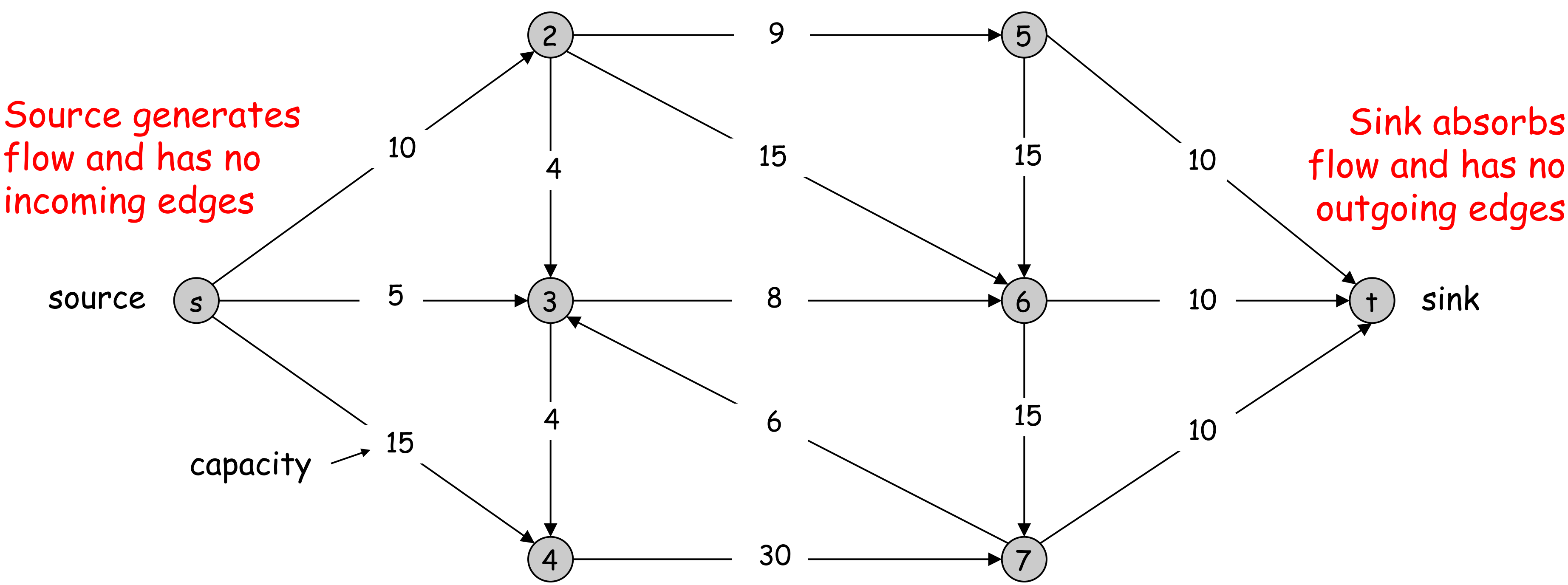
Network Flow

Slides based on Kevin Wayne / Pearson-Addison Wesley

Network Flow

Flow network.

- Abstraction for material flowing through the edges.
  - E.g. traffic on highway, pipes carrying liquid, computer networks
- G = (V, E) = directed graph.
- Two distinguished nodes: s = source, t = sink.
- c(e) = capacity of edge e. $c(e) \geq 0$.

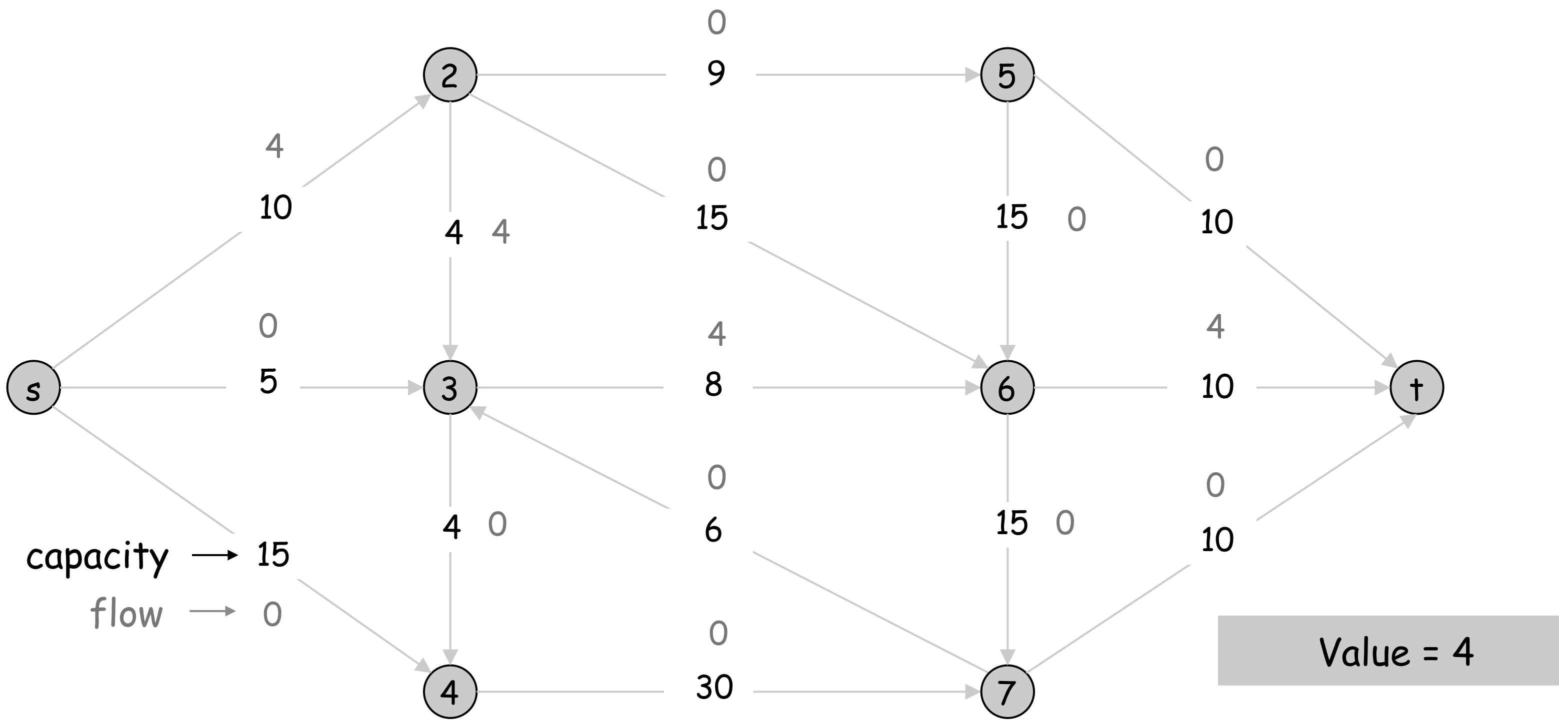


Flows

Def. An s-t flow is a function that satisfies:

- For each $e \in E$: $\quad 0 \le f(e) \le c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\quad \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

Def. The value of a flow f is: $\quad v(f) = \sum_{e \text{ out of } s} f(e)$.

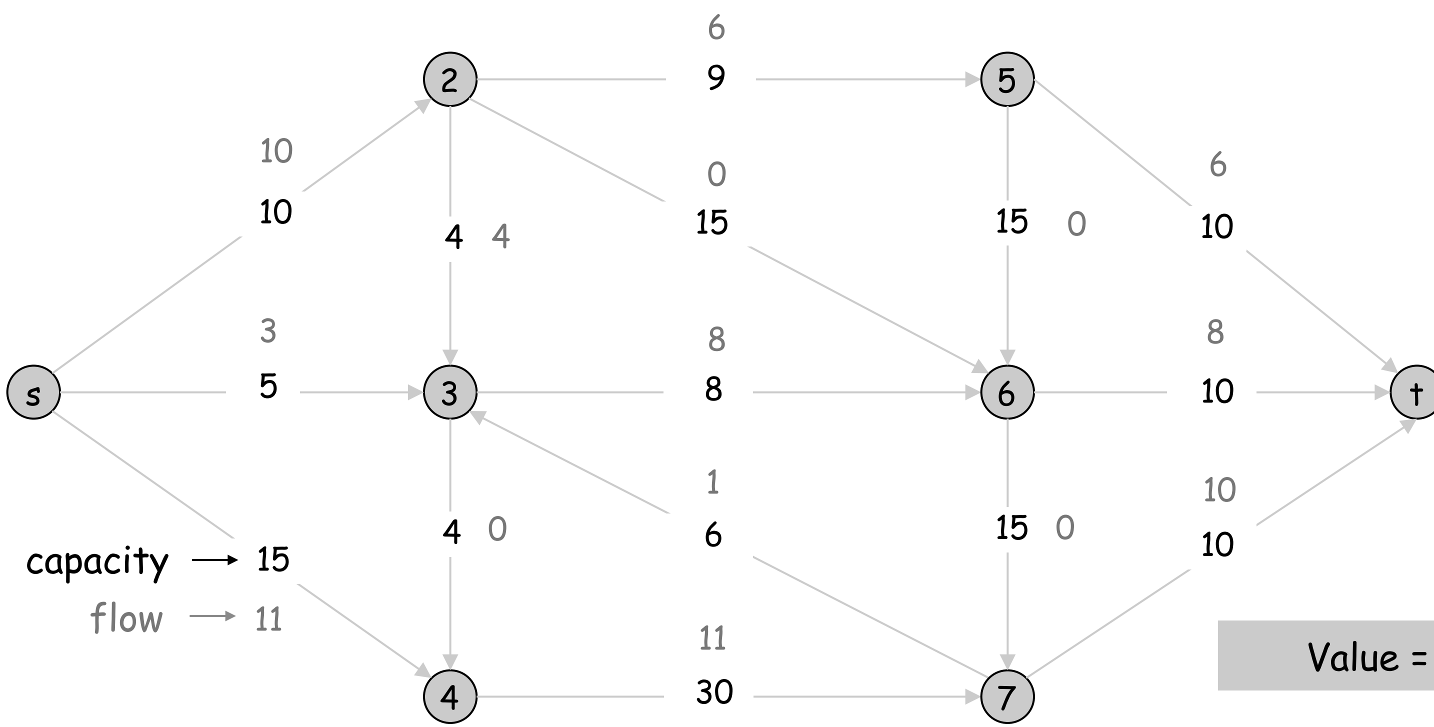


Flows

Def. An s-t flow is a function that satisfies:

- For each $e \in E$: $0 \le f(e) \le c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

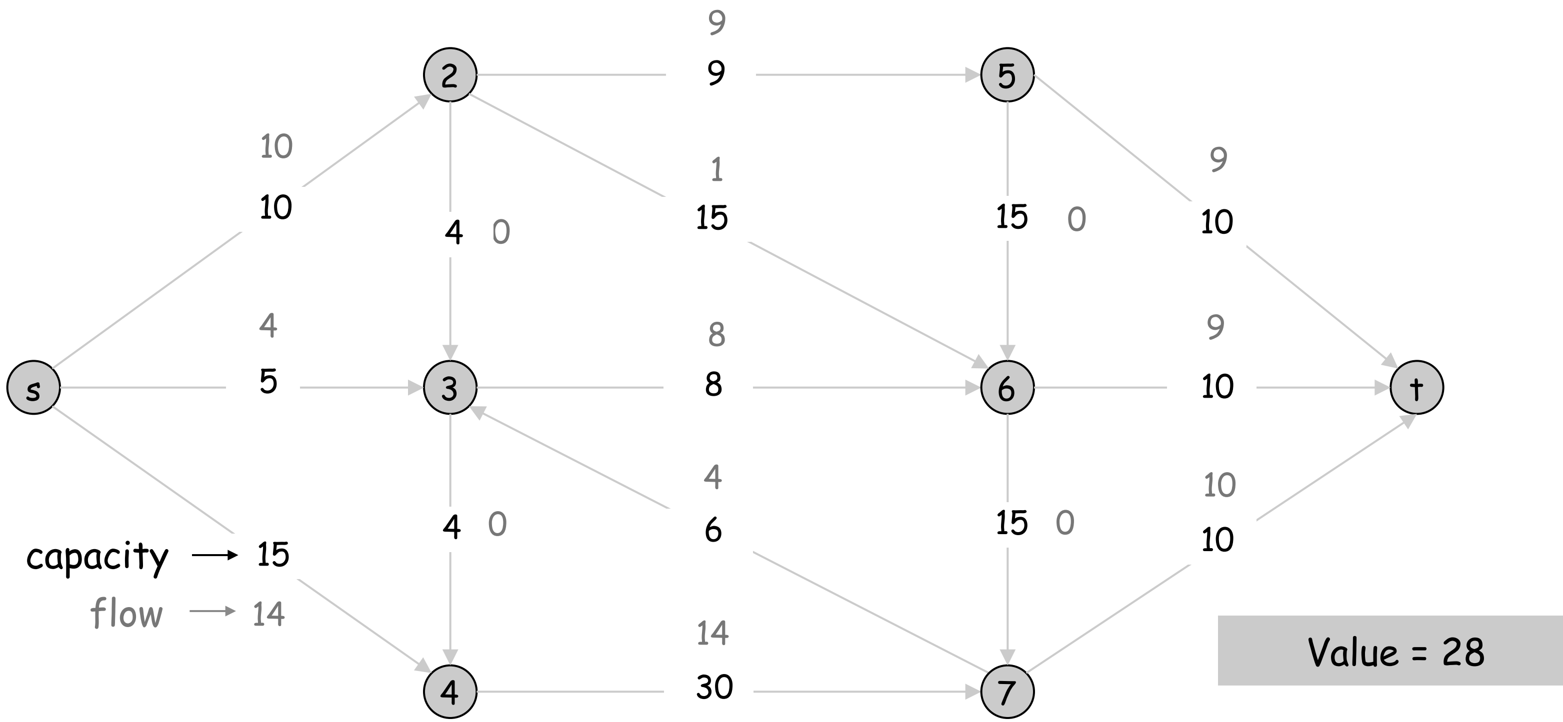
Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Value = 24

Maximum Flow Problem

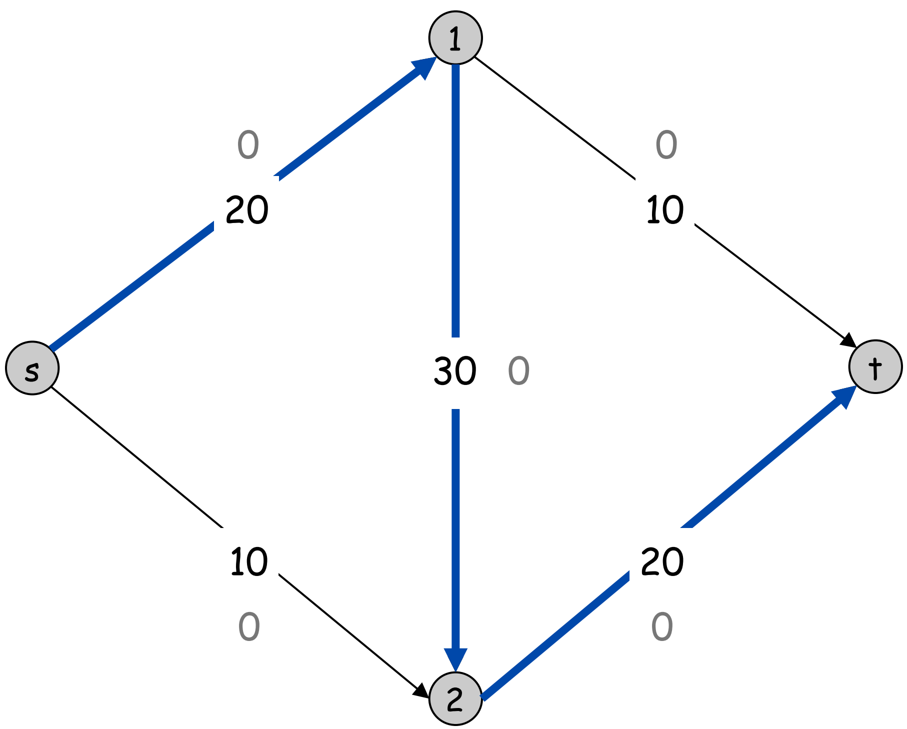
Max flow problem. Find s-t flow of maximum value.



Towards a Max Flow Algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s-t path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

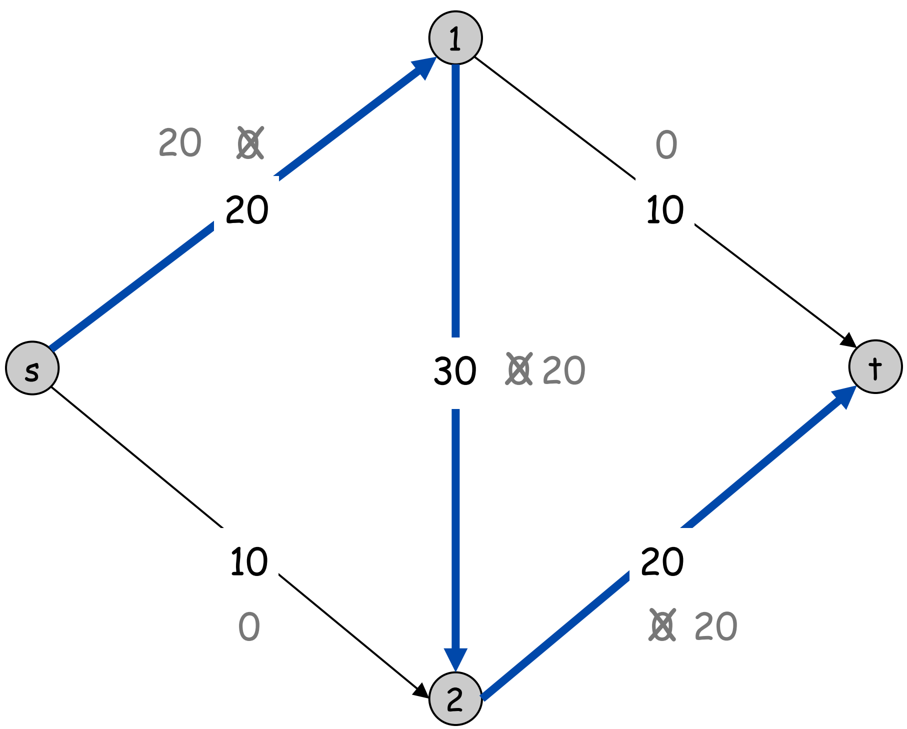


Flow value = 0

Towards a Max Flow Algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s-t path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



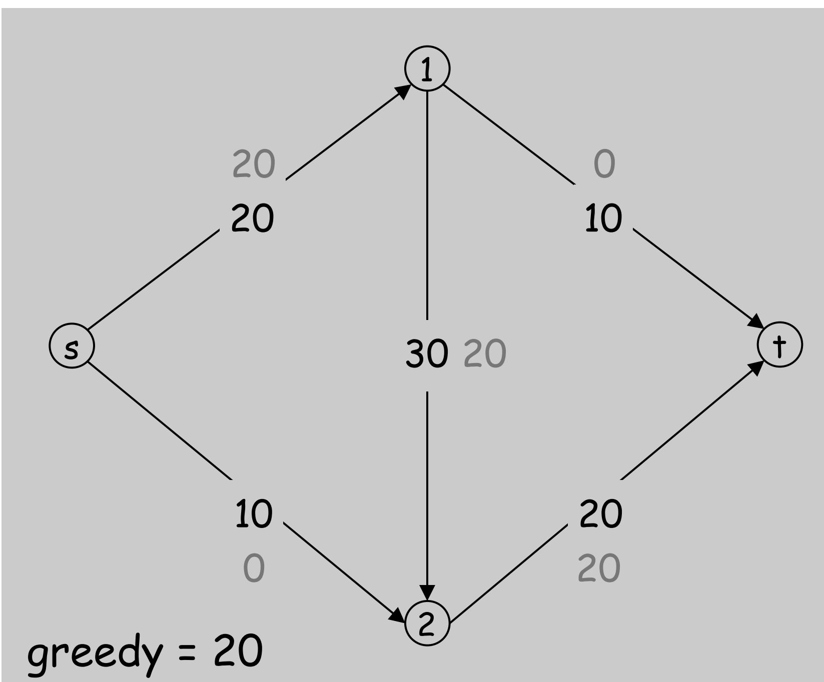
Flow value = 20

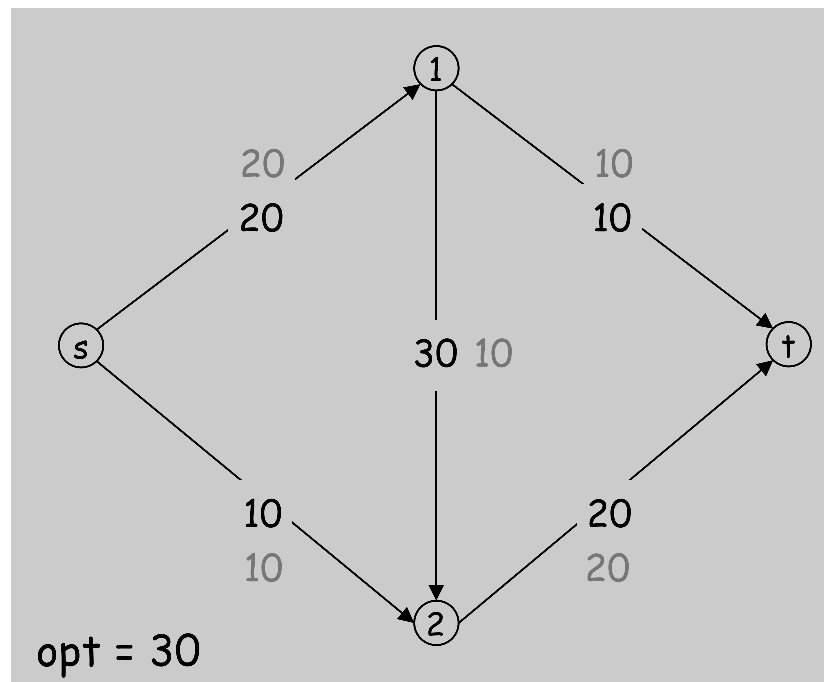
Towards a Max Flow Algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s-t path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

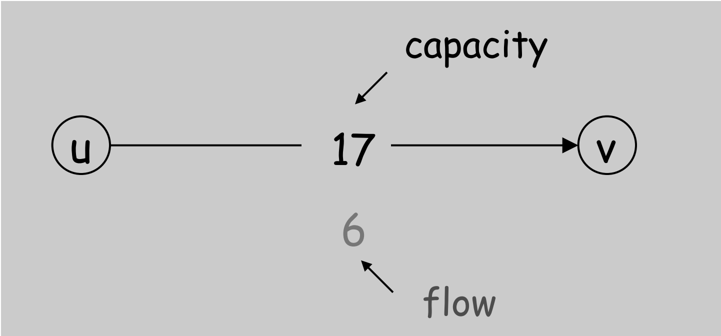
locally optimality $\nRightarrow$ global optimality





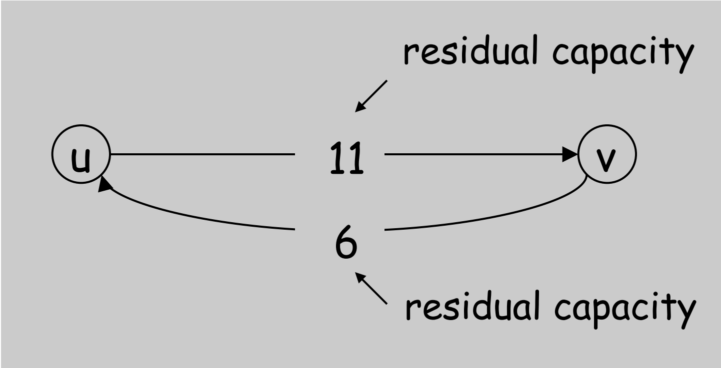
Residual Graph

Original edge: $e = (u, v) \in E$.
- Flow f(e), capacity c(e).



Residual edge.
- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity:

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$



Residual graph: $G_f = (V, E_f)$.
- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

Ford-Fulkerson Algorithm

Basic Algorithm:

- Identify an s-t path and augment the flow along that path
  - How to find a path? BFS or DFS
  - Bottleneck along path determines amount flow can be augmented
- Still need to include concepts of residual graph and forward/ backward edges

```
Ford-Fulkerson ( G=(V,E), c, s, t )
1. Initialize flow f to 0
2. While exists augmenting path p do
3.     Augment flow f along p
4. Return f
```

Ford-Fulkerson Algorithm

Complete Algorithm:

- Compute residual graph including both forward and backward edges

```
Ford-Fulkerson ( G=(V,E), c, s, t )

1. For every edge e let f(e)=0
2. Construct the residual graph G_f
3. While exists s-t path in G_f do
4.     Let p be an s-t path in G_f
5.     Let d=min_{e in p} c_f(e)
6.     For every e on p do
7.         If e is a forward edge then
8.             f(e)+=d
9.         else
10.            f(reverse(e))-=d
11.    Update G_f (construct new G_f)
12.Return f
```

Analysis of Ford-Fulkerson

Questions.

- Does it terminate?

Assumption. Flow capacities are all integer values.

Invariant. Every flow value f(e) and every residual capacities c_f (e) remains an integer throughout the algorithm. (Can be proven by induction over the iterations of the while loop)

Theorem. Let $C = \sum_{e \text{ out of } s} c(e)$

Then the algorithm terminates in at most v(f*) ≤ C iterations.

Pf. Each augmentation increase value by at least 1.

Analysis of Ford-Fulkerson

Questions.

- What is the running time?

How much work for each iteration?

- Finding an s-t path in the residual graph G_f
  - G_f has at most 2m edges
  - Use BFS/DFS: O(m+n) = O(m) since assume G is connected
- Augment flow along the path
  - O(n) to update each edge along the path
- Update the residual graph
  - O(m) to update each edge

- Total running time for Ford-Fulkerson is: O(mC)
  - NOTE – this is pseudo-polynomial (similar to Knapsack problem)
  - Running time grows exponentially with the length (number of bits) in the parameter C

Analysis of Ford-Fulkerson

Questions.

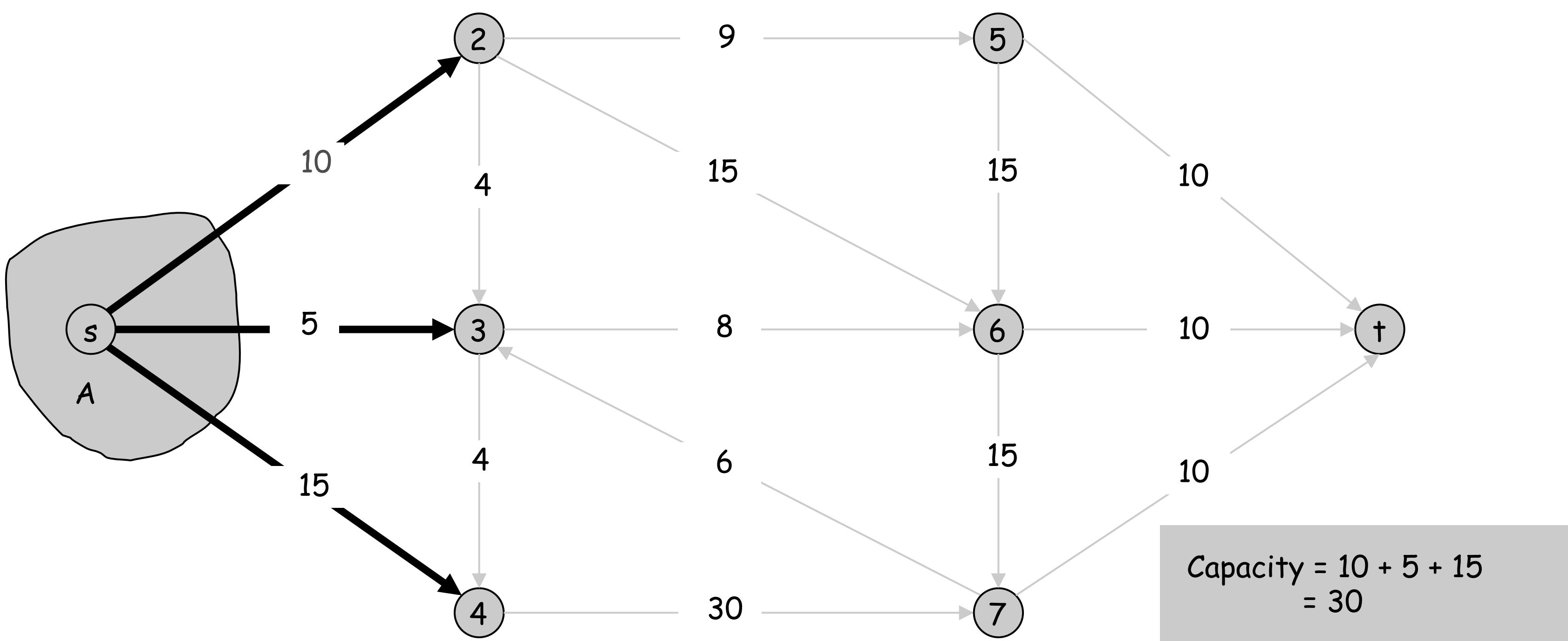
- Does it generate the maximum flow?

To see this we will introduce the concept of a graph cut

Cuts

Def. An s-t cut is a partition (A, B) of V with s ∈ A and t ∈ B.

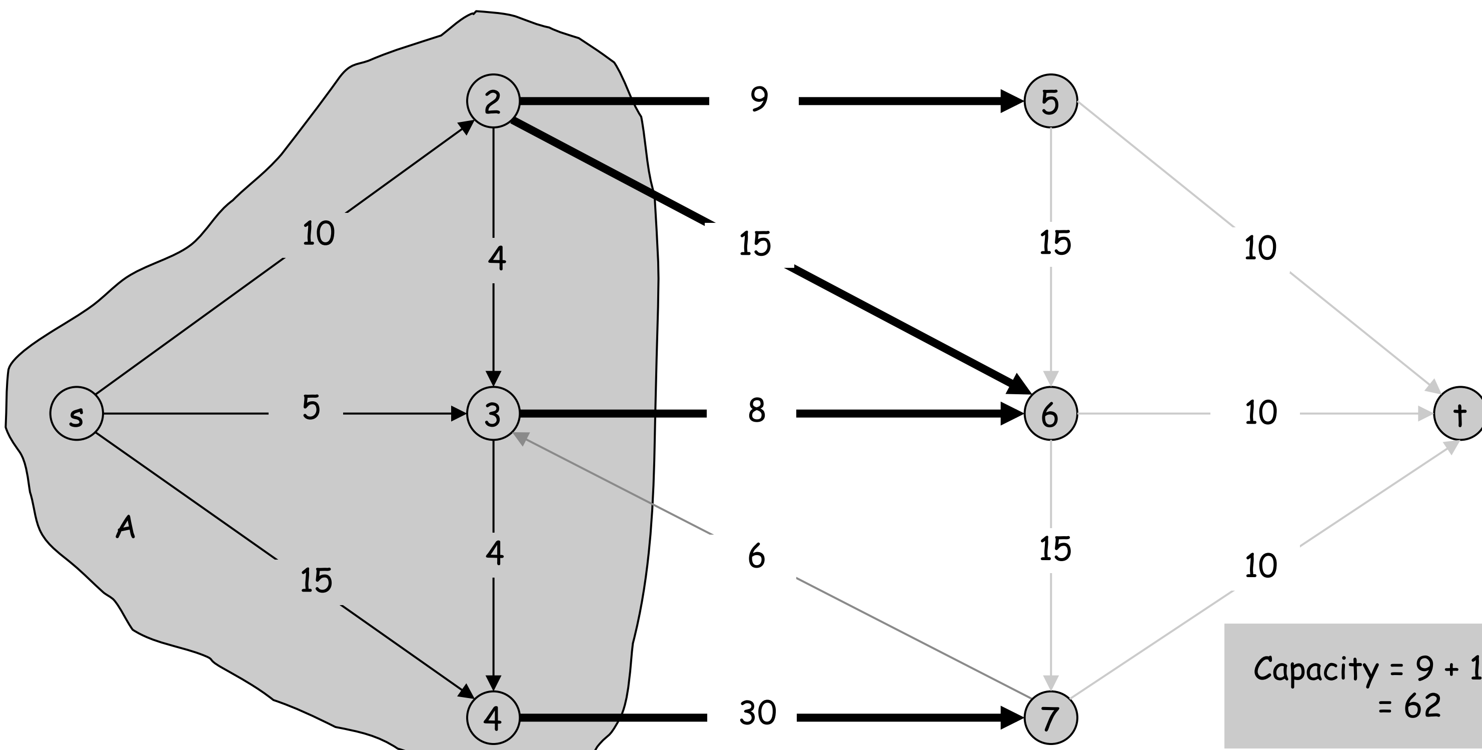
Def. The capacity of a cut (A, B) is: $$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



Cuts

Def. An s-t cut is a partition (A, B) of V with s ∈ A and t ∈ B.

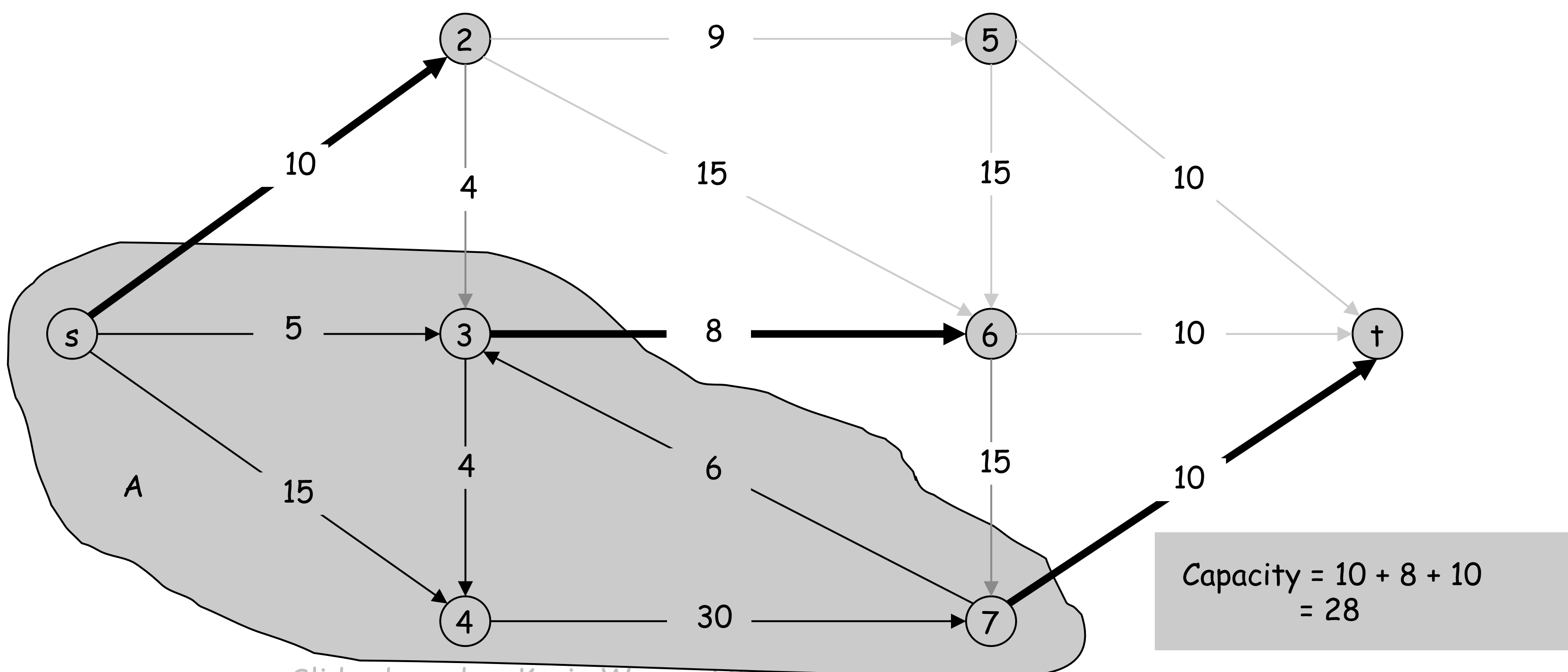
Def. The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



Capacity = 9 + 15 + 8 + 30
= 62

Minimum Cut Problem

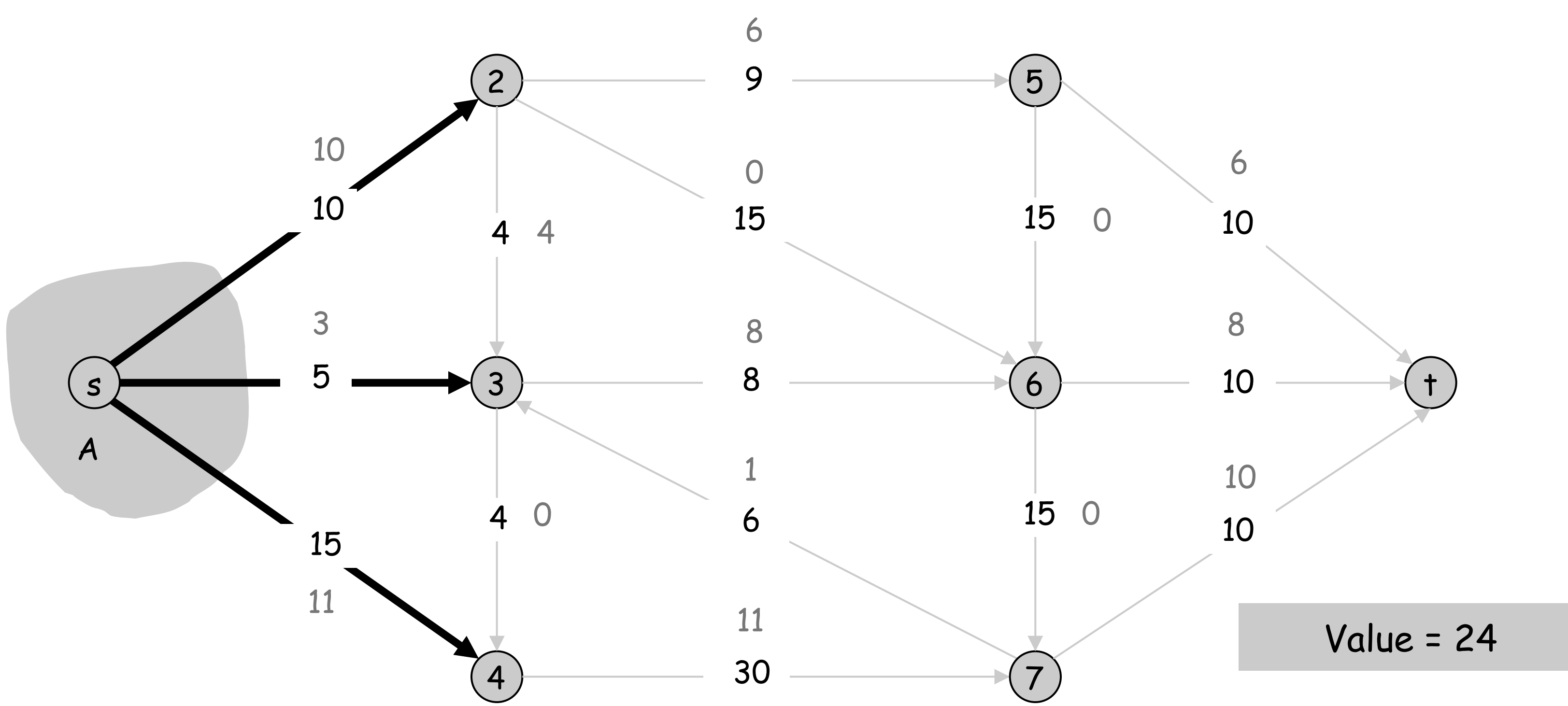
Min s-t cut problem. Find an s-t cut of minimum capacity.



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

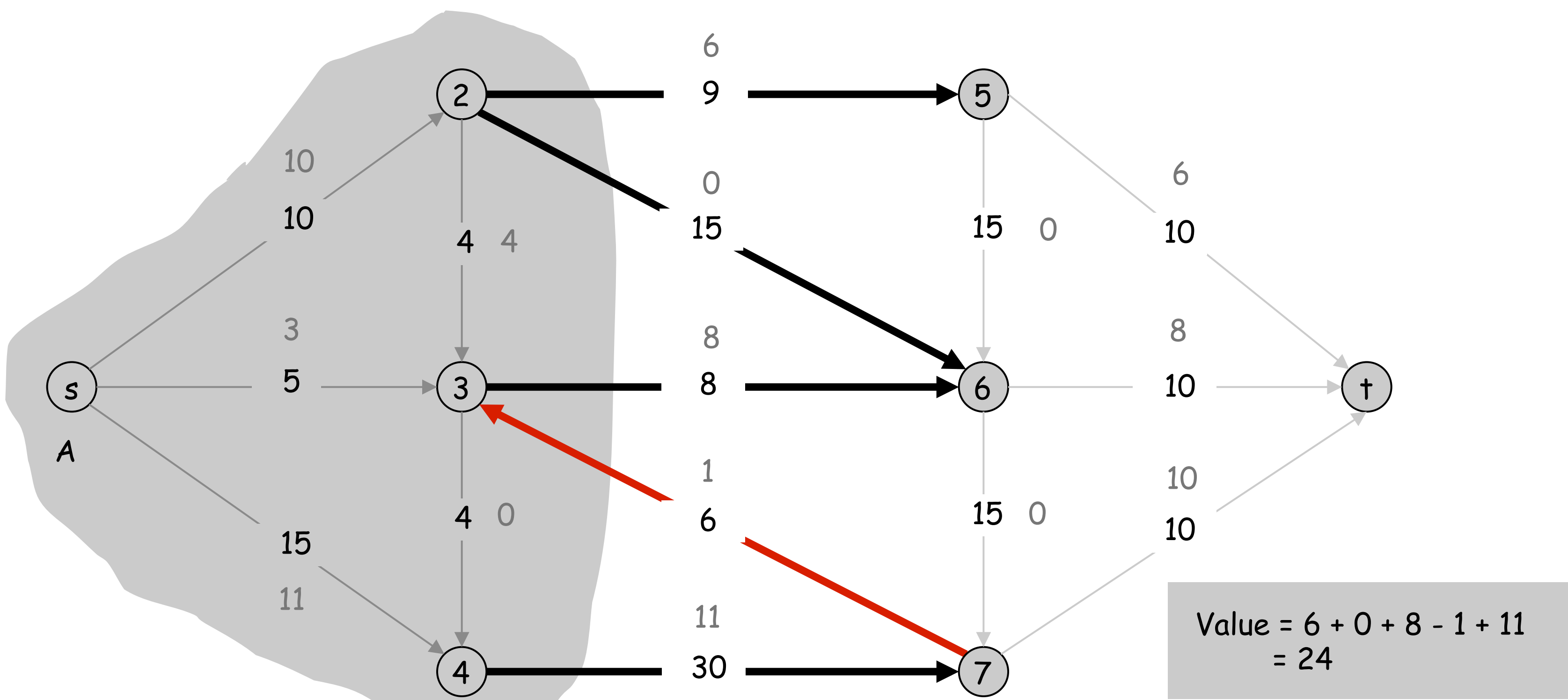
$$\sum_{e \text{ out of } A} f(e) \; - \; \sum_{e \text{ in to } A} f(e) \; = \; v(f)$$



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

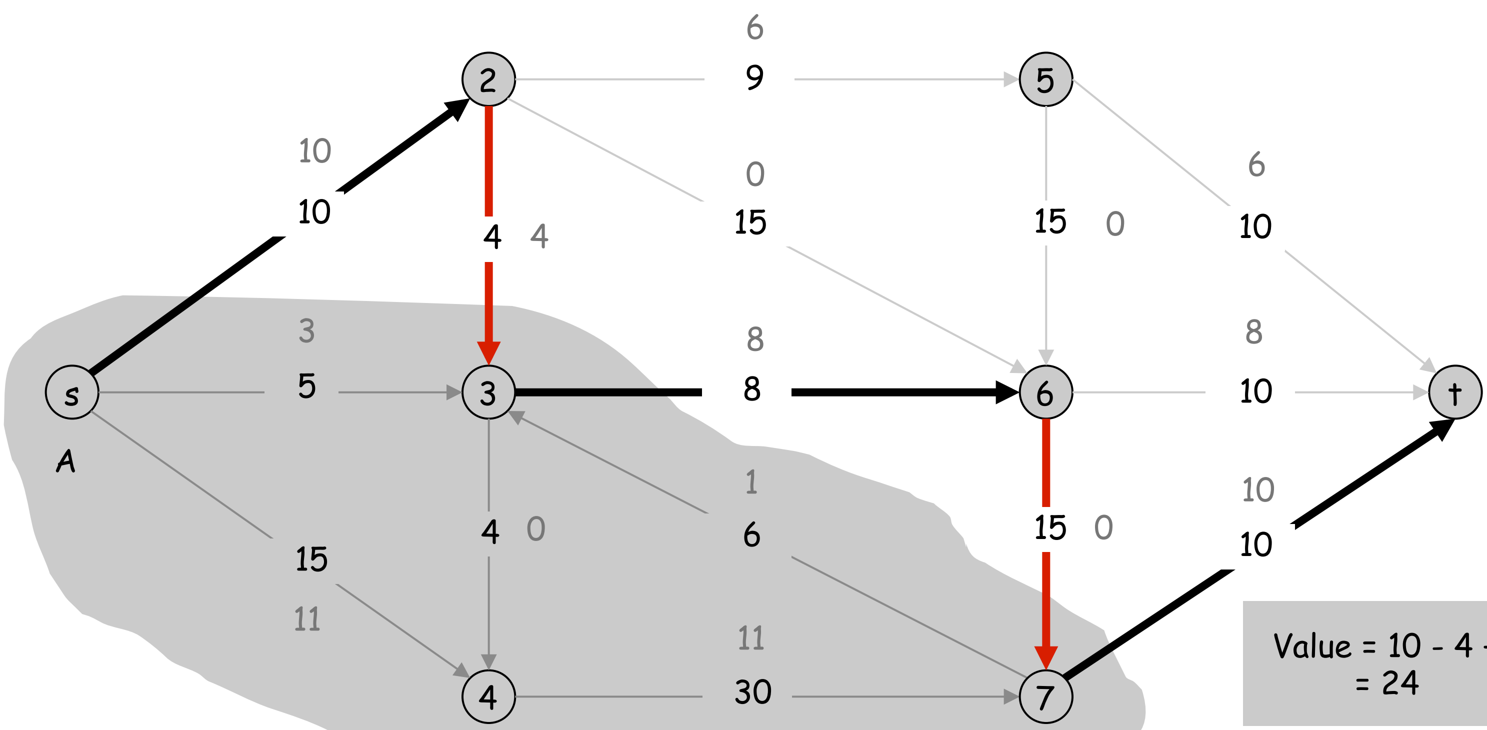
$$\sum_{e \text{ out of } A} f(e) \; - \; \sum_{e \text{ in to } A} f(e) \; = \; v(f)$$



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } A} f(e) \; - \; \sum_{e \text{ in to } A} f(e) \; = \; v(f)$$



Value = 10 - 4 + 8 - 0 + 10
= 24

Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f).$$

Pf.

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

by flow conservation, all terms except v = s are 0 $\longrightarrow$

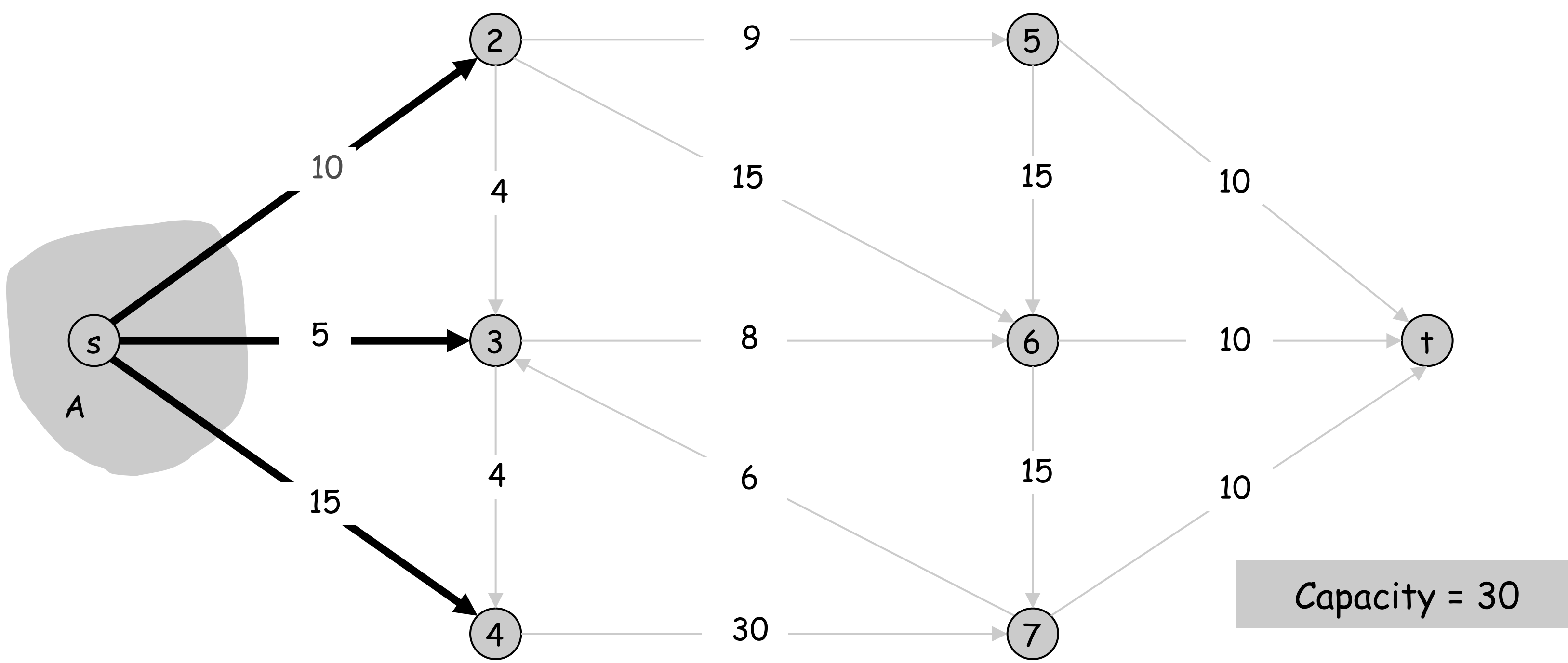
$$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)$$

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e).$$

Flows and Cuts

Weak duality. Let f be any flow, and let (A, B) be any s-t cut. Then the value of the flow is at most the capacity of the cut.

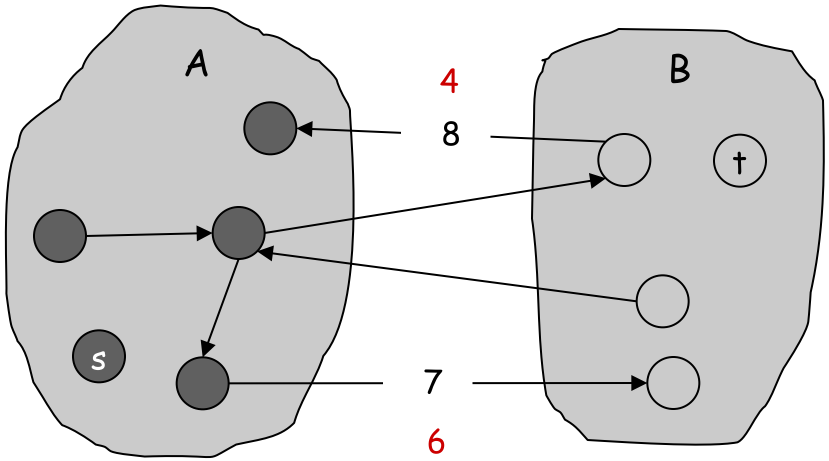
Cut capacity = 30 $\Rightarrow$ Flow value $\leq$ 30



Weak duality. Let f be any flow. Then, for any s-t cut (A, B) we have $v(f) \le cap(A, B)$.

Pf.

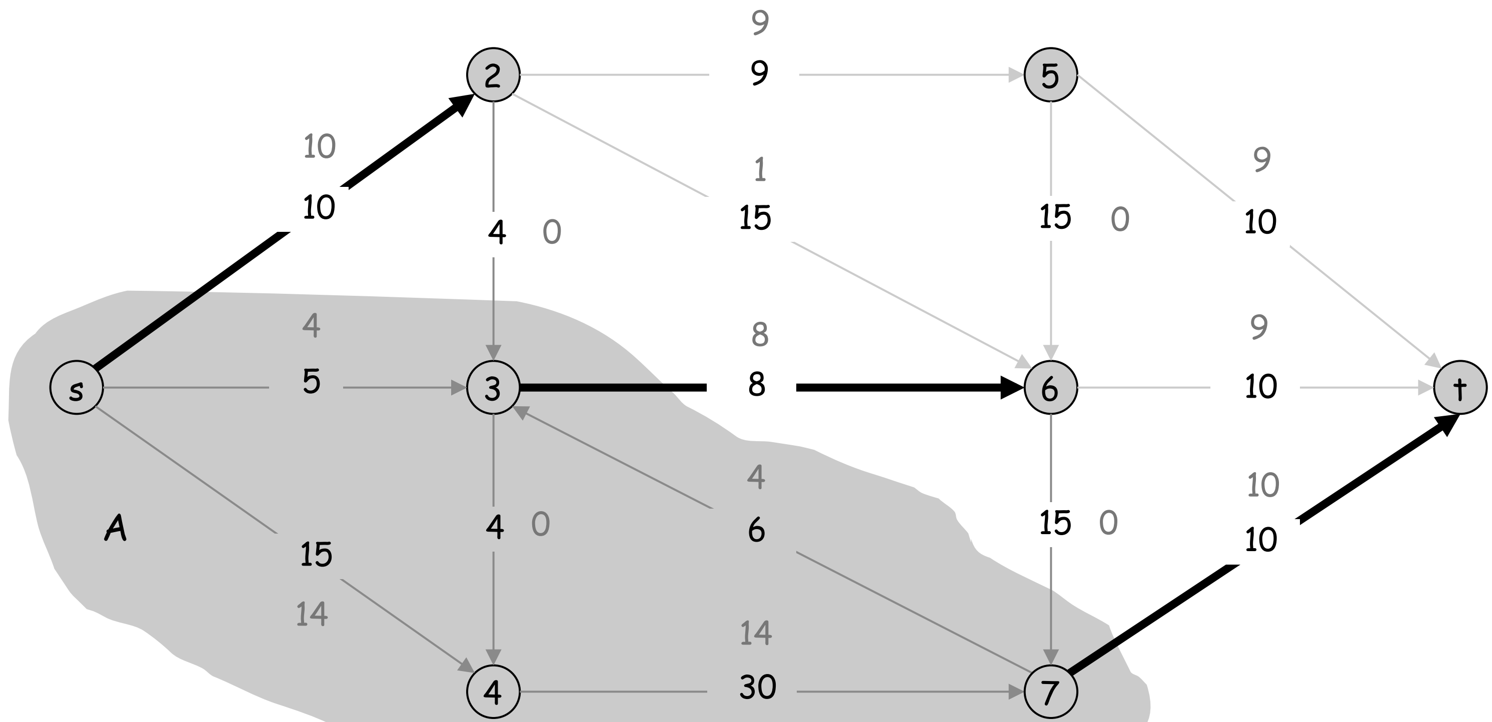
$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
&\le \sum_{e \text{ out of } A} f(e) \\
&\le \sum_{e \text{ out of } A} c(e) \\
&= \operatorname{cap}(A, B) \qquad \blacksquare
\end{aligned}
$$



Certificate of Optimality

Corollary. Let f be any flow, and let (A, B) be any cut.
If v(f) = cap(A, B), then f is a max flow and (A, B) is a min cut.

Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28



Max-Flow Min-Cut Theorem

Max-flow min-cut theorem. [Ford-Fulkerson 1956] The value of the max flow is equal to the value of the min cut.

The Ford-Fulkerson Algorithm generates a max flow

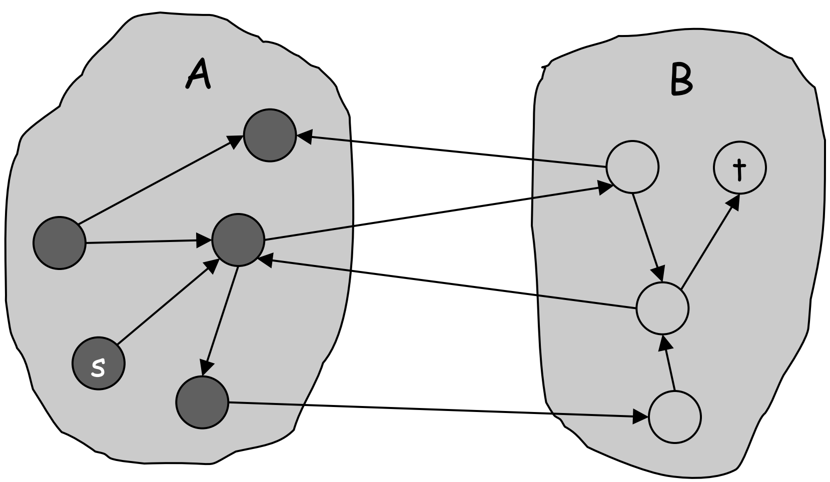
To prove this we exhibit an s-t cut (A,B) with capacity equal to the flow generated by the algorithm

Proof of Max-Flow Min-Cut Theorem

Proof that Ford-Fulkerson generates a maximum flow

- Let f be a flow with no augmenting paths.
- Let A be set of vertices reachable from s in residual graph.
- By definition of A, s $\in$ A.
- By definition of f, t $\notin$ A.

$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e) \\ &= \sum_{e \text{ out of } A} c(e) \\ &= cap(A,B) \quad \blacksquare \end{aligned}$$



original network

All outgoing edges from A must be used at full capacity

All incoming edges to A must be unused

Choosing Good Augmenting Paths

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

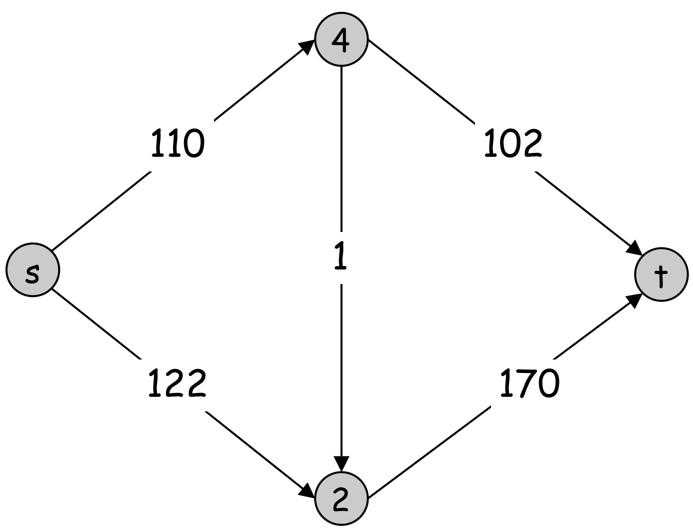
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

- Max bottleneck capacity. (could be expensive to determine)
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

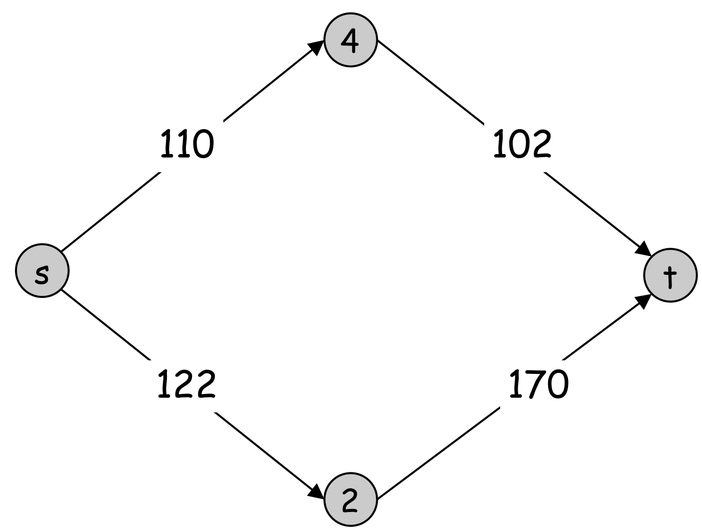
Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only edges with capacity at least Δ.



G_f

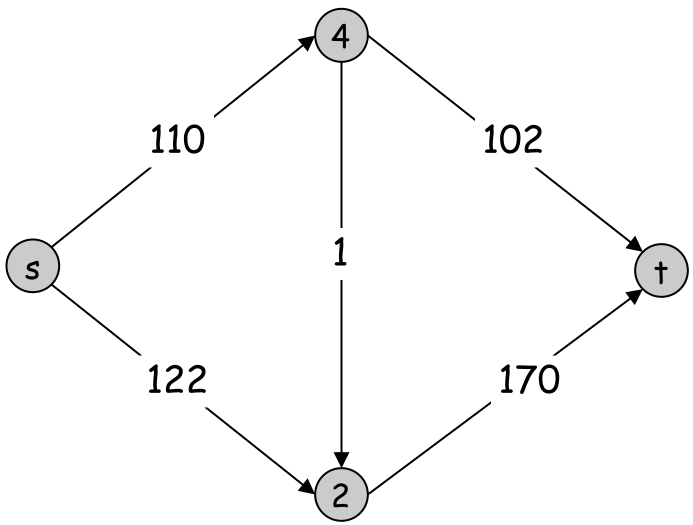


G_f (100)

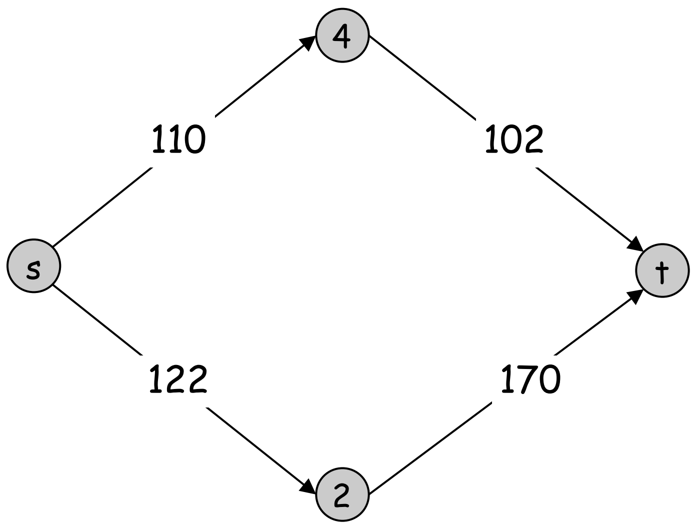
Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

- Start with Δ = largest power of 2 that is smaller than C, the sum of the capacities of all edges out of s.
- Discover all s-t paths in G_f (Δ).
- Divide Δ by 2 and repeat, until finding all paths with Δ = 1.



G_f



G_f (100)

Capacity Scaling: Correctness and Running Time

Correctness

- *Assuming integer constraints for flow capacities*
- When $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths.

Running Time

- The scaling max-flow algorithm finds a max flow in $O(m \log C)$ augmentations. It can be implemented to run in $O(m^2 \log C)$ time.
  - (reducing the dependency on flow capacity to log C)

Edmonds-Karp Algorithm

General Idea: For each iteration, the shortest augmentation path is chosen.

Argument: Each time an edge is a bottleneck edge on an augmentation path, it is temporarily removed from the residual graph. By the time it returns to the residual graph and is again a critical edge, it must be part of a shortest path that is at least length 2 longer than previously.

Running Time: Each edge can be a critical edge at most $n/2$ times, since all shortest paths will be length $n-1$ or less. Every augmentation must involve at least one critical edge. Thus the total number of augmentations is $O(nm)$, and the total running time of Edmonds-Karp is $O(nm^2)$.

Dependency on C has been completely removed.

Maximum Flow and Minimum Cut

Max flow and min cut.
- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

Nontrivial applications / reductions. (See the Kleinberg / Tardos textbook)
- Data mining.
- Open-pit mining.
- Project selection.
- Airline scheduling.
- Bipartite matching.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Network reliability.
- Distributed computing.
- Egalitarian stable matching.
- Security of statistical data.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Many many more . . .