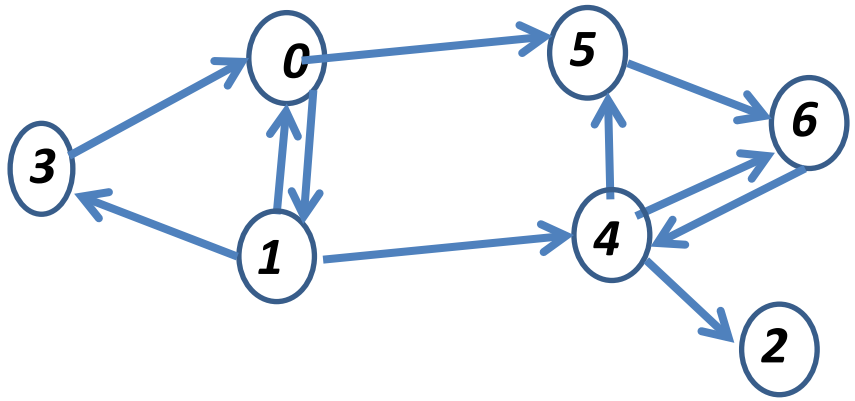


Kürzeste Wege

- Problemstellung
- Algorithmus von Dijkstra
- Gerichtete Graphen
- **Algorithmen in gerichteten Graphen**

Algorithmen für gerichtete Graphen

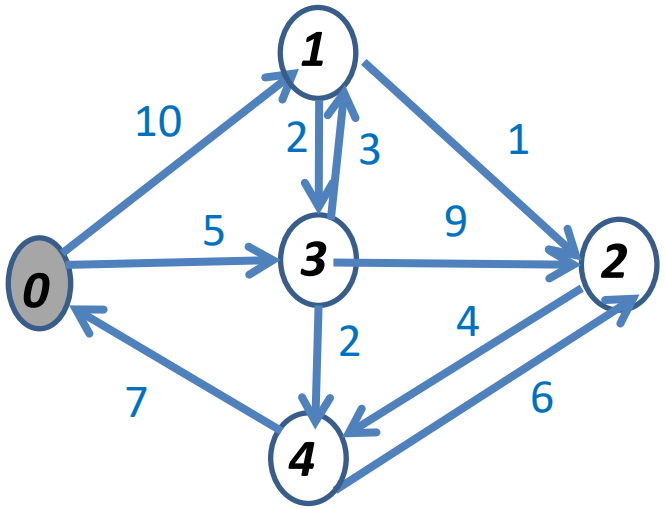
- Tiefensuche
  - Liefert i.a. keinen vollständigen Baum zu einem Startknoten
  - *DFS-Wald* statt DFS-Baum
  - Berechnung der starken Zusammenhangskomponente mit modifizierter Tiefensuche
- Breitensuche
  - Findet kürzesten Weg von Startkonten zu Zielknoten, sofern Pfad existiert



Algorithmen für gerichtete, kantengewichtete Graphen

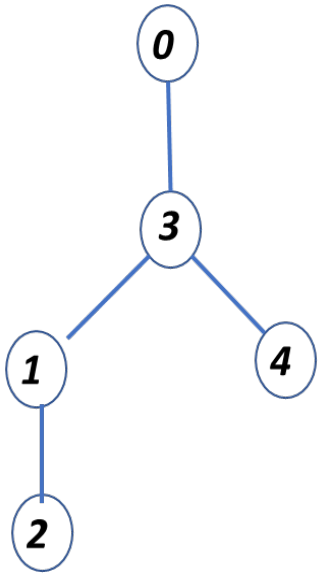
- Kürzeste Wege
  - Single Source – Algorithmus von Dijkstra

Übung: Kürzester Weg von Zentrale (0) zu allen Filialen (1-4)



	0	1	2	3	4
Vorgänger	-	~~0~~ 3	~~3~~ ~~4~~ 1	0	3
Distanz	0	∞ 10 8	∞ 14 13 9	∞ 5	∞ 7
Gewählt?	x	x	x	x	x

Rand: 1,3 (Nachbarn von 0),
1. Minimum 3: 3 aus Rand, 2 und 4 in Rand, Relaxieren 1
2. Minimum 4: 4 aus Rand, Relaxieren 2
3. Minimum 1: 1 aus Rand, Relaxieren 2
4. Minimum 2: 2 aus Rand



Zusammenfassung Kürzeste Wege

- Es gibt kürzesten Weg zwischen zwei Knoten, wenn Pfad existiert und kein Pfad einen Zyklus mit negativen Kantengewichten enthält
- Algorithmus von Dijkstra berechnet für nichtnegative Kantengewichte
  - Kürzesten Weg von einem Startknoten *s* zu einem Zielknoten *t*
  - Kürzesten Weg von einem Startknoten *s* zu *allen* Knoten
- Suche nach kürzestem Weg für *alle Knotenpaare*
  - Dijkstra-Algorithmus für alle Knoten als Startknoten → Aufwändig
  - Effizientere Lösung: Algorithmus von *Floyd-Warshall* (nicht in der Vorlesung)