南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis (H)

CS216

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

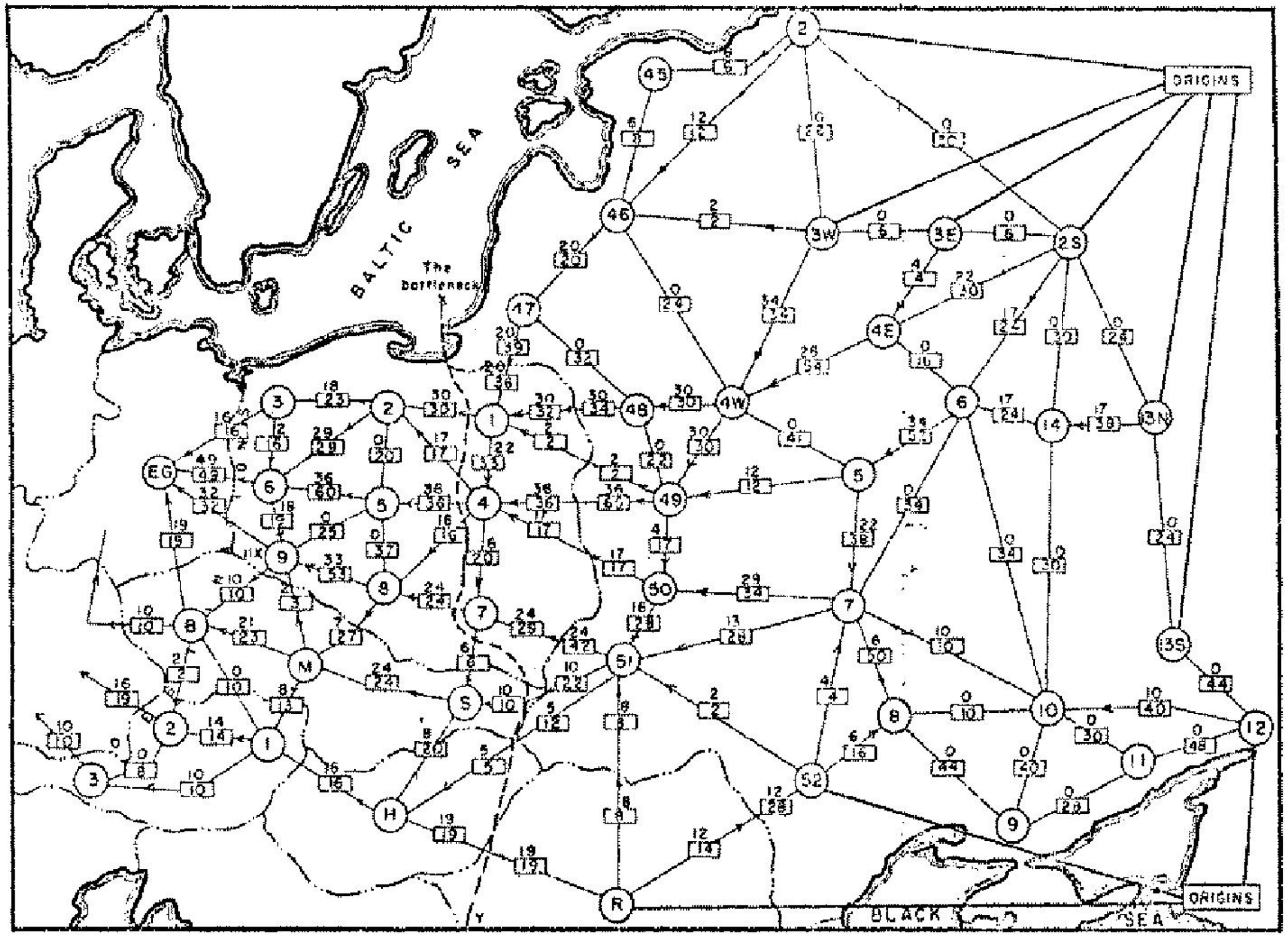
http://faculty.sustech.edu.cn/yusq/

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

1. Network Flow

PEARSON Addison Wesley

Soviet Rail Network, 1955



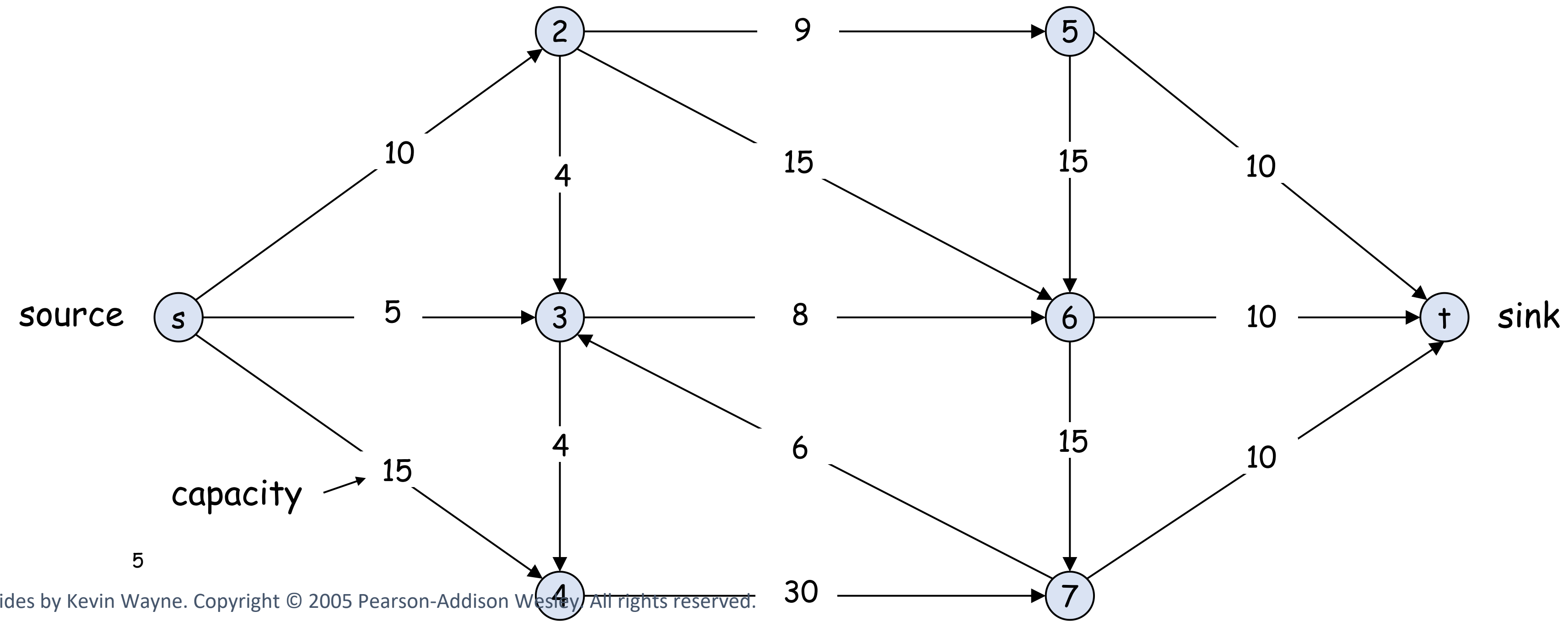
Reference: *On the history of the transportation and maximum flow problems.*
Alexander Schrijver in Math Programming, 91: 3, 2002.

Maximum Flow and Minimum Cut

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

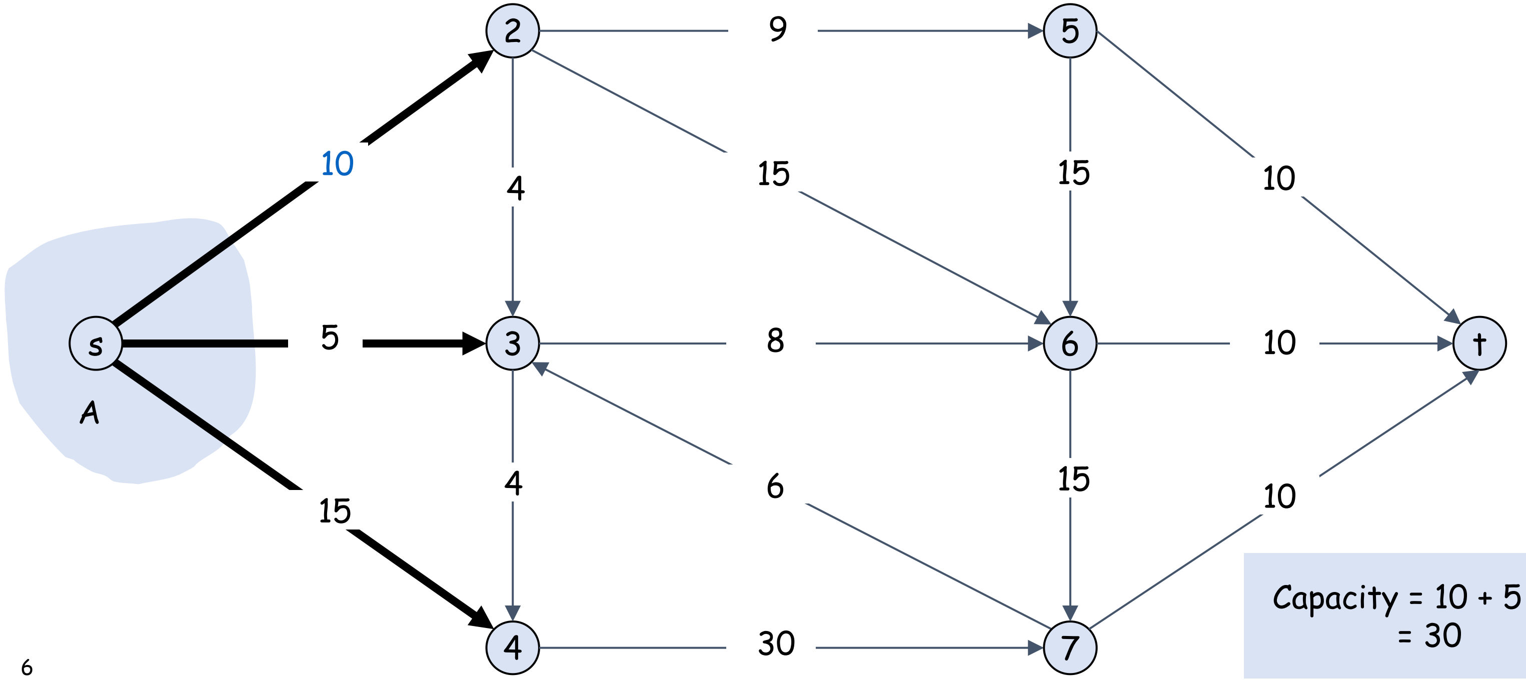
Minimum Cut Problem

- Flow network.
  - Abstraction for material flowing through the edges.
  - G = (V, E) = directed graph, no parallel edges.
  - Two distinguished nodes: s = source, t = sink.
  - c(e) = capacity of edge e.



Cuts

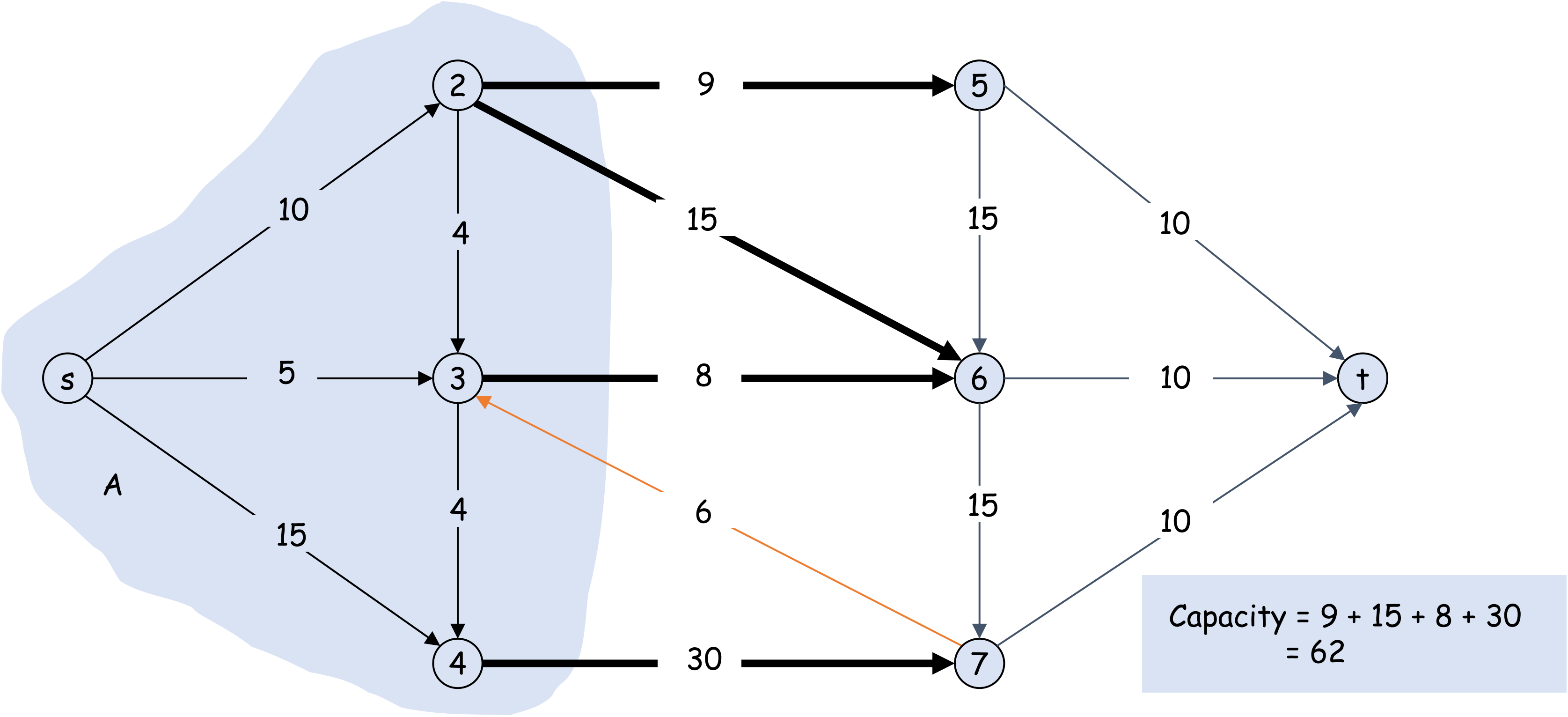
- Def. An s-t cut is a partition (A, B) of V with s $\in$ A and t $\in$ B.
- Def. The capacity of a cut (A, B) is $c(A, B) = \sum_{e \text{ out of } A} c_e$



Capacity = 10 + 5 + 15
= 30

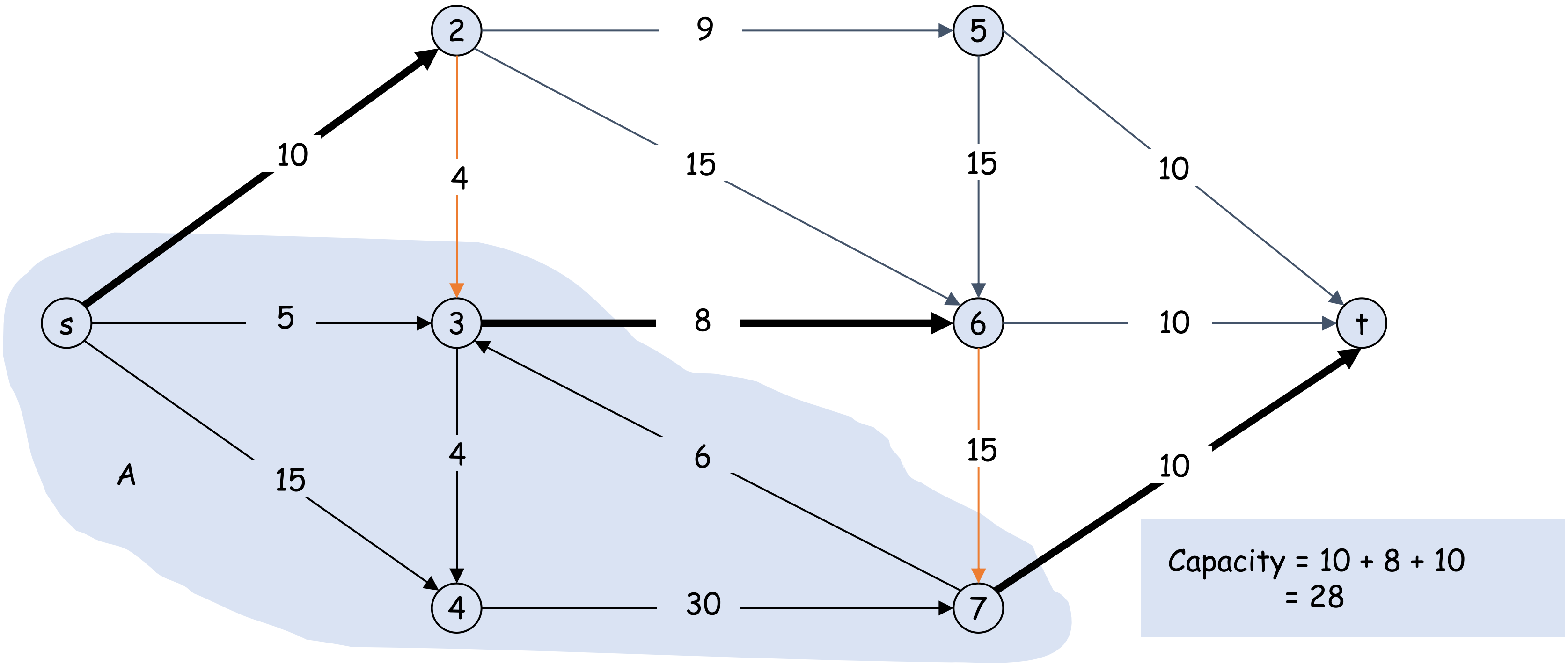
Cuts

- 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 $c(A,B) = \sum_{e \text{ out of } A} c_e$



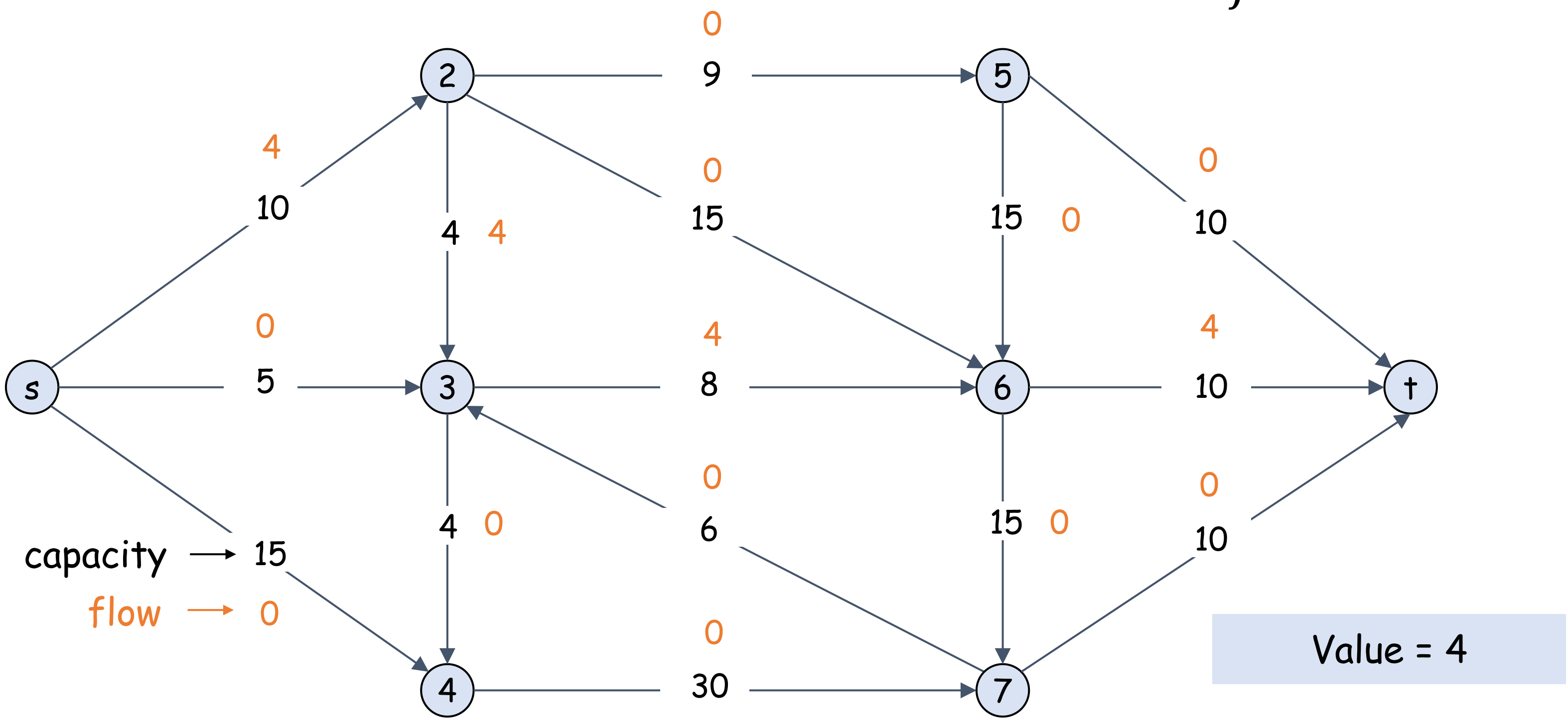
Minimum Cut Problem

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



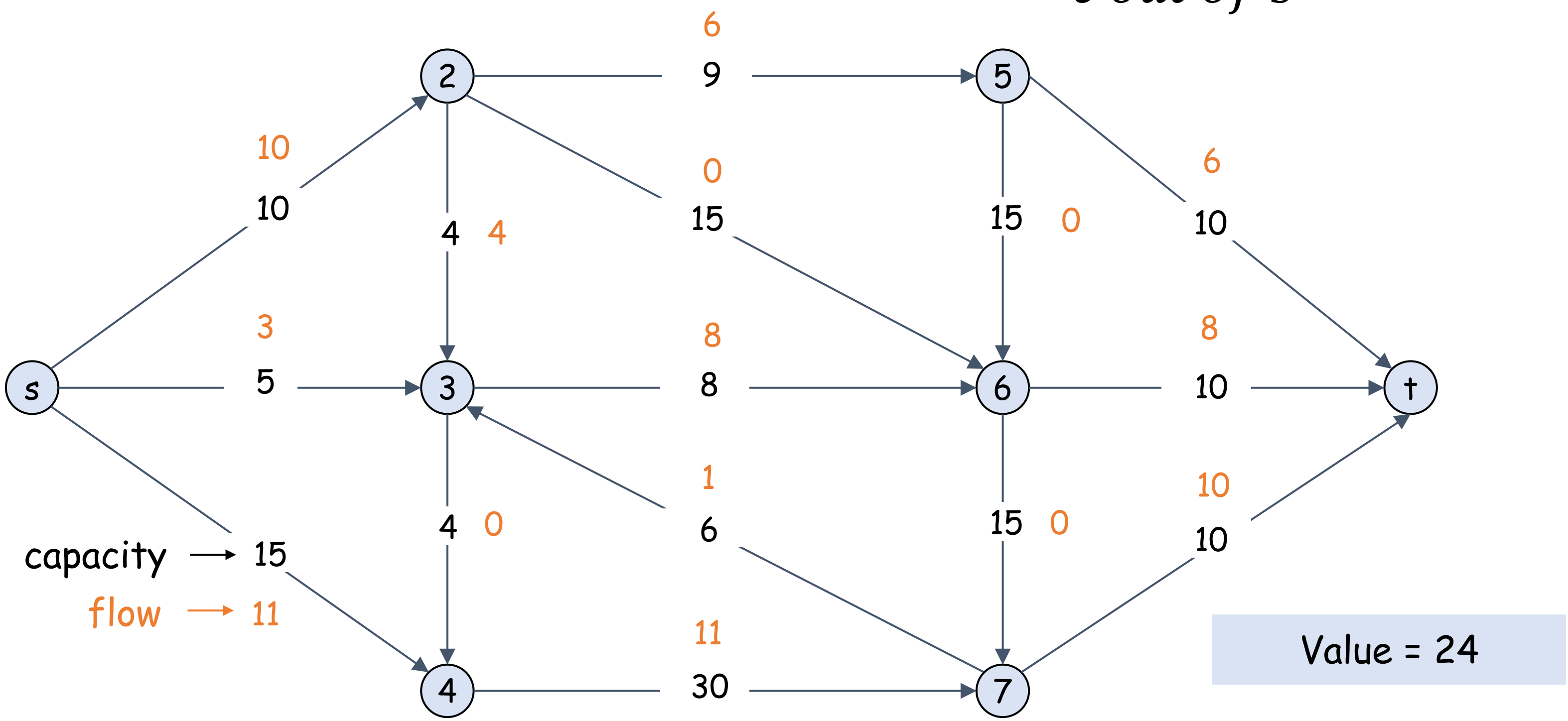
Flows

- Def. An s-t flow is a function that satisfies
  - For each $e \in E$: $0 \leq f(e) \leq c_e$ [capacity]
  - For each $v \in V - \{s, t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]
- Def. The value of a flow f is $v(f) = \sum_{e \text{ out of } s} f(e)$



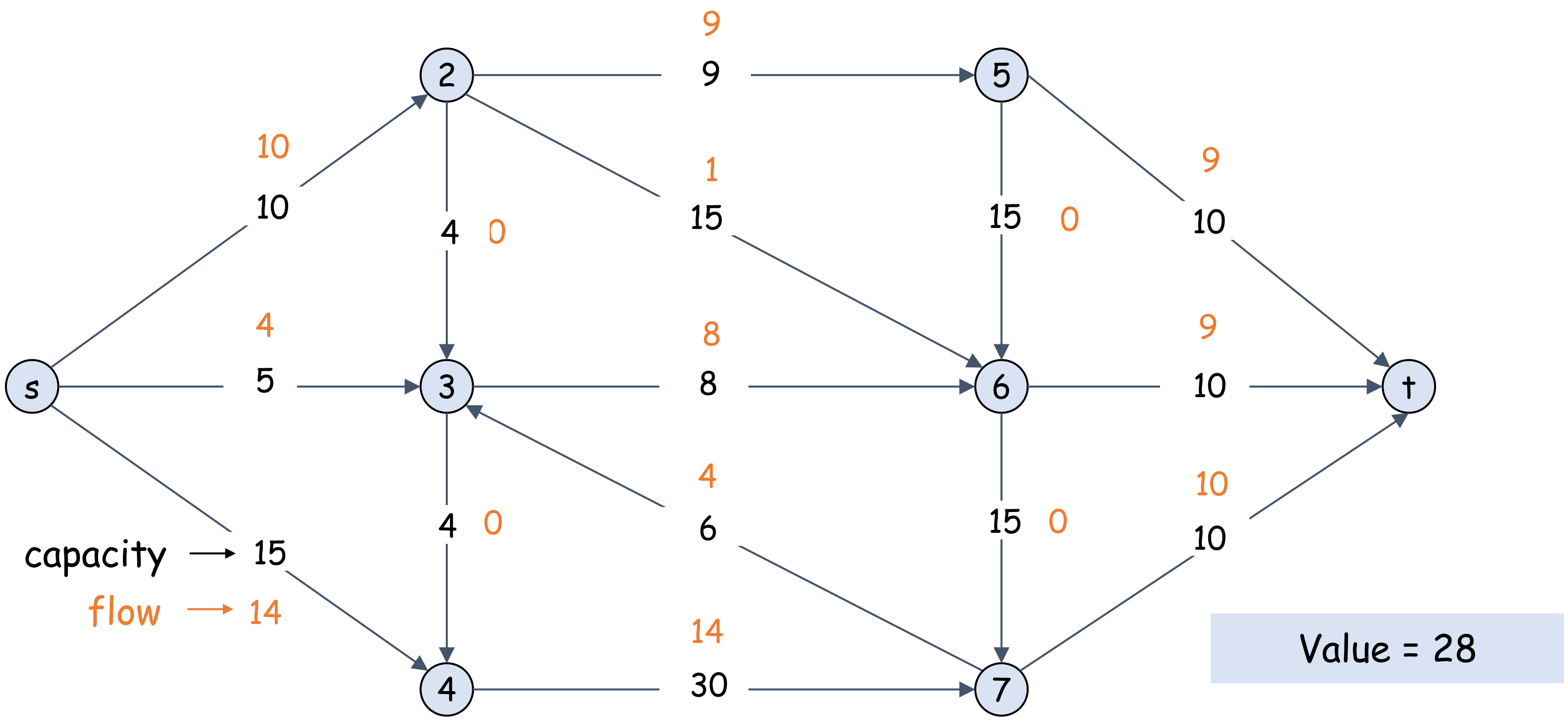
Flows

- Def. An s-t flow is a function that satisfies
  - For each $e \in E$: $0 \leq f(e) \leq c_e$ [capacity]
  - For each $v \in V - \{s, t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]
- Def. The value of a flow f is $v(f) = \sum_{e \text{ out of } s} f(e)$



Maximum Flow Problem

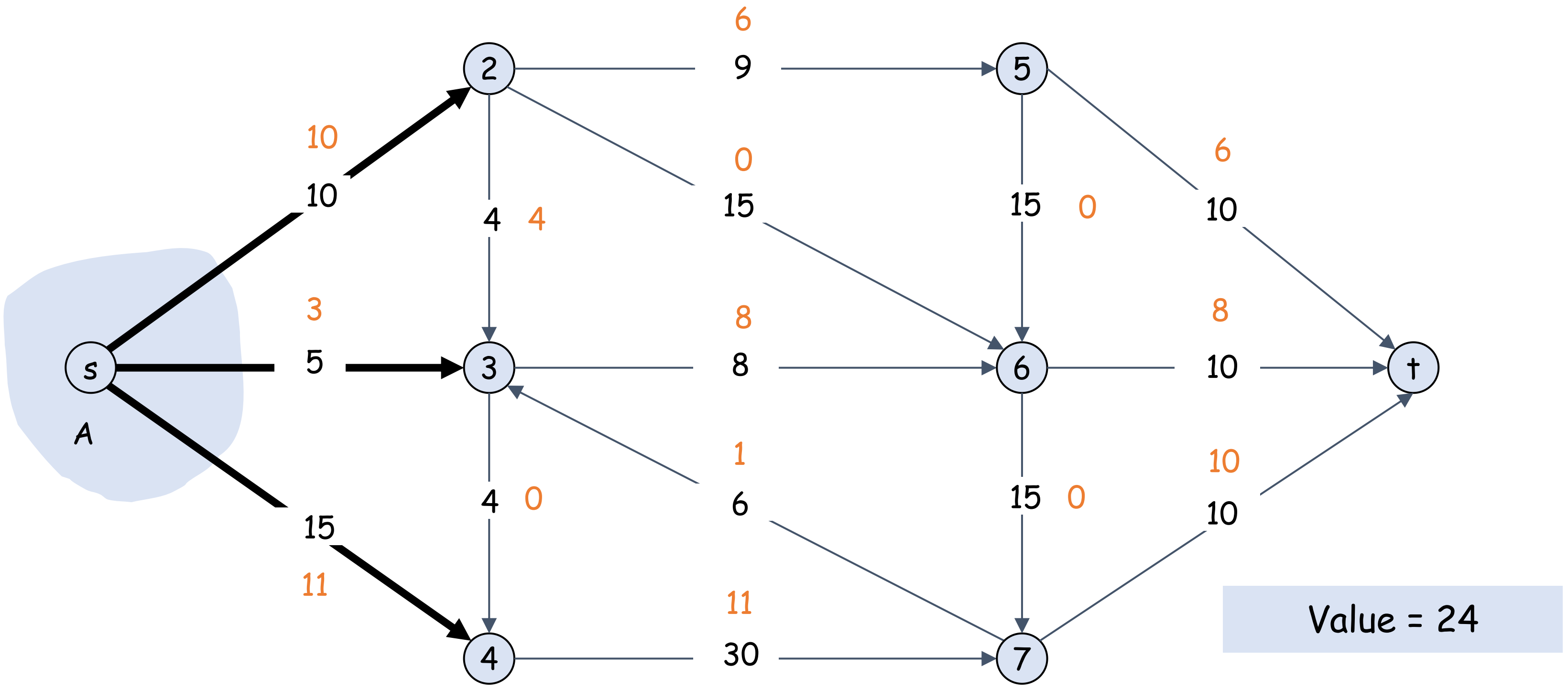
- Max flow problem. Find s-t flow of maximum value.



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.

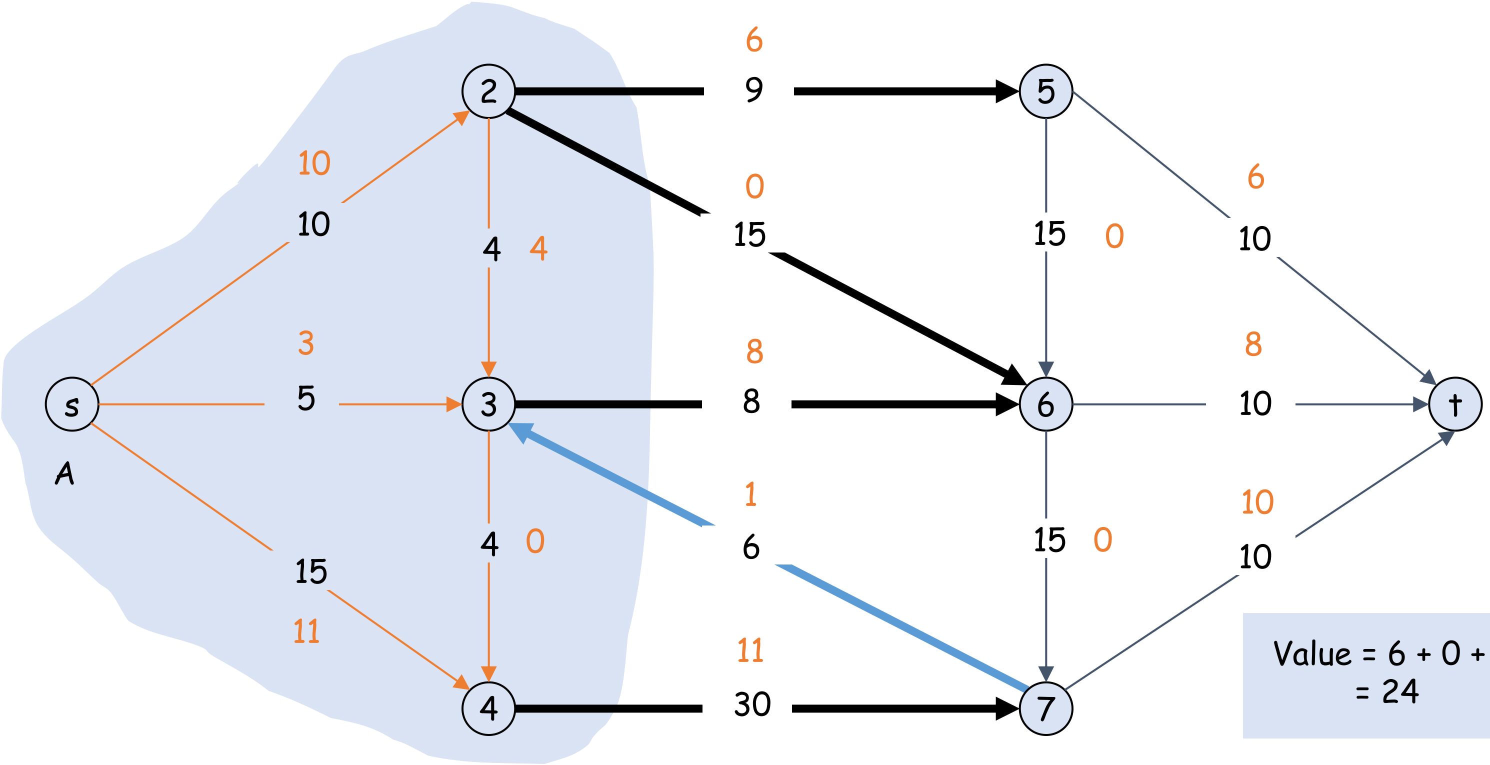
$$v(f) = f^{out}(A) - f^{in}(A)$$



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.

$$v(f) = f^{out}(A) - f^{in}(A)$$

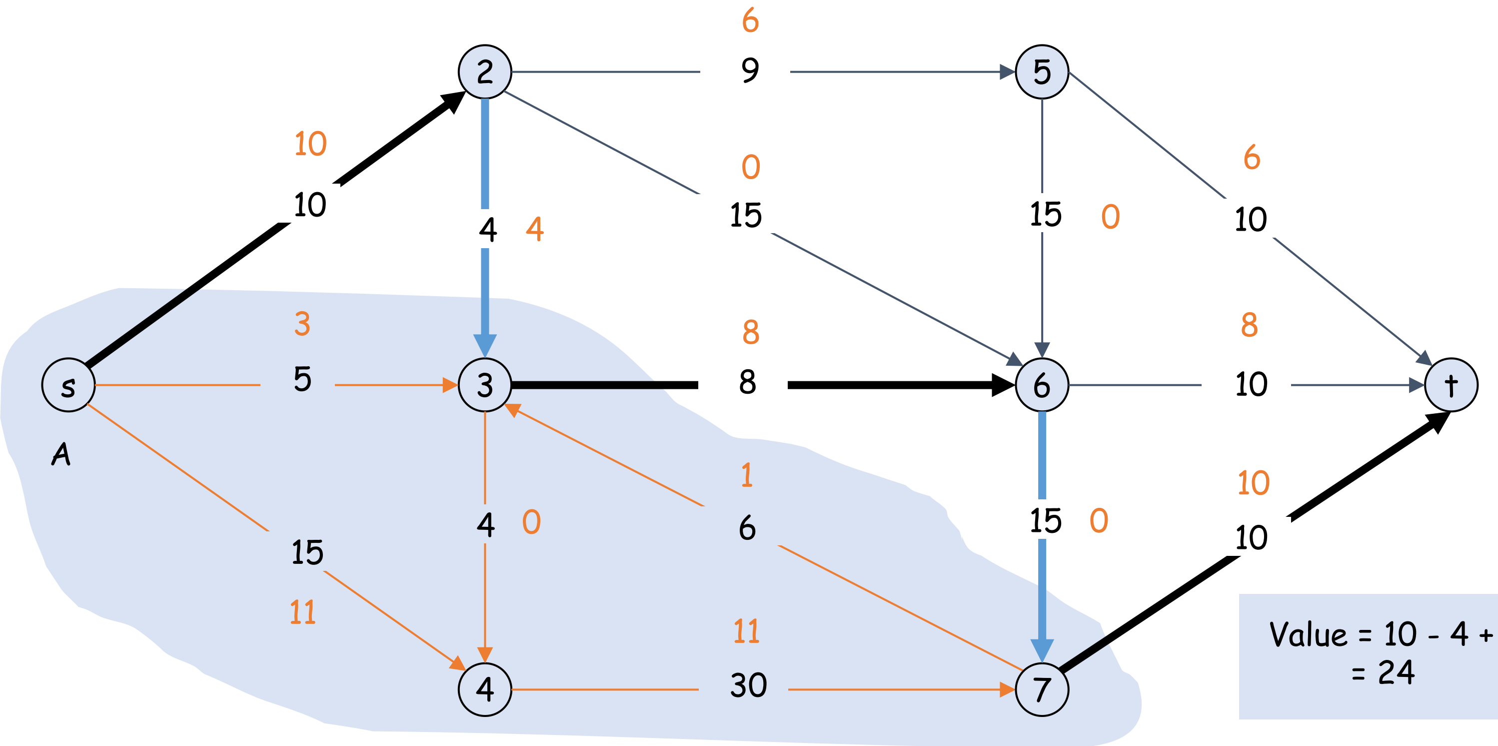


Value = 6 + 0 + 8 - 1 + 11
= 24

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.

$$v(f) = f^{out}(A) - f^{in}(A)$$



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, the net flow sent across the cut is equal to the amount leaving s.

$$v(f) = f^{out}(A) - f^{in}(A)$$

- **Pf.**

$f^{in}(s)$=0

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

$f^{out}(v) - f^{in}(v)$=0

$$= \sum_{v \in A} (f^{out}(v) - f^{in}(v))$$

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

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.

$$v(f) = f^{out}(A) - f^{in}(A)$$

$$v(f) = f^{in}(B) - f^{out}(B)$$

- Pf.

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

$$= \sum_{v \in A} (f^{out}(v) - f^{in}(v))$$

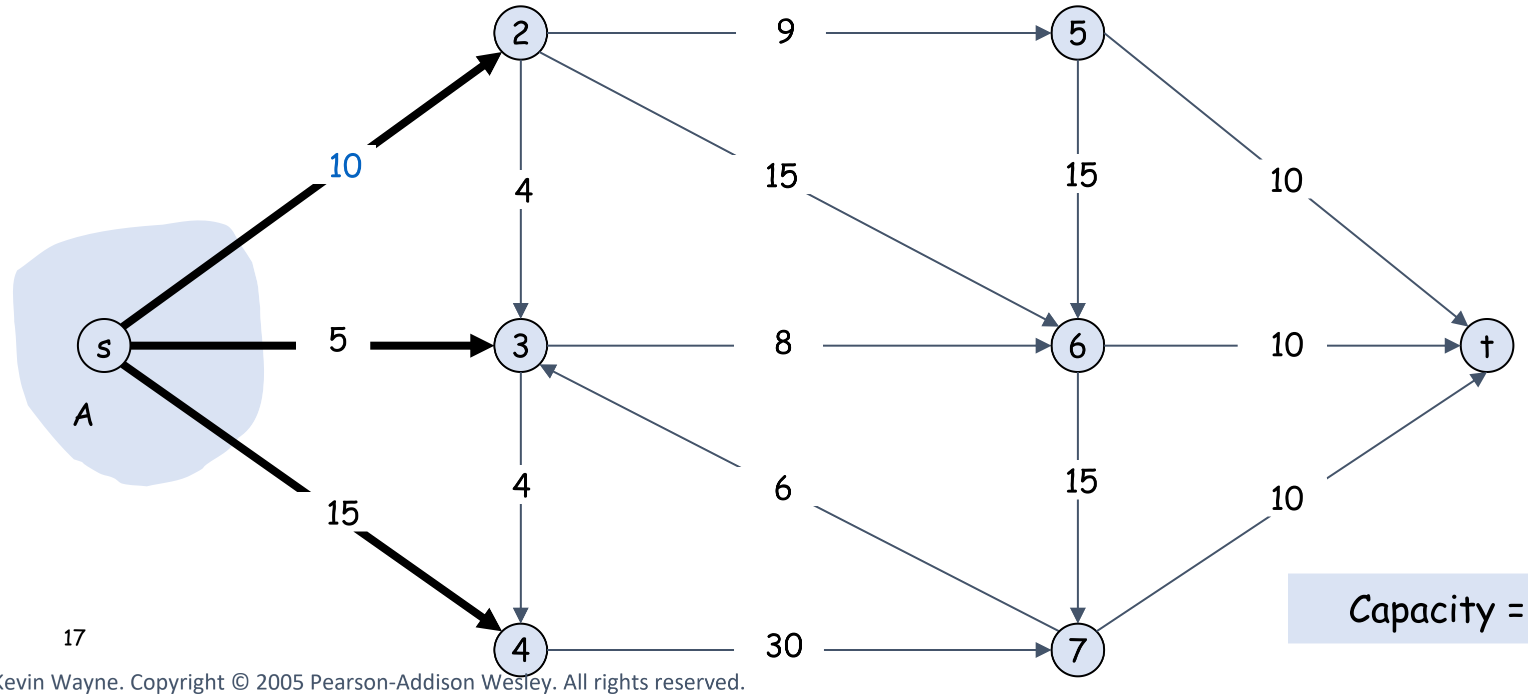
$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = f^{out}(A) - f^{in}(A)$$

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:

$$c(A,B) = \sum_{e \text{ out of } A} c_e$$

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

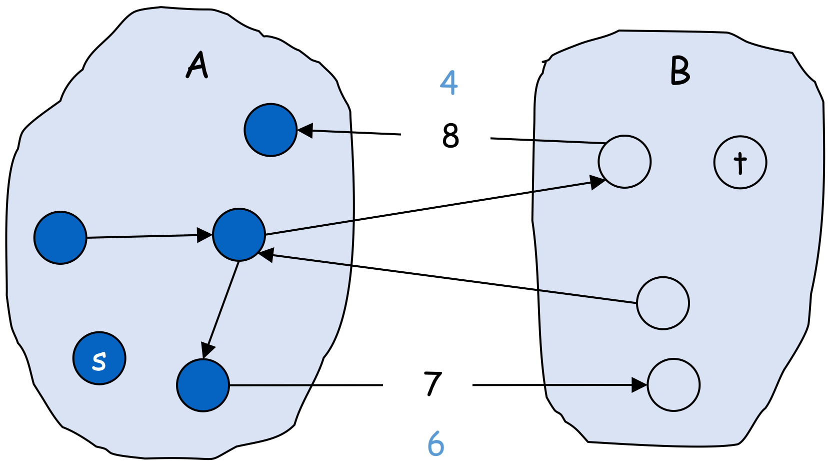


Capacity = 30

Flows and Cuts

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

- Pf.
$$
\begin{aligned}
v(f) &= f^{\text{out}}(A) - f^{\text{in}}(A) \\
&\le f^{\text{out}}(A) \\
&= \sum_{e \text{ out of } A} f(e) \\
&\le \sum_{e \text{ out of } A} c_e \\
&= c(A, B).
\end{aligned}
$$



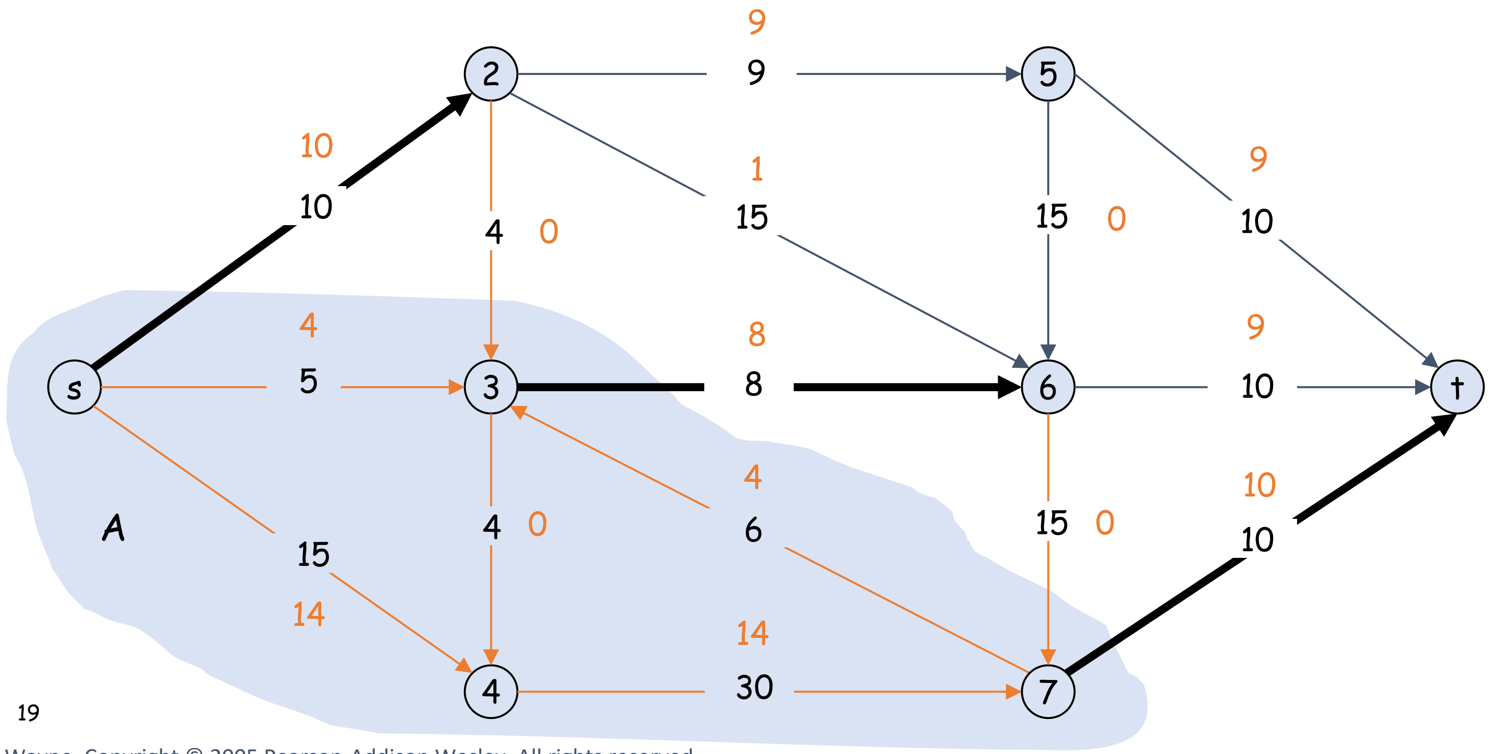
- $\blacksquare$

Certificate of Optimality

- Corollary. Let f be any flow, and let (A, B) be any cut.
  If v(f) = c(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

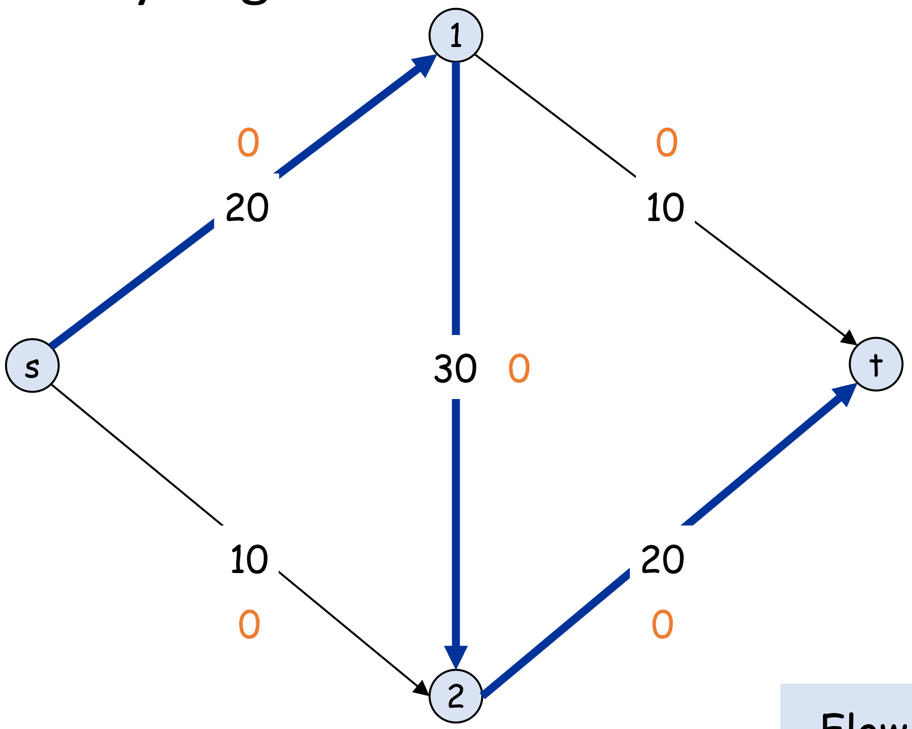
every flow is upper bounded by every cut!



PEARSON Addison Wesley

Towards a Max Flow Algorithm

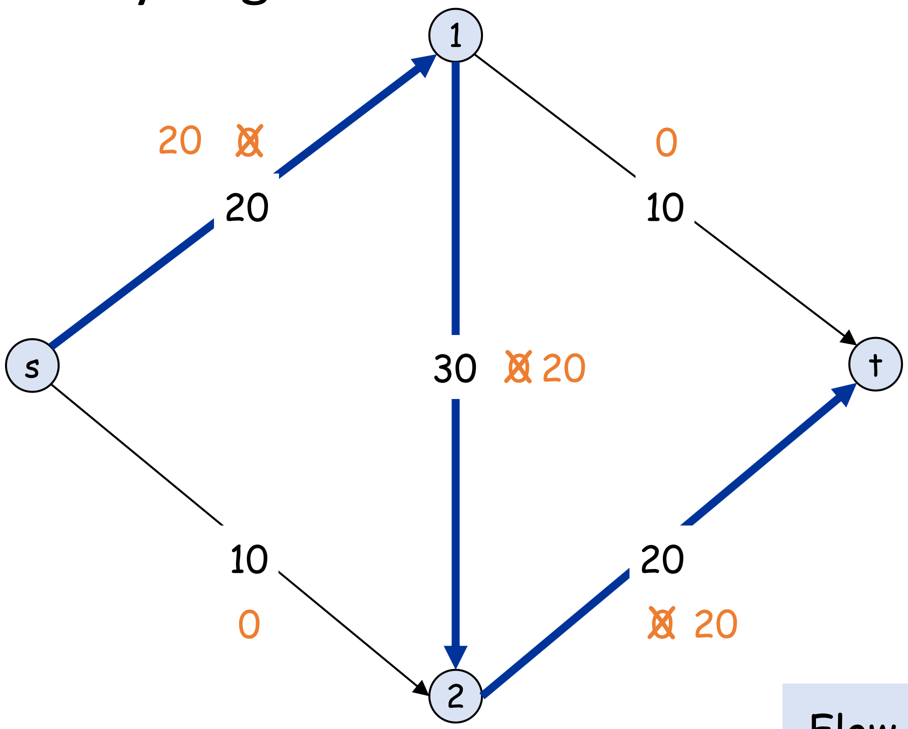
- Greedy algorithm.
  - Start with $f(e) = 0$ for all edge $e \in E$.
  - Find an s-t path P where each edge has $f(e) < c(e)$.
  - Augment flow along path P.
  - Repeat until you get stuck.



Flow value = 0

Towards a Max Flow Algorithm

- Greedy algorithm.
  - Start with f(e) = 0 for all edge e ∈ 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.

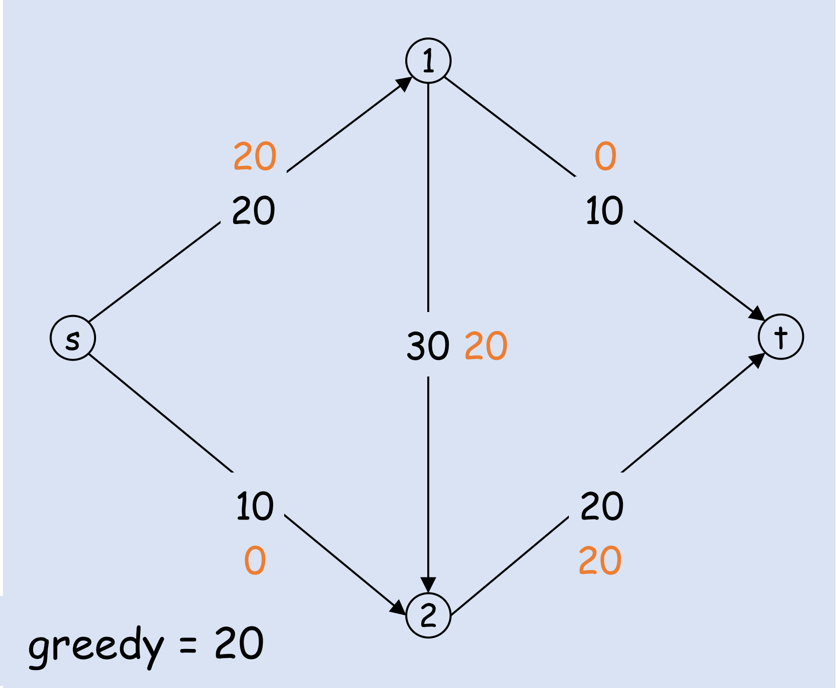


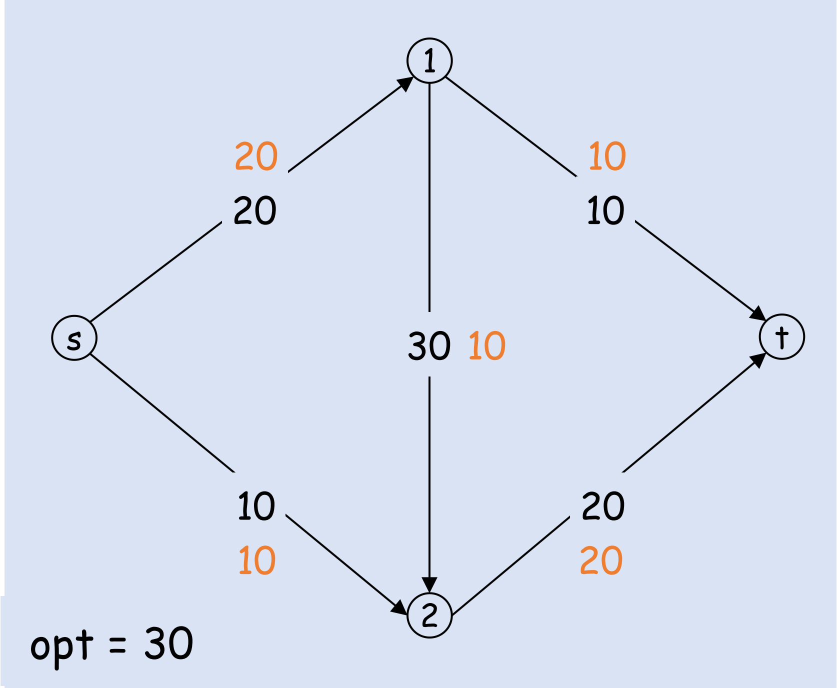
Flow value = 20

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.

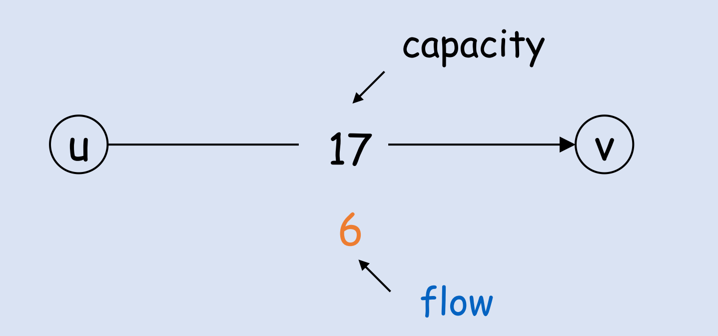
locally optimality $\not\Rightarrow$ global optimality



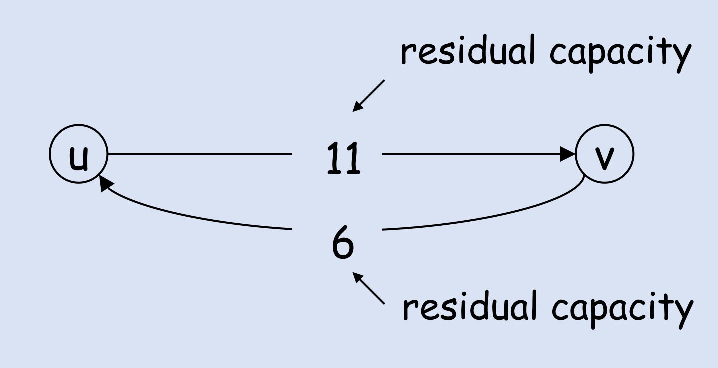


Residual Graph

- 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:
    - ✓ Forward edge: $c(e)-f(e)$
    - ✓ Backward edge: $f(e)$



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

Augmenting Path Algorithm

```
Augment(f, c, P) {
   b ← bottleneck(P)
   foreach e ∈ P {
      if (e ∈ E) f(e) ← f(e) + b
      else       f(eR)← f(eR) - b
   }
   return f
}
```

forward edge

reverse edge

P: simple s-t path in G^f

bottleneck(P): min residual capacity of any edge on P

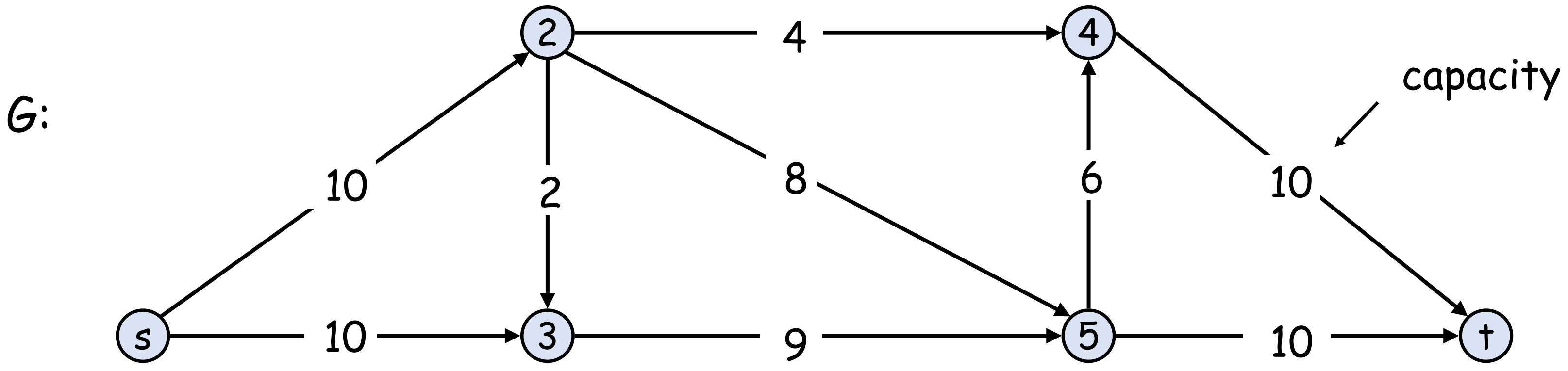
Augmenting path: any s-t path in the residual graph

Augmenting Path Algorithm

```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

   while (there exists augmenting path P) {
      f ← Augment(f, c, P)
      update Gf
   }
   return f
}
```

Ford-Fulkerson Algorithm



Max-Flow Min-Cut Theorem

- Augmenting path theorem. Flow f is a max flow iff there are no augmenting paths.

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

- Pf. We prove both simultaneously by showing TFAE:
  (i) There exists a cut (A, B) such that v(f) = c(A, B).
  (ii) Flow f is a max flow.
  (iii) There is no augmenting path relative to f.

- (i) $\Rightarrow$ (ii) This was the corollary to weak duality lemma.

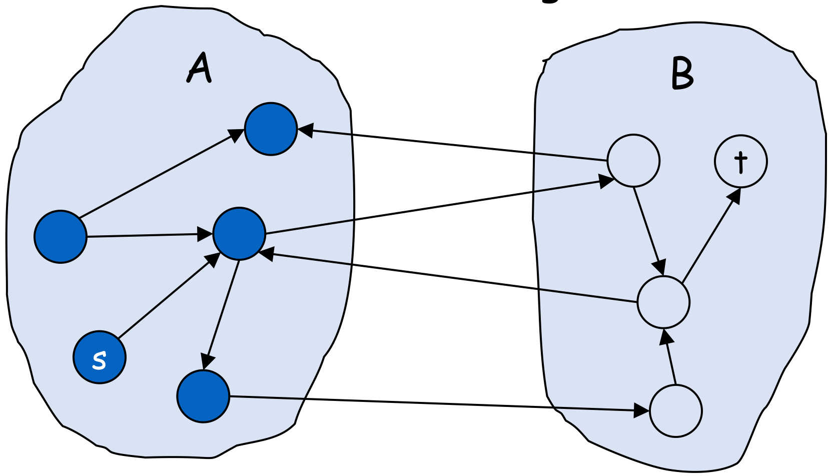
- (ii) $\Rightarrow$ (iii) We show contrapositive.
  - Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.

Proof of Max-Flow Min-Cut Theorem

(iii) $\Rightarrow$ (i)

- Let f be a flow with no augmenting paths.
- Let A be set of vertices reachable from s in residual graph G^f.
- By definition of A, s $\in$ A.
- By definition of f, t $\notin$ A. Let B be the set of all vertices not in A, $t \in B$, so (A, B) is an s-t cut.
- Consider two cases: e is an edge in G
  - e = (u, v), $u \in A$ and $v \in B$
    then f(e) = c_e
  - e = (v, u), $u \in B$ and $v \in A$
    then f(e) = 0

original network



$$v(f) = f^{out}(A) - f^{in}(A)$$

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = \sum_{e \text{ out of } A} c_e - 0 = c(A, B)$$

Running Time

Assumption. All capacities are integers between 1 and C.

Invariant. Every flow value $f(e)$ and every residual capacity $c_f(e)$ remains an integer throughout the algorithm.

Theorem. The algorithm terminates in $v(f) \leq nC$ iterations.
Pf. Each augmentation increase value by at least 1. ▪

Corollary. If $C = 1$, Ford-Fulkerson runs in $O(mn)$ time.

Integrality theorem. If all capacities are integers, then there exists a max flow f for which every flow value $f(e)$ is an integer.
Pf. Since algorithm terminates, theorem follows from invariant. ▪