

Programmierkurs Python II

Stefan Thater & Michaela Regneri
Universität des Saarlandes
FR 4.7 Allgemeine Linguistik (Computerlinguistik)



Übersicht

- Topologische Sortierung (einfach)
- Kürzeste Wege finden (ziemlich einfach)
- Netzwerke
 - Flüsse (flows) berechnen (mittel-einfach)
 - Schnitte (cuts) berechnen (ziemlich einfach)
 - maximal flow / minimum cut (nützlich 😊)

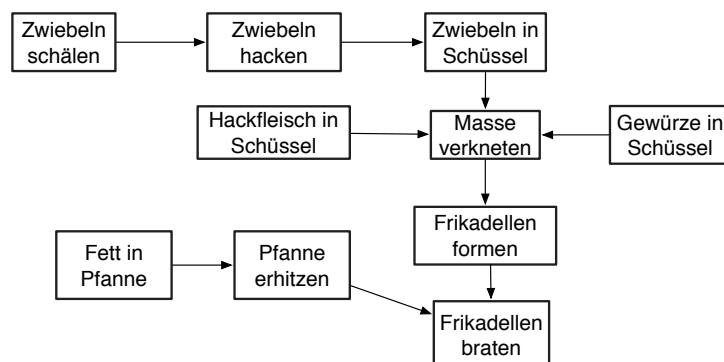
Ein Graph zum Frikadellen-Machen

„**X** → **Y**“: X muss erledigt sein, damit Y gemacht werden kann

- Zwiebel Schälen → Zwiebeln hacken
- Zwiebeln hacken → Zwiebeln in die Schüssel
- Hackfleisch auspacken → Hackfleisch in die Schüssel
- ∅ → Gewürze in die Schüssel
- [alles in die Schüssel] → Masse Verkneten
- Masse Verkneten → Frikadellen formen
- Frikadellen Formen → Frikadellen braten
- [...]

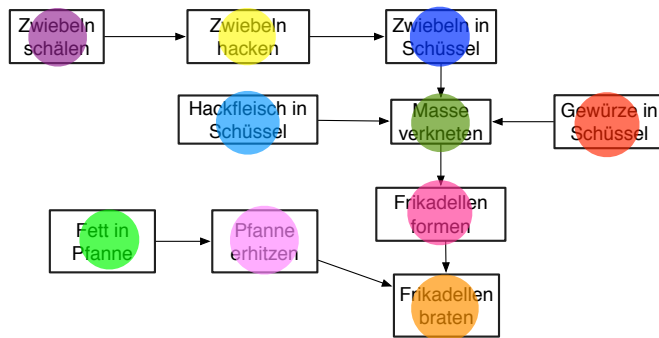
3

Ein Graph zum Frikadellen-Machen

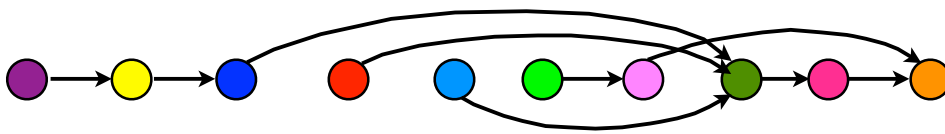


4

Topologische Sortierung - graphische Variante

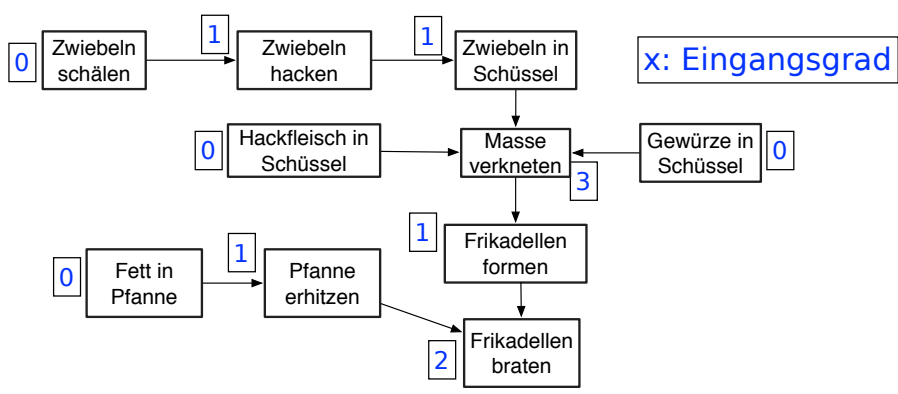


lineare Ordnung,
alle Kanten
zeigen nach
rechts



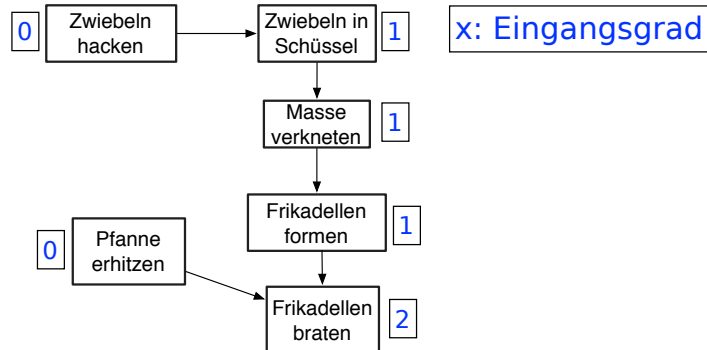
Topologische Sortierung - Algorithmus (im Bild)

Schritt 1



Topologische Sortierung - Algorithmus (im Bild)

Schritt 2

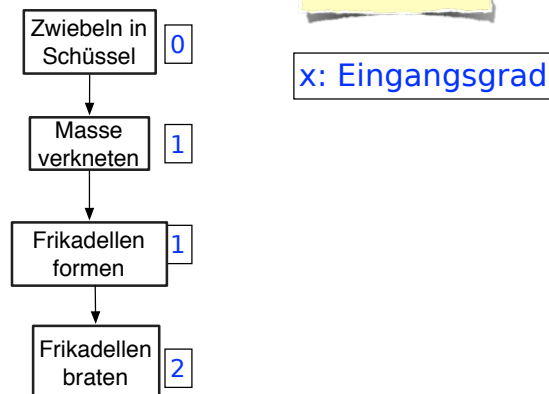


- Zwiebeln schälen
- Fett in Pfanne
- Gewürze in Schüssel
- Hackfleisch in Schüssel

Beliebige Ordnung untereinander

Topologische Sortierung - Algorithmus (im Bild)

Schritt 3



- Zwiebeln schälen
- Fett in Pfanne
- Gewürze in Schüssel
- Hackfleisch in Schüssel
- Zwiebeln hacken
- Pfanne erhitzen

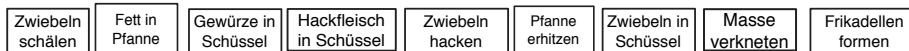
Topologische Sortierung - Algorithmus (im Bild)

Schritt 6

Komplexität:
bester Fall: $O(n)$
worst case: $O(n^2)$

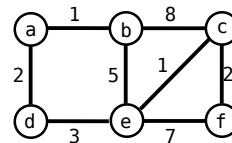
x: Eingangsgrad

Frikadellen
braten 0



9

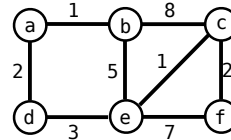
Kürzeste Wege finden



- „Single source shortest path problem“
- Aufgabe: Gegeben einen Knoten im Graph, finde die kürzesten Wege zu allen anderen Knoten
- Meistens assoziiert mit Kantengewichten (kürzester Weg = *billigster* Weg)
- Der Weg vom Startknoten zu unerreichbaren Knoten kann unendlich sein (unzusammenhängende ungerichtete / nicht stark zusammenhängende gerichtete Graphen)

10

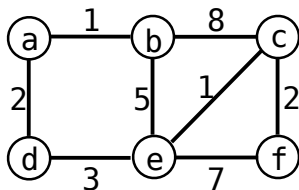
Dijkstra-Algorithmus



- Initialisierung:
 - Startknoten: Distanz 0, *permanent*, aktiv
 - andere: Distanz ∞ , *temporär* (nicht aktiv)
- So lange es Knoten mit temporären Nachbarn gibt...
 - Berechne die Distanz der temp. Nachbarn des aktiven Knoten (Distanz aktiver Knoten + Kantengewicht)
 - wenn berechnete Distanz > notierter Distanz:
 - aktualisiere notierte Distanz;
 - notiere aktiven Knoten als Vorgänger
 - neuer aktiver Knoten: Knoten mit kleinster (Gesamt-)Distanz; markiere den als permanent

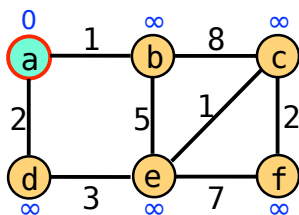
11

Dijkstra-Algorithmus



Startknoten hier: a

Initialisiere Distanzen:
Startknoten = 0, alle anderen = ∞

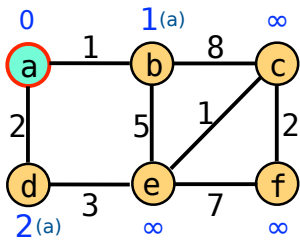


Initialisiere Markierungen:
Startknoten = permanent
alle anderen = temporär

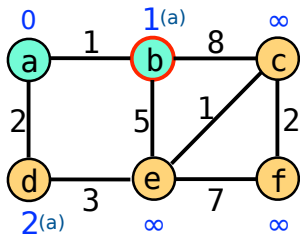
Initialisiere aktiven Knoten:
Startknoten

12

Dijkstra-Algorithmus

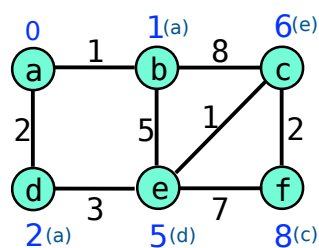
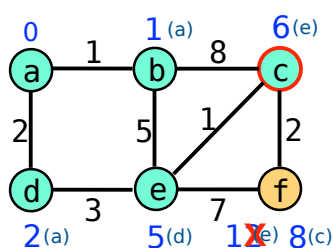
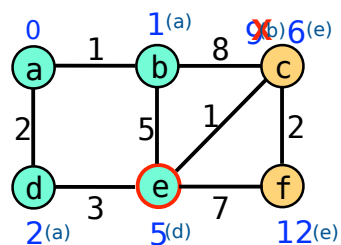
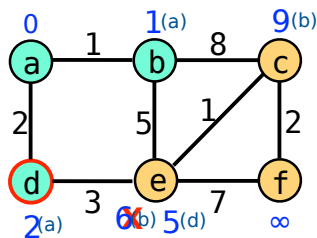
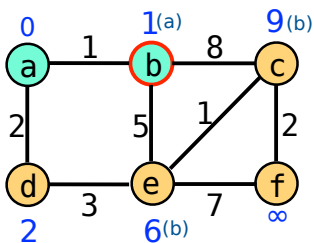


- Berechne alle Nachbar-Distanzen zum aktiven Knoten (nur zu temporären Knoten)
- Wenn die berechnete Distanz kleiner ist als die bisher gefundene, aktualisiere Distanz und Vorgänger



- neuer aktiver Knoten: temporärer Knoten mit kleinster Distanz
- markiere neuen aktiven Knoten als permanent

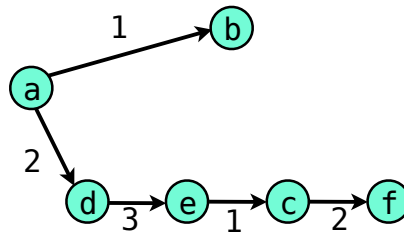
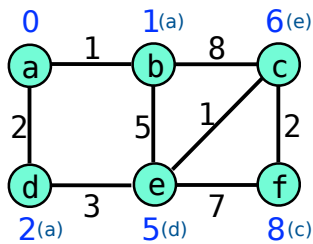
Dijkstra-Algorithmus



fertig

Dijkstra-Algorithmus - Ergebnis

- jeder Knoten: minimale Distanz, und direkter Vorgänger
- Ableitbarer „Spannbaum“: ein Teilgraph des Graphen, der ein Baum ist und alle Knoten des Graphen enthält
- die enthaltenen Kanten markieren hier die kürzesten Pfade



15

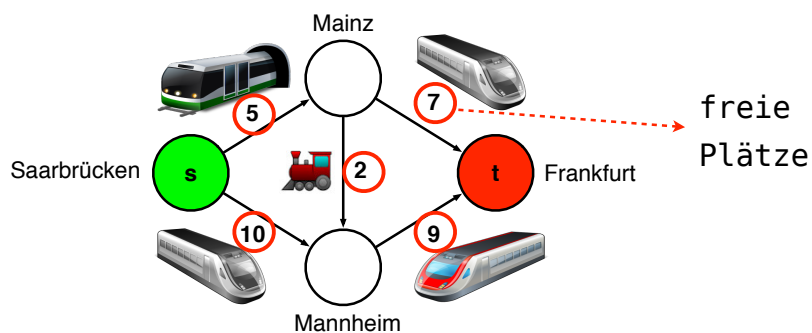
Kürzeste Wege finden

- Komplexität von Dijkstra: $O(\text{Knoten} * \log(\text{Knoten}) + \text{Kanten})$
- SSSP mit neg. gewichteten Kanten
 - Dijkstra erlaubt keine negativen Gewichte
 - SSSP mit negativen Gewichten (aber ohne negative Zyklen):
Bellman-Ford-Algorithmus
- *All pairs shortest path problem*
 - Aufgabe: berechne für alle Paare von Knoten die kürzesten Pfade zwischen den Knoten
 - *Floyd-Warshall-Algorithmus*

16

Netzwerke

- Netzwerke sind gerichtete, gewichtete Graphen mit einer Quelle (s , "start") und einer Senke (t , "target")
- Ein Netzwerk dient zur Darstellung von *Flüssen*
- Gewichte = *Kapazitäten* z.B. $c(\text{Mainz}, \text{Frankfurt}) = 7$



17

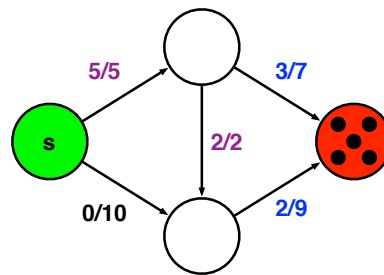
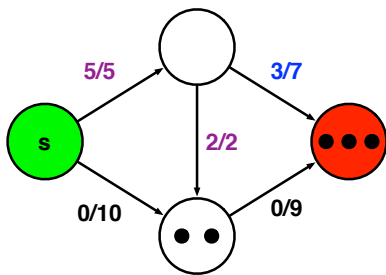
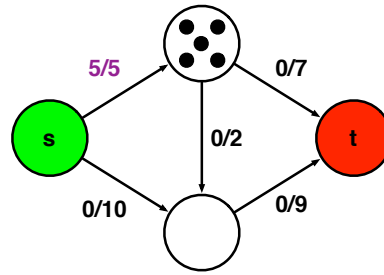
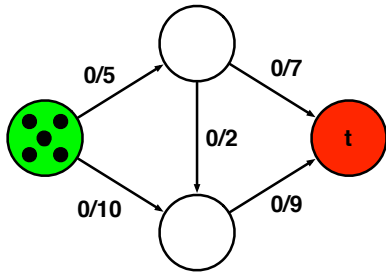
Netzwerke

- Netzwerke simulieren den Fluss (\sim Transport von Einheiten) von s nach t
- Protokolliert wird dabei, wieviel Kapazität der Kanten "verbraucht" ist
- Wenn es keinen Pfad mit (durchgängig) freien Kapazitäten von s nach t gibt, kann kein Fluss mehr stattfinden
- Regeln für den Fluss durch Knoten:
 - alles, was rein geht, muss wieder raus
 - wenn es zwei adjazente Pfade mit freien Kapazitäten gibt, kann ein beliebiger gewählt werden

18

$2/5$ = "2 von 5 Kapazitäten belegt",
 $c = 5$, $\text{flow} = 2$, $\text{residual} = 3$

Netzwerke: Beispiel



19

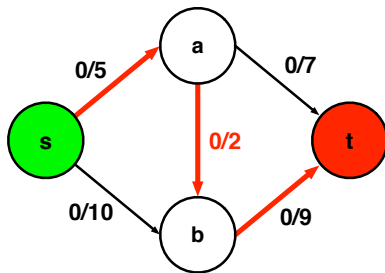
Netzwerke: Maximaler Fluss (max. flow) *Ford-Fulkerson-Algorithmus*

- Aufgabe: berechne die maximale Gesamtkapazität des Netzwerks ("wieviele Leute können von s nach t ?")
- Grundsätzliche Idee: so lange es Pfade von s nach t mit freier Kapazität gibt, schicke die maximale Kapazität durch den Pfad
 - die Kapazität eines Pfades ist die kleinste Kapazität seiner Kanten
 - die Auswahl ist nicht deterministisch!
- Die Gesamtkapazität ist dann die Summe der maximalen Pfadkapazitäten

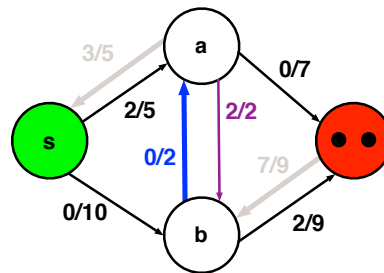
20

Ford-Fulkerson-Algorithmus

- Für jede verbrauchte Kapazität gibt es eine invertierte Kante mit der gleichen freien Kapazität ("Gutschrift"):



Gesamtkapazität bisher: 0
 Nächster Pfad: s-a-b-t
 Max. Pfadkapazität: 2



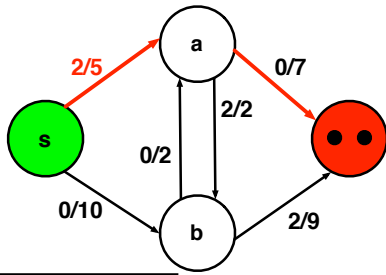
Gesamtkapazität bisher: 2
 ...

Ford-Fulkerson-Algorithmus: *finde max_flow(Netzwerk)*

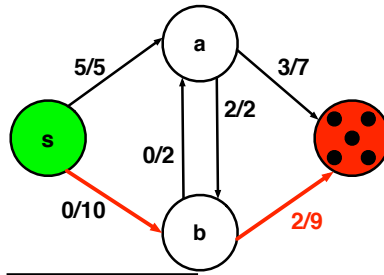
$$\text{residual}(u,v) = c(u,v) - \text{flow}(u,v)$$

- modifiziere das Netzwerk (temporär):
 - jede Kante bekommt eine invertierte Kante (wenn noch nicht da)
 - $c(u,v) = c(v,u)$, mit $\text{flow}(v,u) = c(u,v)$
- so lange es Pfade von s nach t mit freien Kapazitäten gibt
 - wähle einen zufälligen Pfad P mit durchgehend freien Kapazitäten
 - $c(P) = \min(\text{residual}(u_1,v_1), \text{residual}(u_2,v_2), \dots, \text{residual}(u_3,v_3))$ für alle (u,v) in P
 - update für alle (u,v) in P :
 - $\text{flow}(u,v) += c(P)$
 - $\text{flow}(v,u) -= c(P)$
 - $\text{max_flow}(\text{Netzwerk}) += c(P)$

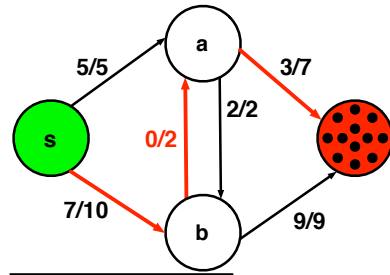
Ford-Fulkerson-Algorithmus



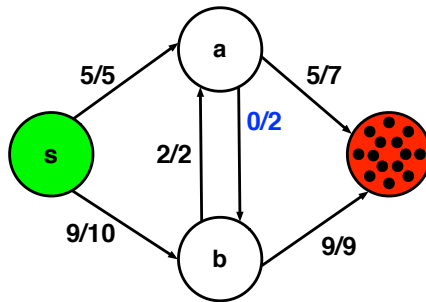
max_flow: 2
 Pfad: s-a-t
 c(Pfad): 3



max_flow: 5
 Pfad: s-b-t
 c(Pfad): 7



max_flow: 12
 Pfad: s-b-a-t
 c(Pfad): 2

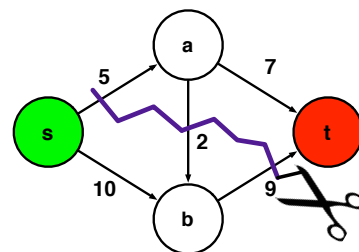


max_flow: 14
 kein Pfad mit durchgängigen Kapazitäten übrig -> fertig.

23

Netzwerke: Minimaler Schnitt (min-cut)

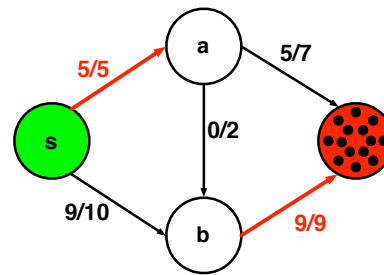
- Ein *Schnitt* teilt ein Netzwerk in zwei Teilgraphen S und T, wobei s in S und t in T ist.
- Die *Kapazität* eines Schnittes ist die Summe aller Kapazitäten von Kanten, die von S nach T zeigen
- ein *minimaler Schnitt* ist der (oder: ein) Schnitt mit der kleinsten möglichen Kapazität
- die Kapazität des minimalen Schnittes ist der maximale Fluss durch das Netzwerk $c(\text{min-cut}(N)) = \text{max_flow}(N)$



24

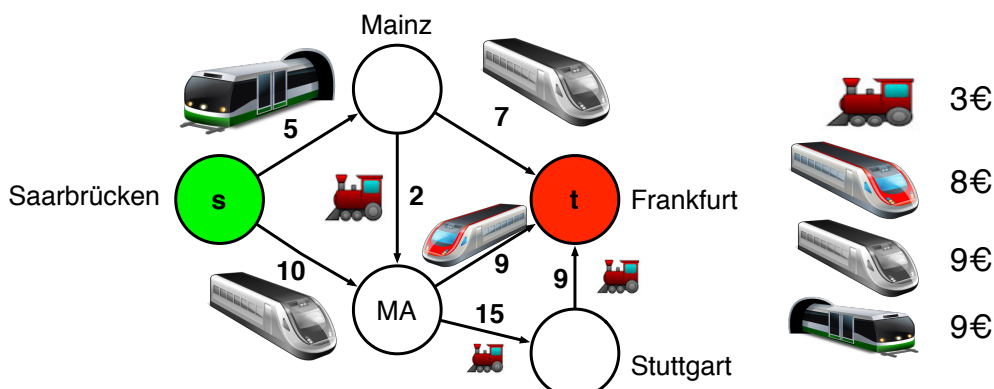
max-flow & min-cut

- der minimale Schnitt kann über den maximalen Fluss berechnet werden
- geschnitten werden Kanten
 - deren Kapazität ausgereizt ist
 - die bei Tiefensuche von der Quelle zuerst besucht werden
- Achtung: hier zählen nur die "Original-Kanten", nicht die inversen aus der Modifikation
- Im Beispiel ist ein minimaler Schnitt $S = \{s, b\}$, $T = \{a, t\}$
- Die Lösung ist nicht unbedingt eindeutig!



max-flow + Kosten

- es kann mehrere maximale Flüsse im gleichen Netzwerk geben
- Manchmal möchte man zusätzliche Kantengewichte haben, um den günstigsten maximalen Fluss zu berechnen:



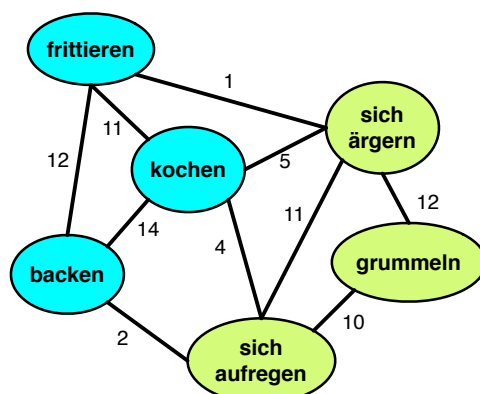


*Aha. Toll. Das ist wieder Zeug das wir
bloß auswendig lernen müssen und das
gar keine Bedeutung für uns hat.*
BLABLABLABLABLABLABLABLABLA

27

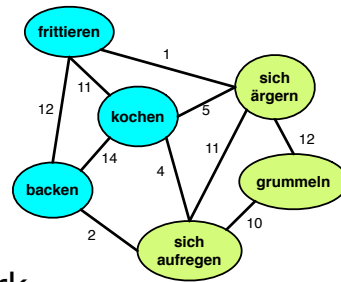
Ähäm.

- die max-flow / min-cut Berechnung ist eine der wichtigsten Grundlagen für graph-basiertes *Clustering*
- “geclustert” werden z.B. Graphen, deren Kanten mit Ähnlichkeiten gewichtet sind, z.B. so:



28

Wie jetzt?



- man kann aus jedem gewichteten Graphen ein Netzwerk machen (Quelle -> alle Knoten -> Senke)
- Ein (minimaler) Cut teilt den Graphen in Teilgraphen mit möglichst unähnlichen Knoten
- rekursiv angewendet gibt es mehrere Cluster
- Auf das Thema Clustering kommen wir im Kapitel maschinelles Lernen noch zurück

29

Zusammenfassung

- Heute: verschiedene Graph-Algorithmen
- Topologische Sortierung: „Plan-Management“
- Dijkstra: Kürzeste Wege
- Netzwerke
 - maximaler Fluss
 - minimaler Schnitt
 - Ausblick auf Anwendungen

30