Proseminar
Effiziente Algorithmen

Kapitel 8: Graphalgorithmen

Prof. Dr. Christian Scheideler

WS 2022

Übersicht

- Kürzeste Wege
- Minimale Spannbäume
- Matching
- Flussprobleme

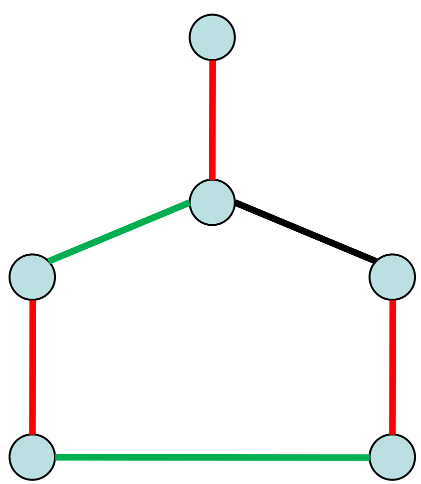
DuA

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.

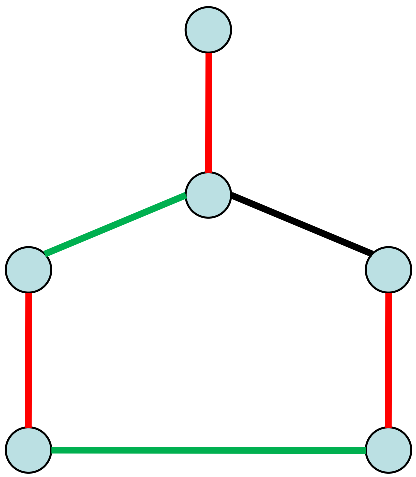


Matching:
- Variante 1
- Variante 2

Grundlagen

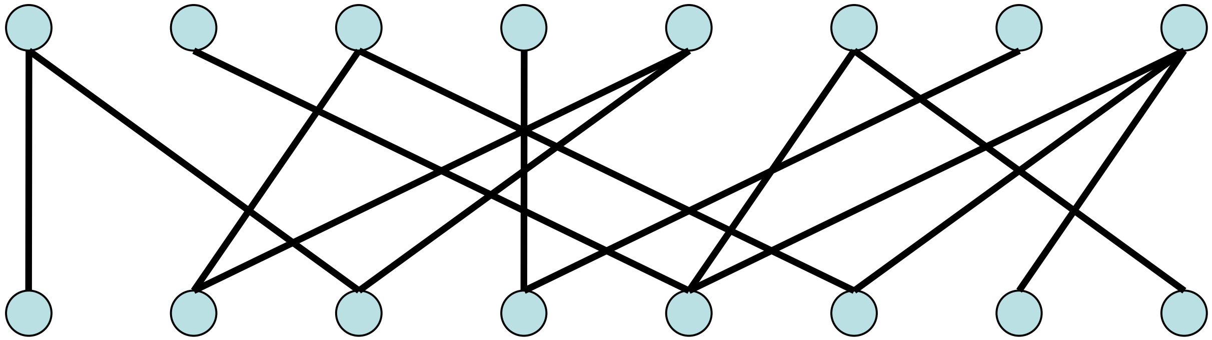
Definition 8.2:

- 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

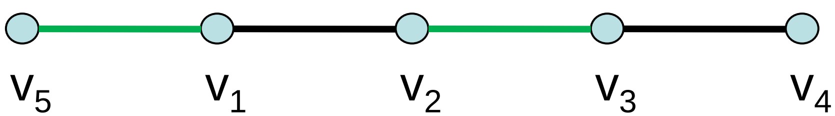
Definition 8.3: Sei $G=(V,E)$ ein ungerichteter Graph. Wenn V in zwei nichtleere Teilmengen V_1 und V_2 partitioniert werden kann (d.h. $V_1 \cup V_2 = V$ und $V_1 \cap V_2 = \emptyset$), so dass $E \subseteq V_1 \times V_2$ ist, dann heißt G bipartit (ausgedrückt durch $G=(V_1,V_2,E)$).



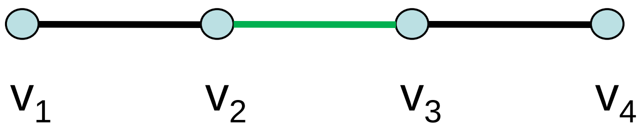
Grundlagen

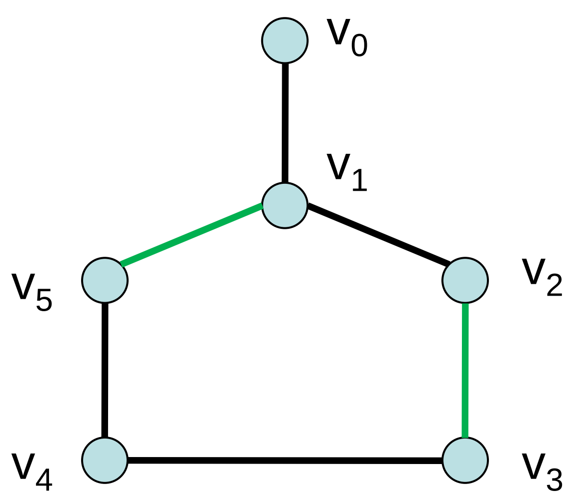
Definition 8.4: Ein einfacher Pfad (Kreis) $v_0,v_1,\ldots,v_k$ heißt alternierend bzgl. eines Matchings M, falls die Kanten $\{v_i,v_{i+1}\}$ abwechselnd in M und nicht in M liegen.

gerade Länge:



ungerade Länge:

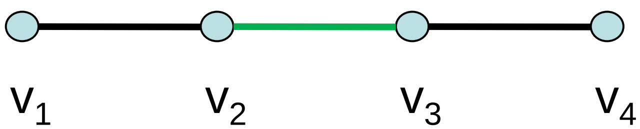




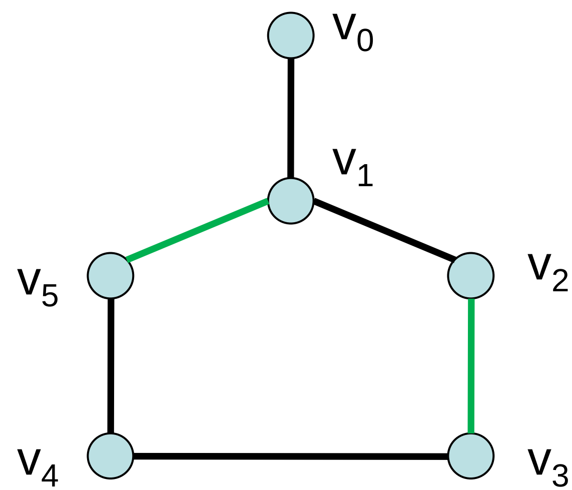
Grundlagen

Definition 8.5: 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:



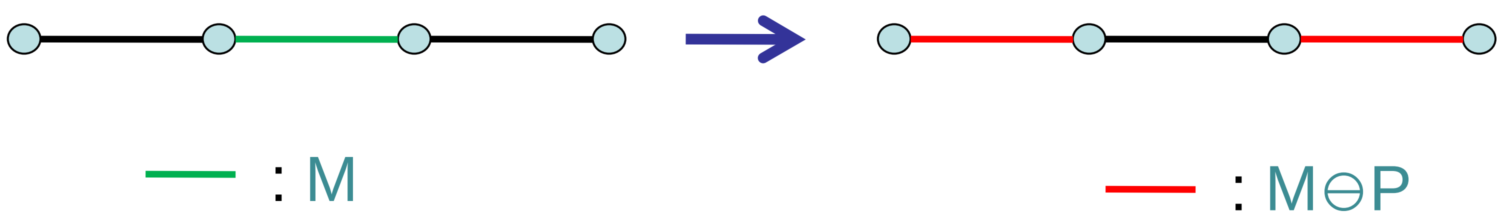
Grundlagen

Definition 8.6: 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 8.7: 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 8.7: (Satz von Berge)
Ein Matching in einem 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.

Matching in beliebigen Graphen

Algorithmus für Matching maximaler Kardinalität:

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

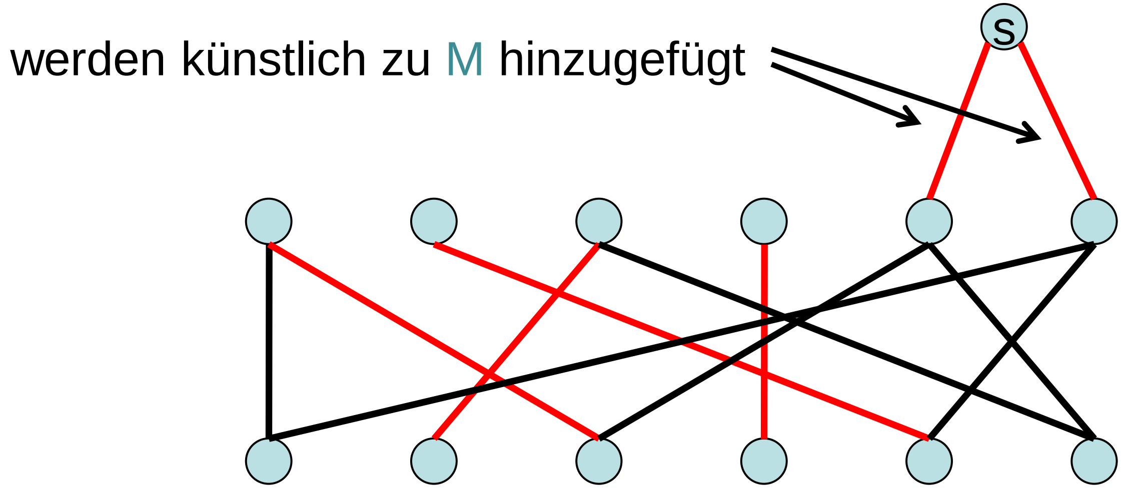
Laufzeit:

- 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.pop()    // entnehme Knoten aus Stack
    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.push(v)  // nein, dann in q einfügen
          d[Key(v)]:=d[Key(u)]+1
          parent[Key(v)]:=u
```

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 $P_1, P_2, \ldots$ die Folge der vom Algorithmus verwendeten kürzesten augmentierenden Pfade.
- Es kann gezeigt werden: $|P_{i+1}| \geq |P_i|$ für alle i.

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: Queue zu besuchender Knoten
  while q ≠<> do          // solange q nicht leer
    u:= q.dequeue()       // entnehme Knoten aus Queue
    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.enqueue(v)  // nein, dann in q einfügen
          d[Key(v)]:=d[Key(u)]+1
          parent[Key(v)]:=u
```

Kürzeste augmentierende Pfade

Weiter 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 $Q_1,\dots,Q_k$ bzgl. M, die alle Länge l haben
- $M := M \ominus Q_1 \ominus \dots \ominus Q_k$

Satz 8.8: Die obige while-Schleife wird höchstens $O(\sqrt{n})$-mal durchlaufen.

Übersicht

- Kürzeste Wege
- Minimale Spannbäume
- Matching
- Flussprobleme

Grundlagen

Definition 8.9: Ein Flussnetz (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 8.10: 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 8.11: 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 $f(u, v) = f(v, u) = 0$.

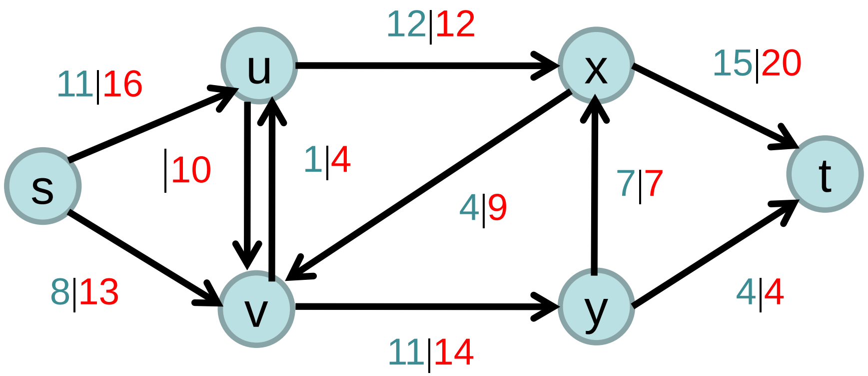
d) Für alle $v \in V \setminus \{s, t\}$ gilt

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

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 8.12: 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 8.13: 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 $|f|$.

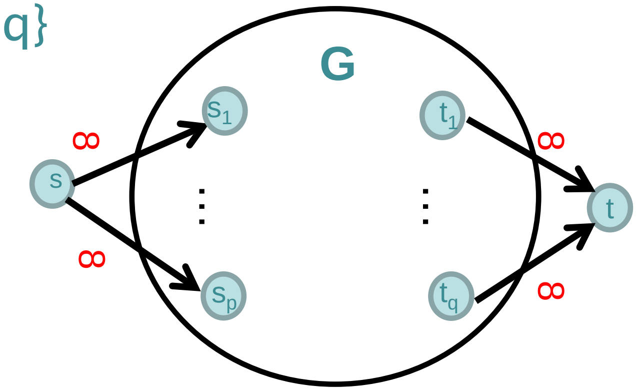
Bemerkung 8.14: Ein maximales Flussproblem $(G, s_1, \ldots, s_p, t_1, \ldots t_q, c)$ mit mehreren Quellen $s_1, \ldots, s_p$ und mehreren Senken $t_1, \ldots t_q$ mit dem Ziel, so viele Güter wie möglich von den Quellen zu den Senken zu transportieren, kann einfach auf das original MAXFLOW-Problem reduziert werden:

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

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

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

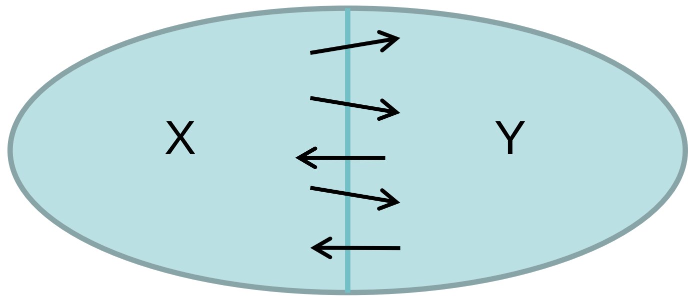
$$c'(u, v) = \begin{cases} c(u, v) & u, v \in V \\ \infty & u = s \ \text{ oder } \ v = t \end{cases}$$



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 8.15: 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 8.16: 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

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

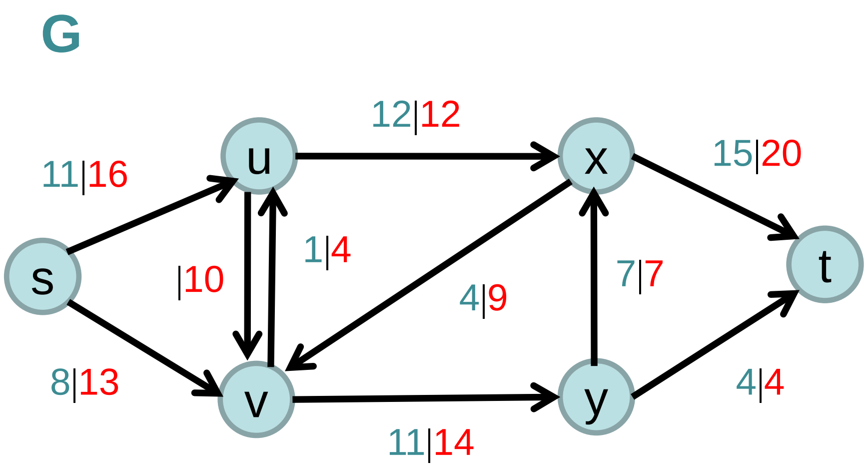
c) Ein einfacher Pfad P von s zu t in G_f wird augmentierender Pfad genannt. Die Restkapazität $c_f\,(P)$ von P ist definiert als

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

Die Ford-Fulkerson Methode

Beispiel: augmentierender Pfad und Flussaugmentierung

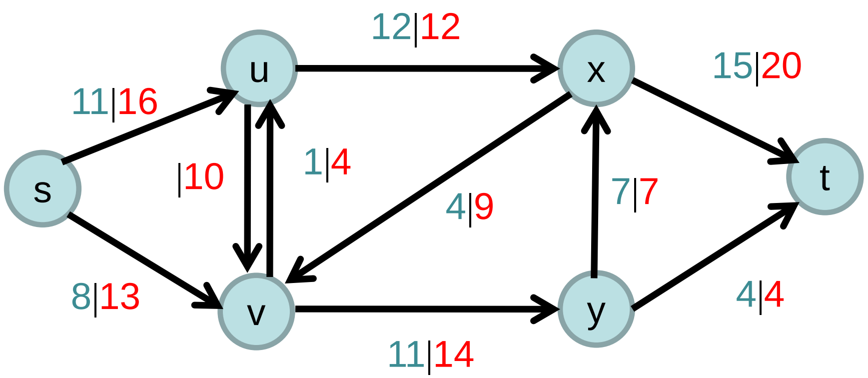
Flussnetzwerk:



Beispiel: augmentierender Pfad und Flussaugmentierung

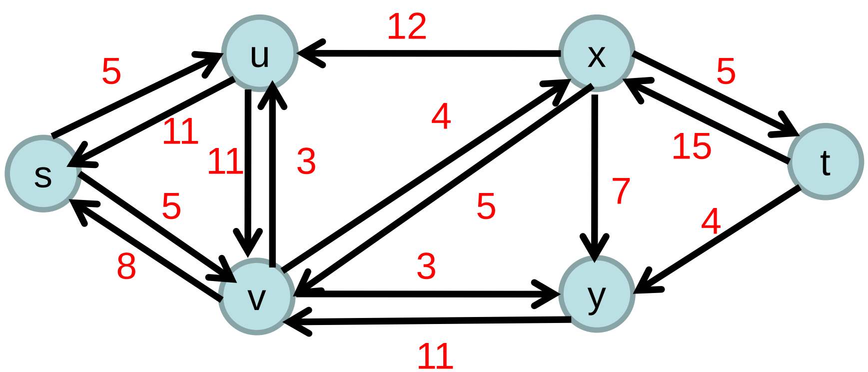
Flussnetzwerk:

G



Residualnetzwerk:

G_f

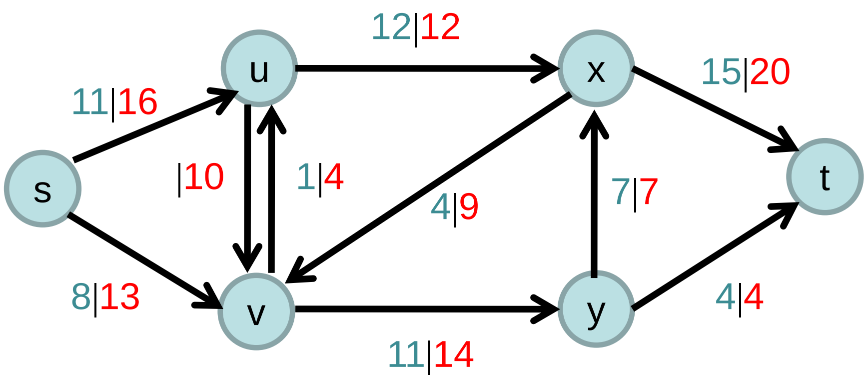


Beispiel: augmentierender Pfad und Flussaugmentierung

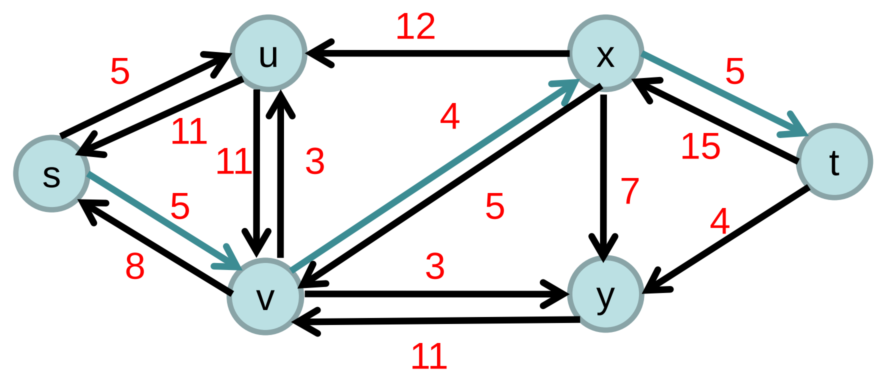
Flussnetzwerk:

Residualnetzwerk mit augmentierendem Pfad:

G

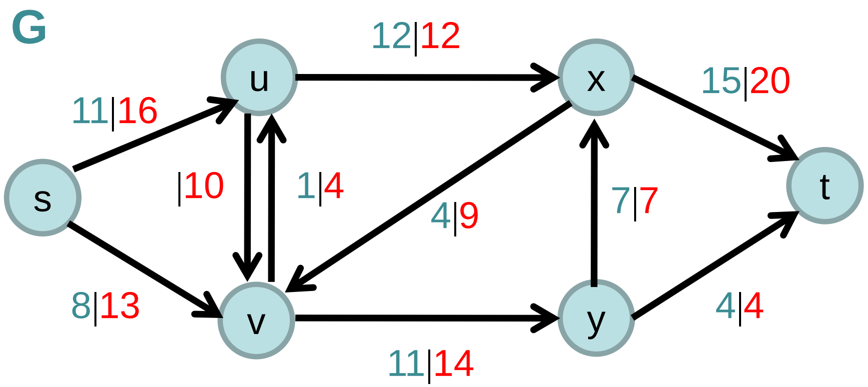


G$_f$

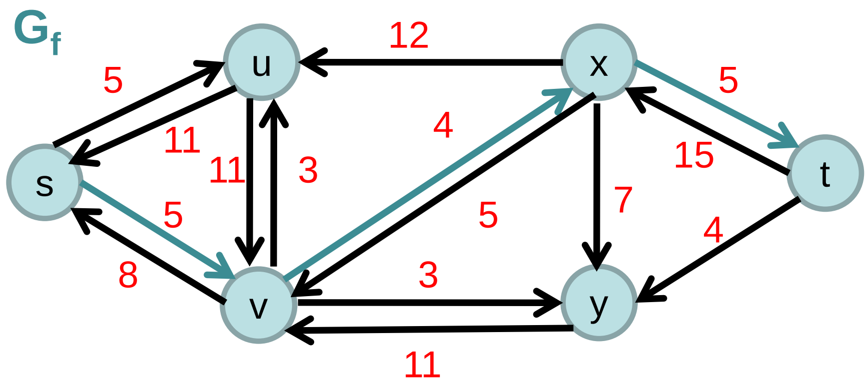


Beispiel: augmentierender Pfad und Flussaugmentierung

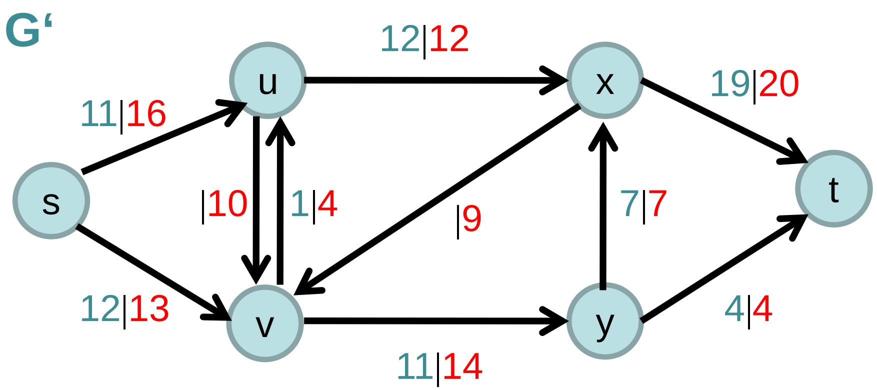
Flussnetzwerk:



Residualnetzwerk mit augmentierendem Pfad:

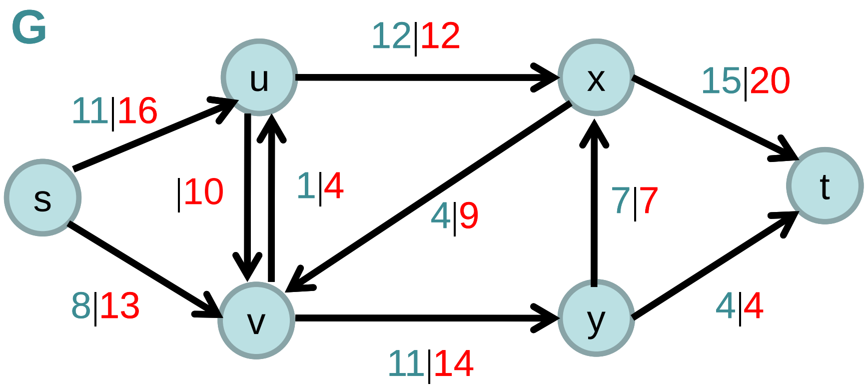


Augmentiertes Flussnetzwerk:

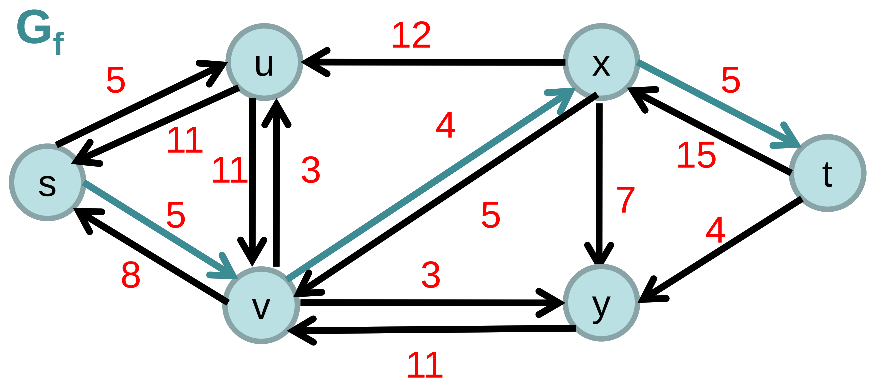


Beispiel: augmentierender Pfad und Flussaugmentierung

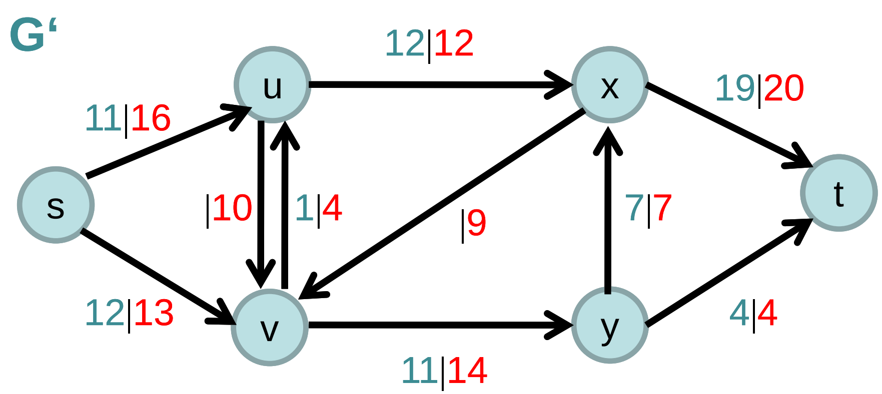
Flussnetzwerk:



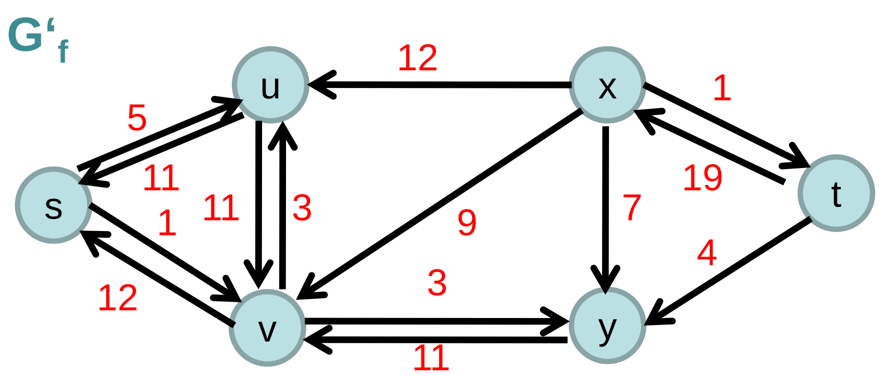
Residualnetzwerk mit augmentierendem Pfad:



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
   Gf = Residualnetzwerk von G  bezüglich f;
   solange (∃ ein Pfad P von s zu t in Gf)                 // P ist der augmentierende Pfad
   {  // berechne maximal hinzufügbaren Fluss via P
      cf (P) := min {cf (u, v) | (u, v) ∈ P)};             // cf (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] + cf (P); f [v, u] := - f [u, v]; }
       Gf := Residualnetzwerk von G bezüglich f;
   }
   gib f aus
}
```

Lemma 8.17: 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 8.18: 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

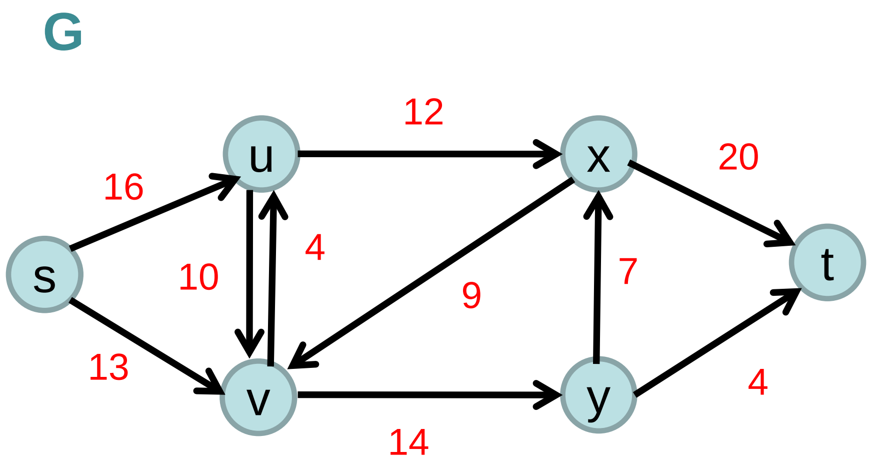
$$f_P(u,v) = \begin{cases} c_f(P) & \text{wenn } (u,v) \text{ auf } P \\ -c_f(P) & \text{wenn } (v,u) \text{ auf } P \\ 0 & \text{sonst} \end{cases}$$

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

Korollar 8.19: 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

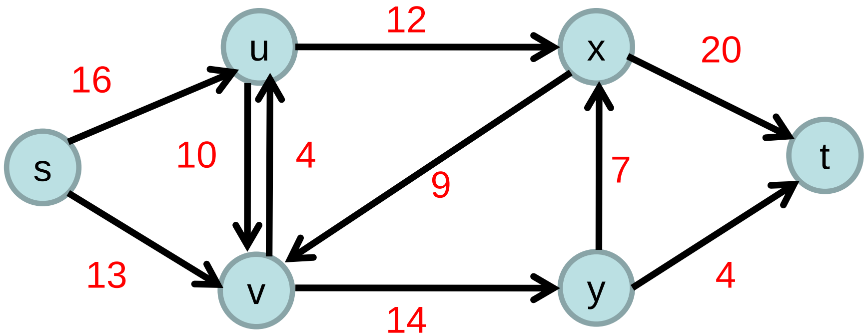
Flussnetzwerk:



Beispiel: Ford-Fulkerson Algorithmus

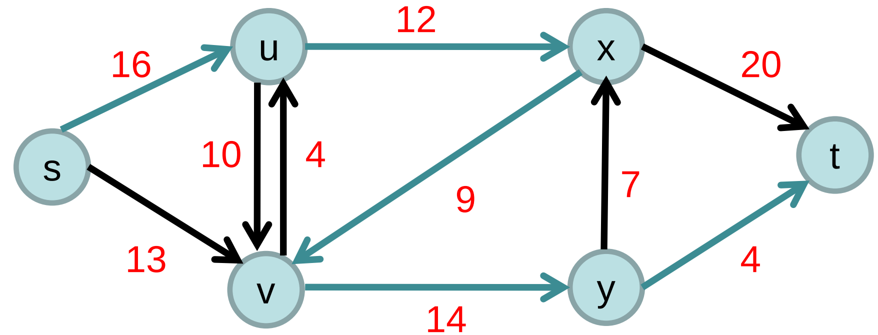
Flussnetzwerk:

G



Residualnetzwerk mit augmentierendem Pfad:

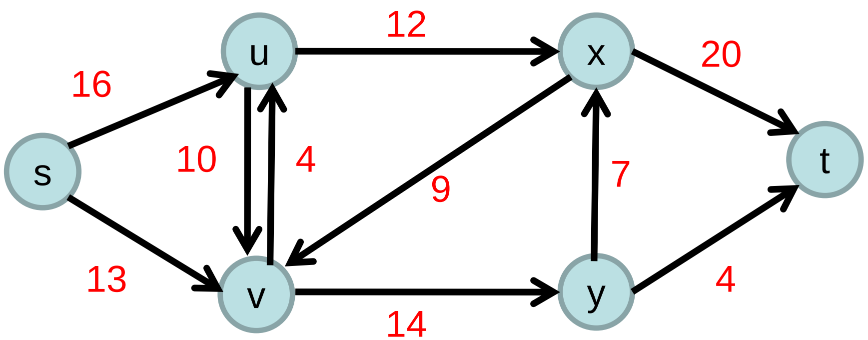
G_f



Beispiel: Ford-Fulkerson Algorithmus

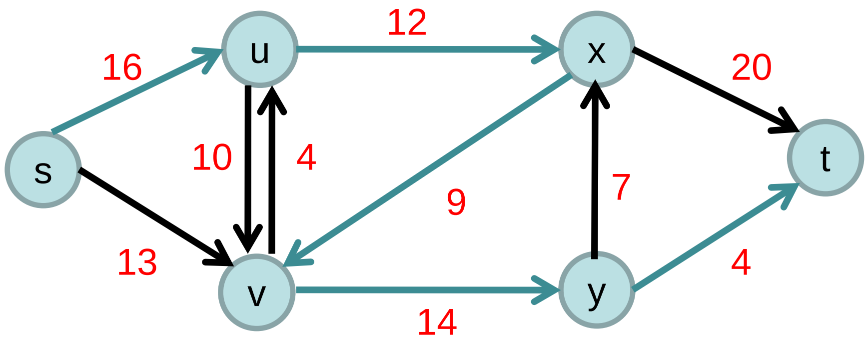
Flussnetzwerk:

G



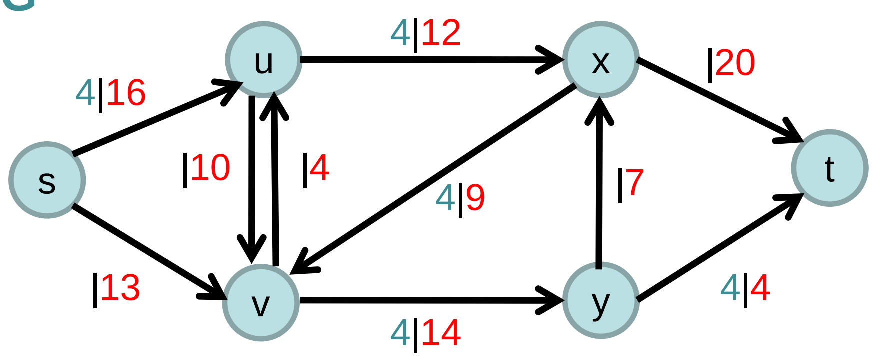
Residualnetzwerk mit augmentierendem Pfad:

G_f



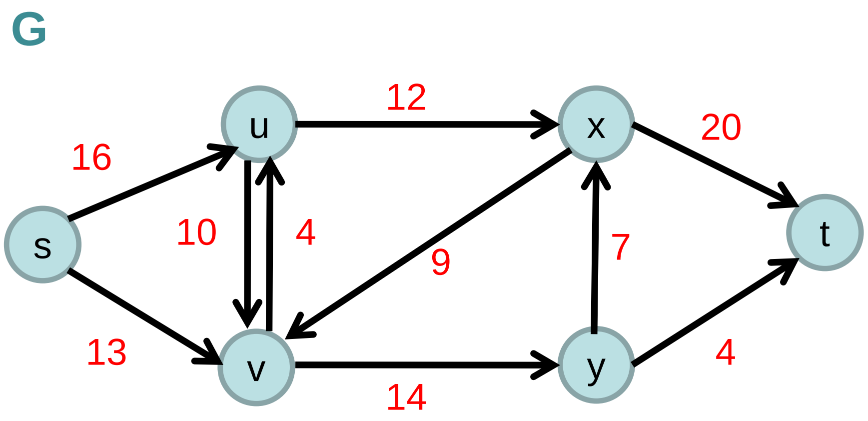
Augmentiertes Flussnetzwerk:

G

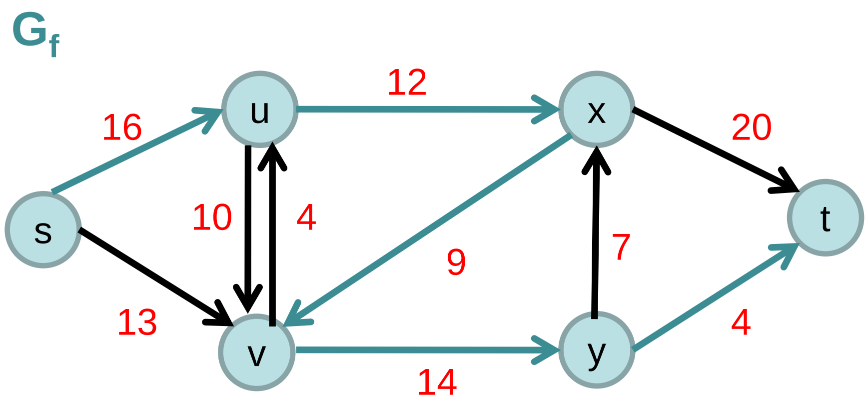


Beispiel: Ford-Fulkerson Algorithmus

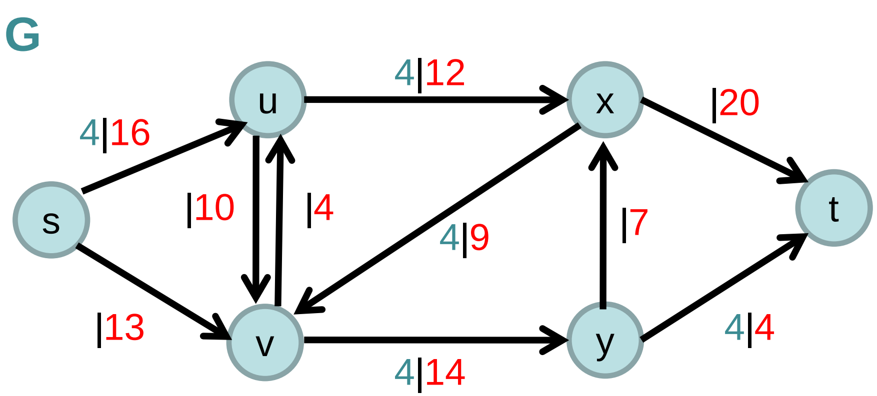
Flussnetzwerk:



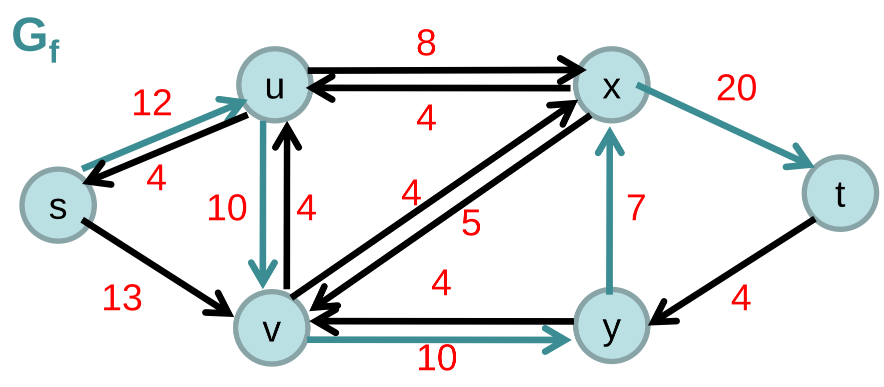
Residualnetzwerk mit augmentierendem Pfad:



Augmentiertes Flussnetzwerk:

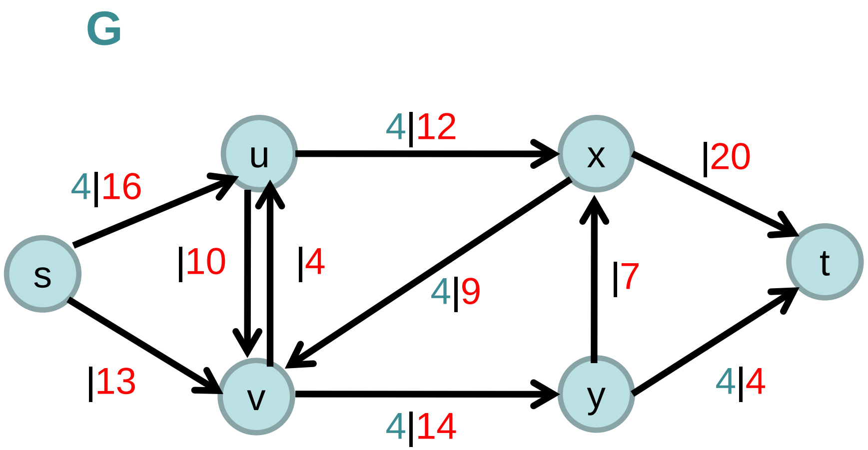


Neues Residualnetzwerk mit augmentierendem Pfad:

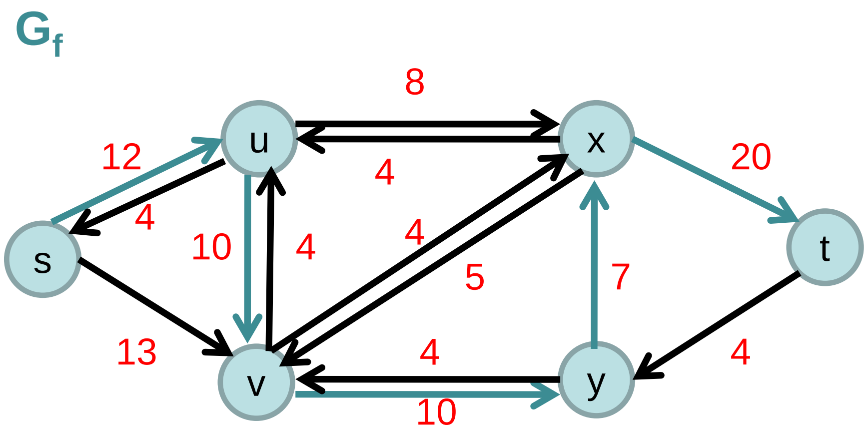


Beispiel: Ford-Fulkerson Algorithmus

Flussnetzwerk:



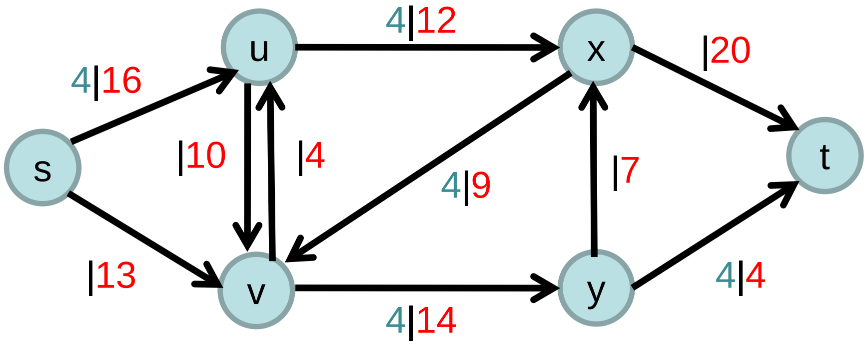
Residualnetzwerk mit augmentierendem Pfad:



Beispiel: Ford-Fulkerson Algorithmus

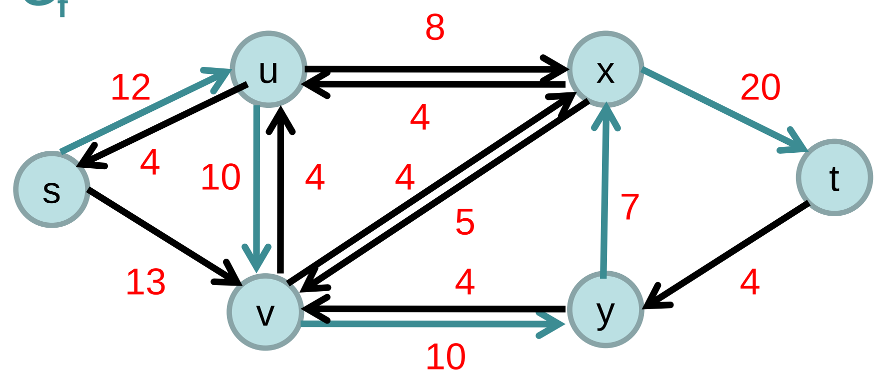
Flussnetzwerk:

G



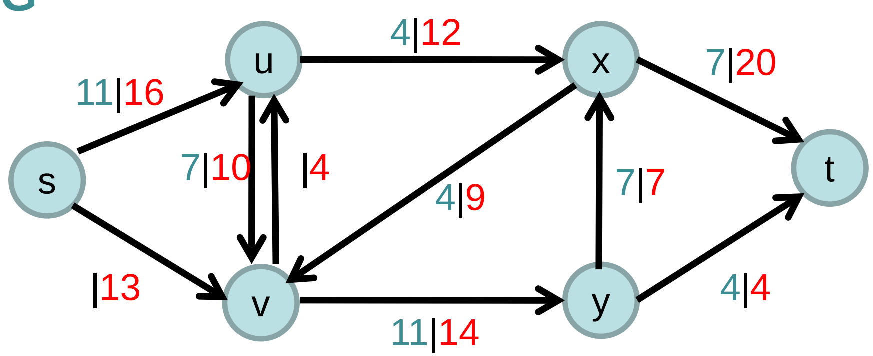
Residualnetzwerk mit augmentierendem Pfad:

G_f



Augmentiertes Flussnetzwerk:

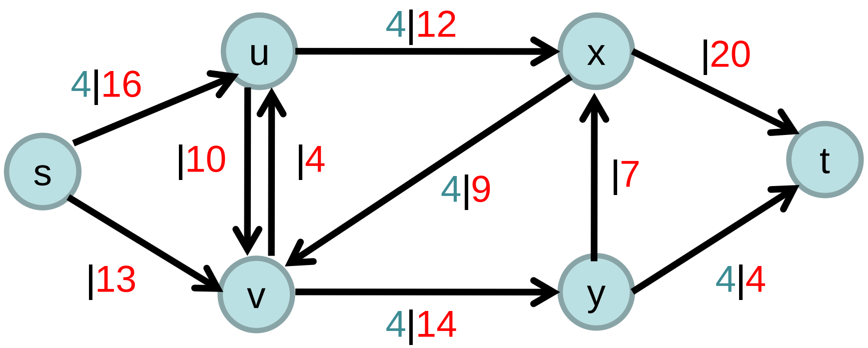
G



Beispiel: Ford-Fulkerson Algorithmus

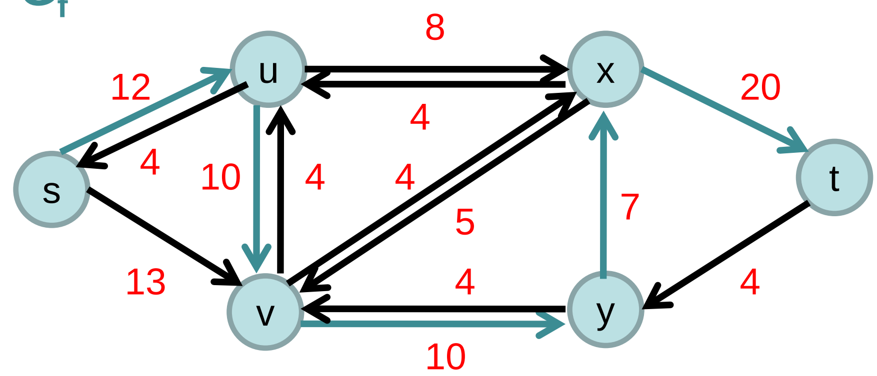
Flussnetzwerk:

G



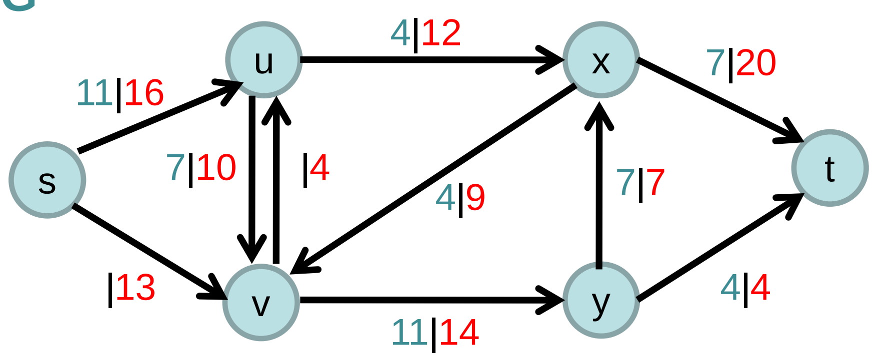
Residualnetzwerk mit augmentierendem Pfad:

G_f



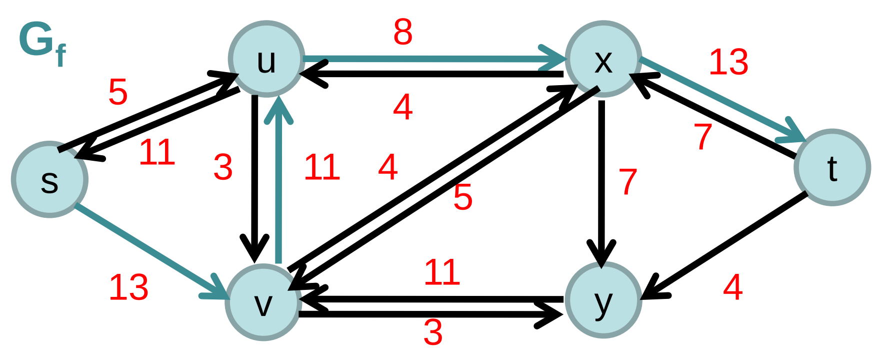
Augmentiertes Flussnetzwerk:

G



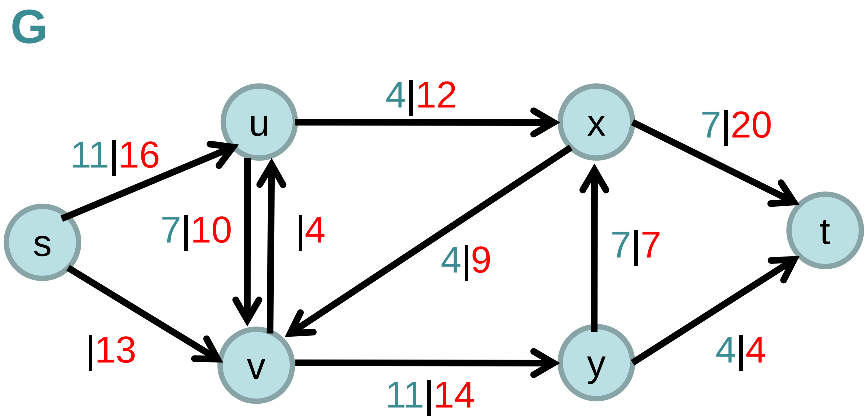
Neues Residualnetzwerk mit augmentierendem Pfad:

G_f

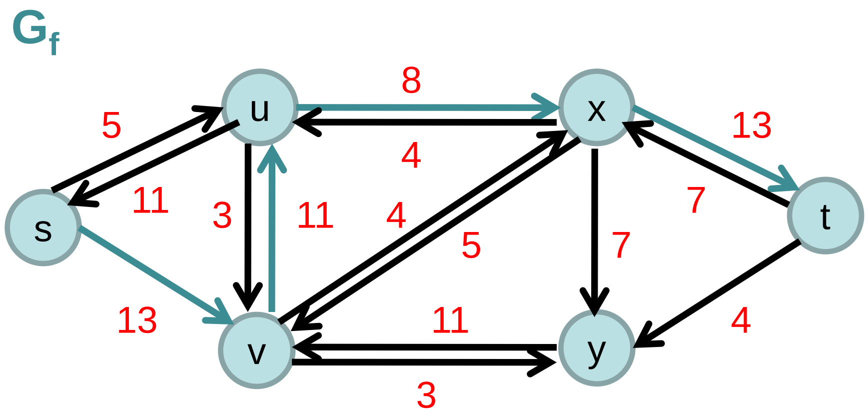


Beispiel: Ford-Fulkerson Algorithmus

Flussnetzwerk:

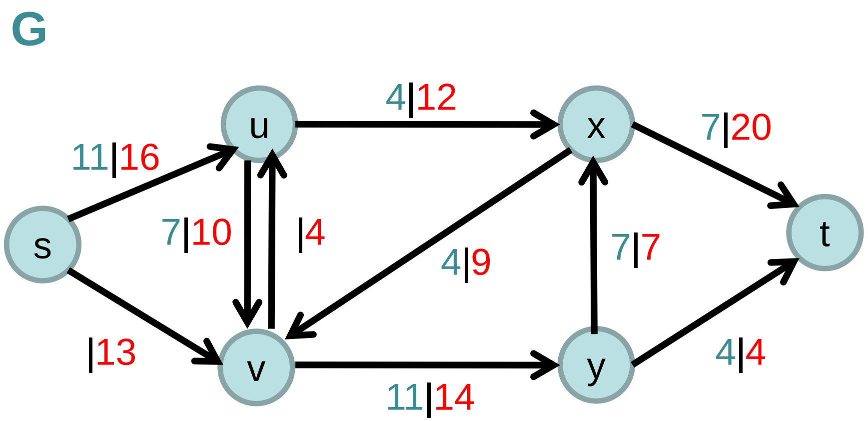


Residualnetzwerk mit augmentierendem Pfad:

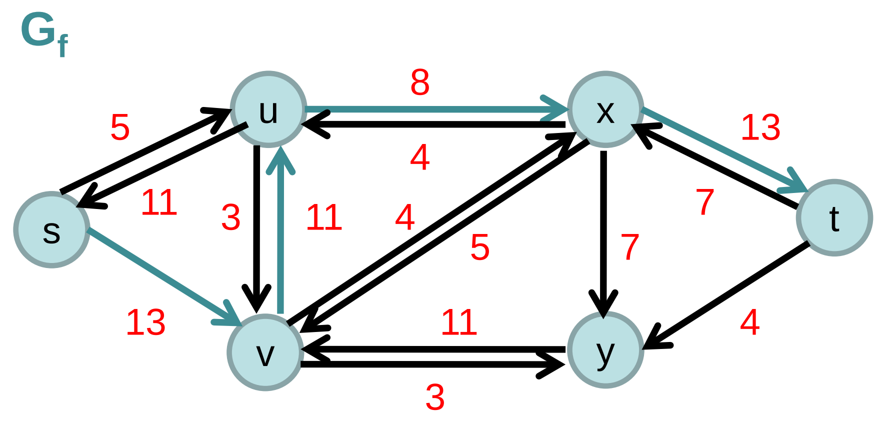


Beispiel: Ford-Fulkerson Algorithmus

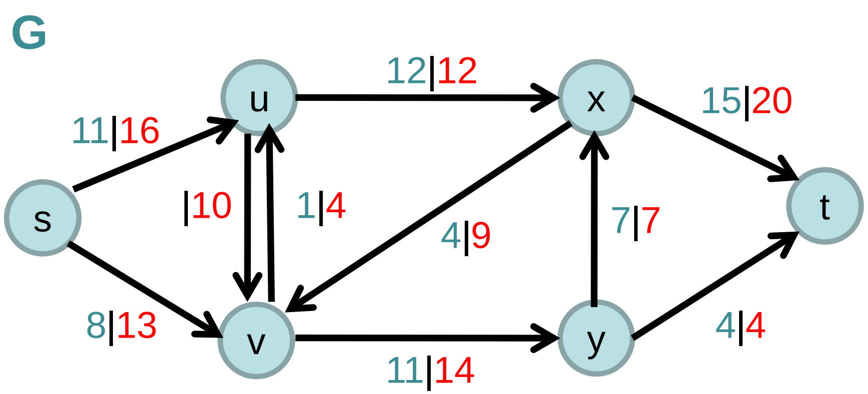
Flussnetzwerk:



Residualnetzwerk mit augmentierendem Pfad:



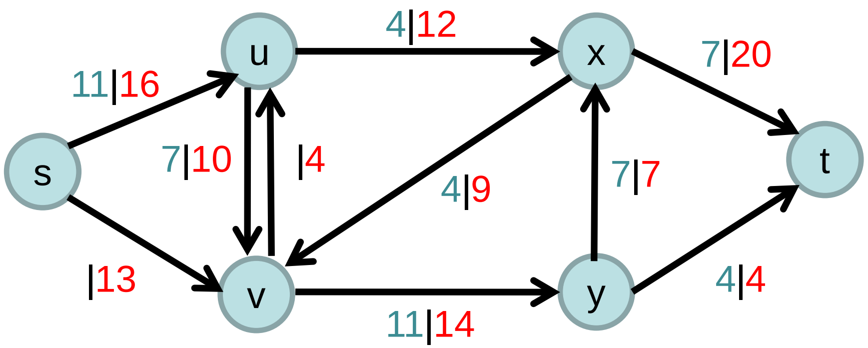
Augmentiertes Flussnetzwerk:



Beispiel: Ford-Fulkerson Algorithmus

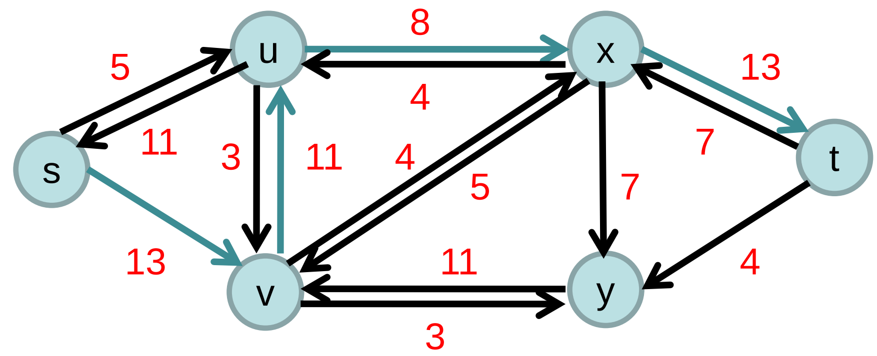
Flussnetzwerk:

G



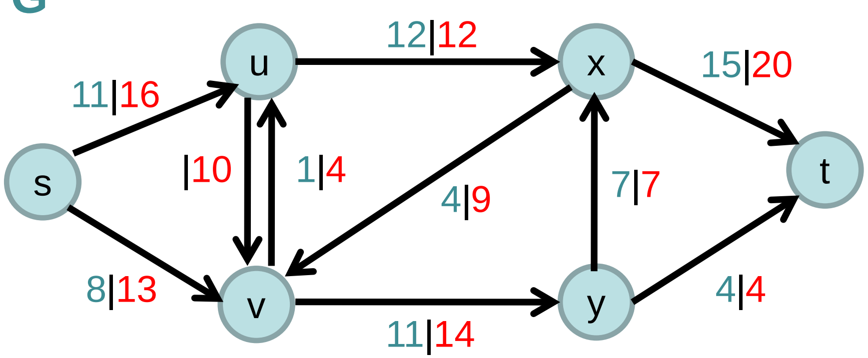
Residualnetzwerk mit augmentierendem Pfad:

G_f



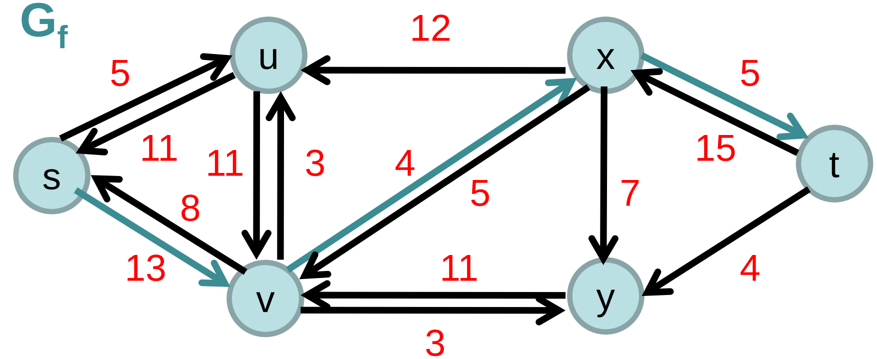
Augmentiertes Flussnetzwerk:

G



Neues Residualnetzwerk mit augmentierendem Pfad:

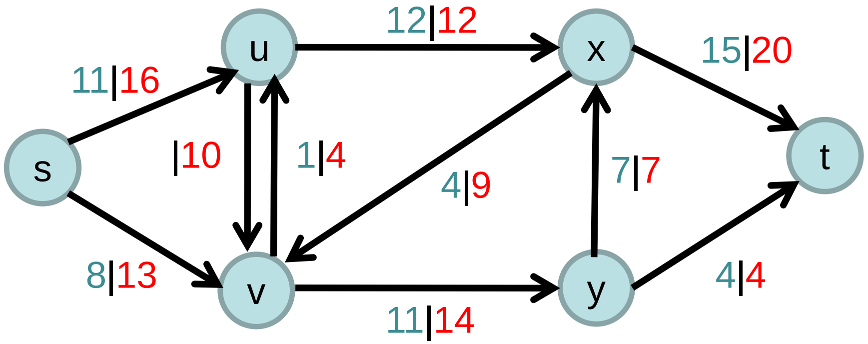
G_f



Beispiel: Ford-Fulkerson Algorithmus

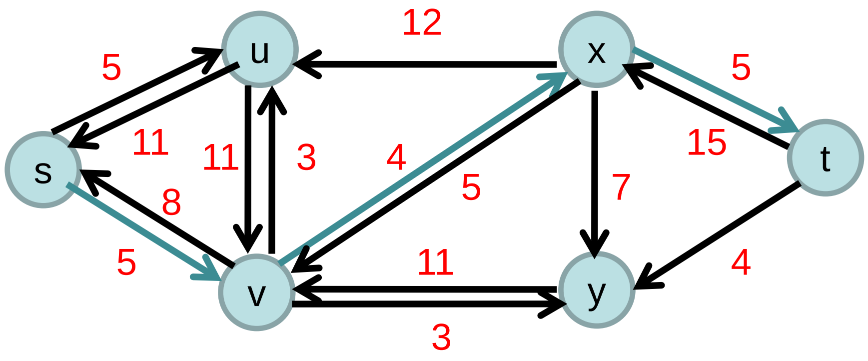
Flussnetzwerk:

G



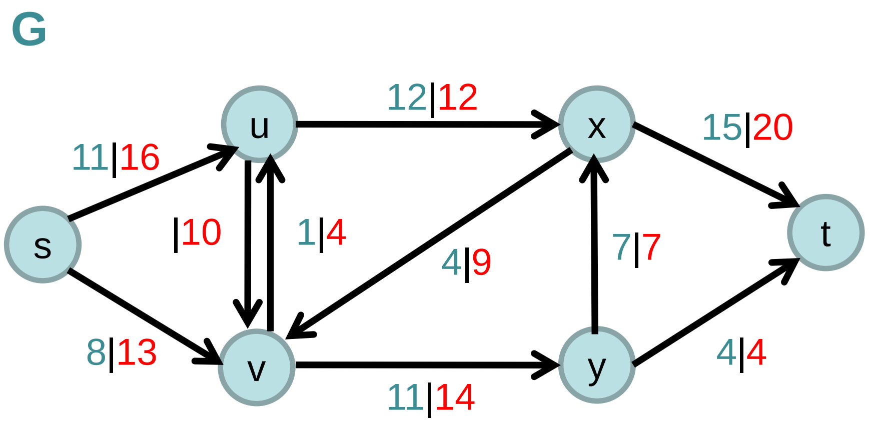
Residualnetzwerk mit augmentierendem Pfad:

G_f

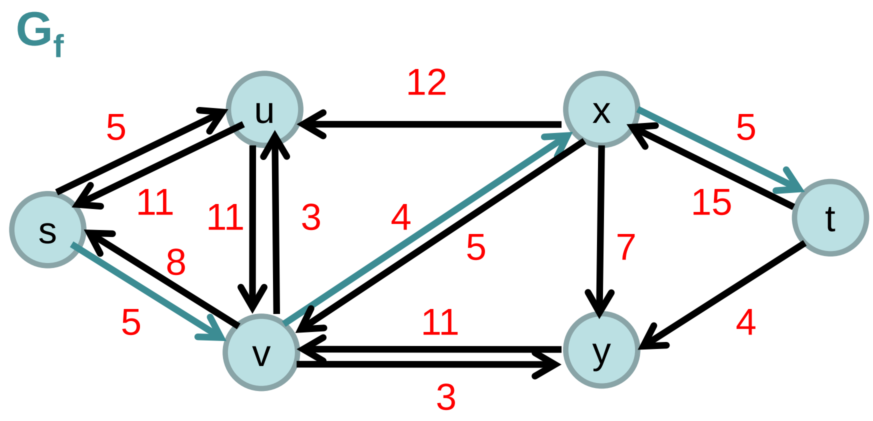


Beispiel: Ford-Fulkerson Algorithmus

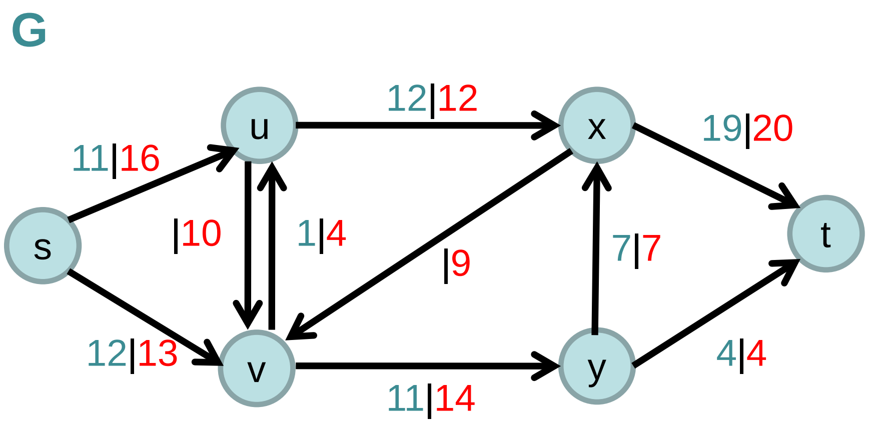
Flussnetzwerk:



Residualnetzwerk mit augmentierendem Pfad:



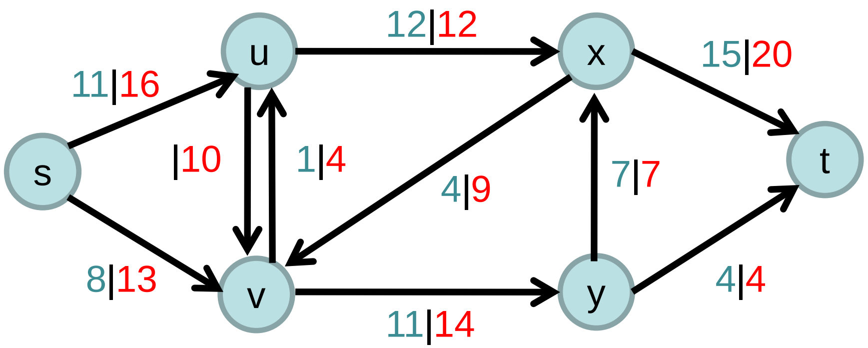
Augmentiertes Flussnetzwerk:



Beispiel: Ford-Fulkerson Algorithmus

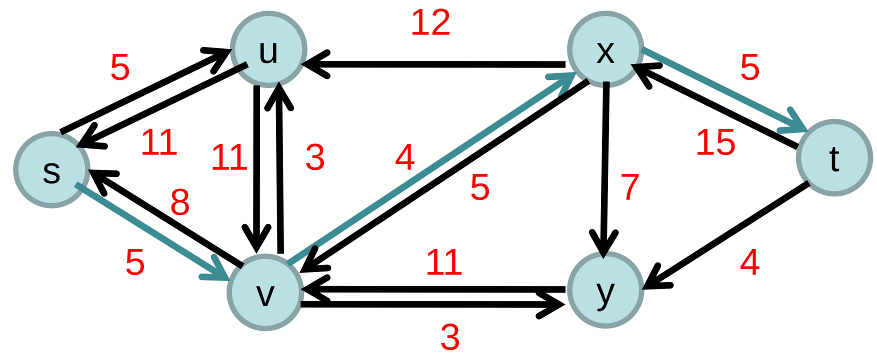
Flussnetzwerk:

G



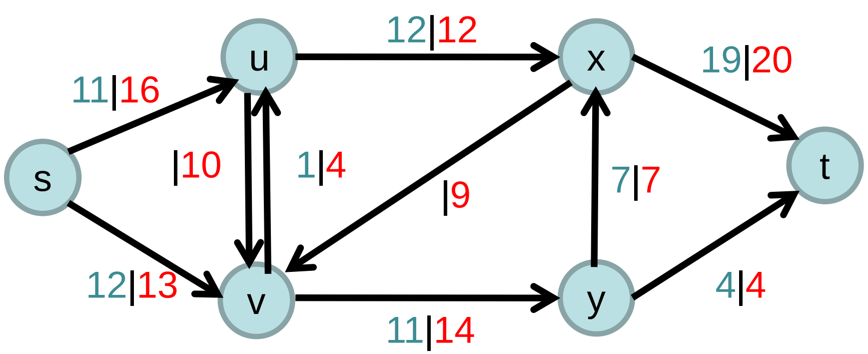
Residualnetzwerk mit augmentierendem Pfad:

G_f



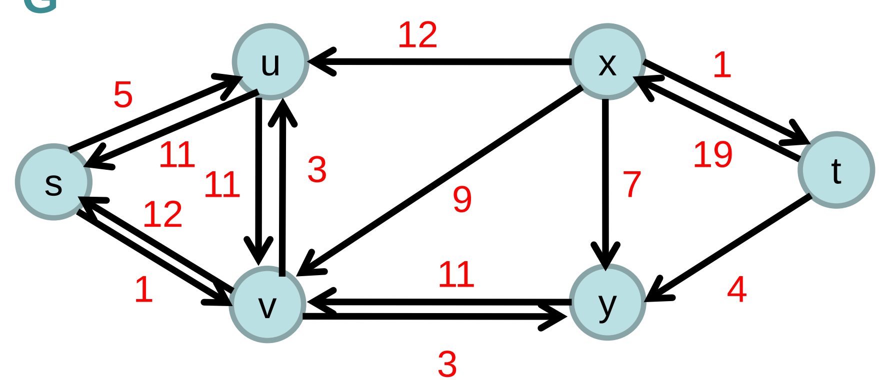
Augmentiertes Flussnetzwerk:

G

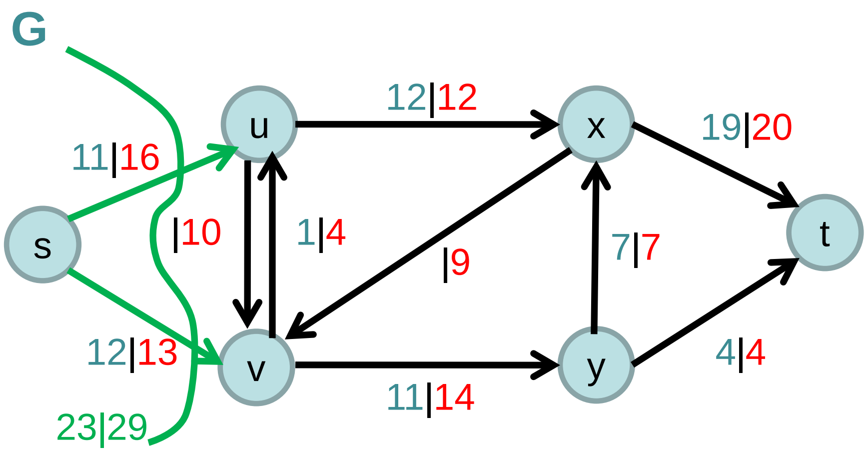


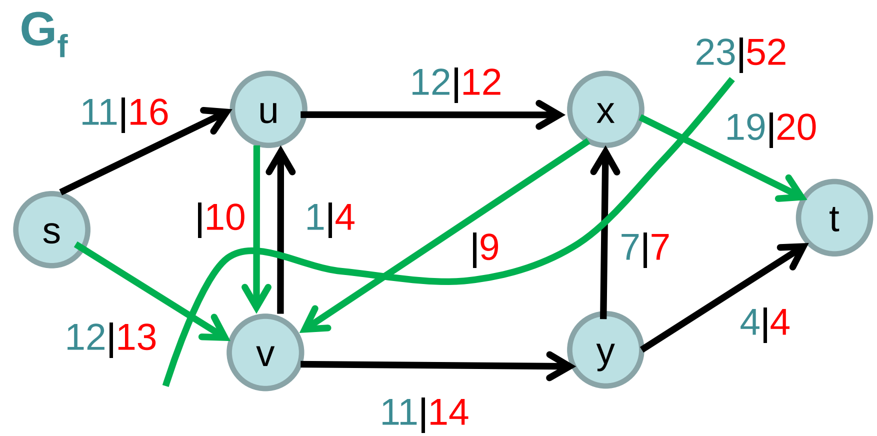
Neues Residualnetzwerk mit augmentierendem Pfad:

G

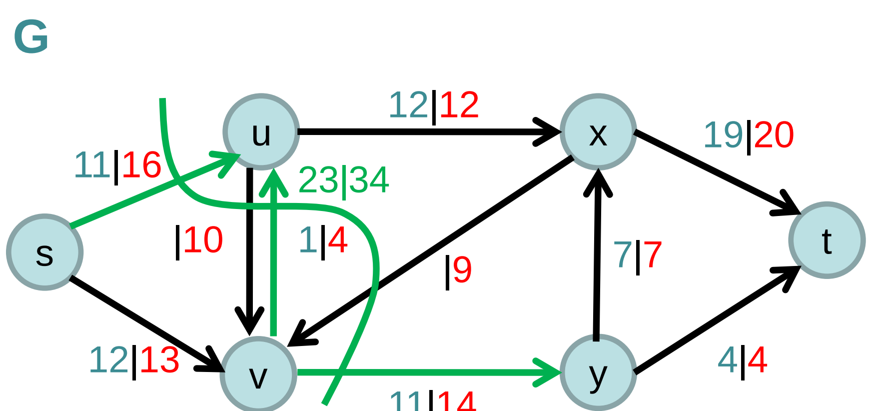


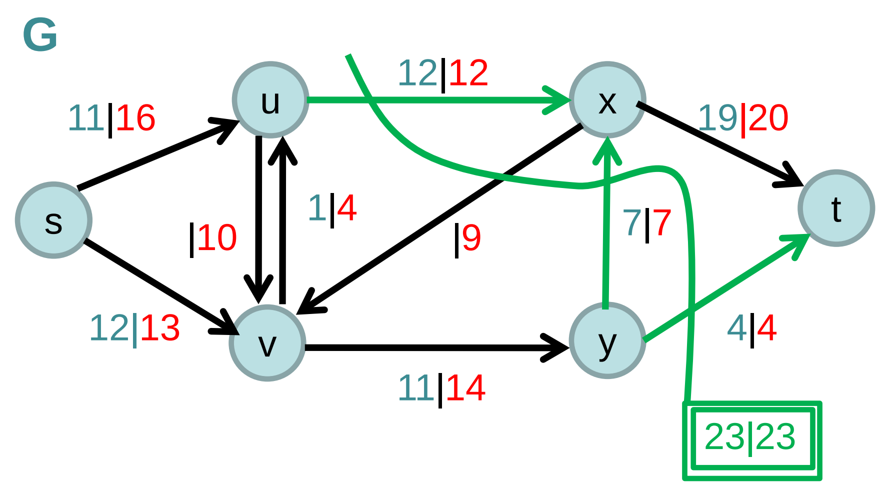
Flussnetzwerk:





Augmentiertes Flussnetzwerk:





Edmonds-Karp Algorithmen

Problem: 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 augmentie-renden Pfad.

Probleme

- 260: Il Gioco dell X
- 10054: The Necklace
- 10034: Freckles
- 318: Domino Effect
- 10080: Gopher II
- 10249: The Grand Dinner

Hausaufgabe:

- 314: Robot