Maximum Flow ***Revised 09/09/2004***

Main Reference: Sections 26.1-26.3 in CLRS.

- Introduction
  - Definitions
  - Multi-Source Multi-Sink
- The Ford-Fulkerson Method
  - Residual Networks
  - Augmenting Paths
  - The Max-Flow Min-Cut Theorem
  - The Edmonds-Karp algorithm
- Max Bipartite Matching
- Odds and Ends

Maximum Flow

Main Reference: Sections 26.1-26.3 in CLRS.

- Given a directed graph,
  $G = (V, E)$: flow network

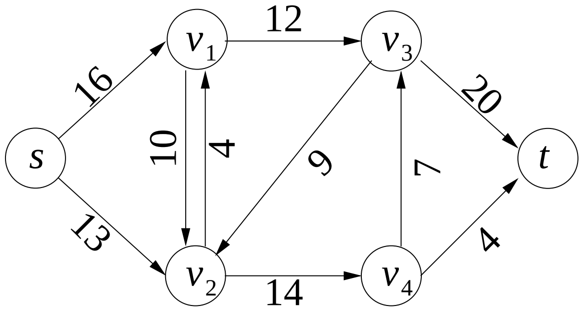
- Source (producer) s and destination t.

- Internal Nodes are *warehouses*

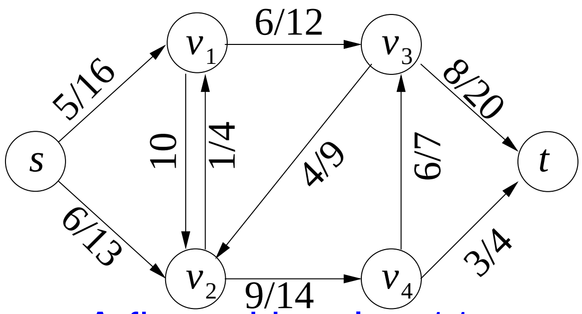
- Edge costs are *capacities*
  Maximum amount that can be shipped over edge

- All goods shipped into a warehouse must leave the warehouse

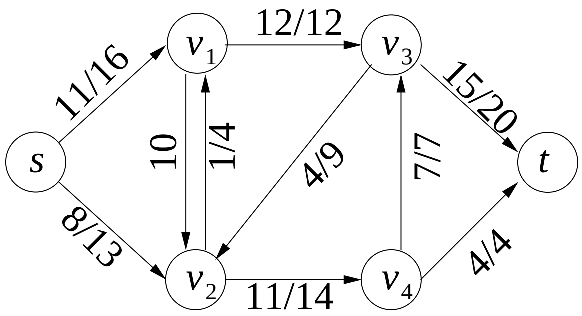
- **Goal:**
  Ship Maximum amount (flow) from s to t.



A Flow Network with capacities



A flow with value 11



A max-flow: value is 19

A flow network is a graph $G = (V, E)$.
Source $s \in V,$, sink $t \in V,$.

Every edge $(u, v) \in E$ has capacity , $c(u, v) \geq 0$.
Assume that for every $v \in V$
there is a path from s to v and from v to t.

A **FLOW** is a function $f : V \times V \rightarrow R$ satisfying:

- Capacity Constraint:
  $\forall u, v, \in V, \quad f(u, v) \leq c(u, v).$

- Skew Symmetry:
  $\forall u, v, \in V, \quad f(u, v) = -f(v, u).$

- Flow Conservation:
  $\forall u \in V - \{s, t\}, \quad \sum_{v \in V} f(u, v) = 0.$

The **VALUE** of flow v is $|f| = \sum_{v \in V} f(s, v).$.

MAXIMUM-FLOW PROBLEM:
Given $G, c, s, t,$ find f that maximizes $|f|$.

Multi-Source Multi-Sink Problem

The basic Max-Flow problem assumes that there is only one source s, and one sink t.

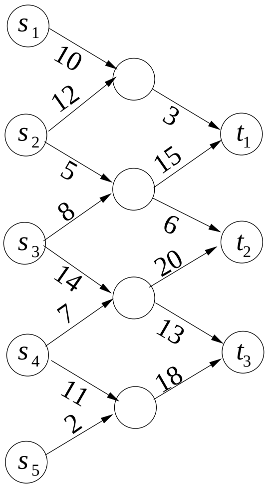
Suppose that there are multiple sources $s_1, s_2, \ldots, s_k$ and multiple sinks $t_1, t_2, \ldots, t_\ell$.

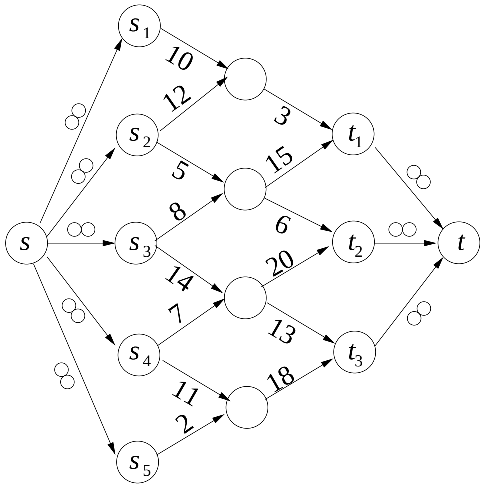
The definition of a flow remains the same except that the Flow Conservation property now becomes
$\forall u \in V - \{s_1, s_2, \ldots, s_k, t_1, t_2, \ldots, t_\ell\}, \quad \sum_{v \in V} f(u,v) = 0$
and our goal is to maximize

$$|f| = \sum_{i=1}^{k} \sum_{v \in V} f(s_i, v).$$

This problem can be reduced to the original one by introducing a *supersource* s_0, a *supersink* t_0 and edges $\cup_i (s_0, s_i)$ and $\cup_j (t_j, t_0)$, all of which have capacity ∞.

A multi-source multi-sink problem and its equivalent single-source single-sink version.





Manipulating Flows

Let $X, Y \subseteq V$. We define

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

The *flow-conservation* constraint then just says

$$\forall u \in V - \{s,t\}, \quad f(u,V) = 0.$$

Lemma: (Proof in Homework)

$\forall X \subseteq V, \quad f(X,X) = 0.$

$\forall X, Y \subseteq V, \quad f(X,Y) = -f(Y,X).$

$\forall X, Y, Z \subseteq V$ with $X \cap Y = \emptyset$

$$f(X \cup Y, Z) = f(X,Z) + f(Y,Z) \quad \text{and}$$

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

Flow f was defined as

amount that leaves source s.

We now see that this is the same as

amount that enters sink t..

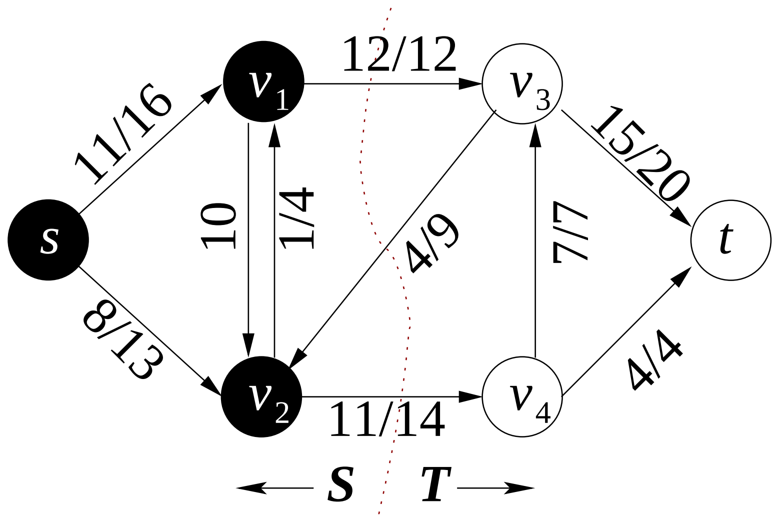
$$
\begin{array}{rcll}
|f| & = & f(s,V) & \textbf{definition} \\
& = & f(V,V) - f(V-s,V) & \textbf{previous page} \\
& = & -f(V-s,V) & \textbf{previous page} \\
& = & f(V,V-s) & \textbf{previous page} \\
& = & f(V,t) + f(V,V-s-t) & \textbf{previous page} \\
& = & f(V,t) & \textbf{flow conservation}
\end{array}
$$

In every optimization problem we have to deal with the question: **How can we prove that our solution is optimal (maximal/minimal)?**

A common technique (for max problems) is to find a good upper-bound on the cost of an optimal solution and then show that our solution satisfies that bound.

A **CUT** S, T of G is a partition of the vertices
$V = S \cup T, \quad S \cap T = \emptyset, \quad s \in S,$ **and** $t \in T.$

The flow **across** the cut is $f(S, T).$
The capacity of a cut is $C(S, T) = \sum_{x \in S, y \in T} c(x, y).$
Note that for *any* cut, $f(S, T) \leq c(S, T).$



Cut (S, T): $S = \{s, v_1, v_2\}, T = \{v_3, v_4, t\}$.

The flow value is $|f| = 19$ and $C(S, T) = 26$.

Note that $|F| \leq C(S, T)$.

Lemma: **If S, T is any cut, f any flow then** $|f| \le C(S, T).$

Proof:

$$\begin{aligned} |f| &= f(s, V) \\ &= f(s, V) + f(S - s, V) \\ &= f(S, V) \\ &= f(S, V) - f(S, S) \\ &= f(S, V - S) \\ &= f(S, T) \\ &\le c(S, T) \end{aligned}$$

We will now develop the Ford-Fulkerson method for finding max-flows. When FF terminates it provides a flow f and a cut S, T such that $|f| = C(S, T),$ so f is maximal.

The Ford-Fulkerson Method

- Is iterative.
- Starts with flow $f = 0$, $(\forall u, v,\ f(u,v) = 0)$
- At each step
  - Constructs a residual network G_f of f indicating how much capacity "remains" to be used .
  - Finds an augmenting path s-t path p in G_f along which flow can be pushed.
  - pushes f' units of flow along p.
    Creates new flow $f = f + f'$.
- Stops when there is no s-t path in current G_f.
- Let S be the set of nodes reachable from s in G_f and $T = V - S$.
  At conclusion of FF algorithm, $|f| = c(S,T)$ so f is optimal.

Residual networks

Given flow f, the residual network G_f consists of the edges along which we can (still) push more flow. The amount that can (still) be pushed across (u,v) is called the *residual capacity* $c_f(u,v)$.

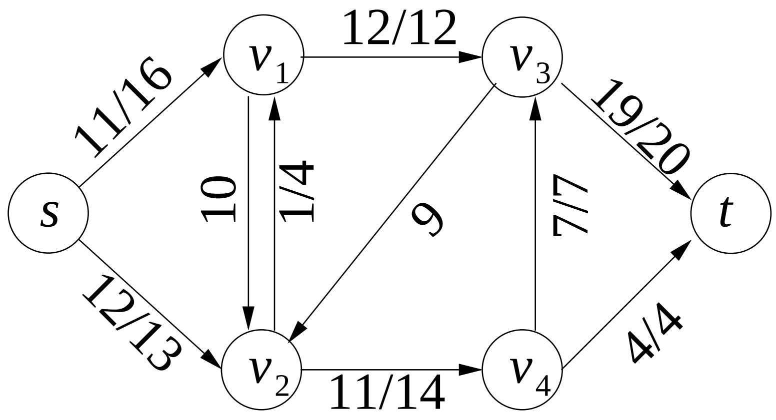
$$c_f(u,v) = c(u,v) - f(u,v).$$

If there is flow from u to v then $f(u,v) > 0$ and $c_f(u,v)$ is the remaining capacity on (u,v).

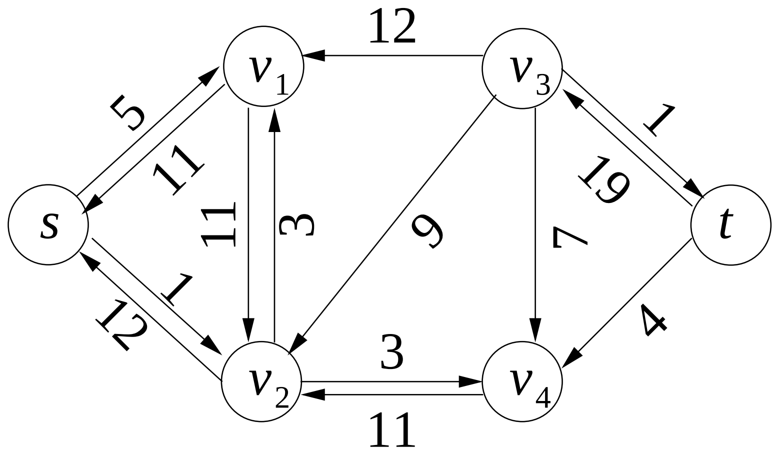
If there is flow from v to u then $f(u,v) < 0$ and $c_f(u,v) = c(u,v) + f(v,u)$ is the capacity of (u,v) plus amount of existing flow that can be pushed **backwards** from u to v.

The *Residual network* G_f is $G_f = (V, E_f)$ where

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



A flow



Its residual network

Lemma: Let f be a flow in $G = (V, E)$ and G_f its residual network. Let f' be a flow in G_f.

Define $f + f'$ as $(f + f')(u, v) = f(u, v) + f'(u, v)$.

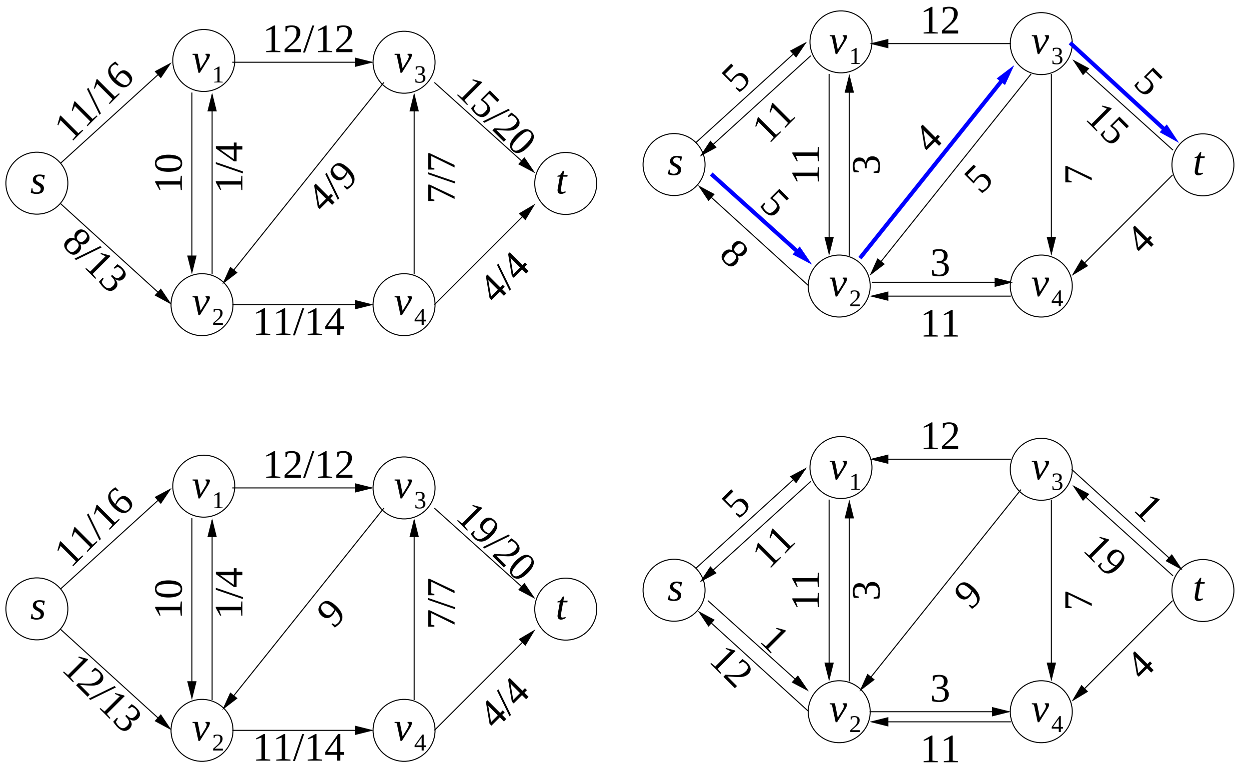
Then $f + f'$ is a flow in G with value $|f + f'| = |f| + |f'|$.

Augmenting path p is a simple s-t path in G_f.
The residual capacity of a.p. p is
$c_f(p) = \min\{c_f(u, v) \,:\, (u, v) \text{ on } p\}$.

Let p be an augmenting path in G_f and define

$$f_p(u, v) = \begin{cases} c_f(p) & \text{if } (u, v) \text{ is on } p \\ -c_f(p) & \text{if } (v, u) \text{ is on } p \\ 0 & \text{otherwise} \end{cases}$$

Lemma: If f is a flow and p an a.p.in G_f then:
f_p is a flow in G_g with $|f_p| = c_f(p) > 0$.
$f' = f + f_p$ is a flow in G with $|f'| = |f| + |f_p| > |f|$.



An initial flow f.

Its residual network G_f

and an augmenting path f' in G_f.

The flow $f + f'$ and its residual network.

Optimality

Theorem: (Max-Flow Min-Cut Theorem)
Let f be a flow. Then the following three conditions are equivalent:

1. f is a maximum flow in G.
2. G_f contains no augmenting paths
3. $|f| = C(S,T)$ for some (S,T) cut.

Proof:

- (1) $\Rightarrow$ (2): If G_f contained an augmenting path p then $|f + f_p| > |f|$ so f could not be maximal.

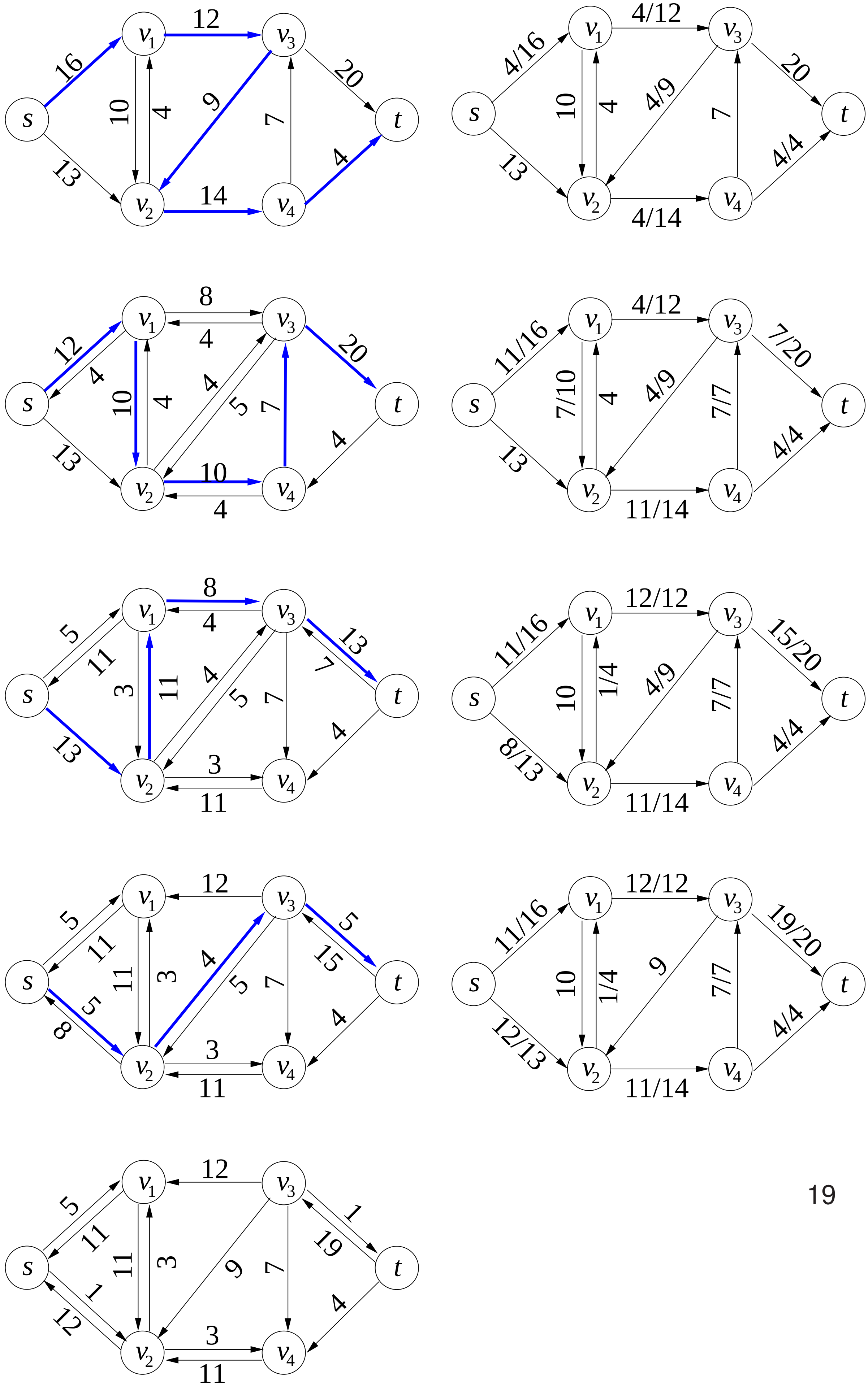
- (2) $\Rightarrow$ (3): Let $S = \{u \in V \ : \ \exists$ path from s to v in $G_f\}$. $T = V - S$. Then
$f(S,T) = f(S,V) - f(S,S) = f(S,V) = f(s,V) + f(S-s,V) = |f|$.
Now note that $\forall u \in S, v \in T$, $f(u,v) = c(u,v)$ since otherwise $c_f(u,v) > 0$ and $v \in S$.
Thus $C(S,T) = f(S,T) = |f|$.

- (3) $\Rightarrow$ (1): We previously saw that every flow f' must satisfy $|f'| \leq C(S,T)$ so if $|f| = C(S,T)$, f must be optimal.

The Ford-Fulkerson Method

- Starts with flow $f \equiv 0$, $(\forall u, v,\ f(u,v) = 0)$
- Construct residual network G_f.
  If G_f contains no augmenting path, stop
  (f is optimal by MFMC theorem).
  Otherwise.

  1. Find an augmenting path ($s - t$ path) p in G_f
  2. Let f_p be the flow in G_f that pushes $c_f(p)$ units of flow along p.
  3. Let $f = f + f_p$ be new flow in G.



s v1 v2 v3 v4 t
16 12 13 10 4 9 7 20 14 4
4/16 4/12 13 10 4 4/9 7 20 4/14 4/4
12 4 8 4 13 10 4 4 5 7 20 10 4 4
11/16 4/12 13 7/10 4 4/9 7/7 7/20 11/14 4/4
5 11 8 4 13 3 11 4 5 7 13 7 3 11 4
11/16 12/12 8/13 10 1/4 4/9 7/7 15/20 11/14 4/4
5 11 12 8 5 11 3 4 5 7 5 15 3 11 4
11/16 12/12 12/13 10 1/4 9 7/7 19/20 11/14 4/4
5 11 12 12 1 11 3 9 7 1 19 3 11 4

Running Time & Finiteness

The FF method is not a completely defined algorithm since it doesn't specify how to *choose* the augmenting paths.

In fact, if the capacities are irrational, it is possible that a "bad" way of choosing the a.p. will lead to a non-terminating algorithm that will never stop (it will keep on adding cheaper and cheaper augmenting paths).

See section 6.3 of the PS book for example.

If the capacities are all integers then each c_p will be an integer ≥ 1 so the algorithm must terminate after $|f^*|$ steps, where f^* is a max-flow. Maintaining the graphs G and G_f and the flow f using adjacency lists while using DFS or BFS to find a s-t path, the algorithm can then be implemented to run in $O(|f^*||E|)$ time.

Note: This can be normalized to work if the capacities are rational.

- Starts with flow $f \equiv 0$, $O(|E|)$

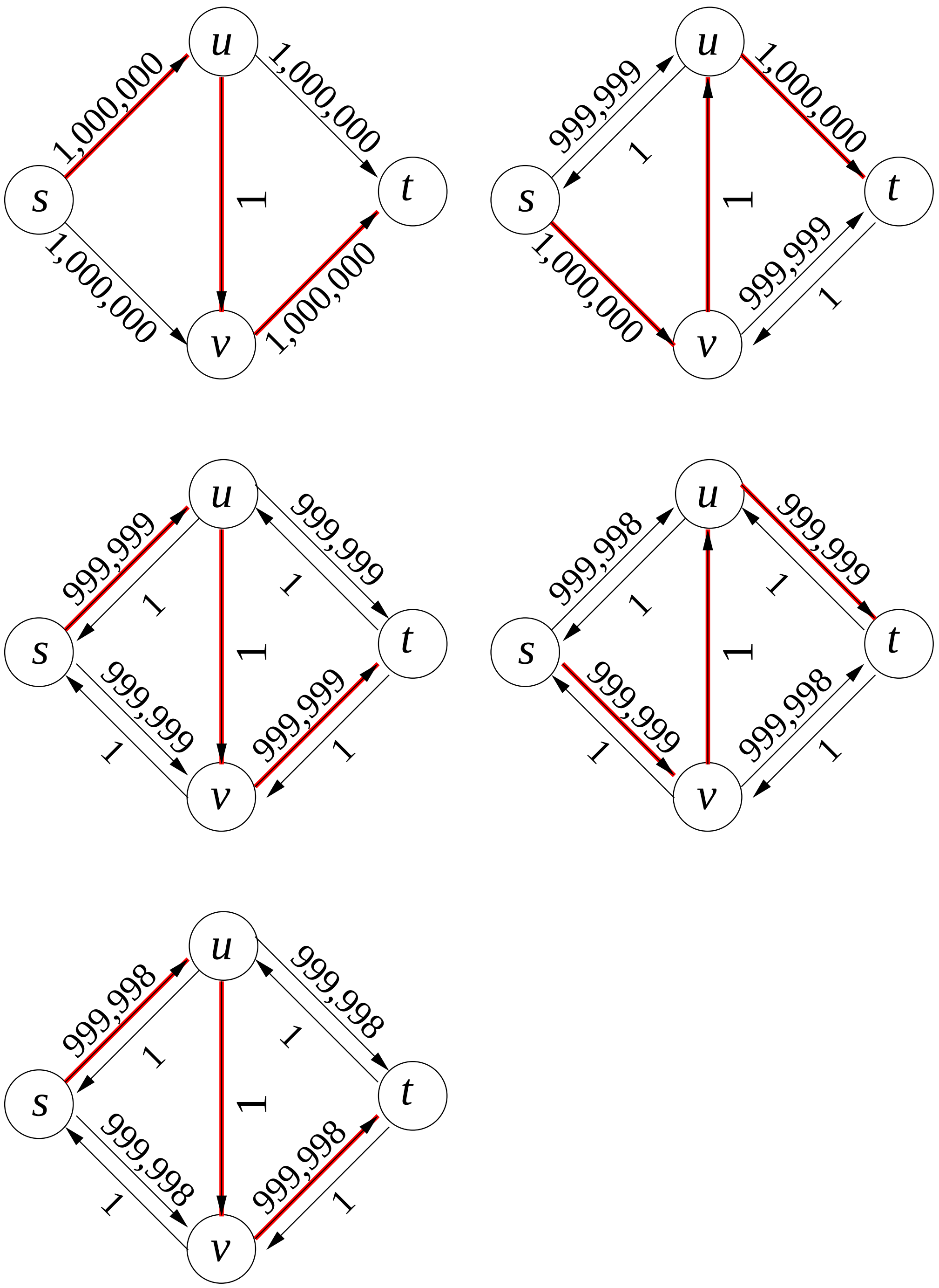
- Construct residual network G_f. $O(|E|)$
  If G_f contains no augmenting path, stop
  (f is optimal by MFMC theorem).
  Otherwise. Can be repeated $O(|f^*|)$ times.

  1. Find an augmenting $s-t$ path p in G_f
     $O(|E|)$

  2. Let f_p be the flow in G_f that pushes $c_f(p)$ units of flow along p.

  3. Let $f = f + f_p$ be new flow in G. $O(|E|)$

A pathological example in which each augmenting path only increases flow value by 1 unit.



The Edmonds-Karp Algorithm

Always choose an augmenting path of minimum-length in G_f (where each edge has unit length). This can be done in $O(E)$ time using BFS.

Theorem: The EK alg performs at most $O(VE)$ path-augmentations, so the E.K. alg runs in $O(VE^2)$ time.

Let $\delta_f(u,v)$ denote shortest-path distance from u to v in G_f.

The proof of the Theorem is a consequence of the following two lemmas:

Lemma: $\forall v \in V - \{s,t\}$, $\delta_f(s,v)$ does not decrease after a flow augmentation.

Lemma:
Edge (u,v) is *critical* on a.p. p if $c_f(u,v) = c_f(p)$. Suppose when running the E.K. algorithm that (u,v) is critical for a.p. p in G_f, and is later critical again for another a.p. p' in $G_{f'}$. Then

$$\delta_{f'}(s,u) \geq \delta_f(s,u) + 2.$$

Application: Max Bipartite Matching

A graph $G = (V, E)$ is *bipartite* if there exists partition $V = L \cup R$ with $L \cap R = \emptyset$ and $E \subseteq L \times R$.

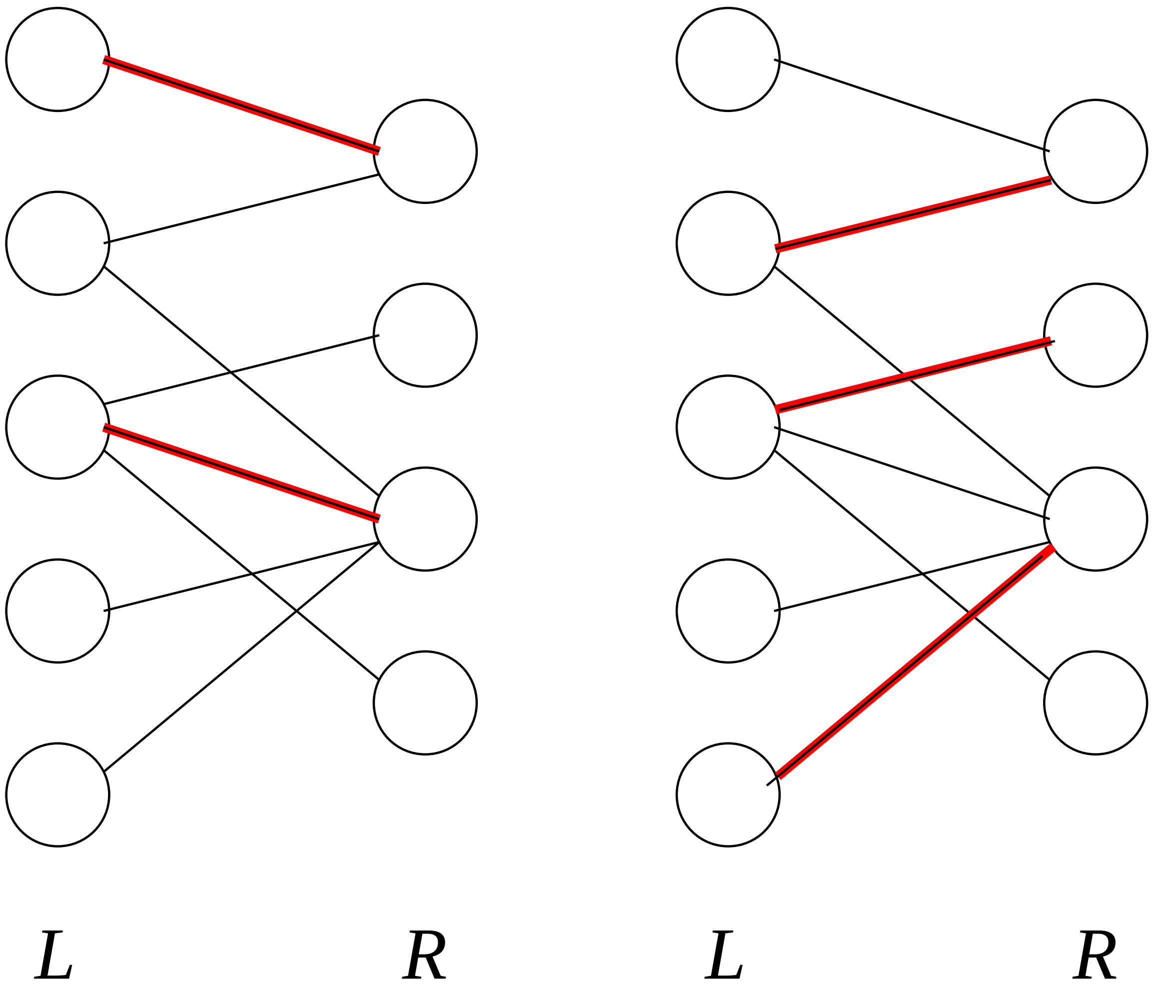
A *Matching* is a subset $M \subseteq E$ such that $\forall v \in V$ at most one edge in M is incident upon v.

The *size* of a matching is $|M|$, the number of edges in M.

A *Maximum Matching* is matching M such that every other matching M' satisfies $|M'| \leq M$.

Problem: Given bipartite graph G, find a maximum matching.

A bipartite graph with 2 matchings



Our approach will be to write the Max Bipartite Matching problem as a Max-Flow problem.

Our *flow network* will be $G' = (V', E')$ where
$V' = V \cup \{s, t\}$ and
$E' = \{(s, u) : u \in L\} \cup \{(u, v) : u \in L, v \in R$ and $(u, v) \in E\} \cup \{(v, t) : t \in R\}$

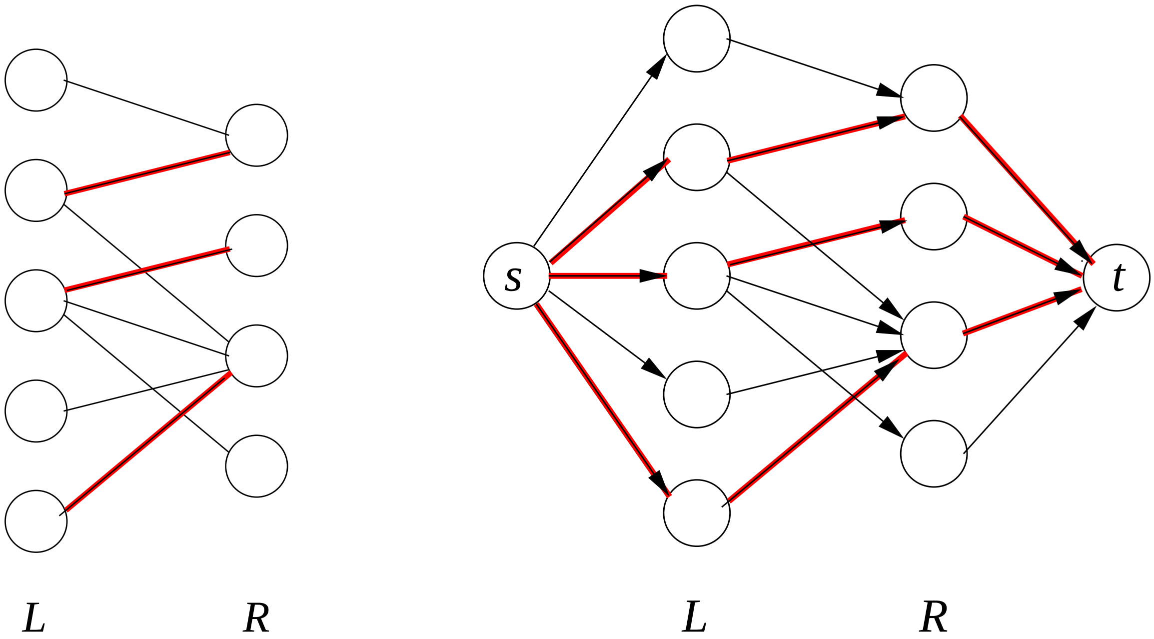
We also assign
$\forall (u, v) \in E'$, $c(u, v) = 1$.

Lemma: If f is an integer valued flow in G' then there is a matching M of G with $|f| = |M|$.
Similarly, if M is a matching of G then there is an integer valued flow f with $|f| = |M|$.

This *almost* tells us that Max-Flow solves our problem. The difficulty is that it's possible that the max-flow might not have integer value (it is possible that $|f|$ might be an integer but some $f(u, v)$ might not be integers).

A bipartite graph and its associated flow network.

A matching and associated flow are illustrated



Theorem: Let $G' = (V', E')$ be a flow network in which c is integral.
Then the max-flow f found by the F.F. method has the property that
$\forall u, v,\ f(u,v)$ is integer valued.

The proof is by induction on the steps in the FF method. At each step the current flow f is integer so the residual capacities are all integer. This implies that the a.p. found has $c_f(p)$ integral so the new flow $f + f'$ created is also integral.

The theorem guarantees that if G' is the flow network corresponding to a bipartite matching problem then max flow value $|f|$ is the value of a maximum matching.

The flow found by the FF algorithm can be modified to yield the max matching.

The FF algorithm run on this special graph will take $O(VE)$ time (why?).

Odds and Ends

- A faster implementation of the FF method uses the idea of *blocking flows* developed by Dinic. This approach finds many augmenting paths at once.

- A totally different approach to the Max-Flow algorithm is the *push-relabel* method (see CLRS for details). This can run in $O(|V|^3)$ time as comapred to the $O(|V||E|^2)$ of FF.

- We will see later that the max-flow problem can be written as a linear program. The FF method is essential a special case of the *primal-dual* algorithm for solving combinatorial LPs.