Algorithms: COMP3121/3821/9101/9801

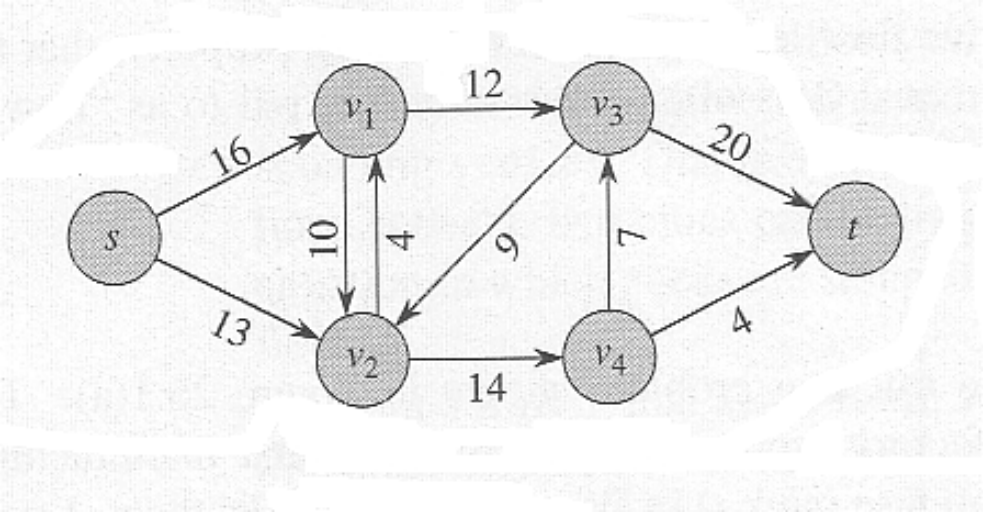
Aleks Ignjatović

School of Computer Science and Engineering
University of New South Wales

LECTURE 7: MAXIMUM FLOW

Flow Networks

- A **flow network** $G = (V, E)$ is a directed graph in which each edge $e = (u, v) \in E$ has a non-negative capacity $c(u, v) \geq 0$.
- There are two distinguished vertices: a **source** s and a **sink** t.

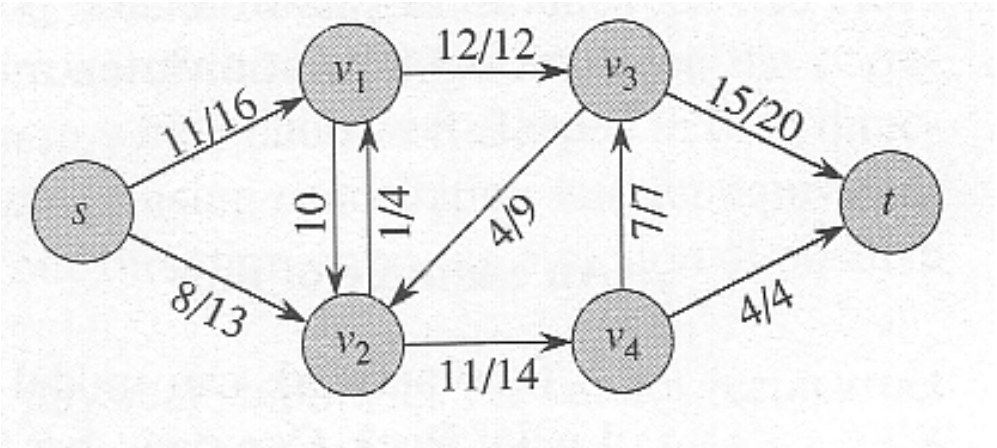


- A **flow** in G is a real valued non-negative function $f : E \to \mathbb{R}^+$, $f(u, v) \geq 0$, which satisfies
  1. **Capacity constraint:** for all edges $e(u, v) \in E$ we require $f(u, v) \leq c(u, v)$.
  2. **Flow conservation:** For all $v \in V - \{s, t\}$ we require

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

Flow Networks

- The **value of the flow** is defined as $|f| = \sum_{v \in V} f(s, v)$.
- Clearly, also $|f| = \sum_{v \in V} f(v, t)$.
- Example of a flow network with some network flow in it:

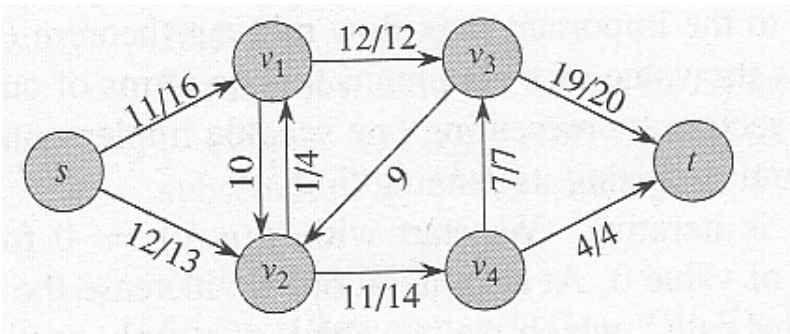


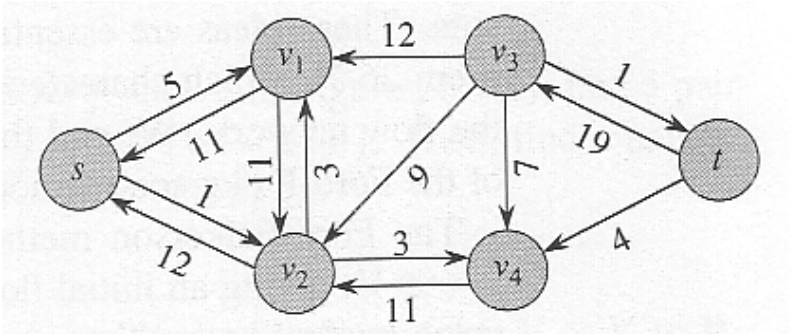
- first number on an edge: flow through that edge;
- second number: the capacity of the edge.

Flow Networks

- Example of flow networks (possibly with several sources and many sinks): transportation networks, gas pipelines, computer networks...

- The **residual flow network** for a flow network with some flow in it: the network with the leftover capacities

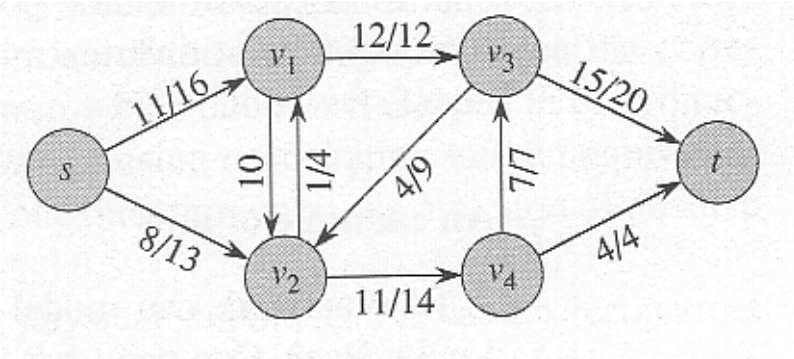


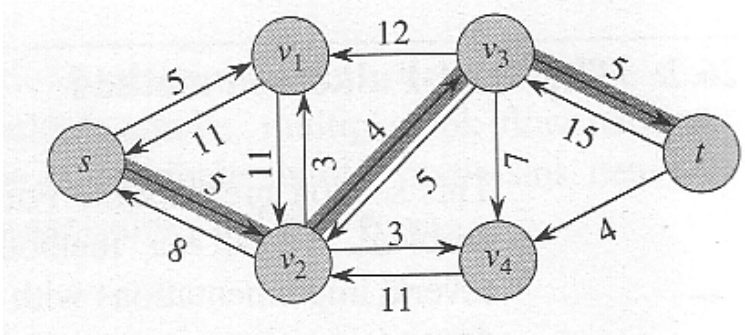


- each edge of the original network has a leftover capacity for more flow equal to the capacity of the edge minus the flow through the edge;
- if the flow through an edge is equal to the capacity of the edge, this edge disappears in the residual network;
- new "virtual" edges appear in opposite direction of an original edge with some flow in it;
- they represent the possibility to reduce the flow through the original edge; thus their capacity is equal to the flow through the original edge.

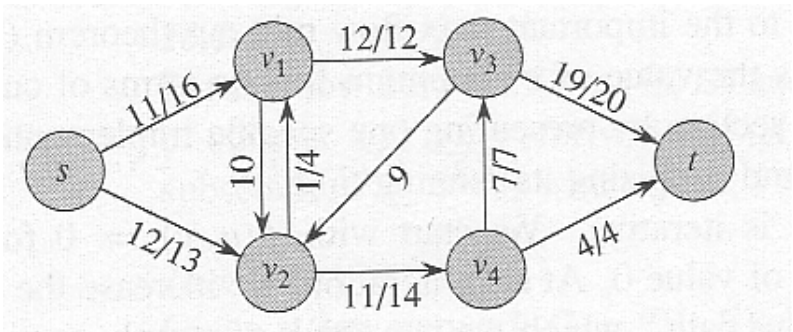
Flow Networks

- Residual flow networks can be used to increase the total flow through the network by adding an **augmenting path**





- The capacity of an augmenting path is the capacity of its "bottleneck" edge, i.e. the capacity of the smallest capacity edge on that path.
- We can now recalculate the flow through all edges along the augmenting path by adding the additional flow through the path if the flow through the augmenting path is in the same direction as the original flow, and subtracting if in opposite direction:

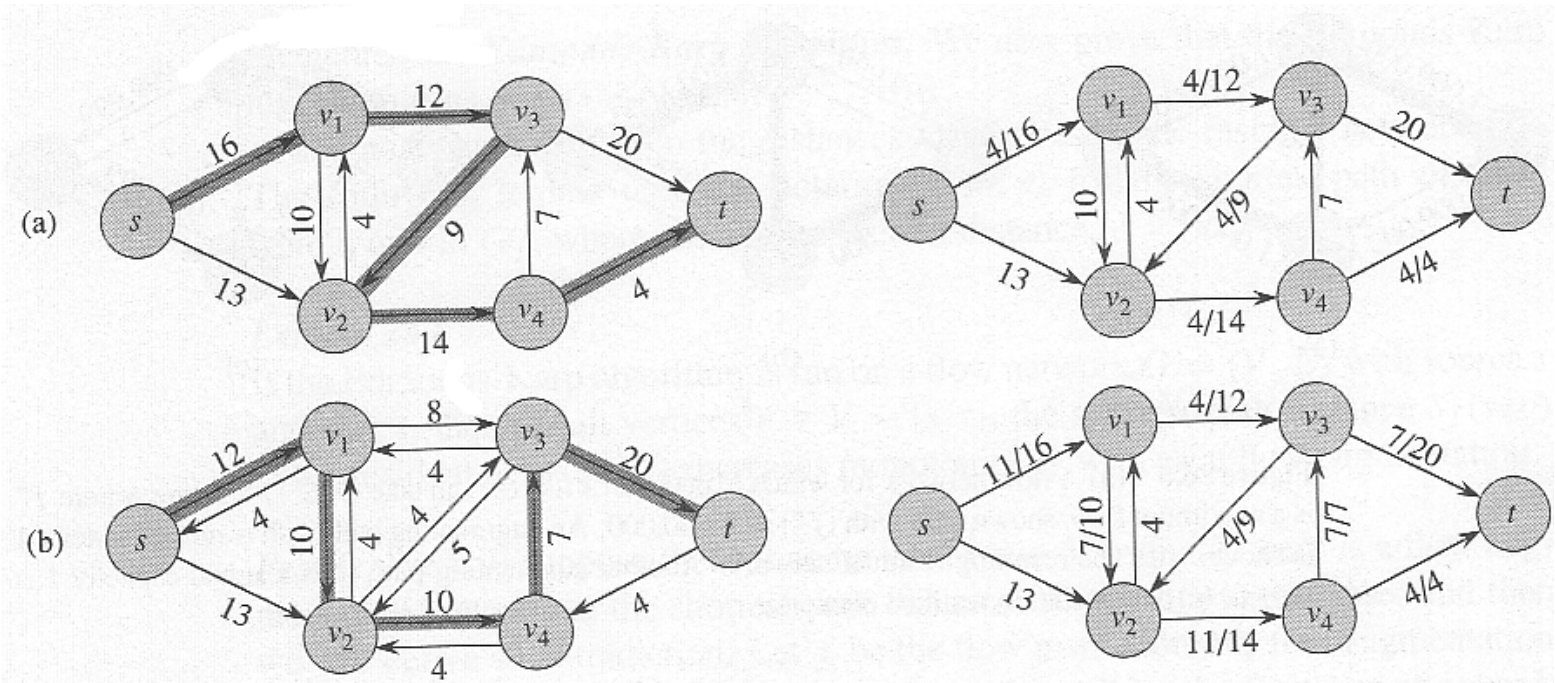


Ford Fulkerson method for finding maximal flow

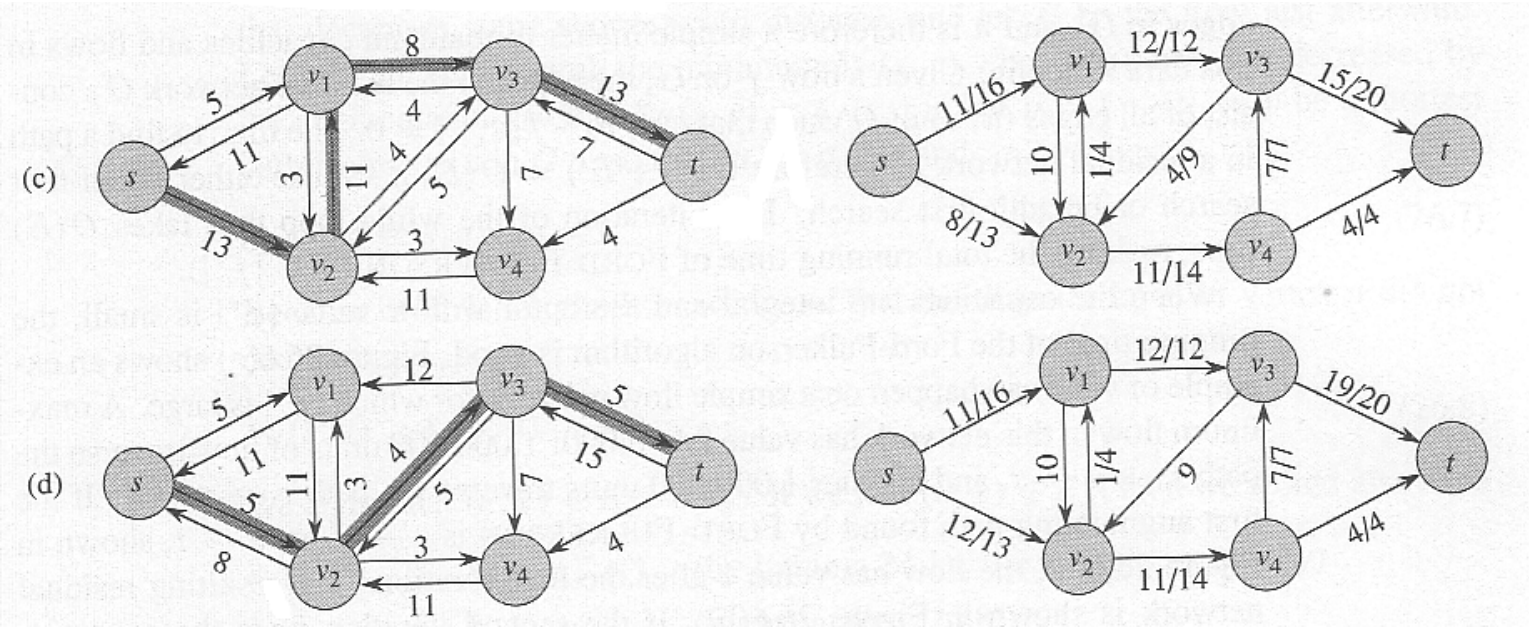
Ford - Fulkerson algorithm for finding maximal flow in a flow network:

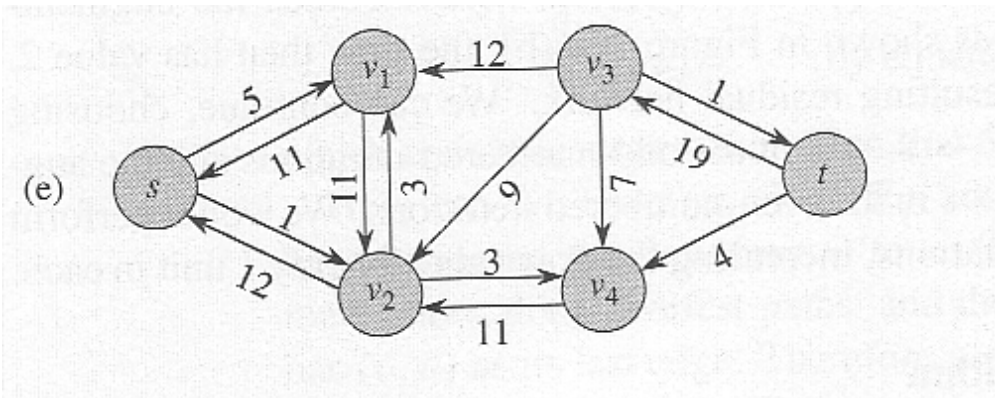
- Keep adding flow through new augmenting paths for as long as it is possible;
- When there are no more augmenting paths, you have achieved the largest possible flow in the network.

- Example:



Ford - Fulkerson method for finding maximal flow





Ford - Fulkerson method for finding maximal flow

Ford Fulkerson algorithm for finding maximal flow in a flow network:

- Keep adding flow through new augmenting paths for as long as it is possible;
- When there are no more augmenting paths, you have achieved the largest possible flow in the network.

- Why does this procedure terminate?
- Why cant we get stuck in a loop, which keeps adding augmenting paths forever?
- If all the capacities are integers, then each augmenting path increases the flow through the network for at least 1 unit;
- the total flow cannot be larger than the sum of all capacities of all edges, so eventually the process must terminate.

Ford - Fulkerson method for finding maximal flow

- Even if the procedure does terminate, why does it produce a flow of the largest possible value?
- Maybe we have created bottlenecks by choosing bad augmenting paths; maybe better choices of augmenting paths could produce a larger total flow through the network?
- This is not at all obvious, and to show that this is not the case we need a mathematical proof!
- The proof is based on the notion of a minimal cut in a flow network:

- A **cut** in a flow network is any partition of the vertices of the underlying graph into two subsets S and T such that:
  1. $S \cup T = V$
  2. $S \cap T = \emptyset$
  3. $s \in S$ and $t \in T$.

Cuts in flow networks

- The **capacity** $c(S,T)$ **of a cut** (S,T) is the sum of capacities of all edges whose left end is in S and the right end in T, i.e.

$$c(S,T) = \sum_{(u,v)\in E} \{c(u,v) : u \in S \ \& \ v \in T\}$$

- Note that the capacities of edges going in the opposite direction, i.e., from T to S do not count.

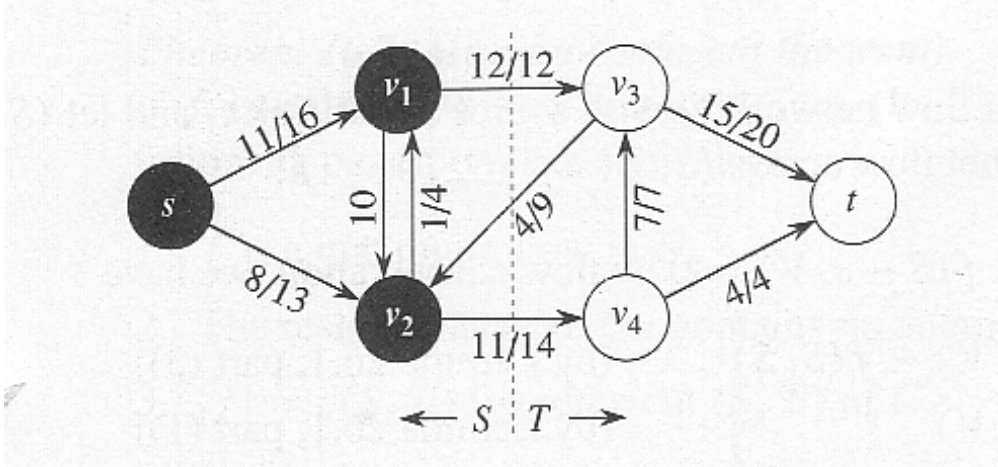
- The **flow through a cut** $f(S,T)$ is the total flow through edges from S to T minus the total flow through edges from T to S:

$$f(S,T) = \sum_{(u,v)\in E} \{f(u,v) : u \in S \ \& \ v \in T\} - \sum_{(u,v)\in E} \{f(u,v) : u \in T \ \& \ v \in S\}$$

- Clearly, $f(S,T) \leq c(S,T)$ because for every edge $(u,v) \in E$ we assumed $f(u,v) \leq c(u,v)$, and $f(u,v) \geq 0$.

Cuts in flow networks

- Example:



- In the above example the net flow across the cut is given by

$$f(S,T) = f(v_1,v_3) + f(v_2,v_4) - f(v_2,v_3) = 12 + 11 - 4 = 19$$

- Note that the flow in the opposite direction (from T to S) is subtracted.
- The capacity of the cut $c(S,T)$ is given by

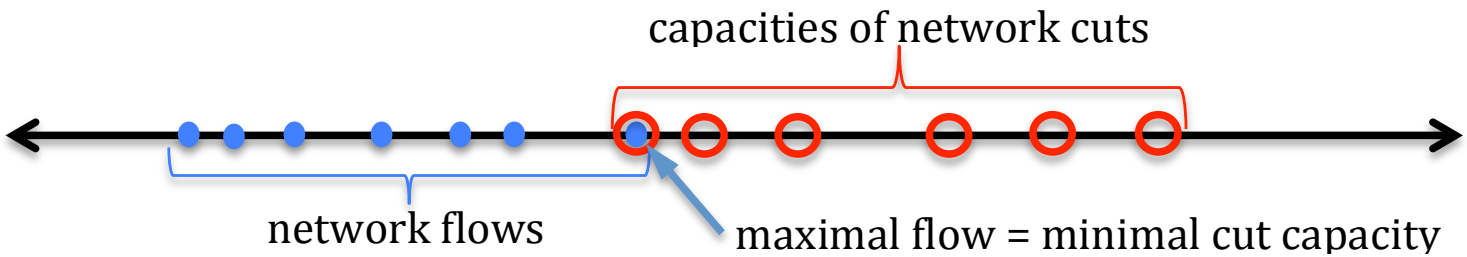
$$c(S,T) = c(v_1,v_3) + c(v_2,v_4).$$

- Note that we add only the capacities of vertices from S to T and not of vertices in the opposite direction.

Cuts in flow networks

- **Theorem:** The maximal amount of flow in a flow network is equal to the capacity of the cut of minimal capacity.

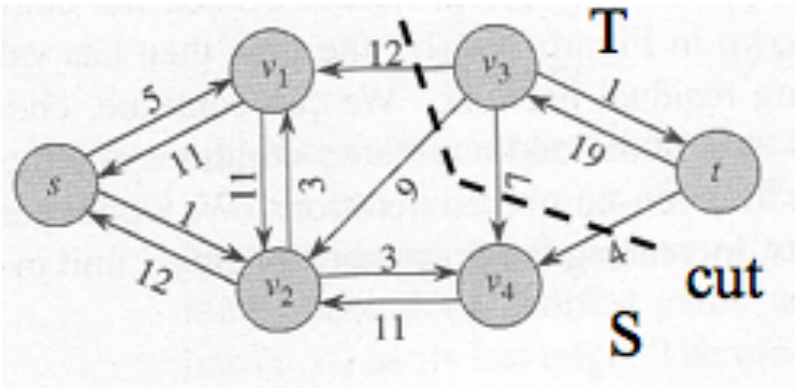
- Since any flow has to cross every cut, any flow must be smaller than the capacity of any cut: $f = f(S,T) \leq c(S,T)$.
- Thus, if we find a flow f which equals the capacity of some cut (S,T), then such flow must be maximal and the capacity of such a cut must be minimal.

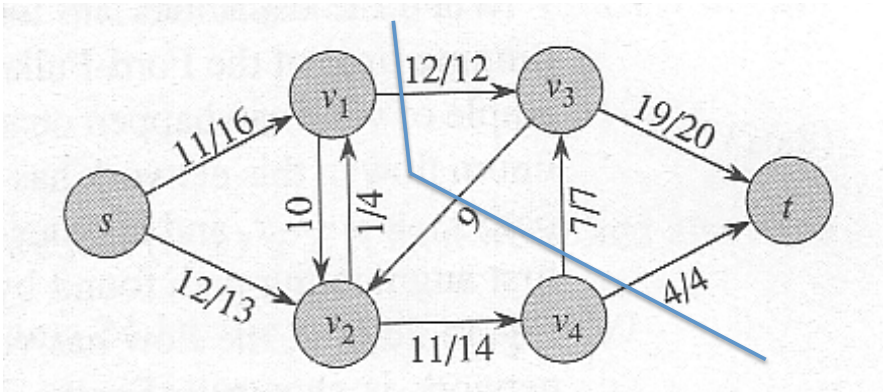


- We now show that when the Ford - Fulkerson algorithm terminates, it produces a flow equal to the capacity of an appropriately defined cut.

Cuts in flow networks

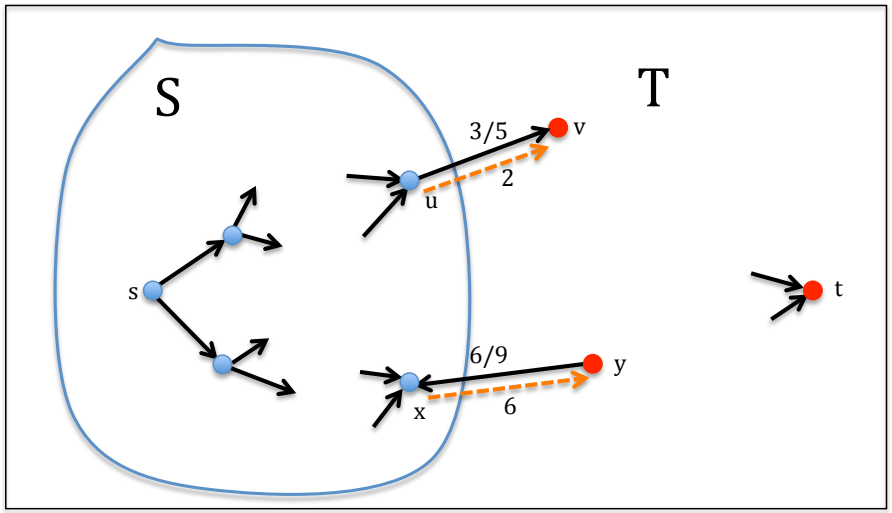
- Assume that the Ford - Fulkerson algorithm has terminated and that there no more augmenting paths from the source s to the sink t in the last residual network flow.
- Define S to be the source s and all vertices u such that there is a path in the residual network flow from the source s to that vertex u.
- Define T to be the set of all vertices for which thee is no such path.
- Since there are no more augmenting paths from s to t, clearly the sink t belongs to T.





Cuts in flow networks

- **Claim:** all the edges from S to T are fully occupied with flow, and all the edges from T to S are empty.



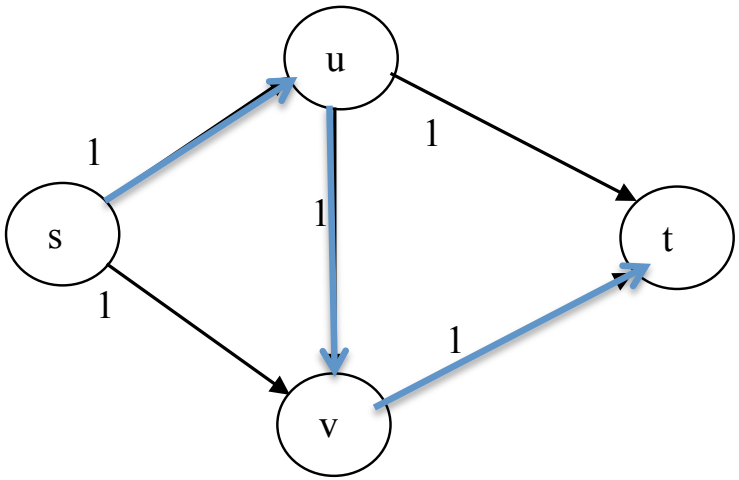
- **Proof:**
  - If an edge (u, v) from S to T had some additional capacity left, then in the residual flow network the path from s to u could be extended to a path from s to v which contradict our assumption that $v \in T$.
  - If and edge (y, x) from T to S had any flow in it, then in the residual flow network the path from s to x could be extended to a path from x to y, which is again a contradiction with our assumption that $y \in T$.

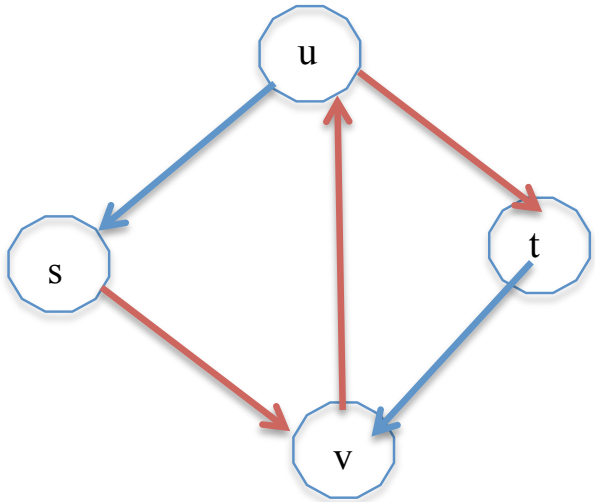
Cuts in flow networks

- Since all edges from S to T are occupied with flows to their full capacity and since there is no flow from T to S, the flow across the cut (S, T) is precisely equal to the capacity of this cut.

- Thus, such a flow is maximal and the corresponding cut is a minimal cut, regardless of the particular way how the augmenting paths were chosen.

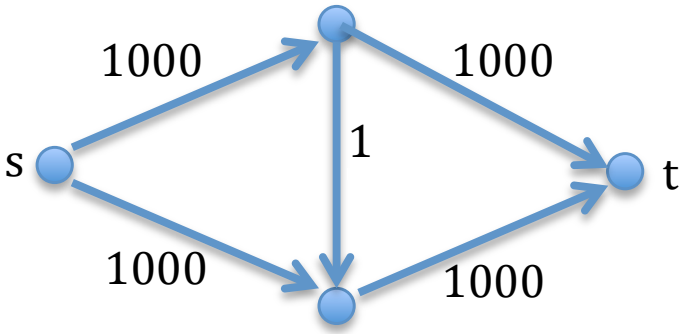
- Trying to do the Ford Fulkerson algorithm without constructing the residual flow can make you miss augmenting paths:

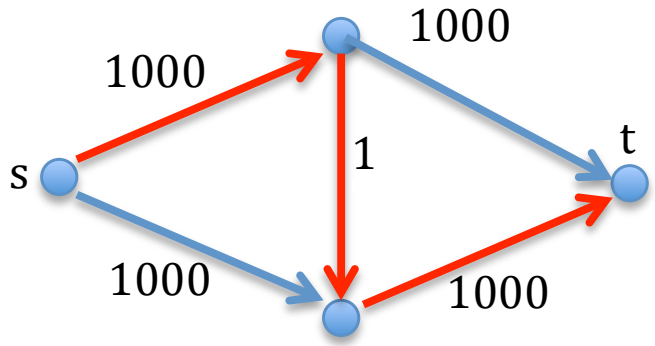


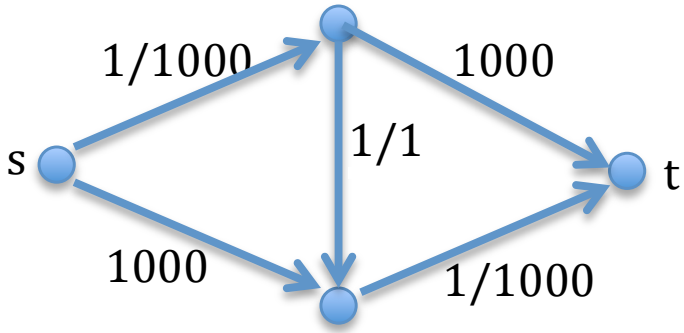


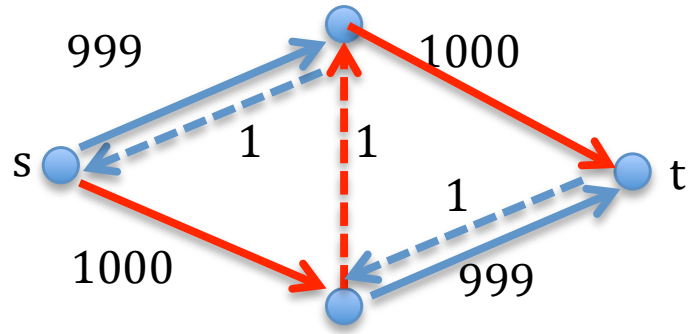
Cuts in flow networks

- How efficient is the Ford-Fulkerson algorithm?

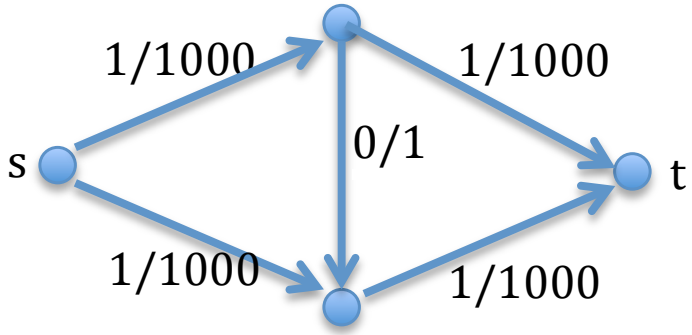


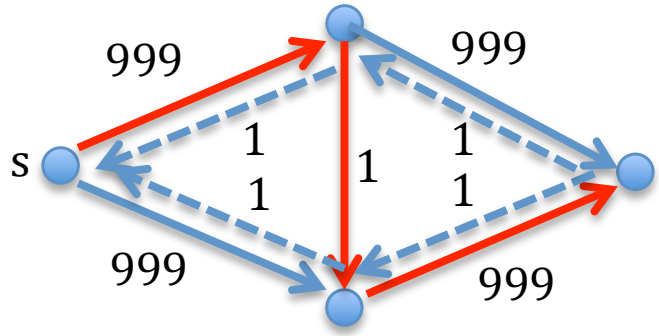


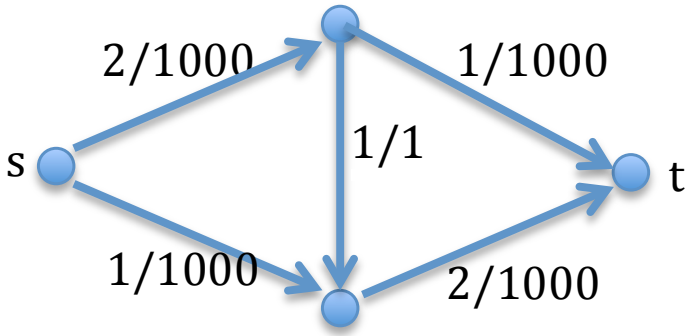




Cuts in flow networks







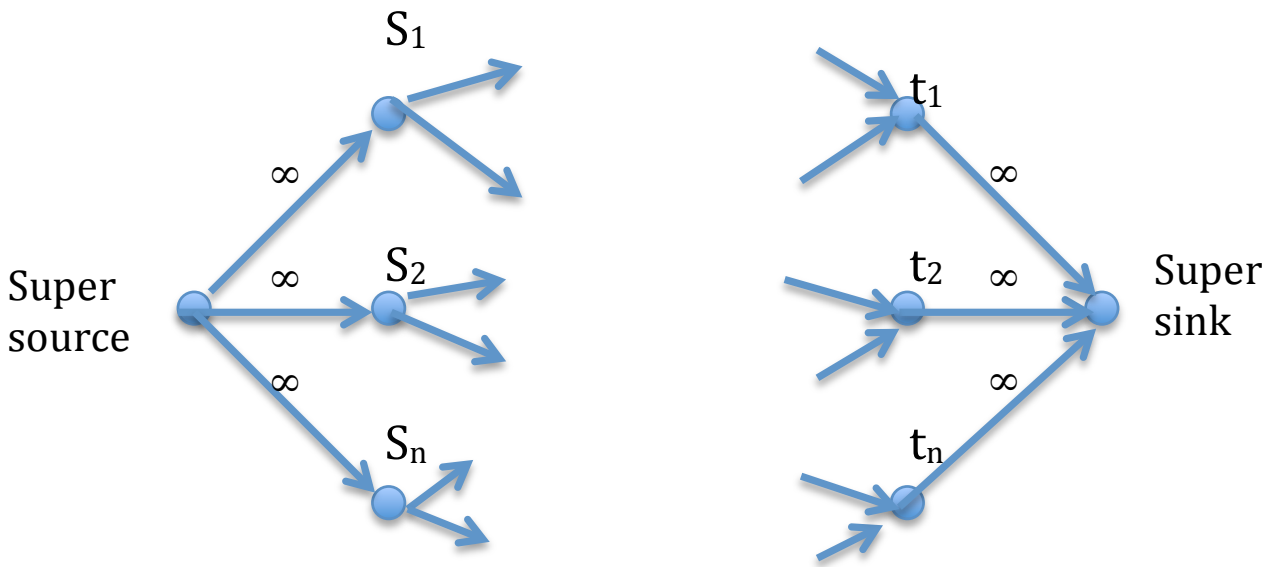
- The Ford-Fulkerson algorithm can potentially run in time proportional to the value of max flow, which can be exponential in the size of the input.

Edmonds-Karp Max Flow Algorithm

- The Ford-Fulkerson algorithm can potentially run in time proportional to the value of max flow, which can be exponential in the size of the input.
- The Edmonds-Karp algorithm improves the Ford Fulkerson algorithm in a simple way: always choose the shortest path from the source s to the sink t, where the "shortest path" means the fewest number of edges, regarding of their capacities (i.e., each edge has the same unit weight).
- Note that this choice is somewhat counter intuitive: we preferably take edges with small capacities over edges with large capacities, for as long as they are along the shortest path from s to t.
- Why does such a choice speed up the Ford - Fulkerson algorithm? To see this, one needs a tricky mathematical proof, see the textbook. One can prove that such algorithm runs in time $O(|V|\,|E|^2)$ time.

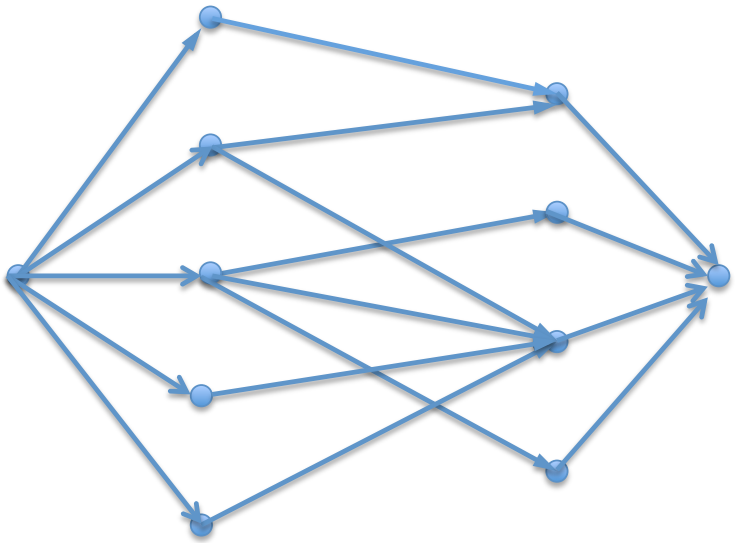
Networks with multiple sources and sinks

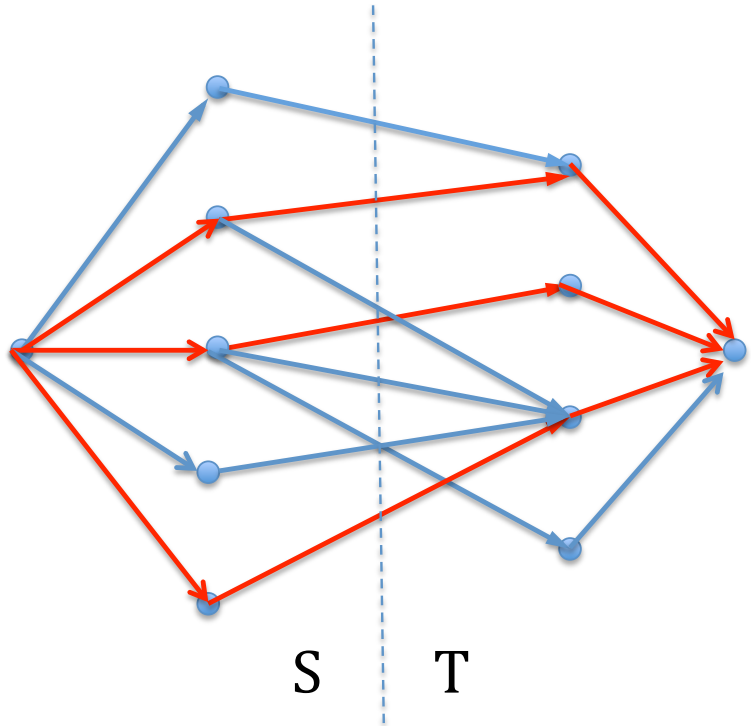
- Flow networks with multiple sources and sinks are reducible to networks with a single source and single sink by adding a "super-sink" and "super-source" and connecting them to all sources and sinks, respectively, by edges of infinite capacities.



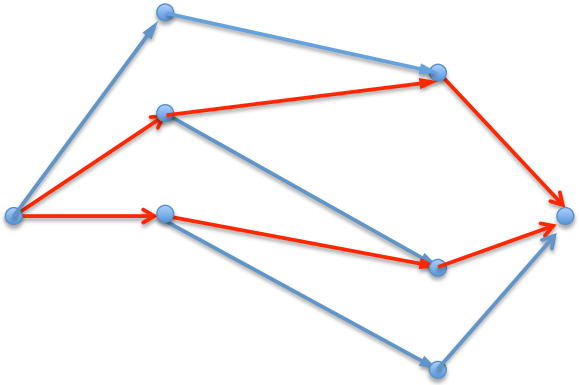
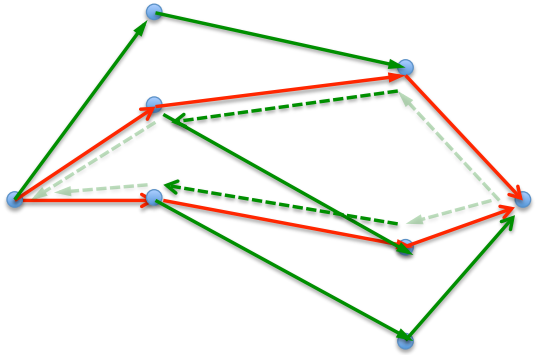
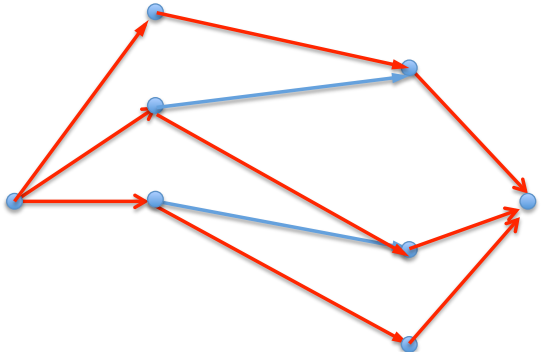
Maximum matching in bipartite graphs

- We turn a Maximum Matching problem into a Max Flow problem by adding a super source and a super sink, and by giving all existing edges a capacity of 1
- A matching in a graph G is a subset M of all edges E such that each vertex of the graph belongs to at most one of the edges in the matching M.





Maximum matching in bipartite graphs

- Note how the residual flow network allows rerouting the flow in order to increase the total throughput.

Other application of Max Flow algorithms

- Assume you have a movie rental agency. At the moment you have k movies in stock, with m_i copies of movie i. Each of n customers can rent out at most 5 movies at a time. The customers fave sent you their preferences which a list of movies they would like to see. Your goal is to dispatch the largest possible number of movies.