Fundamental Algorithms

Chapter 6: Network Flow

Christian Scheideler

WS 2017

Foundations

Definition 6.1: 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 6.2: 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)

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

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

Foundations

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

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

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

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

Remark 6.3: 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 $f(u, v) = f(v, u) = 0$.

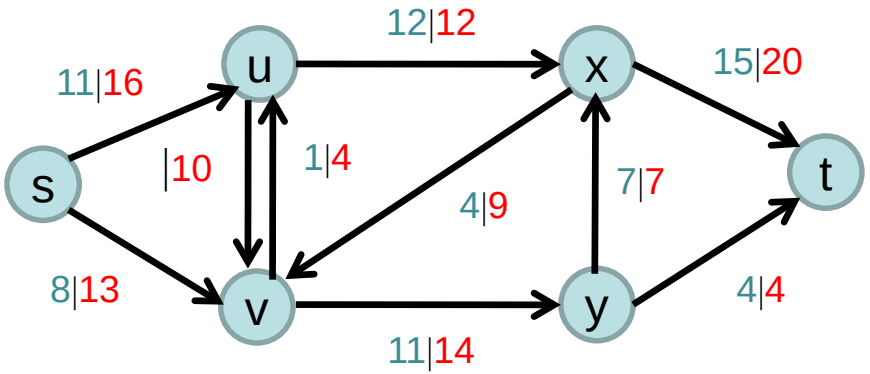
d) For all $v \in V \setminus \{s, t\}$,

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

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

Foundations

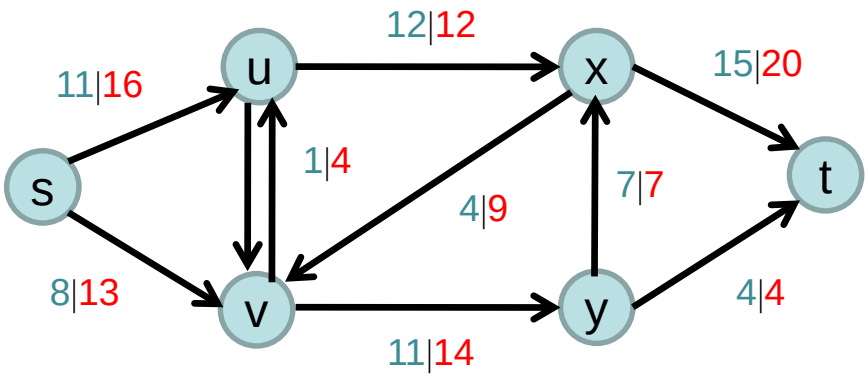
Example of a valid flow:



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 6.4: 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:

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

Foundations

Alternative definition of network flows:

Definition 6.5: Let (G,s,t,c) be a flow network. A network flow in G is a function $f : E \to \mathbb{R}_{\geq 0}$ with

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

i.e., we drop the skew symmetry constraint.

Remarks:

- Definition 6.5 is more intuitive whereas Definition 6.2 is more restrictive and sometimes simplifies the proofs.
- We will use the alternative Definition 6.5 in later parts of this chapter.

MAXFLOW Problem:

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

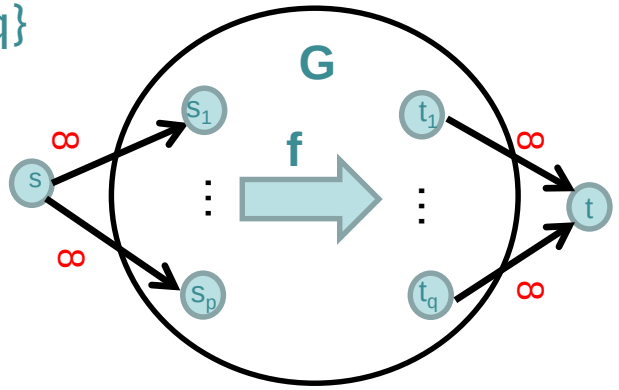
Remark 6.6: A maxflow problem $(G, s_1, \ldots, s_p, t_1, \ldots t_q, c)$ with multiple sources $s_1, \ldots, s_p$ and multiple sinks $t_1, \ldots t_q$ with the goal to transfer as much flow as possible from the sources to the sinks (i.e., find a flow $f: V \times V \to \mathbb{R}$ maximizing $\Sigma_{i=1}^{p} (\Sigma_{v \in V} f(s_i,v))$) 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 6.7: 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

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

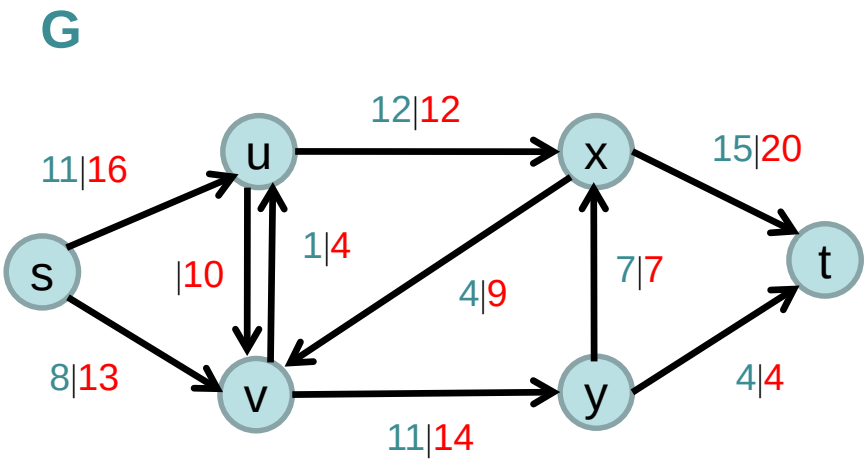
c) A simple path P from s to t in G_f is called an augmenting path.
The residual capacity $c_f(P)$ of P is defined as

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

Ford-Fulkerson Algorithm

Example: augmenting path and flow augmentation

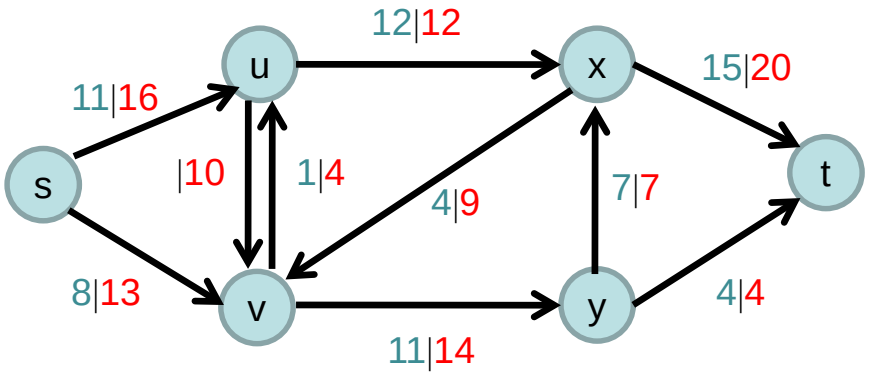
Flow network:



Example: augmenting path and flow augmentation

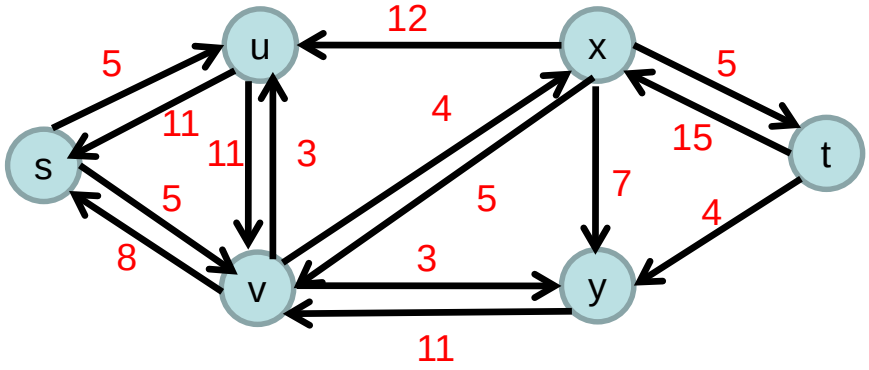
Flow network:

G



Residual network:

G_f

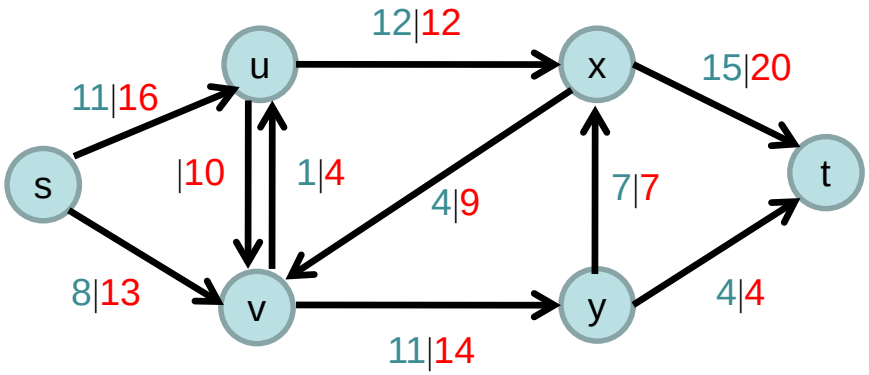


$c_f(u,v) = c\,(u,v) - f\,(u,v)$

Example: augmenting path and flow augmentation

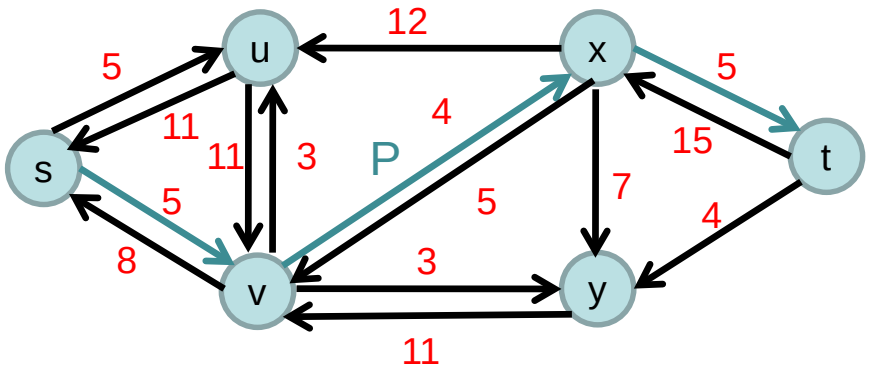
Flow network:

G



Residual network with augmenting path:

G_f

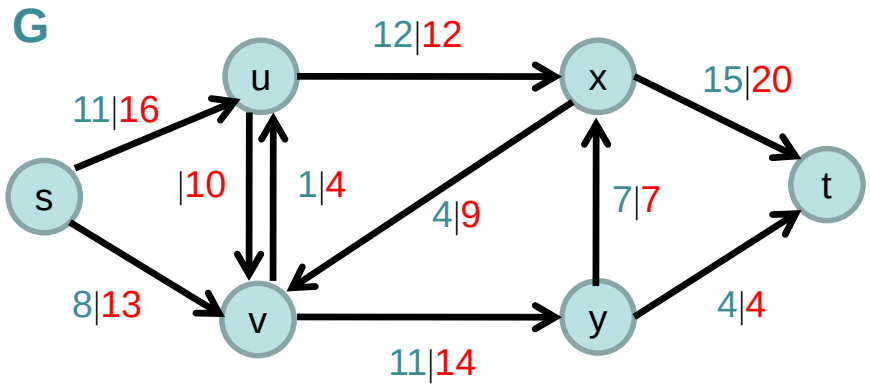


$c_f(P) = \min \{ c_f(u,v) \mid (u,v) \in P \}$

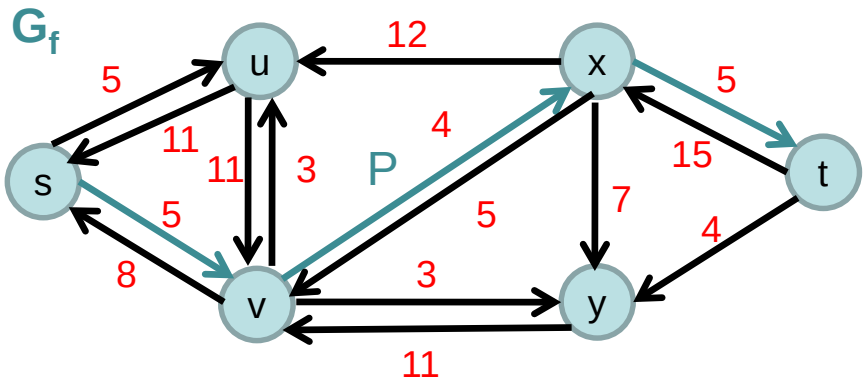
$\rightarrow$ residual capacity of path P: 4

Example: augmenting path and flow augmentation

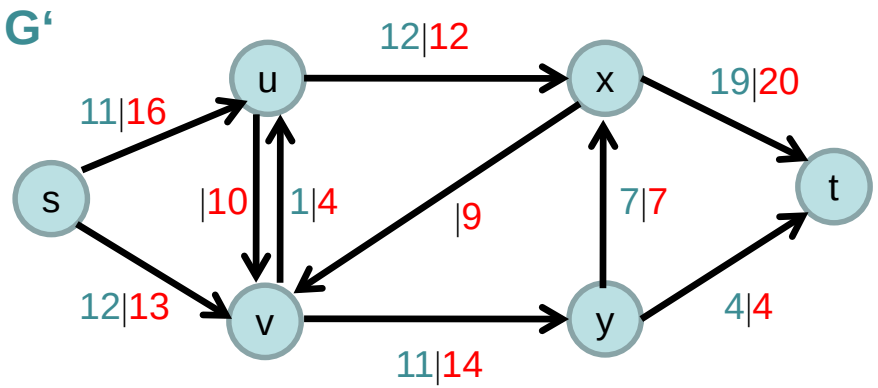
Flow network:



Residual network with augmenting path:



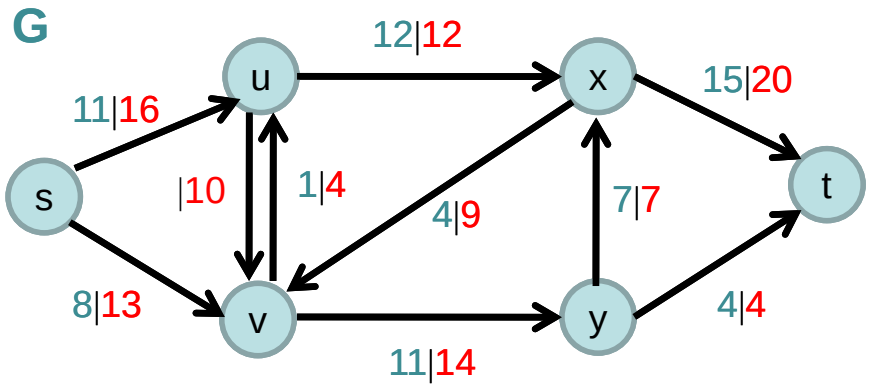
Augmented flow:



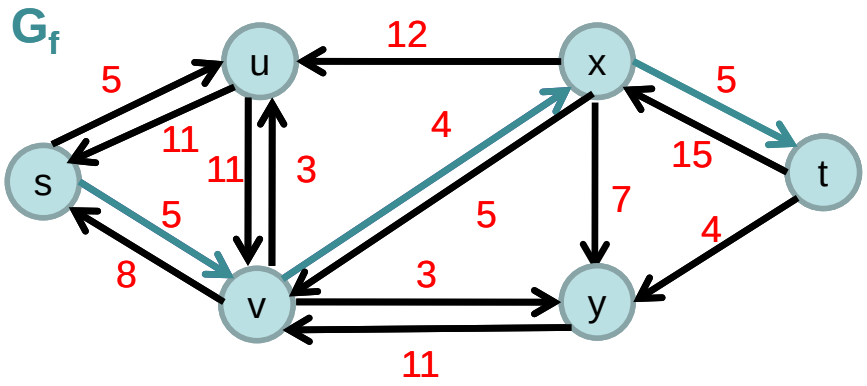
Path P of residual capacity 4 added to G:

Example: augmenting path and flow augmentation

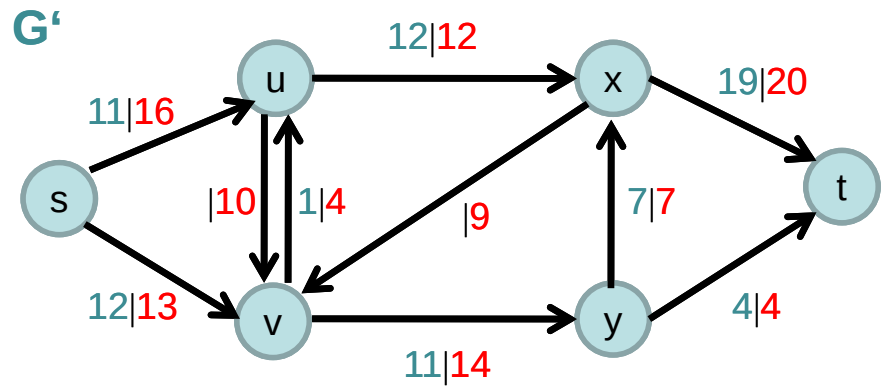
Flow network:



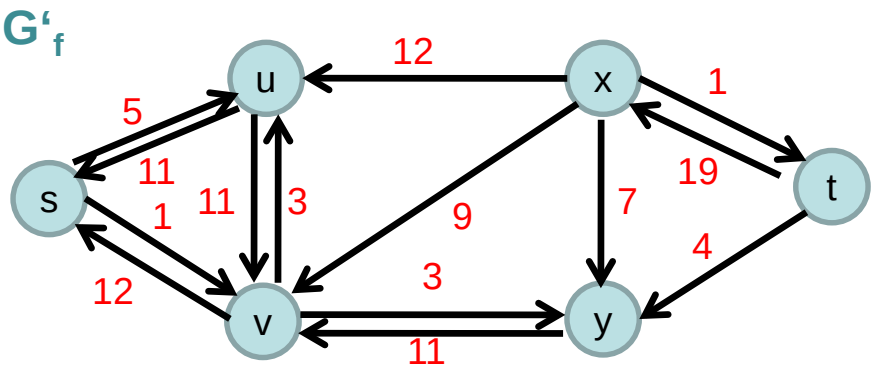
Residual network with augmenting path:



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 6.8: 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 $|f + f'| = |f| + |f'|$.

Proof:

- Capacity constraints:
  $f(u,v) \leq c(u,v)$ and $f'(u,v) \leq c_f(u,v) = c(u,v)-f(u,v)$ for all $u,v \in V$.
  Hence, $(f + f')(u, v) \leq f(u,v) + c(u,v)-f(u,v) \leq c(u,v)$.
- Skew symmetry:
  $f(u,v) = -f(v,u)$ and $f'(u,v) = -f'(v,u)$ for all $u,v \in V$.
  Hence, $(f + f')(u, v) = - (f + f')(v, u)$ for all $u,v \in V$.
- Flow conservation:
  $\Sigma_v f(u,v) = 0$ and $\Sigma_v f'(u,v) = 0$ for all $u \in V\{s,t\}$.
  Hence, $\Sigma_v (f + f')(u,v) = 0$ for all $u \in V\{s,t\}$.

Ford-Fulkerson Algorithm

How to define a flow for an augmenting path in G_f?

Lemma 6.9: 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 P be an augmenting path in G_f. Then $f_P : V \times V \to \mathbb{R}$ with

$$f_P(u,v) = \begin{cases} c_f(P) & \text{if } (u,v) \text{ belongs to } P \\ -c_f(P) & \text{if } (v,u) \text{ belongs to } P \\ 0 & \text{otherwise} \end{cases}$$

is a valid flow in G_f with value $|f_P| = c_f(P) > 0$.

Proof:

Check capacity constraints, skew symmetry and flow conservation.

Corollary 6.10: 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 P be an augmenting path in G_f. Let f_P be defined as in Lemma 6.9. Then $f' = f + f_P$ is a valid flow in G with value

$$|f'| = |f + f_P| = |f| + |f_P| > |f|.$$

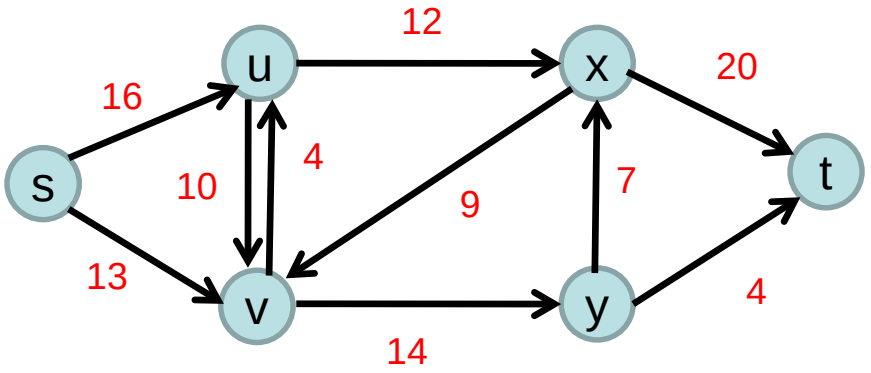
Ford-Fulkerson Algorithm

```
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:

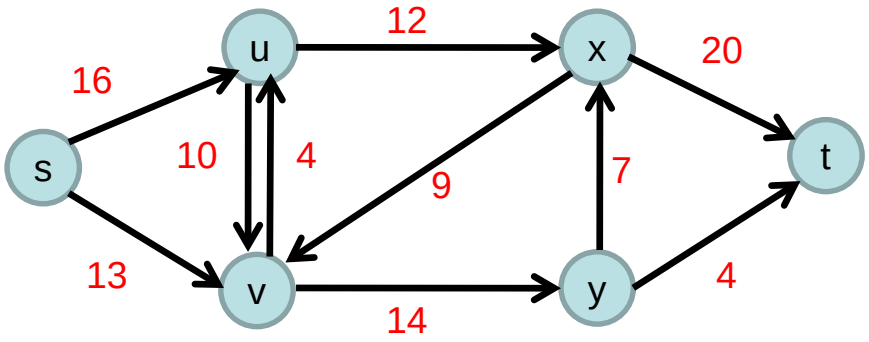
G



Example: Ford-Fulkerson Algorithm

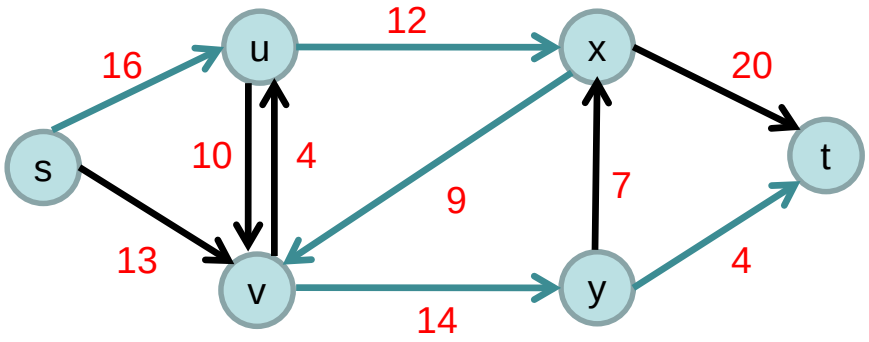
Flow network:

G



Residual network with augmenting path:

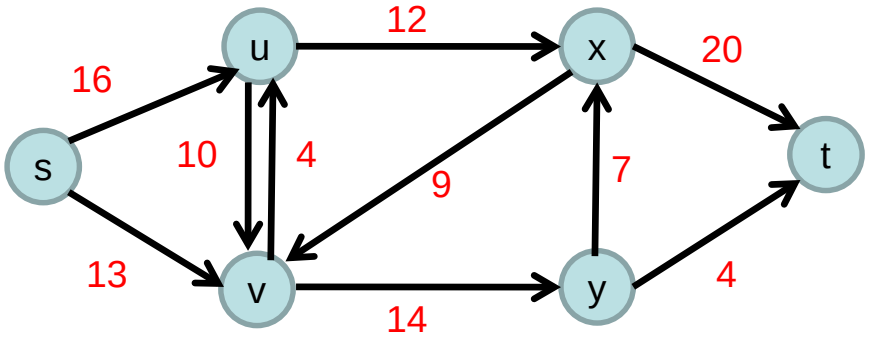
G_f



Example: Ford-Fulkerson Algorithm

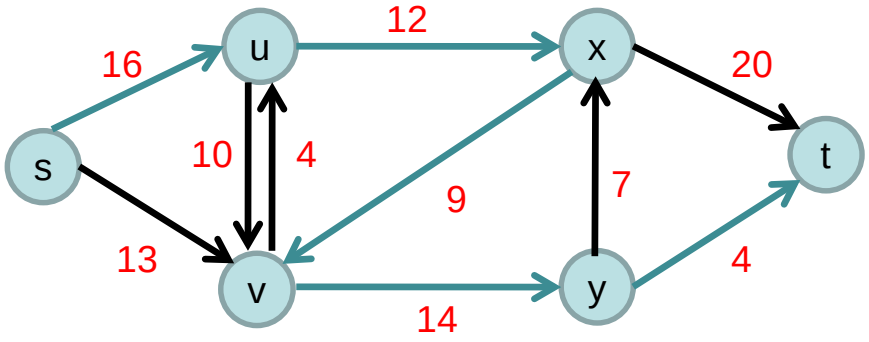
Flow network:

G



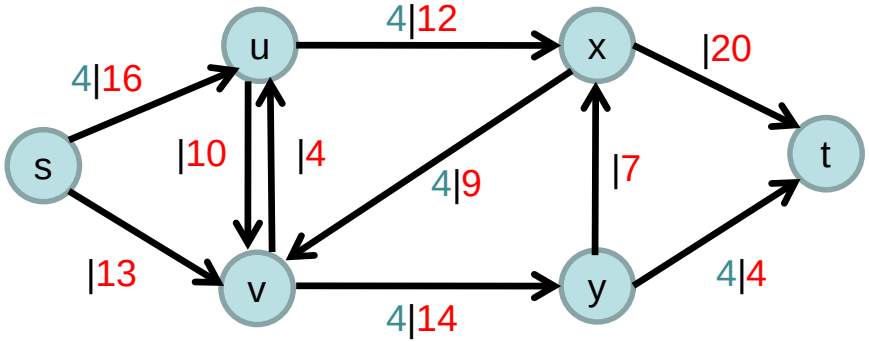
Residual network with augmenting path:

G_f



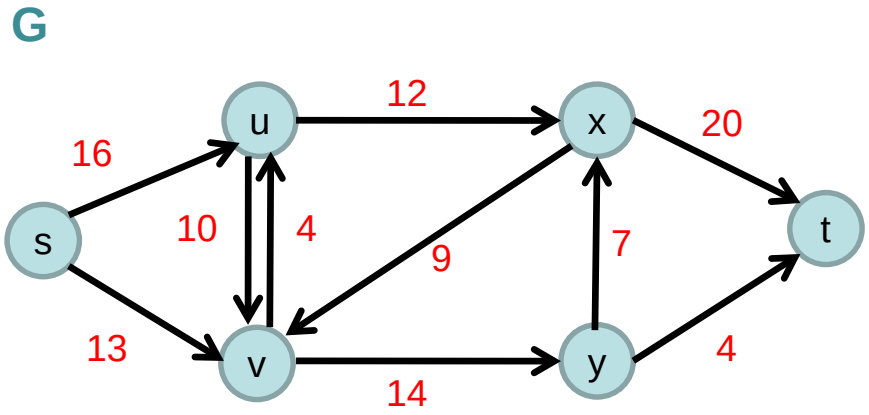
Augmented flow:

G

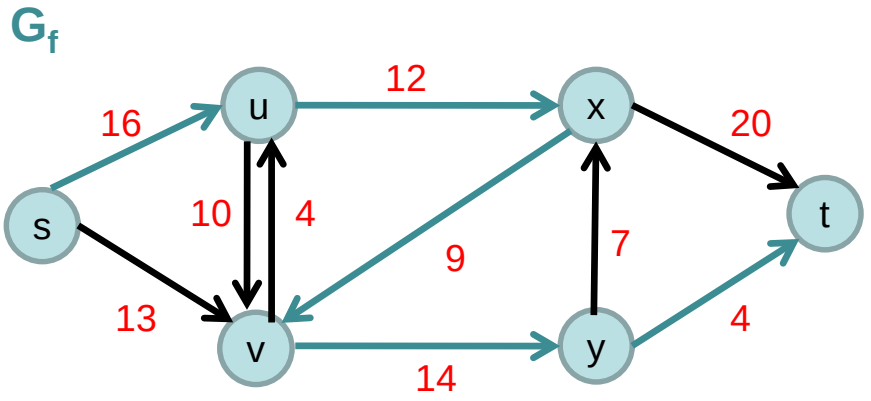


Example: Ford-Fulkerson Algorithm

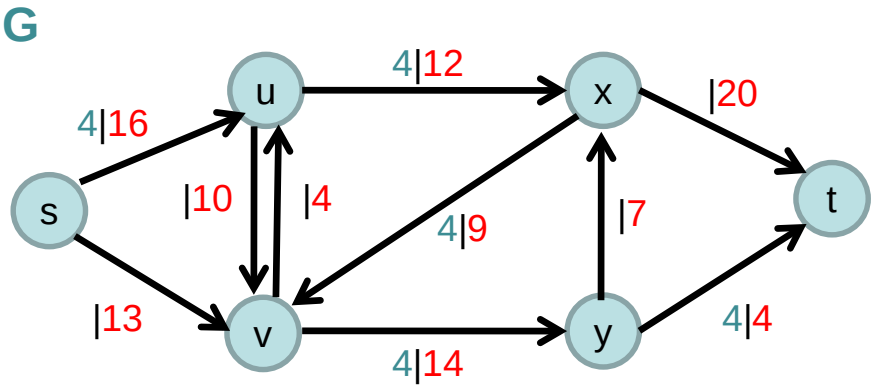
Flow network:



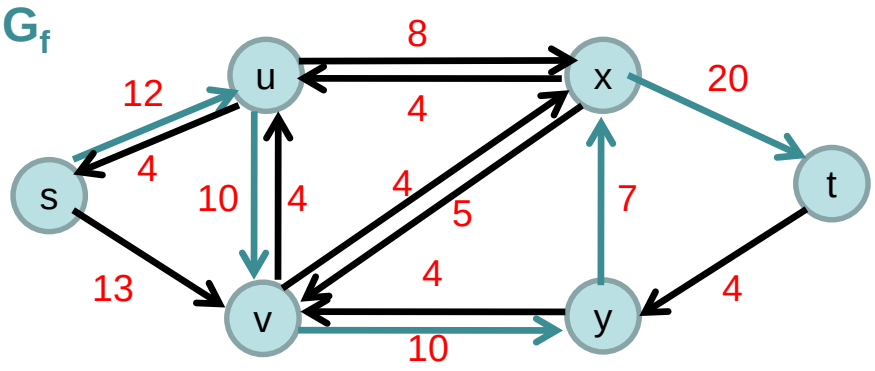
Residual network with augmenting path:



Augmented flow:



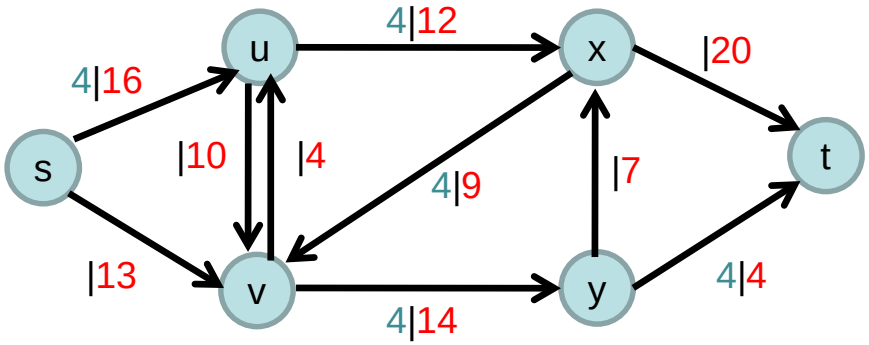
New residual network with augmenting path:



Example: Ford-Fulkerson Algorithm

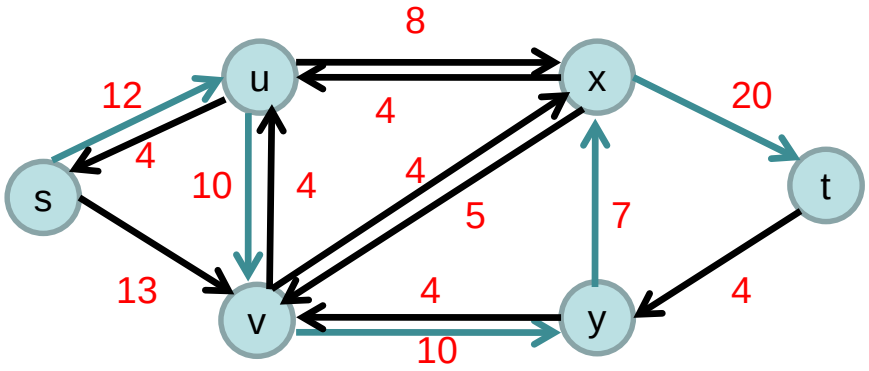
Flow network:

G



Residual network with augmenting path:

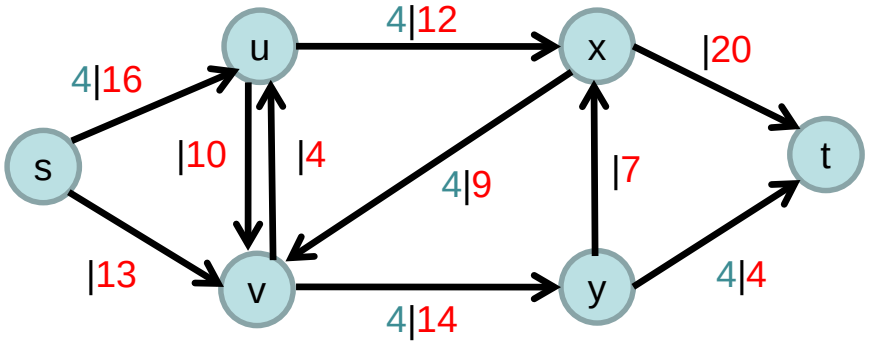
G_f



Example: Ford-Fulkerson Algorithm

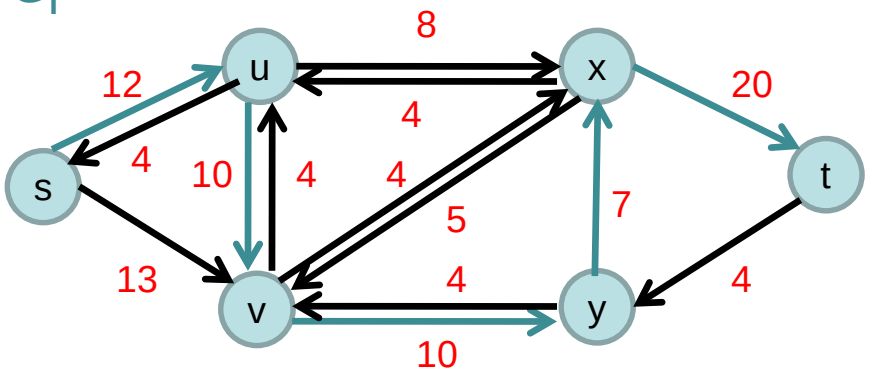
Flow network:

G



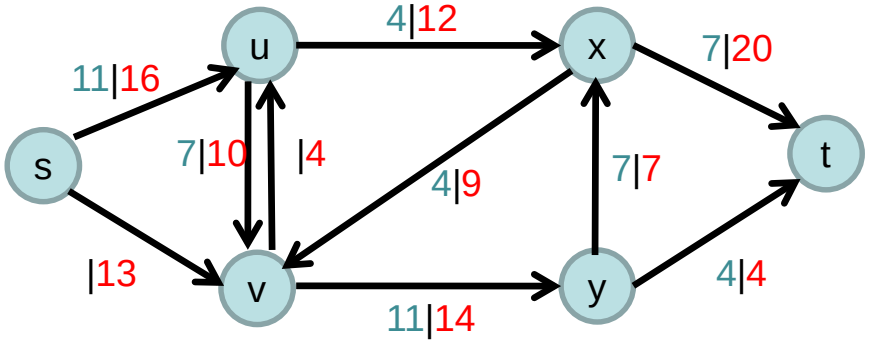
Residual network with augmenting path:

G_f



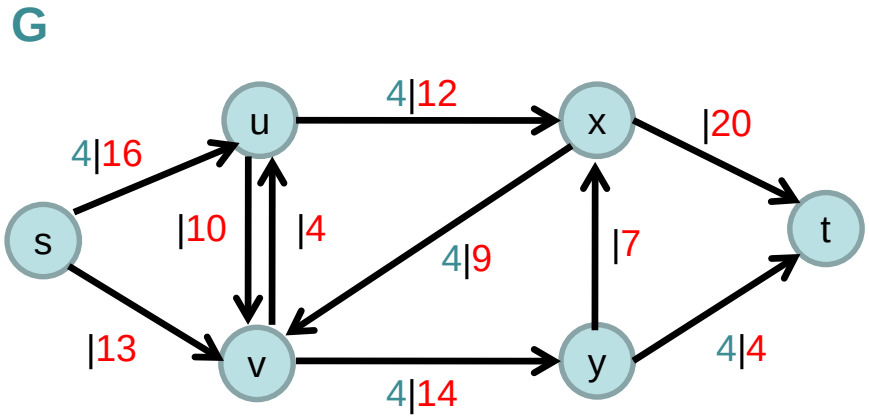
Augmented flow:

G

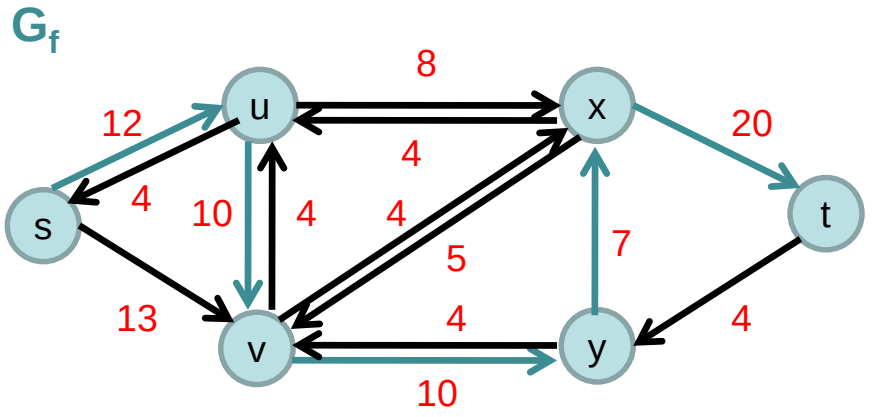


Example: Ford-Fulkerson Algorithm

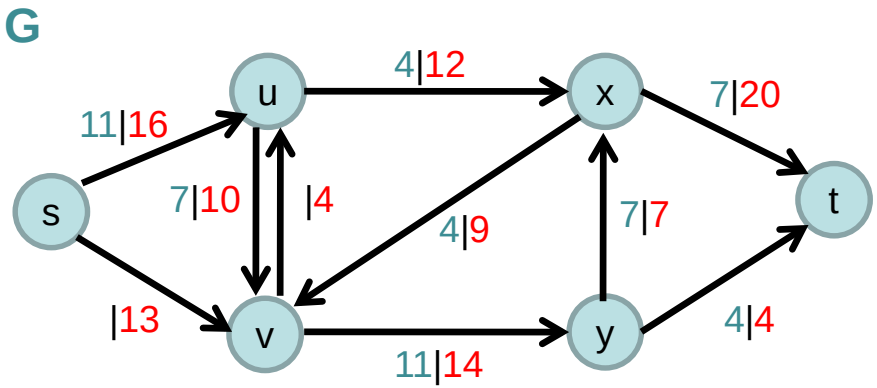
Flow network:



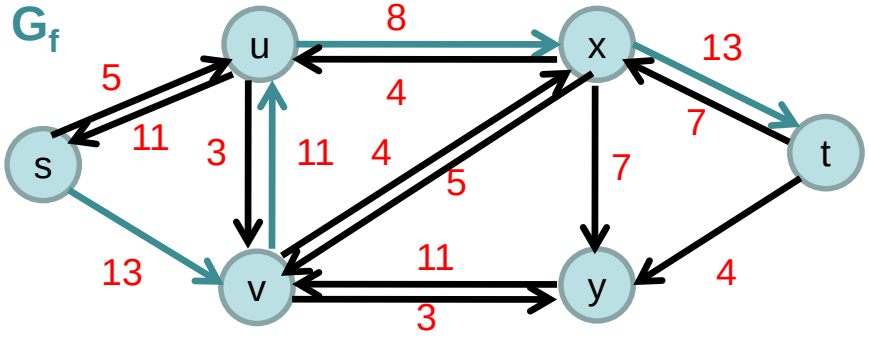
Residual network with augmenting path:



Augmented flow:



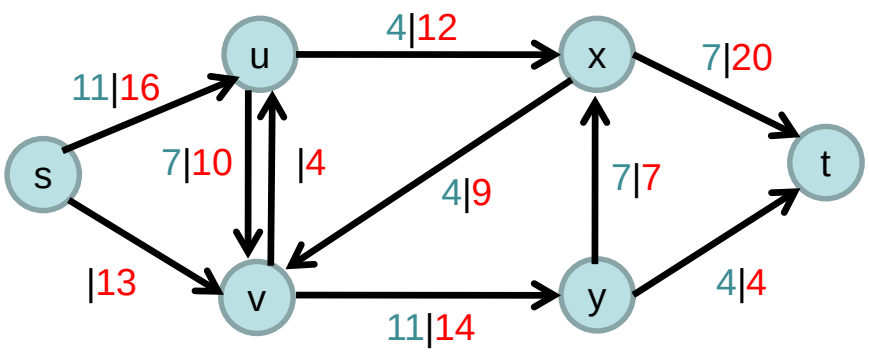
New residual network with augmenting path:



Example: Ford-Fulkerson Algorithm

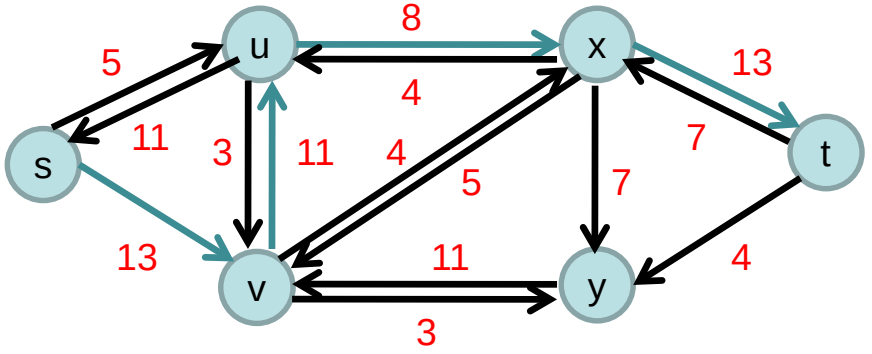
Flow network:

G



Residual network with augmenting path:

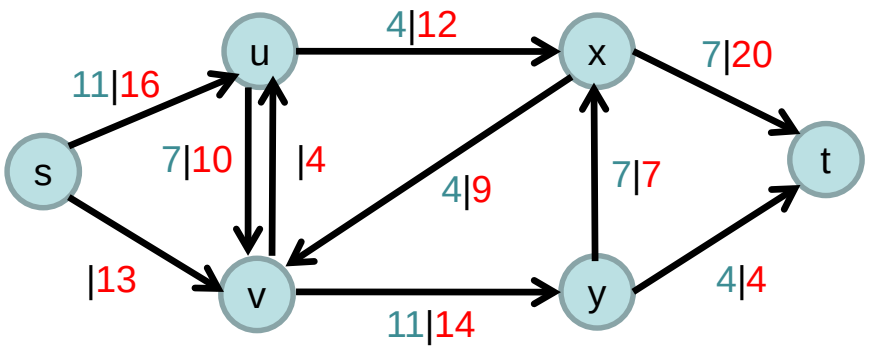
G_f



Example: Ford-Fulkerson Algorithm

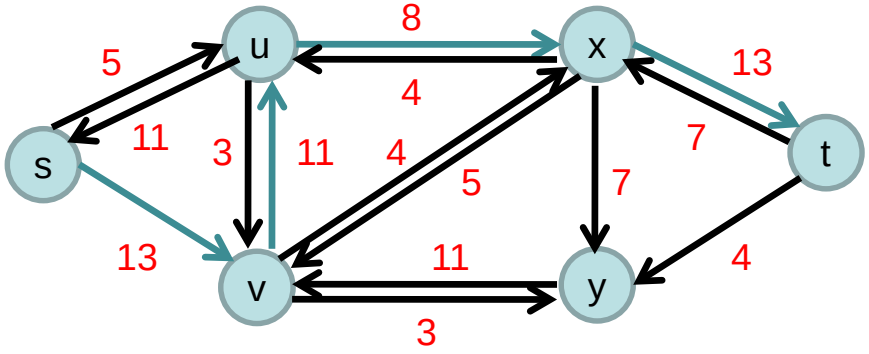
Flow network:

G



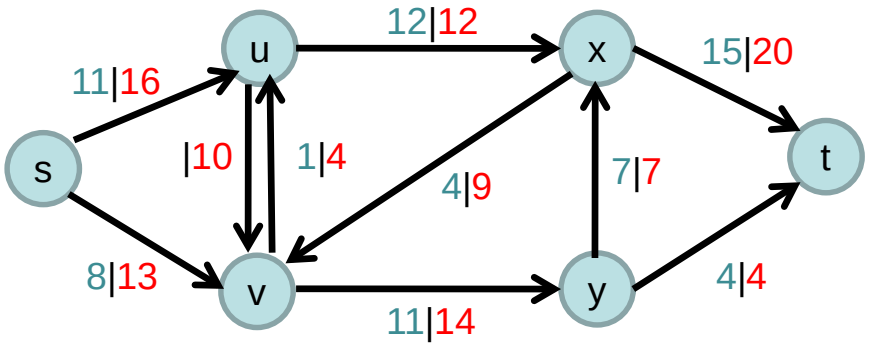
Residual network with augmenting path:

G_f



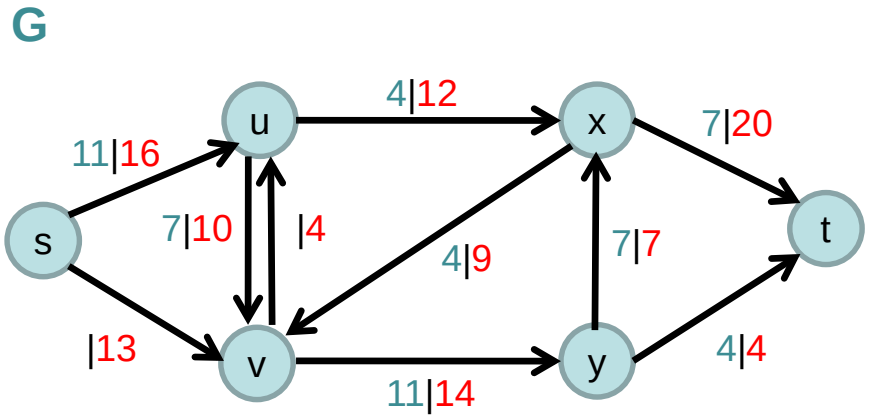
Augmented flow:

G

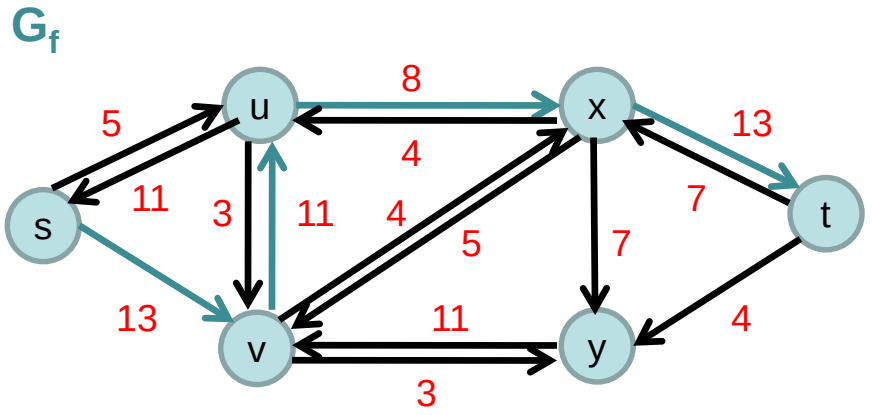


Example: Ford-Fulkerson Algorithm

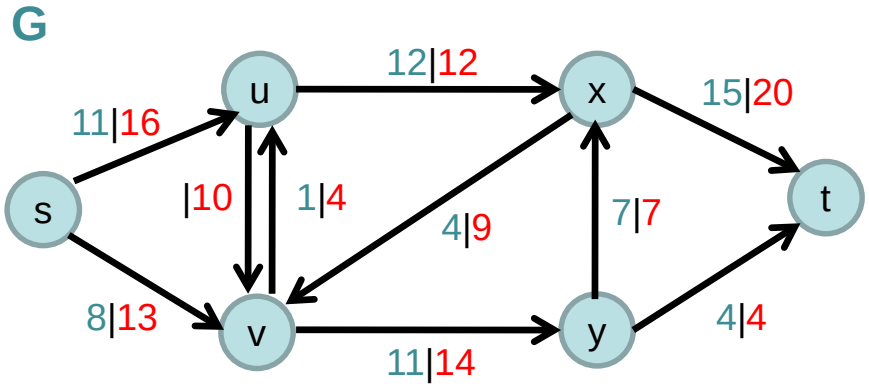
Flow network:



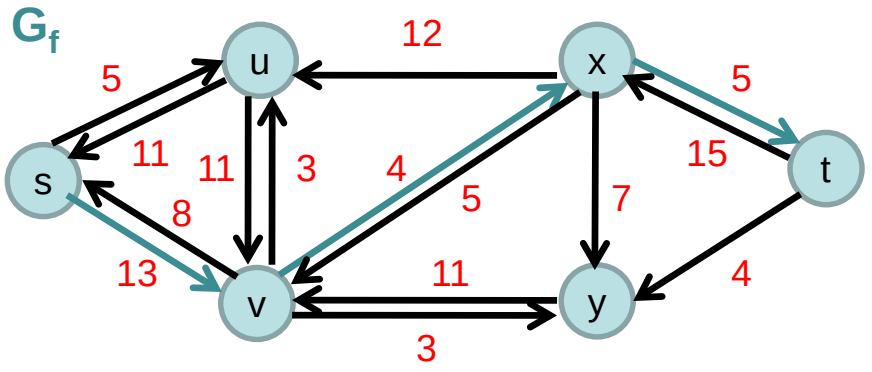
Residual network with augmenting path:



Augmented flow:



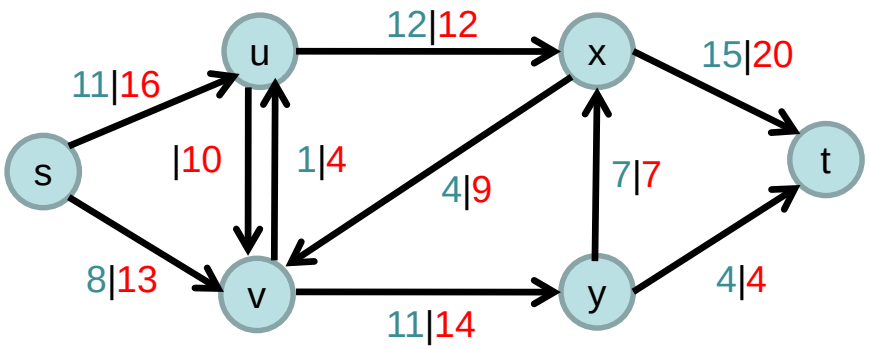
New residual network with augmenting path:



Example: Ford-Fulkerson Algorithm

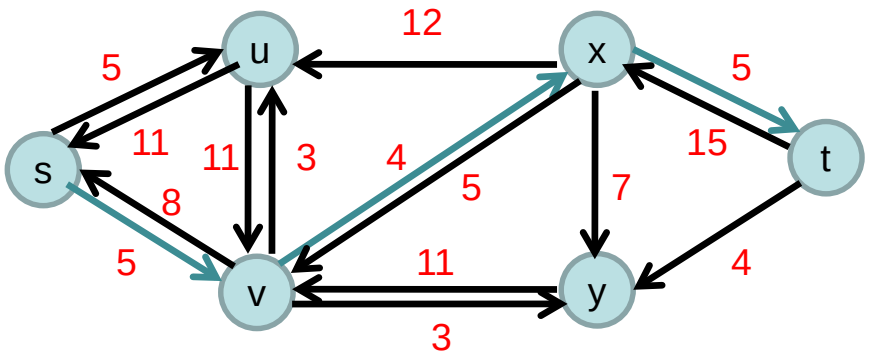
Flow network:

G



Residual network with augmenting path:

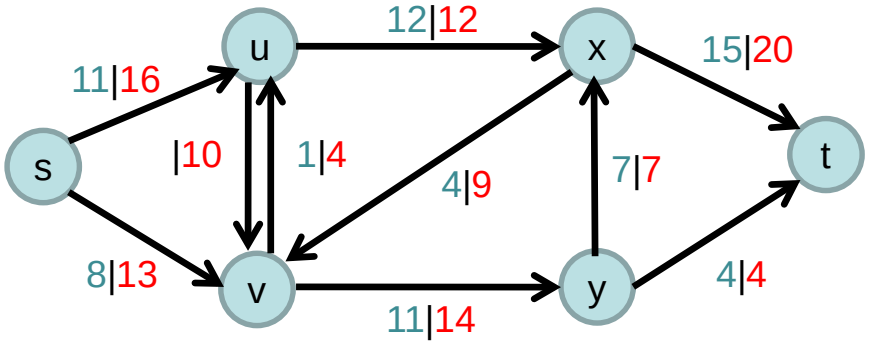
G_f



Example: Ford-Fulkerson Algorithm

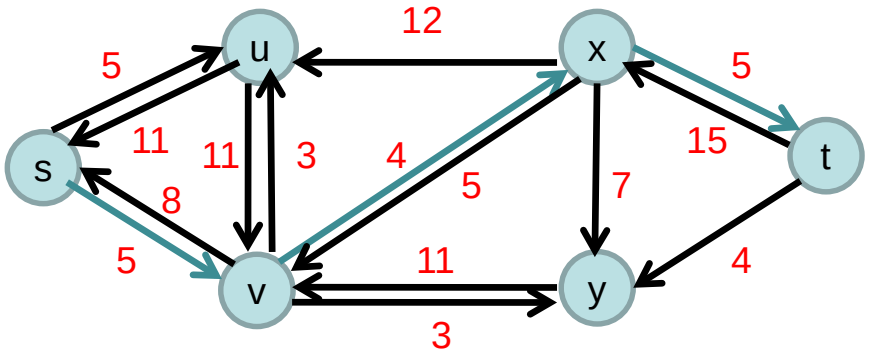
Flow network:

G



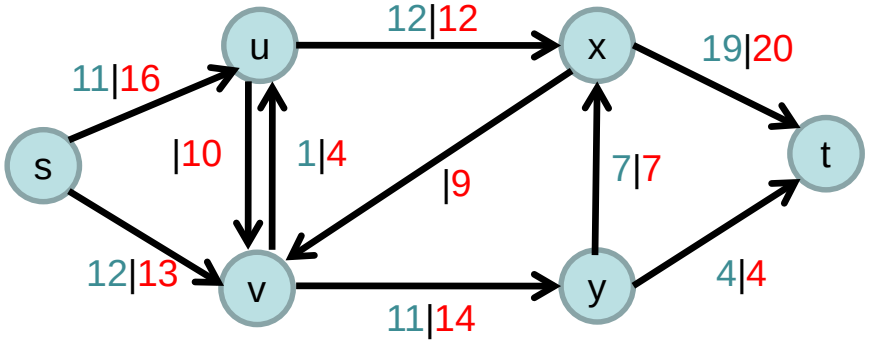
Residual network with augmenting path:

G_f



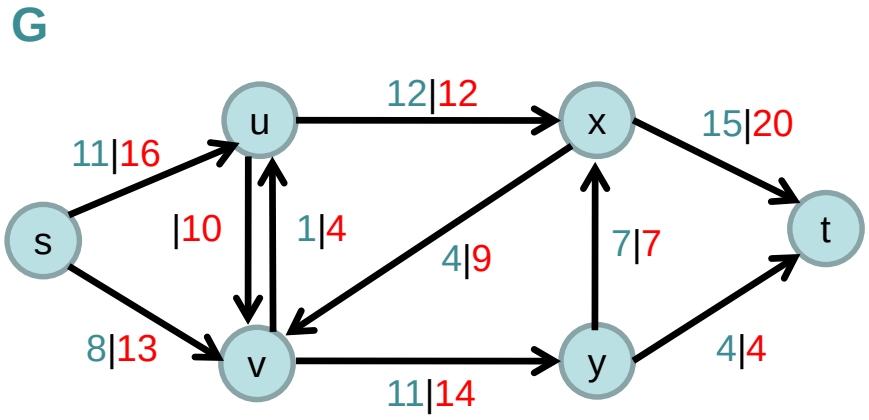
Augmented flow:

G

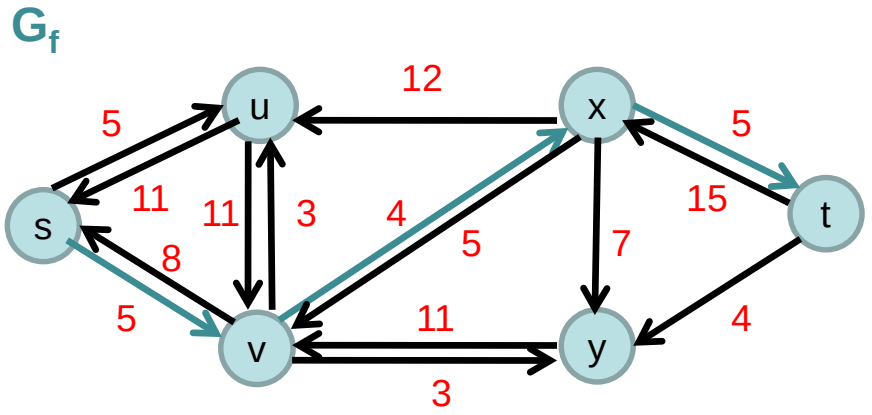


Example: Ford-Fulkerson Algorithm

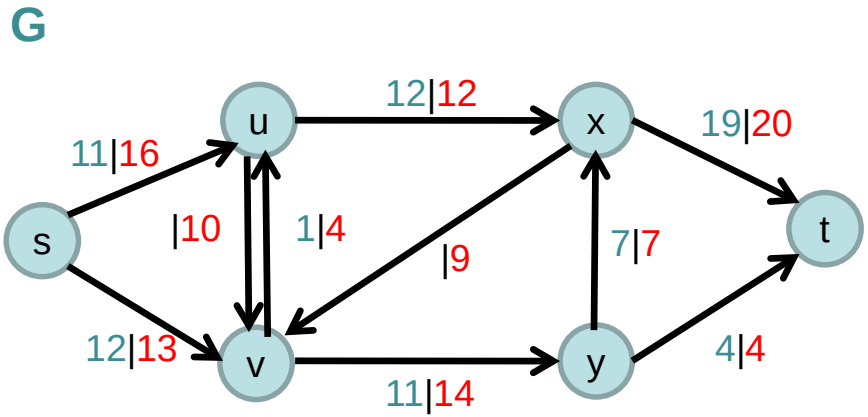
Flow network:



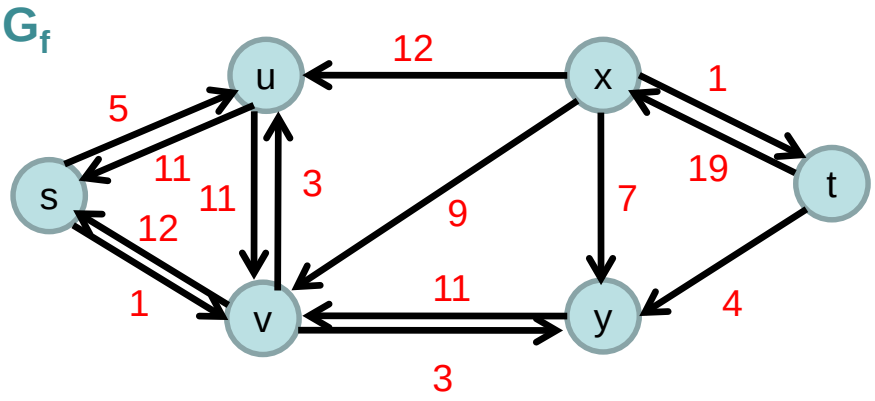
Residual network with augmenting path:



Augmented flow:

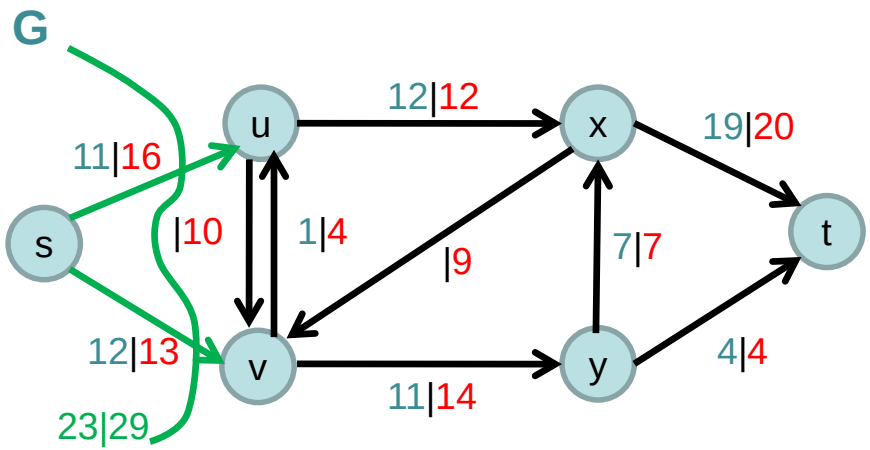


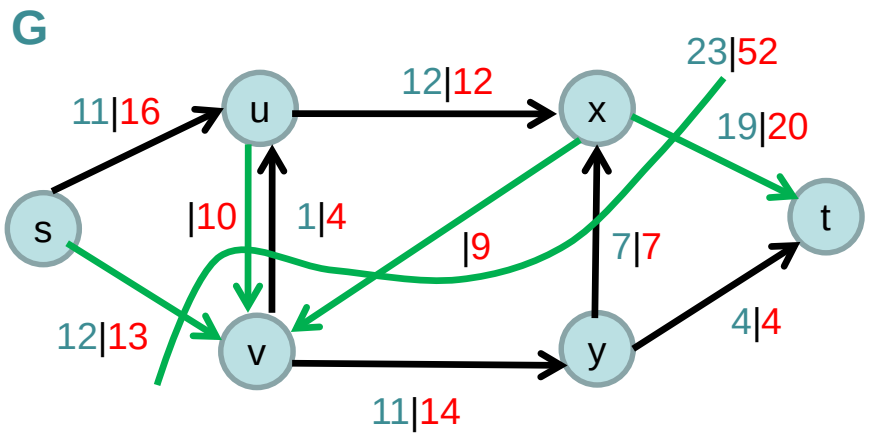
New residual network with augmenting path:



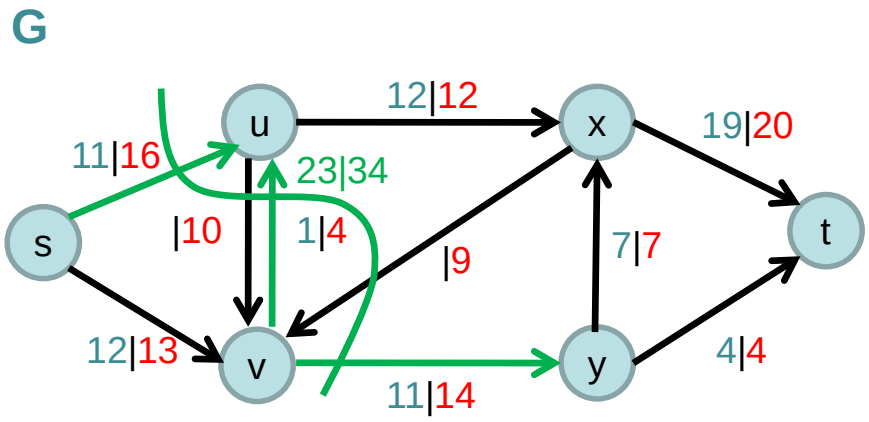
Example: Ford-Fulkerson Algorithm

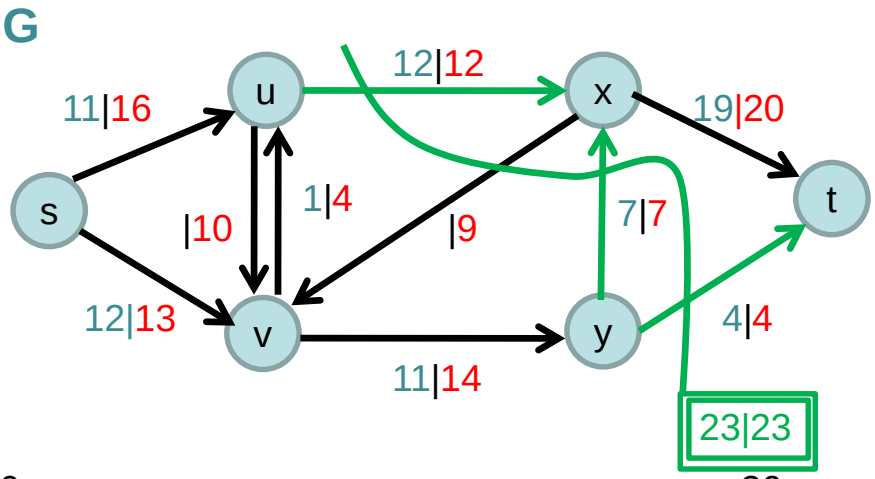
Flow network:





Augmented flow:

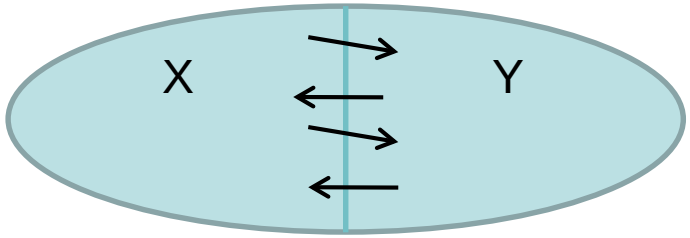




Do we always have a maximum flow f if G_f has no more paths from s to t?

Definition 6.11: Let (G,s,t,c) be a flow network. For $X, Y \subset V$ we define

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



Lemma 6.12: Let (G,s,t,c) be a flow network and let f a network flow in G.
Then it holds for all $X, Y, Z \subseteq V$:

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

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

c) If $X \cap Y = \emptyset$ then

$f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ and $f(Z, X \cup Y) = f(Z, X) + f(Z, Y)$

Proof: Exercise

Cuts in Flow Networks

Definition 6.13: Let (G, s, t, c) be a flow network and f be a flow in G.

a) A cut (S, T) of G is a partition of V into S and $T = V \setminus S$ so that $s \in S$ und $t \in T$.
b) The flow across a cut (S, T) is defined as $f(S, T)$.
c) The capacity of a cut (S, T) is defined as $c(S, T)$.

Remark 6.14:

a) The definition of a flow is consistent with the flows that were considered in the previous examples: flows from T to S are subtracted:

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

b) The definition of the capacity of a cut is consistent with the capacities that were considered in the previous examples: edges from T to S add no capacity to the cut:

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

Cuts in Flow Networks

Lemma 6.15: Let (G, s, t, c) be a flow network and f be a flow in G. Let (S, T) be a cut of G. Then

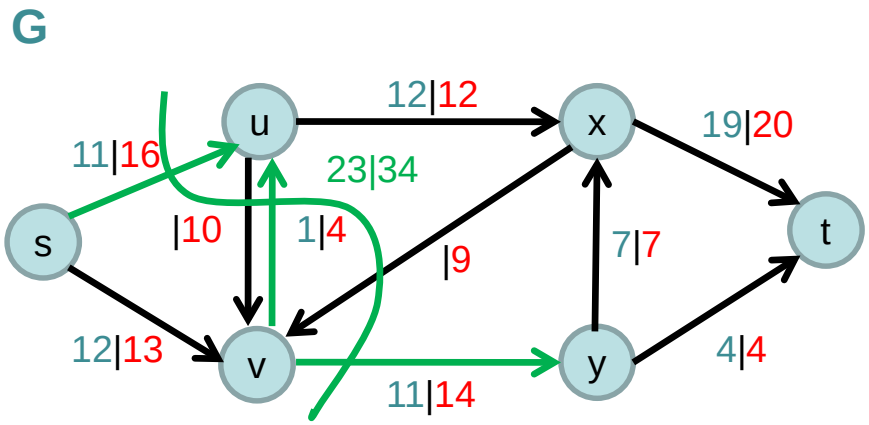
$$f(S, T) = |f|.$$

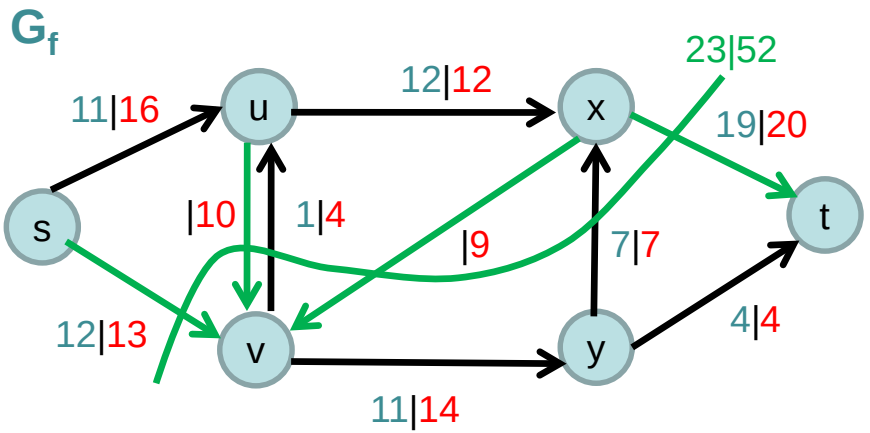
In particular, also

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

Proof: Exercise

Corollary 6.16: Let (G, s, t, c) be a flow network. Then the flow value of any flow f in G is upper bounded by the capacity of an arbitrary cut in G.





Cuts in Flow Networks

Theorem 6.17: (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.

Proof:

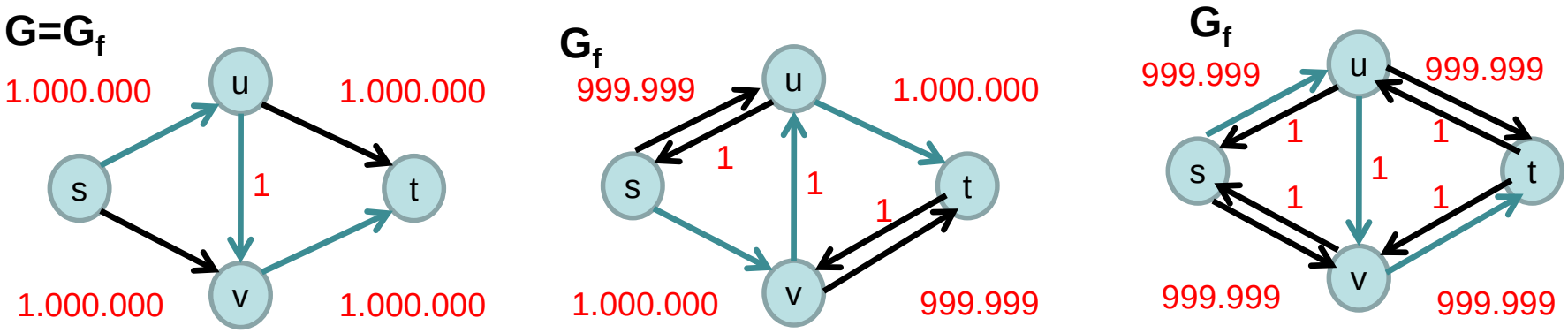
- a)⇒b): ¬b)⇒¬a) holds due to Corollary 6.10 and therefore also a)⇒b).
- b)⇒c): Let S be a set of nodes that are reachable from s in G_f. Then (S,T) with T=V\S is a cut and f(S,T)=c(S,T) according to the definition of G_f. Also, according to Lemma 6.15, |f|=c(S,T).
- c)⇒a): Follows from Corollary 6.16.

Corollary 6.18: Let (G, s, t, c) be a flow network with integer capacities c(u, v). Then the Ford-Fulkerson Algorithm computes a maximal flow f in time

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

Remark 6.19:

a) The bound on the runtime of FORDFULKERSON is sharp:

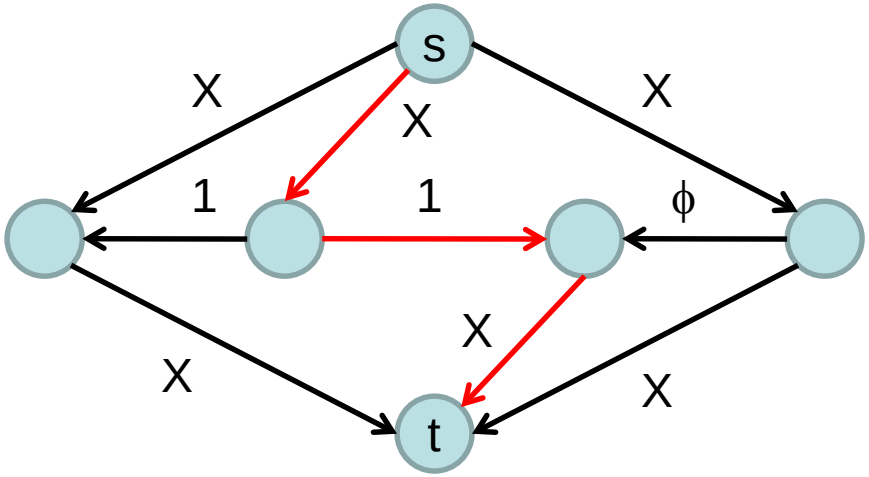


b) If the capacities are rational numbers, then they can be scaled to integer numbers, and FORDFULKERSON can be applied to the scaled network.

c) If the capacities are not rational numbers, then FORDFULKERSON may not terminate, and the flow f computed by FORDFULKERSON may not converge to the maximal flow.

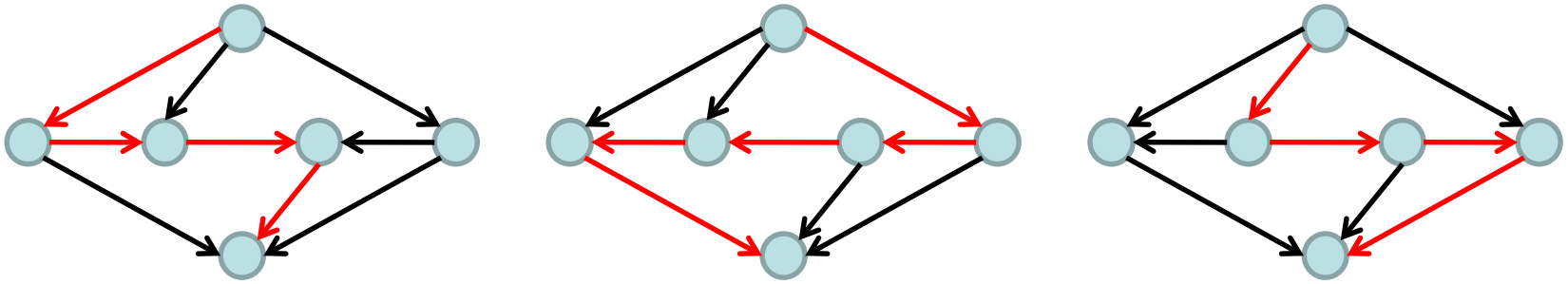
Problems with irrational Capacities

c): Let $\phi=(\sqrt{5}-1)/2 \approx 0{,}618034$ be chosen so that $1-\phi=\phi^2$. In order to show that the Ford-Fulkerson Algorithm gets stuck, consider the following graph (where $X \geq 4$):



- We start with an empty flow.
- After using the red path, the residual capacities of the horizontal edges are 1, 0 and ϕ.

Problems with irrational Capacities



Suppose that the residual capacities of the horizontal edges are ϕ^{k-1}, 0 and ϕ^{k} for some odd $k \in \mathbb{N}$.

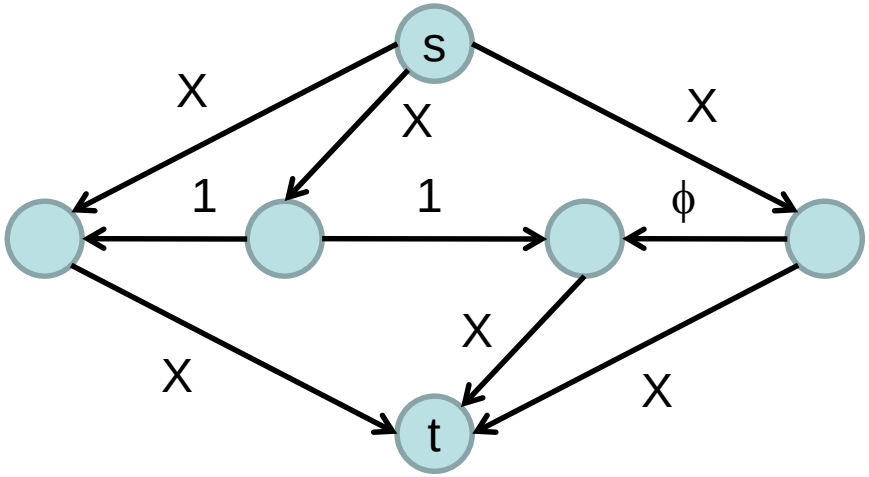
1. Augment along B, which adds ϕ^{k} to the flow. The residual capacities are now ϕ^{k+1}, ϕ^{k} and 0.
2. Augment along C, which adds ϕ^{k} to the flow. The residual capacities are now ϕ^{k+1}, 0 and ϕ^{k}.
3. Augment along B, which adds ϕ^{k+1} to the flow. The residual capacities are now 0, ϕ^{k+1} and ϕ^{k+2}.
4. Augment along A, which adds ϕ^{k+1} to the flow. The residual capacities are now ϕ^{k+1}, 0 and ϕ^{k+2}.

Problems with irrational Capacities

- That is, after $4n+1$ augmentations we arrive at residual capacities ϕ^{2n-2}, 0 and ϕ^{2n-1}.
- As the number of augmentations goes to ∞, the value of the flow converges to

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

  although the maximum flow value is $2X+1$.



Edmonds-Karp Algorithms

Problem: the Ford-Fulkerson Algorithm gives too much freedom to the choice of augmenting paths.

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

Proof:

- Let f^* be a maximal flow in G.
- Let f be an arbitrary flow in G and f' be a maximal flow in the residual network G_f. (Initially, f is empty and therefore $|f'|=|f^*|$.)
- Let e be the bottleneck edge in the augmenting path chosen by heuristic 1.
- $S \subseteq V$: set of nodes that can be reached from s along edges in G_f with residual capacity $>c_f(e)$.
- $T=V\backslash S$: is not empty due to heuristic 1 and the choice of e.
- It holds: $c_f(S,T) \leq c_f(e) \cdot |E|$ and $c_f(S,T) \geq |f'|$. Hence, $c_f(e) \geq |f'|/|E|$.
- Since $|f^*|=|f|+|f'|$, the value of f increases at least by a factor of $(1+1/|E|)$ as long as $|f| \leq |f^*|/2$.

Edmonds-Karp Algorithms

Theorem 6.20: 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 $O(|E|^2 \cdot \log |E| \cdot \log |f^*|)$.

Proof (continued):

- The value of f increases by a factor of at least $(1+1/|E|)$ as long as $|f| \leq |f^*|/2$.
- $(1+1/|E|)^k \geq |f^*|/2$ if $k \geq |E| \ln |f^*|$.
- Therefore, at most $|E| \ln |f^*|$ augmenting paths suffice to obtain a flow of value at least $|f^*|/2$.
- Refining this argument, it takes at most $|E|$ further augmenting paths to increase the flow value from $\geq (1-1/2^k)|f^*|$ to $\geq (1-1/2^{k+1})|f^*|$ for all k.
- Once $k = \lfloor \log |f^*| \rfloor + 1$, we have reached a flow value of $|f^*|$ since we are only dealing with integer values.
- Time to compute an augmenting path with maximal flow value: $O(|E| \log |E|)$. (This is an exercise.)
- Thus, the total runtime is $O(|E|^2 \cdot \log |E| \cdot \log |f^*|)$.

Edmonds-Karp Algorithms

Analysis of Heuristic 2:

- G_i: residual network after i augmenting steps, i.e., $G_0=G$.
- For a node v let $dist_i(v)$ be the distance (i.e., the number of edges along a shortest directed path) of v from s in G_i.
- No directed path from s to v: $dist_i(v)=\infty$.

Lemma 6.21: For every node v with $dist_i(v)=\infty$, also $dist_{i+1}(v)=\infty$.

Edmonds-Karp Algorithms

Lemma 6.21: For every node v with $dist_i(v)=\infty$, also $dist_{i+1}(v)=\infty$.

Proof:

- Consider an arbitrary node $v \in V$ with $dist_i(v)=\infty$.
- U: set of nodes that have a directed path to v in G_i.
- Then for all nodes $u \in U$, $dist_i(u)=\infty$.
- Suppose that $dist_{i+1}(v) \neq \infty$. Then an augmenting path must have been chosen in round i that goes through a node in U. (Why?)
- In this case, there must have been a directed path in G_i from s to a node in U, which contradicts the definition of U!

Edmonds-Karp Algorithms

Lemma 6.22: For every node $v \in V$ it holds that $dist_{i+1}(v) \geq dist_i(v)$.

Proof:

- $v=s$: trivial since $dist_i(s)=0$ for all i.
- $v \neq s$: induction on the distance from s.
- $p=(s,\ldots,u,v)$: shortest path from s to v in G_{i+1}. (No such path, then we are done according to Lemma 6.21.)
- Since this is a shortest path, $dist_{i+1}(u)=dist_{i+1}(v)-1$.
- According to the induction hypothesis, $dist_{i+1}(u) \geq dist_i(u)$.
- Case 1: (u,v) was an edge in G_i. Then $dist_i(v) \leq dist_i(u)+1$. Hence, $dist_{i+1}(v)=dist_{i+1}(u)+1 \geq dist_i(v)$.
- Case 2: (u,v) was not an edge in G_i. Then (v,u) belongs to the i-th augmenting path. In this case, (v,u) is on a shortest path from s in G_i and therefore, $dist_i(v) \leq dist_i(u)+1$.

Edmonds-Karp Algorithms

Lemma 6.23: During the execution of Heuristic 2, every edge (u,v) can disappear at most $|V|/2$ times from the residual graph.

Proof:

- Suppose that (u,v) is in the residual graphs G_i and G_{j+1} but not in the residual graphs $G_{i+1},\ldots,G_j$.
- Then (u,v) must be in the i-th augmenting path, and therefore, $dist_i(v)=dist_i(u)+1$.
- Moreover, (v,u) must be in the j-th augmenting path, and therefore, $dist_j(u)=dist_j(v)+1$.
- Together with Lemma 6.22 it follows that

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

- Since $|V|-1$ is an upper bound on the largest finite distance of a node, (u,v) can disappear at most $|V|/2$ times.

Edmonds-Karp Algorithms

Now we are ready to prove a runtime bound for Heuristic 2.

- Since every edge can disappear at most $|V|/2$ times from the residual network, there are at most $|E| \cdot |V|/2$ events in which an edge disappears.
- But at least one edge disappears in each iteration, which implies that Heuristic 2 runs for at most $|E| \cdot |V|/2$ iterations.
- Since a shortest augmenting path can be computed in time $O(|E|)$ (using breadth first search), we get:

Theorem 6.24: Let (G, s, t, c) be a flow network with integer capacities $c(u, v)$. Then Heuristic 2 computes a maximal flow in time $O(|E|^2 \cdot |V|)$.

Dinic´s Algorithm

- The runtime of Heuristic 2 does not depend any more on the value of the maximum flow, but it is still too large.
- In the following we will present Dinic´s Algorithm, which only needs $O(|V|^2 |E|)$ time.

Definition 6.25: A flow f in a flow network (G, s, t, c) is called blocking if every path from s to t contains at least one saturated edge. An edge e is called saturated if $f(e)=c(e)$.

Remark: Not every blocking flow is also maximal, but every maximal flow is blocking. (Exercise!)

Dinic´s Algorithm

Definition 6.26:

- The level of a node v is defined as $level(v)=\delta_f(s,v)$ (the number of edges along a shortest path in G_f from s to v).
- The level graph L_f is a subgraph of G_f that contains all edges (u,v) with $(u,v)\in G_f$ and $level(u)=level(v)-1$.

Lemma 6.27: L_f contains all shortest augmenting paths and can be constructed in $O(m)$ time (e.g., when using BFS).

Dinic´s Algorithm

Dinic´s Algorithm:

start with an empty flow f
repeat
    find a blocking flow f' in L_f
    set $f:=f+f'$
until sink t is not reachable in L_f

Lemma 6.28: Dinic´s Algorithm stops after at most $n-1$ iterations of the repeat-loop.

Dinic´s Algorithm

Lemma 6.28: Dinic´s Algorithm stops after at most n-1 iterations of the repeat-loop.

Proof:

- Consider some fixed iteration and let
  - f and level denote the flow and levels at the beginning and
  - f´ and level´ denote the flow and levels at the end of the iteration.
- An edge (v,w) in $G_{f'}$ is either
  - an edge in G_f (if the edge has not been saturated or used) or
  - a reverse edge in L_f (if it was not in G_f, i.e., (w,v) must have been used).
- Thus, for every edge $(v,w) \in G_{f'}$, $level(w) \leq level(v)+1$.
- Hence, $level'(t) \geq level(t)$. Consider for this a path in $L_{f'}$:

- It even holds that $level'(t) > level(t)$, as we will see.

Dinic´s Algorithm

Lemma 6.28: Dinic´s Algorithm stops after at most $n-1$ iterations of the repeat-loop.

Proof (continued):

- Suppose that $level'(t)=level(t)$. Let p be any shortest path from s to t in $G_{f'}$.
- For every edge $(v,w)\in p$ we already know that $level(w)\leq level(v)+1$. So it must hold that $level(w)=level(v)+1$ since otherwise $level'(t)<level(t)$.
- If (v,w) was not an edge in G_f, then (w,v) must have been used by the blocking flow, which would mean that $level(w)<level(v)+1$.
- Thus, (v,w) must have also been an edge in G_f, which implies that p must have been a shortest path in G_f .
- Every edge in p is therefore in L_f, and none of these was saturated.
- That, however, contradicts our assumption that we have chosen a blocking flow in L_f.
- Therefore, in each iteration of the repeat-loop, the distance between s and t in G_f increases by at least 1.
- Since a shortest path from s to t cannot be longer than $n-1$ (if such a path exists), the lemma follows.

Dinic´s Algorithm

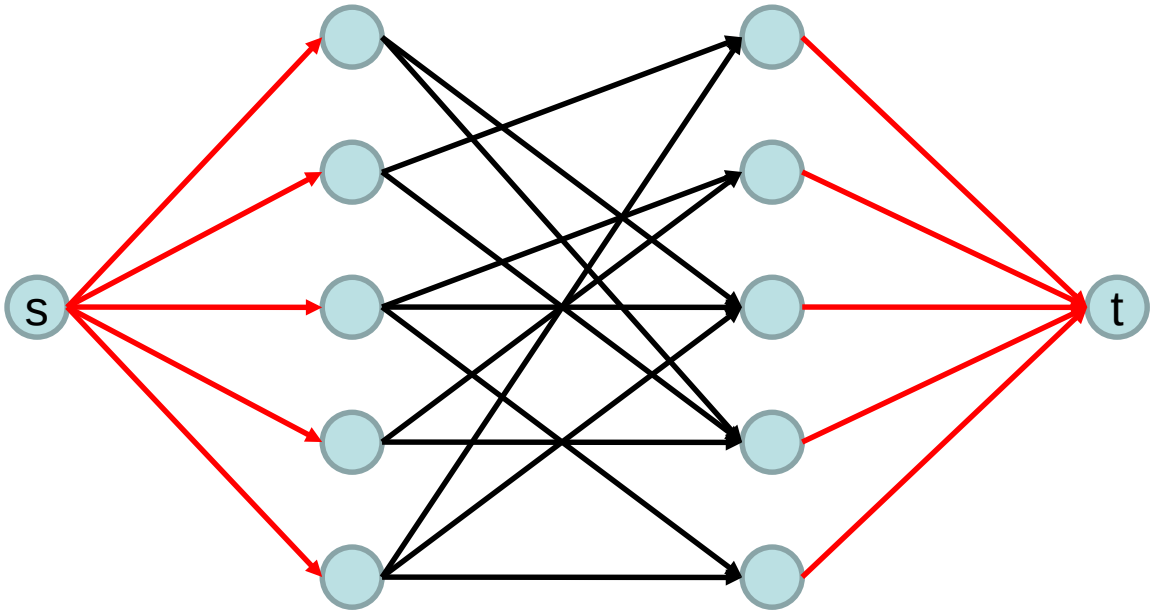
In unit networks, fewer iterations are needed.

Definition 6.29: A unit network is a flow network (G,s,t,c) with integer capacities in which every node $v \in V \setminus \{s,t\}$ has either concerning the incoming or the outgoing edges exactly one edge, and this edge has a capacity of 1.

Remark: If a node has exactly one incoming edge of capacity 1, it can still have many outgoing edges (and vice versa).

Dinic´s Algorithm

Example of a unit network:



: capacity 1 : arbitrary integer capacity

Dinic´s Algorithm

Lemma 6.30: In a unit network, Dinic´s Algorithm stops after at most $2\sqrt{n-2}$ iterations.

Proof:

- Let us consider a fixed iteration.
- Let f be the current flow and f^* be a maximal flow, which have both integer flow values. We assume here that Def. 6.5 applies.
- Then f^*-f is a flow of integer value in G_f.
- Since G is a unit network, $f^*(e)-f(e)$ has a value of $\{-1,0,1\}$ at every edge e.
- We partition the edges e with $f^*(e)-f(e)=1$ into a collection of paths from s to t.
- There are exactly $|f^*|-|f|$ paths from s to t. (proof: exercise)
- These paths are node-disjoint (except for s and t).
- Hence, there is an augmenting path with at most $(n-2)/(|f^*|-|f|)+1$ nodes.
- After $\sqrt{n-2}$ iterations, a shortest augmenting path must contain at least $\sqrt{n-2}+1$ nodes (according to Lemma 6.28, the distance of t from s increases).
- It holds that $\sqrt{n-2} + 1 \leq (n-2)/(|f^*|-|f|)+1 \quad \Leftrightarrow \quad |f^*|-|f| \leq \sqrt{n-2}$.
- Hence, after at most $\sqrt{n-2}$ further iterations we obtain a maximal flow.

Dinic´s Algorithm

How can we find a blocking flow?

Repeatedly use DFS:

repeat
  find a path p from s to t in L_f via DFS and send a flow value of $c_f(p)$ along p

until there is no augmenting path left in L_f

Lemma 6.31: The time needed to compute a blocking flow is $O(n \cdot m)$. (Exercise)

Theorem 6.32: The runtime of Dinic´s Algorithm is $O(n^2 \cdot m)$.

Dinic´s Algorithm

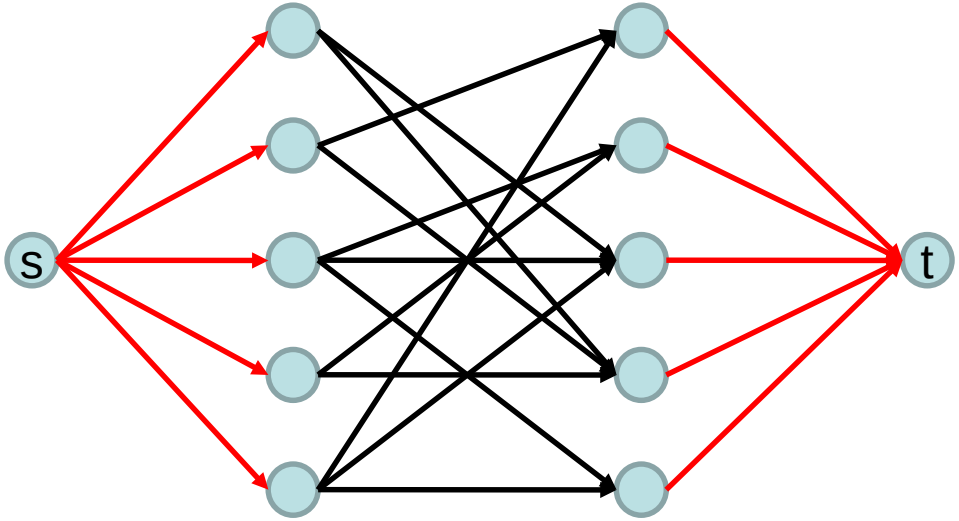
Theorem 6.33: The runtime of Dinic´s algorithm on unit networks is $O(\sqrt{n} \cdot m)$.

Proof:

- When searching for a blocking flow, every edge of the unit network has to be visited at most once since it can only lie on at most one augmenting path.
- Thus, a blocking flow can be found in $O(m)$ time.
- Together with Lemma 6.30 we obtain the runtime bound in the theorem.

Dinic´s Algorithm

Application of Dinic´s Algorithm: maximum matching in bipartite graphs.

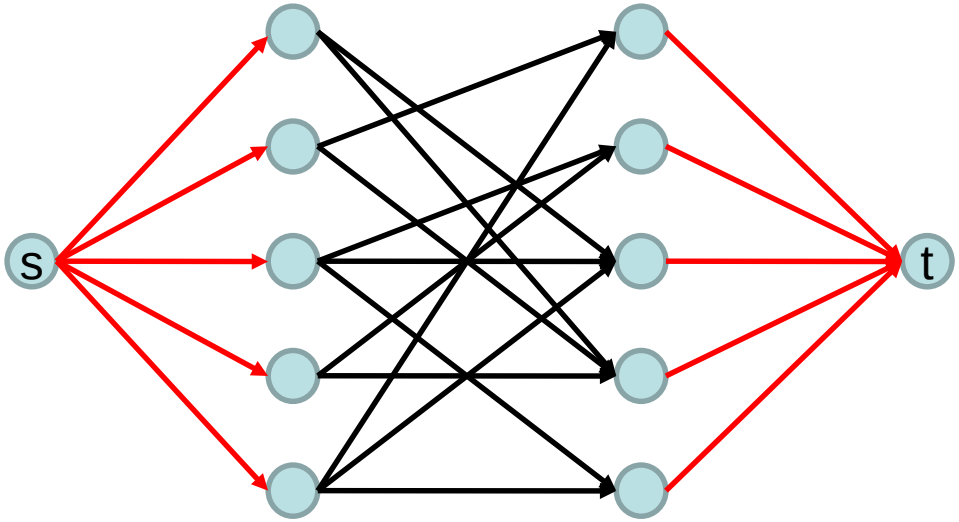


Theorem 6.34: Dinic´s Algorithm on bipartite graphs $G=(V,E)$ extended by a source s and sink t computes a maximum flow f in time $O(\sqrt{n} \cdot m)$, so that $|f|$ is the size of a maximum matching in G.

Dinic´s Algorithm

Proof:

$|f| \geq |M|$:

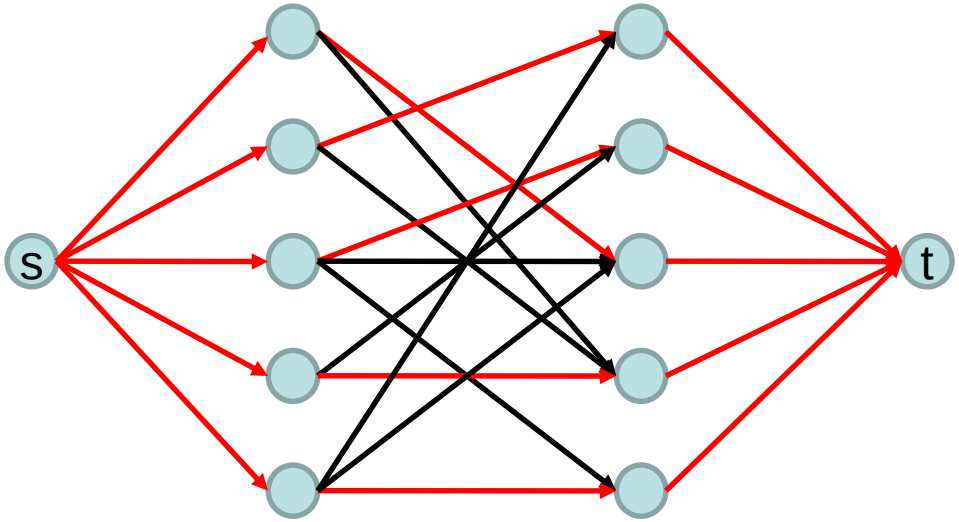


- Let M be a maximum matching in G.
- Then the flow f´ that uses M and all edges of s and t to M, is a legal flow of value $|f'|=|M|$ and therefore, $|f| \geq |M|$.

Dinic´s Algorithm

Proof:

$|M| \geq |f|$:

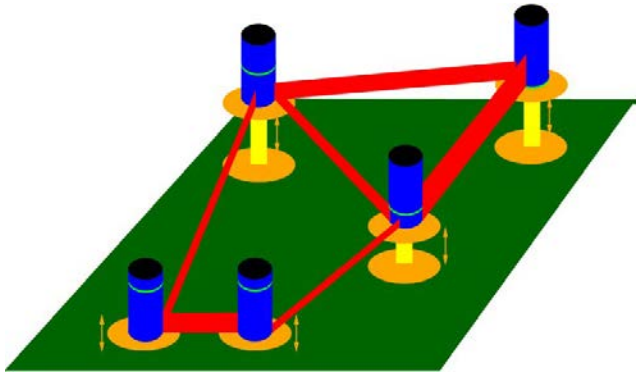


- Let f be a maximum flow in G extended by s and t.
- Then the set of edges $M´$ that f traverses in G is a matching of size $|M´|=|f|$ and therefore, $|M| \geq |f|$.

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.



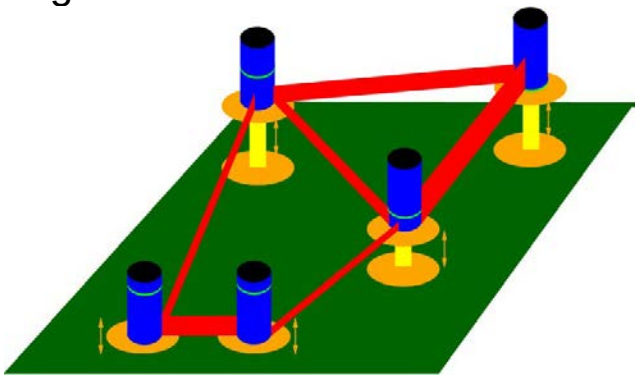
- Every node has a reservoir that can collect an arbitrary amount of liquid.

- 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 6.35: Let (G,s,t,c) be a flow network. A preflow is a function $f:V\times V\to\mathbb{R}$ 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.

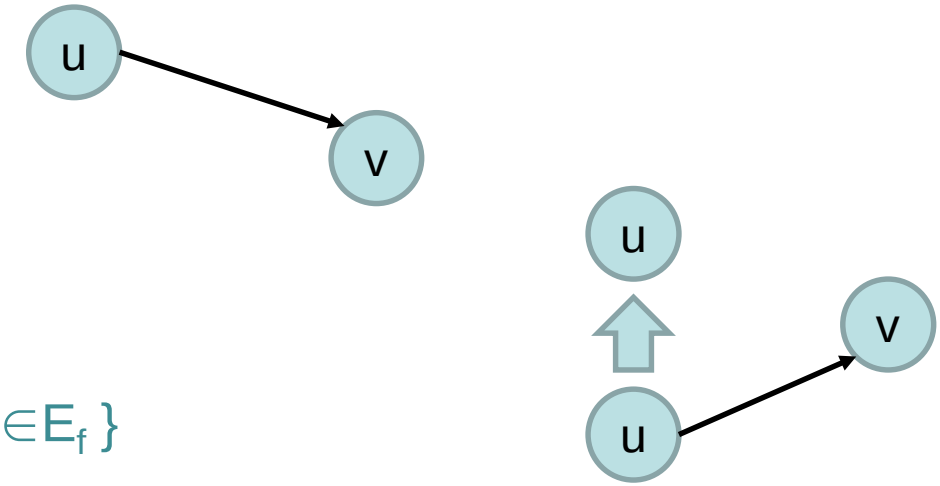
In pseudocode:

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):

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



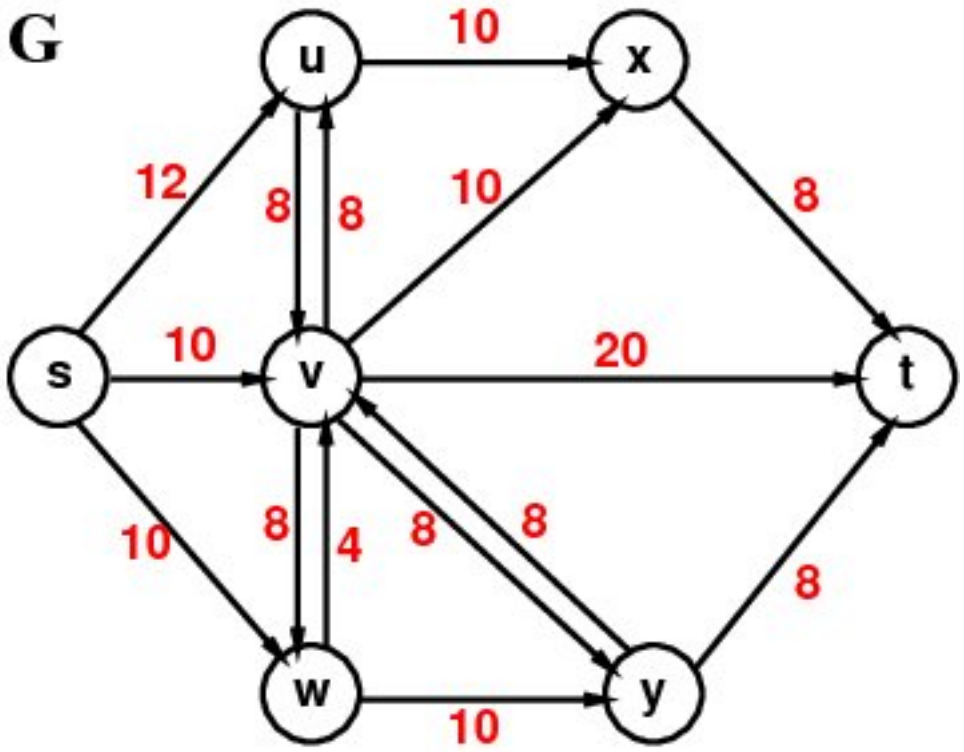
Goldberg´s Algorithm

Goldberg´s Algorithm works as follows:

Preflow-Push Algorithm:

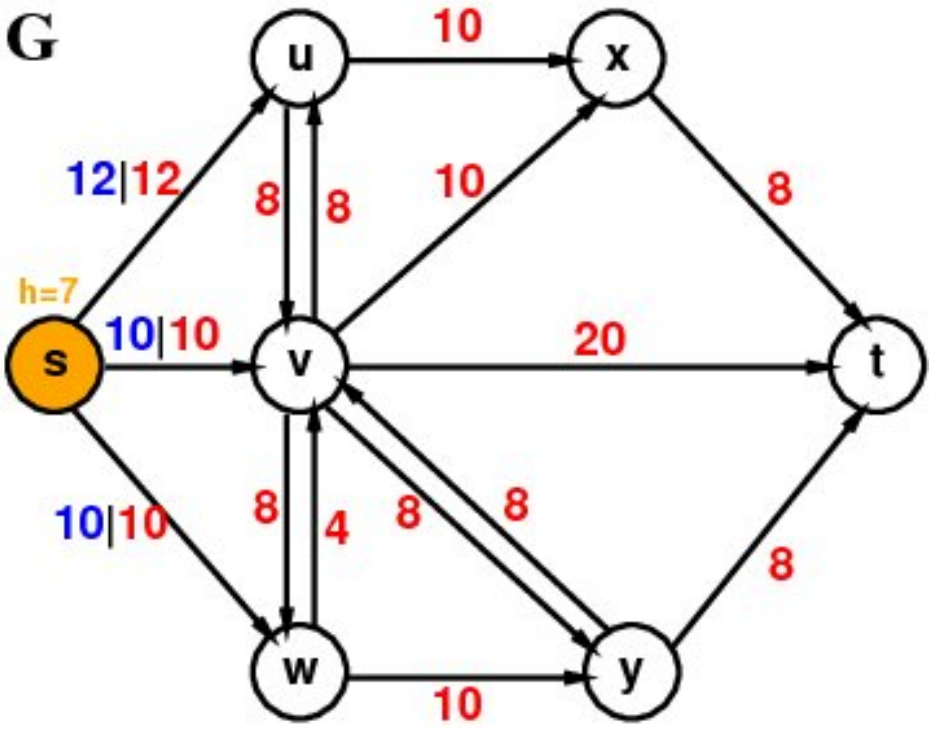
```
for each u∈V\{s} do h(u):=0; e_f(u):=0
for each (u,v)∈E do f(u,v):=0; f(v,u):=0
h(s):=|V|
for each (s,u)∈E do
    f(s,u):=c(s,u); f(u,s):=-f(s,u); e_f(u):=c(s,u)
while (there are active nodes u) do
    if (there is an admissible edge (u,v) )
        then Push(u,v)
        else Lift(u)
```

Example:



Capacities are marked in red

Example:



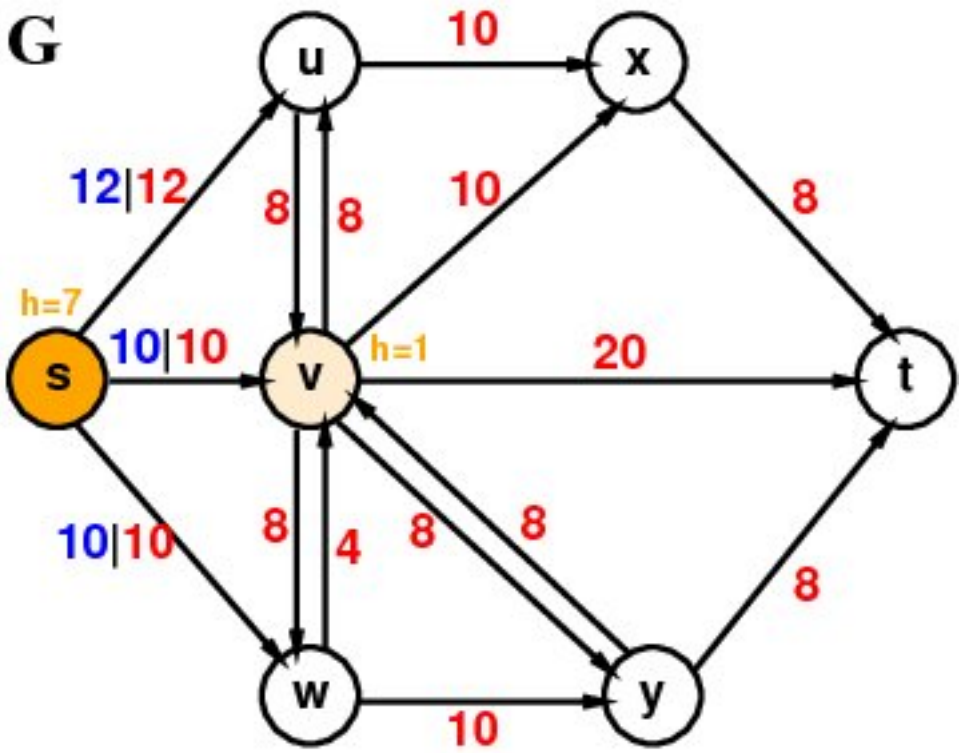
After initialization:

- 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:

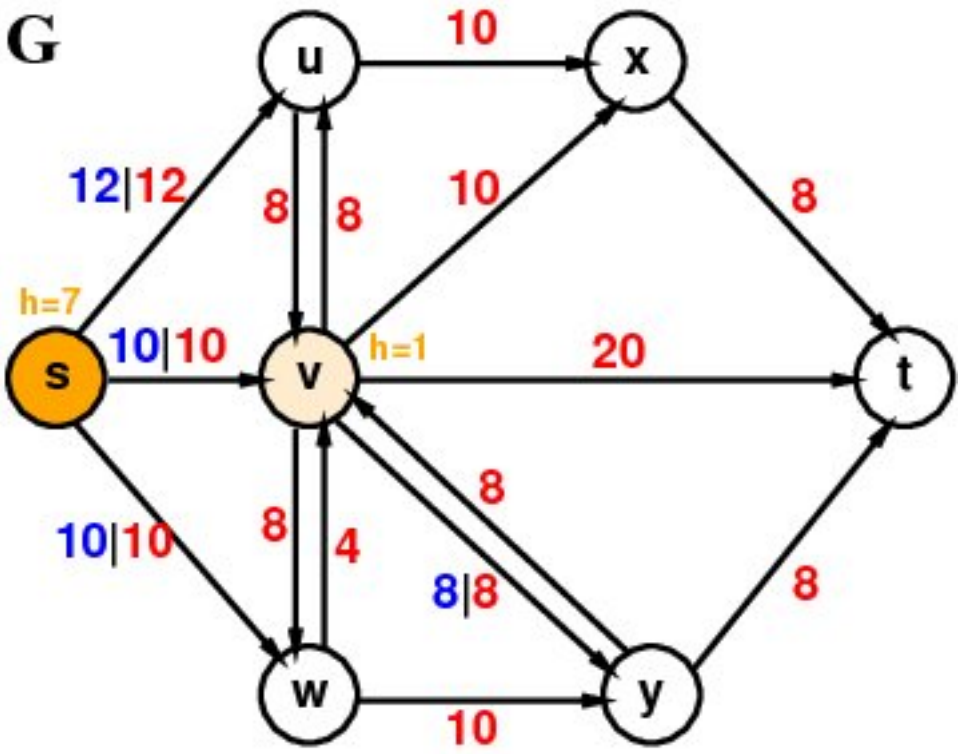


After LIFT(v):

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).

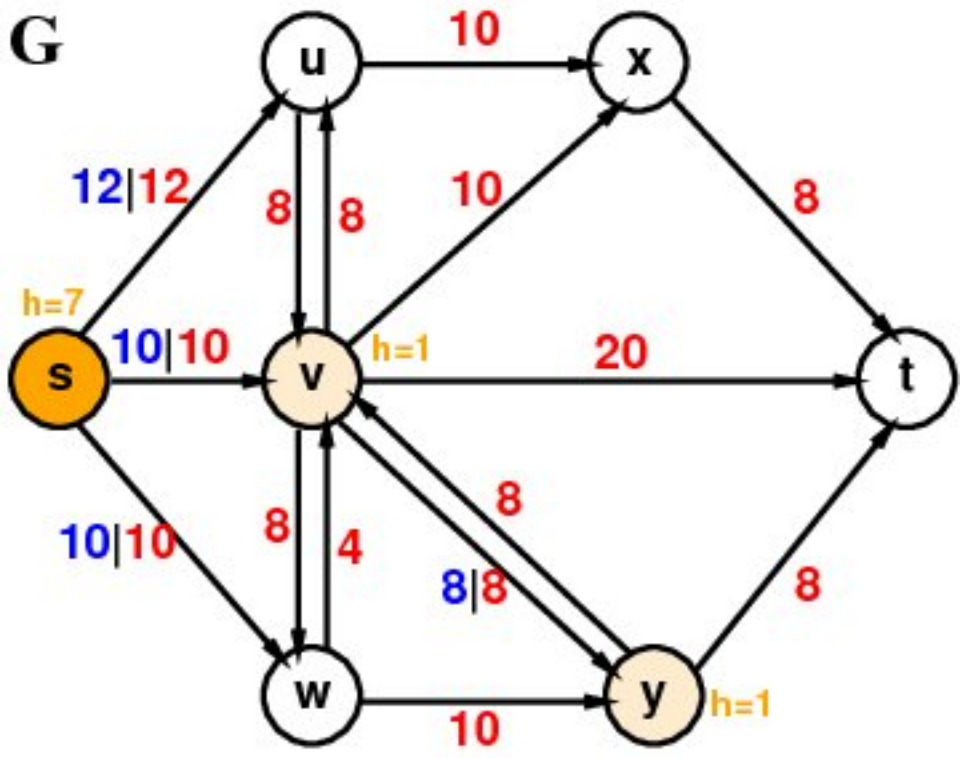
Example:



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:

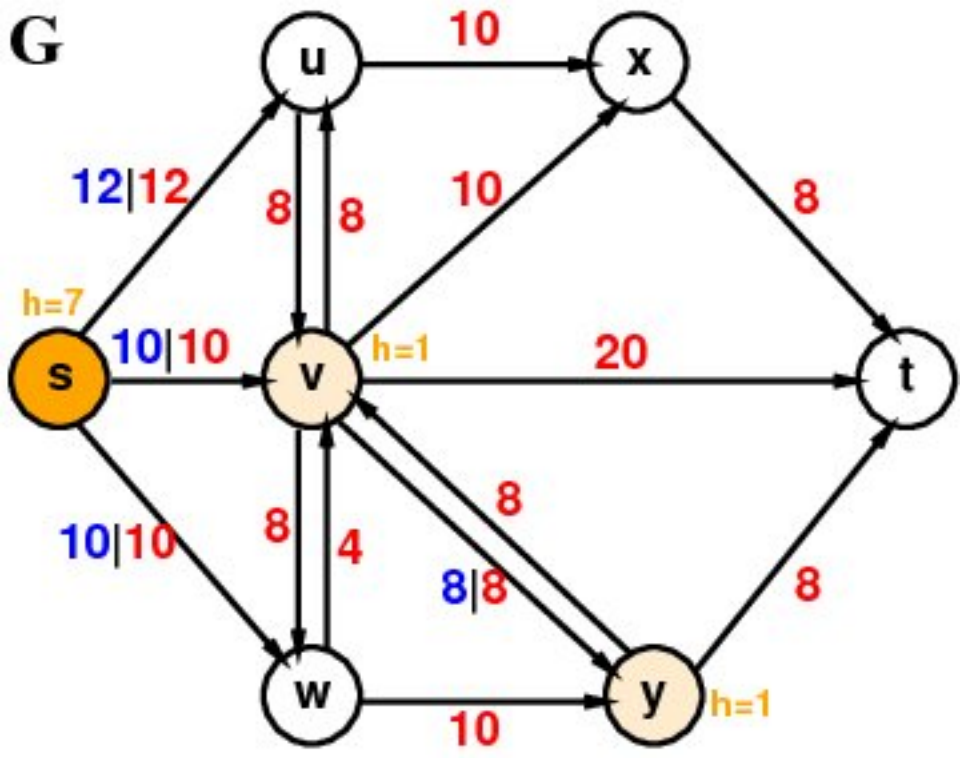


After LIFT(y):

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

Lemma 6.36: At any point in time during the execution, $e_f(s) \leq 0$ and for all nodes $v \in V \setminus \{s\}$, $e_f(v) \geq 0$.

Proof:

- We perform a complete induction over the number of executed Push and Lift operations.
- Initially, the lemma is obviously true.
- Thus we assume that it is true for some point in time.
- Then a Push operation maintains the property that $e_f(v) \geq 0$ for all $v \in V \setminus \{s\}$ due to the choice of δ.
  A Lift operation does not change any $e_f(v)$.
- Therefore, $\Sigma_{u \in V \setminus \{s\}} f(V,u) \geq 0$.
- Moreover, we know that for f it holds

$$\Sigma_{u \in V} f(V,u) = 0.$$

- Hence, $e_f(s) = f(V,s) \leq 0$.

Goldberg´s Algorithm

Lemma 6.37: Every Lift(u) call preserves the legality of the height function and increases $h(u)$ by at least 1.

Proof:

- A Lift operation is only executed for some node u if there is no admissible edge (u,v) and therefore $h(u) \leq h(v)$ for all $(u,v) \in E_f$.
- Since the new height is $h'(u)=\min\{h(v)+1 \mid (u,v) \in E_f\}$, it follows that $h'(u)>h(u)$ and $h'(u) \leq h(v)+1$ for all $(u,v) \in E_f$.

Goldberg´s Algorithm

Lemma 6.38 (Superoptimality): For every legal preflow f and every legal height function h there is no augmenting path in G_f.

Proof:

- Suppose that there is an augmenting path $(s=v_1,v_2,\ldots,v_l=t)$ in G_f.
- Since the heights of the nodes can only increase over time (see Lemma 6.37), it holds that $h(s)\geq n$, and since t can never be active, $h(t)=0$.
- Hence,
  $n \leq h(s) \leq h(v_2)+1 \leq h(v_3)+2 \leq \ldots \leq h(t)+l-1 = l-1 \leq n-1$
  since the augmenting path is simple and can therefore contain at most n nodes.
- Thus, we obtain a contradiction.

Goldberg´s Algorithm

Lemma 6.39 (Optimality): If there is no active node in G_f, then the preflow is a maximal flow.

Proof:

- If there is no node $u \in V \setminus \{s,t\}$ with $e_f(u)>0$, then it holds for all nodes $u \in V \setminus \{s,t\}$ according to Lemma 6.36 that $e_f(u)=0$. Hence, the preflow is a legal flow.
- The maximality of the flow follows from Lemma 6.38 and the Maxflow-Mincut Theorem.

It remains to show that the algorithm terminates.

Goldberg´s Algorithm

Lemma 6.40: For every active node u there is a path in G_f from u to s.

Proof:

- Let U be the set of nodes that are reachable from u in G_f.
- If $s \notin U$, then due to Lemma 6.36, all nodes in U have a non-negative excess flow.
- The flow into U must be 0 because if there was an edge $(v,w) \in E$ with $v \notin U$ and $w \in U$ and $f(v,w)>0$, then $c_f(w,v) \geq f(v,w) > 0$.
- Hence,

$$\begin{aligned} 0 &= \Sigma_{v \in V \setminus U} f(v,U) \geq \Sigma_{v \in V \setminus U} f(v,U) - \Sigma_{v \in U} f(v,V \setminus U) \\ &= \Sigma_{v \in V \setminus U} f(v,U) - \Sigma_{v \in U} f(v,V \setminus U) + \Sigma_{v \in U} f(v,U) - \Sigma_{v \in U} f(v,U) \\ &= \Sigma_{v \in U} (f(V,v) - f(v,V)) \\ &= \Sigma_{v \in U \setminus \{u\}} (f(V,v) - f(v,V)) + 2e_f(u) > 0 \end{aligned}$$

- Thus, we arrive at a contradiction, and therefore $s \in U$.

Goldberg´s Algorithm

Lemma 6.41: For every active node $v \in V$, $h(v) \leq 2n-1$.

Proof:

- Since s can never be active due to Lemma 6.36 and t can never be active by definition, $h(s)=n$ and $h(t)=0$ at any time.
- Consider an arbitrary active node $u \in V \setminus \{s,t\}$.
- According to Lemma 6.40, s is reachable from u in G_f.
- Let $p=(u=v_1,v_2,\ldots,v_l=s)$ be a simple path from u to s.
- We know that $h(s)=n$. Moreover, $h(v_i) \leq h(v_{i+1})+1$ for all $i \in \{1,\ldots,l-1\}$.
- Hence, $h(u) \leq n+l \leq 2n-1$ because the path cannot contain t and therefore $l \leq n-1$.

Now we are ready to determine the runtime of the Lift and Push operations.

Goldberg´s Algorithm

Lemma 6.42: The total number of Lift operations executed by the algorithm is $O(n^2)$ and their total runtime is $O(n \cdot m)$.

Proof:

- Due to Lemmas 6.37 and 6.41 at most $2n-1$ Lift operations can be applied to any node. Hence, altogether at most $O(n^2)$ Lift operations are executed by the algorithm.
- The cost of a Lift(v) operation is equal to the outgoing degree of a node v in G_f because we have to check all nodes reachable from v.
- Thus, the total runtime of the Lift operations is at most

$$\Sigma_{v \in V} (2n-1) \cdot \deg(v) = O(n \cdot m)$$

where $\deg(v)$ denotes the (outgoing) degree of a node v.

Goldberg´s Algorithm

A Push operation is saturating if $\delta=c_f(u,v)$, and otherwise non-saturating.

Lemma 6.43: The total number of saturating Push operations is $O(n \cdot m)$.

Proof:

- After a saturating Push on (u,v), we cannot pump again flow from u to v unless v has performed a Push operation on (v,u).
- Since it must hold that $h(u)=h(v)+1$ at Push(u,v) and $h(v)=h(u)+1$ at Push(v,u) and the heights of the nodes are monotonically increasing, the height of u must have increased by at least 2 for another saturating Push along (u,v).
- Hence, there can be at most $(2n-1)/2$ saturating Push operations via (u,v), which results in a total number of at most $O(n \cdot m)$ saturating Push operations.

Goldberg´s Algorithm

Lemma 6.44: The total number of non-saturating Push operations is $O(n^2m)$.

Proof:

- We use the potential function $\Phi=\Sigma_{\text{active } v\in V} h(v)$.
- Initially, $\Phi=0$ since all heights of active nodes are equal to 0. We distinguish between three types of operations that can change Φ:
  - Non-saturating Push along edge (v,w): Then node v becomes inactive and Φ is reduced by h(v). On the other hand, w can now become active, which can increase Φ by h(w). But since $h(v)=h(w)+1$, Φ is always decreased by at least 1.
  - Saturating Push along edge (v,w): This can increase Φ by at most 2n-1 since in the worst case v remains active while w becomes active and $h(w)\leq 2n-1$.
  - Lift operation: Altogether, the Lift operations increase Φ by at most $(2n-1)n$.
- Since there are only $O(n\cdot m)$ saturating Push operations, Φ can be increased by at most $O(n^2m)$ due to these.
- Moreover, Φ can be increased by at most $O(n^2)$ due to Lift operations.
- Altogether, Φ can be increased by at most $O(n^2m)$. Since $\Phi\geq 0$ at any time, the number of non-saturating Push operations is at most $O(n^2m)$.

Goldberg´s Algorithm

- Since a Push operation only takes constant time, the lemmas imply a total runtime of $O(n^2m)$ for Goldberg´s Algorithm.
- 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

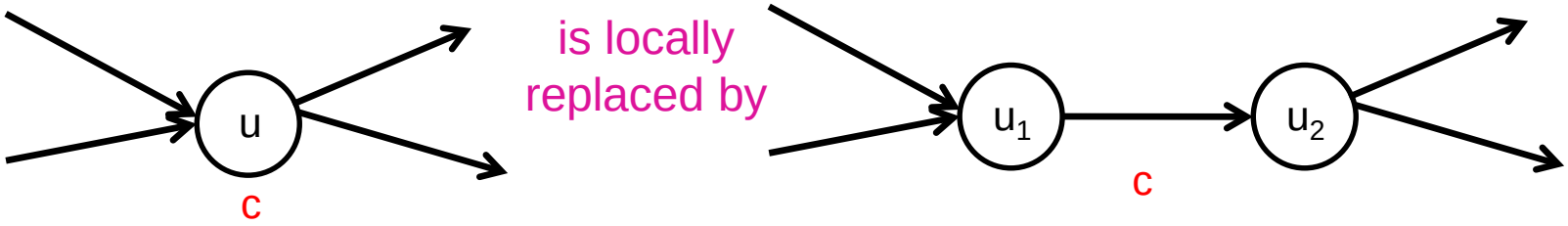
History of maximal flow algorithms:

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

	Year	Researcher	Run time
1.	1951	Dantzig	$O(n^2mU)$
2.	1955	Ford, Fulkerson	$O(nmU)$
3.	1970	Dinitz / Edmonds, Karp	$O(nm^2)$
4.	1970	Dinitz	$O(n^2m)$
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	$O(nm(\log_{m/n} n \mid \log^{2+\varepsilon} n))$
19.	1994	King et al.	$O(nm \log_{m/(n \log n)} n)$
20.	1997	Goldberg, Rao	$O(m^{3/2} \log(n^2/m) \log U)$
			$O(n^{2/3}m \log(n^2/m) \log U)$

Variants of the Maxflow Problem

… node capacities



… undirected graphs

or make use of skew symmetry to ensure flow in only one direction

Minimal Cut with minimal Number of Edges

From now on we assume the use of flows following Definition 6.5.

Problem MINCUTMINEDGES:

Input: flow network (G, s, t, c) with integer edge capacities
Output: minimal cut of (G, s, t, c) with minimal number of edges (among all minimal cuts)

Transform (G, s, t, c) into a flow network (G, s, t, c') with $c'(u, v) = M \cdot c(u, v) + 1$, where $M \geq |E| + 1$ is a sufficiently large constant.

A solution of the MAXFLOW problem in (G, s, t, c') yields a minimal cut (S, T) with

$$c'(S, T) = M \cdot \underbrace{c(S, T)}_{\text{minimal cut in } G} + \underbrace{|\{e \in E \mid e \in S \times T\}|}_{\text{number of edges crossing the cut}}$$

Circulation

Definition 6.45:

a) A circulation network is a tripel (G, b, c), where $G = (V, E)$ is a directed graph and $b, c : E \to \mathbb{R}$ are capacity functions with $b(e) \leq c(e)$ for all $e \in E$.

b) A circulation f is a flow without a source or sink. Formally, $f : E \to \mathbb{R}$ with
   1. $b(e) \leq f(e) \leq c(e)$ for all $e \in E$ (capacity constraints)
   2. $\Sigma_{(u, v) \in E} f(u, v) = \Sigma_{(v, w) \in E} f(v, w)$ for all $v \in V$ (flow conservation)

Problem CIRCULATE:

Input: circulation network (G, b, c)
Output: circulation f for (G, b, c)

Redefine the residual network $G_f = (V, E_f)$ with $E_f = \{(u, v) \in V \times V \mid c_f(u, v) > 0\}$ and

$$c_f(u,v) = \begin{cases} c(u, v) - f(u, v) & \text{if } f(u, v) < b(u, v) \\ \underbrace{c(u, v) - f(u, v)}_{\text{residual capacity as before}} + \underbrace{\max\{f(v, u) - b(v, u), 0\}}_{\text{maximal flow that can be removed in opp. direction}} & \text{if } f(u, v) \geq b(u, v) \end{cases}$$

$\longleftarrow$ increases net flow from u to v!

Circulation

```
Algorithm CIRCULATE (circulation network (G, b, c)) {
    f(e) = 0 for all e ∈ E;                          /* initialize to trivial non-feasible flow */
    while (∃e ∈ E : f(e) < b(e)) {
        choose (u, v) ∈ E with f(u, v) < b(u, v);    /* flow on edge (u, v) is too small*/
        if (∃ path P from v to u in G_f) {           /* notice the new definition of G_f ! */
            C = (P, (u, v)) is a cycle with (u, v) ∈ C;
            send δ = min_{e∈C} c_f (e) flow along cycle C;
        } else return(NULL);                         /* there is no circulation */
    } /* while */
    return(f);
} /* CIRCULATE */
```

Algorithm CIRCULATE terminates. Whenever it outputs a function f, then f is a feasible circulation for (G, b, c). Whenever the algorithm outputs *NULL*, there is no feasible circulation for (G, b, c).

Lemma 6.46: (G, b, c) with G = (V, E) has a feasible circulation $\Leftrightarrow$ for every subset $U \subseteq V$

$$\sum_{(u, v) \in (U, \bar{U})} b(u, v) \leq \sum_{(v, u) \in (\bar{U}, U)} c(v, u)$$

Circulation

Lemma 6.46: (G, b, c) with $G = (V, E)$ has a feasible circulation $\Leftrightarrow$ for every subset $U \subseteq V$

$$\sum_{(u, v) \in (U,\bar{U})} b(u, v) \leq \sum_{(u, v) \in (\bar{U},U)} c(v, u)$$

Proof:

$\Rightarrow$: Let f be a circulation for (G,b,c).

- Then it holds for all $v \in V$

$$\Sigma_{(u,v)\in E} f(u,v) = \Sigma_{(v,w)\in E} f(v,w)$$

- Therefore, it holds for all $U \subseteq V$:

$$\begin{aligned}\Sigma_{(u,v)\in(U,\bar{U})} b(u,v) &\leq \Sigma_{(u,v)\in(U,\bar{U})} f(u,v) \\ &= \Sigma_{(v,w)\in(\bar{U},U)} f(v,w) \\ &\leq \Sigma_{(v,w)\in(\bar{U},U)} c(v,w)\end{aligned}$$

Circulation

Proof (continued):

$\Leftarrow$: suppose there is no circulation f for (G,b,c).

- Then algorithm CIRCULATE outputs NULL.
- That is, there is an edge $e=(u,v)\in E$ with $f(e)<b(e)$ so that there is no path in G_f from v to u.
- Define $U=\{w\in V \mid v \rightsquigarrow w \text{ in } G_f\}$.
- Then $v\in U$ and $u\in \bar{U}$ and for all $(x,y)\in(U,\bar{U})$: $c_f(x,y)=0$.
- Hence, $f(e)=c(e)$ for all $e\in(U,\bar{U})$ and $f(e)\leq b(e)$ for all $e\in(\bar{U},U)$.
- Therefore,

$$\begin{aligned}\Sigma_{(u,v)\in(\bar{U},U)}\, b(u,v) &> \Sigma_{(u,v)\in(\bar{U},U)}\, f(u,v)\\ &= \Sigma_{(v,w)\in(U,\bar{U})}\, f(v,w)\\ &= \Sigma_{(v,w)\in(U,\bar{U})}\, c(v,w)\end{aligned}$$

Lower Bounds for Edge Flows

Definition 6.47:

a) A constrained flow network is a tuple (G, s, t, b, c), where (G, s, t, c) is a flow network and $b : E \to \mathbb{R}$ is a function with $b(e) \leq c(e)$ for all $e \in E$.

b) A constrained flow f for (G, s, t, b, c) is a feasible flow for (G, s, t, c) with $b(e) \leq f(e)$ for all $e \in E$.

Problem LOWERBOUNDEDFLOW:

Input: constrained flow network (G, s, t, b, c)
Output: constrained flow f for (G, s, t, b, c) of maximal value $|f|$

Lower Bounds for Edge Flows

Algorithm BOUNDEDFLOW (constrained flow network (G, s, t, b, c)){
  construct circulation network (G', b', c') with
    $G' = G + (t, s)$;
    $b'(e) = b(e)$; $c'(e) = c(e)$ for all $e \in E$;
    $b'(t, s) = 0$; $c'(t, s) = \infty$;
  f' = CIRCULATE(G', b', c');
  if (f' = *NULL*) **return** (*NULL*);
  f = feasible flow for (G, s, t, b, c) resulting from f' by deleting edge (t, s) and its flow;
  solve the MAXFLOW problem for (G, s, t, c) with
    f as initial flow and
    residual capacities $c'_f(u, v) = \underbrace{c(u, v) - f(u, v)}_{\text{as before}} + \underbrace{\max\{f(v, u) - b(v, u), 0\}}_{\text{max. flow to be deleted}}$ for all $(u, v) \in E$
  let g be the maximal flow obtained for (G, s, t, c);
  return(g);
} /* BOUNDEDFLOW */

If Algorithm BOUNDEDFLOW outputs *NULL*, then there is no constrained flow for (G, s, t, b, c). Otherwise, Algorithm BOUNDEDFLOW outputs a solution f for the LOWERBOUNDEDFLOW problem.

Minimal Flow Problem

Problem MINFLOW:

Input: constrained flow network (G, s, t, b, c)
Output: constrained flow f for (G, s, t, b, c) with minimal value $|f|$

Algorithm MINIMUMFLOW (constrained flow network (G, s, t, b, c)) {
  h = BOUNDEDFLOW(G, s, t, b, c);
  if (h = *NULL*) **return**(*NULL*);
  construct flow network (G^R, s^R, t^R, c^R) with
    $G^R = (V, E^R)$, $E^R = \{(v, u) | (u, v) \in E\}$;
    $s^R = t$; $t^R = s$;
    $c^R(u, v) = c_f(u, v)$ for all $(u, v) \in E^R$,
    where $c_f()$ represents the redefined residual capacitites;
  solve the MAXFLOW problem for (G^R, s^R, t^R, c^R) and let
  g be the maximal flow for (G^R, s^R, t^R, c^R);
  $f = h - g$;
  return(f);
} /* MINIMUMFLOW */

If Algorithm MINIMUMFLOW outputs *NULL,* then there is no feasible constrained flow for (G, s, t, b, c). Otherwise, Algorithm MINIMUMFLOW outputs a solution f for the MINFLOW problem.

Flows of minimal Cost

a) A flow network with edge costs is tuple (G, s, t, c, γ) where $G = (V, E)$ is a directed graph, $c : E \to \mathbb{N}_0$ defines edge capacities, and $\gamma : E \to \mathbb{N}_0$ defines edge costs, i.e., $\gamma(e)$ is the cost of sending one unit of flow across e.

b) The cost of a flow f from s to t in G is defined as

$$\gamma(f) := \Sigma_{e \in E} \gamma(e) f(e).$$

Problem MINCOSTMAXFLOW:

Input: flow network (G, s, t, c, γ)

Output: maximal flow f with minimal cost $\gamma(f)$.

W.lo.g. we assume that $G=(V,E)$ is a directed graph with at most one edge for each pair of nodes, i.e., there is no pair of nodes $u,v \in V$ with $(u,v),(v,u) \in E$.

Flows of minimal Cost

Let f be a flow.

- cost of an augmenting path p for f:

  $\gamma(p) = \Sigma_{(u,v)\in p:(u,v)\in E} c_f(p)\cdot\gamma(u,v) - \Sigma_{(u,v)\in p:(v,u)\in E} c_f(p)\cdot\gamma(v,u)$
- An augmenting cycle w.r.t. f is an augmenting path whose startpoint and endpoint are identical.

Lemma 6.49: For every augmenting cycle p for a flow f there is a flow f' with $|f'|=|f|$ and

$$\gamma(f')=\gamma(f) + \gamma(p) .$$

Proof:

- Let $f'=f+p$. Then f' is still a feasible flow.
- Furthermore, the cost of f' satisfies:

$\gamma(f') = \gamma(f) + \Sigma_{(u,v)\in p:(u,v)\in E} c_f(p)\cdot\gamma(u,v) - \Sigma_{(u,v)\in p:(v,u)\in E} c_f(p)\cdot\gamma(v,u)$
$= \gamma(f) + \gamma(p)$

Flows of minimal Cost

Theorem 6.50: A flow f has minimal cost among all flows of value $|f|$ if and only if there is no augmenting cycle of negative cost for f.

Proof:

$\Rightarrow$: Follows from Lemma 6.49.

$\Leftarrow$: Let f be a flow that does not have a minimal cost. Let g be a flow of minimal cost and $|g|=|f|$.

- Consider the residual network $G_{g-f} = (V,E_{g-f})$ with $E_{g-f} = E^{+}_{g-f} \cup E^{-}_{g-f}$, where
  $E^{+}_{g-f} = \{(u,v) \mid g(u,v) > f(u,v) \}$ and
  $E^{-}_{g-f} = \{(v,u) \mid g(u,v) < f(u,v) \}$
- Let the edge capacities be defined as $c_{g-f}=|g(e)-f(e)|$.
- Then it holds for all nodes $v \in V$ that

  $\Sigma_{u \in V} c_{g-f}(u,v) - \Sigma_{w \in V} c_{g-f}(v,w) = 0$ (sketch on blackboard)

Flows of minimal Cost

Proof (continued):

- Now, let $c_{min} = \min\{c_{g-f}(u,v) \mid (u,v) \in E_{g-f}\}$.
- If we start at an arbitrary node $v \in V$ with edges in G_{g-f}, then there is always an edge (v,w) with $c_{g-f}(v,w) \geq c_{min}$, and every node not visited so far can be left due to the flow conservation principle.
- Since the number of nodes is limited by $|V|$, there must be an augmenting cycle in G_{g-f}.
- When removing this cycle, the flow conservation and capacity constraints are still satisfied.
- Thus, G_{g-f} can be decomposed into a set of augmenting cycles. When applying these cycles to f, we obtain g.
- Since $\gamma(g) < \gamma(f)$, at least one of these cycles must have negative cost.

Flows of minimal Cost

If we have integer edge capacities and edge costs that are bounded by a constant C, we can use the following polynomial time algorithm to compute a maximal flow with minimal cost:

- Use Ford-Fulkerson to compute a maximal flow. Runtime: $O(C \cdot n \cdot m)$.
- Search for augmenting cycles of negative cost in G_f until no such cycles can be found. A negative augmenting cycle can be found via Bellman-Ford in $O(n \cdot m)$ time (why?). Every such cycle has an integer capacity of >0 and integer cost <0. I.e., the cost of the maximal flow reduces by at least 1 for each such augmenting cycle. The total runtime for this part is therefore $O((C^2 \cdot m) \cdot (n \cdot m)) = O(C^2 \cdot n \cdot m^2)$.

- If in each iteration the most negative (non-simple) cycle in G_{g-f} is chosen, the number of iterations is $O(n \ln(C^2 \cdot m))$ because the improvement at each iteration is at least $(\gamma(f)-\gamma(g))/n$ (why?). However, computing such a cycle is NP-hard.
- More advanced techniques can also compute a maximal flow of minimal cost for arbitrary capacities and cost values (see, e.g., the book by Ahuja, Magnanti und Orlin: Network Flows).

Flows of minimal Cost

Definition 6.48:

a) A flow network with edge costs is a tuple (G, c, γ, δ) where $G = (V, E)$ is a directed graph, $c : E \to \mathbb{N}_0$ defines the edge capacities, $\delta : V \to \mathbb{Z}$ defines the node demands, $\Sigma_{v \in V} \delta(v) = 0$, and $\gamma : E \to \mathbb{N}_0$ defines the edge costs, i.e., $\gamma(e)$ is the cost to send one unit of flow across e.
Note that $\delta(v) > 0$ means a node that consumes flow while $\delta(v) < 0$ means a node that produces flow.

b) A flow f of minimal cost for (G, c, γ, δ) is a function $f : E \to \mathbb{N}_0$ with
1. $f(e) \leq c(e)$ for all $e \in E$ (capacity constraints)
2. $\Sigma_{(u,v) \in E} f(u, v) - \Sigma_{(v,w) \in E} f(v, w) = \delta(v)$ for all $v \in V$, (flow conservation)
which minimizes the cost $\gamma(f) := \Sigma_{e \in E} \gamma(e) f(e)$.

Problem MINCOSTFLOW:
Input: flow network with edge costs (G, c, γ, δ)
Output: flow f of minimal cost for (G, c, γ, δ)

Solution possible by combining LOWERBOUNDEDFLOW problems with the MINCOSTMAXFLOW problem.

Next Chapter

String Matching Algorithms...