Proseminar
Effiziente Algorithmen

Kapitel 8: Graphalgorithmen

Prof. Dr. Christian Scheideler

WS 2016

Übersicht

- Kürzeste Wege
- Minimale Spannbäume
- Matching
- Flussprobleme

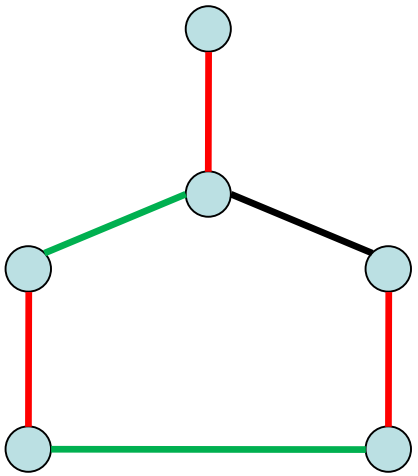
} DuA

Übersicht

- Kürzeste Wege
- Minimale Spannbäume
- Matching
- Flussprobleme

Grundlagen

Definition 8.1: Sei G=(V,E) ein ungerichteter Graph. Ein Matching M in G ist eine Teilmenge von E, so dass keine zwei Kanten aus M einen Endpunkt gemeinsam haben.



Matching:

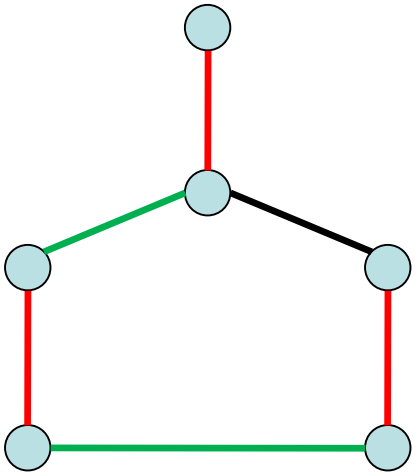
— Variante 1

— Variante 2

Grundlagen

Definition 8.2:

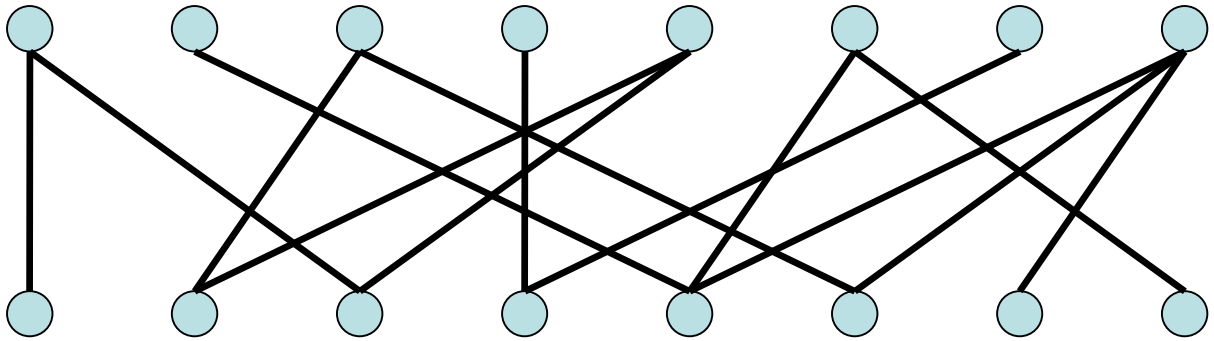
- Ein Matching M in $G=(V,E)$ heißt perfekt, falls $|M|=|V|/2$.
- Ein Matching M heißt Matching maximaler Kardinalität (engl. maximum matching) in G, falls es in G kein Matching M' mit $|M'|>|M|$ gibt (rot im Beispiel)



- Ein Matching M heißt maximal in G, falls es bezüglich „$\subseteq$“ maximal ist (engl. maximal matching, grün im Beispiel)

Grundlagen

Definition 8.3: Sei $G=(V,E)$ ein ungerichteter Graph. Wenn V in zwei nichtleere Teilmengen V_1 und V_2 partitioniert werden kann (d.h. $V_1 \cup V_2 = V$ und $V_1 \cap V_2 = \emptyset$), so dass $E \subseteq V_1 \times V_2$ ist, dann heißt G bipartit (ausgedrückt durch $G=(V_1,V_2,E)$).

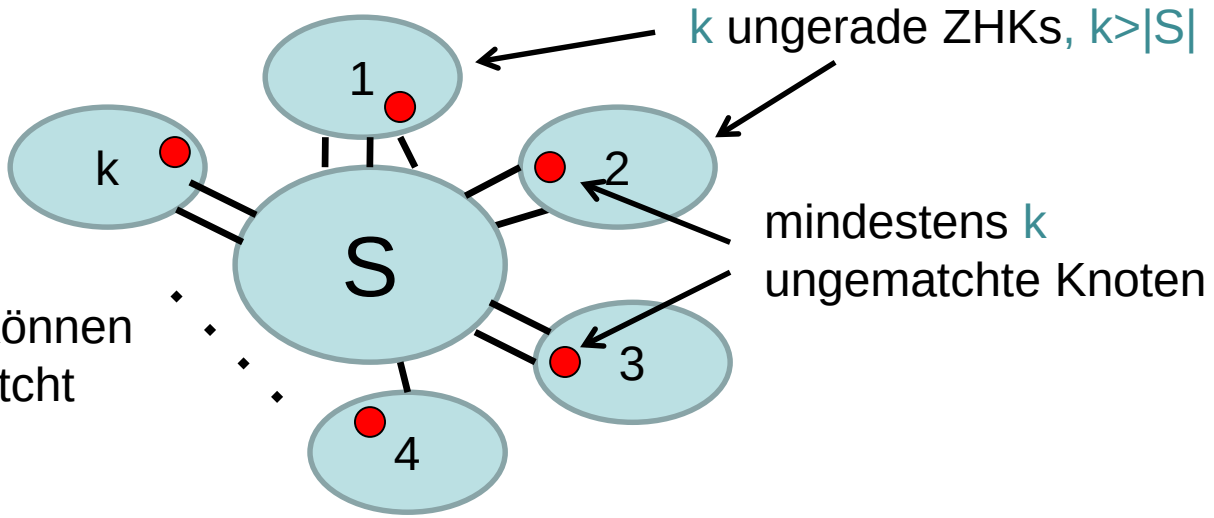


Grundlagen

Satz 8.4: Ein Graph G=(V,E) hat ein perfektes Matching genau dann, wenn |V| gerade ist und es kein S⊆V gibt, so dass der durch V\S induzierte Teilgraph mehr als |S| Zusammenhangskomponenten ungerader Größe enthält.

Beweis:

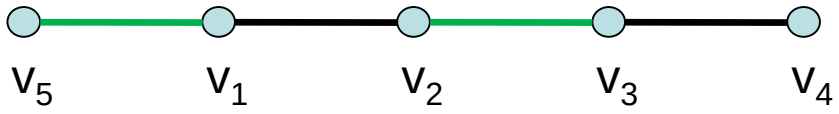
„⇒“:



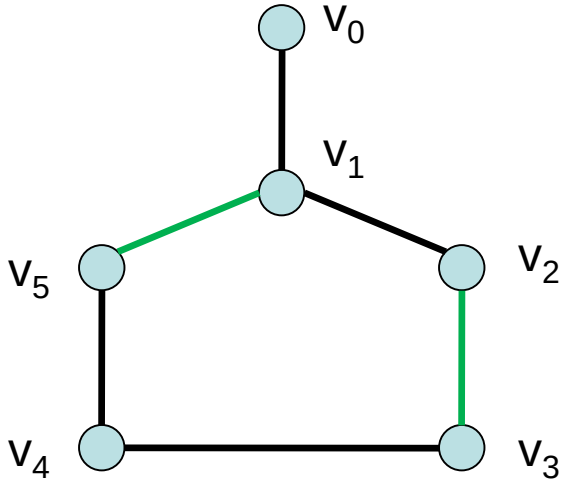
Grundlagen

Definition 8.5: Ein einfacher Pfad (Kreis) $v_0,v_1,\ldots,v_k$ heißt alternierend bzgl. eines Matchings M, falls die Kanten $\{v_i,v_{j+1}\}$ abwechselnd in M und nicht in M liegen.

gerade Länge:



ungerade Länge:

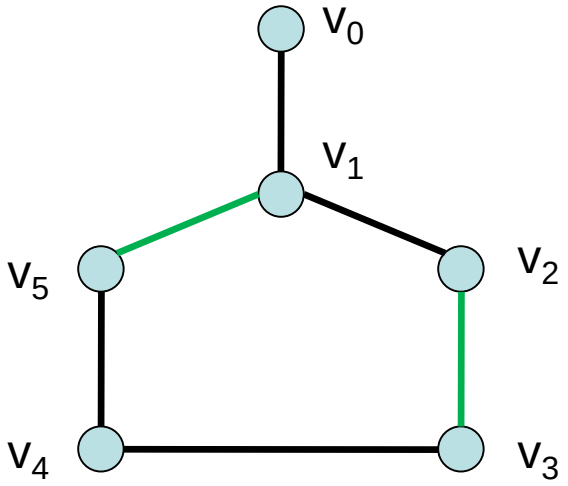


Grundlagen

Definition 8.6: Ein alternierender Pfad bzgl. eines Matchings M heißt augmentierend, falls er an beiden Enden ungematchte Knoten hat und kein Kreis ist.

nicht augmentierend (v_1 gematcht):

augmentierend:



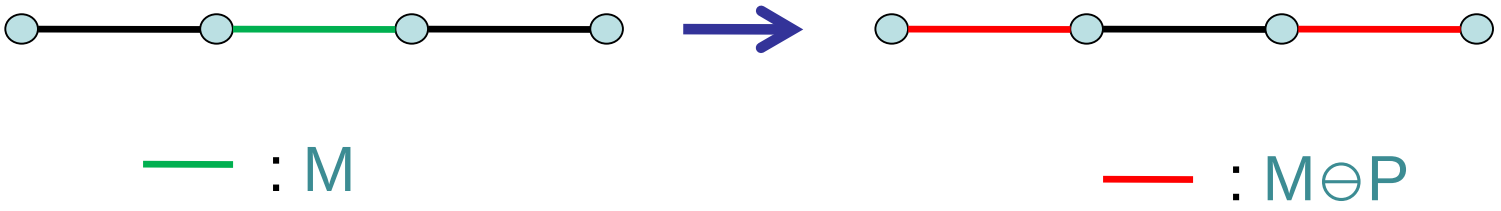
Grundlagen

Definition 8.7: Seien S und T zwei Mengen, dann bezeichne $S \ominus T$ die symmetrische Differenz von S und T, d.h. $S \ominus T = (S \setminus T) \cup (T \setminus S)$.

Lemma 8.8: Sei M ein Matching und P ein augmentierender Pfad bzgl. M. Dann ist auch $M \ominus P$ ein Matching, und es gilt $|M \ominus P| = |M|+1$.

Beweis:

Veränderung bzgl. Pfad P:



Grundlagen

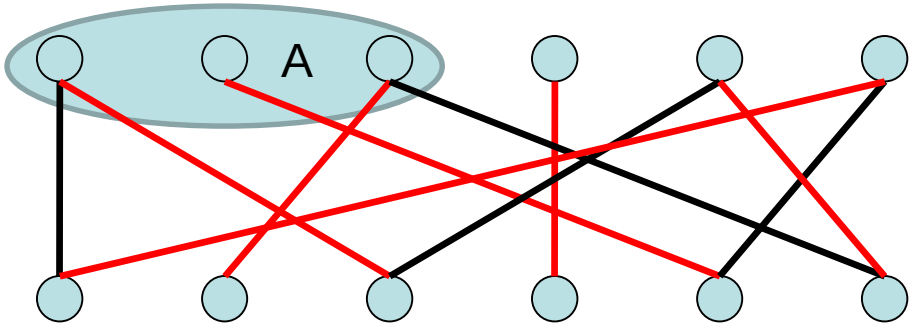
Satz 8.9: Heiratssatz

Sei $G=(U,V,E)$ ein bipartiter Graph. G enthält ein Matching der Kardinalität $|U|$ genau dann, wenn gilt:

$$\forall A \subseteq U: |N(A)| \geq |A|$$

Beweis:

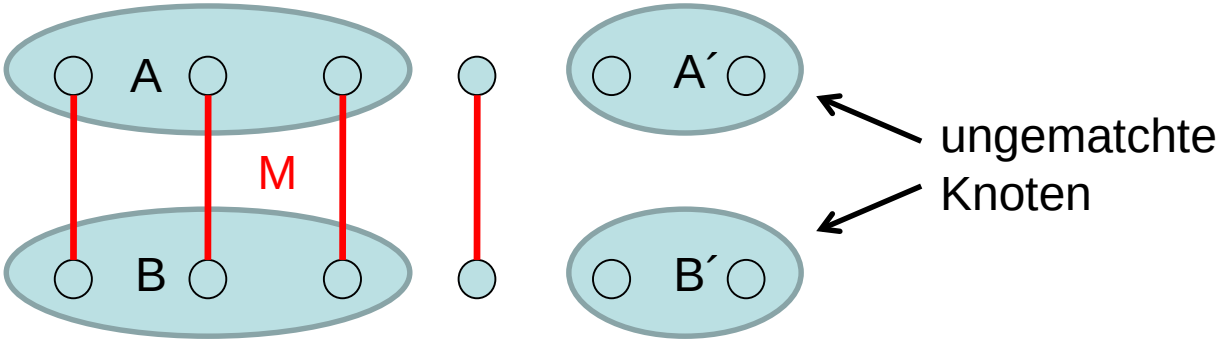
„$\Rightarrow$“: klar



Grundlagen

Beweis:

„⇐“: Sei M ein maximum Matching in G mit |M|<|U|.



A⊆U: Knoten erreichbar durch alternierende Pfade von A´

B⊆V: Knoten erreichbar durch alternierende Pfade von A´

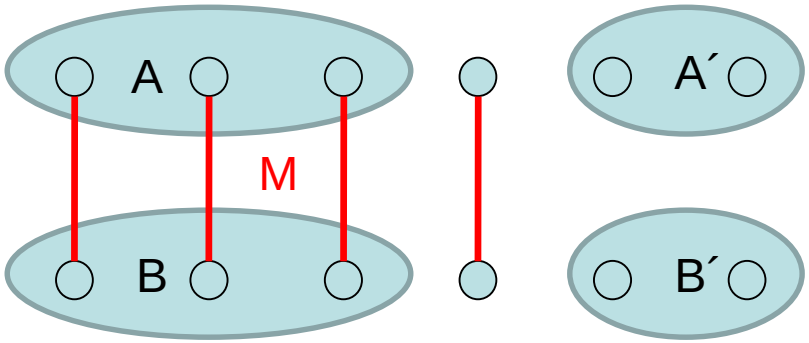
Fälle: (führen beide zum Widerspruch!)

- B∩B´≠∅: es gibt augmentierenden Pfad, also M nicht max.
- B∩B´=∅: da |A|=|B| ist und N(A)=B, folgt |N(A∪A´)| < |A∪A´|

Grundlagen

Beweis:

„⇐“: Sei M ein maximum Matching in G mit |M|<|U|.



B∩B´=∅:

- |A|=|B|, da A={ u∈U | v∈B mit {u,v}∈M }
- N(A)=B: B⊆N(A) ist klar, und wäre B⊂N(A), dann wäre B erweiterbar, was der Definition von B widerspricht.

Grundlagen

Alternativer Beweis:

- Sei M ein Matching in G mit $|M|<|U|$, und sei $u_0 \in U$ ein ungematchter Knoten.
- Da $|N(\{u_0\})| \geq 1$ ist, hat u_0 einen Nachbarn $v_1 \in V$. Falls v_1 ungemacht ist, sind wir fertig, da wir einen augmentierenden Pfad gefunden haben.
- Andernfalls sei $u_1 \in U$ der mit v_1 gematchte Knoten. Da $u_1 \notin \{u_0\}$ ist und $|N(\{u_0,u_1\})| \geq 2$, gibt es einen Knoten $v_2 \notin \{v_1\}$, der zu u_0 oder u_1 adjazent ist. Falls v_2 ungemacht ist, sind wir wieder fertig, da wir einen augmentierenden Pfad gefunden haben.
- Sonst sei $u_2 \in U$ der mit v_2 gematchte Knoten. Da $u_2 \notin \{u_0,u_1\}$ ist und $|N(\{u_0,u_1,u_2\})| \geq 3$, gibt es einen Knoten $v_3 \notin \{v_1, v_2\}$, der zu einem Knoten in $\{u_0,u_1,u_2\}$ adjazent ist. Falls v_3 ungemacht ist, sind wir fertig, sonst machen wir weiter wie oben.
- Da $|M|<|V|$ und $|V|<\infty$, müssen wir schließlich einen ungematchten Knoten v_k finden, und wir können das Matching vergrößern.

Grundlagen

Satz 8.10: (Satz von Berge, bipartite Graphen)
Ein Matching in einem bipartiten Graphen hat maximale Kardinalität genau dann, wenn es keinen augmentierenden Pfad dafür gibt.

Beweis:

„$\Rightarrow$“:

- Angenommen, es gibt einen augmentierenden Pfad P zu einem Matching M.
- Dann folgt aus Lemma 5.8, dass $|M \ominus P| = |M|+1$, also M kein Matching maximaler Kardinalität sein kann.

Grundlagen

Satz 8.10: (Satz von Berge, bipartite Graphen)
Ein Matching in einem bipartiten Graphen hat maximale Kardinalität genau dann, wenn es keinen augmentierenden Pfad dafür gibt.

Beweis:

„$\Leftarrow$“:

- Gilt auf jeden Fall für bipartite Graphen, die den Heiratssatz erfüllen.
- Die allgemeine Gültigkeit zeigen wir später.

Grundlagen

Satz 8.10: (Satz von Berge, bipartite Graphen)
Ein Matching in einem bipartiten Graphen hat maximale Kardinalität genau dann, wenn es keinen augmentierenden Pfad dafür gibt.

Dieser Satz gilt auch allgemein:

Satz 8.11: (Satz von Berge)
Ein Matching in einem Graphen hat maximale Kardinalität genau dann, wenn es keinen augmentierenden Pfad dafür gibt.

Matching in bipartiten Graphen

Algorithmus für Matching maximaler Kardinalität:

$M := \emptyset$
while $\exists$augmentierender Pfad P bzgl. M do
    $M := M \ominus P$
gib M aus

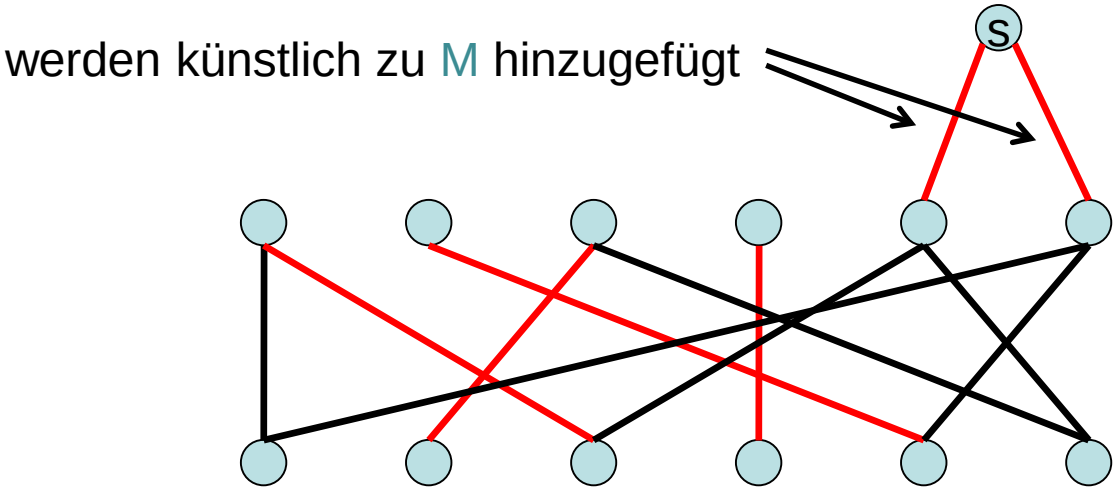
Laufzeit:

- Die While Schleife wird höchstens n-mal durchlaufen.
- Die Suche eines augmentierenden Pfades kann über alternierendes DFS in $O(n+m)$ Zeit gelöst werden.

Also Laufzeit $O(n \cdot (n+m))$ möglich.

Matching in bipartiten Graphen

Vereinfachung für alternierendes DFS in bipartiten Graphen: künstliche Quelle s zu allen ungematchten Knoten



Matching in bipartiten Graphen

- E(u): Kantenmenge von Knoten u

```
Procedure AlternatingBipartiteDFS(s: Node, M: Matching)
    d = <∞,…, ∞>: Array [1..n] of IN
    parent = <⊥,…,⊥>: Array [1..n] of Node
    d[Key(s)]:=0              // s hat Distanz 0 zu sich
    parent[Key(s)]:=s         // s ist sein eigener Vater
    q:=<s>: List of Node      // q:Stack zu besuchender Knoten
    while q ≠<> do            // solange q nicht leer
        u:= q.popFront()      // nimm Knoten nach FIFO-Regel
        if (d[u] is even) then A:=M else A:=M\E
        if A∩E(u)=∅ and (d[u] is even) then
            augmentierender Pfad (über parent[]), stop
        else
            foreach {u,v}∈A do
                if parent(Key(v))=⊥ then  // v schon besucht?
                    q.pushFront(v)  // nein, dann in q vorne einfügen
                    d[Key(v)]:=d[Key(u)]+1
                    parent[Key(v)]:=u
```

Grundlagen

Beweis von Satz 8.11: folgt aus folgendem Lemma.

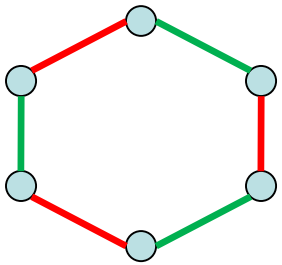
Lemma 8.12: Seien M und N Matchings in G, und sei $|N|>|M|$. Dann enthält $N \ominus M$ mindestens $|N|-|M|$ knotendisjunkte augmentierende Pfade bzgl. M.

Beweis:

Der Grad eines Knotens in $(V, N \ominus M)$ ist maximal 2. Die ZHKs von $(V, N \ominus M)$ sind also

- isolierte Knoten

- einfache Kreise (gerader Länge)



- alternierende Pfade

Grundlagen

Beweis von Lemma 8.11:

- Seien $C_1,\ldots,C_k$ die ZHKs in $(V,N \ominus M)$.
- Dann gilt $M \ominus \underbrace{C_1 \ominus \ldots \ominus C_k}_{N \ominus M} = N$

- Nur die C_i´s, die augmentierende Pfade bzgl. M sind, können das Matching vergrößern, und zwar nur um genau 1.
- Also muss es mindestens $|N|-|M|$ C_i´s geben, die augmentierende (und knotendisjunkte) Pfade bzgl. M sind.

Matching in beliebigen Graphen

Konsequenz: Algorithmus für Matching maximaler Kardinalität in bipartiten Graphen funktioniert auch für beliebige Graphen:

$M := \emptyset$

while $\exists$augmentierender Pfad P bzgl. M do
    $M := M \ominus P$

gib M aus

Im folgenden seien

- P_i: augmentierender Pfad in Durchlauf i
- M_i: Matching am Ende von Durchlauf i

Kürzeste augmentierende Pfade

Lemma 8.12: Sei M ein Matching der Kardinalität r, und sei s die maximale Kardinalität eines Matchings in $G=(V,E)$, $s>r$. Dann gibt es einen augmentierenden Pfad bzgl. M der Länge $\leq 2\lfloor r/(s-r) \rfloor + 1$.

Beweis:

- Sei N ein Matching maximaler Kardinalität in G, d.h. $|N|=s$.
- $N \ominus M$ enthält $\geq s-r$ augmentierende Pfade bzgl. M, die alle knotendisjunkt und damit auch kantendisjunkt sind.
- Mindestens einer dieser augmentierenden Pfade enthält daher $\leq \lfloor r/(s-r) \rfloor$ Kanten aus M.

Kürzeste augmentierende Pfade

Lemma 8.13: Sei s die maximale Kardinalität eines Matchings in $G=(V,E)$. Dann enthält die Folge $|P_1|, |P_2|,\ldots$ der augmentierenden Pfade im Algorithmus höchstens $2\sqrt{s}+1$ verschiedene Werte.

Beweis:

- Sei $r := s-\sqrt{s}$. Per Konstruktion ist $|M_i|=i$, also $|M_r|=r$. Mit Lemma 5.12 folgt

$$|P_r| \leq 2\left\lfloor \frac{\lfloor s-\sqrt{s} \rfloor}{s - \lfloor s-\sqrt{s} \rfloor} \right\rfloor + 1 \leq 2\lfloor s/\sqrt{s} \rfloor + 1 \leq 2\lfloor \sqrt{s} \rfloor + 1$$

- Für $i \leq r$ ist also $|P_i|$ eine der ungeraden Zahlen in $[1, 2\sqrt{s}+1]$, also eine von $\lfloor \sqrt{s} \rfloor + 1$ ungeraden Zahlen. $P_{r+1},\ldots,P_{s-1}$ tragen höchstens $s-r-1 < \sqrt{s}$ zusätzliche Längen bei.

Kürzeste augmentierende Pfade

Lemma 8.14: Sei P ein kürzester augmentierender Pfad bzgl. M und P' ein augmentierender Pfad bzgl. des Matchings $M \ominus P$. Dann gilt:

$$|P'| \geq |P| + 2|P \cap P'|$$

Beweis:

- Sei $N = M \ominus P \ominus P'$, also $|N| = |M| + 2$.
- Dann enthält $M \ominus N$ mindestens 2 knotendisjunkte augmentierende Pfade bzgl. M, etwa P_1 und P_2.
- Es gilt: $|M \ominus N| = |P \ominus P'| = |(P \setminus P') \cup (P' \setminus P)|$
  $= |P| + |P'| - 2|P \cap P'|$
  $\geq |P_1| + |P_2| \geq 2|P|$ (nach Def. von P)
- Also ist $|P| + |P'| - 2|P \cap P'| \geq 2|P|$
  $\Rightarrow \quad |P'| \geq 2|P| - |P| + 2|P \cap P'|$

Kürzeste augmentierende Pfade

Verfeinerter Matching Algorithmus:

$M:=\emptyset$

while $\exists$augmentierender Pfad bzgl. M do
- bestimme den kürzesten augmentierenden Pfad P bzgl. M
- $M:=M \ominus P$

gib M aus

- Sei P_1, P_2,... die Folge der vom Algorithmus verwendeten kürzesten augmentierenden Pfade.
- Lemma 8.14: $|P_{i+1}| \geq |P_i|$ für alle i.

Kürzeste augmentierende Pfade

Lemma 8.15: Für jede Folge $P_1, P_2,...$ kürzester augmentierender Pfade gilt für alle P_i und P_j mit $|P_i|=|P_j|$, dass P_i und P_j knotendisjunkt sind.

Beweis:

- Annahme: es gibt Folge $(P_k)_{k\geq 1}$ mit $|P_i|=|P_j|$ für ein $j>i$, für das P_i und P_j knotendisjunkt sind, wobei $j-i$ minimal sei.
- Dann sind die Pfade $P_{i+1},...,P_j$ knotendisjunkt.
- Also ist P_j auch ein augmentierender Pfad bzgl. des Matchings M nach dem Augmentieren durch $P_1,...,P_i$.
- Aus Lemma 8.14 folgt $|P_j|\geq|P_i|+2|P_i\cap P_j|$, und da $|P_i|=|P_j|$, müssen P_i und P_j kantendisjunkt sein.
- Die durch P_i erzeugten Matchingkanten sind nach wie vor in $M\ominus P_{i+1}\ominus P_{i+2}\ominus...\ominus P_{j-1}$ vorhanden, da $P_i,...,P_{j-1}$ nach Voraussetzung knotendisjunkt sind. Hätte also P_j einen Knoten mit P_i gemeinsam, dann müsste P_j auch eine Kante (nämlich eine Matchingkante) mit P_i gemeinsam haben, was wir oben schon ausgeschlossen haben.
- Also können P_i und P_j auch keinen Knoten gemeinsam haben.

Kürzeste augmentierende Pfade

Verfeinerter Algorithmus:

$M := \emptyset$

while $\exists$augmentierender Pfad bzgl. M do
- l:=Länge eines kürzesten augm. Pfades bzgl. M
- bestimme bzgl. „$\subseteq$“ maximale Menge knoten-disjunkter augm. Pfade $Q_1,...,Q_k$ bzgl. M, die alle Länge l haben
- $M := M \ominus Q_1 \ominus \ldots \ominus Q_k$

Korollar 8.14: Die obige while-Schleife wird höchstens $O(\sqrt{n})$-mal durchlaufen.

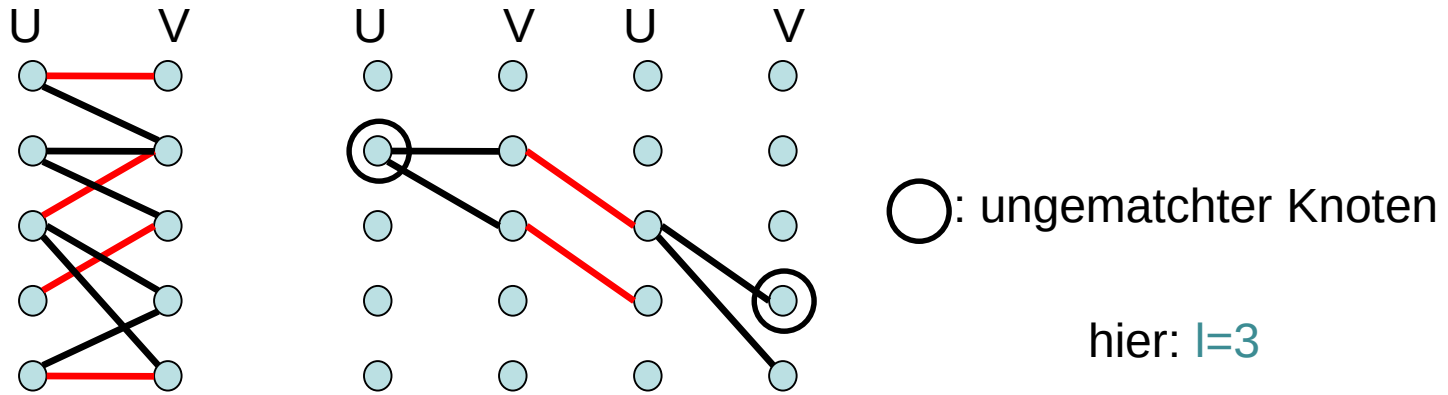
Beweis: folgt aus Lemma 8.13-8.15

Kürzeste augmentierende Pfade

Frage: Wie finden wir schnell eine maximale Menge kürzester augmentierender Pfade bzgl. Matching M?

Graph G bipartit, d.h. G=(U,V,E):

- Bestimmung der kürzesten Länge l: alternierendes BFS, angefangen mit ungematchten Knoten in U, bis ungemachte Knoten in V gefunden



Kürzeste augmentierende Wege

- s: künstlicher Knoten, E(u): Kantenmenge von Knoten u

```
Procedure AlternatingBipartiteBFS(s: Node, M: Matching)
    d = <∞,…, ∞>: Array [1..n] of IN
    parent = <⊥,…,⊥>: Array [1..n] of Node
    d[Key(s)]:=0              // s hat Distanz 0 zu sich
    parent[Key(s)]:=s         // s ist sein eigener Vater
    q:=<s>: List of Node      // q:Stack zu besuchender Knoten
    while q ≠<> do            // solange q nicht leer
        u:= q.popFront()      // nimm Knoten nach FIFO-Regel
        if (d[u] is even) then A:=M else A:=M\E
        if A∩E(u)=∅ and (d[u] is even) then
            augmentierender Pfad (über parent[]), stop
        else
            foreach {u,v}∈A do
                if parent(Key(v))=⊥ then  // v schon besucht?
                    q.pushBack(v)  // nein, dann in q hinten einfügen
                    d[Key(v)]:=d[Key(u)]+1
                    parent[Key(v)]:=u
```

Kürzeste augmentierende Pfade

Graph G bipartit, d.h. G=(U,V,E):

- Bestimmung der kürzesten Länge l: alternierendes BFS, angefangen mit ungematchten Knoten in U, bis ungemachte Knoten in V gefunden oder alle Knoten exploriert.
- Bestimmung einer maximalen Menge kürzester augmentierender Pfade:
  Führe nacheinander von jedem ungematchten Knoten in U eine alternierende DFS bis zur Tiefe l aus, bis wir einen augm. Pfad Q_i gefunden oder alle Kanten exploriert haben. Dabei gilt:
  - Für jeden gefundenen Pfad Q_i werden alle Knoten in Q_i markiert und zum DFS eines noch ungemachten Knotens in U übergegangen.
  - Jeder Knoten, über den der DFS zurücksetzt, wird markiert.
  - Bereits markierte Kanten werden von späteren DFS-Durchläufen nicht nochmal besucht.

Der Zeitaufwand beträgt dann insgesamt O(n+m).

Kürzeste augmentierende Wege

Korollar 8.15: In bipartiten Graphen kann ein Matching maximaler Kardinalität in Zeit $O(\sqrt{n}\,(n+m))$ gefunden werden.

Geht das auch für beliebige Graphen?

Ja, ist aber deutlich aufwändiger:

- Vijay V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}\,E)$ general graph maximum matching algorithm. Combinatorica 14(1), pp. 71-109 (1994).

Übersicht

- Kürzeste Wege
- Minimale Spannbäume
- Matching
- Flussprobleme

Grundlagen

Definition 8.16: Ein Flussnetzwerk (G,s,t,c) besteht aus einem gerichteten Graph $G=(V,E)$, einer Quelle $s \in V$, einer Senke $t \in V$ und einer Kapazitätsfunktion $c:V\times V \to \mathbb{R}_{\geq 0}$, so dass $c(u,v) = 0$ falls $(u,v) \notin E$.

Wir werden im folgenden annehmen, dass $s \rightsquigarrow_G u \rightsquigarrow_G t$ für alle $u \in V$ ist .

Definition 8.17: Sei (G,s,t,c) ein Flussnetzwerk.

a) Ein Fluss in G ist eine Funktion $f:V\times V \to \mathbb{R}$, so dass

$f(u, v) \leq c(u, v)$ für alle $u, v \in V$ (Kapazitätsbedingungen)

$f(u, v) = - f(v, u)$ für alle $u, v \in V$ (Antisymmetrie)

$\Sigma_{v\in V} f(u, v) = 0$ für alle $u \in V \setminus \{s, t\}$ (Flusserhaltungsbedingungen)

b) Der Wert $|f|$ einer Flussfunktion f ist definiert als $|f| = \Sigma_{v\in V} f(s, v)$.

Grundlagen

Bemerkung 8.18: Es sei f ein Fluss oder ein Flussnetzwerk (G,s,t,c). Dann

a) $f(v, v) = 0$ für alle $v \in V$ (wegen Antisymmetrie).
b) $\Sigma_{u \in V} f(u, v) = 0$ für alle $v \in V \setminus \{s, t\}$ (Flusserhaltung & Antisymm.).
c) Für alle $u, v \in V$ mit $(u, v), (v, u) \notin E$ gilt $f(u, v) = f(v, u) = 0$.

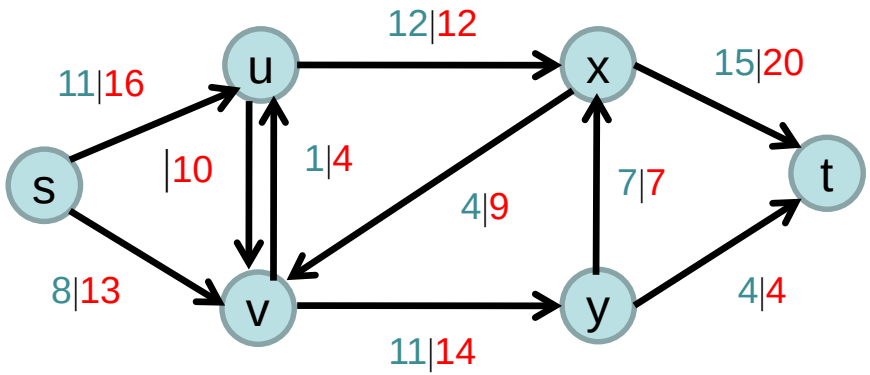
d) Für alle $v \in V \setminus \{s, t\}$ gilt

$$\sum_{\substack{u \in V \\ f(u,v) > 0}} f(u, v) = - \sum_{\substack{u \in V \\ f(u,v) < 0}} f(u,v)$$

e) f mit $f(u, v) = 0$ für alle $u, v \in V$ ist ein Fluss.

Grundlagen

Beispiel für gültigen Fluss:



f (u, v)|c(u, v), |f| = 19.

- f(v,u)=1, also f(u,v)=-1 nach Antisymmetrie.
- Das impliziert, dass für das Paar {u,v} nicht gleich-zeitig in beide Richtungen Fluss fließen kann.
- Warum ist das nicht notwendig?

Grundlagen

Bemerkung 8.19: Der in s ausfließende Fluss ist gleich dem in t einfließenden Fluss, was nicht schwer zu sehen ist. Zunächst stellen wir fest wegen der Antisymmetrie fest:

$$\begin{aligned} & \Sigma_{v\in V} \Sigma_{w\in V} f(v,w) \\ = \; & \Sigma_{\{v,w\}} (f(v,w)+f(w,v)) + \Sigma_{v\in V} f(v,v) \\ = \; & 0 \end{aligned}$$

Weiterhin gilt wegen der Flusserhaltung:

$$\begin{aligned} \Sigma_{v\in V} \Sigma_{w\in V} f(v,w) &= \Sigma_{w\in V} f(s,w) + \Sigma_{w\in V} f(t,w) \\ &= |f| + \Sigma_{w\in V} f(t,w) \end{aligned}$$

Also gilt wegen der Antisymmetrie:

$$|f| = \Sigma_{w\in V} f(w,t)$$

Grundlagen

Alternative Definition von Flüssen:

Definition 8.20: Sei (G,s,t,c) ein Flussnetzwerk. Ein Fluss in G ist eine Funktion $f : E \to \mathbb{R}_{\geq 0}$ so dass

- $0 \leq f(u, v) \leq c(u, v)$ für alle $(u, v) \in E$ (Kapazitätsbedingungen)
- $\Sigma_{v \in V} f(u, v) - \Sigma_{v \in V} f(v, u) = 0$ für alle $u \in V \setminus \{s, t\}$ (Flusserhaltungsbedingungen)

Definition 8.20 ist intuitiver, wobei Definition 8.17 restriktiver ist und manchmal leichtere Beweise erlaubt. Wir benutzen die alternative Definition 8.20 in späteren Teilen dieses Kapitels.

Problem MAXFLOW:

Eingabe: Ein Flussnetzwerk (G,s,t,c).
Ausgabe: Ein Fluss f in G mit maximalem Wert $|f|$.

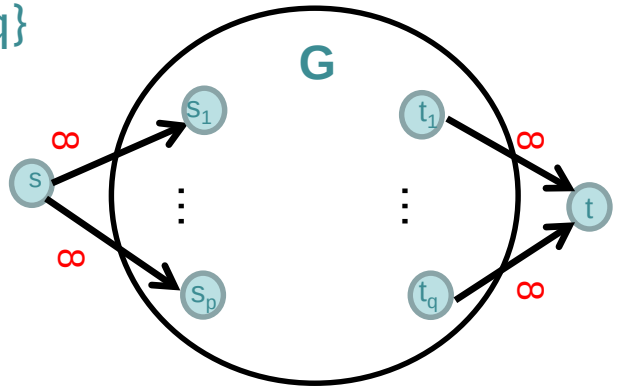
Bemerkung 8.21: Ein maximales Flussproblem $(G, s_1, \ldots, s_p, t_1, \ldots t_q, c)$ mit mehreren Quellen $s_1, \ldots, s_p$ und mehreren Senken $t_1, \ldots t_q$ mit dem Ziel, so viele Güter wie möglich von den Quellen zu den Senken zu transportieren, kann einfach auf das original MAXFLOW-Problem reduziert werden:

Konstruiere $G' = (V', E')$ und c' wie folgt:

$V' = V \cup \{s, t\}$

$E' = E \cup \{(s, s_i) \mid 1 \leq i \leq p\} \cup \{(t_i, t) \mid 1 \leq i \leq q\}$

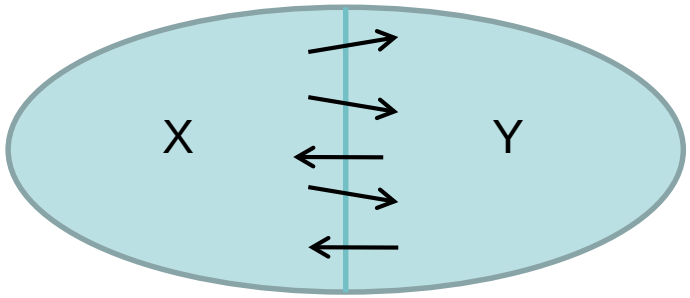
$$c'(u, v) = \begin{cases} c(u, v) & u, v \in V \\ \infty & u = s \text{ oder } v = t \end{cases}$$



Dann existiert ein Fluss f von $s_1, \ldots, s_p$ mit $t_1, \ldots, t_q$ Wert φ in $(G, s_1, \ldots, s_p, t_1, \ldots t_q, c)$ genau dann, wenn ein Fluss f' von s nach t in (G', s, t, c') mit Wert φ existiert.

Definition 8.22: Es sei (G,s,t,c) ein Flussnetzwerk. Für $X, Y \subset V$ bezeichnen wir

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y), \quad c(X, Y) = \sum_{x \in X} \sum_{y \in Y} c(x, y) \text{ und } X - v = X \setminus \{v\}$$



Lemma 8.23: Es sei (G,s,t,c) ein Flussnetzwerk und es sei f ein Fluss in G.
Dann gilt für alle $X, Y, Z \subseteq V$.

a) $f(X, X) = 0$

b) $f(X, Y) = -f(Y, X)$

c) Wenn $X \cap Y = \emptyset$ dann
$f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ und $f(Z, X \cup Y) = f(Z, X) + f(Z, Y)$

Beweis: Übung

Die Ford-Fulkerson Methode

Definition 8.24: Es sei (G,s,t,c) ein Flussnetzwerk und f ein Fluss in G.

a) Für $u, v \in V$ ist die Restkapazität $c_f(u,v)$ definiert als

$$c_f(u,v) = c(u,v) - f(u,v).$$

b) Das Residualnetzwerk $G_f = (V,E_f)$ ist definiert als

$$E_f = \{ (u,v) \in V \times V \mid c_f(u,v) > 0 \}$$

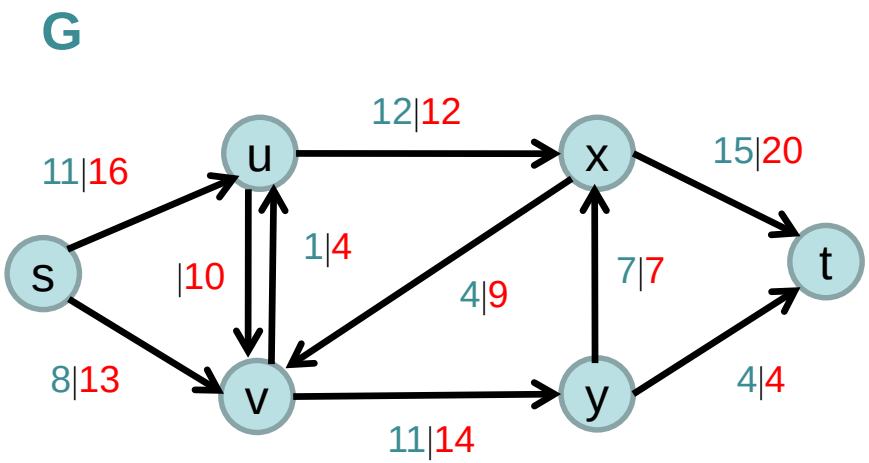
c) Ein einfacher Pfad P von s zu t in G_f wird augmentierender Pfad genannt. Die Restkapazität $c_f(P)$ von P ist definiert als

$$c_f(P) = \min \{ c_f(u,v) \mid (u,v) \in P \}.$$

Die Ford-Fulkerson Methode

Beispiel: augmentierender Pfad und Flussaugmentierung

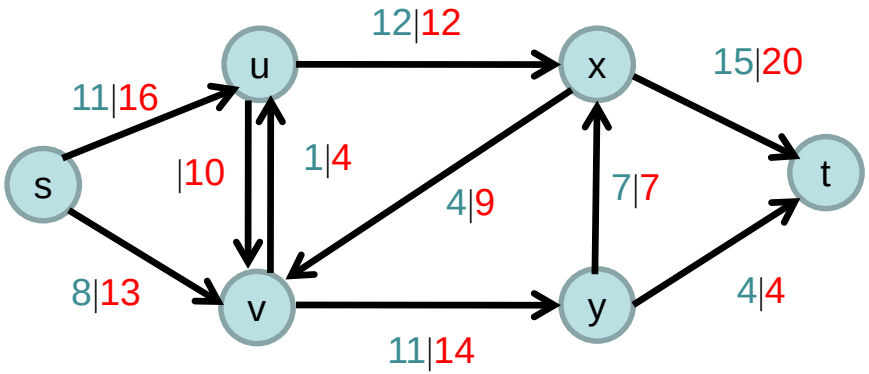
Flussnetzwerk:



Beispiel: augmentierender Pfad und Flussaugmentierung

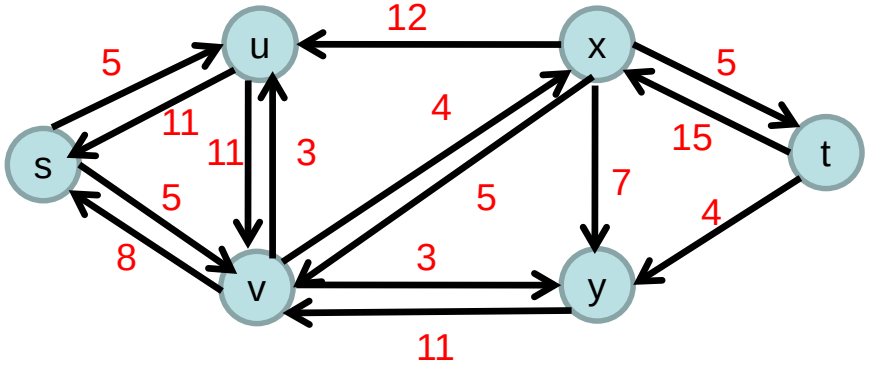
Flussnetzwerk:

G



Residualnetzwerk:

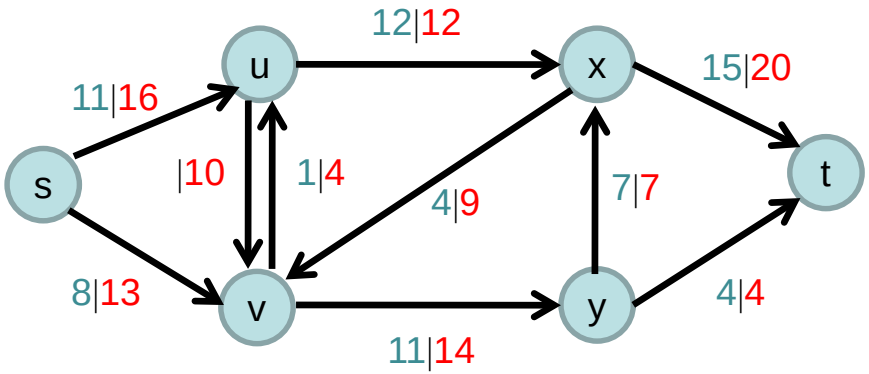
G_f



Beispiel: augmentierender Pfad und Flussaugmentierung

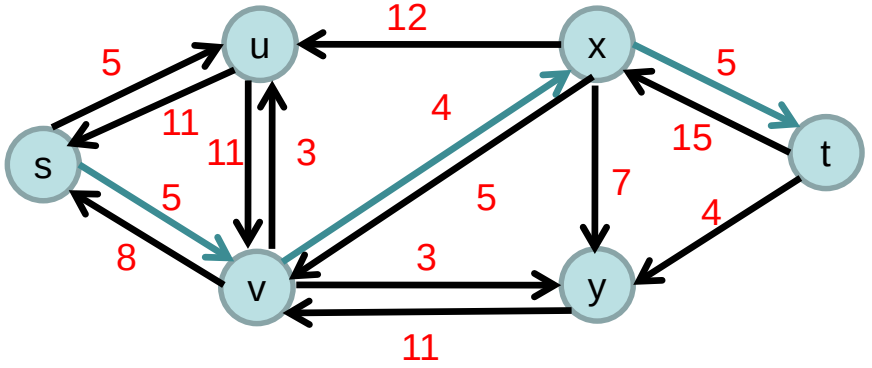
Flussnetzwerk:

G



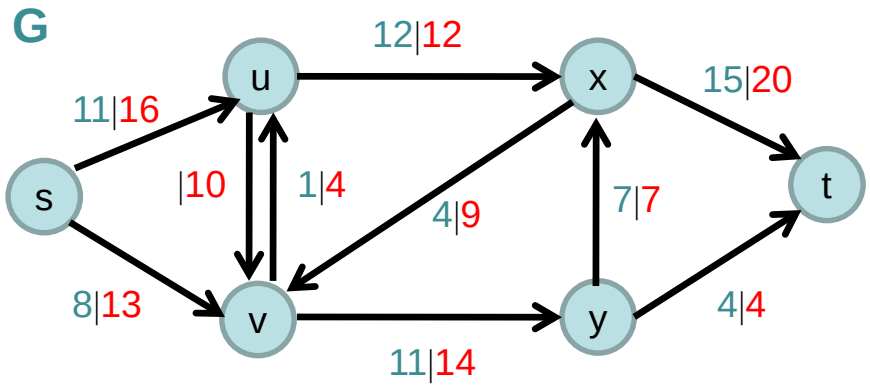
Residualnetzwerk mit augmentierendem Pfad:

G_f

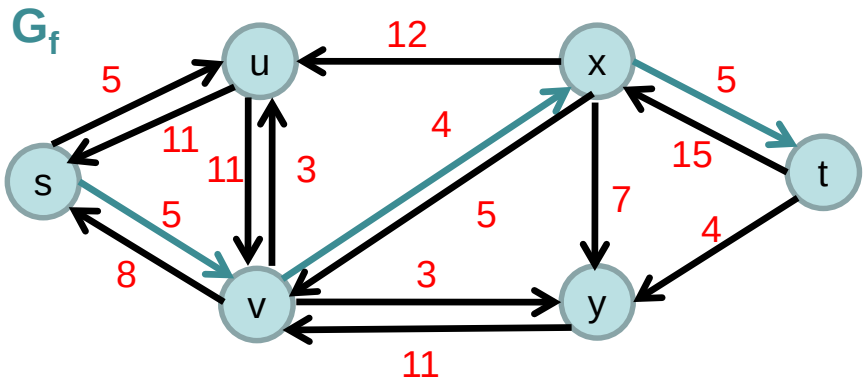


Beispiel: augmentierender Pfad und Flussaugmentierung

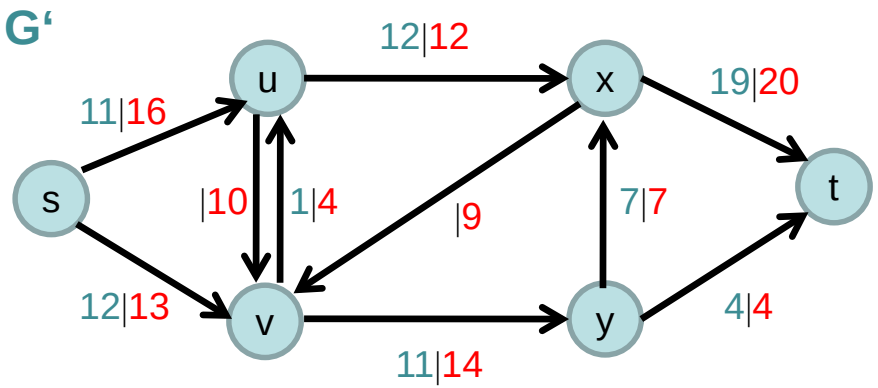
Flussnetzwerk:



Residualnetzwerk mit augmentierendem Pfad:

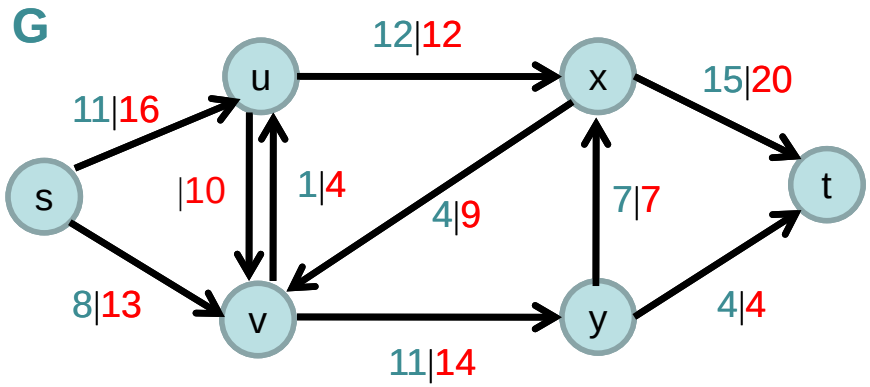


Augmentiertes Flussnetzwerk:

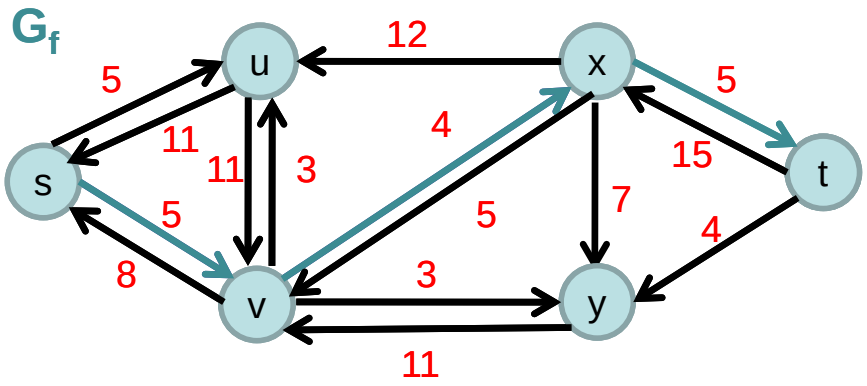


Beispiel: augmentierender Pfad und Flussaugmentierung

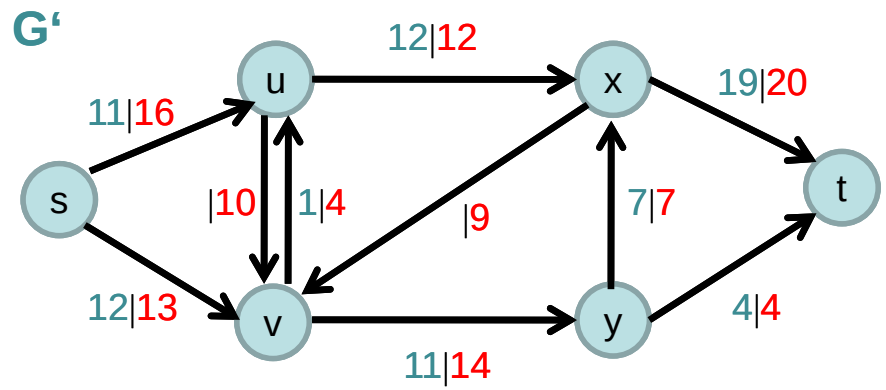
Flussnetzwerk:



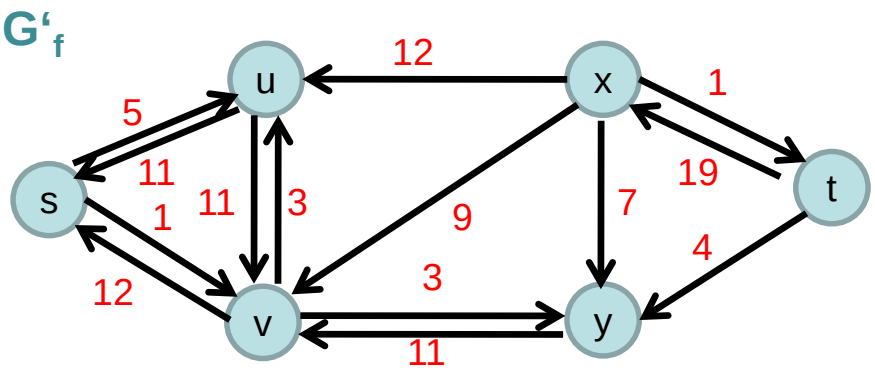
Residualnetzwerk mit augmentierendem Pfad:



Augmentiertes Flussnetzwerk:



Neues Residualnetzwerk:



Der Ford-Fulkerson Algorithmus

```
FORDFULKERSON (Flussnetzwerk G = (V, E), s, t, c))
{
   für jede Kante (u, v) ∈ E
        { f [u, v] := 0; f [v, u] := 0; }                    // initialisiere leeren Fluss
   G_f = Residualnetzwerk von G  bezüglich f;
   solange (∃ ein Pfad P von s zu t in G_f)                  // P ist der augmentierende Pfad
   {  // berechne maximal hinzufügbaren Fluss via P
        c_f (P) := min {c_f (u, v) | (u, v) ∈ P)};           // c_f (u, v) = c (u, v) – f (u, v)
        für jede Kante (u, v) ∈ P                            // aktualisiere den Fluss von P
           { f [u, v] := f [u, v] + c_f (P); f [v, u] := - f [u, v]; }
         G_f := Residualnetzwerk von G bezüglich f;
   }
   gib f aus
}
```

Lemma 8.25: Es sei (G, s, t, c) ein Flussnetzwerk und f ein Fluss in G. Es sei G_f ein Residualnetzwerk von G induziert durch f und es sei f' ein Fluss in G_f. Dann ist

$$(f + f')(u, v) = f(u, v) + f'(u, v)$$

ein gültiger Fluss in G mit Wert $|f + f'| = |f| + |f'|$.

Lemma 8.26: Es sei (G, s, t, c) ein Flussnetzwerk und f ein Fluss in G. Es sei G_f das Residualnetzwerk von G induziert durch f und es sei P ein augmentierender Pfad in G_f. Dann ist $f_P : V \times V \to \mathbb{R}$ mit

$$f_P(u,v) = \begin{cases} c_f(P) & \text{wenn } (u,v) \text{ auf } P \\ -c_f(P) & \text{wenn } (v,u) \text{ auf } P \\ 0 & \text{sonst} \end{cases}$$

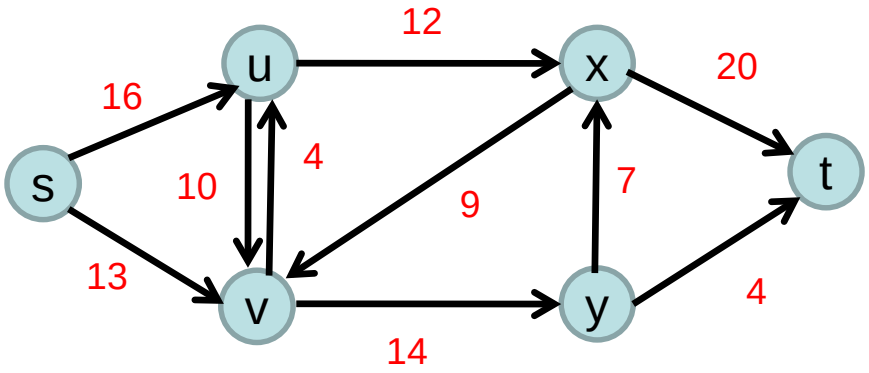
ein gültiger Fluss in G_f mit Wert $|f_P| = c_f(P) > 0$.

Korollar 8.27: Es sei (G, s, t, c) ein Flussnetzwerk und f ein Fluss in G. Es sei G_f das Residualnetzwerk von G induziert durch f und es sei P ein augmentierender Pfad in G_f. Es sei f_P definiert wie in Lemma 6.11. Dann ist $f' = f + f_P$ ein gültiger Fluss in G mit Wert $|f'| = |f + f_P| = |f| + |f_P| > |f|$.

Beispiel: Ford-Fulkerson Algorithmus

Flussnetzwerk:

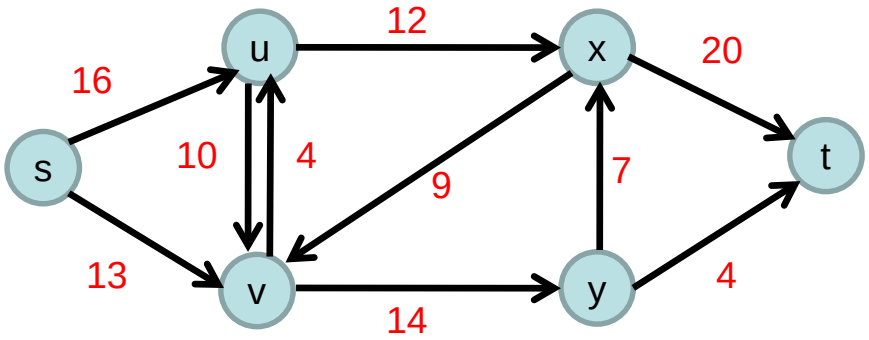
G



Beispiel: Ford-Fulkerson Algorithmus

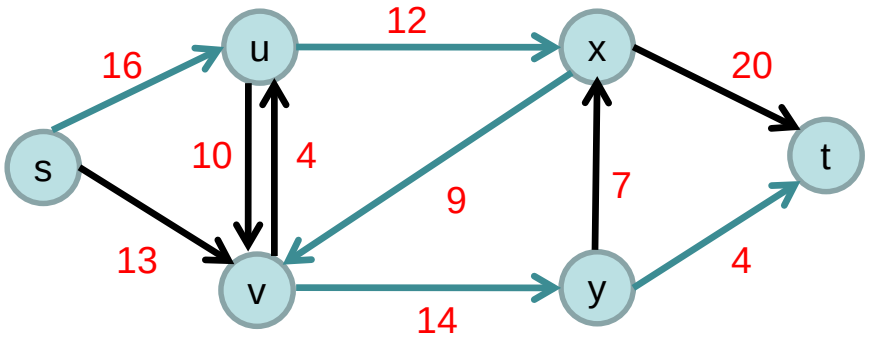
Flussnetzwerk:

G



Residualnetzwerk mit augmentierendem Pfad:

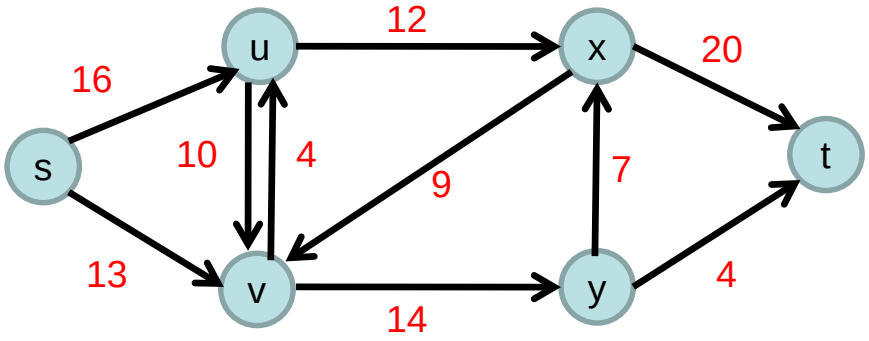
G_f



Beispiel: Ford-Fulkerson Algorithmus

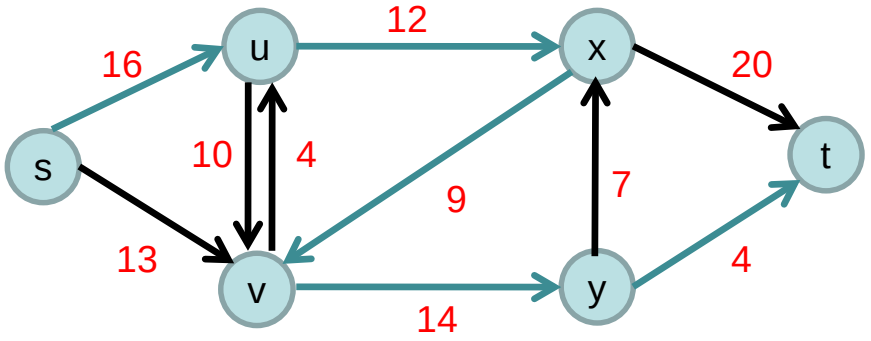
Flussnetzwerk:

G



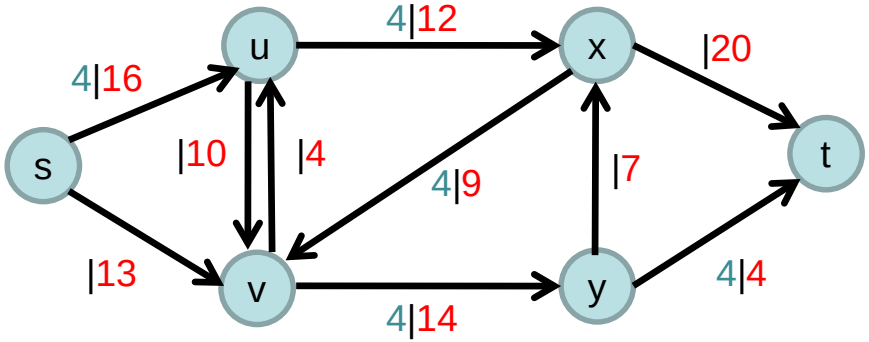
Residualnetzwerk mit augmentierendem Pfad:

G_f



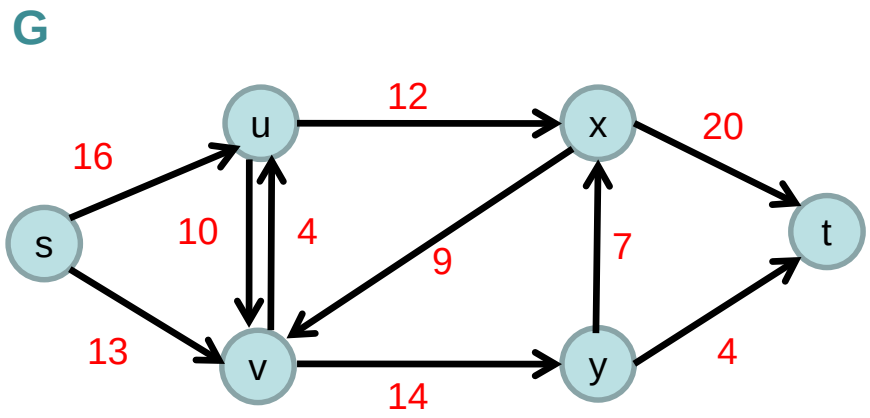
Augmentiertes Flussnetzwerk:

G

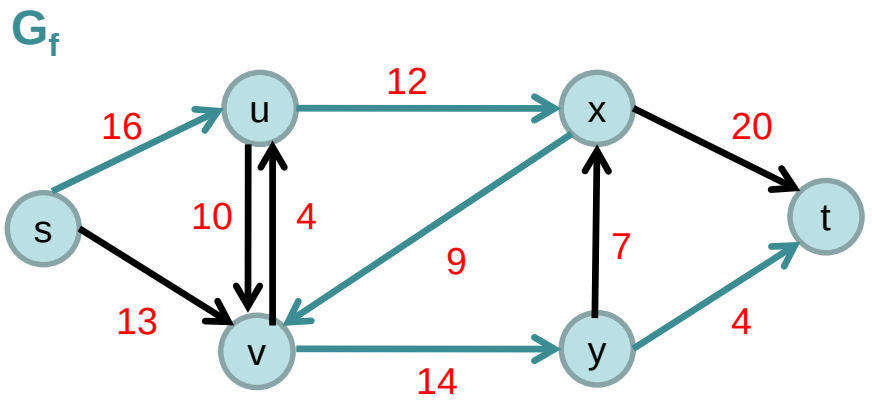


Beispiel: Ford-Fulkerson Algorithmus

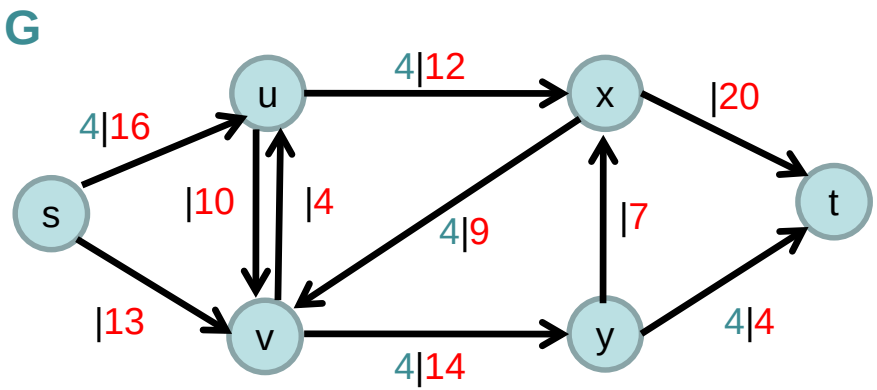
Flussnetzwerk:



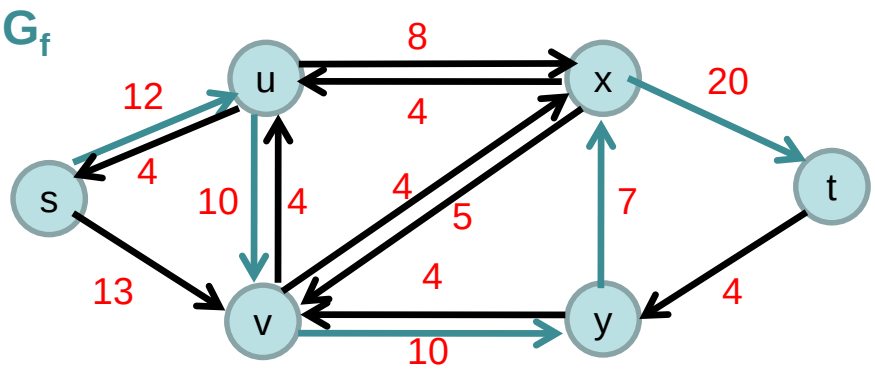
Residualnetzwerk mit augmentierendem Pfad:



Augmentiertes Flussnetzwerk:

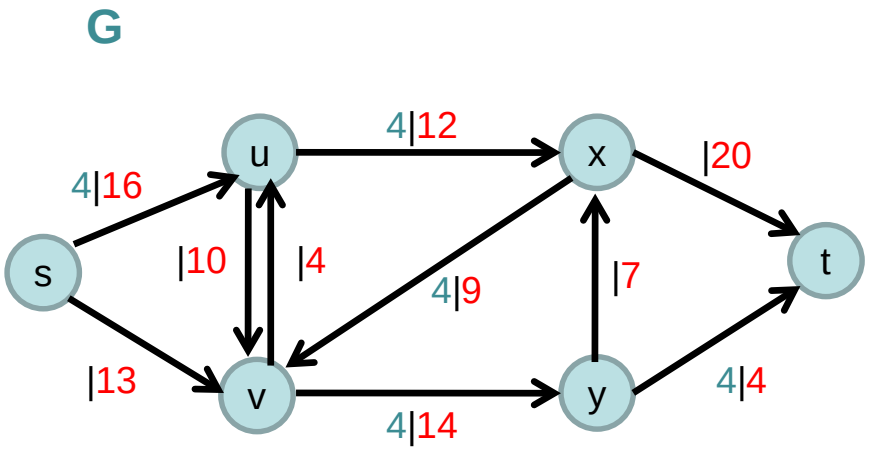


Neues Residualnetzwerk mit augmentierendem Pfad:

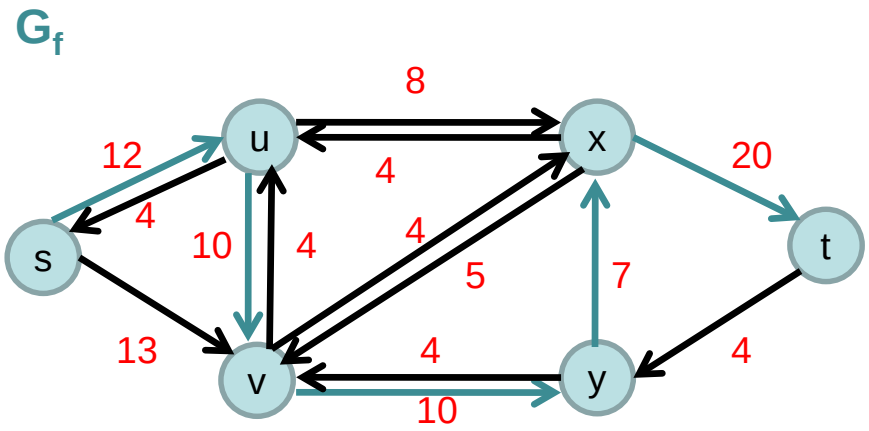


Beispiel: Ford-Fulkerson Algorithmus

Flussnetzwerk:



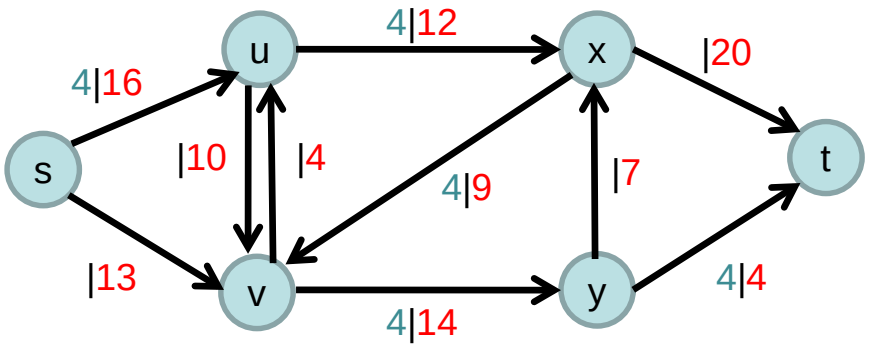
Residualnetzwerk mit augmentierendem Pfad:



Beispiel: Ford-Fulkerson Algorithmus

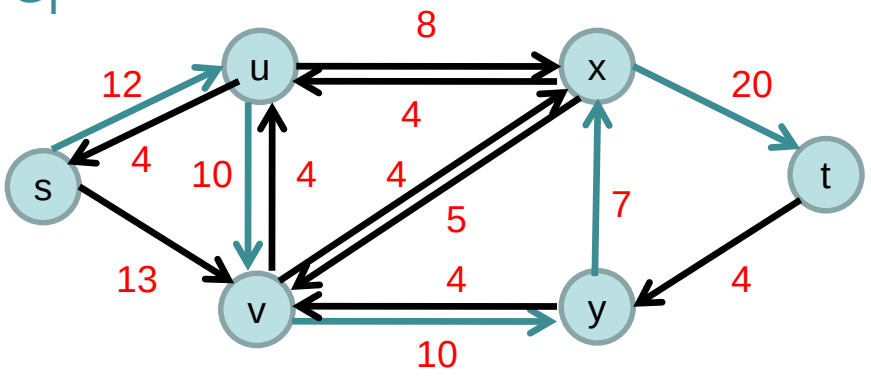
Flussnetzwerk:

G



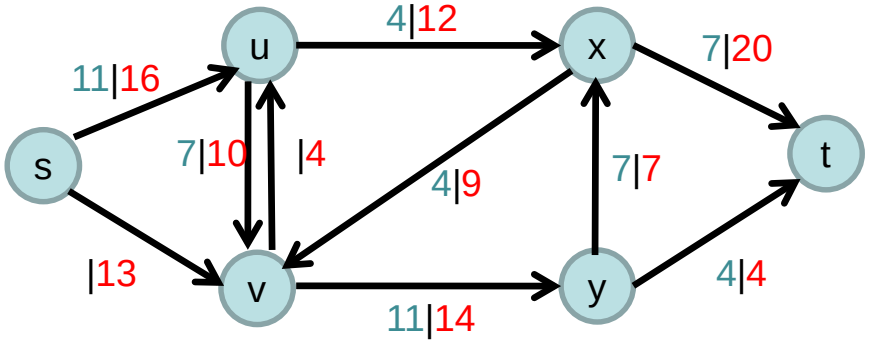
Residualnetzwerk mit augmentierendem Pfad:

G_f



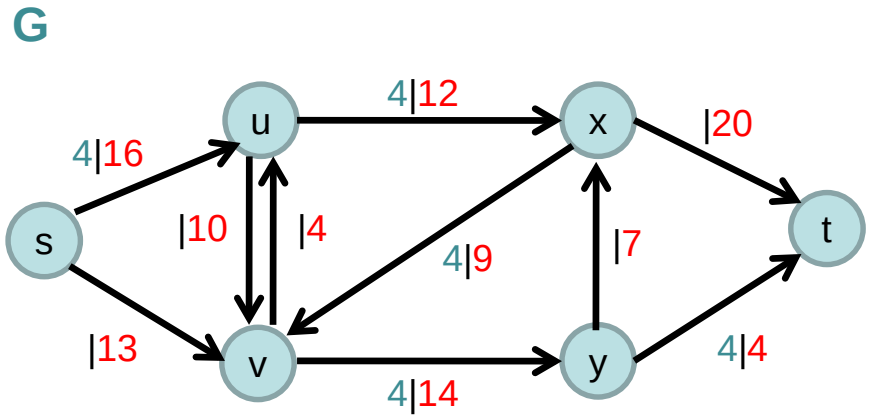
Augmentiertes Flussnetzwerk:

G

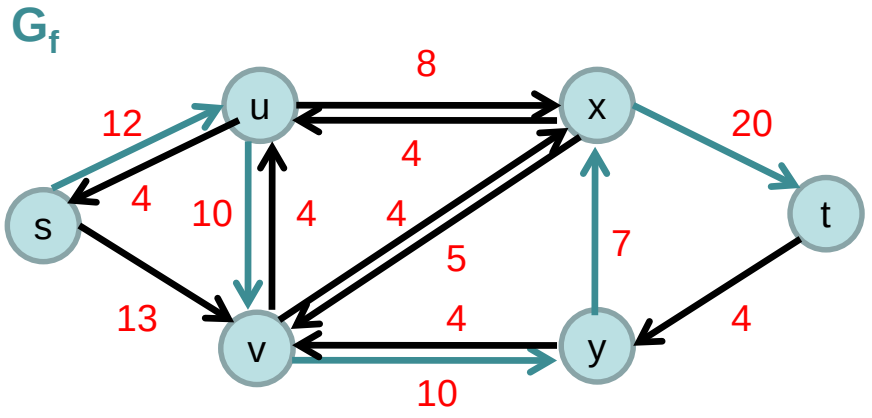


Beispiel: Ford-Fulkerson Algorithmus

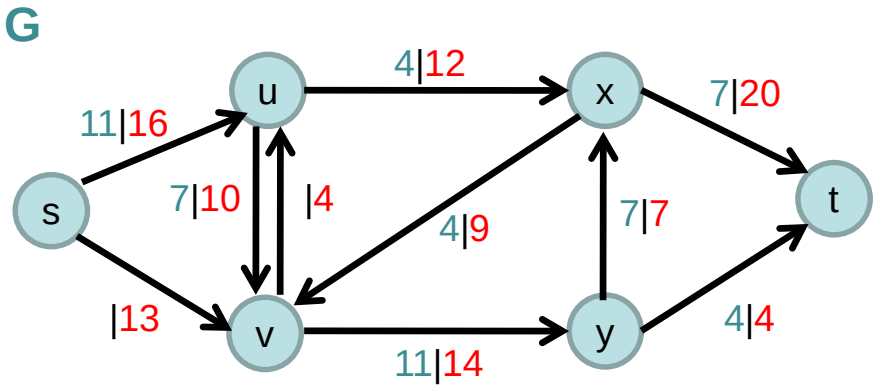
Flussnetzwerk:



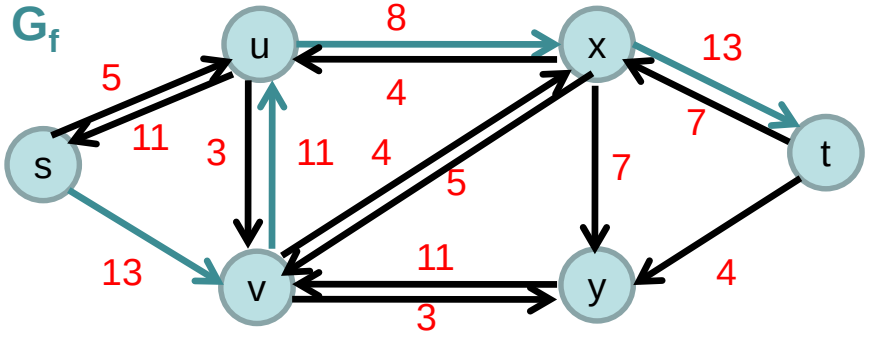
Residualnetzwerk mit augmentierendem Pfad:



Augmentiertes Flussnetzwerk:



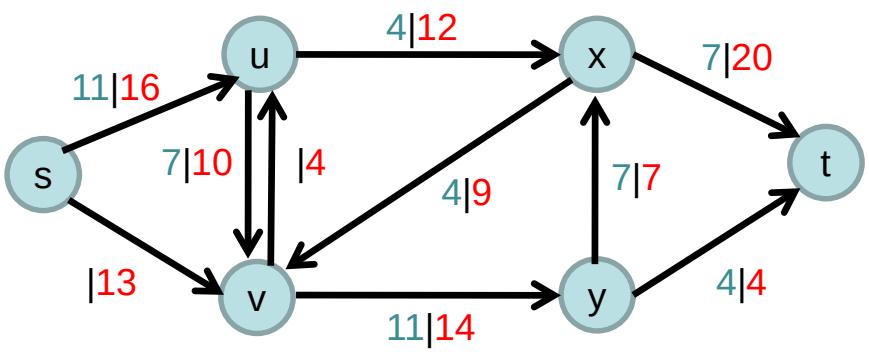
Neues Residualnetzwerk mit augmentierendem Pfad:



Beispiel: Ford-Fulkerson Algorithmus

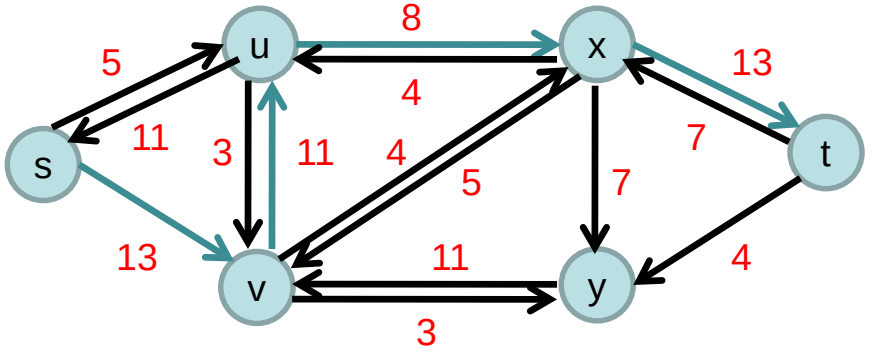
Flussnetzwerk:

G



Residualnetzwerk mit augmentierendem Pfad:

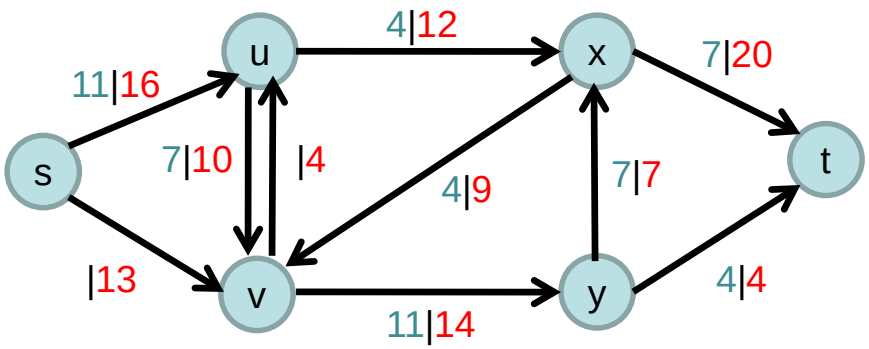
G_f



Beispiel: Ford-Fulkerson Algorithmus

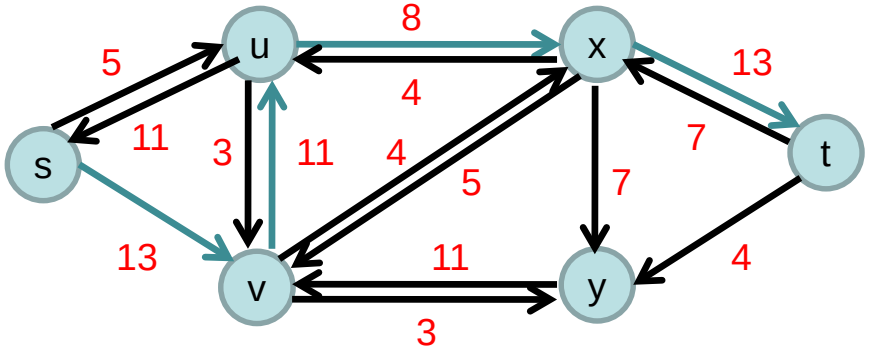
Flussnetzwerk:

G



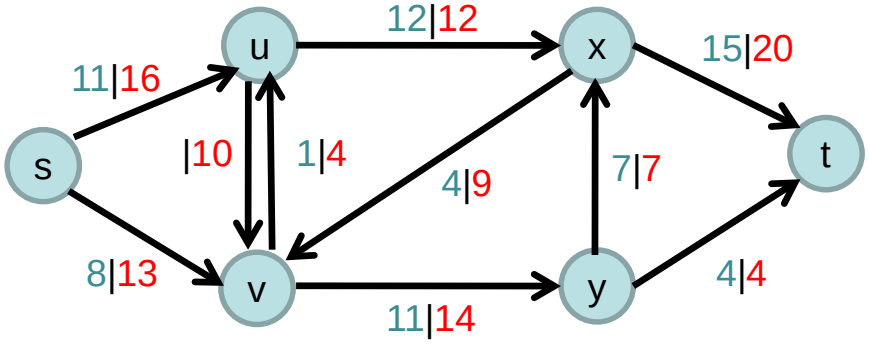
Residualnetzwerk mit augmentierendem Pfad:

G_f



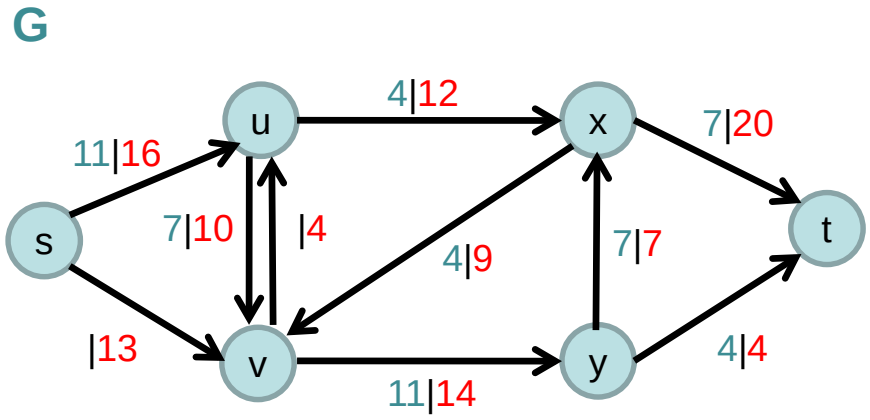
Augmentiertes Flussnetzwerk:

G

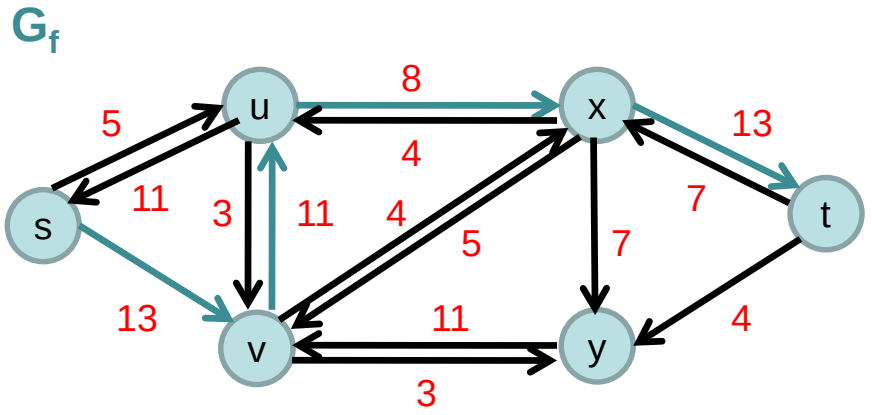


Beispiel: Ford-Fulkerson Algorithmus

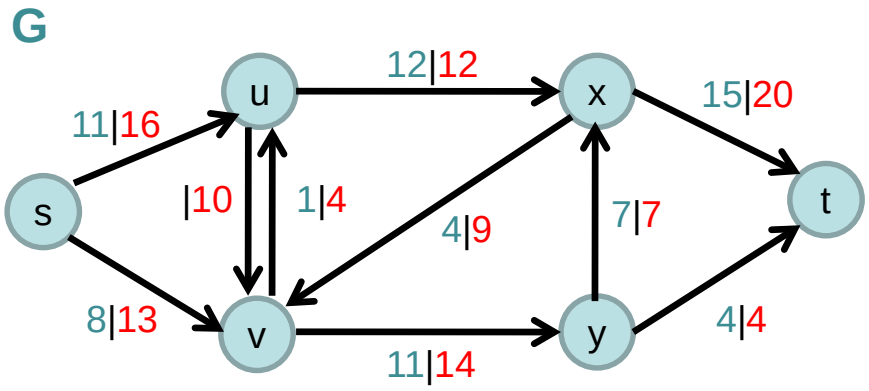
Flussnetzwerk:



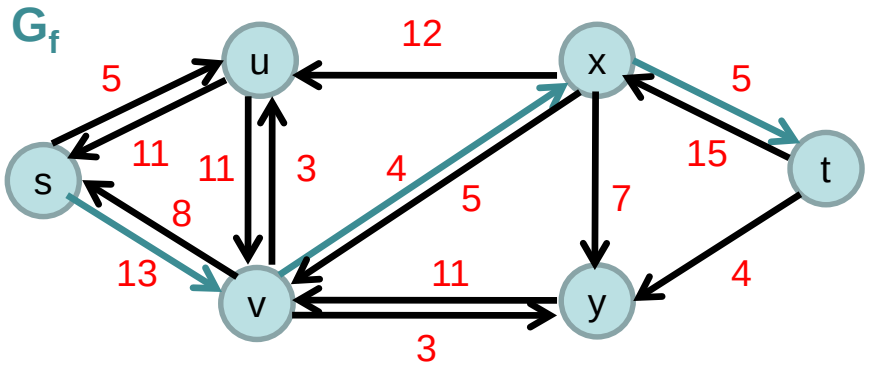
Residualnetzwerk mit augmentierendem Pfad:



Augmentiertes Flussnetzwerk:



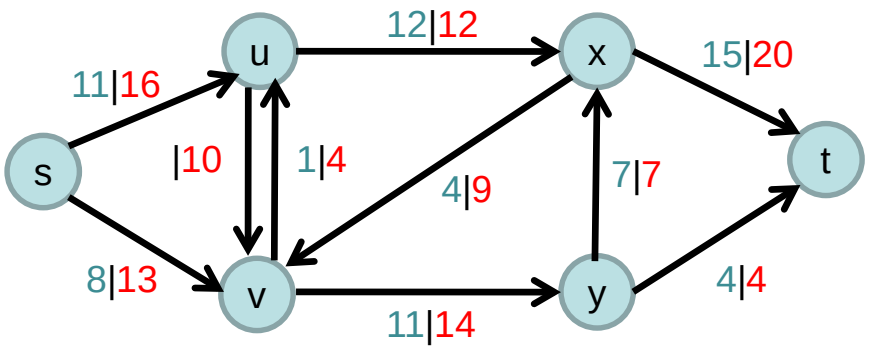
Neues Residualnetzwerk mit augmentierendem Pfad:



Beispiel: Ford-Fulkerson Algorithmus

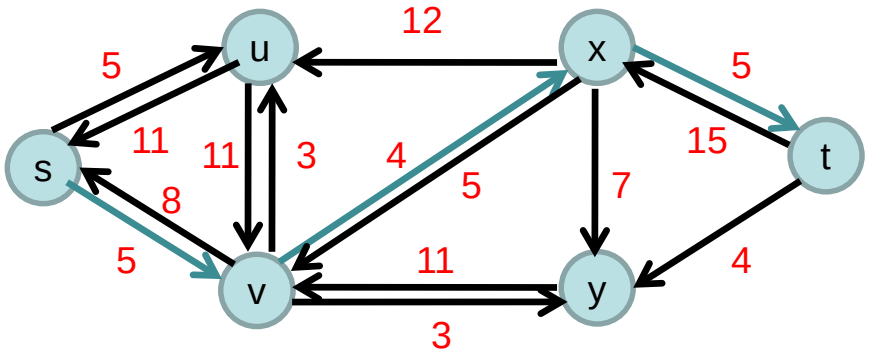
Flussnetzwerk:

G



Residualnetzwerk mit augmentierendem Pfad:

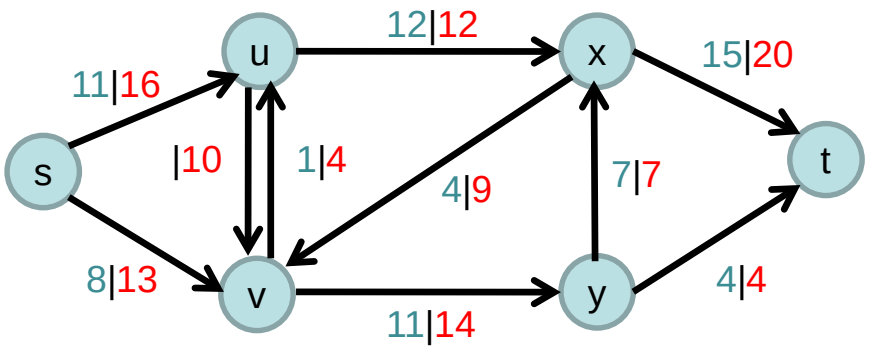
G_f



Beispiel: Ford-Fulkerson Algorithmus

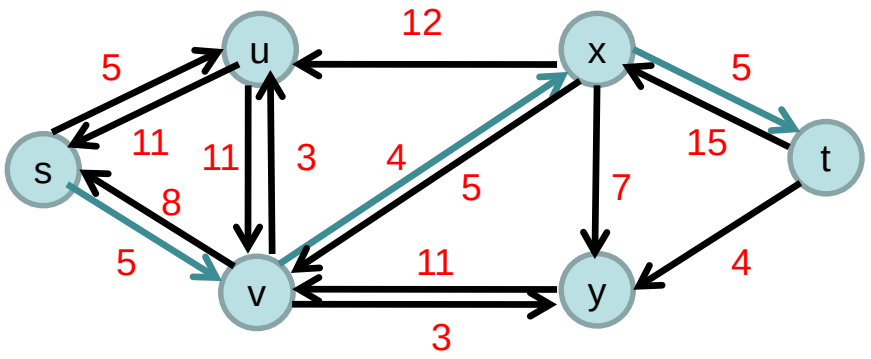
Flussnetzwerk:

G



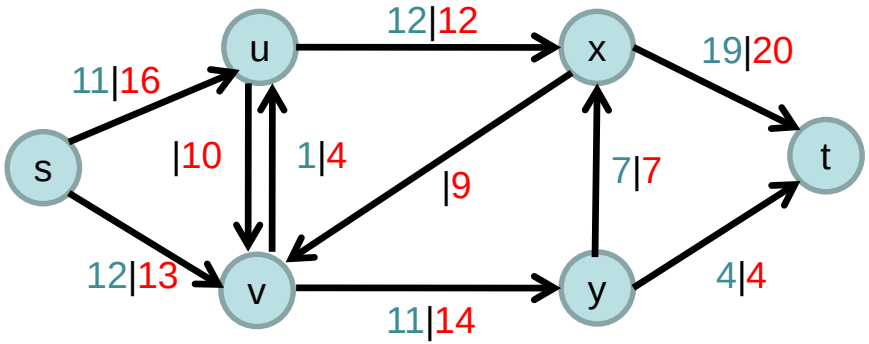
Residualnetzwerk mit augmentierendem Pfad:

G_f



Augmentiertes Flussnetzwerk:

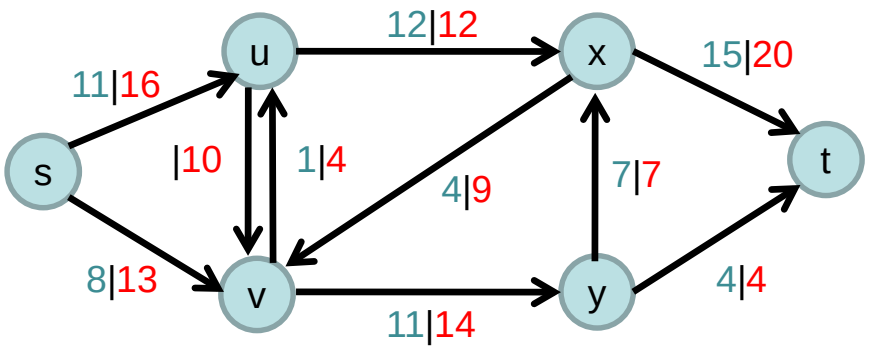
G



Beispiel: Ford-Fulkerson Algorithmus

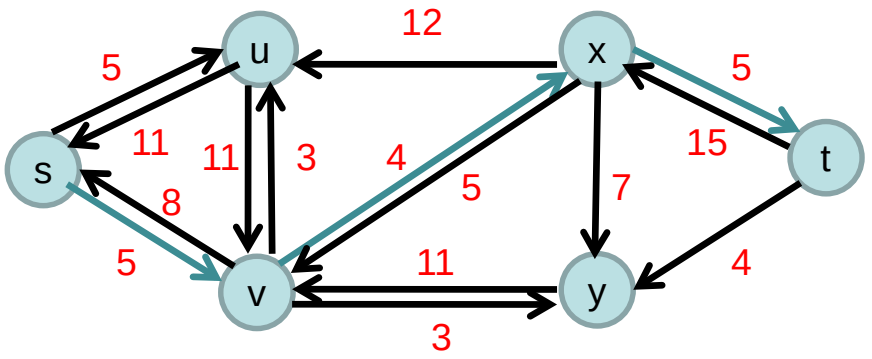
Flussnetzwerk:

G



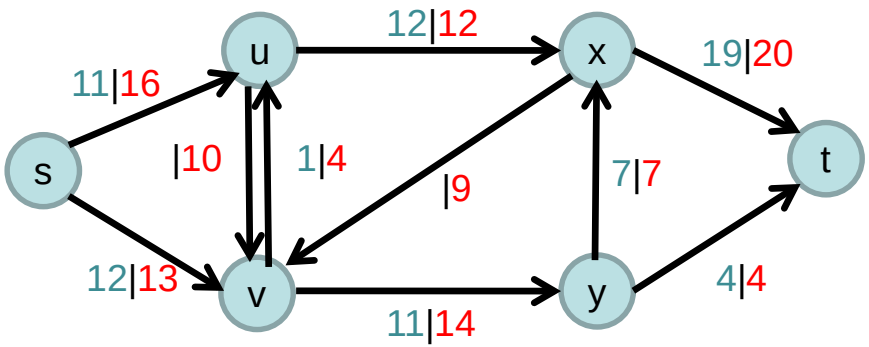
Residualnetzwerk mit augmentierendem Pfad:

G_f



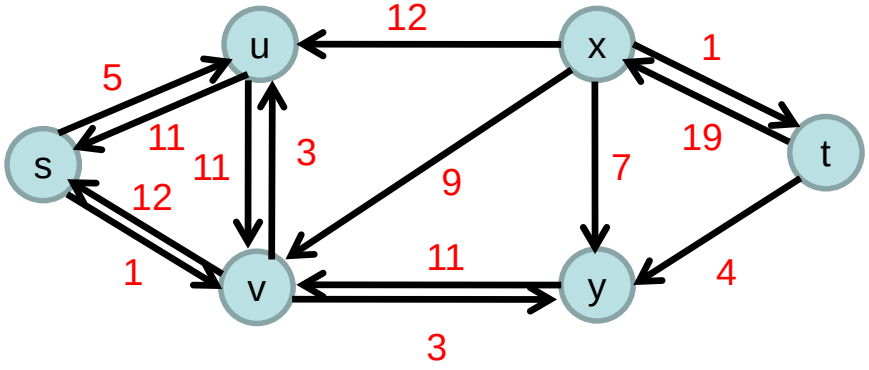
Augmentiertes Flussnetzwerk:

G



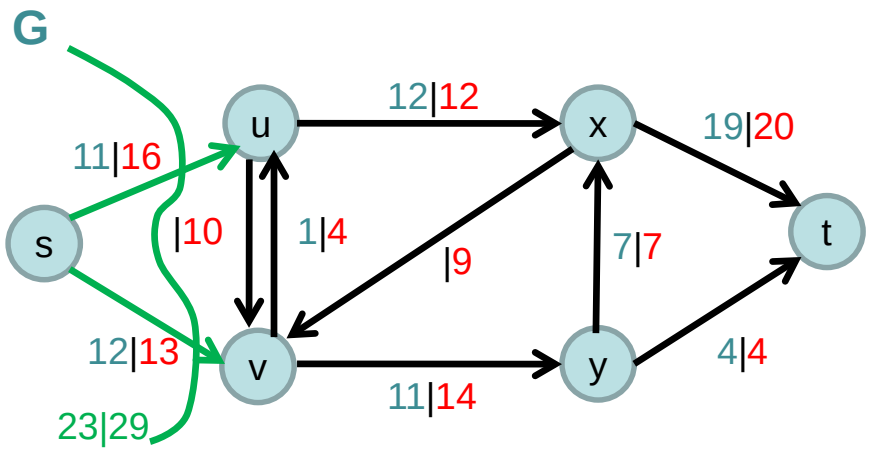
Neues Residualnetzwerk mit augmentierendem Pfad:

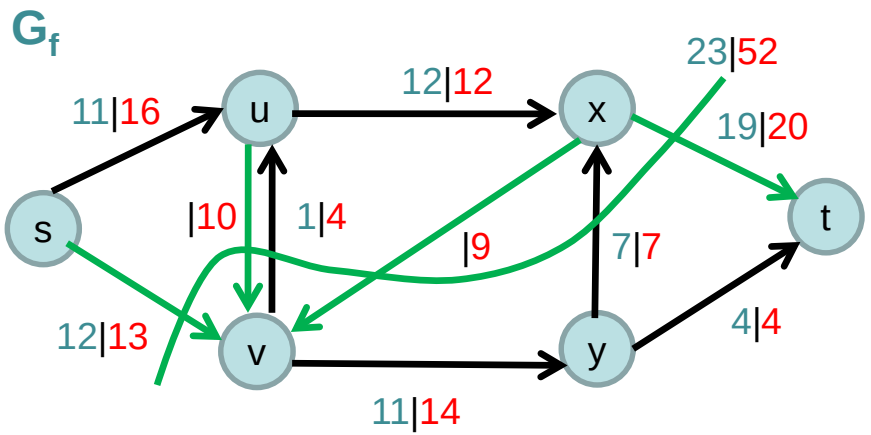
G



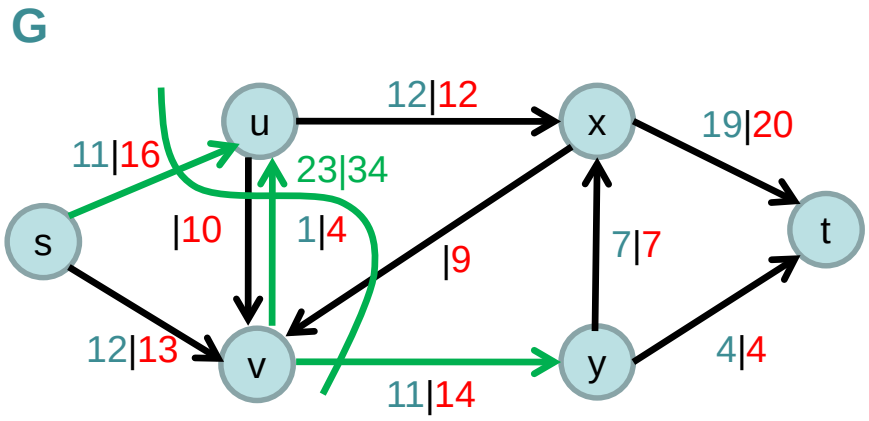
Beispiel: Ford-Fulkerson Algorithmus

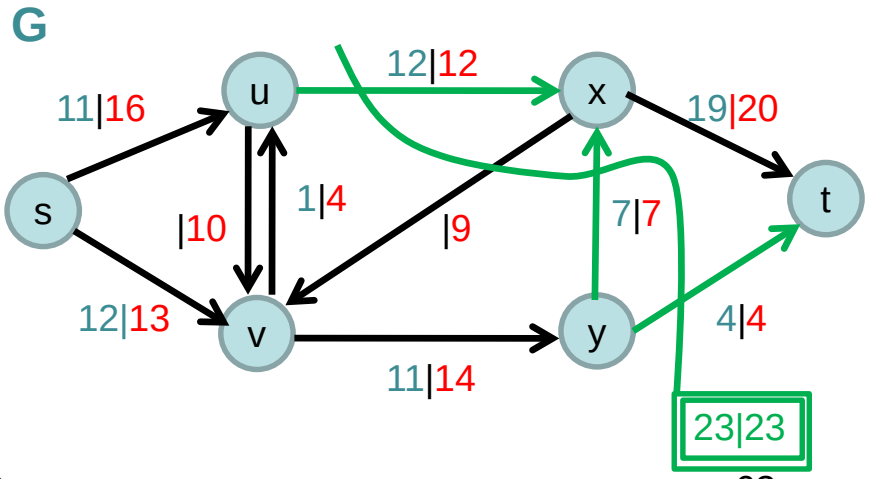
Flussnetzwerk:





Augmentiertes Flussnetzwerk:





Schnitte von Flussnetzwerken

Definition 8.28: Es sei (G, s, t, c) ein Flussnetzwerk und f ein Fluss in G.

a) Ein Schnitt (S, T) von G ist eine Aufteilung von V in S und $T = V \setminus S$ so dass $s \in S$ und $t \in T$.
b) Der Fluss über einen Schnitt (S, T) ist $f(S, T)$.
c) Die Kapazität eines Schnittes (S, T) ist $c(S, T)$.

Bemerkung 8.29:

a) Die Definition eines Flusses ist konsistent mit den Flüssen, die in den Beispielen berechnet wurden: Flüsse von T zu S werden abgezogen:

$$f(S, T) = \sum_{x \in S} \sum_{y \in T} f(x, y)$$ (wobei $f(x, y) < 0$ falls $f(y,x) > 0$).

b) Die Definition einer Kapazität eines Schnittes ist konsistent mit den Kapazitäten, die am Ende des vorigen Beispiels berechnet wurden: Kanten von T zu S fügen der Kapazität des Schnittes nichts hinzu:

$$c(S, T) = \sum_{x \in S} \sum_{y \in T} c(x, y) \quad \text{wobei } c(x, y) \geq 0.$$

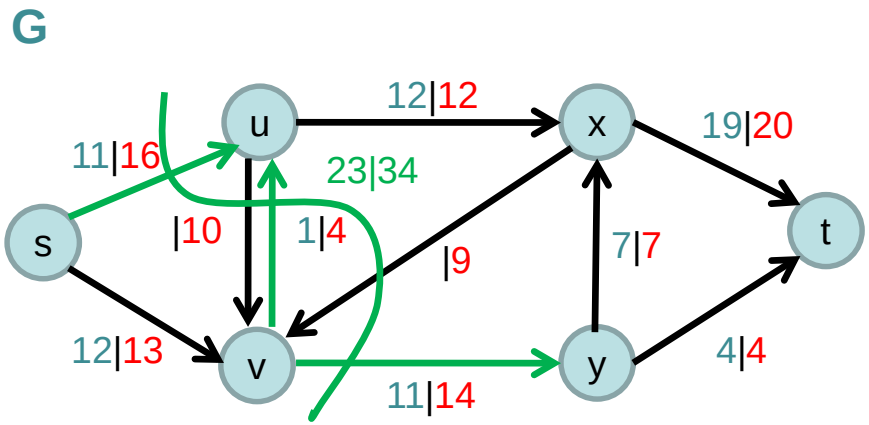
Lemma 8.30: Es sei (G, s, t, c) ein Flussnetzwerk und f ein Fluss in G. Es sei (S, T) ein Schnitt von G. Dann gilt

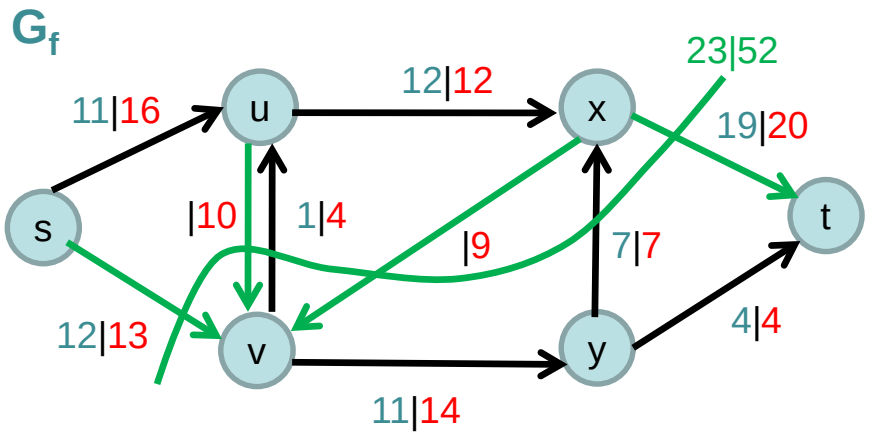
$$f(S, T) = |f|.$$

Und insbesondere auch

$$|f| = f(s, V - s) = f(V - t, t).$$

Korollar 8.31: Es sei (G, s, t, c) ein Flussnetzwerk. Dann ist der Wert von jedem Fluss f in G nach oben beschränkt durch die Kapazität eines beliebigen Schnitts in G.





Theorem 8.32: (Max-Flow Min-Cut Theorem)
Es sei (G,s,t,c) ein Flussnetzwerk und f ein Fluss in G. Dann sind folgende Aussagen äquivalent.

a) f ist ein maximaler Fluss in G.
b) Das Residualnetzwerk G_f von G bezüglich f enthält keinen augmentierenden Pfad.
c) $|f| = c(S, T)$ für einen Schnitt (S, T) von G.

Beweis:

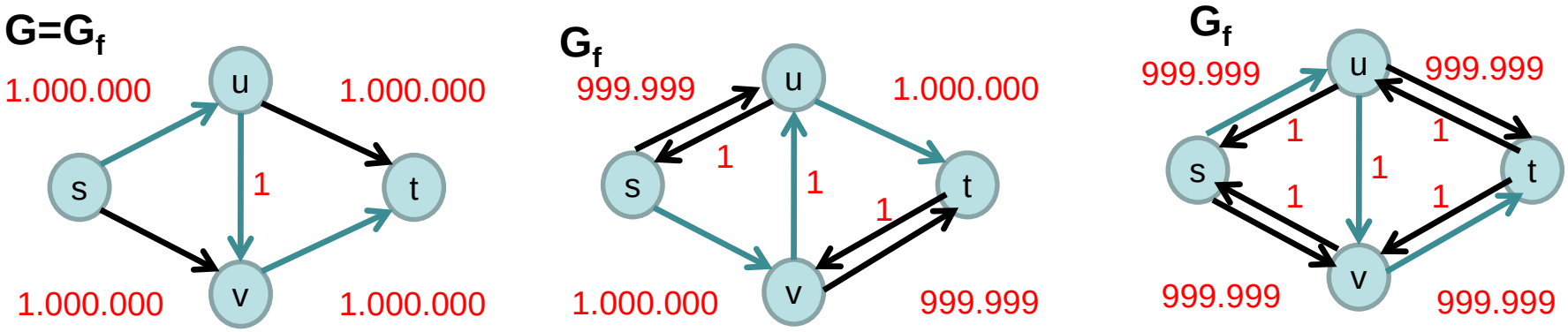
- a)⇒b): ¬b)⇒¬a) gilt wegen Korollar 8.27 und damit auch a)⇒b).
- b)⇒c): Sei S die Menge der Knoten, die von s in G_f erreichbar sind. Dann ist (S,T) mit T=V\S ein Schnitt und f(S,T)=c(S,T) nach Definition von G_f. Also ist nach Lemma 8.30 |f|=c(S,T).
- c)⇒a): Folgt aus Korollar 8.31.

Korollar 8.33: Sei (G, s, t, c) ein Flussnetzwerk mit ganzzahlige Kapazitäten c(u, v). Dann berechnet der Ford-Fulkerson Algorithmus einen maximalen Fluss f in Zeit

$$O(|E| \cdot |f|).$$

Bemerkung 6.19:

a) Die Schranke an die Laufzeit von FORDFULKERSON ist scharf:

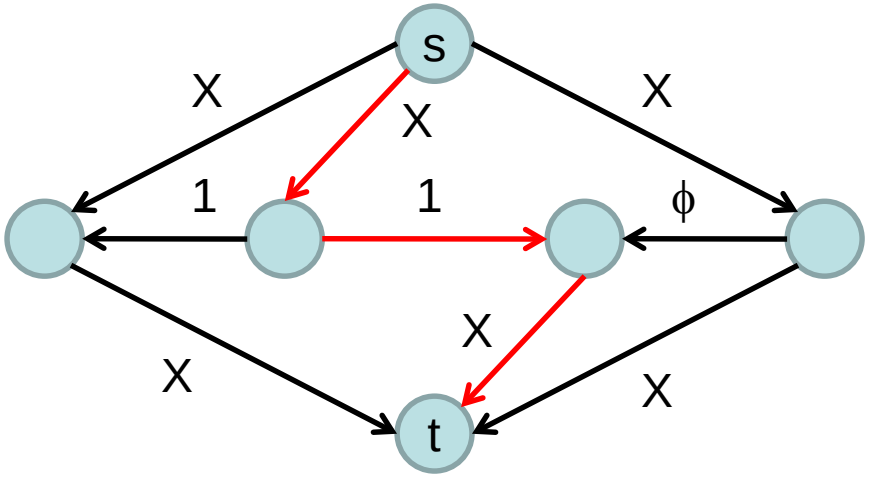


b) Wenn die Kapazitäten rationale Zahlen sind, dann kann man sie auf ganze Zahlen skalieren und die Funktion FORDFULKERSON auf das skalierte Netzwerk anwenden.

c) Wenn die Kapazitäten nicht ganzzahlig sind, dann terminiert FORDFULKERSON unter Umständen nicht, und der Fluss f, den FORDFULKERSON aktualisiert, konvertiert unter Umständen nicht gegen den maximalen Fluss.

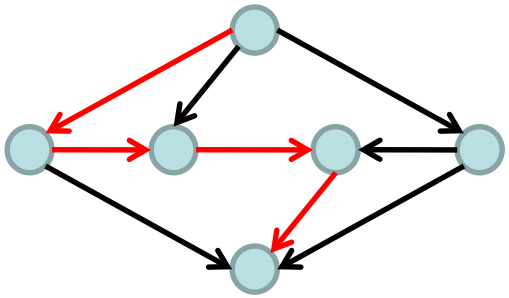
Probleme mit irrationalen Kapazitäten

Zu c): Sei $\phi=(\sqrt{5}-1)/2 \approx 0{,}618034$ so gewählt, dass $1-\phi=\phi^2$. Um zu zeigen, dass der Ford-Fulkerson Algorithmus hängen bleibt, betrachten wir den folgenden Graphen:

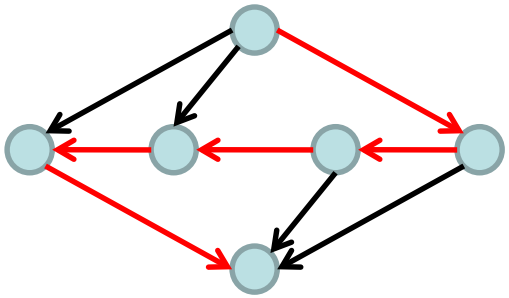


- Wir starten mit dem leeren Fluss.
- Nach Anwendung des roten Pfades sind die residualen Kapazitäten der horizontalen Kanten 1, 0 und ϕ.

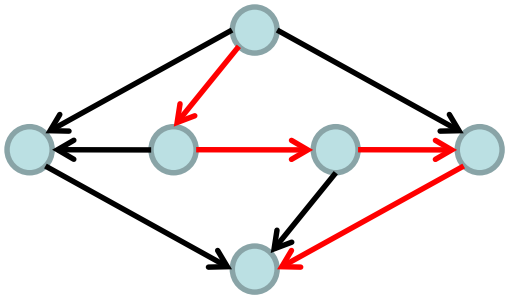
Probleme mit irrationalen Kapazitäten



A



B



C

Angenommen, die horizontalen residualen Kapazitäten sind ϕ^{k-1}, 0 und ϕ^{k} für ein ungerades $k \in \mathbb{N}$.

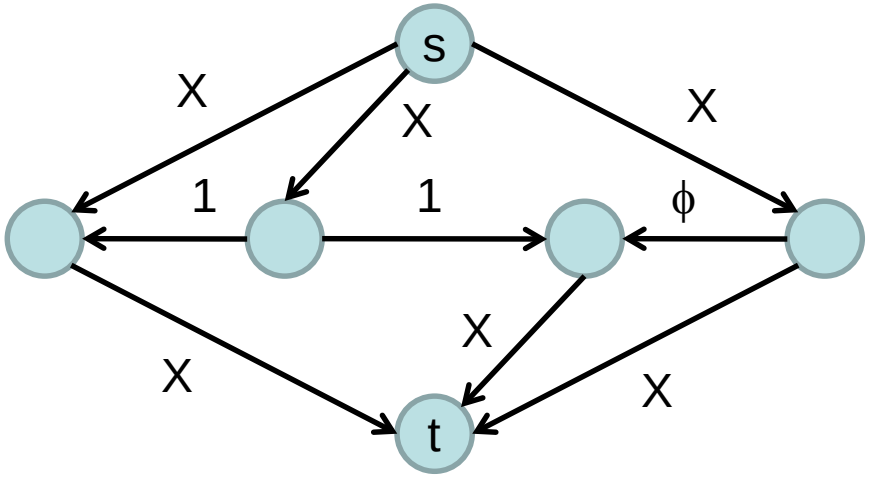
1. Augmentiere entlang B, was ϕ^{k} zum Fluss hinzufügt. Die residualen Kapazitäten sind nun ϕ^{k+1}, ϕ^{k} und 0.
2. Augmentiere entlang C, was ϕ^{k} zum Fluss hinzufügt. Die residualen Kapazitäten sind nun ϕ^{k+1}, 0 und ϕ^{k}.
3. Augmentiere entlang B, was ϕ^{k+1} zum Fluss hinzufügt. Die residualen Kapazitäten sind nun 0, ϕ^{k+1} und ϕ^{k+2}.
4. Augmentiere entlang A, was ϕ^{k+1} zum Fluss hinzufügt. Die residualen Kapazitäten sind nun ϕ^{k+1}, 0 und ϕ^{k+2}.

Probleme mit irrationalen Kapazitäten

D.h. Nach $4n+1$ Augmentierungen sind die residualen Kapazitäten ϕ^{2n-2}, 0 und ϕ^{2n-1}. Wenn die Anzahl der Augmentierungen gegen ∞ strebt, konvergiert der Wert des Flusses gegen

$$1+2\Sigma_{i\geq 0}\,\phi^i = 1+2/(1-\phi) = 4+\sqrt{5} < 7$$

obwohl der maximale Flusswert $2X+1$ ist.



Edmonds-Karp Algorithmen

Fazit: Ford-Fulkerson Algorithmus lässt zuviele Freiräume in der Wahl des augementierenden Pfades.

1972 stellten Edmonds und Karp zwei Heuristiken vor, um effizient maximale Flüsse zu bestimmen.

Heuristik 1: Wähle den augmentierenden Pfad mit dem größten Wert.

Heuristik 2: Wähle den kürzesten augmentierenden Pfad.

Edmonds-Karp Algorithmen

Theorem 8.35: Sei (G, s, t, c) ein Flussnetzwerk mit ganzzahlige Kapazitäten $c(u, v)$. Dann berechnet Heuristik 1 einen maximalen Fluss f in Zeit

$$O(|E|^2 \cdot \log |E| \cdot \log |f|).$$

Beweis:

- Sei f^* ein maximaler Fluss in G.
- Sei f ein beliebiger Fluss in G und f' ein maximaler Fluss im residualen Graph G_f. (Anfangs ist f leer und damit $f'=f^*$.)
- Sei e die Engpasskante im gewählten augmentierenden Pfad.
- $S \subseteq V$: Knoten, die von s aus mit Kanten der Kapazität $>c(e)$ erreichbar sind.
- $T=V\backslash S$: nicht leer wegen Wahl von e.
- Es gilt: $c(S,T) \leq c(e) \cdot |E|$ und $c(S,T) \geq |f|$. Also ist $c(e) \geq |f|/|E|$.
- Wert von f steigt also mindestens um Faktor $(1+1/|E|)$ an.

Edmonds-Karp Algorithmen

Theorem 8.35: Sei (G, s, t, c) ein Flussnetzwerk mit ganzzahlige Kapazitäten $c(u, v)$. Dann berechnet Heuristik 1 einen maximalen Fluss f in Zeit

$$O(|E|^2 \cdot \log |E| \cdot \log |f|).$$

Beweis (Fortsetzung):

- Wert von f steigt also mindestens um Faktor $(1+1/|E|)$ an.
- $(1+1/|E|)^k \geq |f^*|$ genau dann, wenn $k \geq |E| \ln |f^*|$.
- Es reichen also $|E| \ln |f^*|$ augmentierende Pfade, um zum maximalen Fluss zu gelangen.
- Zeit für Berechnung eines augmentierenden Pfades mit größtem Wert: $O(|E| \log |E|)$. (Das ist eine Übung.)
- Also Gesamtzeit $O(|E|^2 \cdot \log |E| \cdot \log |f|)$.

Edmonds-Karp Algorithmen

Analyse von Heuristik 2:

- G_i: residuales Netzwerk nach i. Augmentierungsschritten, d.h. $G_0=G$.
- Für Knoten v sei $dist_i(v)$ die Distanz (d.h. die Anzahl Kanten des kürzesten gerichteten Pfades) von s zu v in G_i.
- Kein Weg von s nach v: $dist_i(v)=\infty$.

Lemma 8.36: Für jeden Knoten v mit $dist_i(v)=\infty$ gilt auch $dist_{i+1}(v)=\infty$.

Edmonds-Karp Algorithmen

Lemma 8.36: Für jeden Knoten v mit $dist_i(v)=\infty$ gilt auch $dist_{i+1}(v)=\infty$.

Beweis:

- Betrachte beliebigen Knoten $v \in V$ mit $dist_i(v)=\infty$.
- U: Menge aller Knoten, die Weg zu v haben.
- Dann gilt auch für alle $u \in U$, dass $dist_i(u)=\infty$.
- Angenommen, $dist_{i+1}(v) \neq \infty$. Dann muss in Runde i ein augmentierende Pfad ausgewählt worden sein, der durch einen Knoten in U führt. (Warum?)
- In diesem Fall muss es aber einen Weg von s zu einem Knoten in U gegeben haben, ein Widerspruch!

Edmonds-Karp Algorithmen

Lemma 8.37: Für jeden Knoten $v \in V$ gilt $dist_{i+1}(v) \geq dist_i(v)$.

Beweis:

- $v=s$: trivial, da immer $dist_i(s)=0$.
- $v \neq s$: Induktion über Distanz zu s.
- $p=(s,\ldots,u,v)$: kürzester Weg von s nach v in G_{i+1}. (Kein Weg, dann sind wir nach Lemma 6.21 fertig.)
- Da dieser ein kürzester Weg ist, gilt $dist_{i+1}(u)=dist_{i+1}(v)-1$.
- Nach Induktionsvoraussetzung ist $dist_{i+1}(u) \geq dist_i(u)$.
- Fall 1: (u,v) war Kante in G_i. Dann ist $dist_i(v) \leq dist_i(u)+1$. Also ist $dist_{i+1}(v)=dist_{i+1}(u)+1 \geq dist_i(v)$.
- Fall 2: (u,v) war keine Kante in G_i. Dann war (v,u) eine Kante im i-ten augmentierenden Pfad. In dem Fall ist (v,u) auf einem kürzesten Weg von s in G_i und damit auch $dist_i(v) \leq dist_i(u)+1$.

Edmonds-Karp Algorithmen

Lemma 8.38: Während der Abarbeitung von Heuristik 2 kann jede Kante (u,v) höchstens $|V|/2$-mal vom residualen Graphen verschwinden.

Beweis:

- Angenommen, (u,v) ist in den residualen Graphen G_i und G_{j+1} aber nicht in den residualen Graphen $G_{i+1},\ldots,G_j$.
- Dann muss (u,v) im i-ten augmentierenden Pfad liegen, also $dist_i(v)=dist_i(u)+1$ sein.
- Weiterhin muss (v,u) im j-ten augmentierenden Pfad liegen, also $dist_j(u)=dist_j(v)+1$ sein.
- Zusammen mit Lemma 6.22 folgt

  $$dist_j(u) = dist_j(v)+1 \geq dist_i(v)+1 = dist_i(u)+2$$

- Da jede endliche Distanz höchstens $|V|-1$ sein kann, kann (u,v) höchstens $|V|/2$-mal verschwinden.

Edmonds-Karp Algorithmen

Jetzt sind wir soweit, eine Laufzeitschranke für Heuristik 2 zu berechnen.

- Da jede Kante höchstens $|V|/2$-mal verschwinden kann, gibt es höchstens $|E| \cdot |V|/2$ Ereignisse, in denen eine Kante verschwindet.
- Aber mindestens eine Kante verschwindet in jeder Iteration, so dass der Algorithmus nach höchstens $|E| \cdot |V|/2$ Iterationen halten muss.
- Da ein kürzester augmentierender Pfad in $O(|E|)$ Zeit gefunden werden kann, gilt:

Theorem 8.39: Sei (G, s, t, c) ein Flussnetzwerk mit ganzzahlige Kapazitäten $c(u, v)$. Dann berechnet Heuristik 2 einen maximalen Fluss in Zeit $O(|E|^2 \cdot |V|)$.

Probleme

- 10054: The Necklace
- 260: Il Gioco dell X
- 10034: Freckles
- 10278: Fire Station
- 10039: Railroads
- 314: Robot
- 10158: War
- 10249: The Grand Dinner
- 10080: Gopher II

Hausaufgabe:

- 10092: The Problem with the Problem Setter