Network Flow

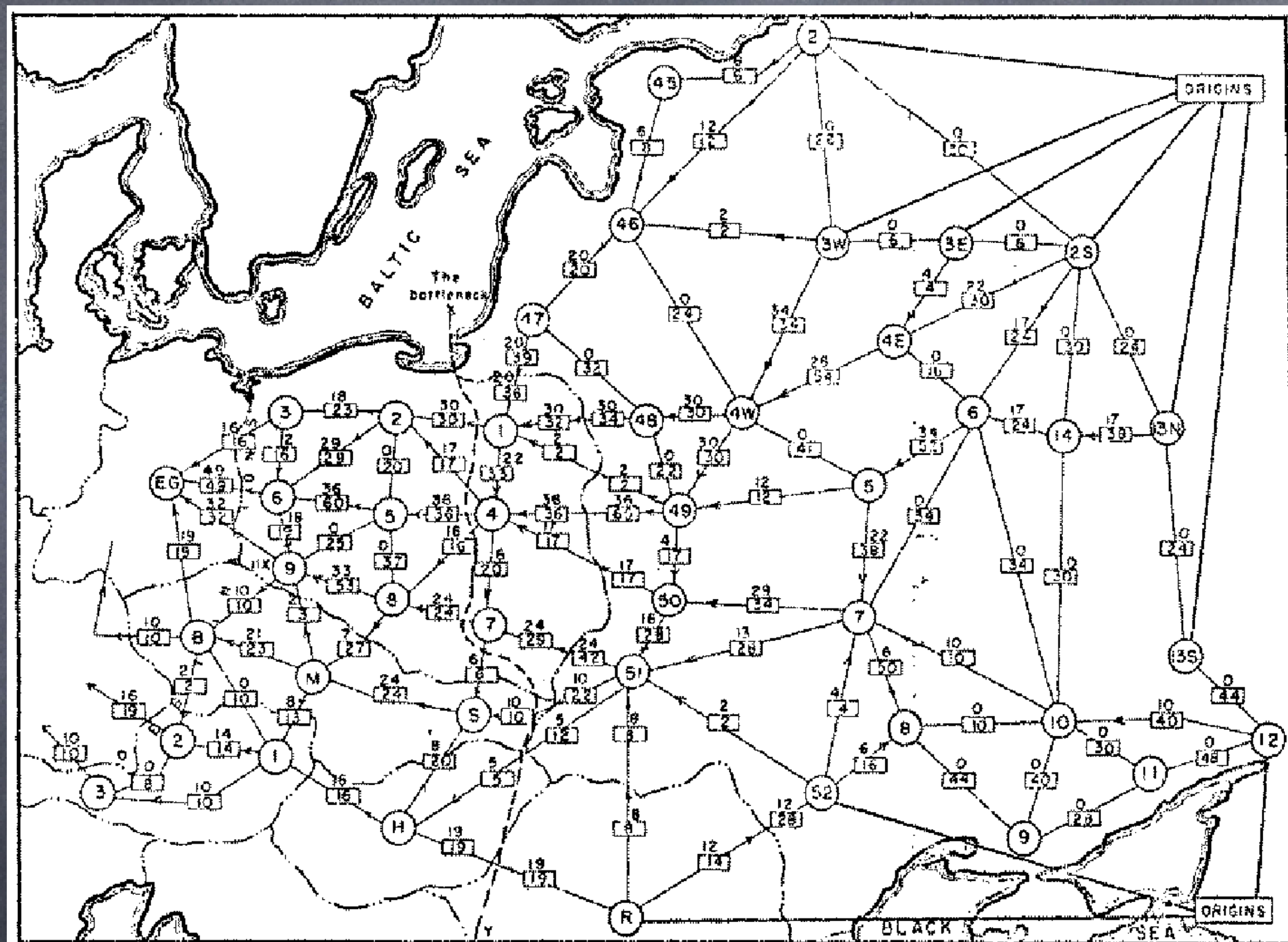
Algorithm Design

	Greedy	Divide and Conquer	Dynamic Programming
Formulate problem	?	?	?
Design algorithm	less work	more work	more work
Prove correctness	more work	less work	less work
Analyze running time	less work	more work	less work

Network Flow

- Greedy, Divide-and-Conquer, and Dynamic Programming were design techniques
- Network flow → a specific class of problems.
  - Useful in many different applications! (matching, transportation, network design, etc.)
- Goal: design and analyze algorithms for max-flow problem, then apply to solve other problems

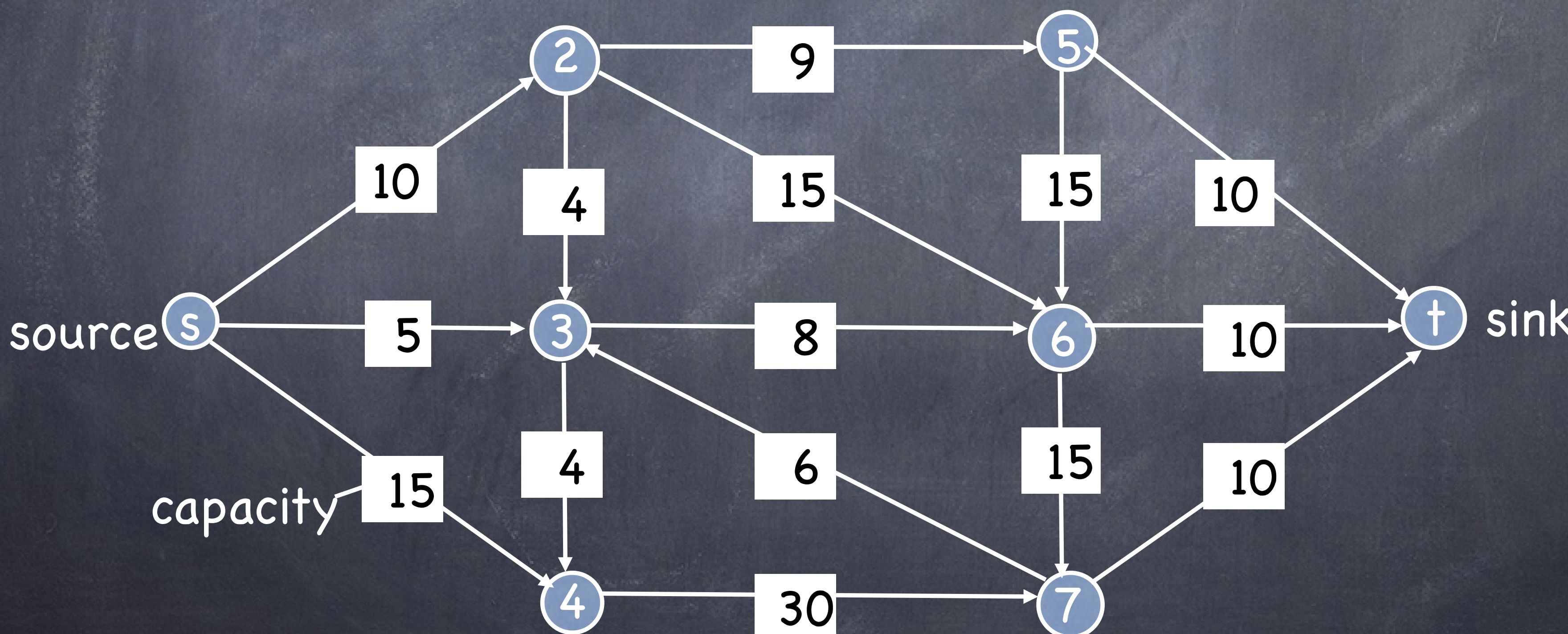
Soviet Rail Network, 1955



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

Flow Networks

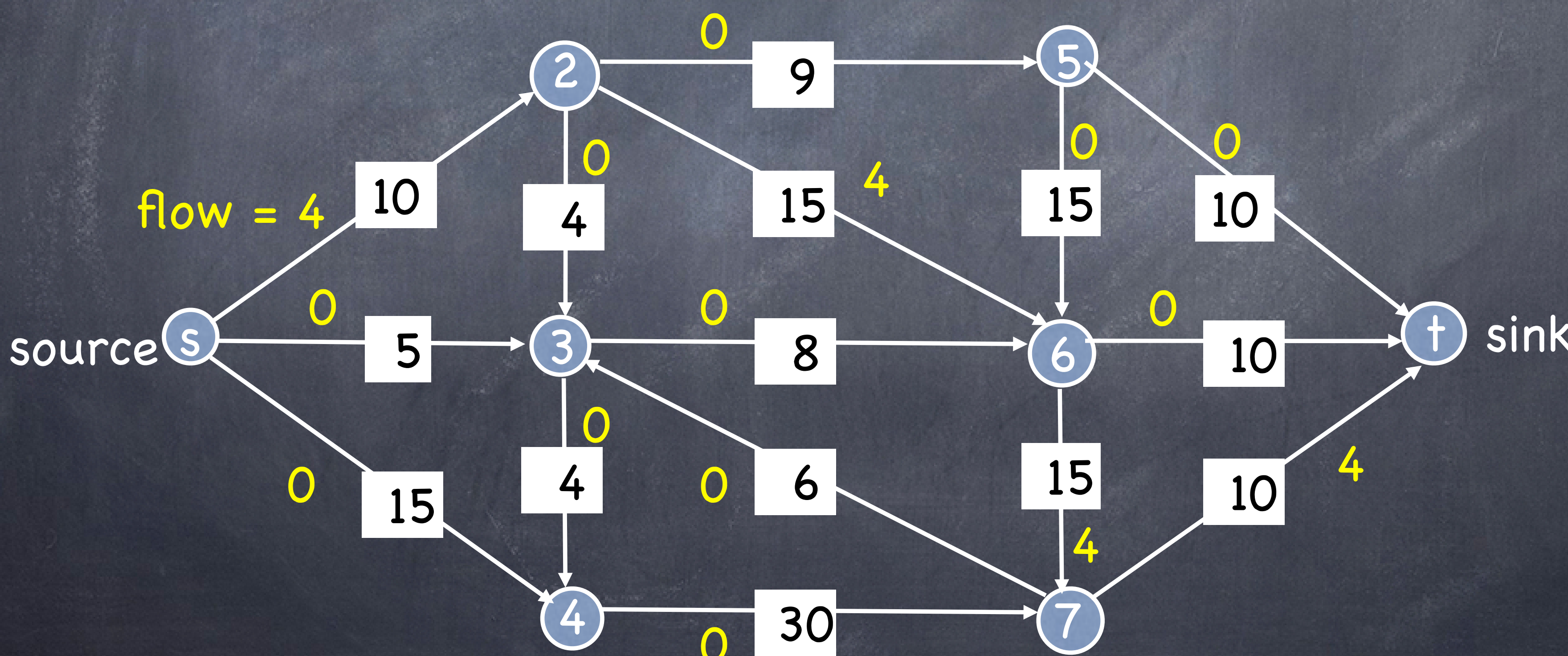
- Flow network.
  - Abstraction for material flowing through the edges.
  - G = (V, E) = directed graph
  - Two distinguished nodes: s = source, t = sink.
  - c(e) = capacity of edge e.



Flows

- An s-t flow is a function f: $E \rightarrow R^+$ that satisfies:
  - Capacity condition: For each $e \in E$: $0 \leq f(e) \leq c(e)$
  - Conservation condition: For each $v \in V - \{s, t\}$:

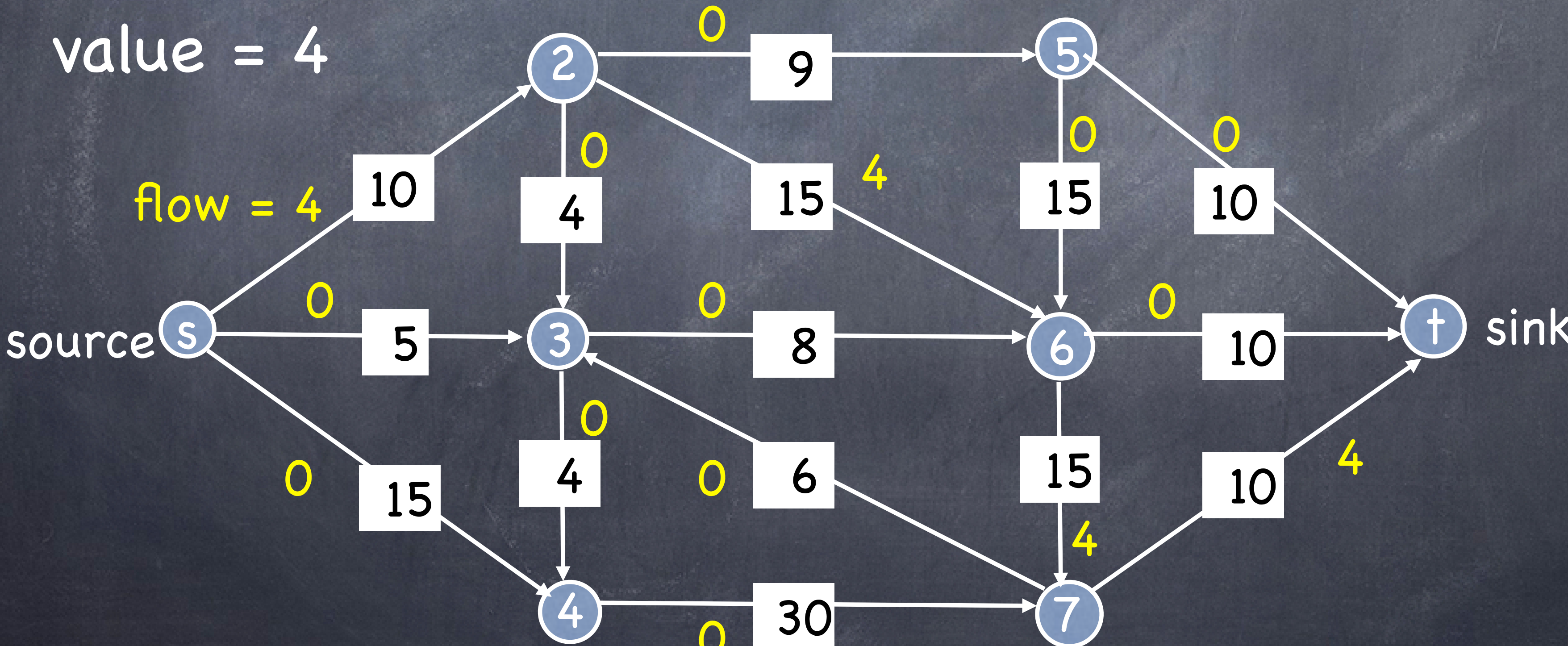
$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$



Flows

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

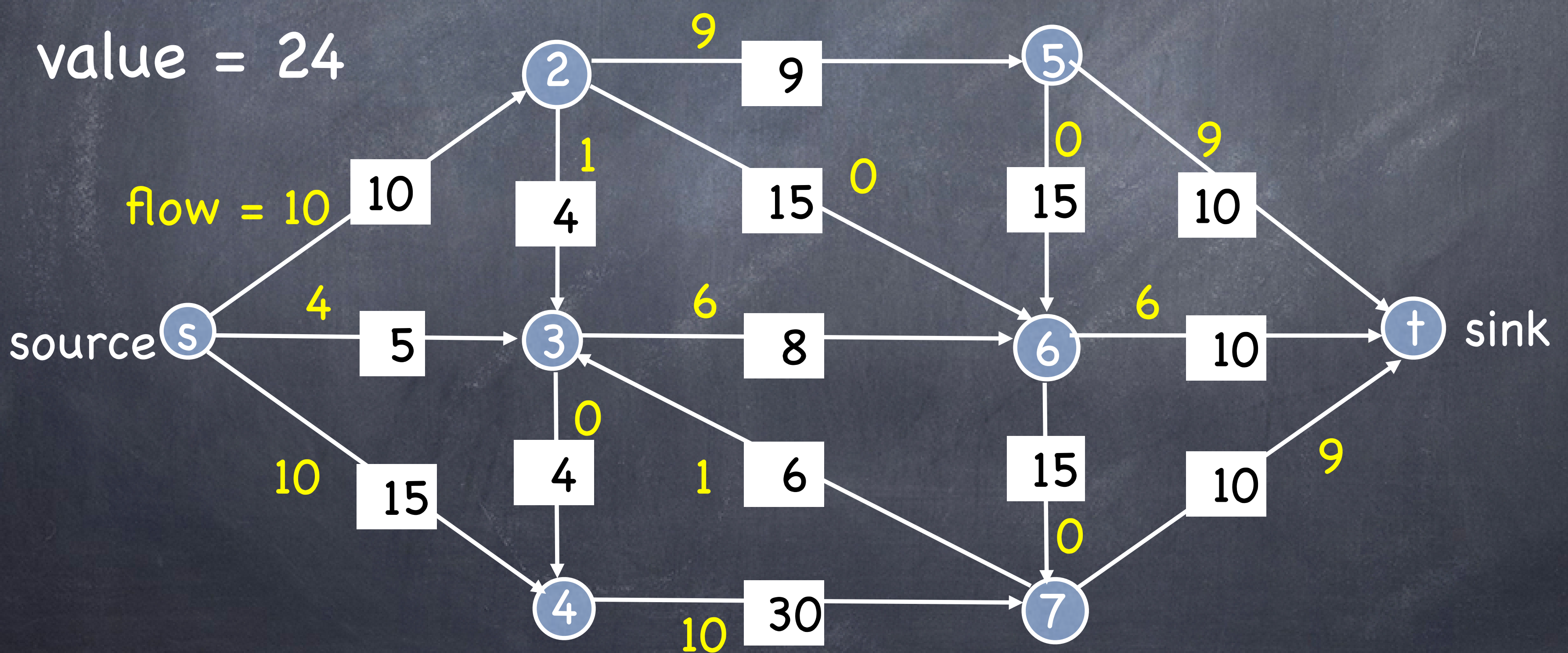
value = 4



Flows

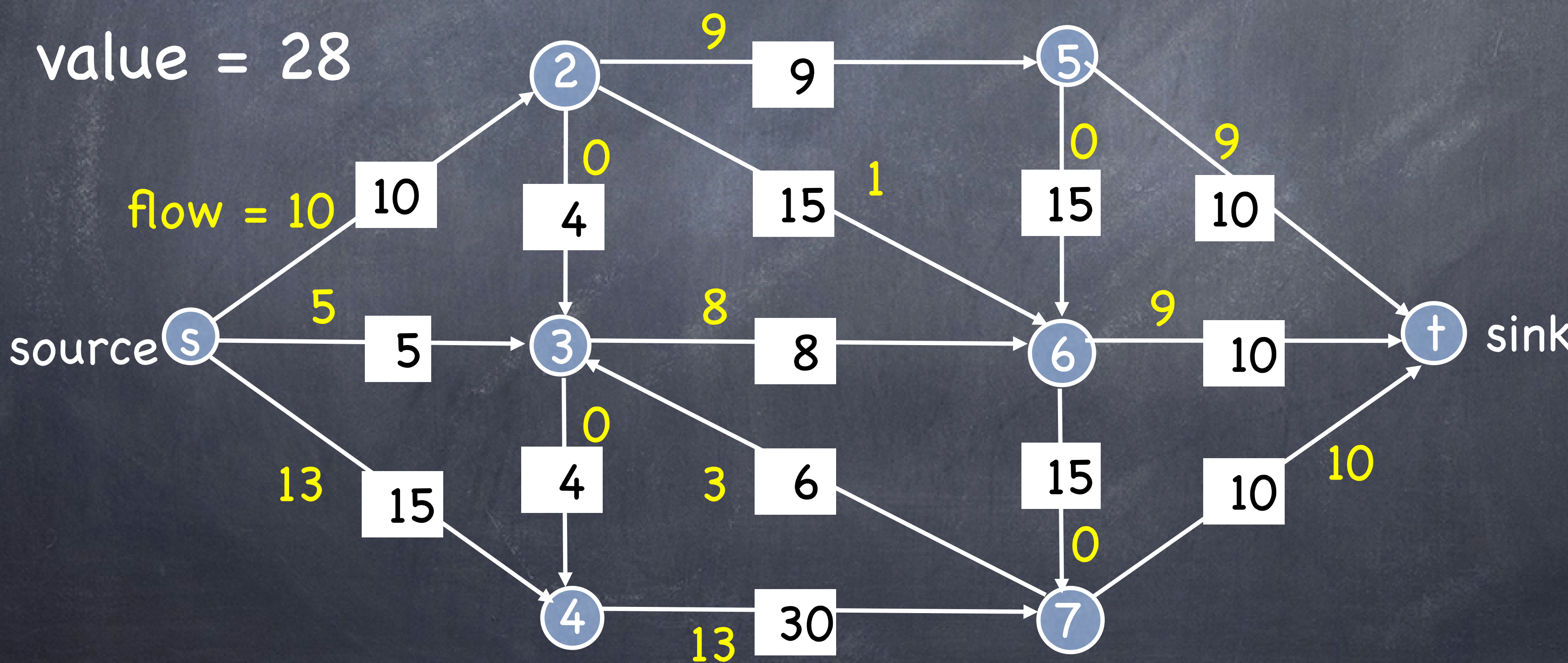
- The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$

value = 24



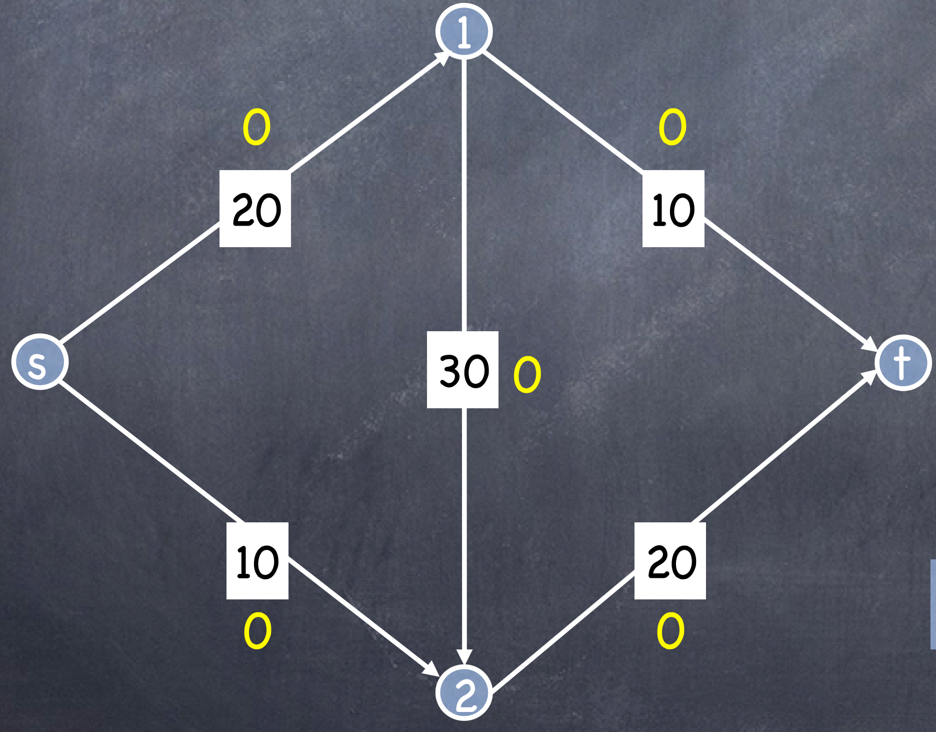
Maximum Flow Problem

Find s-t flow of maximum value.



Towards a Max Flow Algorithm

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



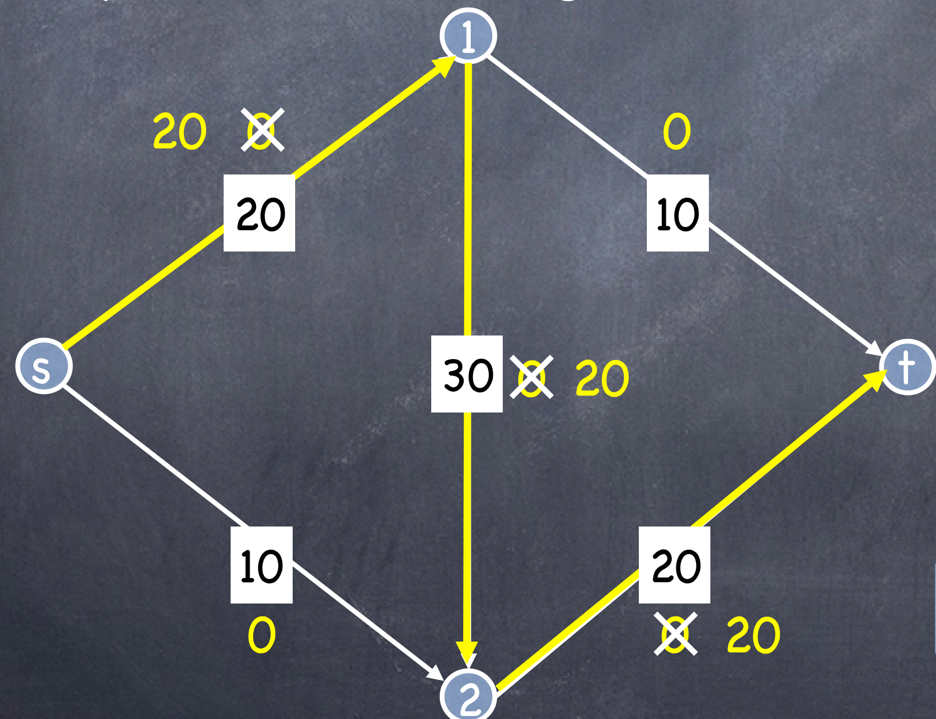
Flow value = 0

Towards a Max-Flow Algorithm

Key idea: repeatedly choose paths and "augment" the amount of flow on those paths as much as possible until capacities are met

Towards a Max Flow Algorithm

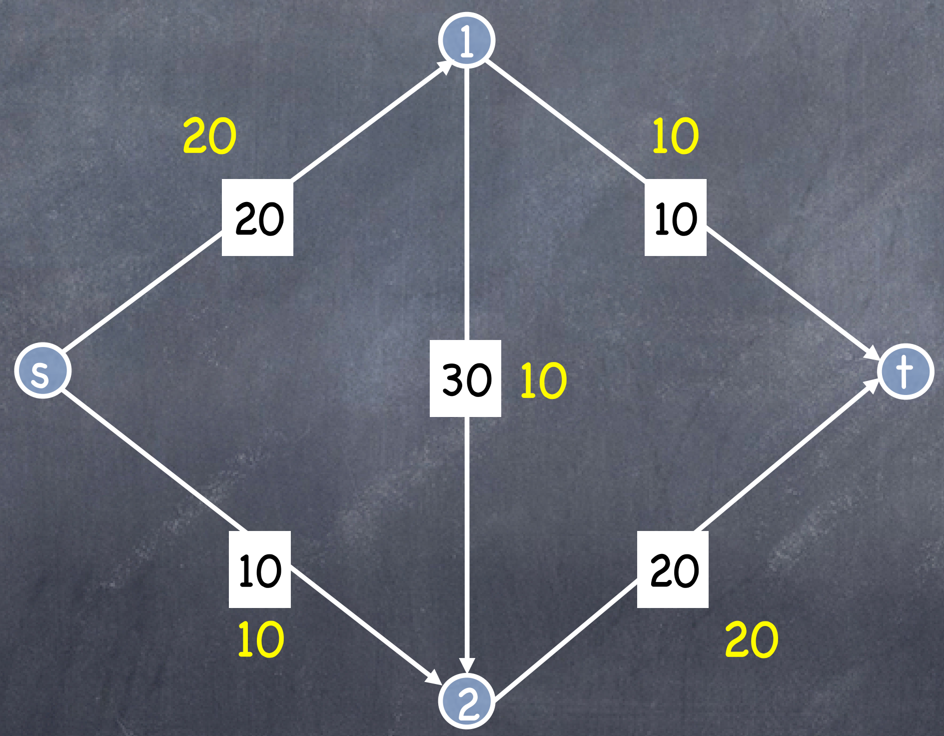
- Greedy algorithm.
  - Start with $f(e) = 0$ for all edges $e \in E$.
  - Find an s-t path P where each edge has $f(e) < c(e)$.
  - Augment flow along path P.
  - Repeat until you get stuck.



Flow value = ~~0~~ 20

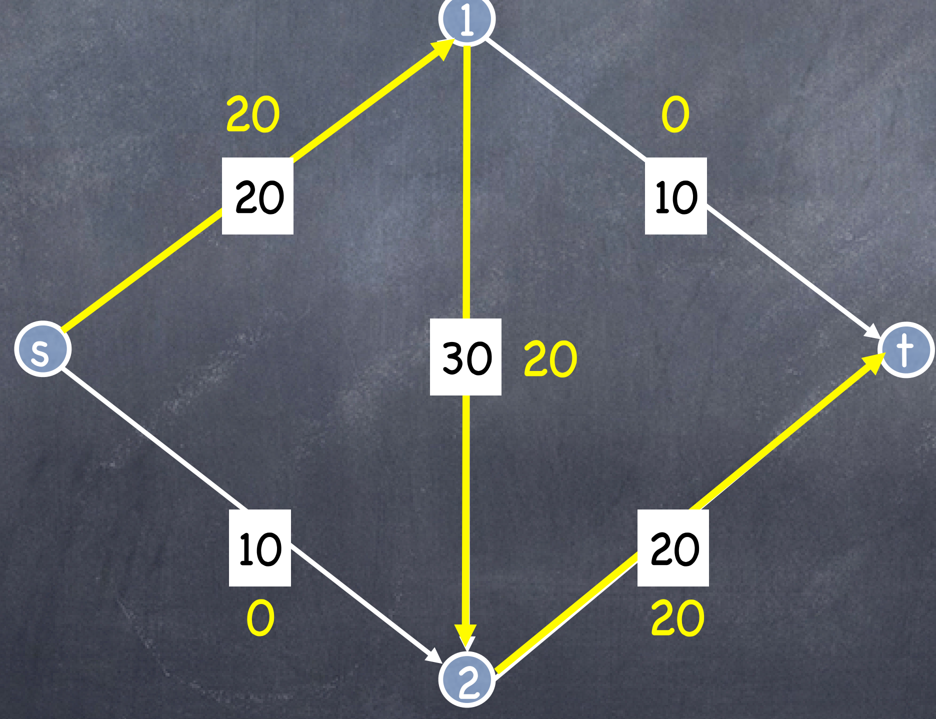
Optimal Solution

Flow value = 30



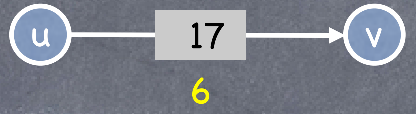
Problem

To fix the greedy algorithm, we need a way to track:
(1) how much more flow can we send on any edge?
(2) how much flow can we "undo" on each edge?

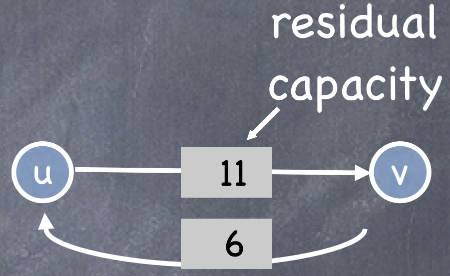


Residual Graph

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



- Create two residual edges
  - "Forward edge"
    $e = (u, v)$ with capacity $c(e) - f(e)$
  - "Backward edge"
    $e' = (v, u)$ with capacity $f(e)$



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

Augmenting Path

Use path P in G_f to to update flow in G

```
Augment(f, P) {
    b = bottleneck(P)         // edge on P with least residual capacity
    foreach e = (u,v) ∈ P {
        if e is a forward edge
            f(e) = f(e) + b        // forward edge: increase flow
        else
            let e′ = (v, u)
            f(e′) = f(e′) - b      // backward edge: decrease flow
    }
    return f
}
```

Example on board

Ford-Fulkerson Algorithm

Repat: find an augmenting path, and augment!

```
Ford-Fulkerson(G, s, t) {
   foreach e ∈ E  f(e) = 0 // initially, no flow
   Gf = copy of G          // residual graph = original graph

   while (there exists an s-t path P in Gf) {
      f = Augment(f, P)    // change the flow
      update Gf            // build a new residual graph
   }
   return f
}
```

Next Time

- Termination and running time (easy)
- Correctness: Max-Flow Min-Cut Theorem