Premaster Course Algorithms 1

Chapter 7: Network Flow

Christian Scheideler

SS 2019

Network Flow

Overview:

- Foundations
- Ford-Fulkerson algorithm
- Edmonds-Karp algorithm
- Goldberg's algorithm

Foundations

Definition 1: A flow network (G,s,t,c) consists of a directed graph $G=(V,E)$, a source $s \in V$, a sink $t \in V$, and a capacity function $c: V \times V \to \mathbb{R}_{\geq 0}$, with $c(u,v) = 0$ if $(u,v) \notin E$.

In the following, we assume that $s \rightsquigarrow_G u \rightsquigarrow_G t$ for all $u \in V$, where $u \rightsquigarrow_G v$ means that there is a directed path from u to v in G. (Otherwise, we can remove u and all of its edges from G, because a flow from s to t cannot be sent via u.)

Definition 2: Let (G,s,t,c) be a flow network.

a) A network flow in G is a function $f: V \times V \to \mathbb{R}$ with the property that

$f(u, v) \leq c(u, v)$ for all $u, v \in V$	(capacity constraints)
$f(u, v) = -f(v, u)$ for all $u, v \in V$	(skew symmetry)
$\sum_{v \in V} f(u, v) = 0$ for all $u \in V \setminus \{s, t\}$	(flow conservation)

b) The value $|f|$ of a network flow f is defined as
$|f| = \sum_{v \in V} f(s, v)$.

Foundations

A network flow in G is a function $f: V \times V \to \mathbb{R}$ with the property that

$f(u, v) \le c(u, v)$ for all $u, v \in V$ (capacity constraints)

$f(u, v) = -f(v, u)$ for all $u, v \in V$ (skew symmetry)

$\Sigma_{v \in V} f(u, v) = 0$ for all $u \in V \setminus \{s, t\}$ (flow conservation)

Remark 3: Let f be a flow in a flow network (G,s,t,c). Then

a) $f(v, v) = 0$ for all $v \in V$ (due to skew symmetry).

b) $\Sigma_{u \in V} f(u, v) = 0$ for all $v \in V \setminus \{s, t\}$ (flow conservation & skew symmetry).

c) For all $u, v \in V$ with $(u, v), (v, u) \notin E$ it holds that $f(u, v) = f(v, u) = 0$.

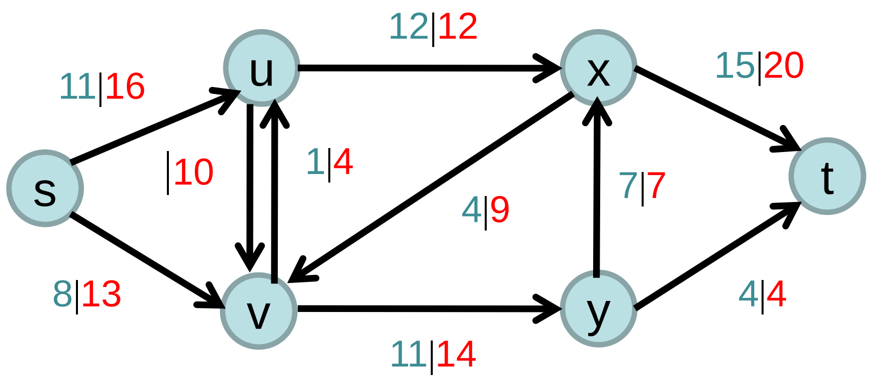
d) For all $v \in V \setminus \{s, t\}$,

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

e) A function f with $f(u, v) = 0$ for all $u, v \in V$ is a valid flow.

Foundations

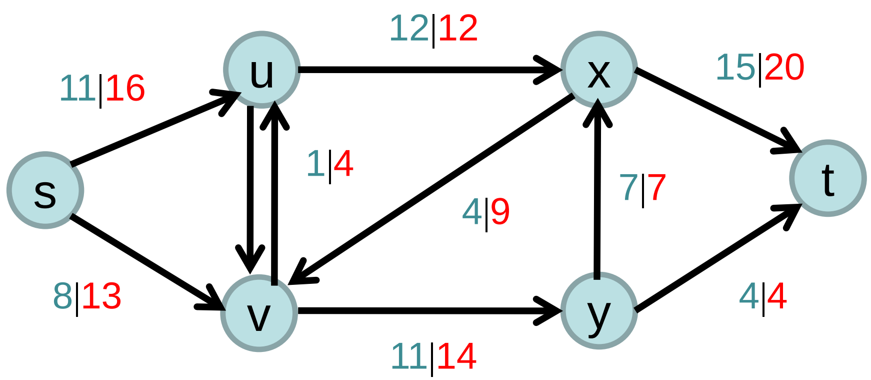
Example of a valid flow:



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

- Only positive flows are shown (negative flows are implied by skew symmetry).
- For example, f(v,u)=1, so f(u,v)=-1.
- This implies that flow cannot flow at the same time in both directions for a pair {u,v}.
- Why is it fine to have that restriction? (Concretely, why can we ignore instances having positive flows in both directions between u and v without loss of generality, when just focusing on |f|?)

Foundations



Remark 4: The outgoing flow of s is equal to the incoming flow at t.

Proof:

- It follows from skew symmetry:

$$\Sigma_{v\in V} \Sigma_{w\in V} f(v,w) = \Sigma_{\{v,w\}} (f(v,w)+f(w,v)) + \Sigma_{v\in V} f(v,v) = 0$$

- Moreover, it follows from flow conservation:

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

- Hence, due to skew symmetry:

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

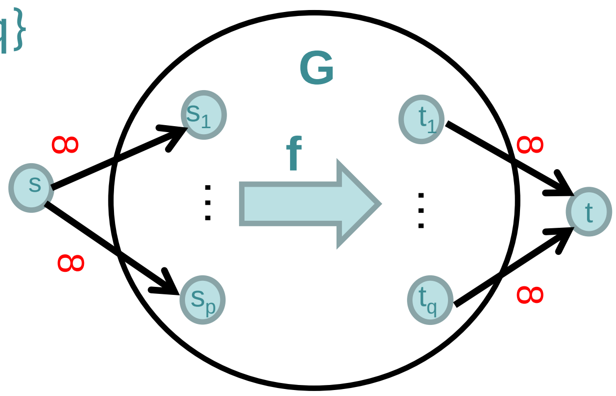
MAXFLOW Problem:

Input: a flow network (G,s,t,c).
Output: a flow f in G with maximum value $|f|$.

Remark 5: A maxflow problem $(G, s_1, \ldots, s_p, t_1, \ldots t_q, c)$ with multiple sources $s_1, \ldots, s_p$ and multiple sinks $t_1, \ldots t_q$ with the goal to transfer as much flow as possible from the sources to the sinks (i.e., find a flow $f: V \times V \to \mathbb{R}$ maximizing $\Sigma_{i=1}^{p} (\Sigma_{v \in V} f(s_i,v))$) can be reduced to the original maxflow problem:

Construct $G' = (V', E')$ and c' as follows:
$V' = V \cup \{s, t\}$
$E' = E \cup \{(s, s_i) \mid 1 \le i \le p\} \cup \{(t_i, t) \mid 1 \le i \le q\}$

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



Then there is a flow f from $s_1, \ldots, s_p$ to $t_1, \ldots, t_q$ of value φ in $(G, s_1, \ldots, s_p, t_1, \ldots t_q, c)$ if and only if there is a flow f' from s to t in (G', s, t, c') of value φ (see the figure).

Ford-Fulkerson Algorithm

How do we solve the maxflow problem?

Definition 6: Let (G,s,t,c) be a flow network and f be a flow in G.
a) For any $u, v \in V$, the residual capacity $c_f(u,v)$ is defined as

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

b) The residual network $G_f = (V,E_f)$ is defined as

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

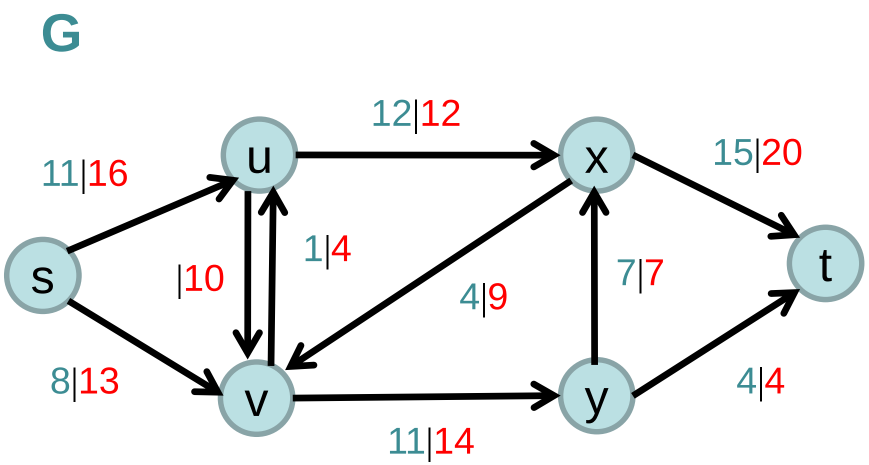
c) A simple path P from s to t in G_f is called an augmenting path.
The residual capacity $c_f\,(P)$ of P is defined as

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

Ford-Fulkerson Algorithm

Example: augmenting path and flow augmentation

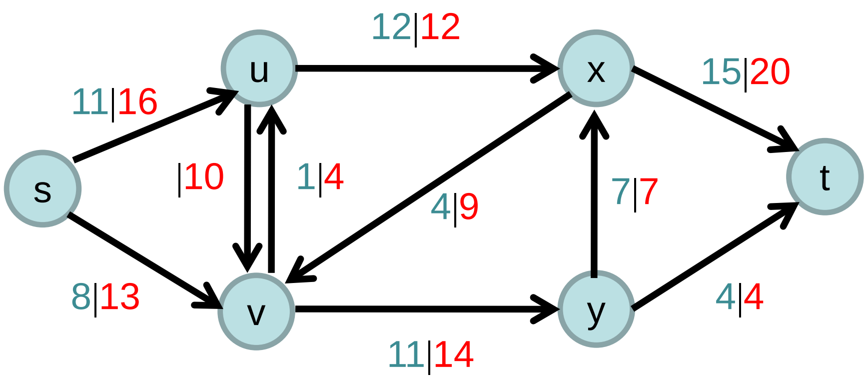
Flow network:



Example: augmenting path and flow augmentation

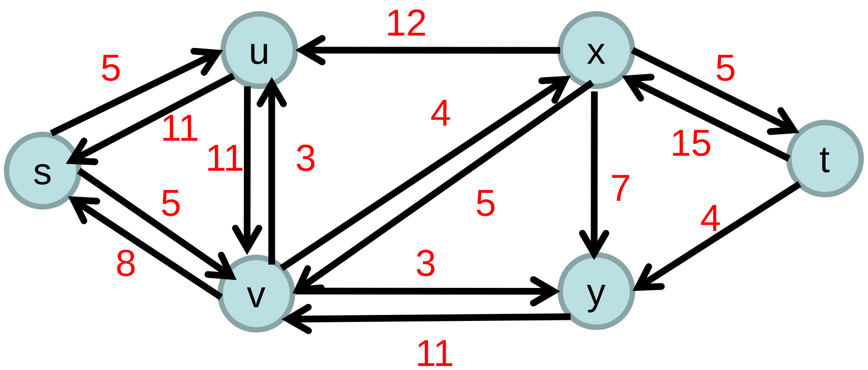
Flow network:

G



Residual network:

G_f

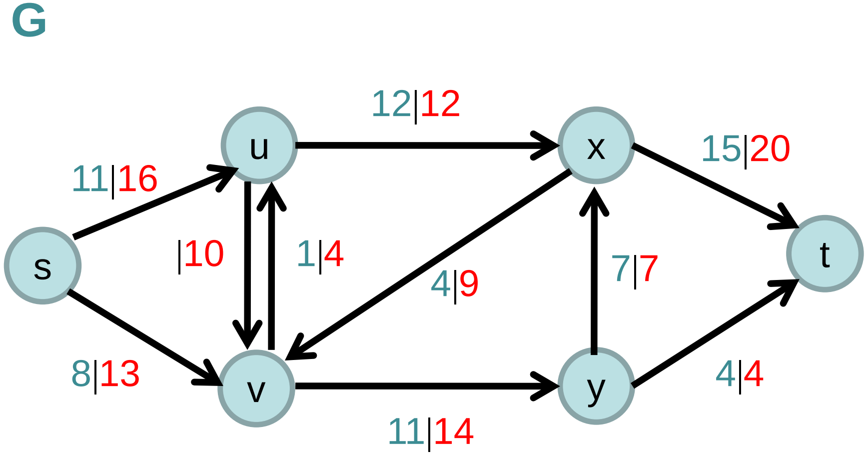


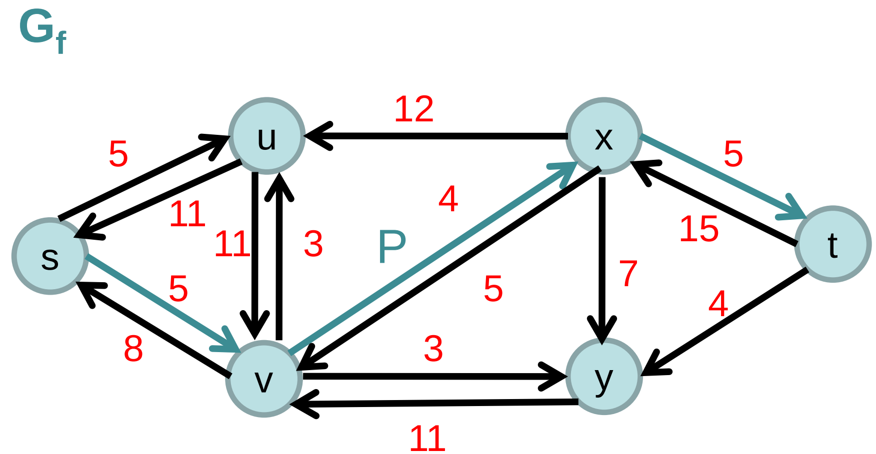
$c_f(u,v) = c\ (u,v) - f\ (u,v)$

Example: augmenting path and flow augmentation

Flow network:

Residual network with augmenting path:



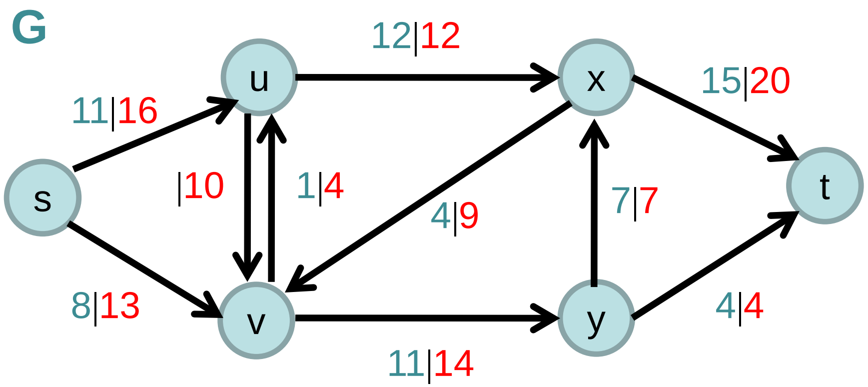


$c_f(P) = \min \{ c_f(u,v) \mid (u,v) \in P \}$

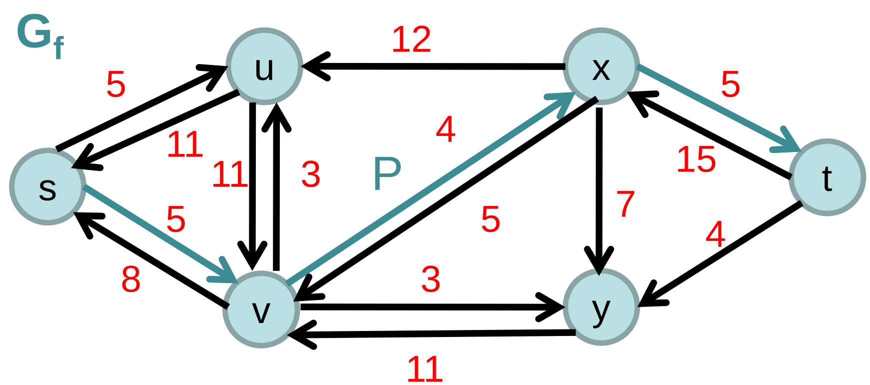
$\rightarrow$ residual capacity of path P: 4

Example: augmenting path and flow augmentation

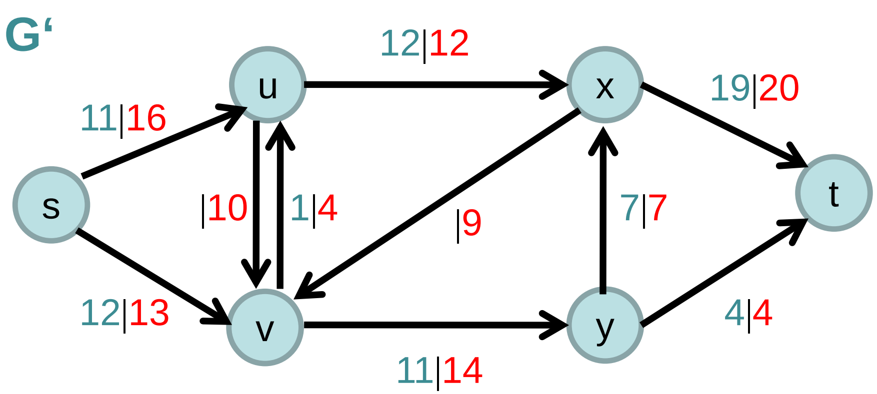
Flow network:



Residual network with augmenting path:



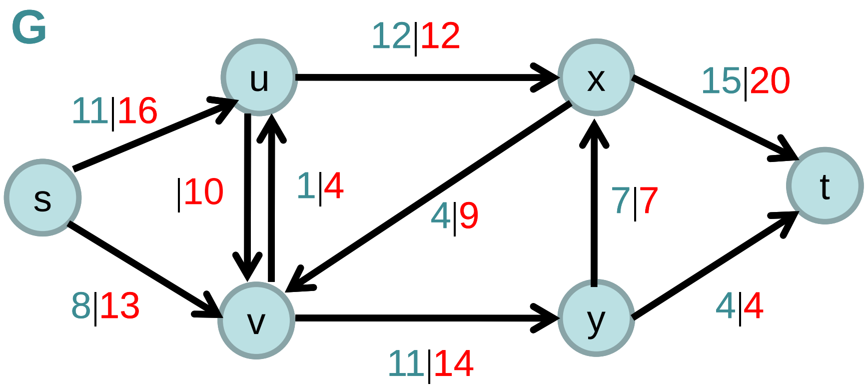
Augmented flow:



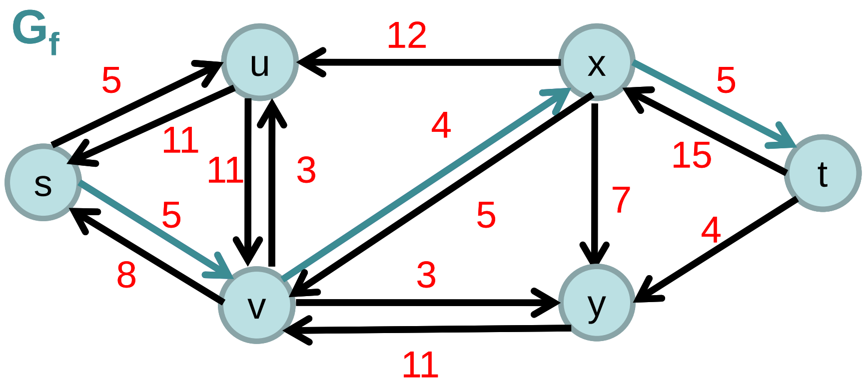
Path P of residual capacity 4 added to G:

Example: augmenting path and flow augmentation

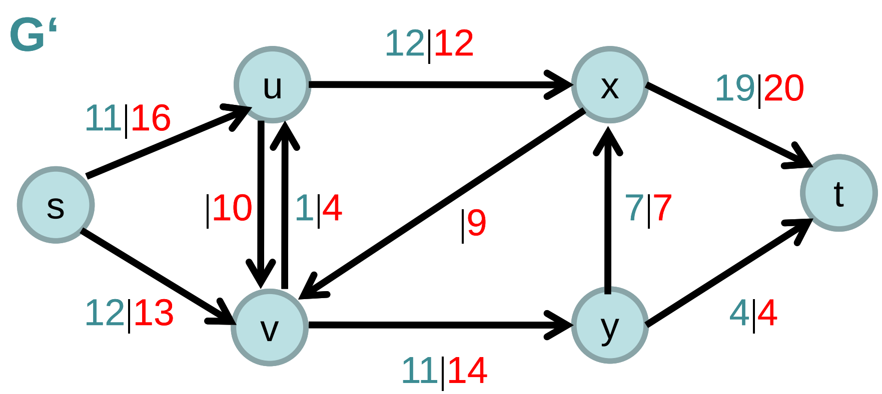
Flow network:



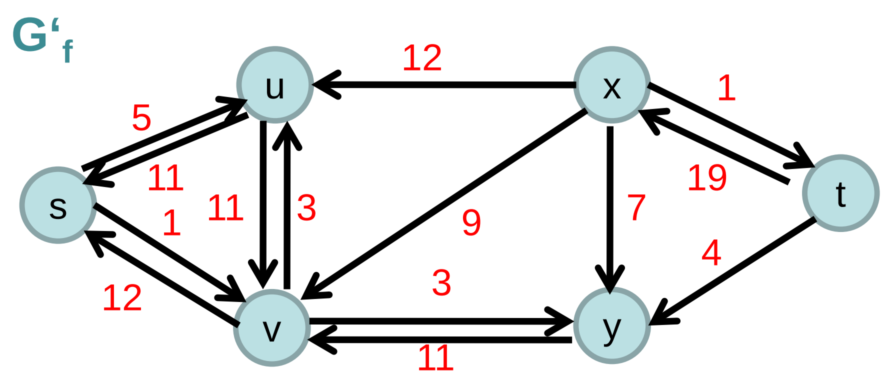
Residual network with augmenting path:



Augmented flow:



New residual network:



Ford-Fulkerson Algorithm

Are we allowed to add a valid flow in G_f to a flow in G?

Lemma 7: Let (G, s, t, c) be a flow network and f be a flow in G. Let G_f be the residual network of G induced by f, and let f' be a flow in G_f. Then

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

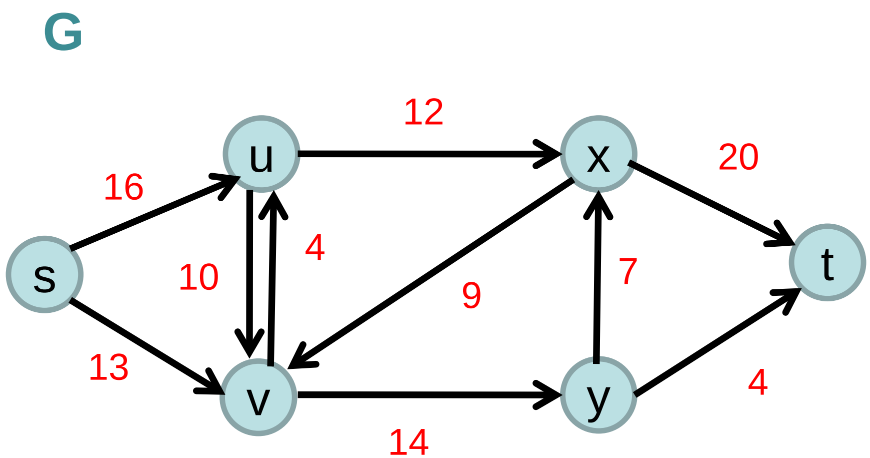
is a valid flow in G with value $|f + f'| = |f| + |f'|$.

Ford-Fulkerson Algorithm

```
FORDFULKERSON (Flow network G = (V, E), s, t, c))
{
   for each edge (u, v) ∈ E
      { f [u, v] := 0; f [v, u] := 0; }                       // initially empty flow
   G_f := residual network of G  w.r.t. f;
   while (∃ a path P from s to t in G_f)                     // P is an augmenting path
   {  // compute maximal flow along P
      c_f (P) := min {c_f (u, v) | (u, v) ∈ P)};             // c_f (u, v) = c (u, v) – f (u, v)
      for each edge (u, v) ∈ P                               // update flow along P
         { f [u, v] := f [u, v] + c_f (P); f [v, u] := - f [u, v]; }
      G_f := residual network of G w.r.t. f;
   }
   output f
}
```

Example: Ford-Fulkerson Algorithm

Flow network:

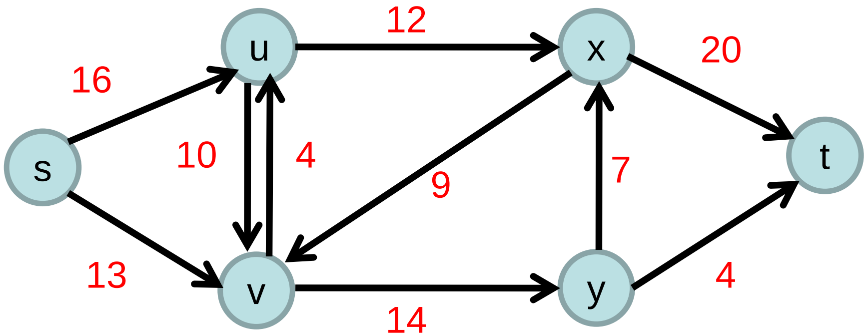


Example: Ford-Fulkerson Algorithm

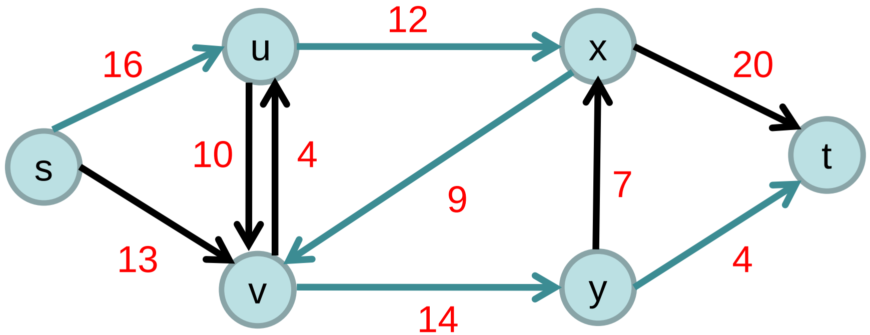
Flow network:

Residual network with augmenting path:

G



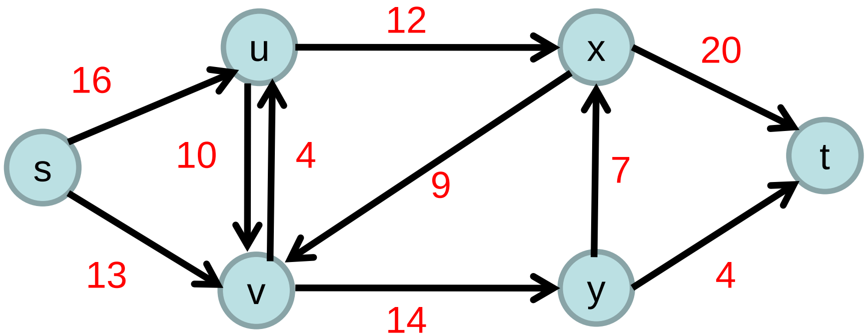
G_f



Example: Ford-Fulkerson Algorithm

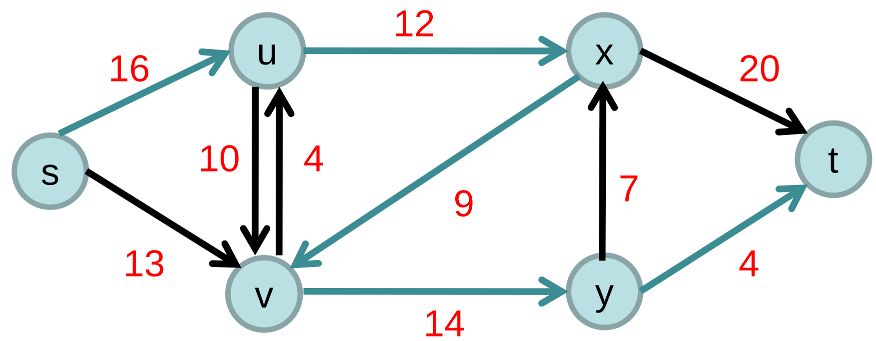
Flow network:

G



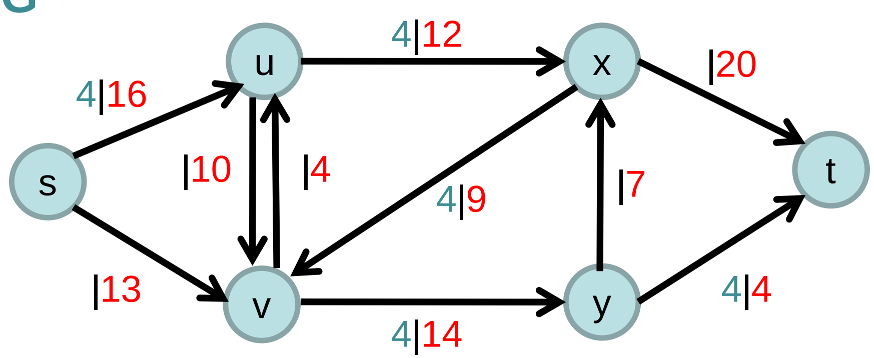
Residual network with augmenting path:

G_f



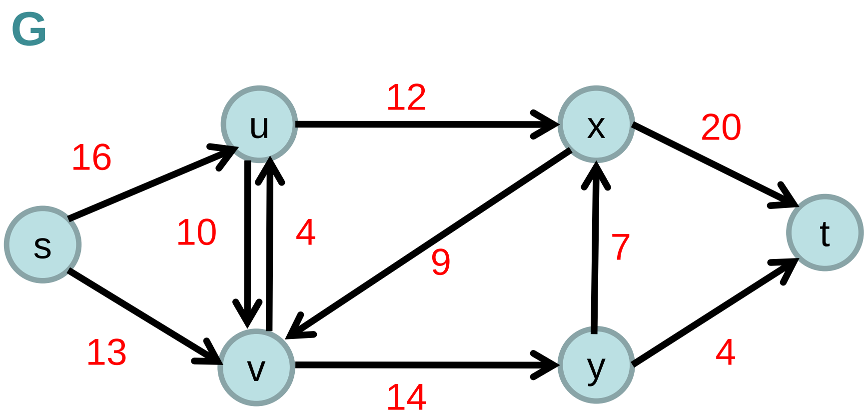
Augmented flow:

G

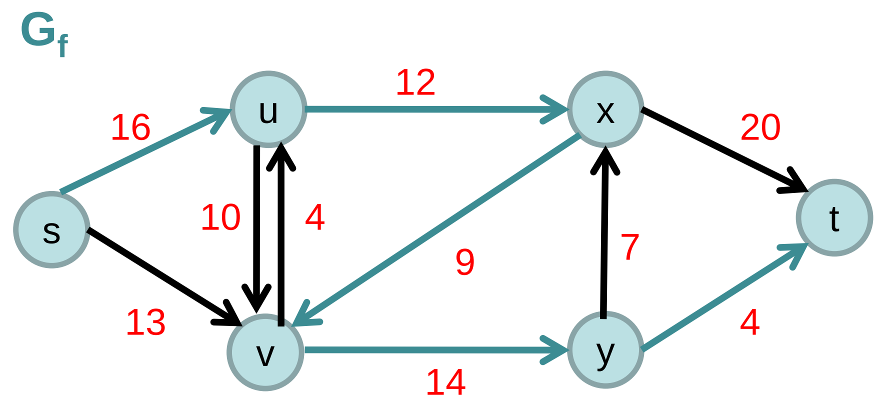


Example: Ford-Fulkerson Algorithm

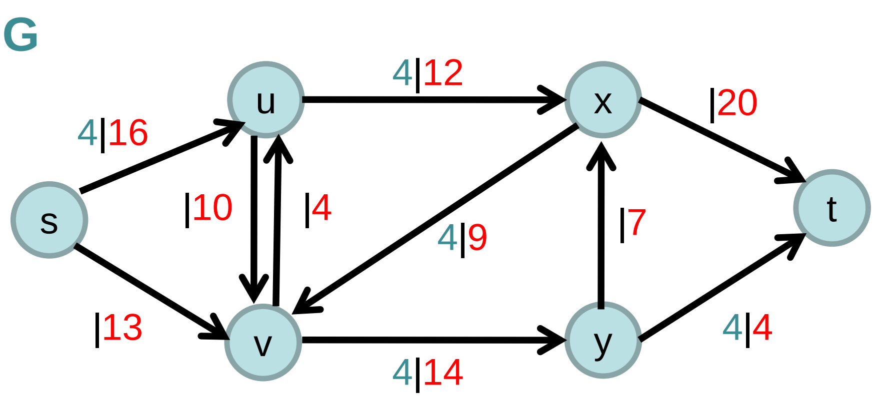
Flow network:



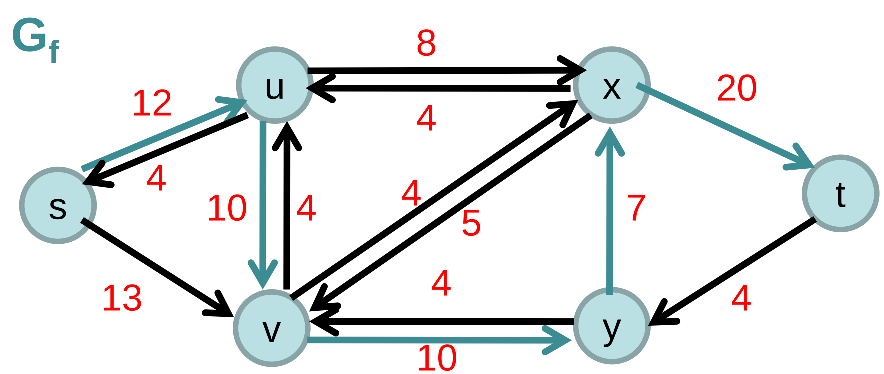
Residual network with augmenting path:



Augmented flow:



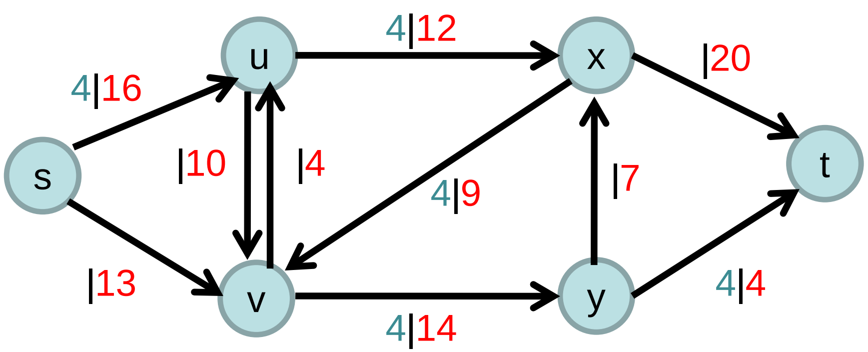
New residual network with augmenting path:



Example: Ford-Fulkerson Algorithm

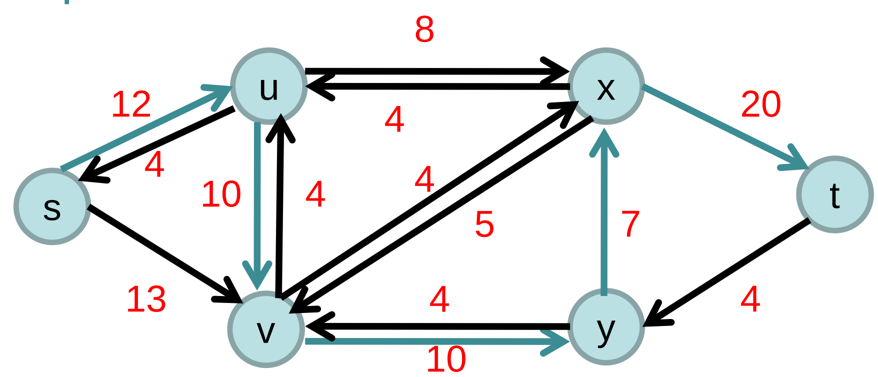
Flow network:

G



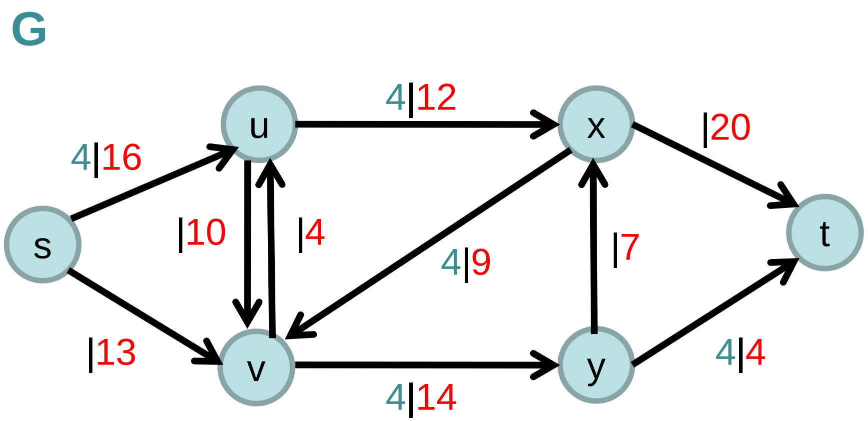
Residual network with augmenting path:

G_f

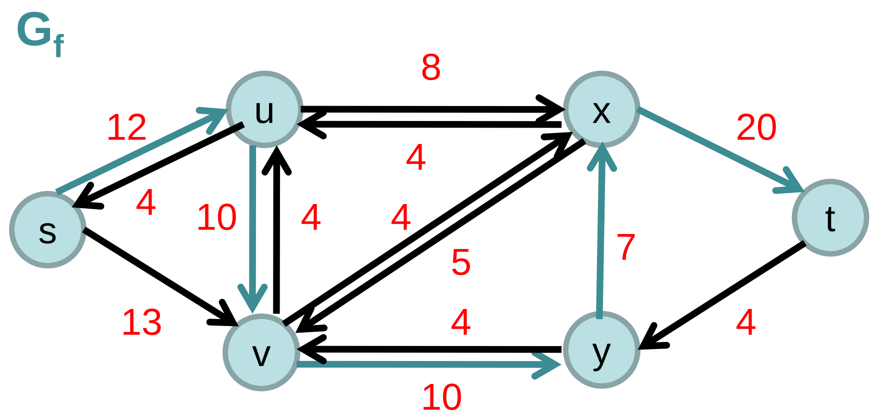


Example: Ford-Fulkerson Algorithm

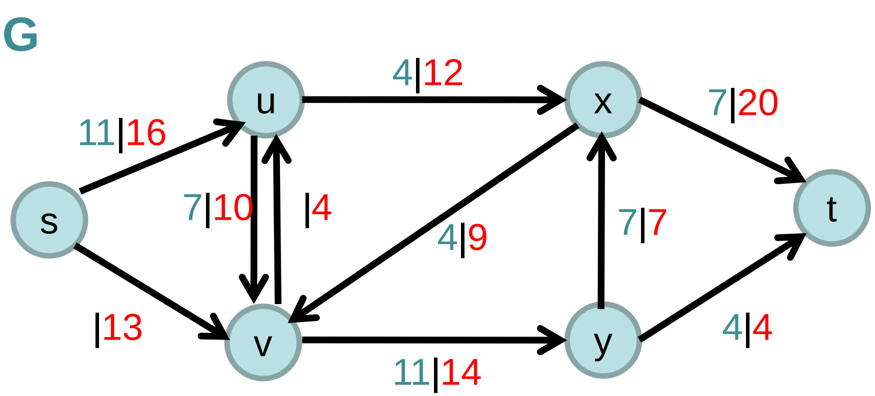
Flow network:



Residual network with augmenting path:



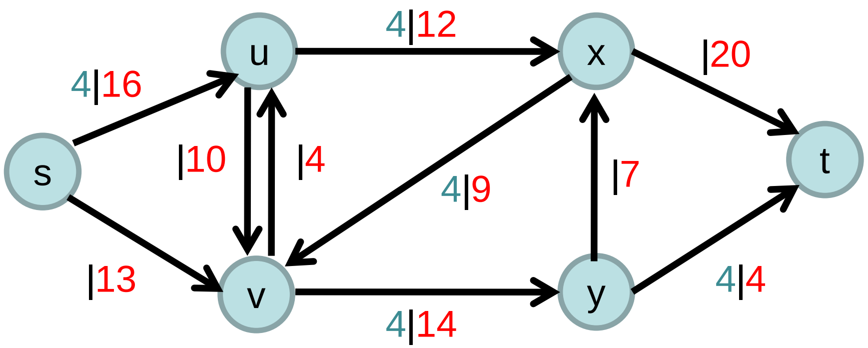
Augmented flow:



Example: Ford-Fulkerson Algorithm

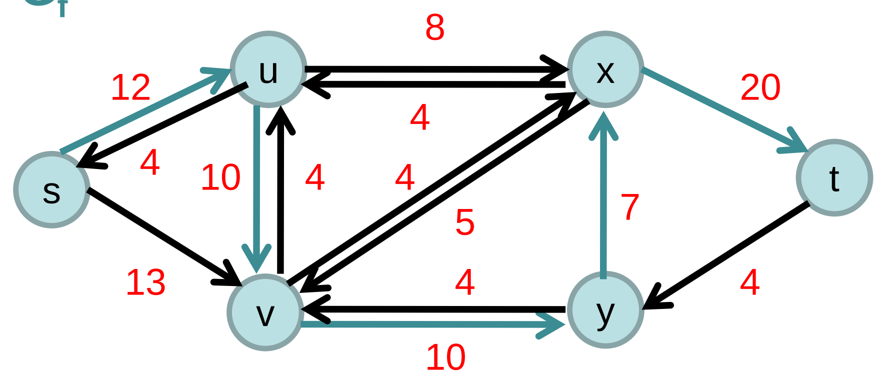
Flow network:

G



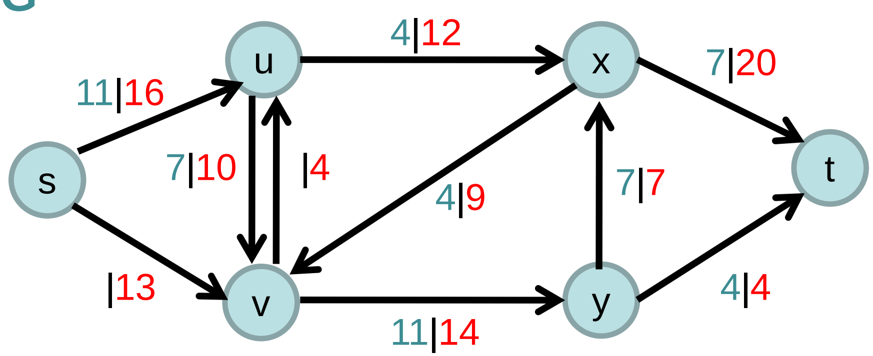
Residual network with augmenting path:

G_f



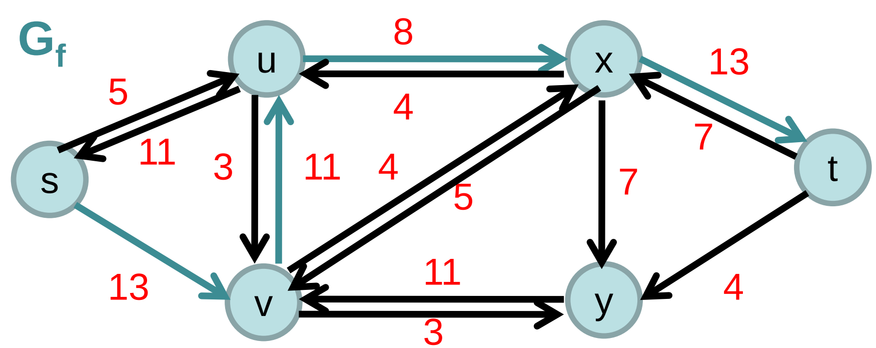
Augmented flow:

G



New residual network with augmenting path:

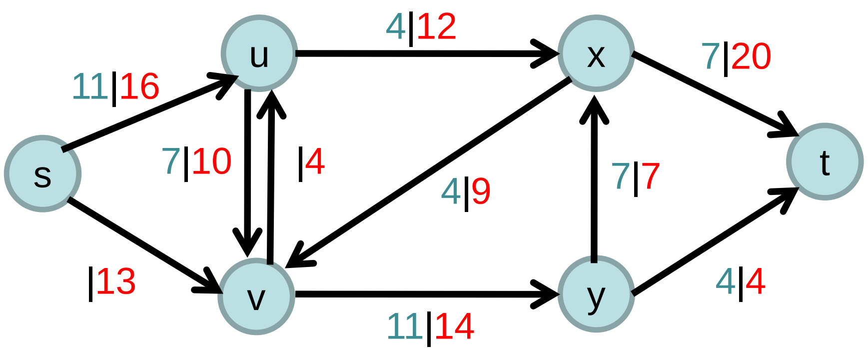
G_f



Example: Ford-Fulkerson Algorithm

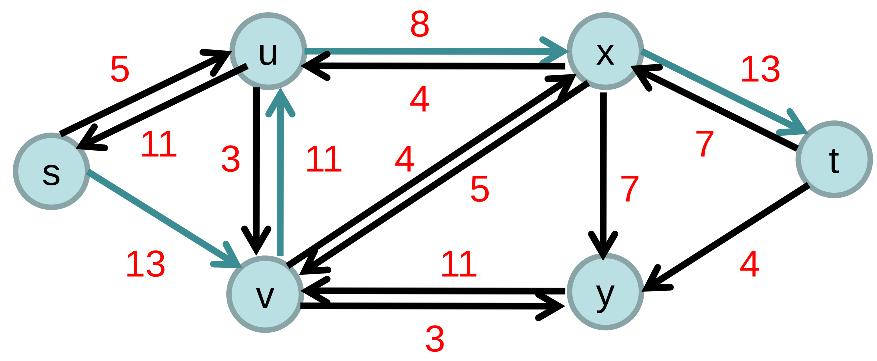
Flow network:

G



Residual network with augmenting path:

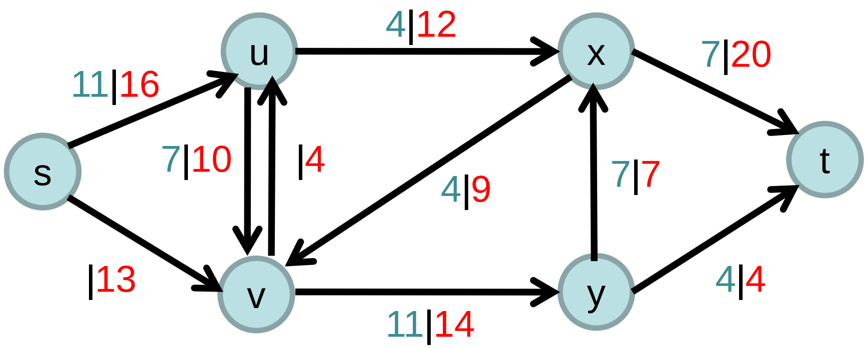
G_f



Example: Ford-Fulkerson Algorithm

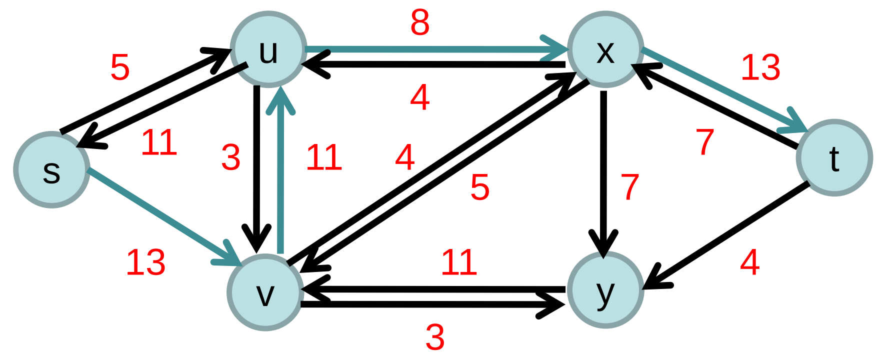
Flow network:

G



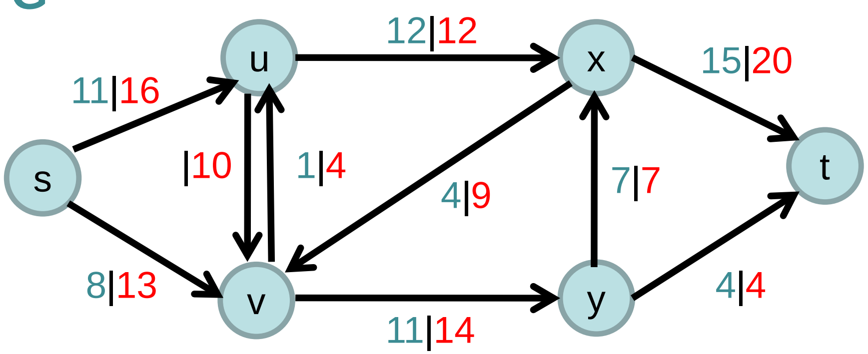
Residual network with augmenting path:

G_f



Augmented flow:

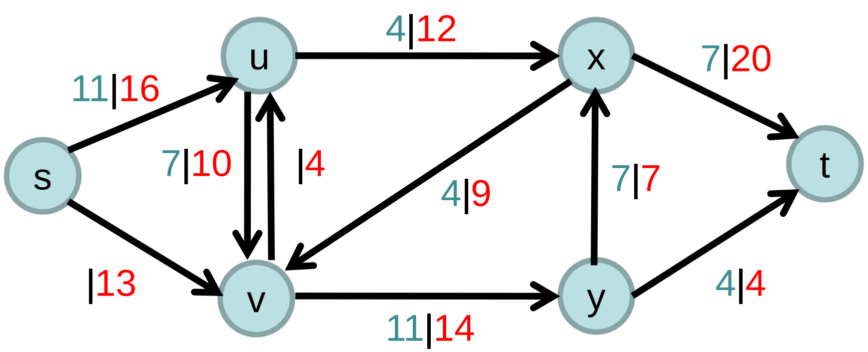
G



Example: Ford-Fulkerson Algorithm

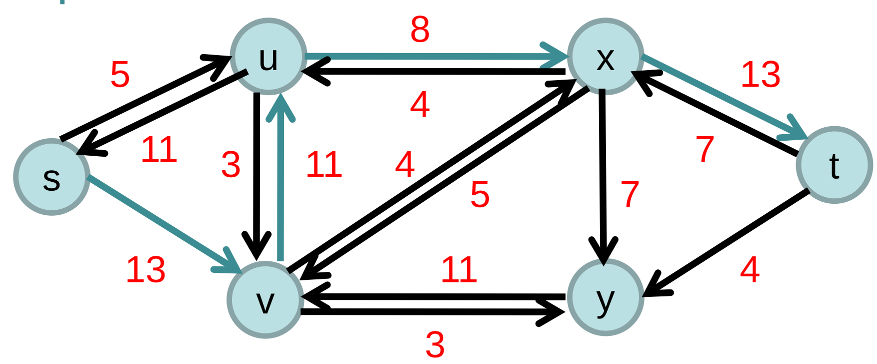
Flow network:

G



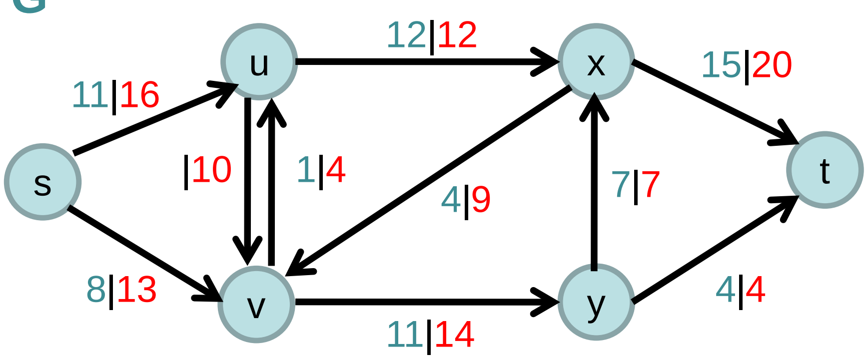
Residual network with augmenting path:

G_f



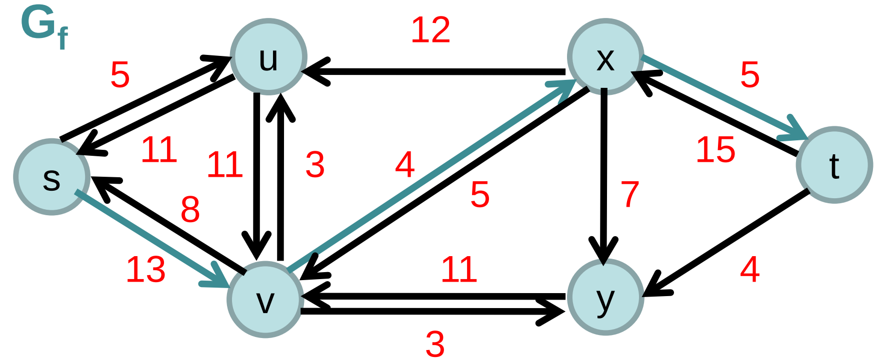
Augmented flow:

G



New residual network with augmenting path:

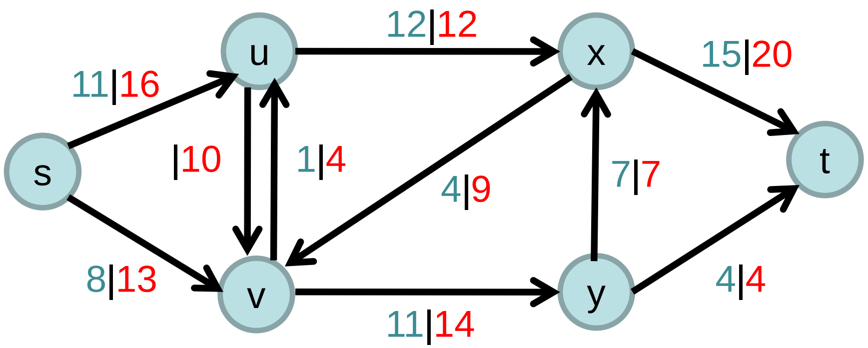
G_f



Example: Ford-Fulkerson Algorithm

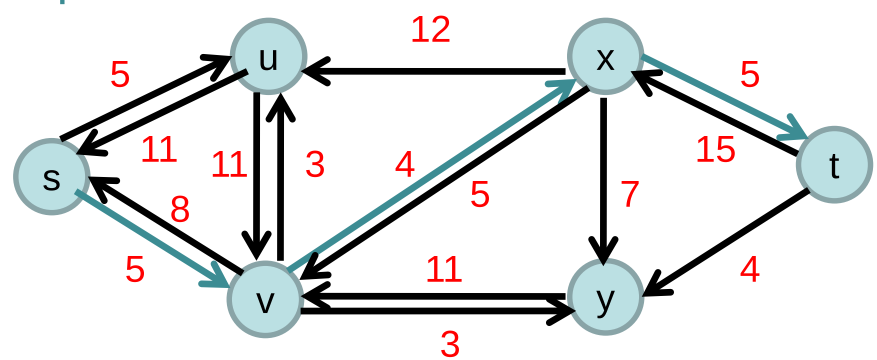
Flow network:

G



Residual network with augmenting path:

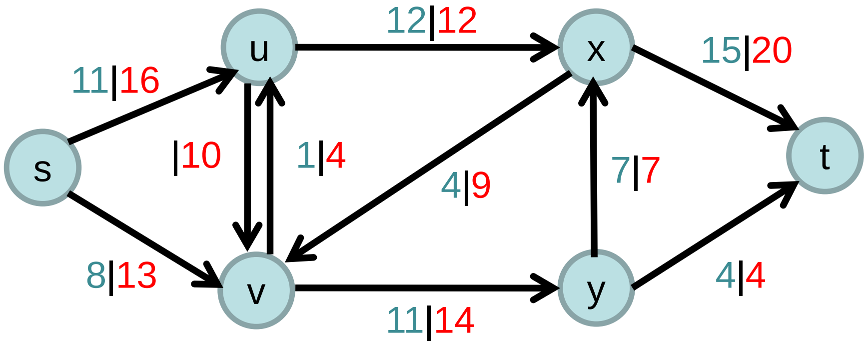
G_f



Example: Ford-Fulkerson Algorithm

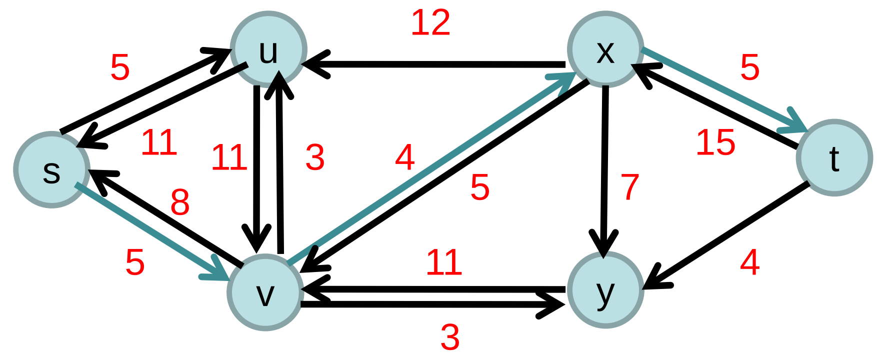
Flow network:

G



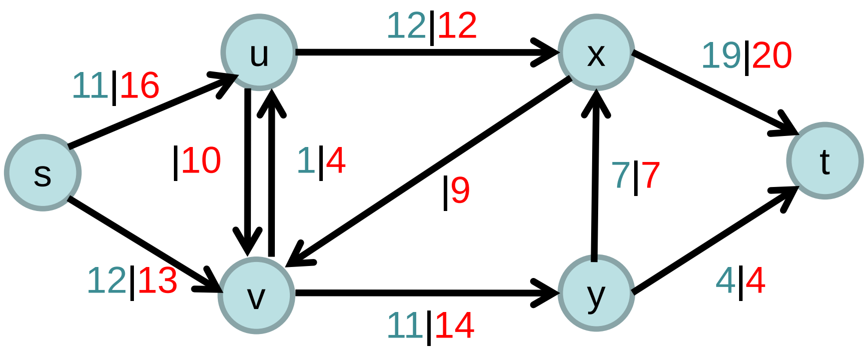
Residual network with augmenting path:

G_f



Augmented flow:

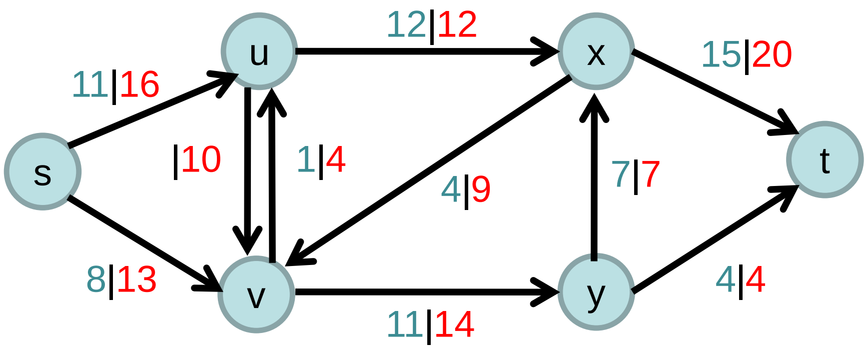
G



Example: Ford-Fulkerson Algorithm

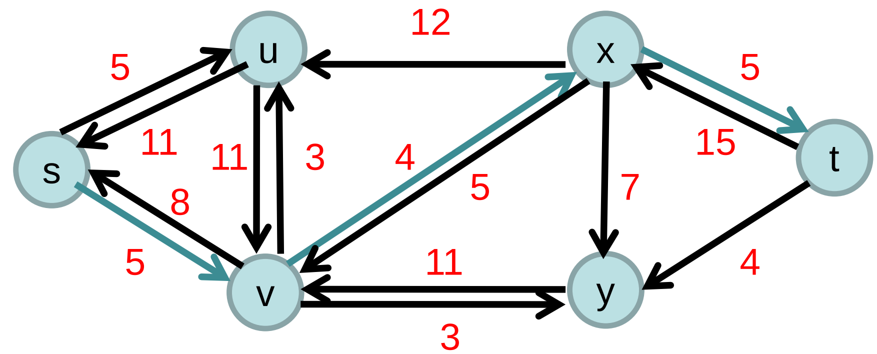
Flow network:

G



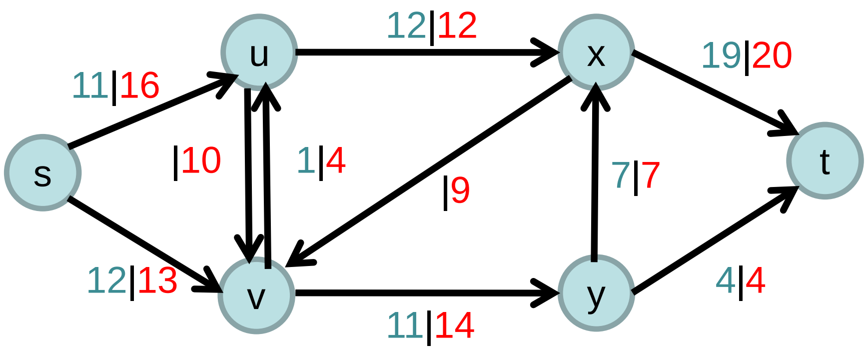
Residual network with augmenting path:

G_f



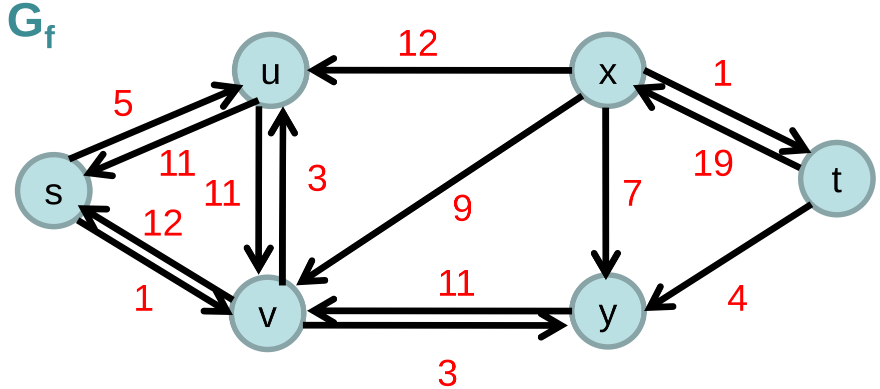
Augmented flow:

G



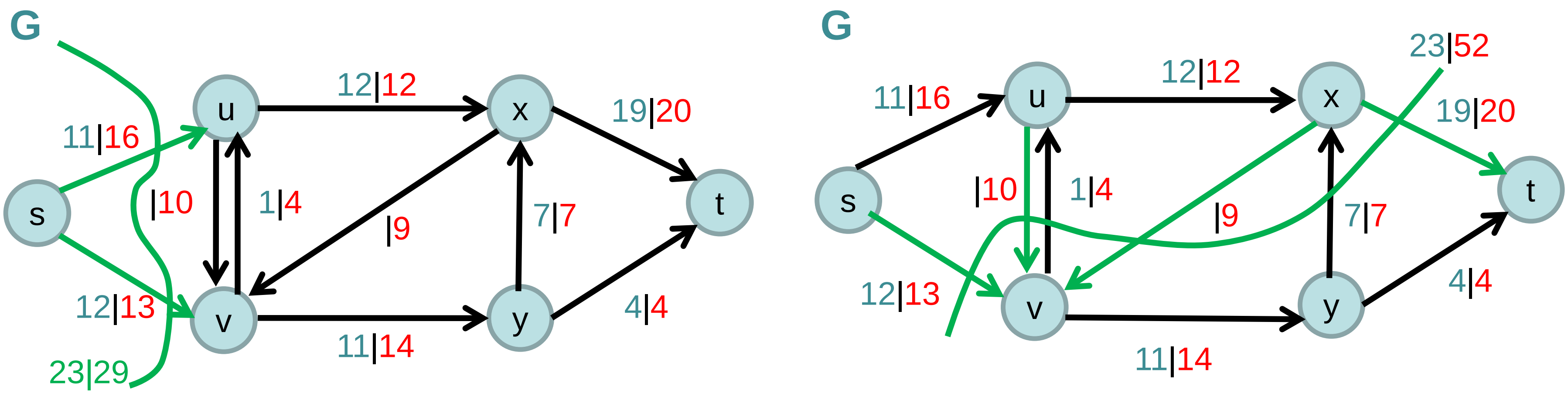
New residual network with augmenting path:

G_f

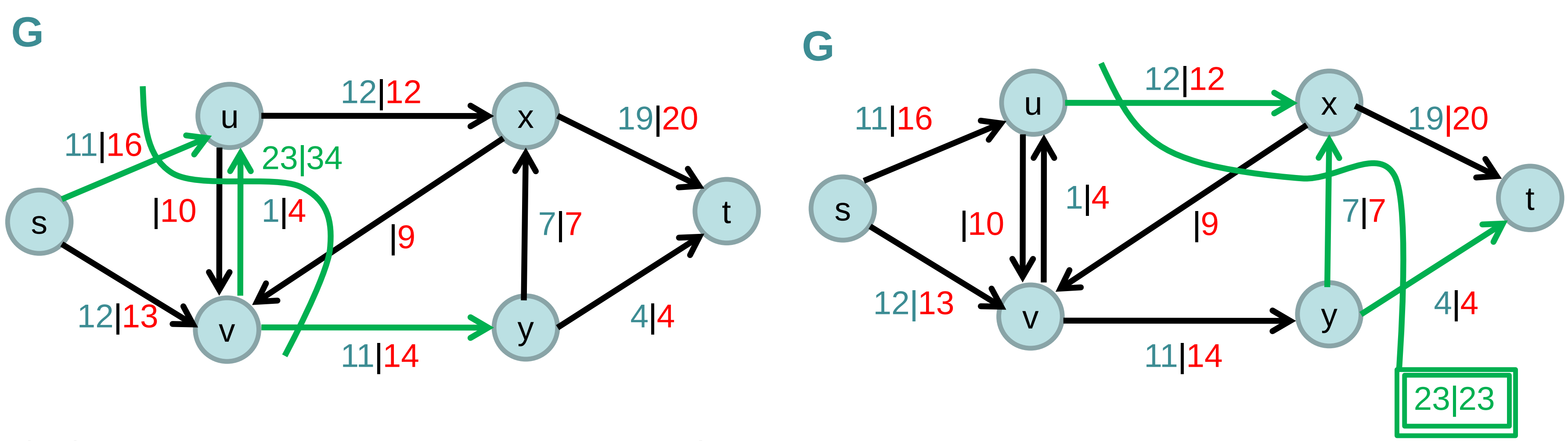


Example: Ford-Fulkerson Algorithm

Flow network:



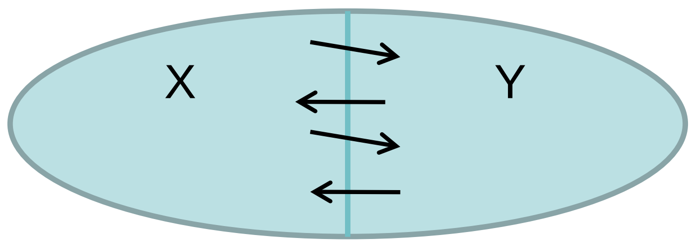
Augmented flow:



Do we always have a maximum flow f if G_f has no more paths from s to t?

Definition 8: Let (G,s,t,c) be a flow network. For some cut (X,Y) of V we define

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y), \quad c(X, Y) = \sum_{x \in X} \sum_{y \in Y} c(x, y)$$



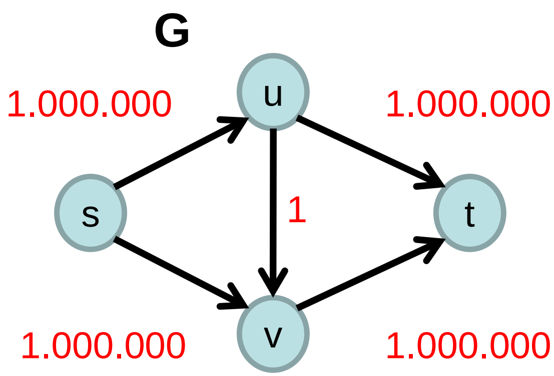
Theorem 9: (Max-Flow Min-Cut Theorem)

Let (G,s,t,c) be a flow network and f be a flow in G. Then the following statements are equivalent.

a) f is a maximal flow in G.
b) The residual network G_f of G w.r.t. f does not contain any augmenting path.
c) $|f| = c(S, T)$ for some cut (S, T) of G with $s \in S$ and $t \in T$.

Edmonds-Karp Algorithms

Problem: in the worst case, the Ford-Fulkerson Algorithm is too slow



If we always pick an augmenting path along the edge of capacity 1, it takes 2.000.000 (!) augmentations to reach a maximum flow.

In 1972, Edmonds and Karp proposed two heuristics in order to compute maximal flows more efficiently.

Heuristic 1: Choose the augmenting path of largest value.

Heuristic 2: Choose the shortest augmenting path.

Edmonds-Karp Algorithms

Theorem 10: Let (G, s, t, c) be a flow network with integer capacities $c(u, v)$. Then heuristic 1 computes a maximal flow f^* in time

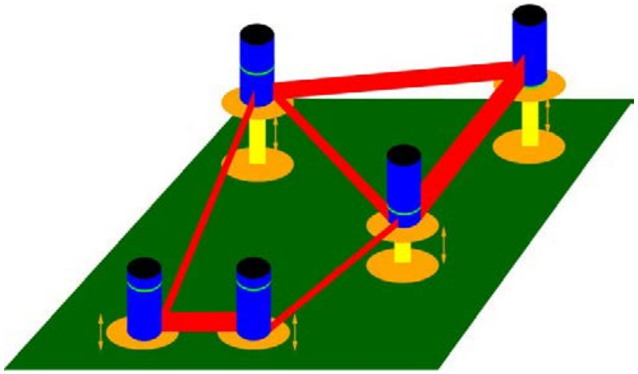
$$O(|E|^2 \cdot \log |E| \cdot \log |f^*|).$$

Theorem 11: Let (G, s, t, c) be a flow network with integer capacities $c(u, v)$. Then heuristic 2 computes a maximal flow in time $O(|E|^2 \cdot |V|)$.

Goldberg´s Algorithm

Intuition:

- A flow network can be seen as a network of liquids: edges correspond to pipes and nodes correspond to pipe connections.

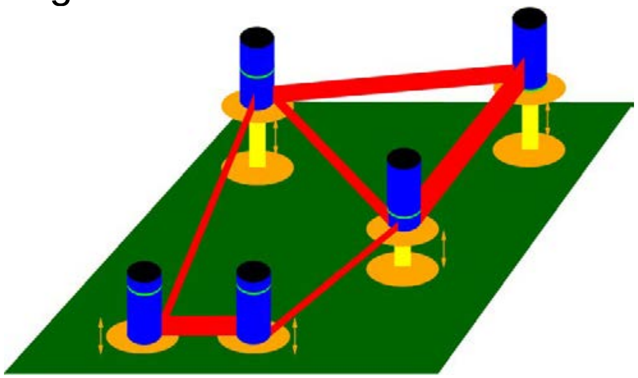


- Every node has a reservoir that can collect an arbitrary amount of liquid.
- Every node, its reservoir, and all of its pipes are arranged on a platform whose height may increase during the execution of the algorithm.

Goldberg´s Algorithm

Intuition:

• The node heights determine how the flow is moved through the network: flow always flows downhill.



• Initially, the source s pumps as much flow as possible into the network (= c(s, V – s)).

• If the flow reaches some intermediate node, it is collected in its reservoir. From there it will be sent downhill later.

• If all non-saturated pipes that leave a node u lead to nodes v that are above u, then the height of u will be increased, i. e., we lift u.

• If the total flow that can flow to a sink, reaches it, then the excess flow in the reservoirs is sent back to the source by lifting the heights of the intermediate nodes beyond the height of the source.

Goldberg´s Algorithm

Definition 12: Let (G,s,t,c) be a flow network. A preflow is a function $f:V\times V\rightarrow\mathbb{R}$ satisfying the following properties:

- $f(u, v) \leq c(u, v)$ for all $u, v \in V$ (capacity constraints)
- $f(u, v) = - f(v, u)$ for all $u, v \in V$ (skew symmetry)
- $f(V, u) \geq 0$ for all $u \in V \setminus \{s\}$ (preflow condition)

- The excess flow of a node v is defined as $e_f(v)=f(V,v)$. A node $v\neq t$ is called active if $e_f(v)>0$.

- Goldberg´s Algorithm assigns to each node v a height $h(v)\in\mathbb{N}_0$. The height function is called legal if $h(s)=|V|$, $h(t)=0$, and for all edges (v,w) in the residual network G_f, $h(v)\leq h(w)+1$.
(I.e., for all $(v,w)\in E$ with $h(v)>h(w)+1$, $(v,w)\notin E_f$.)

- An edge (v,w) in G_f is called admissible if $h(v)>h(w)$.
(Together with the previous condition it follows that $h(v)=h(w)+1$.)

Goldberg´s Algorithm

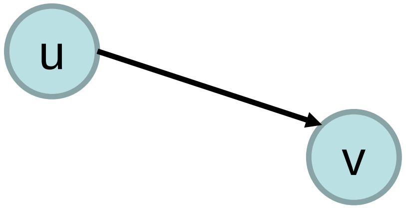
Basic Operations:

- Push(u,v): push as much flow as possible from u to v
- Lift(u): lift u as much as possible without violating the legality of the height function.

In pseudocode:

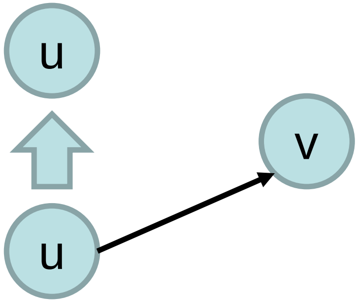
Push(u,v):

$\delta:=\min\{e_f(u),c_f(u,v)\}$
$f(u,v):=f(u,v)+\delta$
$c_f(u,v):=c_f(u,v)-\delta$
$c_f(v,u):=c_f(v,u)+\delta$
$e_f(u):=e_f(u)-\delta$
$e_f(v):=e_f(v)+\delta$



Lift(u):

$h(u):=\min\{ h(v)+1 \mid (u,v)\in E_f \}$



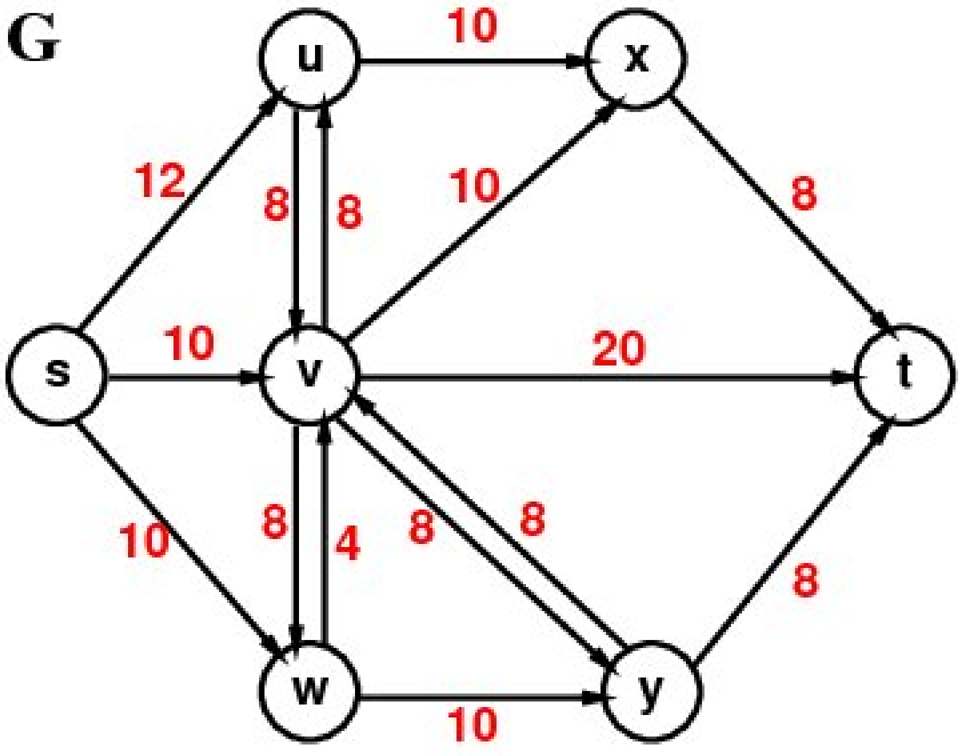
Goldberg´s Algorithm

Goldberg´s Algorithm works as follows:

Preflow-Push Algorithm:

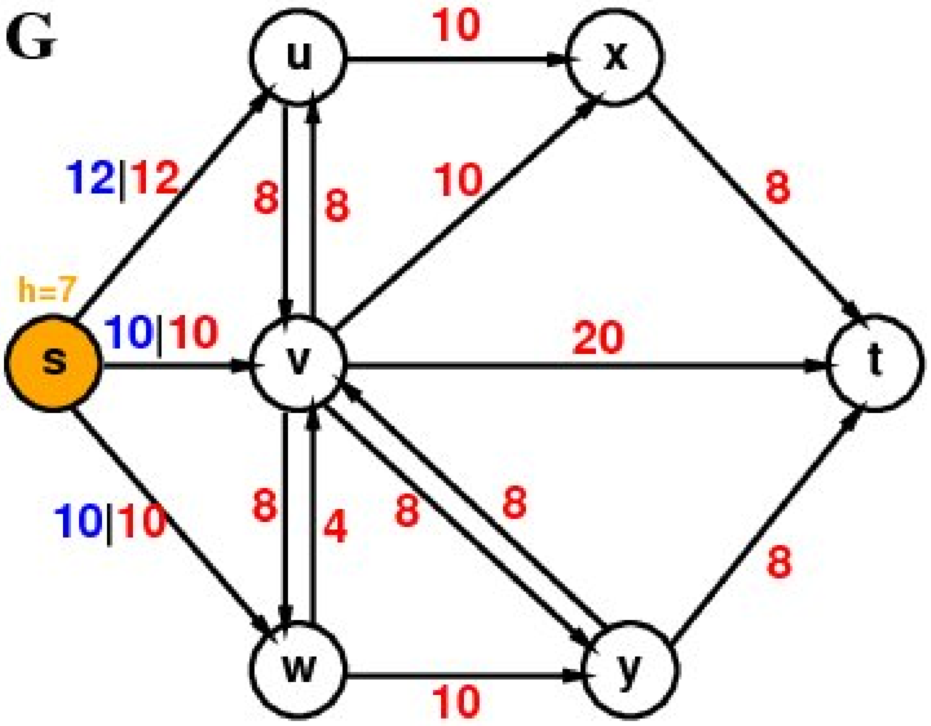
for each $u \in V \setminus \{s\}$ do $h(u):=0$; $e_f(u):=0$
for each $(u,v) \in E$ do $f(u,v):=0$; $f(v,u):=0$
$h(s):=|V|$
for each $(s,u) \in E$ do
  $f(s,u):=c(s,u)$; $f(u,s):=-f(s,u)$; $e_f(u):=c(s,u)$
while (there are active nodes u) do
  if (there is an admissible edge (u,v))
    then Push(u,v)
    else Lift(u)

Example:



Capacities are marked in red

Example:



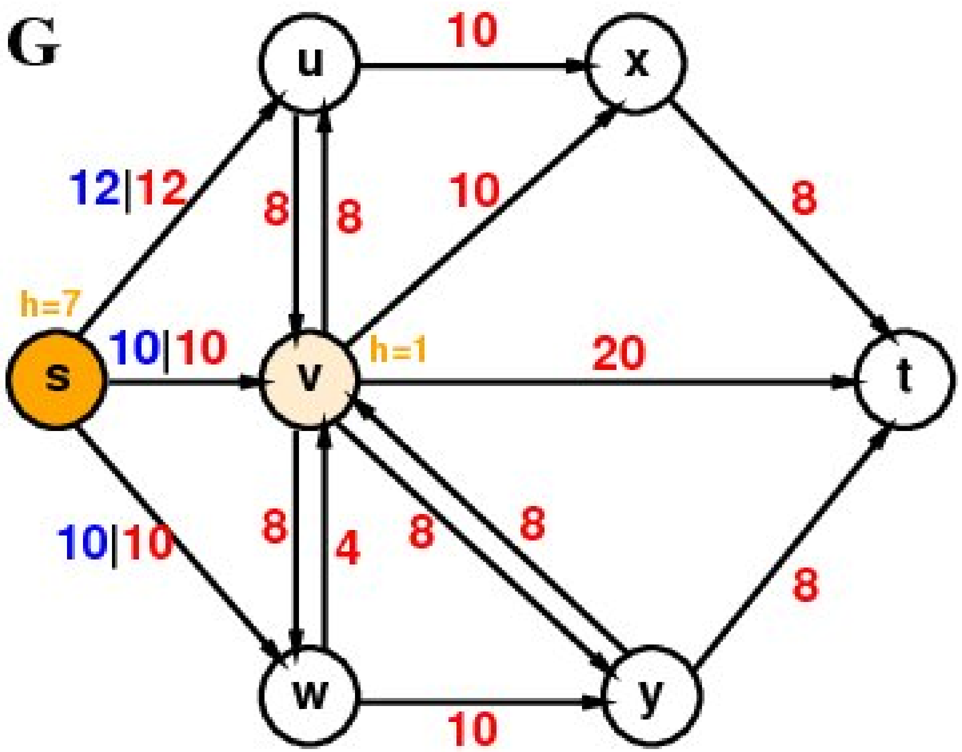
After initialization:

- s is lifted to height 7. The heights of all other nodes are set to 0.
- Every edge from s is saturated. All other edges have a flow of 0.

No PUSH-operation can currently be executed.

Operations that can be executed are LIFT(u), LIFT(v) or LIFT(w).

Example:

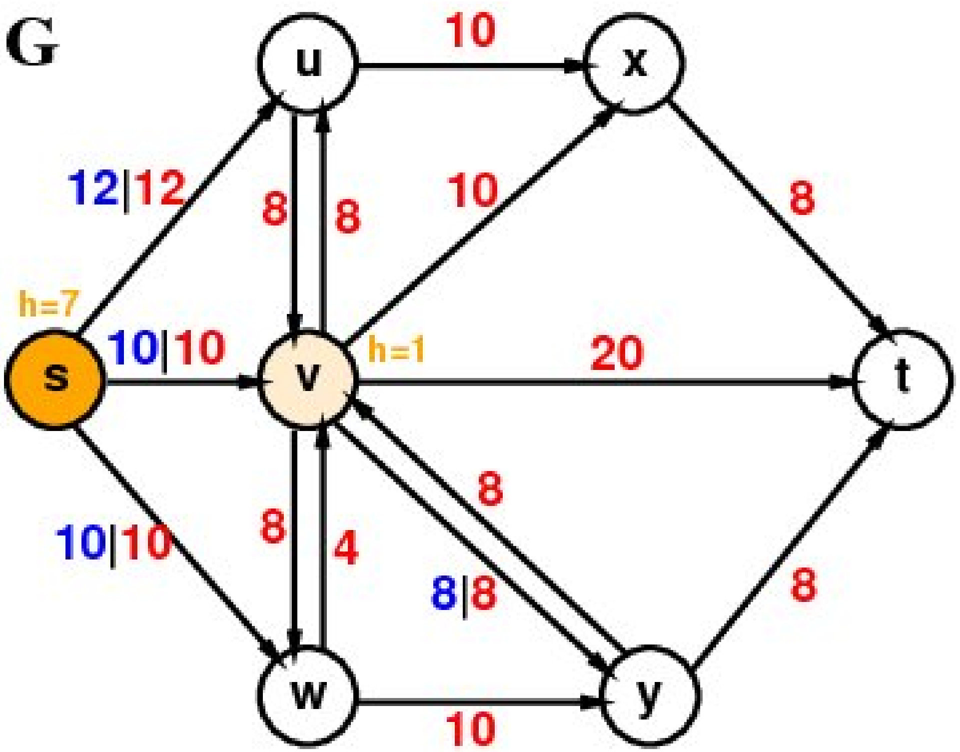


After LIFT(v):

The height h(v) is set to
$1 + \min \{h[u] \mid (v, u) \in E_f\}$
$= 1 + 0 = 1$.

Now, operations that can be executed are LIFT(u), LIFT(w) or PUSH(v, u), PUSH(v, w), PUSH(v, x), PUSH(v, y), PUSH(v, t).

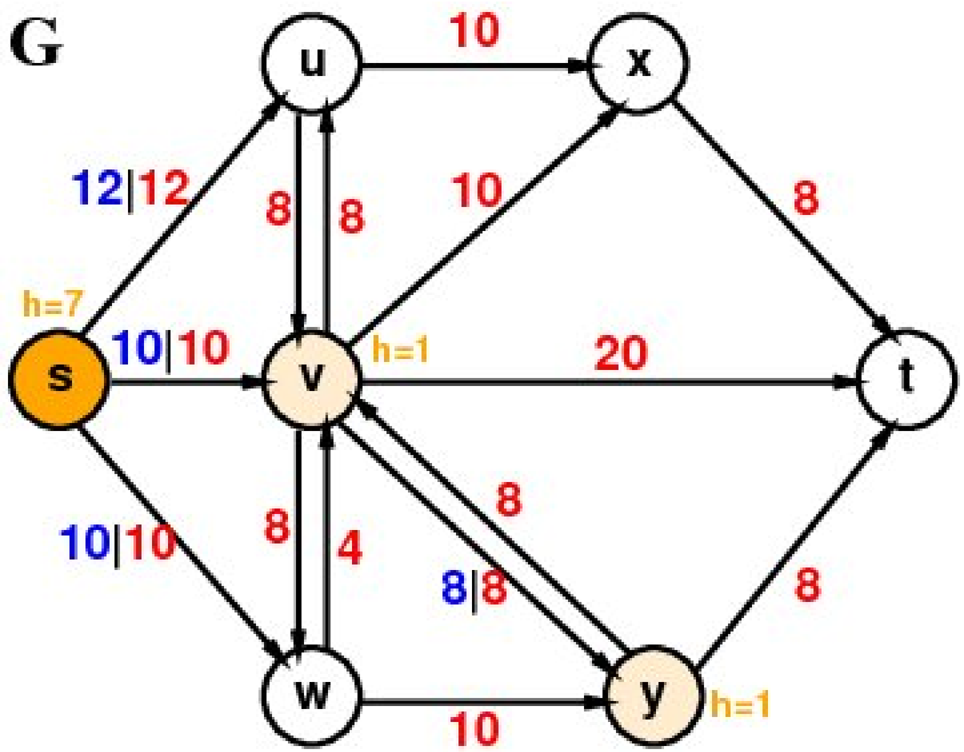
Example:



After PUSH(v, y):

Operatons that can be executed are LIFT(u), LIFT(w), LIFT(y) or PUSH(v, u), PUSH(v, w), PUSH(v, x), PUSH(v, t).

Example:

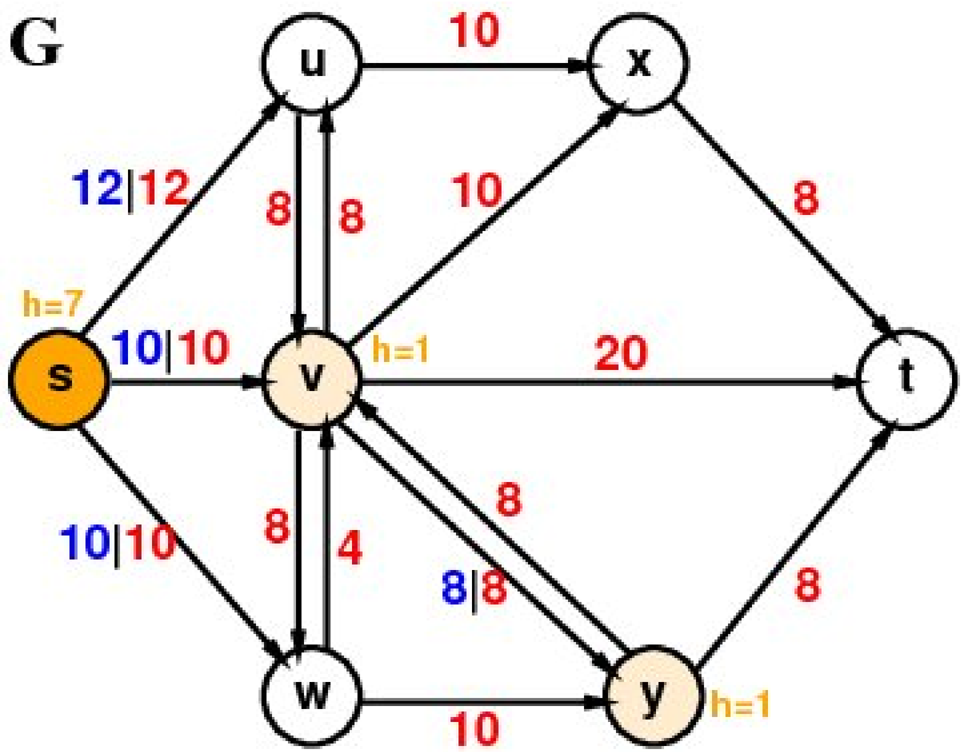


After LIFT(y):

The height h(y) is set to
$1 + \min\{h[u] \mid (y, u) \in E_f\}$
$= 1 + 0 = 1$.

Operations that can be executed are LIFT(u), LIFT(w) or PUSH(v, u), PUSH(v, w), PUSH(v, x), PUSH(v, t), PUSH(y, t).

Example:



After PUSH(y, t):

Operations that can be executed are LIFT(u), LIFT(w) or PUSH(v, u), PUSH(v, w), PUSH(v, x), PUSH(v, t).

The algorithm continues to run until no PUSH or LIFT operation can be executed.

Goldberg´s Algorithm

Theorem 13: Let (G, s, t, c) be any flow network with n nodes and m edges. Then Goldberg´s Algorithm has a runtime of $O(n^2 m)$.

With an improved selection of Push and Lift Operations, this runtime can be improved.

Rules for the choice of active nodes:

- FIFO: The active nodes are organized in a FIFO queue, i.e., new active nodes are added to the back of the queue and active nodes to be processed are taken from the front. With this rule, a runtime of $O(n^3)$ can be reached.
- Highest-Label-First: Always take the active node of largest height. In this case, one can reach a runtime of $O(\sqrt{m} \cdot n^2)$.

Other Variants

- Goldberg, 1985: FIFO PPA: $O(|V|^3)$.
- Goldberg, Tarjan, 1986:
  Improved FIFO PPA: $O(|V| \cdot |E| \cdot \log (|V|^2 \cdot |E|))$.
- Goldberg, Tarjan, 1986, Cheriyan, Maheshwari 1989:
  Highest Label PPA: $O(|V|^2 \cdot \sqrt{|E|})$.
- King, Rao, Tarjan, 1994:
  $O(|V| \cdot |E| \log_{|E|/(|V| \log |V|)} |V|)$.
- Orlin, 2013:
  $O(|V| \cdot |E|)$.
- Randomized Variants

History of maximal flow algorithms:

G = (V, E) with |V| = n, |E| = m, U: value of maximal flow.

	Year	Researcher	Run time
1.	1951	Dantzig	$O(n^2 mU)$
2.	1955	Ford, Fulkerson	$O(nmU)$
3.	1970	Dinitz / Edmonds, Karp	$O(nm^2)$
4.	1970	Dinitz	$O(n^2 m)$
5.	1972	Edmonds, Karp / Dinitz	$O(m^2 \log U)$
6.	1973	Dinitz / Gabow	$O(nm \log U)$
7.	1974	Karzanov	$O(n^3)$
8.	1977	Cherkassky	$O(n^2 \sqrt{m})$
9.	1980	Galil, Naamad	$O(nm \log^2 n)$
10.	1983	Sleator, Tarjan	$O(nm \log n)$
11.	1986	Goldberg, Tarjan	$O(nm \log(n^2/m))$
12.	1987	Ahuja, Orlin	$O(nm + n^2 \log U)$
13.	1987	Ahuja et al.	$O(nm \log(n\sqrt{\log U}/(m+2)))$
14.	1989	Cheriyan, Hagerup	$E(nm + n^2 \log^2 n)$
15.	1990	Cheriyan et al.	$O(n^3/\log n)$
16.	1990	Alon	$O(nm + n^{8/3} \log n)$
17.	1992	King et al.	$O(nm + n^{2+\varepsilon})$
18.	1993	Philipps, Westbrook	$O(nm(\log_{m/n} n + \log^{2+\varepsilon} n))$
19.	1994	King et al.	$O(nm \log_{m/(n \log n)} n)$
20.	1997	Goldberg, Rao	$O(m^{3/2} \log(n^2/m) \log U)$
			$O(n^{2/3} m \log(n^2/m) \log U)$