2.6 Graphen

2.6.5 Maximaler Fluss

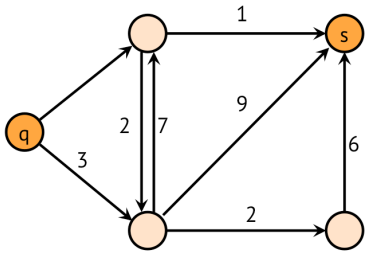
Flussnetzwerke

- Ein Flussnetzwerk (G,c) ist ein Graph G=(V,E) zusammen mit einer Kapazitätsfunktion $c:E \rightarrow \mathbb{N}_{>0}$ und zwei Knoten q,s aus V mit

  Eingangsgrad(q) = Ausgangsgrad(s) = 0

- Der Knoten q heißt Quelle, der Knoten s heißt Senke.

Flussnetzwerke

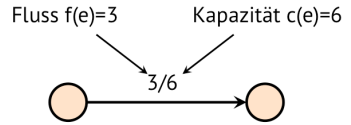


Flussnetzwerke

- Ein Fluss in einem Netzwerk $G=(V,E)$ ist eine reellwertige Funktion $f : E \to R$ mit den Eigenschaften
  - Flusserhaltung und
  - Kapazitätsbeschränkung

Flussnetzwerke

- Kapazitätsbeschränkung: $\forall e \in E: 0 \leqslant f(e) \leqslant c(e)$



Flussnetzwerke

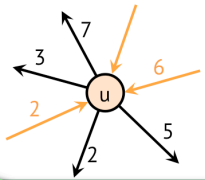
- Flusserhaltung: $\forall u \in V - \{q, s\}$:

$$\underbrace{\sum_{v:(v,u)\in E} f(v,u)}_{f_{in}(u)} = \underbrace{\sum_{v:(u,v)\in E} f(u,v)}_{f_{out}(u)}$$

Flussnetzwerke

- Flusserhaltung: $\forall u \in V - \{q, s\}$:

$$\underbrace{\sum_{v:(v,u)\in E} f(v,u)}_{f_{in}(u)} = \underbrace{\sum_{v:(u,v)\in E} f(u,v)}_{f_{out}(u)}$$

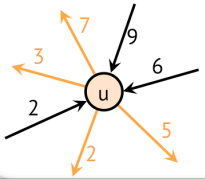


$f_{in}(u) = 2 + 6 + 9 = 17$

Flussnetzwerke

- Flusserhaltung: $\forall u \in V - \{q, s\}:$

$$\underbrace{\sum_{v:(v,u)\in E} f(v,u)}_{f_{in}(u)} = \underbrace{\sum_{v:(u,v)\in E} f(u,v)}_{f_{out}(u)}$$

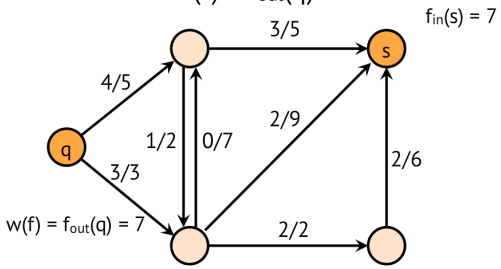


$f_{in}(u) = 2 + 6 + 9 = 17$

$f_{out}(u) = 2 + 5 + 7 + 3 = 17$

Flussnetzwerke

- Der Wert w(f) eines Flusses f ist definiert als $w(f) = f_{out}(q)$



Flussnetzwerke

- Problemstellung
  Gegeben sei ein Flussnetzwerk (G,c).
  Gesucht ist ein Fluss f auf (G,c) mit maximalem Wert w(f).

Vereinfachungen

- Erweitere $c:E \to \mathbb{N}_{>0}$ auf $c:V \times V \to \mathbb{N}_{\geq 0}$ durch
  $c(u,v) = 0$, falls $(u,v) \notin E$
- Analog: Erweitere f …
- Wir nehmen an, dass nicht gleichzeitig $(u,v) \in E$ und $(v,u) \in E$ gilt.
- G sei zusammenhängend, und damit ist $V-1 \leq E \leq V^2$

2.6.5 Maximaler Fluss

Restgraph

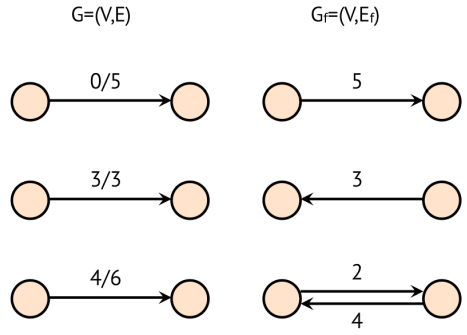
- Es sei G ein Flussnetzwerk und f ein Fluss auf G. Für alle Paare $(u,v) \in V^2$ definiere

$$\text{rest}_f(u,v) = \begin{cases} c(u,v) - f(u,v) & (u,v) \in E \\ f(v,u) & (v,u) \in E \\ 0 & \text{sonst} \end{cases}$$

- Das Restgraph $G_f = (V,E_f)$ ist dann definiert durch die Kantenmenge

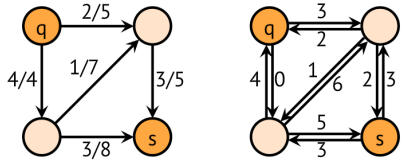
$$E_f = \{ (u,v) : \text{rest}_f(u,v) > 0 \}$$

Restgraph



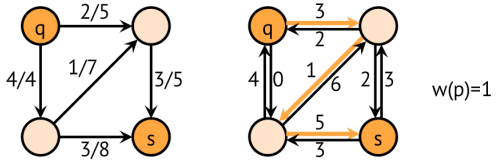
Restgraph

- Es sei G ein Flussnetzwerk und f ein Fluss auf G. Ein flussvergrößernder Weg $p: q=v_0,v_1,\ldots,v_k=s$ ist ein einfacher Weg von q nach s im Restgraph G_f. Der Wert von p ist $w(p) = \min_i w(v_i,v_{i+1})$.



Restgraph

- Es sei G ein Flussnetzwerk und f ein Fluss auf G. Ein flussvergrößernder Weg p: $q=v_0,v_1,\ldots,v_k=s$ ist ein einfacher Weg von q nach s im Restgraph G_f. Der Wert von p ist $w(p) = \min_i w(v_i,v_{i+1})$.

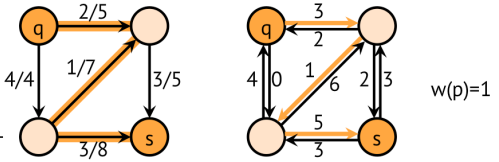


w(p)=1

Restgraph

- Es sei G ein Flussnetzwerk und f ein Fluss auf G. Ein flussvergrößernder Weg p: $q=v_0,v_1,\ldots,v_k=s$ ist ein einfacher Weg von q nach s im Restgraph G_f. Der Wert von p ist $w(p) = \min_i w(v_i,v_{i+1})$.

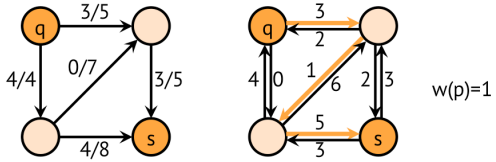
Achtung: "Rückwärts-kante"



w(p)=1

Restgraph

- Es sei G ein Flussnetzwerk und f ein Fluss auf G. Ein flussvergrößernder Weg p: $q=v_0,v_1,\ldots,v_k=s$ ist ein einfacher Weg von q nach s im Restgraph G_f. Der Wert von p ist $w(p) = \min_i w(v_i,v_{i+1})$.



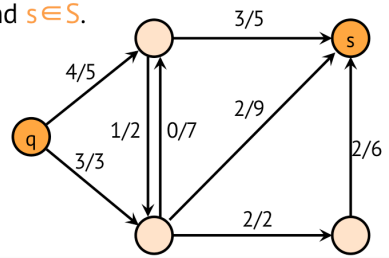
w(p)=1

Ford-Fulkerson-Methode

- FordFulkersonMethod(V,E,c)
  for all e∈E do
    f[e] ← 0
  while exists fv-path p do
    increase f along p by w(p)
  return f
- Zum Beweis der Korrektheit dieser Methode machen wir einen kleinen Exkurs...

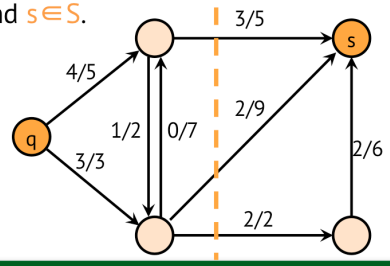
Schnitte

- Es sei G ein Flussnetzwerk. Ein Schnitt (Q,S) in G ist eine Partitionierung der Knotenmenge $V=Q \cup S, Q \cap S=\emptyset$ mit $q \in Q$ und $s \in S$.



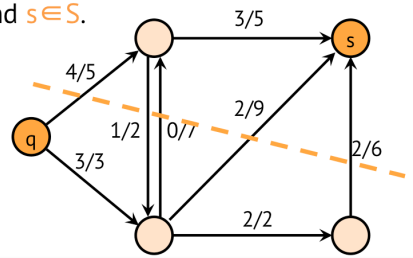
Schnitte

- Es sei G ein Flussnetzwerk. Ein Schnitt (Q,S) in G ist eine Partitionierung der Knotenmenge $V=Q \cup S, Q \cap S=\emptyset$ mit $q \in Q$ und $s \in S$.



Schnitte

- Es sei G ein Flussnetzwerk. Ein Schnitt (Q,S) in G ist eine Partitionierung der Knotenmenge $V=Q \cup S, Q \cap S=\emptyset$ mit $q \in Q$ und $s \in S$.



Schnitte

- Die Kapazität des Schnitts (Q,S) ist definiert als

$$c(Q,S) = \sum_{u \in Q, v \in S} c(u,v)$$

- Der Fluss über den Schnitt (Q,S) ist definiert als

$$f(Q,S) = \sum_{u \in Q, v \in S} f(u,v) - \sum_{u \in Q, v \in S} f(v,u)$$

Schnitte

- Lemma (Max-Flow ⩽ Min-Cut)
  Für jeden Fluss f und jeden Schnitt (Q,S) gilt:

  $$w(f) = f(Q,S) \leqslant c(Q,S)$$

  Insbesondere: $f_{out}(q) = f_{in}(s)$
- Beweis: $f(Q,S) \leqslant c(Q,S)$ klar

Argumente für Beweis: c wird durch f ersetzt und f muss <= c sein, ansonsten wird nur zusätzlich etwas abgezogen, was das Ergebnis höchstens noch kleiner macht

Schnitte

- Zeige durch Induktion über #Q:
  w(f) = f(Q,S)
- IA: falls #Q=1, so ist Q={q}, S=V−{q} und damit $w(f)=f_{out}(q)=f(Q,S)$
- IV: Es sei $V=Q \cup \{x\} \cup S$,
  mit $Q \cap S=Q \cap \{x\}=S \cap \{x\}=\varnothing$
  und es gelte $w(f)=f(Q,S \cup \{x\})$

Induktionsstruktur: Fange mit dem Schnitt an, der in Q nur die Quelle enthält. Der Induktionsschritt schiebt jeweils einen beliebigen Knoten aus der Menge S in die Menge Q.

Schnitte

$$
\begin{aligned}
f(Q \cup x, S) &= \sum_{u \in Q \cup x, v \in S} f(u,v) - \sum_{u \in Q \cup x, v \in S} f(v,u) \\
&= \sum_{u \in Q, v \in S} f(u,v) - \sum_{u \in Q, v \in S} f(v,u) \\
&\quad + \sum_{v \in S} f(x,v) - \sum_{v \in S} f(v,x) \\
&\quad \boxed{\begin{aligned} &+ \sum_{u \in Q} f(x,u) - \sum_{u \in Q} f(u,x) \\ &- \sum_{u \in Q} f(x,u) + \sum_{u \in Q} f(u,x) \end{aligned}} \quad =0 \\
&= \sum_{u \in Q, v \in S \cup x} f(u,v) - \sum_{u \in Q, v \in S \cup x} f(v,u) = f(Q, S \cup x)
\end{aligned}
$$

Schnitte

$$
\begin{aligned}
f(Q \cup x, S) &= \sum_{u \in Q \cup x, v \in S} f(u,v) - \sum_{u \in Q \cup x, v \in S} f(v,u) \\
&= \sum_{u \in Q, v \in S} f(u,v) - \sum_{u \in Q, v \in S} f(v,u) \\
&\quad \boxed{+ \sum_{v \in S} f(x,v)} - \boxed{\sum_{v \in S} f(v,x)} \quad = f_{in}(x) \\
&\quad \text{=} f_{out}(x) \quad \boxed{+ \sum_{u \in Q} f(x,u)} - \boxed{\sum_{u \in Q} f(u,x)} \\
&\quad - \sum_{u \in Q} f(x,u) + \sum_{u \in Q} f(u,x) \\
&= \sum_{u \in Q, v \in S \cup x} f(u,v) - \sum_{u \in Q, v \in S \cup x} f(v,u) = f(Q, S \cup x)
\end{aligned}
$$

Schnitte

$$
\begin{aligned}
f(Q \cup x, S) &= \sum_{u \in Q \cup x, v \in S} f(u,v) - \sum_{u \in Q \cup x, v \in S} f(v,u) \\
&= \sum_{u \in Q, v \in S} f(u,v) - \sum_{u \in Q, v \in S} f(v,u) \\
&\quad \boxed{\begin{aligned} &+ \sum_{v \in S} f(x,v) - \sum_{v \in S} f(v,x) \\ &+ \sum_{u \in Q} f(x,u) - \sum_{u \in Q} f(u,x) \end{aligned}} \quad =0 \\
&\quad - \sum_{u \in Q} f(x,u) + \sum_{u \in Q} f(u,x) \\
&= \sum_{u \in Q, v \in S \cup x} f(u,v) - \sum_{u \in Q, v \in S \cup x} f(v,u) = f(Q, S \cup x)
\end{aligned}
$$

Hier wird gezeigt: Der Fluss vor der Verschiebung von x von S zu Q ist gleich dem Fluss nach der Verschiebung.

Schnitte

- Satz (Max-Flow = Min-Cut)
  Die folgenden Aussagen sind äquivalent:
  a) f ist ein maximaler Fluss
  b) G_f enthält keinen fv-Weg
  c) Es gibt einen Schnitt (Q,S) mit w(f) = c(Q,S)

Schnitte

- Satz (Max-Flow = Min-Cut)
  Die folgenden Aussagen sind äquivalent:
  a) f ist ein maximaler Fluss
  b) G_f enthält keinen fv-Weg
  c) Es gibt einen Schnitt (Q,S) mit w(f) = c(Q,S)
- Beweis: a → b → c → a klar!

Schnitte

- Satz (Max-Flow = Min-Cut)
  Die folgenden Aussagen sind äquivalent:
  a) f ist ein maximaler Fluss
  b) G_f enthält keinen fv-Weg
  c) Es gibt einen Schnitt (Q,S) mit w(f) = c(Q,S)
- Beweis: a → b → c → a
  – w(f) = c(Q,S) ≥ Min-Cut ≥ Max-Flow
  Lemma

Schnitte

- Satz (Max-Flow = Min-Cut)
  Die folgenden Aussagen sind äquivalent:
  a) f ist ein maximaler Fluss
  b) G_f enthält keinen fv-Weg
  c) Es gibt einen Schnitt (Q,S) mit w(f) = c(Q,S)
- Beweis: a ⟶ b ⟶ c ⟶ a
  – Finde einen Schnitt …

Schnitte

- Definiere
  $Q = \{ v \in V : \exists$ Weg von q nach v in $G_f \}$
  und $S = V - Q$
- Der Schnitt (Q,S) ist wohldefiniert: $q \in Q, s \in S$
- Für jedes Paar $(u,v) \in Q \times S$ gilt $f(u,v)=c(u,v)$, da sonst $rest_f(u,v)>0$ und $v \in Q$.
- Für jedes Paar $(v,u) \in S \times Q$ gilt $f(v,u)=0$, da sonst $rest_f(u,v)>0$ und $v \in Q$.
- Damit …

Schnitte

$$c(Q,S) = \sum_{u \in Q, v \in S} c(u,v)$$

$$= \sum_{u \in Q, v \in S} f(u,v) - \sum_{u \in Q, v \in S} f(v,u)$$

$$= f(Q,S)$$

$$= w(f)$$

qed

Ford-Fulkerson-Methode

- Lemma
  Falls die Ford-Fulkerson-Methode terminiert, so liefert sie einen maximalen Fluss.
- Beweis: Terminierung ⟶ kein fv-Weg in G_f ⟶ f maximal

2. Gleichung wahr wegen Argumenten auf voriger Folie

Ford-Fulkerson-Methode

- Da der Fluss in jeder Iteration um mindestens eine Einheit erhöht wird, terminiert die Ford-Folkerson-Methode.

Laufzeit

- Satz (Edmonds, Karp)
  Die Laufzeit der Ford-Fulkerson-Methode mit Breitensuche zur Bestimmung des fv-Weges ist

$$O(V \times E^2) = O(V^5)$$

- Beweis: Zeige, dass der Algorithmus nach O(V×E) Iterationen terminiert.

Laufzeit

- Lemma
  Von Iteration zu Iteration verringert sich die Distanz von q zu einem Knoten v im Restgraph nicht.
- Beweis des Lemmas: Entlang des fv-Weges können Kanten verschwinden oder entstehen…

Laufzeit

- Sei (u,v) eine Kante auf einem fv-Weg p, d.h. $rest_f(v,u)$ wird erhöht.
- Die Kante (v,u) kann entstehen, falls $rest_f(v,u)$ zuvor = 0 war.
- Breitensuche → p ist ein kürzester Weg → dist(q,v) = dist(q,u) + 1
- Es erhöht sich $rest_f(v,u)$ und (v,u) wird evtl. eingefügt. Da dist(q,v) > dist(q,u) verringern sich die Abstände der Knoten zu q in diesem Fall natürlich nicht.

Wichtig: es werden bei der Breitensuche immer kürzeste Pfade bzgl. der Knotenanzahl gewählt (ohne diese Eigenschaft stimmt die Komplexitätsabschätzung nicht)!

Laufzeit

- Sei (u,v) eine Kante auf einem fv-Weg p, d.h. $rest_f(u,v)$ wird verringert.
- Die Kante (u,v) kann verschwinden, falls $rest_f(u,v)$ zuvor >0 war.
- Kantenlöschungen können aber Distanzen offensichtlich nicht verringern.

Laufzeit

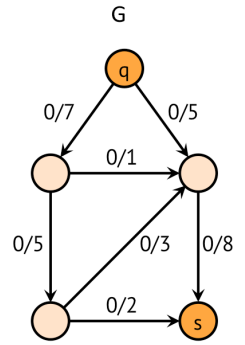
- Beweis des Satzes von Edmond-Karp
  - Wenn (u,v) zu Zeitpunkt 1 verschwindet, so liegt (u,v) auf einem kürzesten Pfad und es gilt: $dist_1(q,v) = dist_1(q,u) + 1$
  - Wenn (u,v) zu einem späteren Zeitpunkt 2 wieder eingefügt wird, so liegt (v,u) auf einem kürzesten Pfad, d.h. $dist_2(q,u) = dist_2(q,v) + 1$
  - Zwischen den Iterationen haben sich die Abstände nach Lemma nicht verringert, also $dist_2(q,u) > dist_2(q,v) \geqslant dist_1(q,v) > dist_1(q,u)$ und damit $dist_2(q,u) \geqslant dist_1(q,u) + 2$

Laufzeit

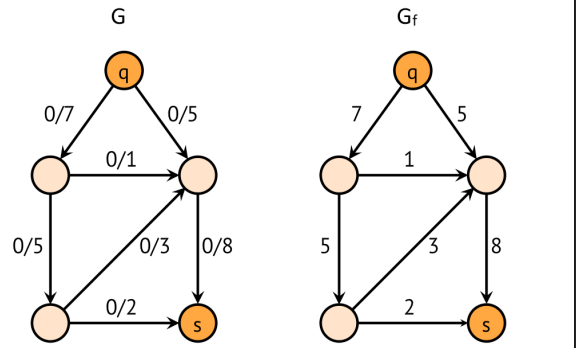
- Beweis des Satzes von Edmond-Karp
  - Da die maximale Distanz $V-1$ ist, kann eine Kante höchstens $V/2$ mal entfernt und wieder eingefügt werden.
  - In jeder Iteration wird mindestens eine Kante entfernt.
  - Es gibt $2 \times E$ Kanten im Restgraph, also ist die Zahl der Iterationen höchstens $1/2 \times V \times 2 \times E = V \times E$

qed

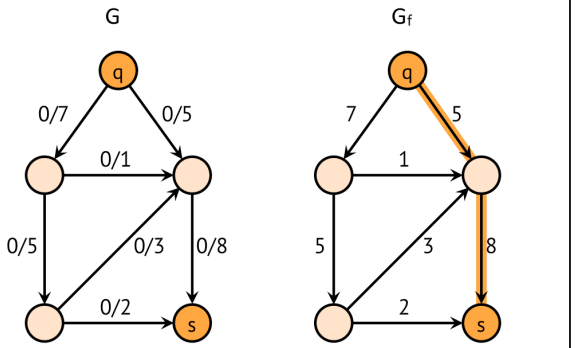
Beispiel



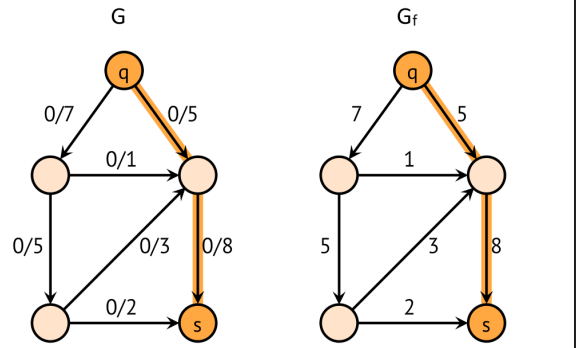
Beispiel



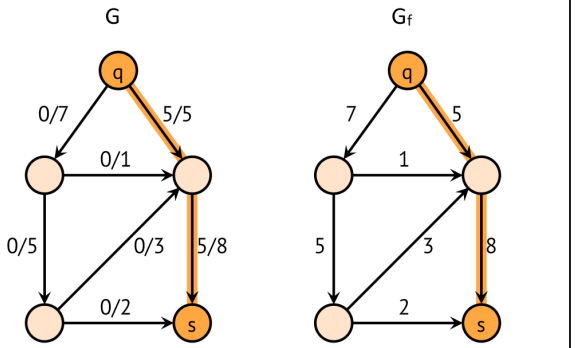
Beispiel



Beispiel

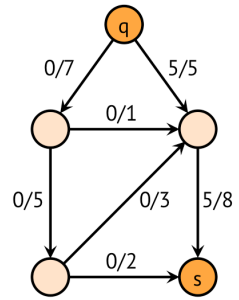


Beispiel



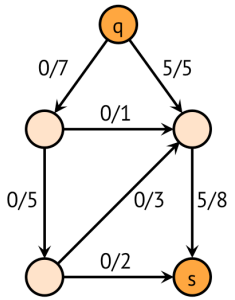
Beispiel

G

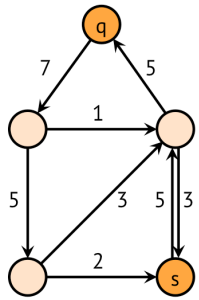


Beispiel

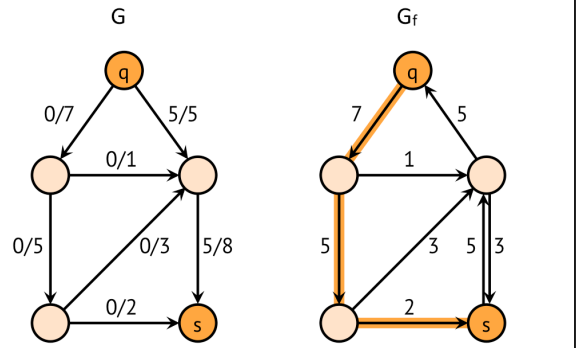
G



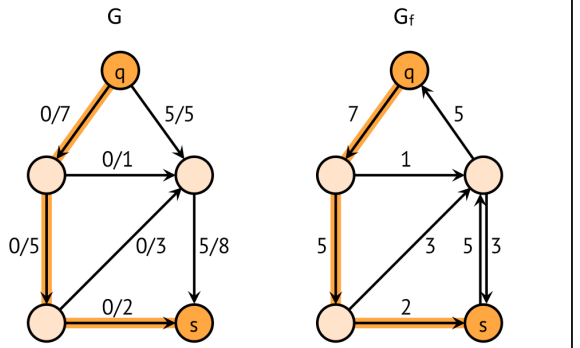
G_f



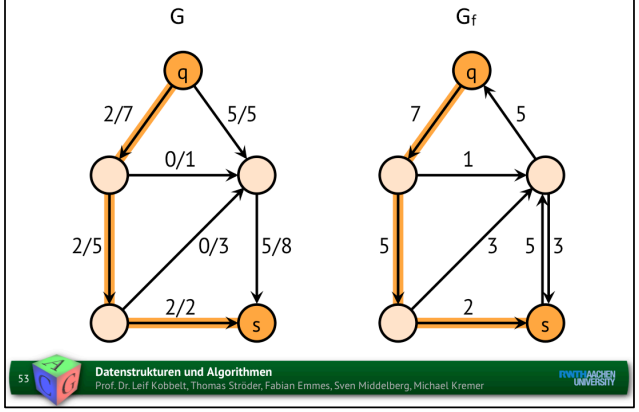
Beispiel



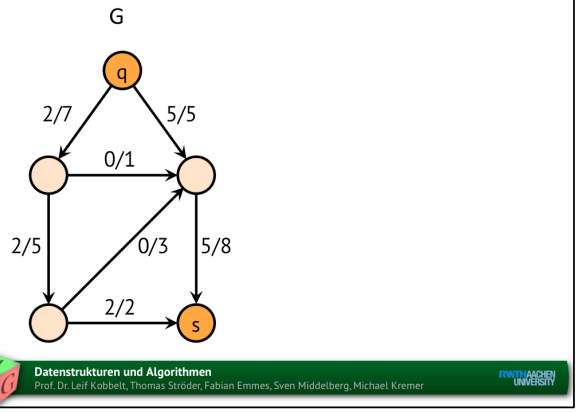
Beispiel



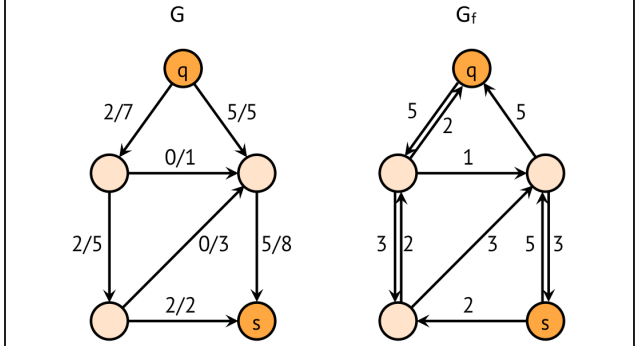
Beispiel



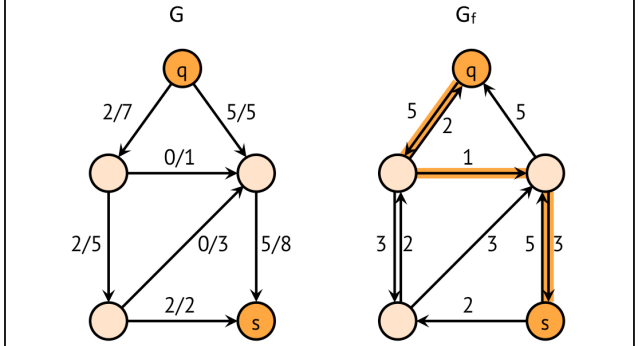
Beispiel



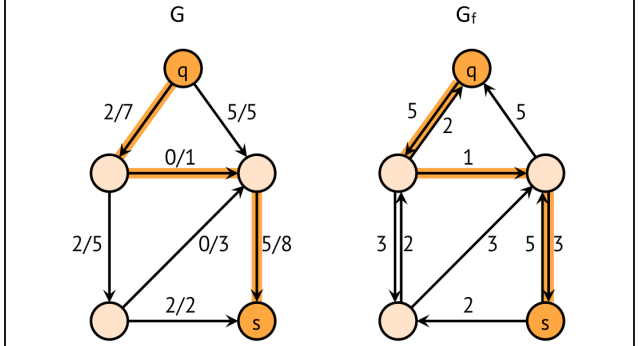
Beispiel



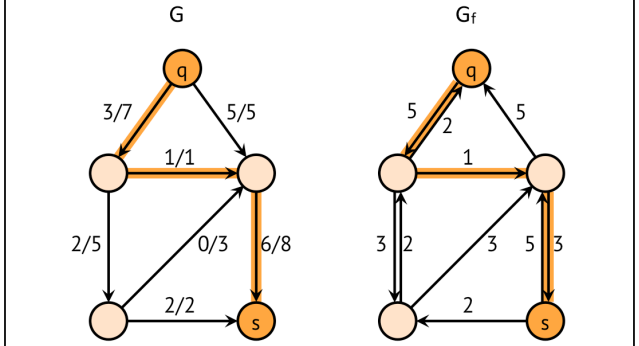
Beispiel



Beispiel

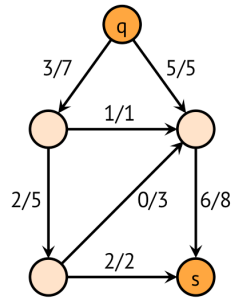


Beispiel



Beispiel

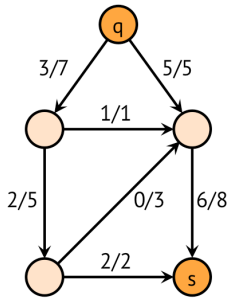
G



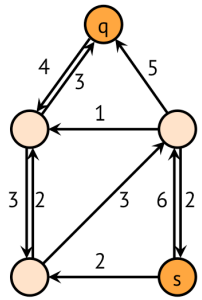
G_f

Beispiel

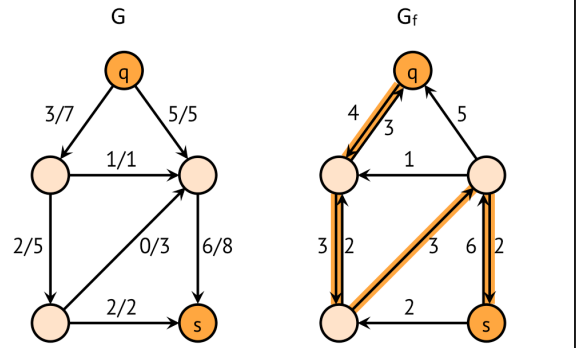
G



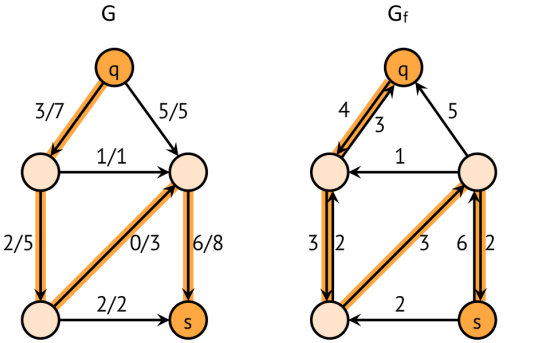
G_f



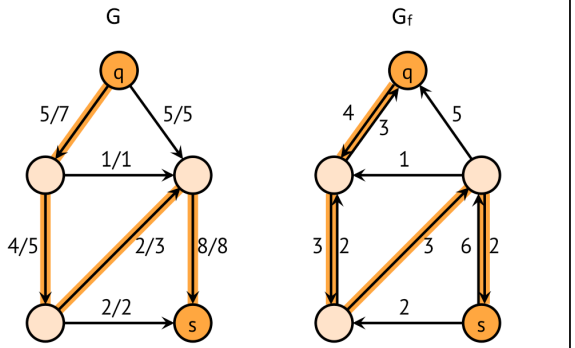
Beispiel



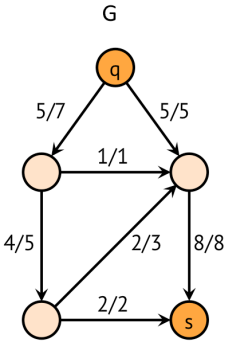
Beispiel



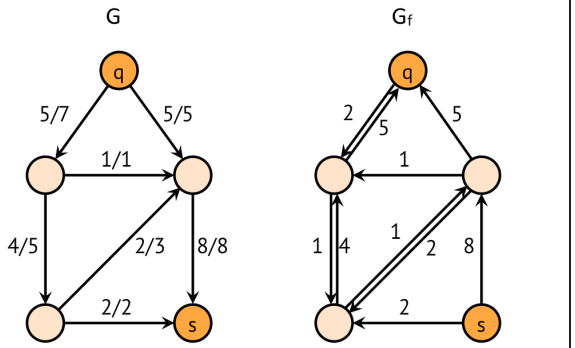
Beispiel



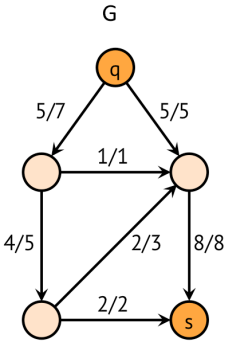
Beispiel



Beispiel



Beispiel

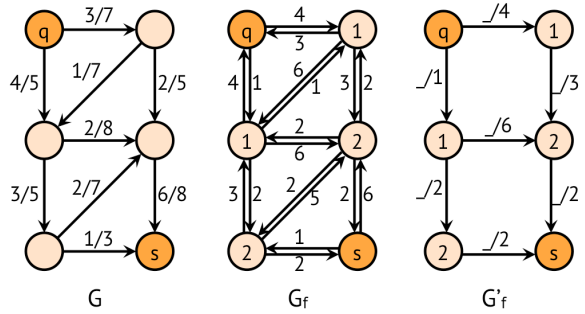


2.6.5 Maximaler Fluss

Niveaunetzwerk

- Für $i \in N$ sei
  $V_i = \{ v \in V : \text{dist}(q,v) \text{ in } G_f \text{ ist } i \}$
- Das Niveaunetzwerk $G'_f = (V, E'_f, \text{rest}_f)$ ist definiert durch
  $E'_f = \{ (u,v) \in E_f : \exists\, i \text{ mit } u \in V_i \text{ und } v \in V_{i+1} \}$
- Das Niveaunetzwerk G'_f kann aus dem Restgraph G_f durch Breitensuche in Zeit $O(E)$ berechnet werden.

Niveaunetzwerk



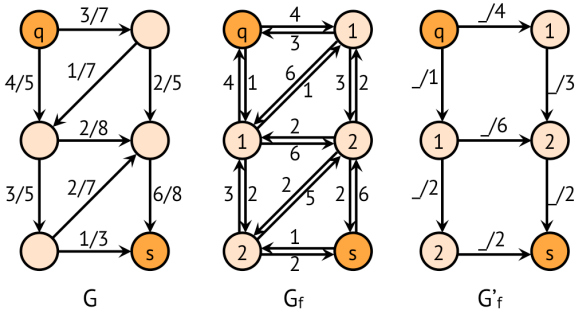
Sperrfluss

- Es sei g ein Fluss in G'_f
- Eine Kante $e \in E'_f$ heißt saturiert, falls $g(e) = rest_f(e)$.
- g heißt Sperrfluss, wenn jeder Weg von q nach s in G'_f mindestens eine saturierte Kante enthält.

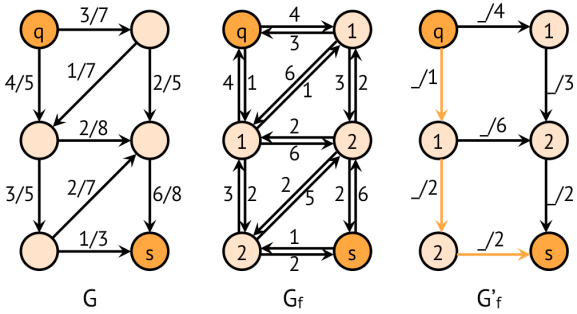
Sperrfluss

- Sperrfluss()
  - for all e in E'_f do
    - g(e) ← 0
  - while exists q-s-path p in G'_f do
    - increase g along p
    - remove all saturated edges from G'_f
  - return g

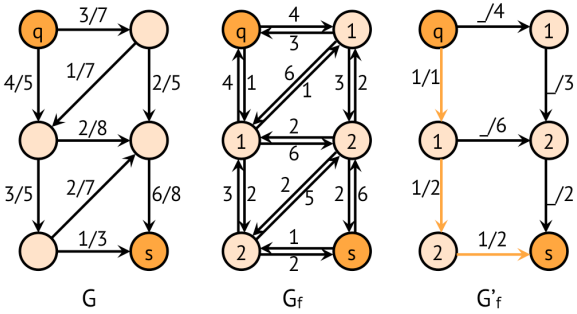
Sperrfluss



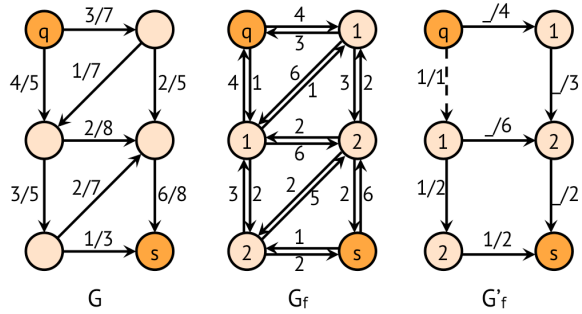
Sperrfluss



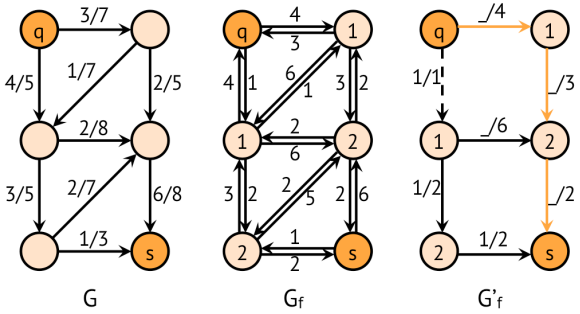
Sperrfluss



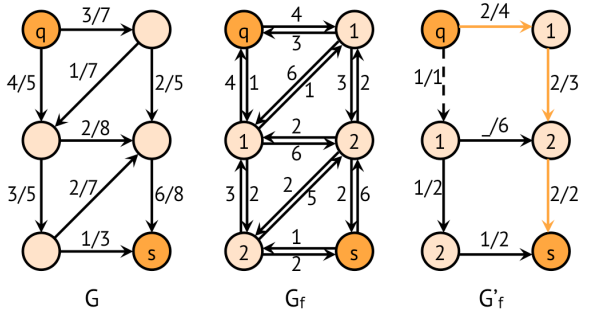
Sperrfluss



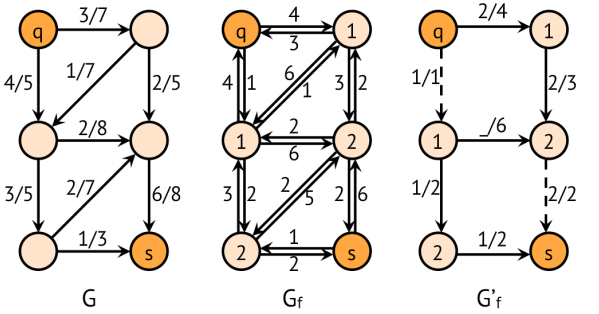
Sperrfluss



Sperrfluss



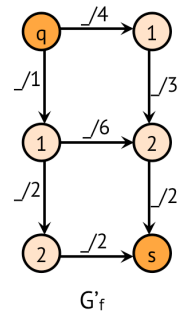
Sperrfluss



Sperrfluss

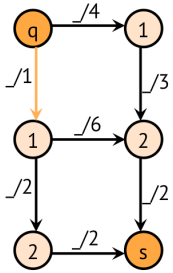
- Lemma
  Bei geeigneter Implementierung hat der Algorithmus zur Sperrflussberechnung eine Laufzeit von $O(V \times E)$.
- Idee: Tiefensuche mit Backtracking und Löschen von Sackgassen

Sperrfluss



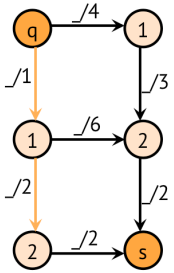
G'_f

Sperrfluss



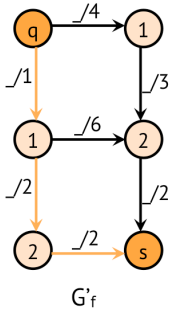
G'_f

Sperrfluss



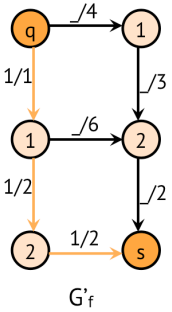
G'_f

Sperrfluss



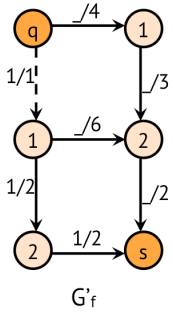
G'_f

Sperrfluss



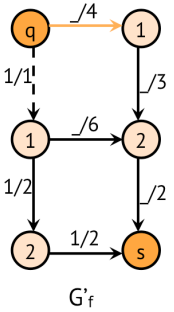
G'_f

Sperrfluss



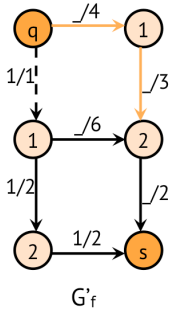
G'_f

Sperrfluss



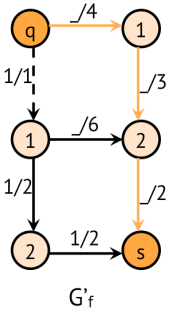
G'_f

Sperrfluss



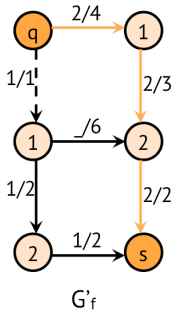
G'_f

Sperrfluss



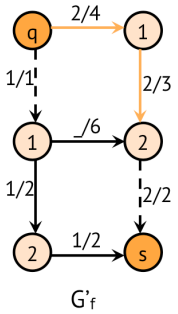
G'_f

Sperrfluss



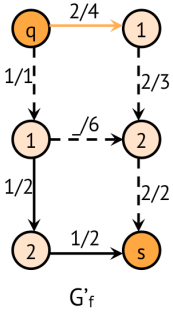
G'_f

Sperrfluss



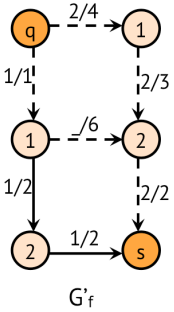
G'_f

Sperrfluss



G'_f

Sperrfluss



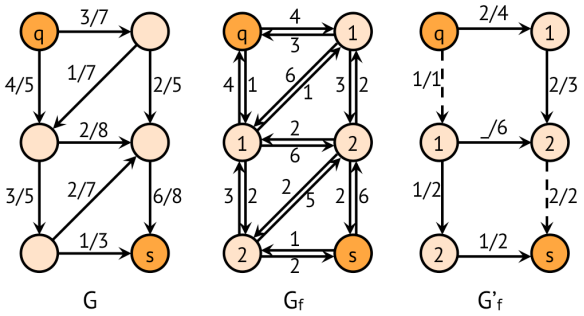
G'_f

Dinics Algorithmus

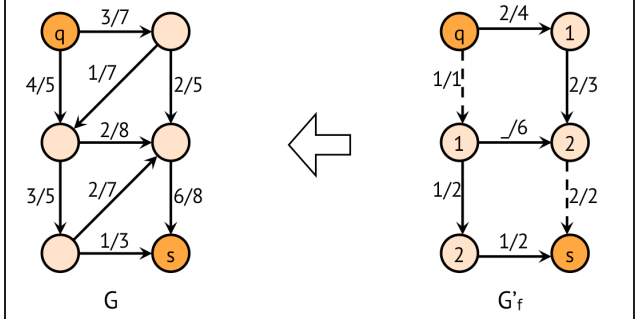
- Dinic()
  - for all e in E do
    - f(e) ← 0
  - while exists q-s-path p in G_f do
    - berechne Niveaunetzwerk G'_f
    - berechne Sperrfluss g in G'_f
    - addiere g zu f
  - return f

Beachte Rückwärtskanten!

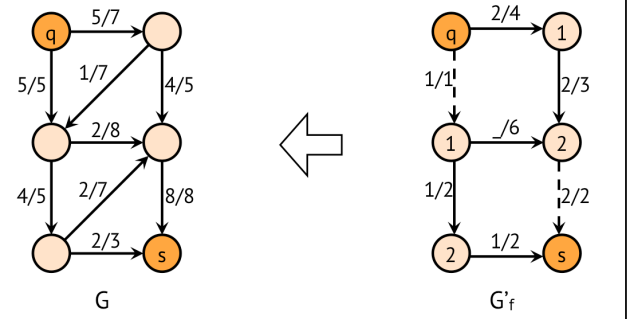
Dinics Algorithmus



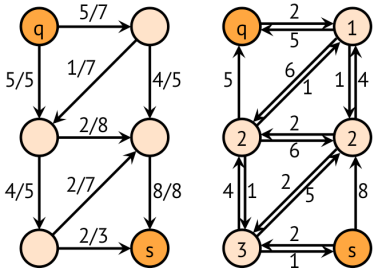
Dinics Algorithmus



Dinics Algorithmus



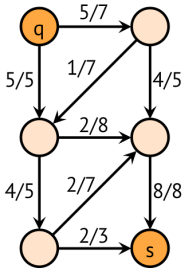
Dinics Algorithmus



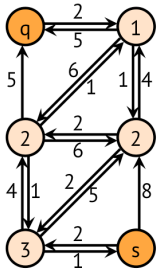
G

G_f

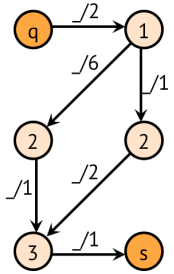
Dinics Algorithmus



G

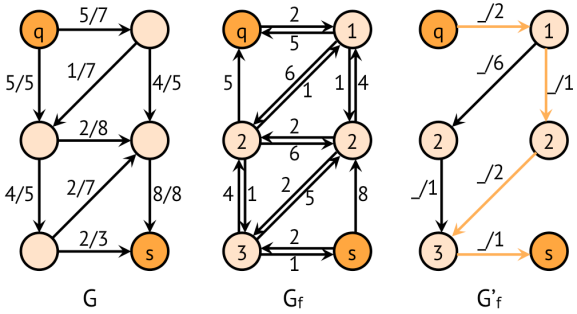


G_f

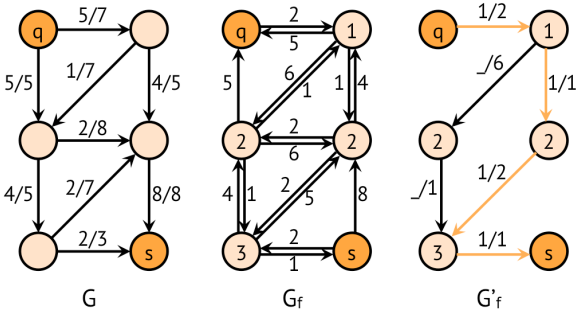


G'_f

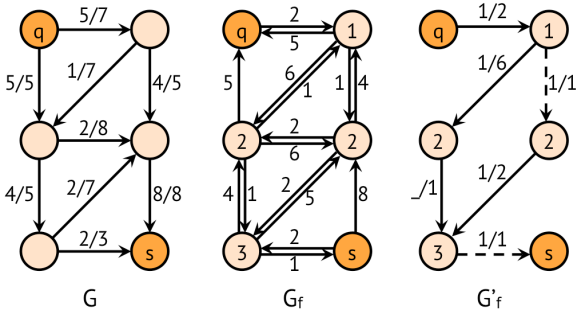
Dinics Algorithmus



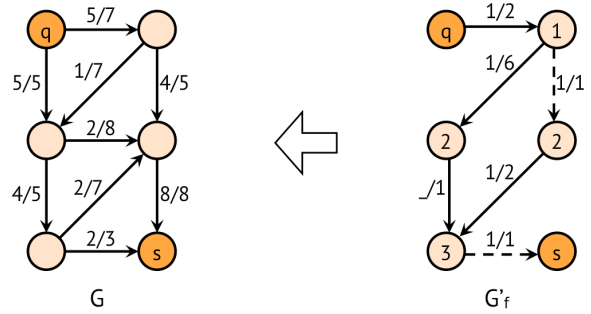
Dinics Algorithmus



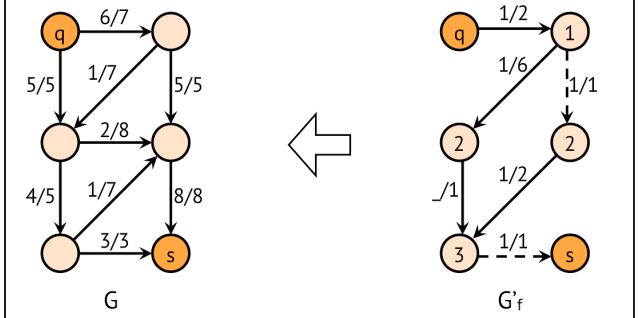
Dinics Algorithmus



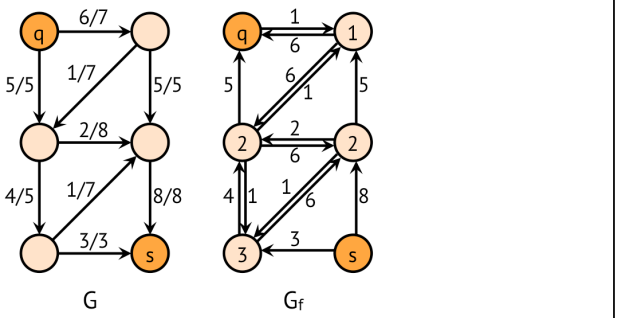
Dinics Algorithmus



Dinics Algorithmus



Dinics Algorithmus



Dinics Algorithmus

- Beobachtung:
  - Niveau-Graph enthält immer genau alle kürzesten Pfade
  - Länge kürzester Pfade erhöht sich um mindestens 1 nach Sperrfluss
  - Maximale Länge: |V|

→ Laufzeit: O(V) × O(Sperrfluss)

Dinics Algorithmus

- Beobachtung:
  - Niveau-Graph enthält immer genau alle kürzesten Pfade
  - Länge kürzester Pfade erhöht sich um mindestens 1 nach Sperrfluss
  - Maximale Länge: |V|

→ Laufzeit: O(V) × O(V×E)

Dinics Algorithmus

- Satz
  Die Laufzeit von Dinics Algorithmus ist $O(V^2 \times E) = O(V^4)$.

2.6.5 Maximaler Fluss

Forward-backward propagation

- Lemma
  Mit forward-backward propagation lässt sich ein Sperrfluss sogar in Zeit $O(V^2)$ berechnen. Der entstehende Algorithmus hat nur noch einen Aufwand von $O(V^3)$.

2.6.5 Maximaler Fluss

- Zusammenfassung

Ford-Fulkerson/Edmonds-Karp	**$O(V^5)$**
Dinic	**$O(V^4)$**
Forward-Backward Propagation	**$O(V^3)$**

Schnitt-Trick

- Falls nur Wert des maximalen Flusses berechnet werden soll
- und das Netzwerk nur wenige Knoten enthält:
  - Berechne direkt minimalen Schnitt
  - Max-Flow = Min-Cut => Wert des maximalen Flusses

Schnitt-Trick

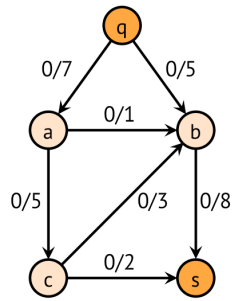
- Anzahl Schnitte: $2^{|V|-2}$
  - ➢ exponentielle Laufzeit
- Aber: keine Residualnetzwerke etc.

➢ Auf Papier oft schneller

- Viele Schnitte können ignoriert werden
  - Jeder Knoten in Q muss von q aus erreichbar sein, ohne Knoten aus S zu besuchen
  - s muss von jedem Knoten in S aus erreichbar sein, ohne Knoten aus Q zu besuchen

Beispiel

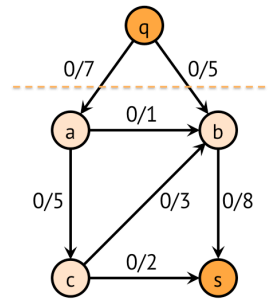
G



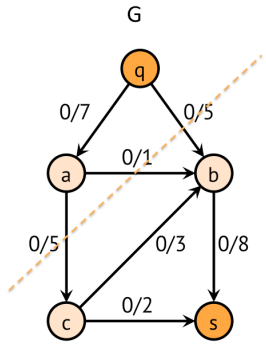
Beispiel

G

q / a,b,c,s: 12

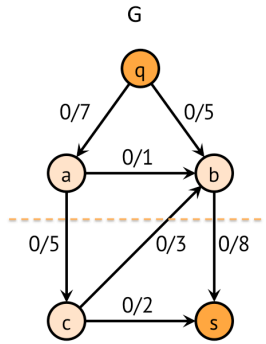


Beispiel



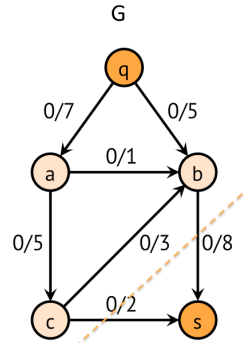
q / a,b,c,s: 12
q,a / b,c,s: 11

Beispiel



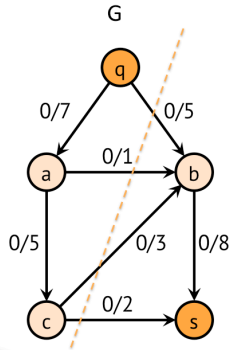
q / a,b,c,s: 12
q,a / b,c,s: 11
q,a,b / c,s: 13

Beispiel



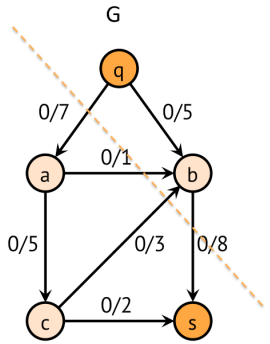
q / a,b,c,s: 12
q,a / b,c,s: 11
q,a,b / c,s: 13
q,a,b,c / s: 10

Beispiel



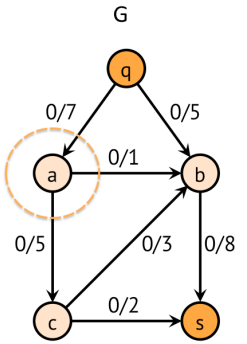
q / a,b,c,s: 12
q,a / b,c,s: 11
q,a,b / c,s: 13
q,a,b,c / s: 10
q,a,c / b,s: 11

Beispiel



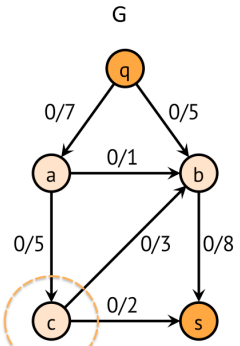
q / a,b,c,s: 12
q,a / b,c,s: 11
q,a,b / c,s: 13
q,a,b,c / s: 10
q,a,c / b,s: 11
q,b / a,c,s: 15

Beispiel



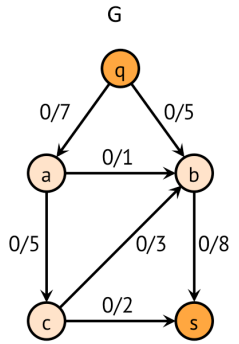
q / a,b,c,s: 12
q,a / b,c,s: 11
q,a,b / c,s: 13
q,a,b,c / s: 10
q,a,c / b,s: 11
q,b / a,c,s: 15
q,b,c / a,s: ignorieren

Beispiel



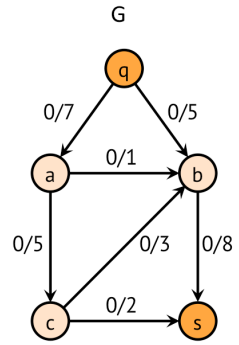
q / a,b,c,s: 12
q,a / b,c,s: 11
q,a,b / c,s: 13
q,a,b,c / s: 10
q,a,c / b,s: 11
q,b / a,c,s: 15
q,b,c / a,s: ignorieren
q,c / a,b,s: ignorieren

Beispiel



q / a,b,c,s: 12
q,a / b,c,s: 11
q,a,b / c,s: 13
q,a,b,c / s: 10
q,a,c / b,s: 11
q,b / a,c,s: 15
q,b,c / a,s: ignorieren
q,c / a,b,s: ignorieren

Beispiel

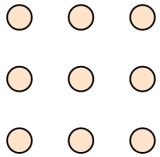


q / a,b,c,s: 12
q,a / b,c,s: 11
q,a,b / c,s: 13
q,a,b,c / s: 10
q,a,c / b,s: 11
q,b / a,c,s: 15
q,b,c / a,s: ignorieren
q,c / a,b,s: ignorieren

Antwort: 10

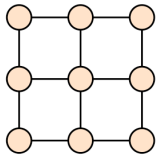
Beispiel

- Bildverarbeitung: Intelligent Scissors, Montage



Beispiel

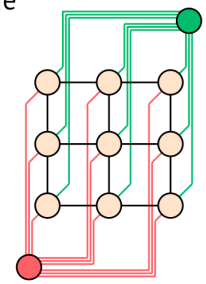
- Bildverarbeitung: Intelligent Scissors, Montage



t-links bestrafen Sprünge in den Farbwerten

Beispiel

- Bildverarbeitung: Intelligent Scissors, Montage



t-links bestrafen Sprünge in den Farbwerten

n-links bestimmen die Komponenten