Programmierkurs Python II

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

Organisatorisches

- Übung ab jetzt im Seminarraum (statt im CIP-Raum)
- Für die TaCoS gibt's frei (31.05.)
  - unbedingter Hingeh-und-Mitmach-Befehl!
  - die “Wunsch-Session” muss dafür dann ausfallen
  - aber: wenn bestimmte Themen ausführlicher wiederholt werden sollen, machen wir das in der letzten Sitzung

Heute: mehr zu Graphen

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

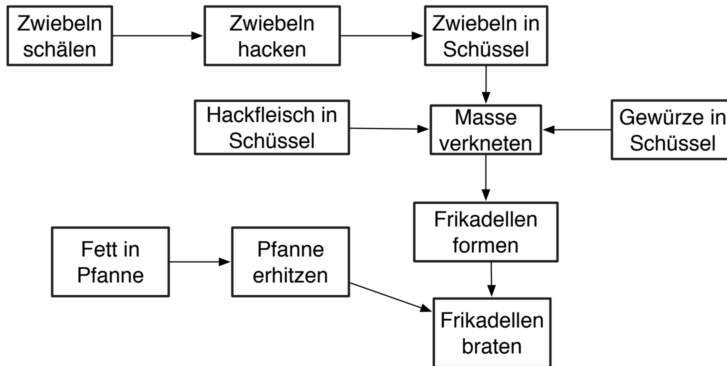
Ein Graph zum Frikadellen-Machen

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

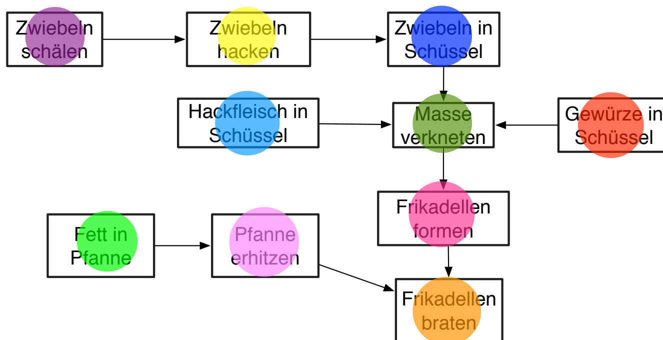
Zwiebel Schälen	→	Zwiebeln hacken
Zwiebeln hacken	→	Zwiebeln in die Schüssel
Hackfleisch auspacken	→	Hackfleisch in die Schüssel
∅	→	Gewürze in die Schüssel
[*alles* in die Schüssel]	→	Masse Verkneten
Masse Verkneten	→	Frikadellen formen
Frikadellen Formen	→	Frikadellen braten

[...]

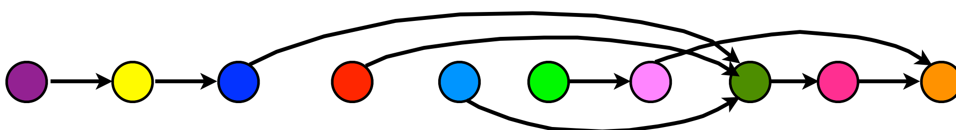
Ein Graph zum Frikadellen-Machen



Topologische Sortierung - graphische Variante

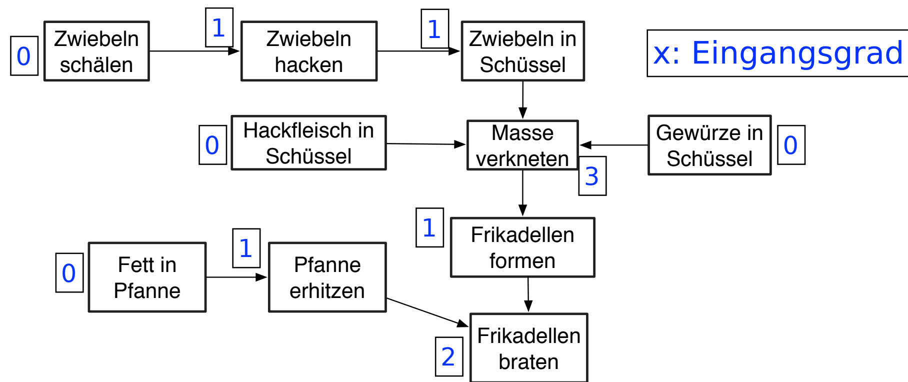


lineare Ordnung, alle Kanten zeigen nach rechts



Topologische Sortierung - Algorithmus (im Bild)

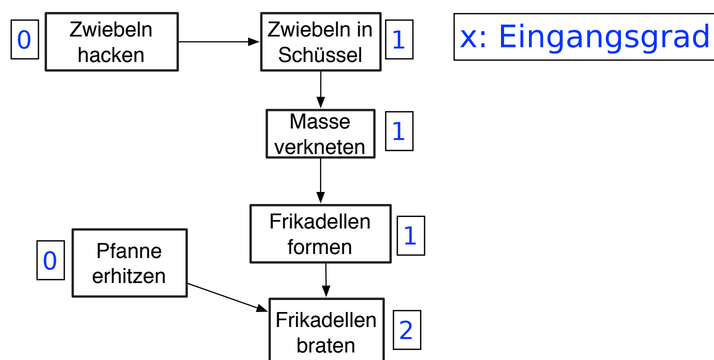
Schritt 1



x: Eingangsgrad

Topologische Sortierung - Algorithmus (im Bild)

Schritt 2



x: Eingangsgrad

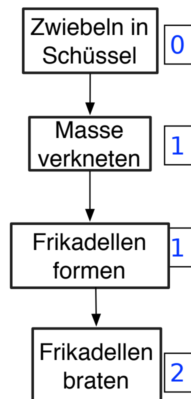
Zwiebeln schälen | Fett in Pfanne | Gewürze in Schüssel | Hackfleisch in Schüssel

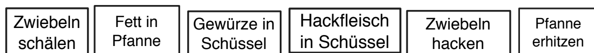
Beliebige Ordnung untereinander

Topologische Sortierung - Algorithmus (im Bild)

Schritt 3

x: Eingangsgrad



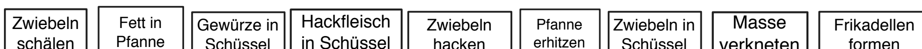


Topologische Sortierung - Algorithmus (im Bild)

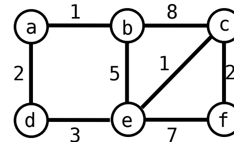
Schritt 6

x: Eingangsgrad

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

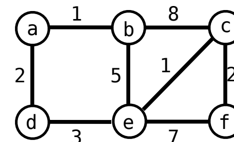


Kürzeste Wege finden



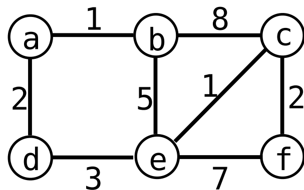
- *„Single source shortest path problem“*
- Aufgabe: Gegeben einen Knoten im Graph, finde die kürzesten Wege zu allen anderen Knoten
- Meistens assoziiert mit Kantengewichten (kürzester Weg = *billigster* Weg)
- Der Weg vom Startknoten zu unerreichbaren Knoten kann unendlich sein (unzusammenhängende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

Dijkstra-Algorithmus



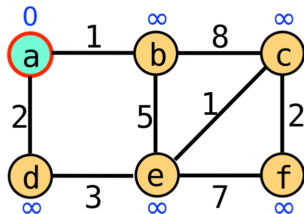
- Initialisierung:
  - Startknoten: Distanz 0, *permanent*, aktiv
  - andere: Distanz ∞, *temporär* (nicht aktiv)
- So lange es Knoten mit temporären Nachbarn gibt...
  - Berechne die Distanz der temp. Nachbarn des aktiven Knoten (Distanz aktiver Knoten + Kantengewicht)
  - wenn berechnete Distanz >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

Dijkstra-Algorithmus



Startknoten hier: a

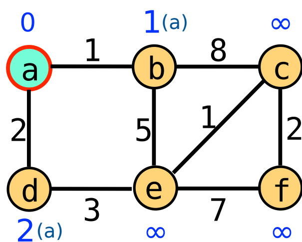
Initialisiere Distanzen:
Startknoten = 0, alle anderen = ∞



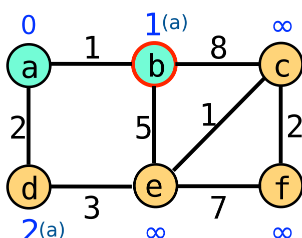
Initialisiere Markierungen:
Startknoten = permanent
alle anderen = temporär

Initialisiere aktiven Knoten:
Startknoten

Dijkstra-Algorithmus

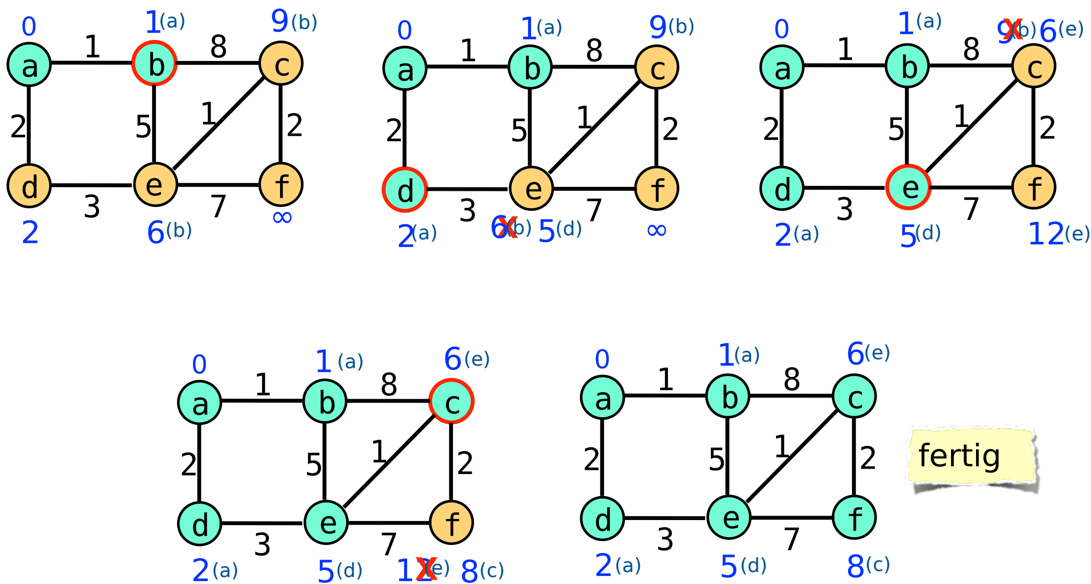


- Berechne alle Nachbar-Distanzen zum aktiven Knoten (nur zu temporären Knoten)
- Wenn die berechnete Distanz kleiner ist als die bisher gefundene, aktualisiere Distanz und Vorgänger



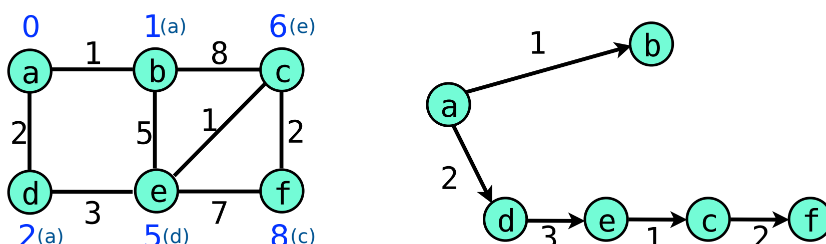
- neuer aktiver Knoten: temporärer Knoten mit kleinster Distanz
- markiere neuen aktiven Knoten als permanent

Dijkstra-Algorithmus



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

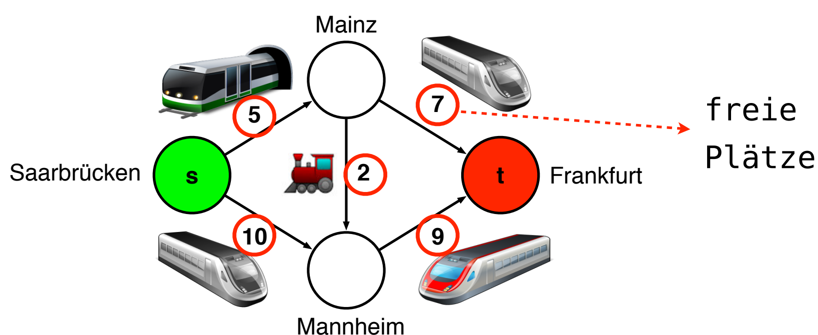


Kürzeste Wege finden

- Komplexität von Dijkstra: *O(Knoten * log(Knoten) + 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*

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****(Mainz,Frankfurt) = 7*

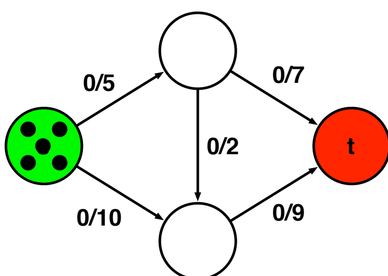


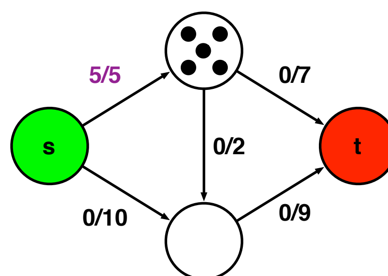
Netzwerke

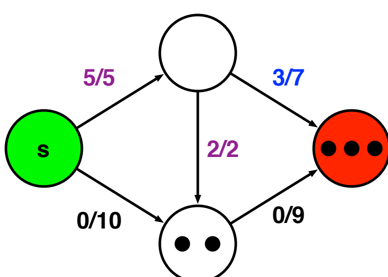
- Netzwerke simulieren den Fluss (~Transport von Einheiten) von *s* nach *t*
- Protokolliert wird dabei, wieviel Kapazität der Kanten "verbraucht" ist
- Wenn es keinen Pfad mit (durchgängig) freien Kapazitäten von *s* nach *t* gibt, kann kein Fluss mehr stattfinden
- Regeln für den Fluss durch Knoten: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

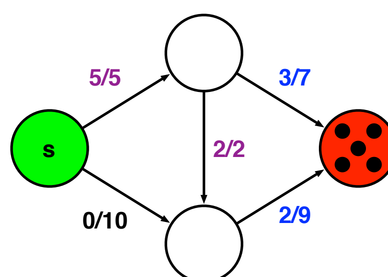
2/5 = "2 von 5 Kapazitäten belegt", c = 5, flow=2, *residual* = 3

Netzwerke: Beispiel





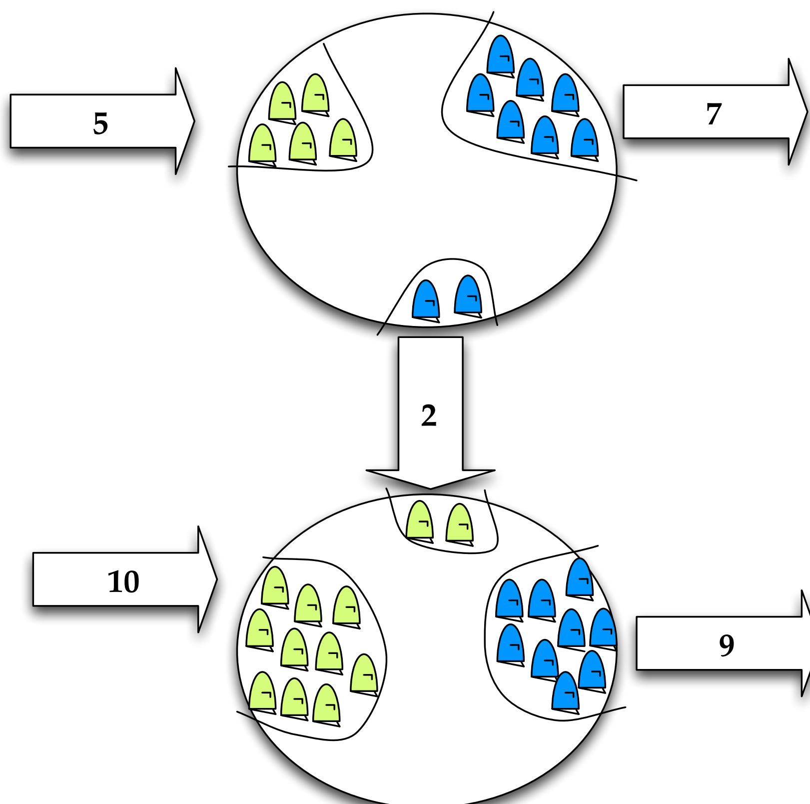




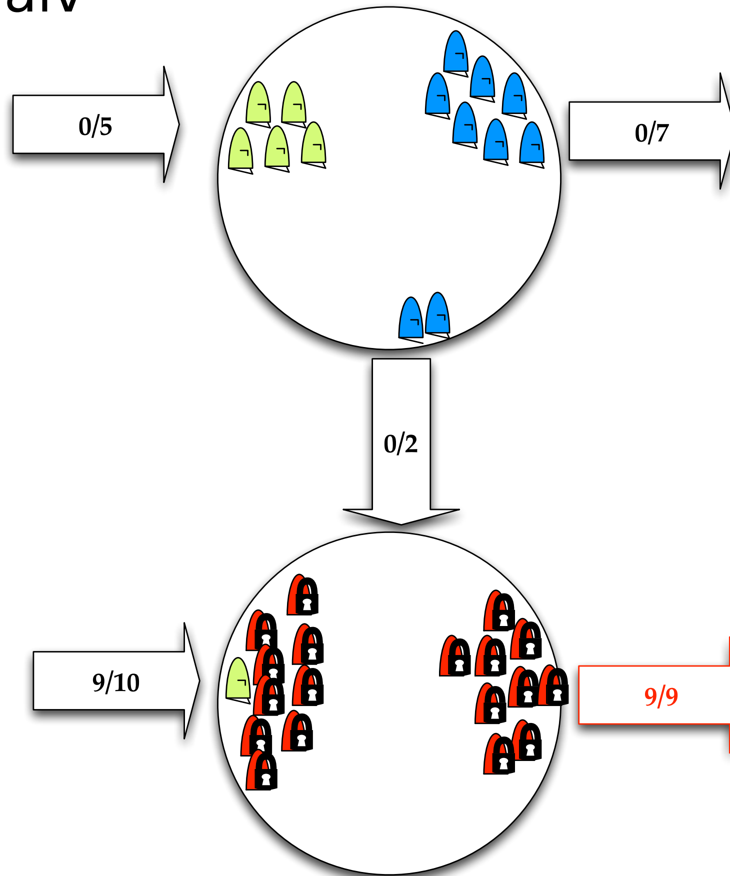
Netzwerke: Maximaler Fluss (max. flow)
Ford-Fulkerson-Algorithmus

- 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

Max flow "naiv"

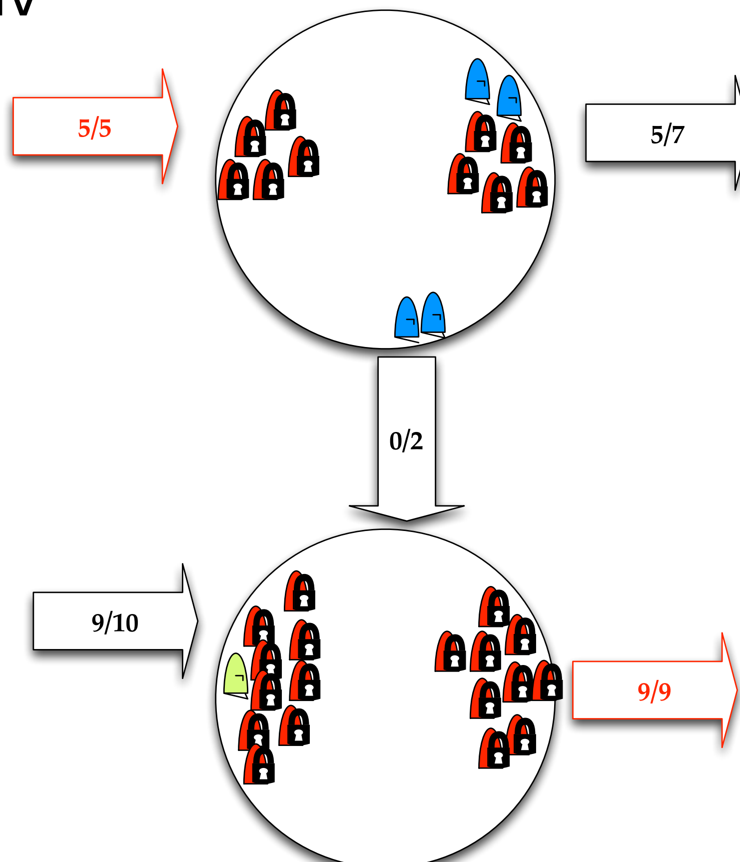


Max flow "naiv"




```
Durchlauf:    1
Flow bisher: 9
```


Max flow "naiv"

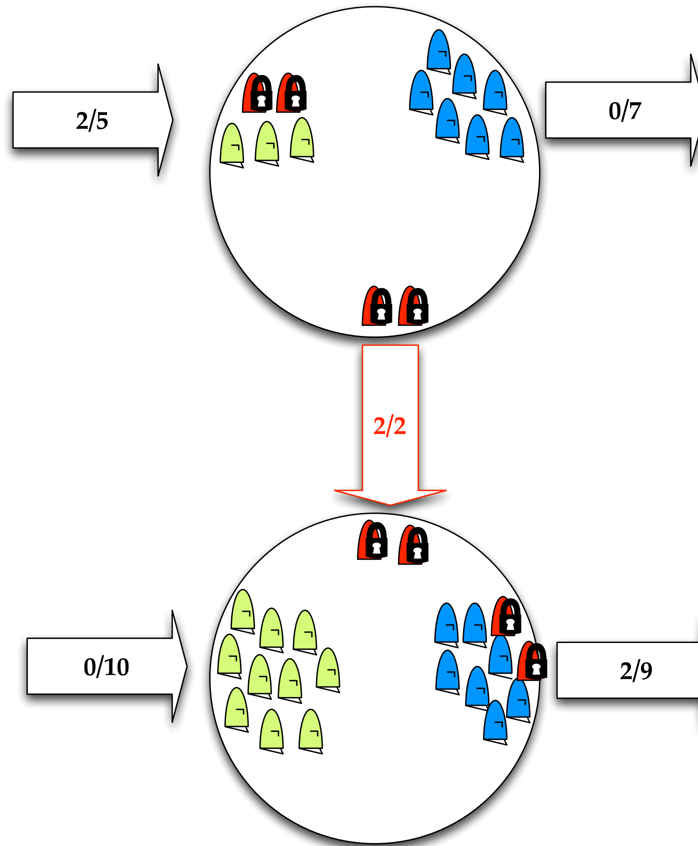



```
Durchlauf:    2
Flow bisher: 14
```

-> Kapazität: 14

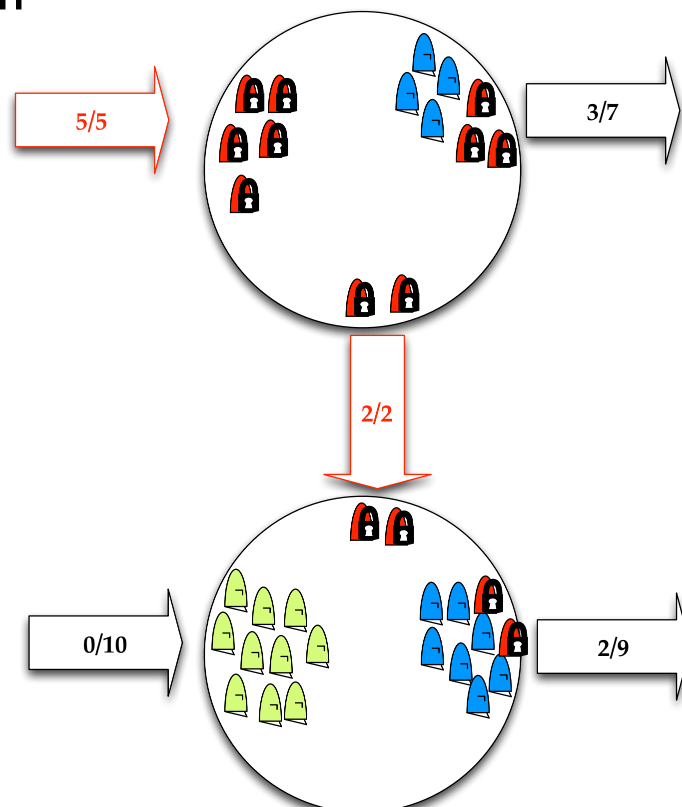
Max flow “naiv”, II

Durchlauf: 1
Flow bisher: 2

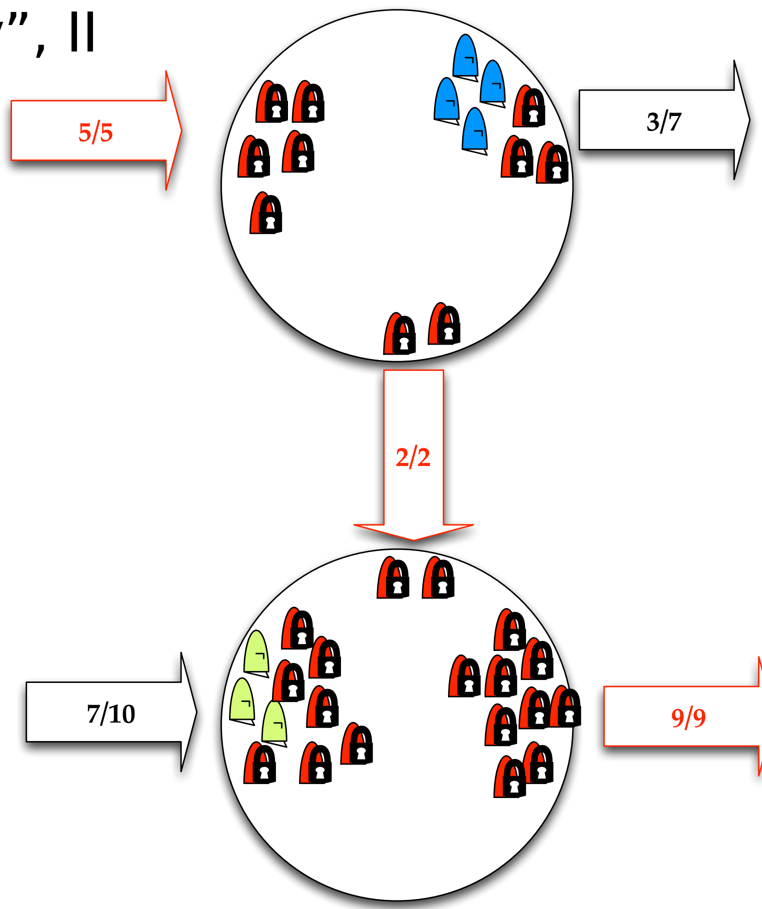


Max flow “naiv”, II

Durchlauf: 2
Flow bisher: 5



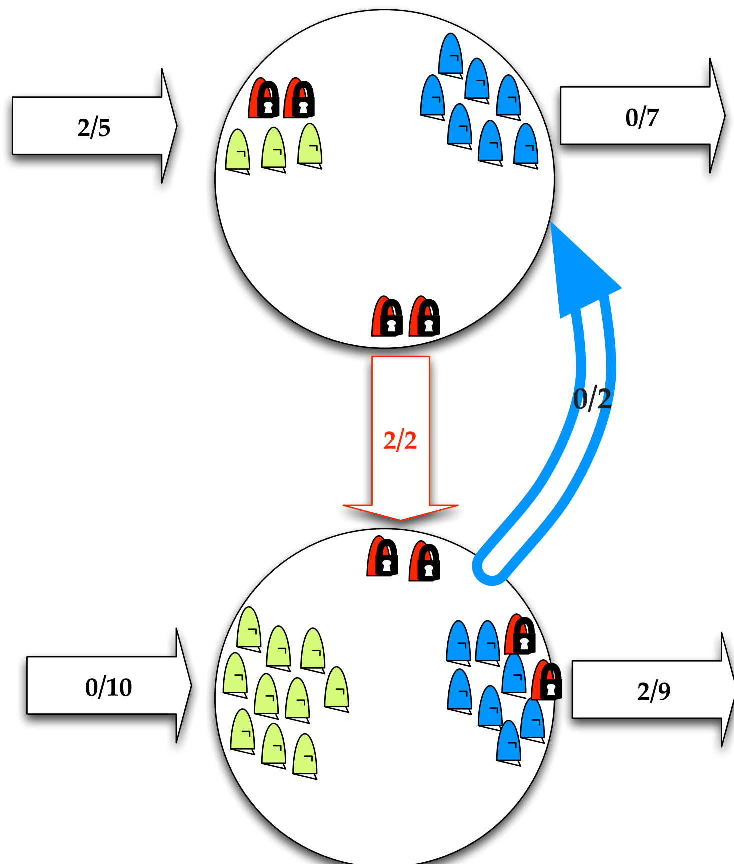
Max flow “naiv”, II




```
Durchlauf:   3
Flow bisher: 12
```

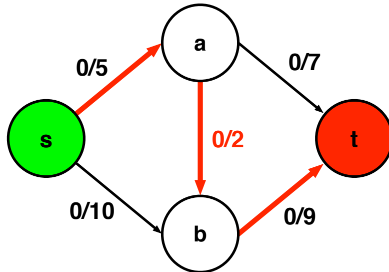
-> Kapazität: 12 !?

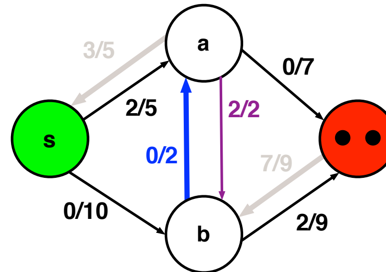
Max flow: der “Trick”



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
```

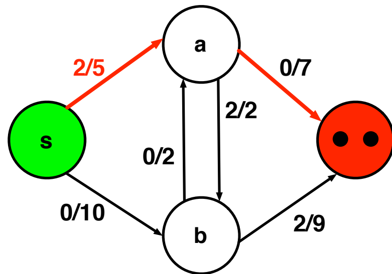
```
Gesamtkapazität bisher: 2
...
```


residual(u,v) = c(u,v) - flow(u,v)

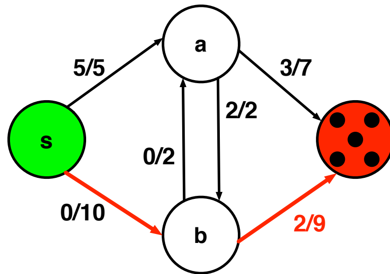
Ford-Fulkerson-Algorithmus: *finde max_flow(Netzwerk)*

- modifiziere das Netzwerk (temporär):
  - jede Kante bekommt eine invertierte Kante (wenn noch nicht da)
  - c(u,v) = c(v,u), mit 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(residual(u1,v1), residual(u2,v2),..., residual(u3,v3)) für alle (u,v) in P
  - update für alle (u,v) in *P*:
    - flow(u,v) += c(*P*)
    - flow(v,u) -= c(*P*)
  - max_flow(Netzwerk) += c(*P*)

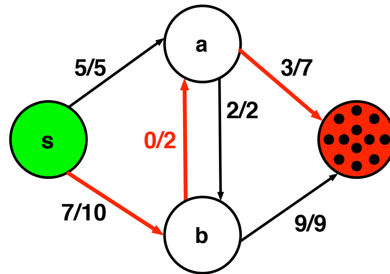
Ford-Fulkerson-Algorithmus



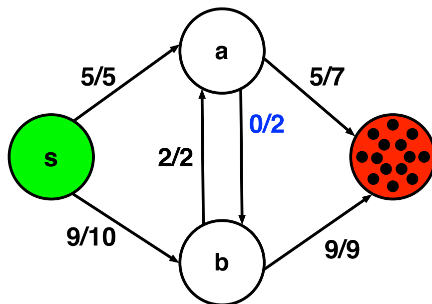
max_flow: 2
Pfad: s-a-t
c(Pfad): 3



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



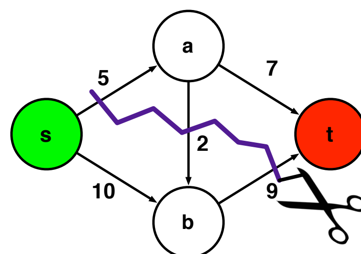
max_flow: 12
Pfad: s-b-a-t
c(Pfad): 2



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

Netzwerke: Minimaler Schnitt (min-cut)

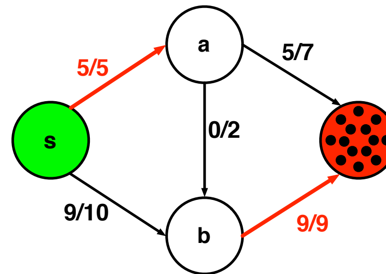
- Ein *Schnitt* teilt ein Netzwerk in zwei Teilgraphen S ung T, wobei s in S und t in T ist.



- 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 Schittes ist der maximale Fluss durch das Netzwerk *c(min-cut(N)) = max_flow(N)*

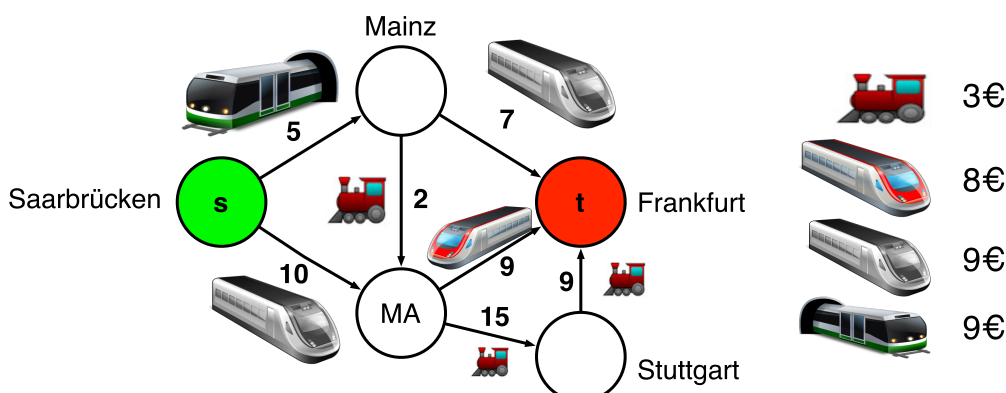
max-flow & min-cut

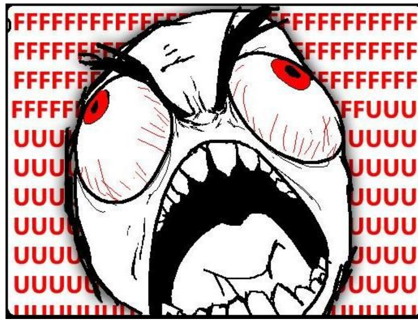
- der minimale Schnitt kann über den maximalen Fluss berechnet werden
- geschnitten werden Kannten
  - deren Kapazität ausgereizt ist
  - die bei Tiefensuche von der Quelle zuerst besucht werden
- Achtung: hier zählen nur die "Original-Kanten", nicht die inversen aus der Modifikation
- Im Beispiel ist 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
- Machmal möchte man zusätzliche Kantengewichte haben, um den günstigsten maximalen Fluss zu berechnen:

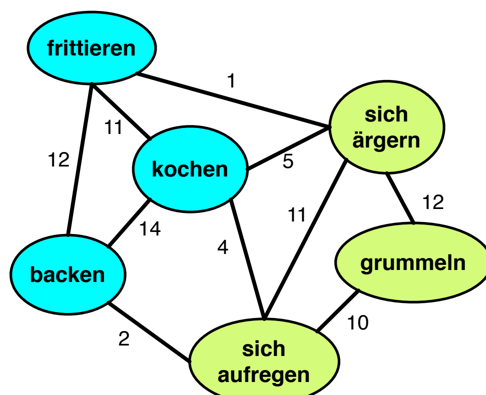




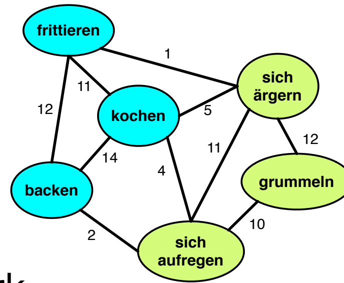
Aha. Toll. Das ist wieder Zeug das wir bloß auswendig lernen müssen und das gar keine Bedeutung für uns hat. BLABLABLABLABLABLABLABLABLABLA

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:



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

Zusammenfassung

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