Agenda

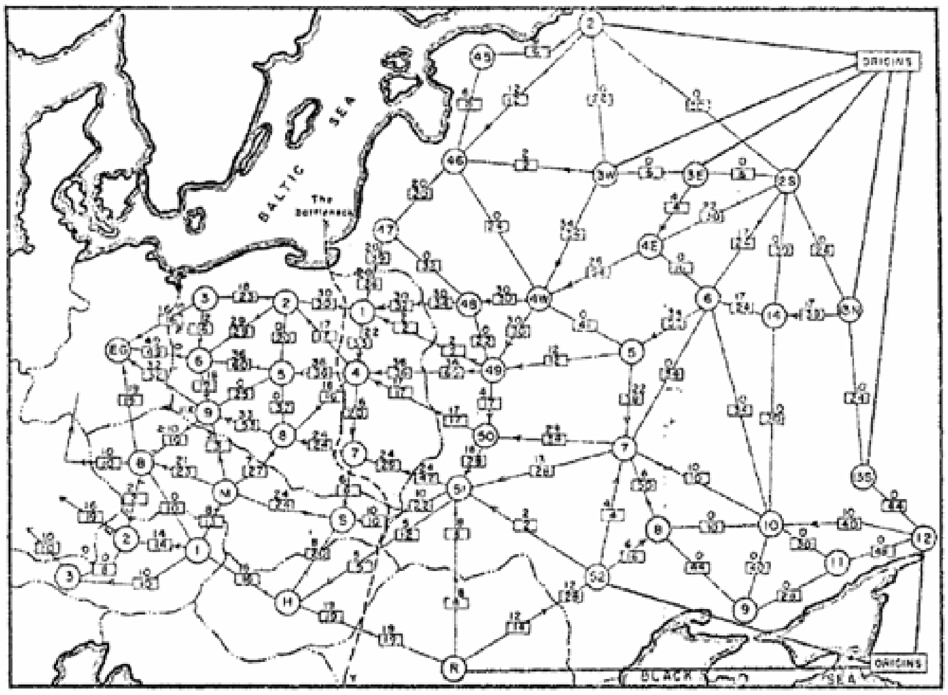
We've done

- Greedy Method
- Divide and Conquer
- Dynamic Programming

Now

- Flow Networks, Max-flow Min-cut and Applications

Soviet Rail Network, 1955



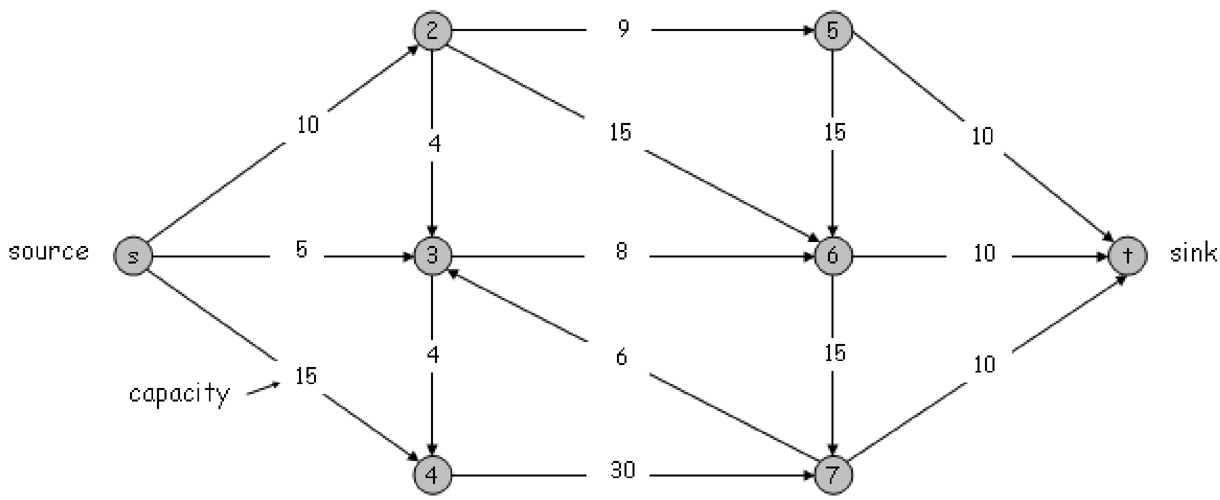
Reference: *On the history of the transportation and maximum flow problems.* Alexander Schrijver in Math Programming, 91: 3, 2002.

Maximum Flow and Minimum Cut Problems

- Cornerstone problems in combinatorial optimization
- Many non-trivial applications/reductions: airline scheduling, data mining, bipartite matching, image segmentation, network survivability, many many many more ...
- Simple Example: on the Internet with error-free transmission, what is the maximum data rate that a router s can send to a router t (assuming no network coding is allowed), given that each link has limited capacity
- More examples and applications to come

Flow Networks

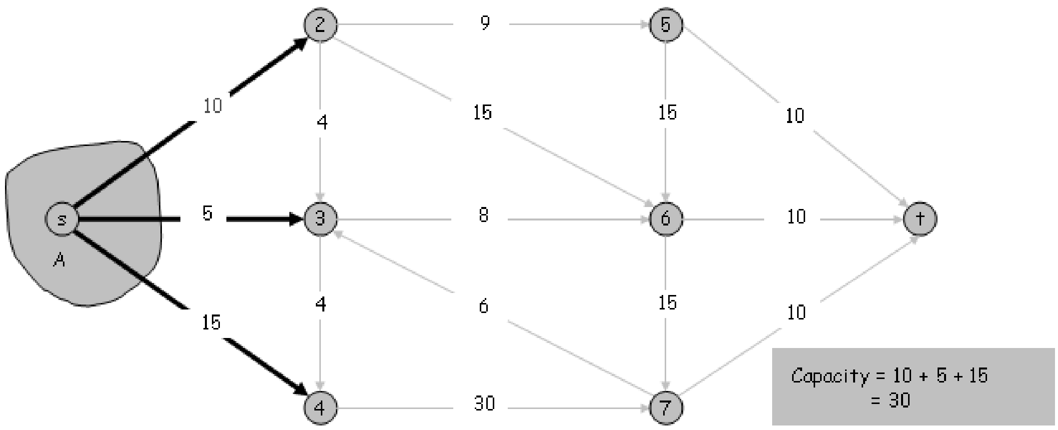
- A flow network is a directed graph $G = (V, E)$ where each edge e has a capacity $c(e) > 0$
- Also, there are two distinguished nodes: the source s and the sink t



Cuts

- An s,t-cut is a partition (A,B) of V where $s \in A$, $t \in B$
- Let $[A,B]$ = set of edges (u,v) with $u \in A, v \in B$
- The capacity of the cut (A,B) is defined by

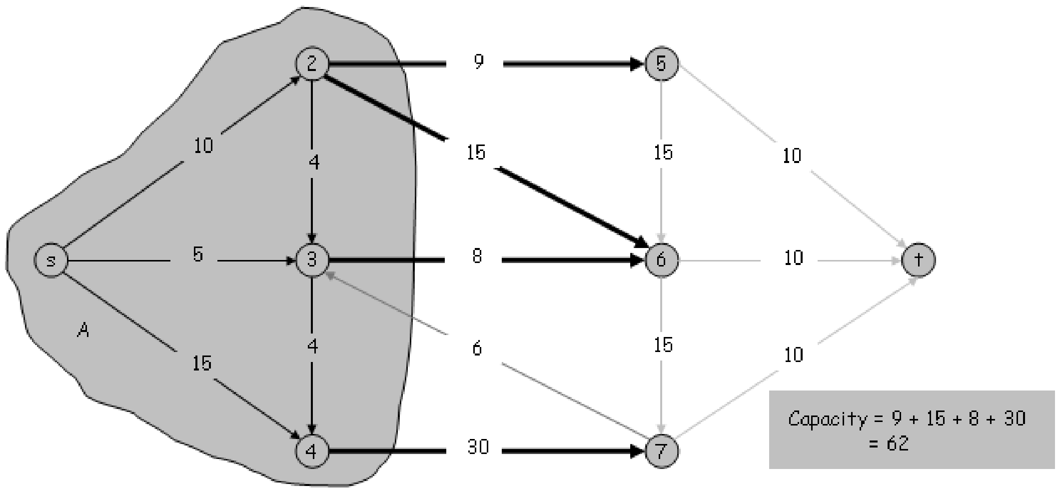
$$\text{cap}(A,B) = \sum_{e \in [A,B]} c(e)$$



Cuts

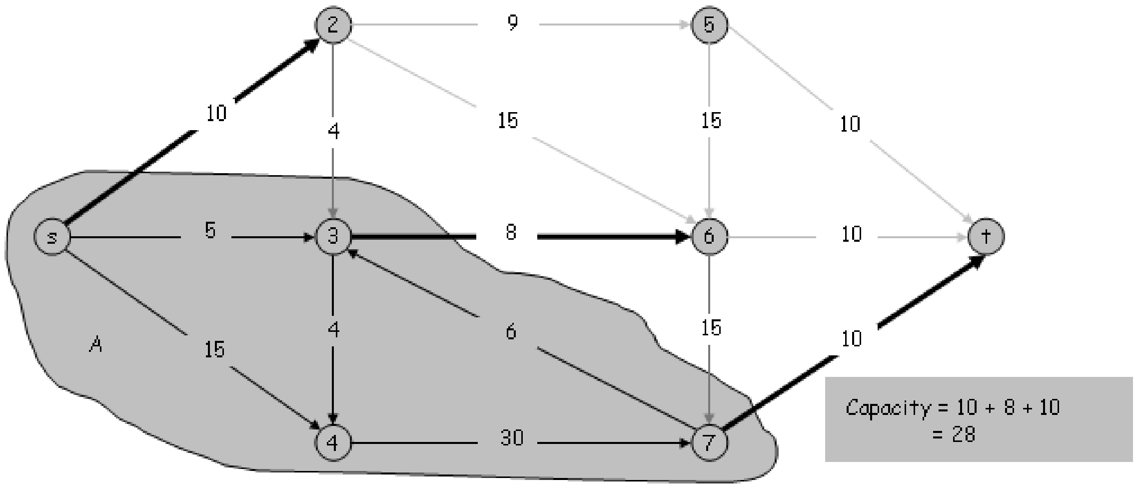
- An s,t-cut is a partition (A,B) of V where $s \in A$, $t \in B$
- Let $[A,B]$ = set of edges (u,v) with $u \in A, v \in B$
- The capacity of the cut (A,B) is defined by

$$\text{cap}(A,B) = \sum_{e \in [A,B]} c(e)$$



Minimum Cut – Problem Definition

Given a flow network, find an s,t-cut with minimum capacity



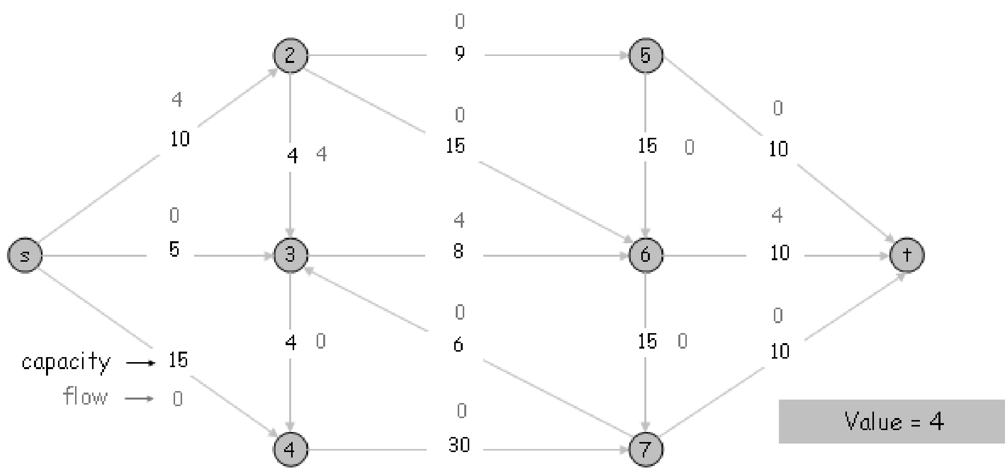
Flows

An s,t-flow is a function $f : E \to \mathbb{R}$ satisfying

– Capacity constraint: $0 \leq f(e) \leq c(e)$, $\forall e \in E$

– Flow Conservation constraint: $\displaystyle\sum_{e=(u,v)\in E} f(e) = \sum_{e=(v,w)\in E} f(e), \forall v \neq s,t$

The value of f: $\text{val}(f) = \displaystyle\sum_{e=(s,v)\in E} f(e)$



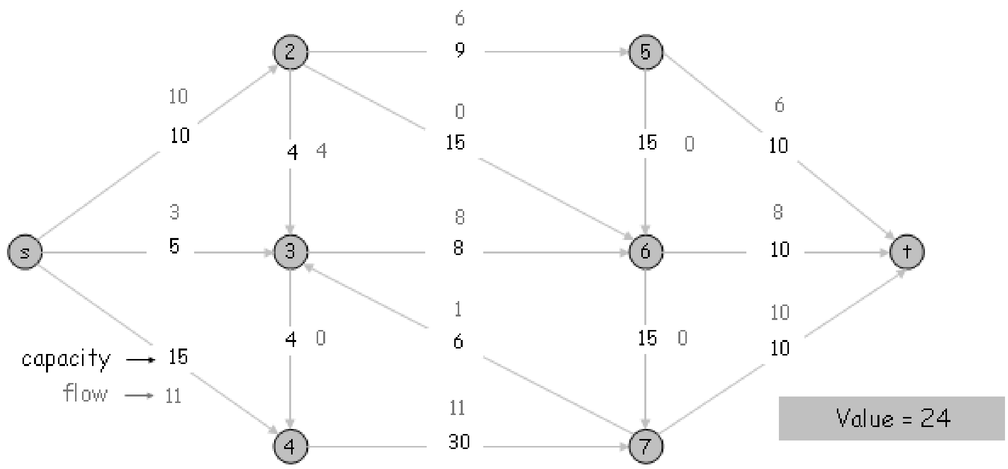
Flows

An s,t-flow is a function $f : E \to \mathbb{R}$ satisfying

– Capacity constraint: $0 \leq f(e) \leq c(e)$, $\forall e \in E$

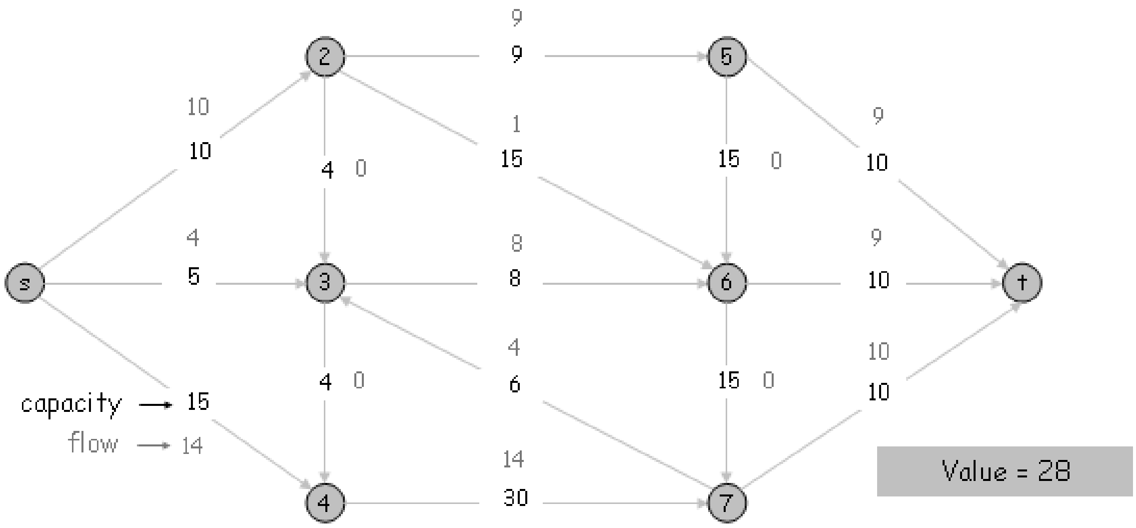
– Flow Conservation constraint: $\displaystyle\sum_{e=(u,v)\in E} f(e) = \sum_{e=(v,w)\in E} f(e), \forall v \neq s,t$

The value of f: $\text{val}(f) = \displaystyle\sum_{e=(s,v)\in E} f(e)$



Maximum Flow – Problem Definition

Given a flow network, find a flow f with maximum capacity

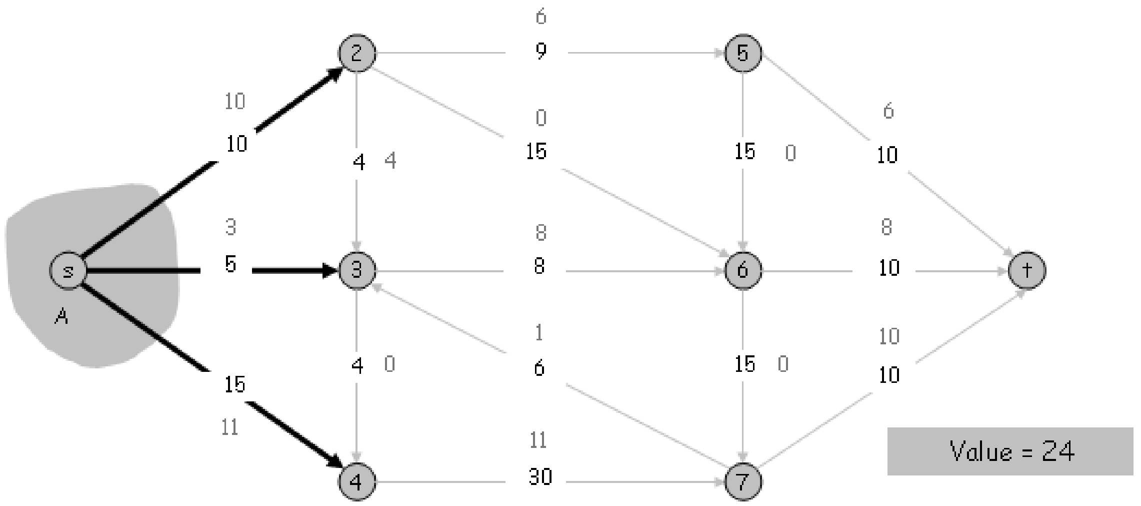


Flows and Cuts

Lemma (Flow Value Lemma)

For any flow f and any cut (A, B)

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = \text{val}(f)$$

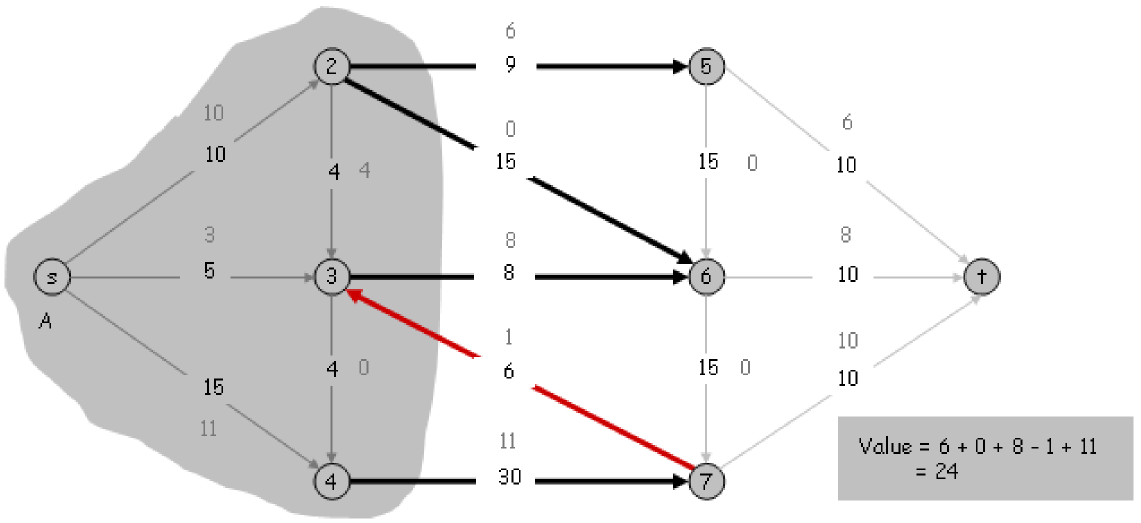


Flows and Cuts

Lemma (Flow Value Lemma)

For any flow f and any cut (A, B)

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = \text{val}(f)$$

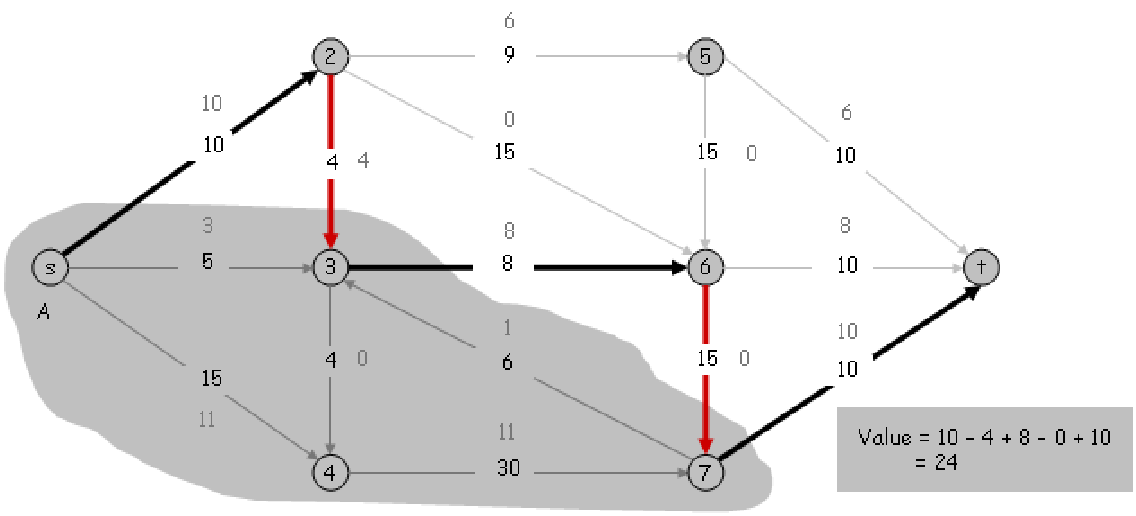


Flows and Cuts

Lemma (Flow Value Lemma)

For any flow f and any cut (A, B)

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = \text{val}(f)$$

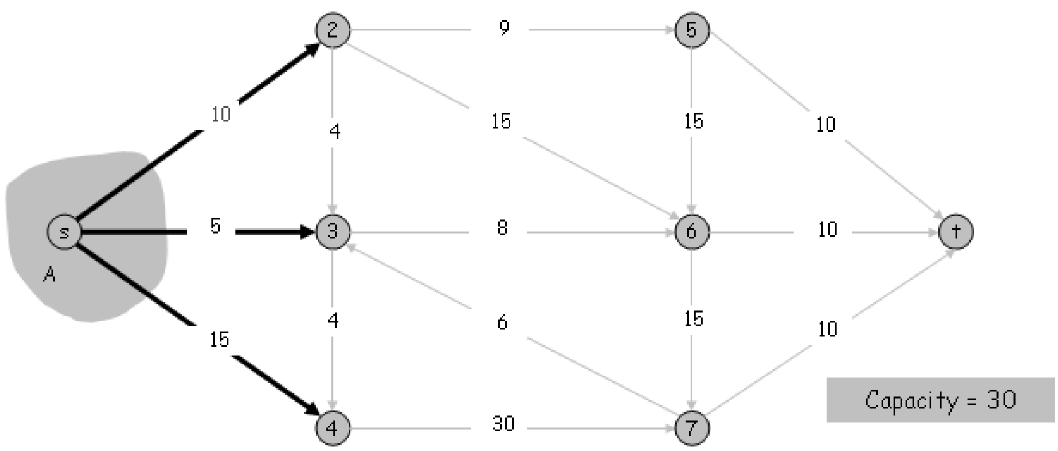


Weak Duality

Lemma (Weak Duality)

Given any s, t-flow f and any s, t-cut (A, B), the flow value is at most the cut capacity: $\text{val}(f) \leq \text{cap}(A, B)$

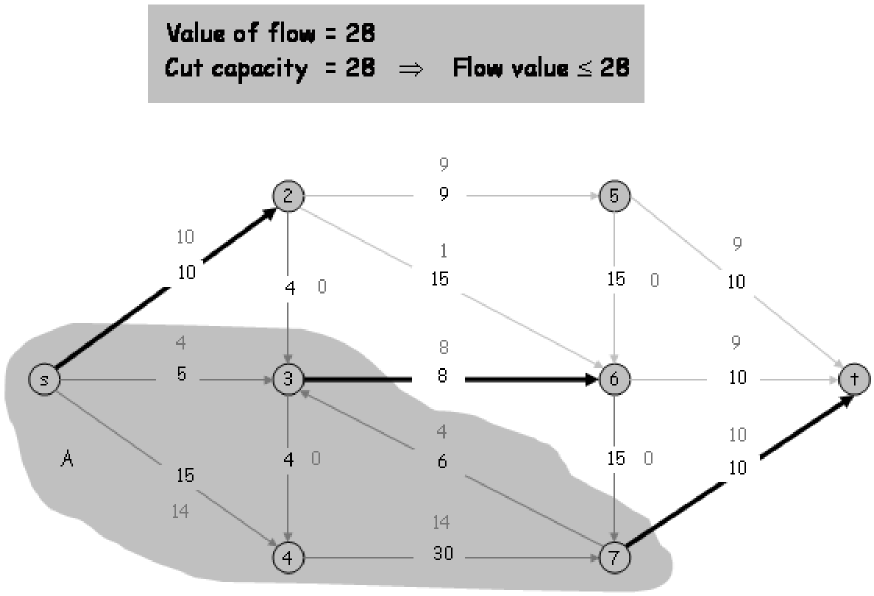
Cut capacity = 30 $\Rightarrow$ Flow value $\leq$ 30



Certificate of Optimality

Corollary

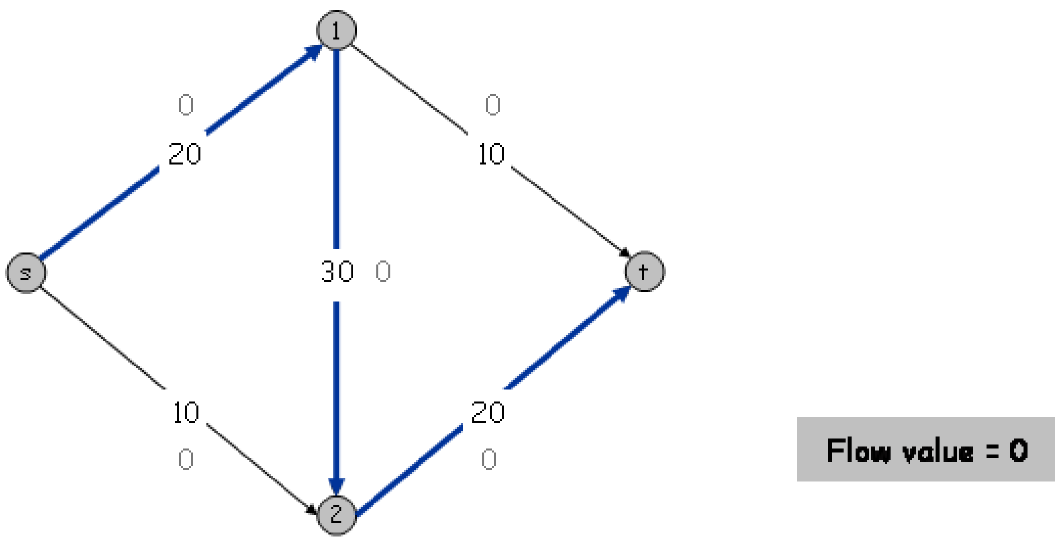
If val(f) = cap(A, B) *for any flow* f *and any cut* (A, B)*, then* f *is a maximum flow and* (A, B) *is a minimum cut*



Computing Max Flow - First Attempt

A greedy algorithm:

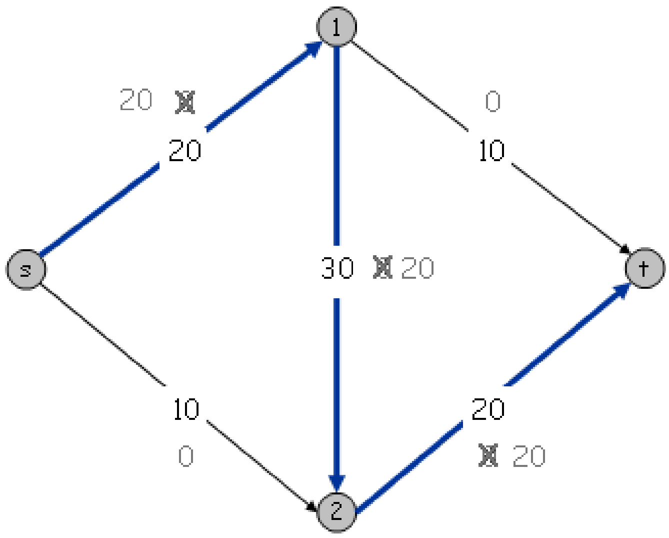
- start with $f(e) = 0, \forall e$
- find a path P with $f(e) < c(e)$ for all e on the path
- augment flow along P
- repeat until stuck



Computing Max Flow - First Attempt

A greedy algorithm:

- start with $f(e) = 0, \forall e$
- find a path P with $f(e) < c(e)$ for all e on the path
- augment flow along P
- repeat until stuck

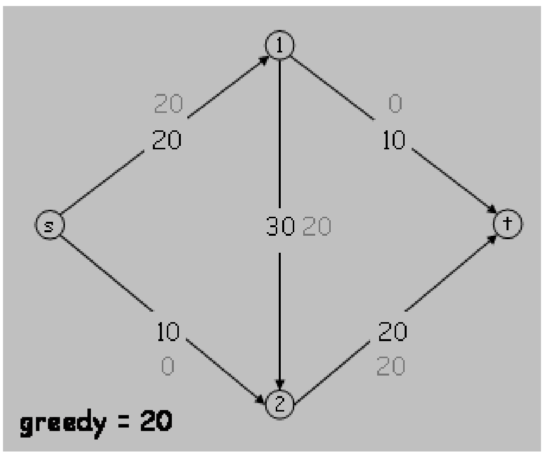


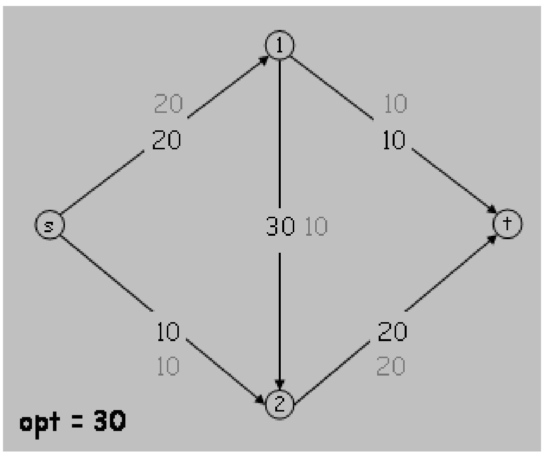
Flow value = 20

Computing Max Flow - First Attempt

A greedy algorithm:

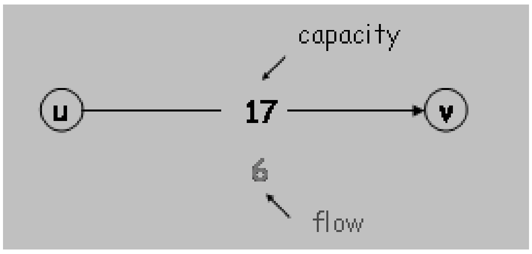
- start with $f(e) = 0, \forall e$
- find a path P with $f(e) < c(e)$ for all e on the path
- augment flow along P
- repeat until stuck (local opt $\not\Rightarrow$ global opt)

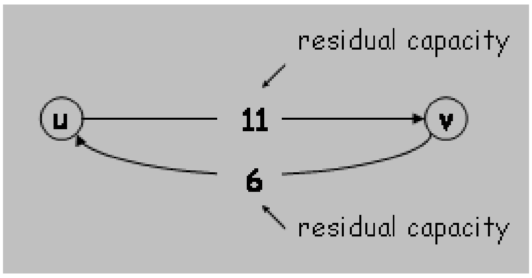




Residual Graph

- Define $G_f = (V, E_f)$ for each flow f, each edge in E_f has a residual capacity $c_f(e)$
- E_f and c_f are determined as follows
  - Original edge $e = (u, v) \in E$, flow $f(e)$, capacity $c(e)$
  - If $f(e) < c(e)$, then $e \in E_f$ and $c_f(e) = c(e) - f(e)$
  - If $f(e) > 0$, $e = (u, v)$, then $e' = (v, u) \in E_f$ and $c_f(e') = f(e)$





Ford-Fulkerson (Augmenting Path) Algorithm

AUGMENT(f, c, P)

```
b ← bottleneck(P), i.e. min residual capacity on P
for each edge e = (u, v) on P do
    if e is a forward edge then
        Increase f(e) in G by b
    else
        Decrease f(e) in G by b
    end if
end for
```

FORD-FULKERSON(G, c)

```
Initially, set f(e) = 0 for all e ∈ E
while there is an s,t-path P in G_f do
    Choose a simple s,t-path P in G_f (crucial for running time!)
    f ← AUGMENT(f, c, P)
end while
```

Max-Flow Min-Cut Theorem

Theorem (Ford-Fulkerson, 1956)

The value of a max-flow is equal to the capacity of a min-cut

Proof.

Let f be any feasible flow, the following are equivalent

- f is a maximum flow
- there's no augmenting path wrt f
- there's a cut (A, B) where $\text{val}(f) = \text{cap}(A, B)$

□

Termination and Running Time

If $1 \leq c(e) \leq C \in \mathbb{N}$, and $c(e) \in \mathbb{N}$ for all e, then all flow values and residual capacities remain integers throughout

Theorem

Number of iterations is at most $\text{val}(f^*)$, *which is at most* nC

Corollary

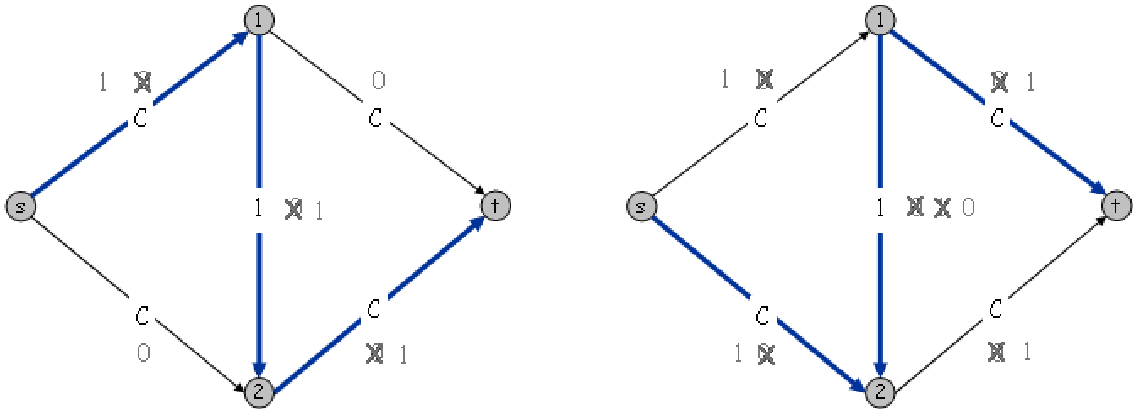
If $C = 1$, *i.e.* $c(e) = 1$ *for all* e, *then Ford-Fulkerson runs in time* $O(mn)$

Theorem (Ingegrality Theorem)

If all capacities are integers, then there exists an integral maximum flow, i.e. a flow whose $f(e)$ *are all integers.*

Generic Ford-Fulkerson: Exponential Running Time

- It could take C iterations.
- Recall: input size is a polynomial in $m, n, \log C$



Choosing Good Augmenting Paths

Augmenting path selection:

- Bad choices lead to exponential algorithms
- Good choices lead to polynomial-time algorithms
- If capacities are irrational, may not even terminate at all

Some good choices [Edmonds-Karp 1972, Dinitz 1970]

- Max bottleneck capacity
- Sufficiently large bottleneck capacity
- Fewest number of edges

Some Strategies

Choose augmenting path with

- no specific strategy $\Rightarrow O(mC)$
- sufficiently large bottleneck capacity $\Rightarrow O(m^2 \log C)$
- maximum bottleneck capacity $\Rightarrow O(m \log C)$
- shortest length $\Rightarrow O(m^2 n)$

Note: there are also strategies not based on the augmenting path method.

The Edmonds-Karp Algorithm

- Choose shortest augmenting path

Lemma

Let $d_f(s,u)$ be the distance from s to u in G_f, then $d_f(s,u)$ increases monotonically with each augmentation

Theorem

The Edmonds-Karp algorithm makes at most $O(mn)$ augmentations, in particular its running time is $O(m^2 n)$

Proof of the Lemma

- Suppose augmenting f gives f' for which some $d_f(s,v) > d_{f'}(s,v)$.
- Let v be such a vertex with smallest $d_{f'}(s,v)$, and $P = s \rightsquigarrow u \to v$ is a path with length $d_{f'}(s,v)$
- Then,

$$\begin{aligned} d_{f'}(s,u) &= d_{f'}(s,v) - 1 \\ d_{f'}(s,u) &\geq d_f(s,u) \end{aligned}$$

- Thus, $(u,v) \notin E_f$; but $(u,v) \in E_{f'}$, hence $(v,u) \in E_f$ and Edmonds-Karp pushed some flow from v to u in f
- Since the flow is pushed along a shortest path, we have a contradiction

$$d_f(s,v) + 1 = d_f(s,u) \leq d_{f'}(s,u) = d_{f'}(s,v) - 1$$

Proof of the Theorem

- For each augmentation, some bottleneck edge (u,v) in G_f will disappear in $G_{f'}$, where f' is the next flow
- Suppose (u,v) is a bottleneck edge a few times, then there will be a time when (u,v) is a bottleneck for some f and later (v,u) is a bottleneck for some f'. We have

$$\begin{aligned} d_f(u) &= d_f(v) - 1 \\ d_{f'}(v) &= d_{f'}(u) - 1 \end{aligned}$$

- Thus,

$$d_f(u) = d_f(v) - 1 \leq d_{f'}(v) - 1 = d_{f'}(u) - 2$$

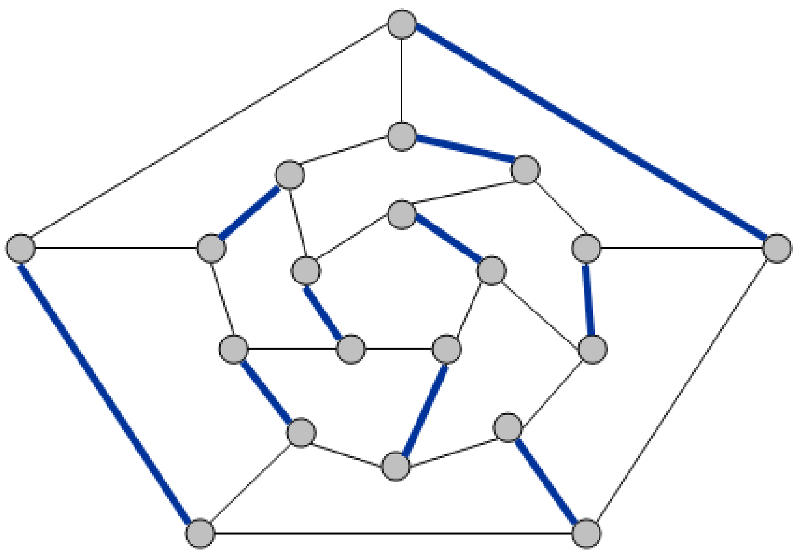
- Each time (u,v) becomes a bottleneck, $d_f(s,u)$ is increased by at least 2; thus, the number of times (u,v) is a bottleneck is at most $(n-2)/2$.

General Idea for Employing Max-Flow Min-Cut

- Set up a new problem as a network flow problem
- Use max-flow algorithm to solve new problem
- and/or Apply max-flow min-cut and integrality theorems to derive some combinatorial properties of the new problem

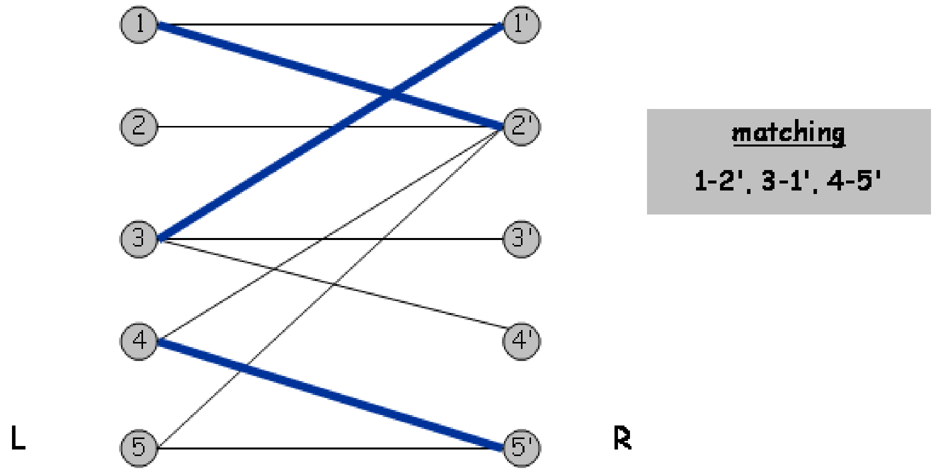
Maximum Matching in Graphs

- $G = (V, E)$, a matching is a subset $M \subset E$ no two of which share an end point
- Maximum matching: find a maximum cardinality matching
  - This is a fundamental problem in combinatorial optimization with numerous applications



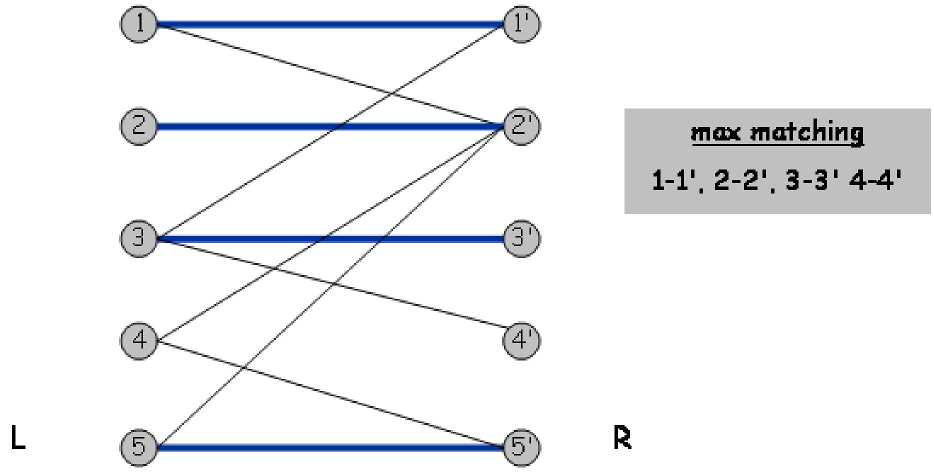
Maximum Matching in Bipartite Graphs

- Given a bipartite graph $G = (L \cup R, E)$, find a $\max$ matching



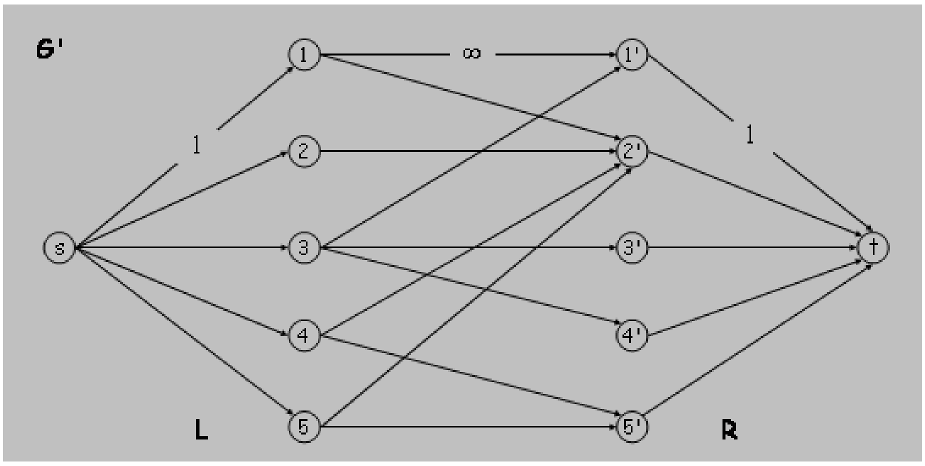
Maximum Matching in Bipartite Graphs

- Given a bipartite graph $G = (L \cup R, E)$, find a $\max$ matching



Max-Flow Formulation for Bipartite Matching

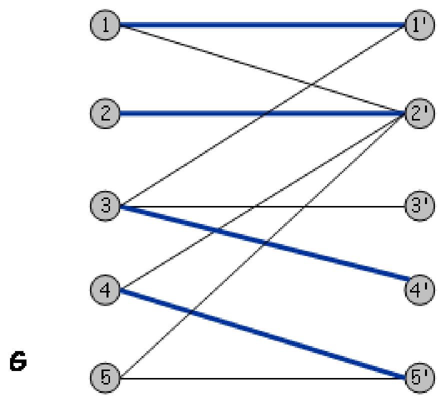
- Create a new digraph $G' = (V \cup \{s,t\}, E')$ as follows
- Orient edges from left to right (L to R) with capacities ∞ (or any positive integer, doesn't matter which)
- Add a fake source s, fake sink t, and edges with capacities 1 as shown

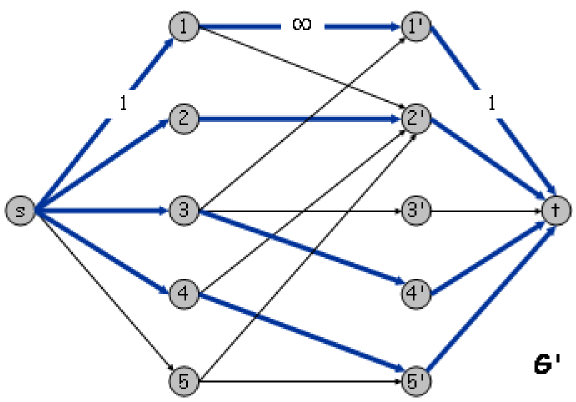


Correctness of the Formulation

Theorem

The maximum matching cardinality of G is equal to the maximum flow value of G'. Moreover, Ford-Fulkerson yields a maximum matching.





Running Times of Matching Algorithms

Bipartite Matching

- Generic Ford-Fulkerson: $O(mn)$ – pretty good!
- Largest Bottleneck Path : $O(m^2)$
- Edmonds-Karp: $O(m\sqrt{n})$
- ...

Non-bipartite Matching

- More difficult, but very well studied
- Blossom algorithm (Edmonds, 1964): $O(n^4)$
- Best known (Micali-Vazirani, 1980): $O(m\sqrt{n})$

Marriage Theorem

- Given a bipartite $G = (L \cup R, E)$.
- A complete matching from L into R is a matching in which every vertex in L is matched.
- A perfect matching is a matching in which every vertex is matched
- Questions: When does G have a complete matching? When does it have a perfect matching?
- $\exists$ a perfect matching iff $|L| = |R|$ and $\exists$ a complete matching

Theorem (P. Hall 1935, Frobenius 1917, König 1916)

Let $\Gamma(X)$ denote the set of neighbors of $X \subset L$, then G has a complete matching iff $|\Gamma(X)| \geq |X|, \forall X \subseteq L$.

König-Egerváry Theorem

- Given $G = (V, E)$, a vertex cover is a subset $C \subseteq V$ such that each edge in E has an end point in C
- Let $\tau(G)$ denote the size of a maximum vertex cover, $\nu(G)$ the size of a maximum matching

Theorem (König 1931, Egerváry 1932)

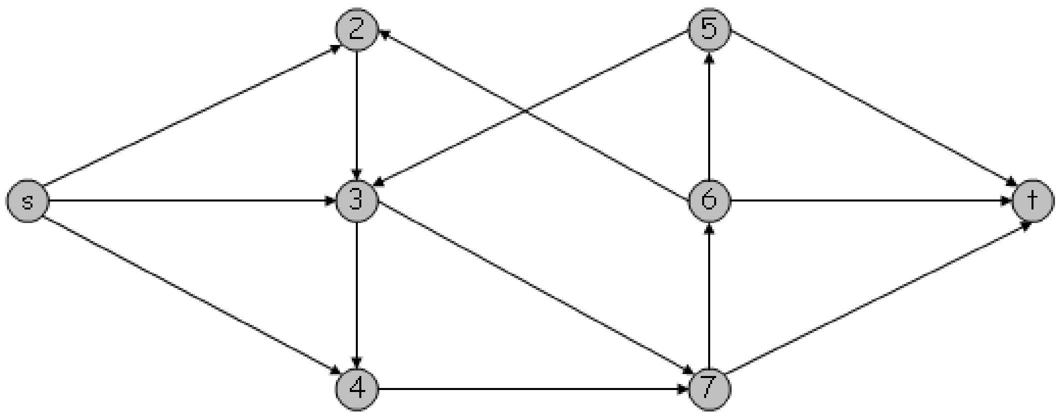
If G is bipartite, then $\tau(G) = \nu(G)$

Proof.

A "direct" consequence of max-flow min-cut. □

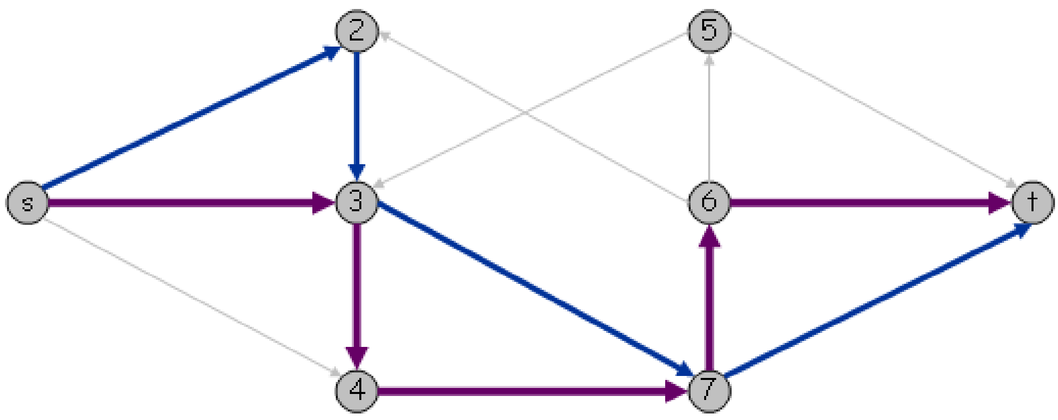
Edge Disjoint Paths in Directed Graphs

- Given a directed graph $G = (V, E)$, a source s and a target t, find the maximum number $\lambda'(s, t)$ of edge-disjoint s, t-paths



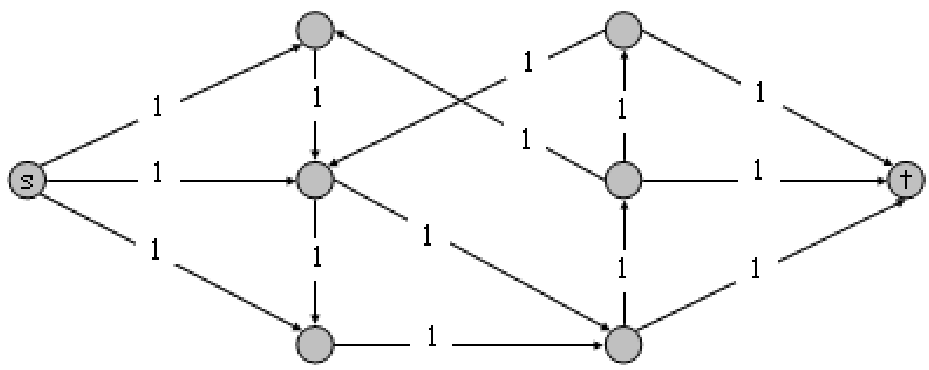
Edge Disjoint Paths in Directed Graphs

- Given a directed graph $G = (V, E)$, a source s and a target t, find the maximum number $\lambda'(s, t)$ of edge-disjoint s, t-paths



Max-Flow Formulation

- Assign capacity 1 to each edge



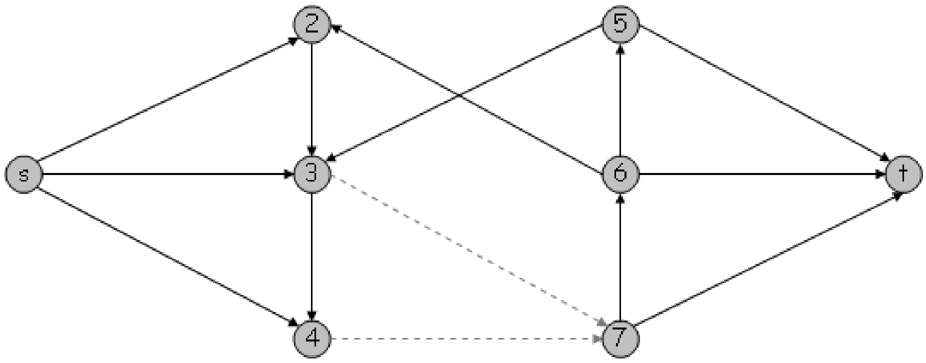
Theorem

The max number of edge-disjoint s, t-paths is equal to the max flow value; moreover, Ford-Fulkerson can find a max set of paths

Note: only need to eliminate cycles from output of max-flow algorithm

Disconnecting Sets

- Given digraph G and s, t, an s, t-disconnecting set is a set of edges whose removal separates s from t, i.e. no s, t-path remains
- Let $\kappa'(s,t)$ denote the minimum size of an s, t-disconnecting set
- Applications: network reliability, among many others

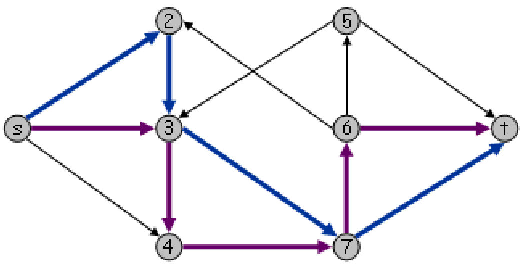


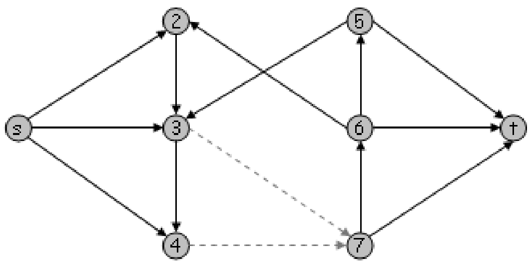
Menger's Theorem

Theorem (Menger 1927)

Given a digraph G and s, t, then $\lambda'(s,t) = \kappa'(s,t)$

Proof: $\lambda'(s,t) \leq \kappa'(s,t)$



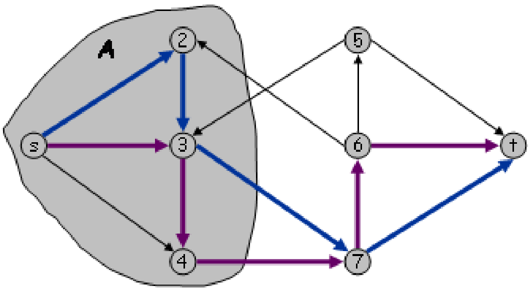


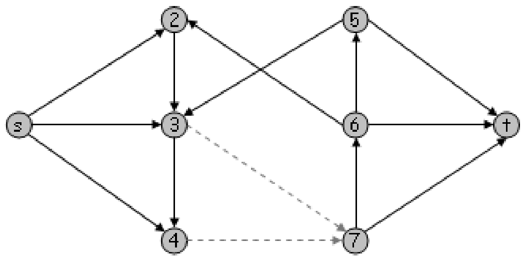
Menger's Theorem

Theorem (Menger 1927)

Given a digraph G and s, t, then $\lambda'(s,t) = \kappa'(s,t)$

Proof: $\lambda'(s,t) \geq \kappa'(s,t)$





Vertex Disjoint Paths in Directed Graphs

- Given digraph G and s, t, an s, t-separating set is a set of vertices whose removal separates s from t
- Let $\kappa(s,t)$ denote the minimum size of an s, t-separating set
- A set of paths from s to t is internally vertex disjoint if they only share vertices s and t; naturally let $\lambda(s,t)$ denote the max number of internally vertex disjoint s, t-paths

Theorem (Menger 1927)

Given a digraph G and s, t, then $\lambda(s,t) = \kappa(s,t)$

Undirected Versions of Menger's Theorem

There are also corresponding versions of Menger's Theorem for undirected graphs