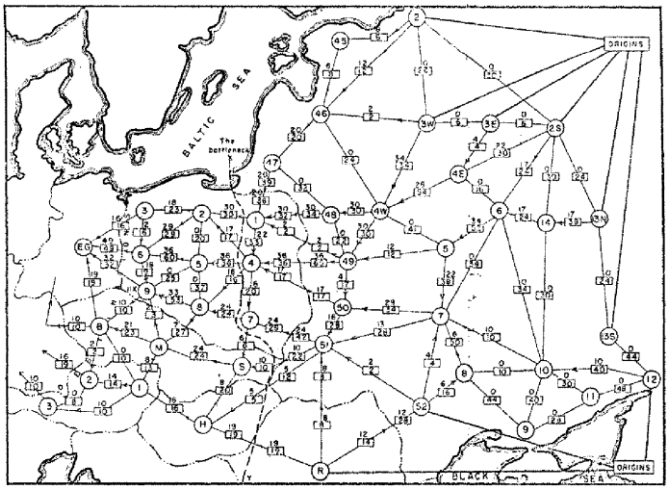
COMP 355
Advanced Algorithms

Network Flows:
Basics & Ford-Fulkerson Algorithm
Section 7.1-7.3 (KT)

Rhodes College
—1848—

1

Soviet Rail Network, 1955



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

2

Network Flows

Def. Name of a variety of related graph optimization problems

Given a flow network, which is essentially a directed graph with nonnegative edge weights.

- Think of the edges as "pipes" that are capable of carrying some sort of "stuff."
- Each edge of the network has a given *capacity*
- How much flow we can push from a designated source node to a designated sink node?

3

Maximum Flow and Minimum Cut

Max flow and min cut.

- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

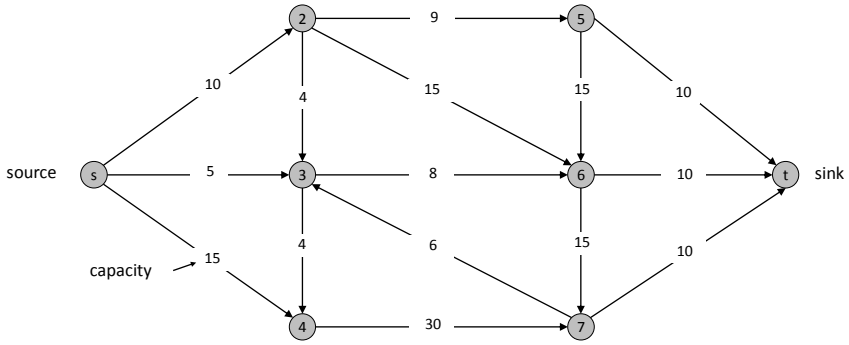
Nontrivial applications / reductions.

- 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.
- And many more . . .

4

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. (non-negative)



5

Flows, Capacities, and Conservation

Given an s-t network, a *flow* is a function f that maps each edge to a nonnegative real number and satisfies the following properties:

- **Capacity Constraint**: For all e ∈ E, f(u, v) ≤ c(u, v) = c(e)
- **Flow conservation (or flow balance)**: For all v ∈ V \ {s, t}, the sum of flow along edges into v equals the sum of flows along edges out of v. $f^{in}(v) = f^{out}(v)$

If edge (u,v) not in E, then f(u, v) = 0

$$f^{in}(v) = \sum_{u \in V} f(u, v) \qquad f^{out}(v) = \sum_{w \in V} f(v, w)$$

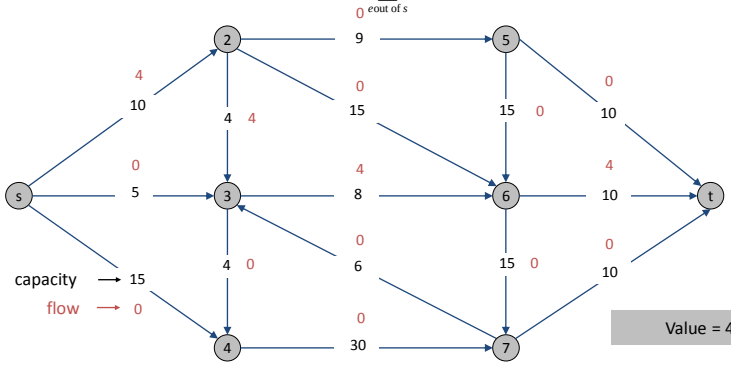
6

Flows

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

- For each e ∈ E: $0 \le f(e) \le c(e)$ (capacity)
- For each v ∈ 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: $v(f) = \sum_{e \text{out of } s} f(e)$.



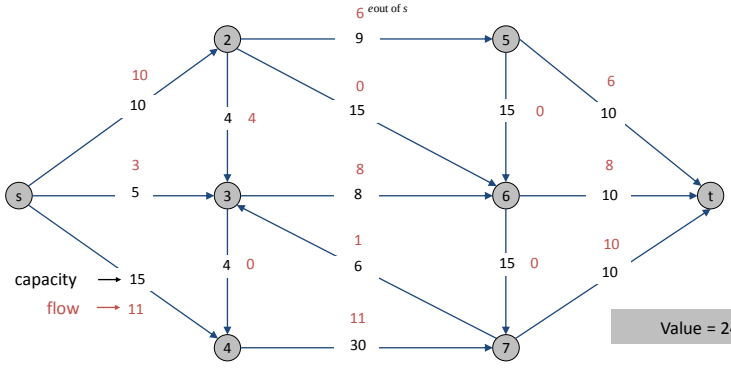
7

Flows

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

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

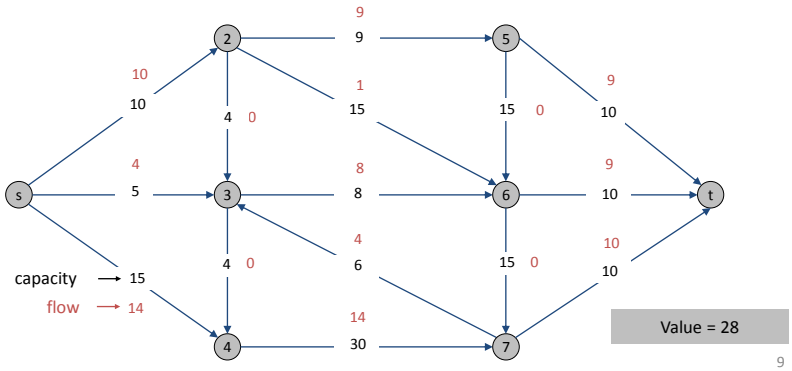
Def. The value of a flow f is: $v(f) = \sum_{e \text{out of } s} f(e)$.



8

Maximum Flow Problem

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



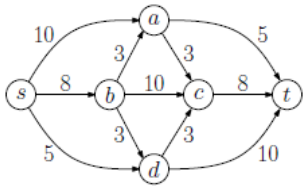
9

Path-Based Flows

Define an *s-t path* to be any simple path from s to t.

Ex. <s, a, t> <s, b, a, c, t> and <s, d, c, t> are all examples of s-t paths.

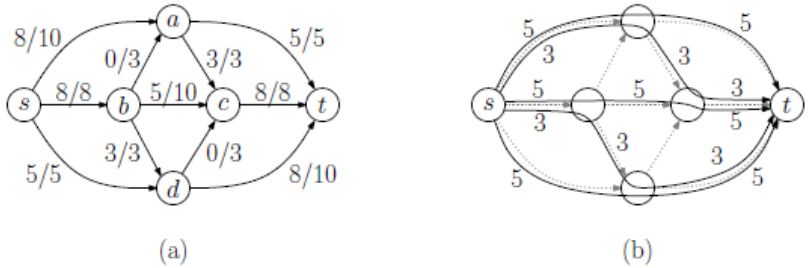
Def. A *path-based flow* is a function that assigns each s-t path a nonnegative real number such that, for every edge (u, v) ∈ E, the sum of the flows on all the paths containing this edge is at most c(u, v).



10

Path-Based Flows

No need to provide a flow conservation constraint (each path that carries a flow into a vertex (excluding s and t), carries an equivalent amount of flow out of that vertex)



(a) An edge-based flow and (b) its path-based equivalent.

11

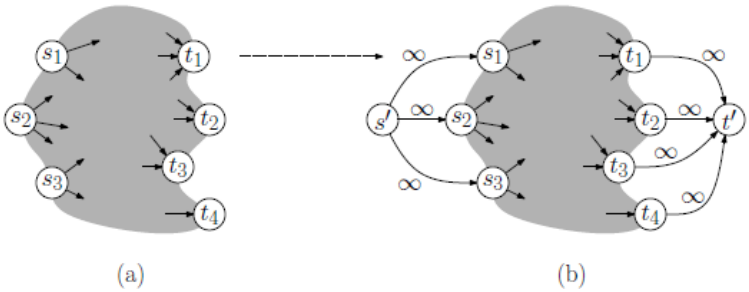
Path-Based Flows

Def. The *value* of a path-based flow is defined to be the total sum of all the flows on all the s-t paths of the network.

Claim: Given an s-t network G, under the assumption that there are no edges entering s or leaving t, G has an edge-based flow of value x if and only if G has a path-based flow of value x.

12

Multi-source, multi-sink networks

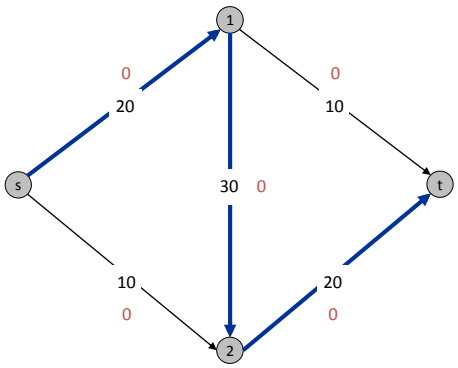


Reduction from (a) multi-source/multi-sink to (b) single-source/single-sink.

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.

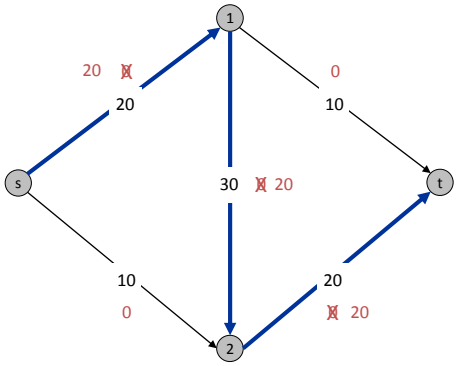


Flow value = 0

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

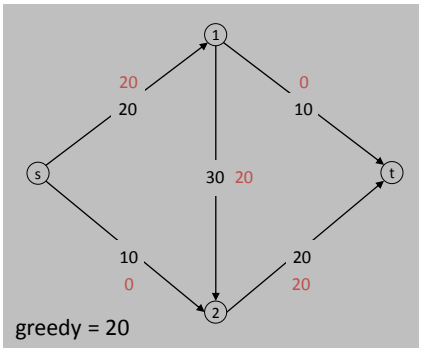
15

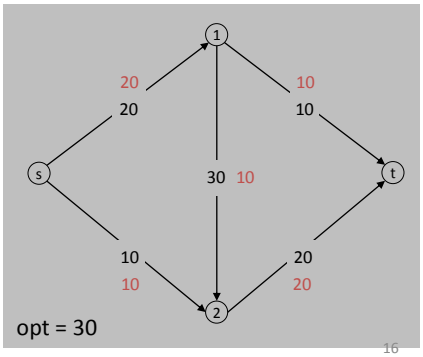
Towards a Max Flow Algorithm

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.

locally optimality ⇏ global optimality

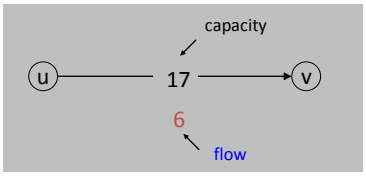




16

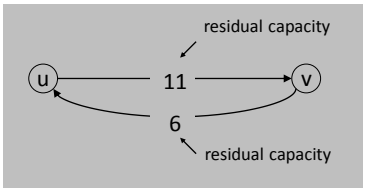
Residual Graph

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:

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



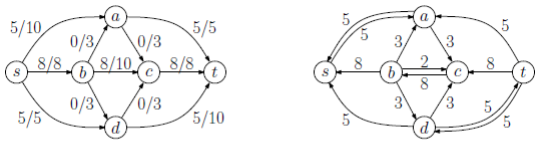
Residual graph: $G_f = (V, E_f)$.
- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : c(e) > 0\}$.

17

Residual Graph

Forward edges: For each edge (u, v) for which $f(u, v) < c(u, v)$, create an edge (u, v) in G_f and assign it the capacity $c_f(u, v) = c(u, v) - f(u, v)$. Intuitively, this edge signifies that we can add up to $c_f(u, v)$ additional units of flow to this edge without violating the original capacity constraint.

Backward edges: For each edge (u, v) for which $f(u, v) > 0$, create an edge (v, u) in G_f and assign it a capacity of $c_f(v, u) = f(u, v)$. Intuitively, this edge signifies that we can cancel up to $f(u, v)$ units of flow along (u, v). Conceptually, by pushing positive flow along the reverse edge (v, u) we are decreasing the flow along the original edge (u, v).



(a): A flow f in network G (b): Residual network G_f

A flow f and the residual network G_f.

18

Augmenting Paths

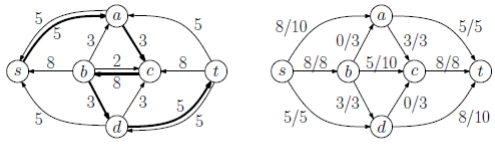
Consider a network G, let f be a flow in G, and let G_f be the associated residual network.

Def. An *augmenting path* is a simple path P from s to t in G_f.

Def. The *residual capacity* (also called the bottleneck capacity) of the path is the minimum capacity of any edge on the path. It is denoted $c_f(P)$.

Recall: all the edges of G_f are of strictly positive capacity, so $c_f(P) > 0$.

By pushing $c_f(P)$ units of flow along each edge of the path, we obtain a valid flow in G_f, and by the previous lemma, adding this to f results in a valid flow in G of strictly higher value.



(a): Augmenting path of capacity 3 (b): The flow after augmentation

19

Augmenting Path Algorithm

```
Augment(f, c, P) {
   b ← bottleneck(P)
   foreach e ∈ P {
      if (e ∈ E) f(e) ← f(e) + b      forward edge
      else       f(e^R) ← f(e) - b    reverse edge
   }
   return f
}
```

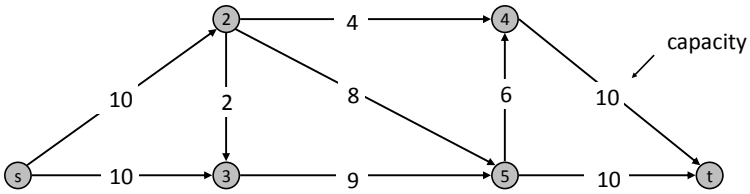
```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   G_f ← residual graph

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

20

Ford-Fulkerson Algorithm

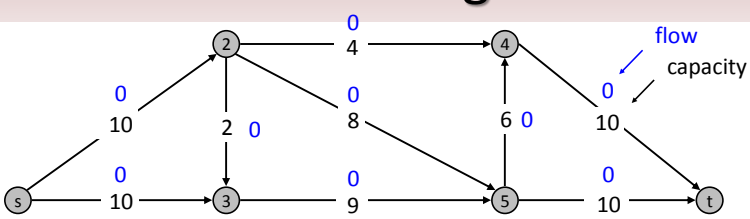
G:



21

Ford-Fulkerson Algorithm

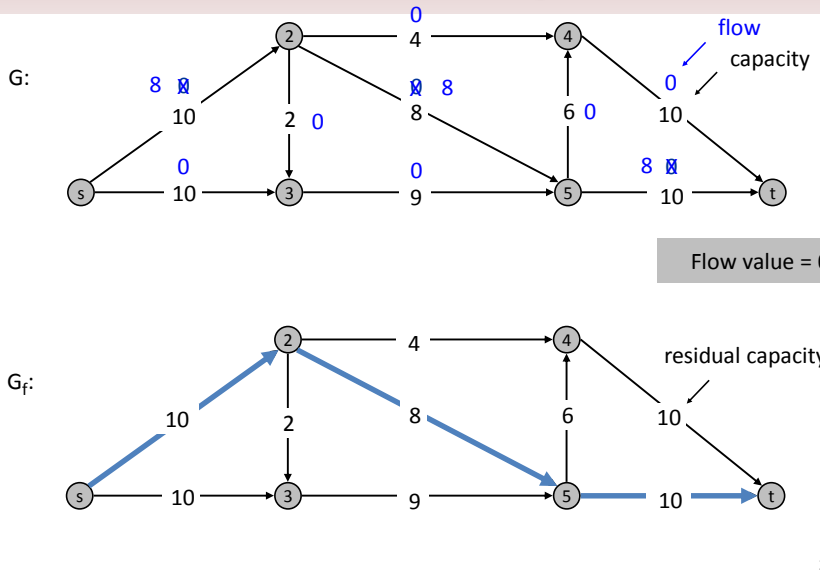
G:



Flow value = 0

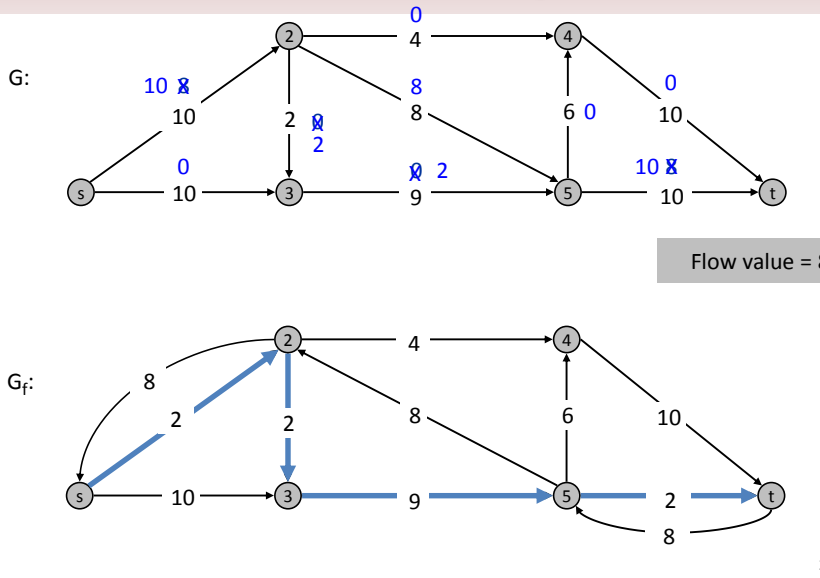
22

Ford-Fulkerson Algorithm



23

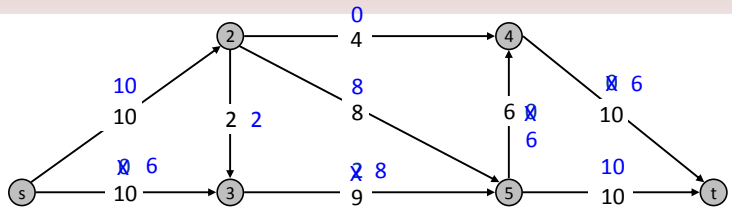
Ford-Fulkerson Algorithm



24

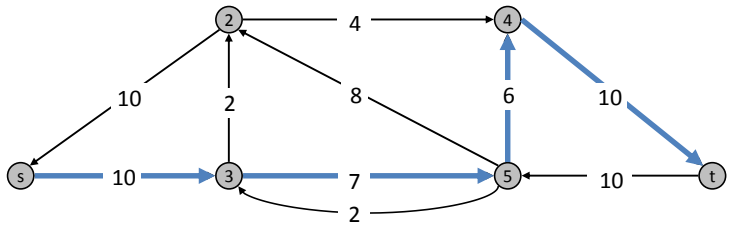
Ford-Fulkerson Algorithm

G:



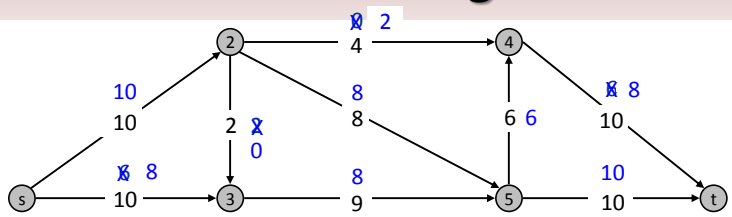
Flow value = 10

G_f:



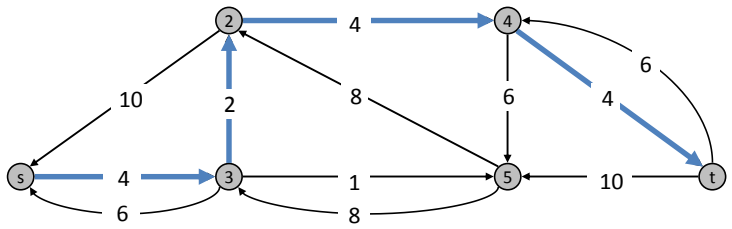
Ford-Fulkerson Algorithm

G:

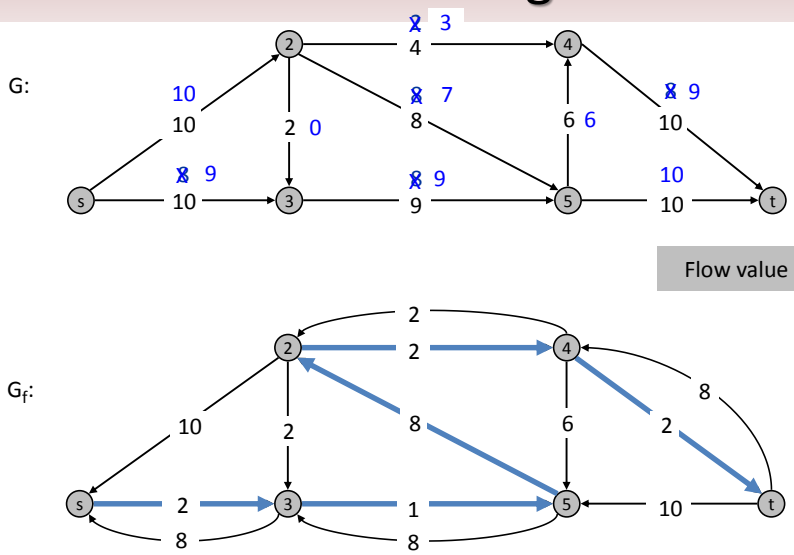


Flow value = 16

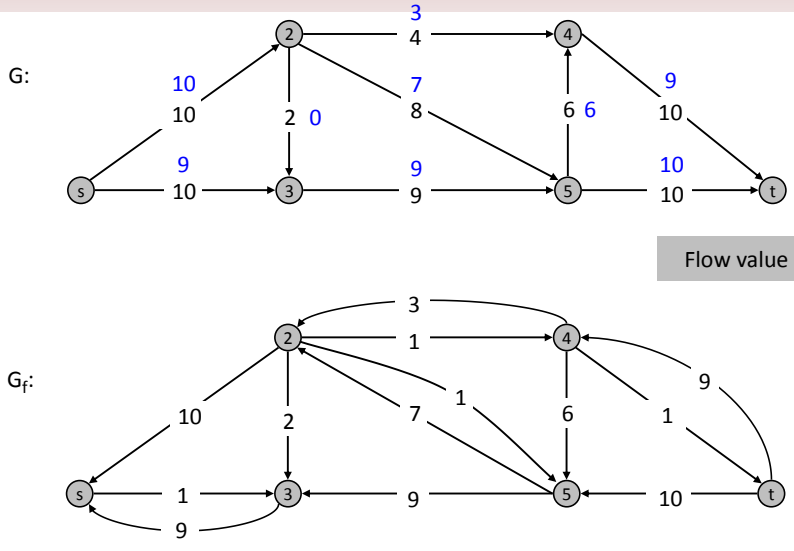
G_f:



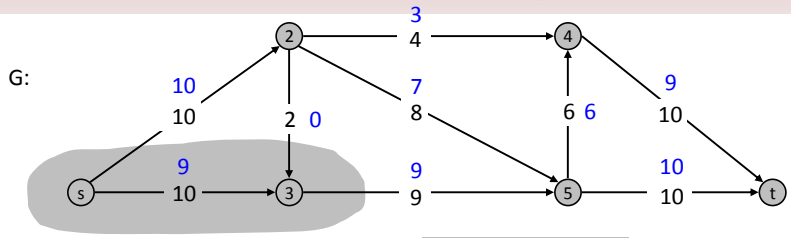
Ford-Fulkerson Algorithm



Ford-Fulkerson Algorithm

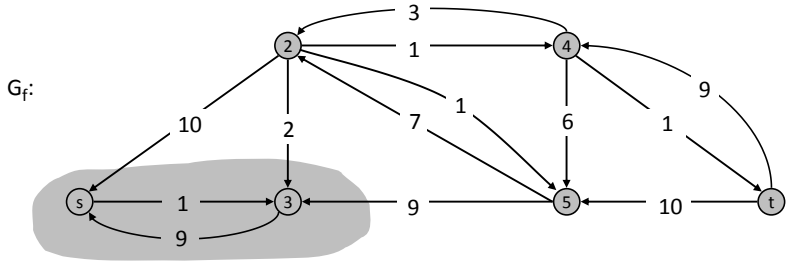


Ford-Fulkerson Algorithm



Cut capacity = 19

Flow value = 19

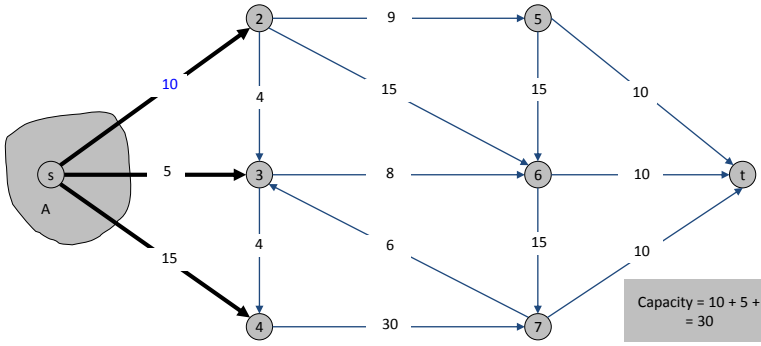


29

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



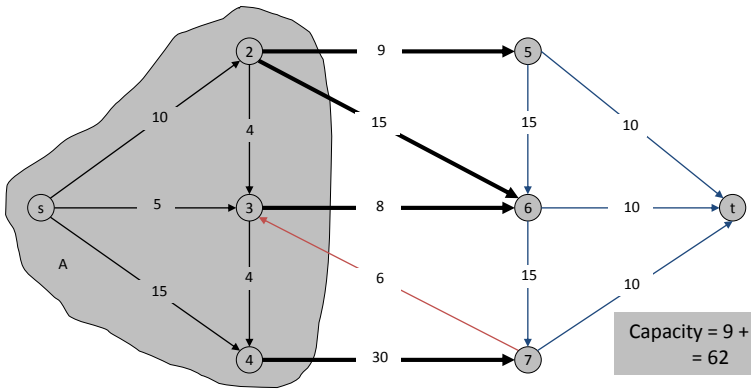
Capacity = 10 + 5 + 15
= 30

30

Cuts

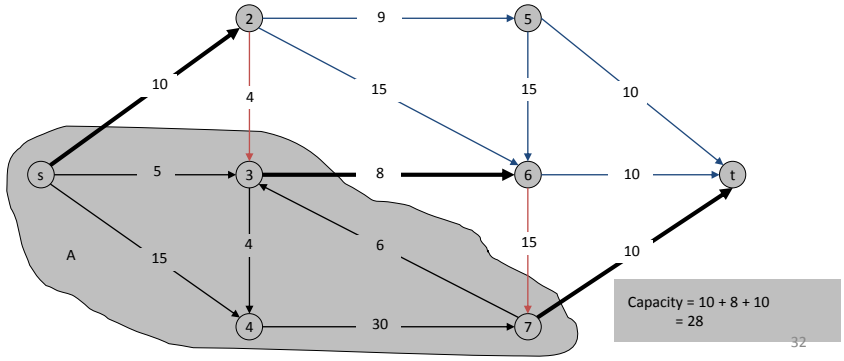
Def. An s-t cut is a partition (A, B) of V with s ∈ A and t ∈ B.

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



Minimum Cut Problem

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



32