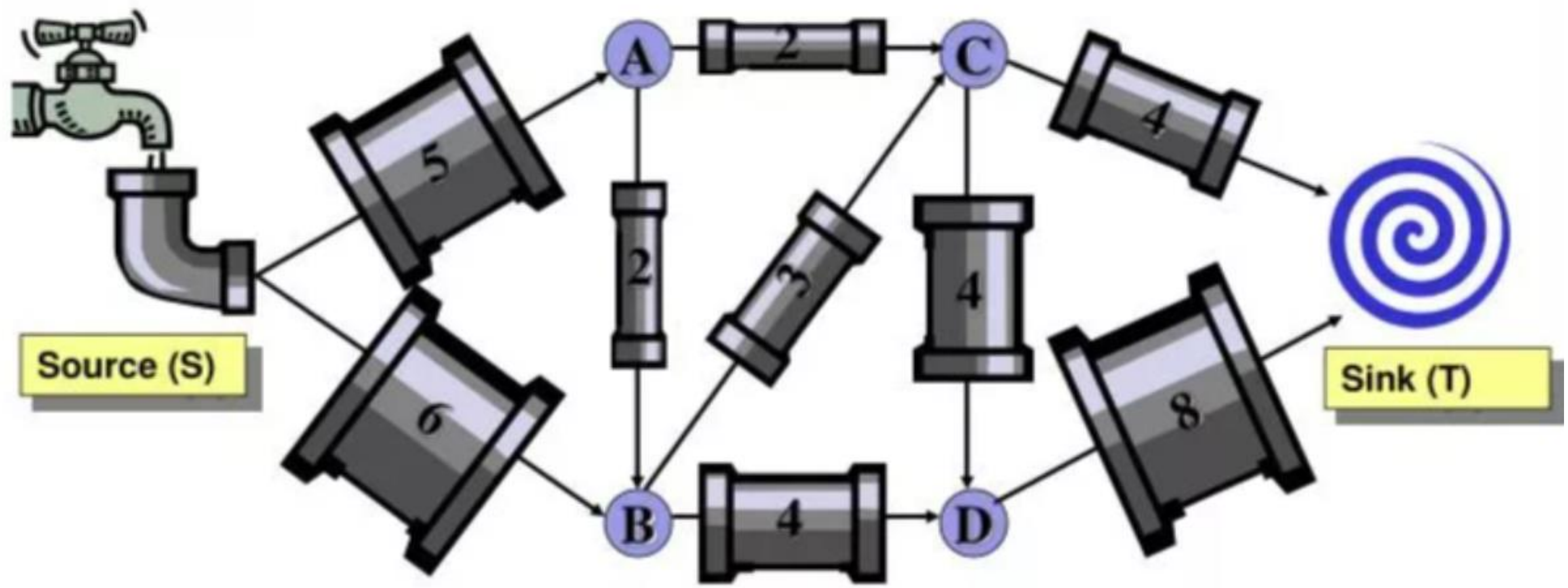
Lecture 23: Maximum Flow



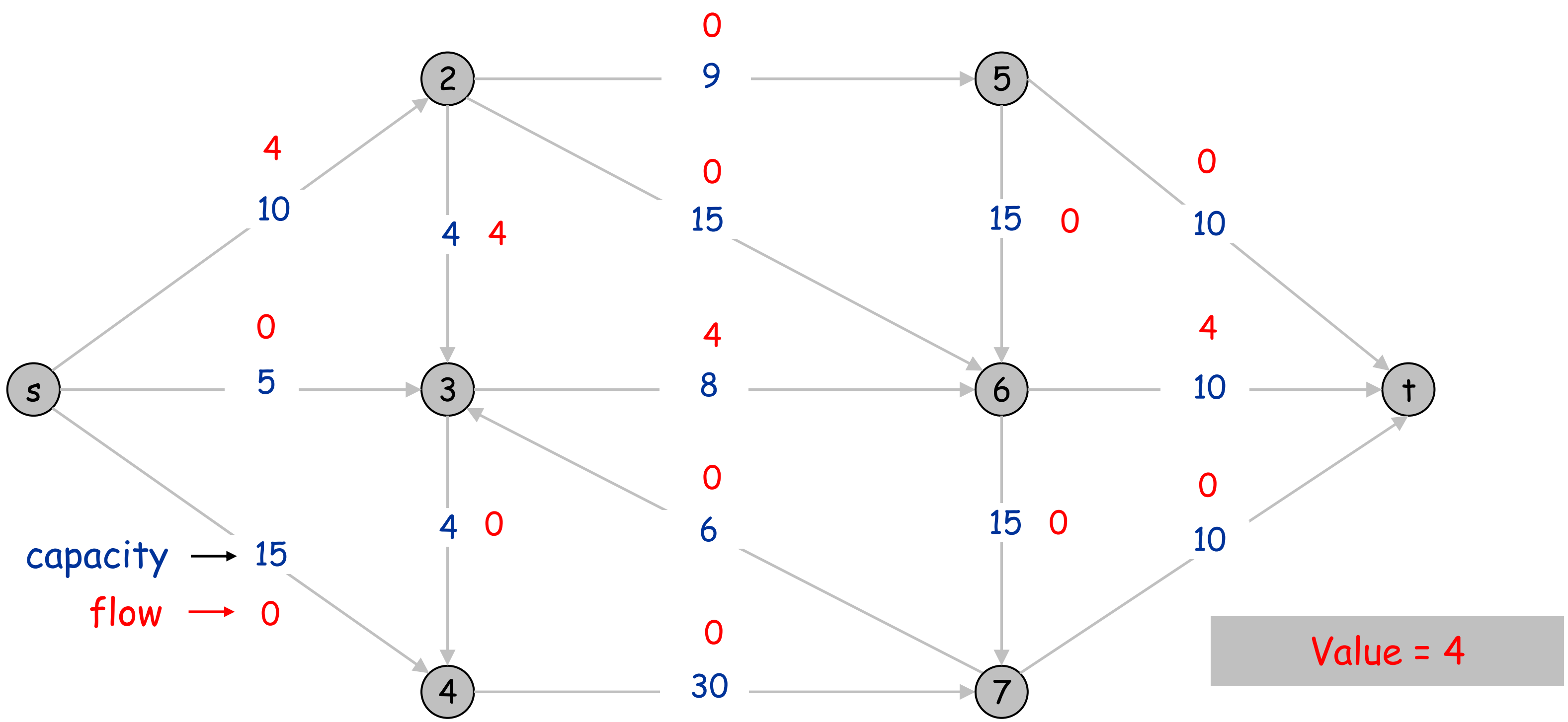
Flow

Input: A directed connected graph $G = (V, E)$, where

- every edge $e \in E$ has a capacity $c(e)$;
- a source vertex s and a target vertex t.

Output: A flow $f: E \to \mathbf{R}$ from s to t, such that

- **For all** $\boldsymbol{e} \in \boldsymbol{E}$, $0 \le f(e) \le c(e)$ (capacity)
- **For all** $\boldsymbol{v} \in \boldsymbol{V} - \{\boldsymbol{s}, \boldsymbol{t}\}$, $\sum_{\boldsymbol{e} \text{ out of } \boldsymbol{v}} \boldsymbol{f}(\boldsymbol{e}) = \sum_{\boldsymbol{e} \text{ into } \boldsymbol{v}} \boldsymbol{f}(\boldsymbol{e})$ **(conservation)**



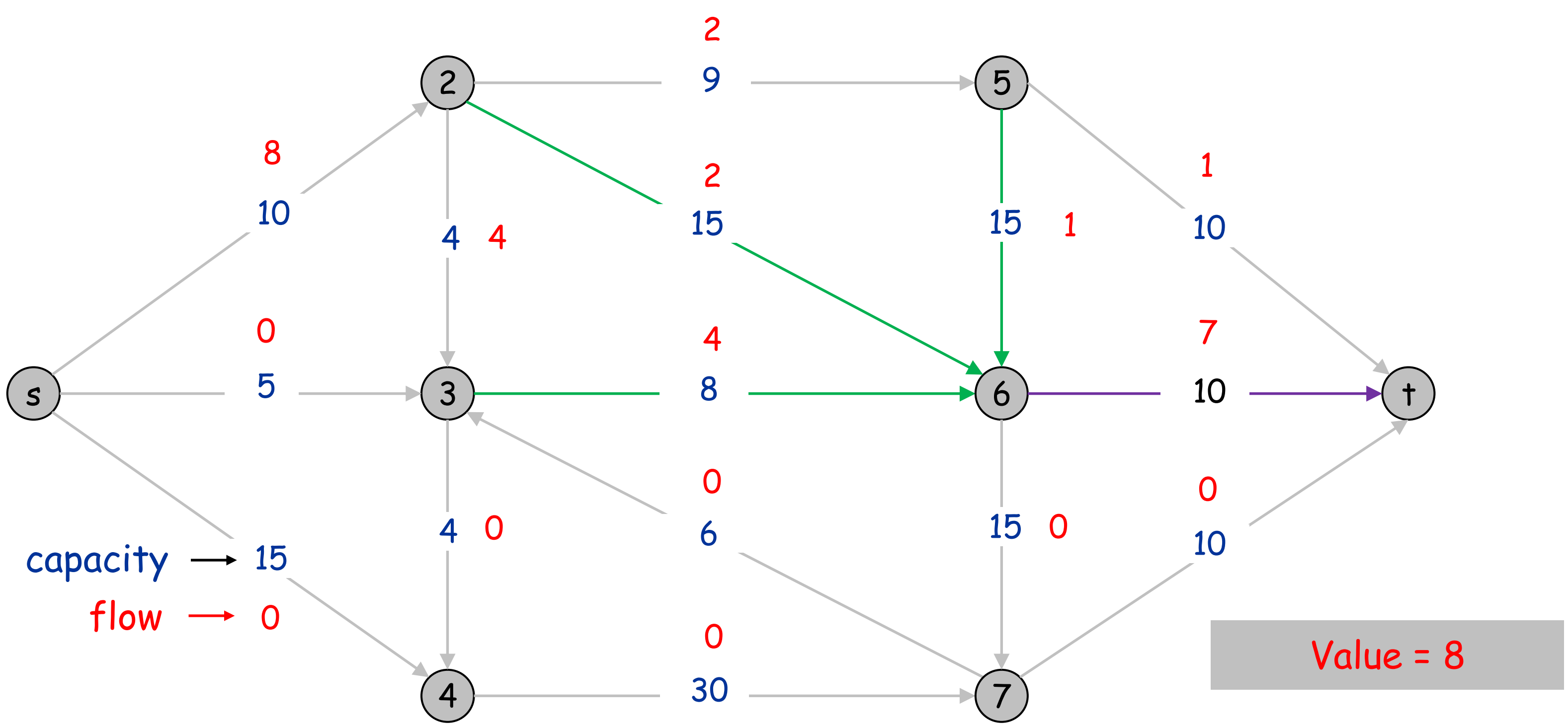
Flow

Input: A directed connected graph $G = (V, E)$, where

- every edge $e \in E$ has a capacity $c(e)$;
- a source vertex s and a target vertex t.

Output: A flow $f: E \to \mathbf{R}$ from s to t, such that

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



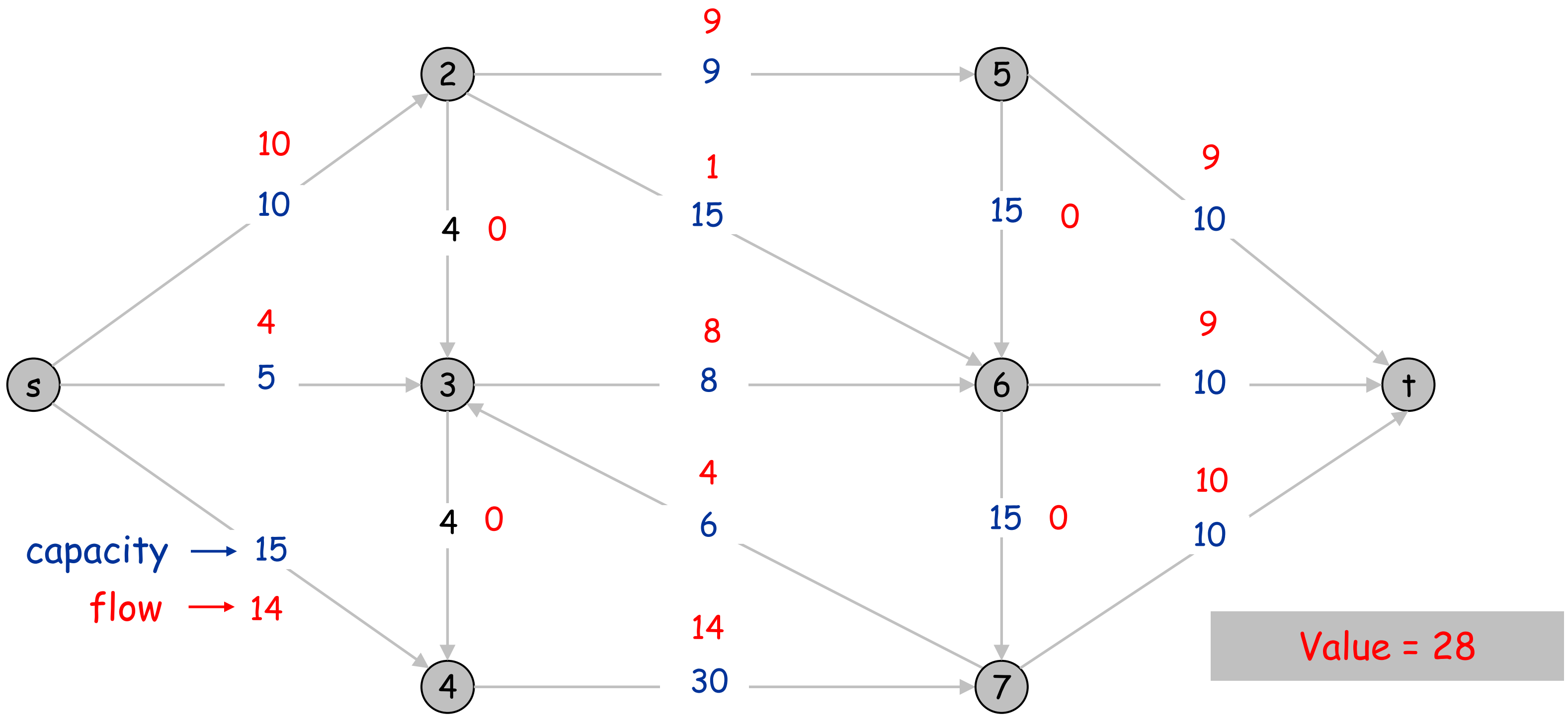
Maximum Flow

Def: The value of a flow f is $|f| = \sum_v f(s,v) = \sum_v f(v,t)$

The maximum flow problem is to find the flow with maximum value.

Example: The flow below is a maximum flow.

Q: How can we be sure this flow achieves the maximum value possible?



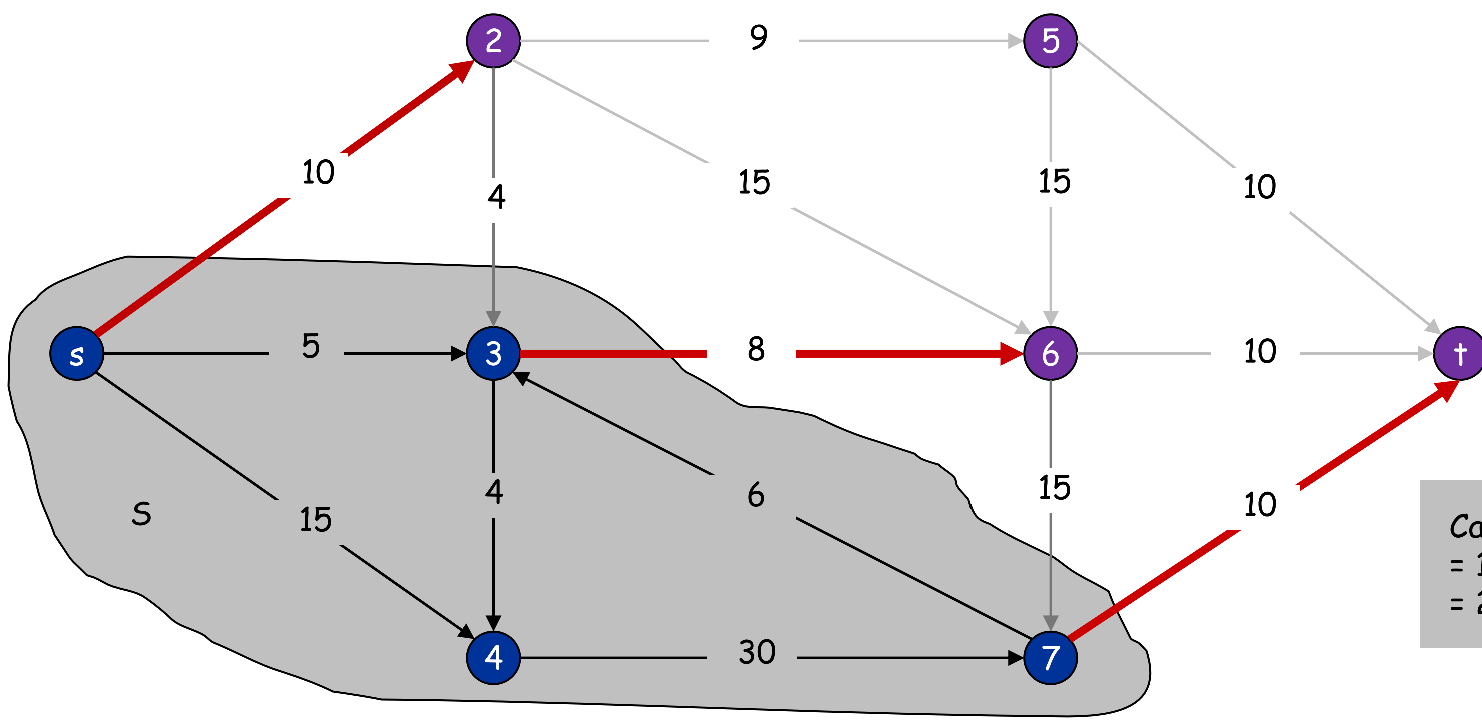
s-t Cut

Def: An s-t cut is a partition (S, T) of V with $s \in S$ and $t \in T$.

Def: The capacity of the cut (S, T) is $c(S, T) = \sum_{e \text{ from } S \text{ to } T} c(e)$

Claim: The value of any s-t flow cannot exceed the capacity of any s-t cut.

Observation (proved later): An s-t cut with capacity matching the value of a flow is a "proof" that the flow is a max flow.



Capacity(S,T)
= 10 + 8 + 10
= 28

Flow Applications

Direct applications

- Water flowing in pipes
- Electricity flows
- Vehicle traffic flows
- Communication network traffic flows

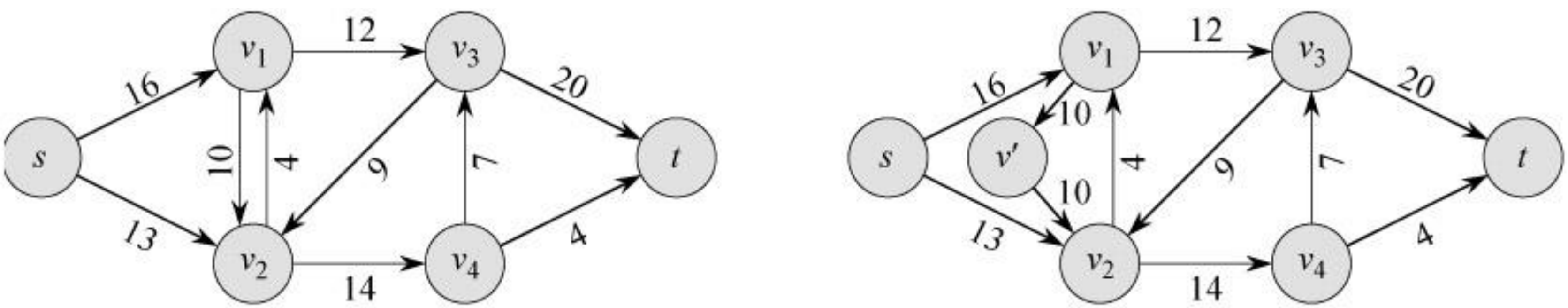
Indirect applications

- Bipartite matching
- Circulation-demand problem
- Baseball elimination
- Airline scheduling
- Fairness in car sharing (carpool)
- ...

Assumptions

Antiparallel edges

- $(u, v), (v, u) \in E$
- Models two-way traffic
- Causes problems in algorithms
- But can be removed by adding an auxiliary vertex
- Will assume no antiparallel edges



(a) (b)

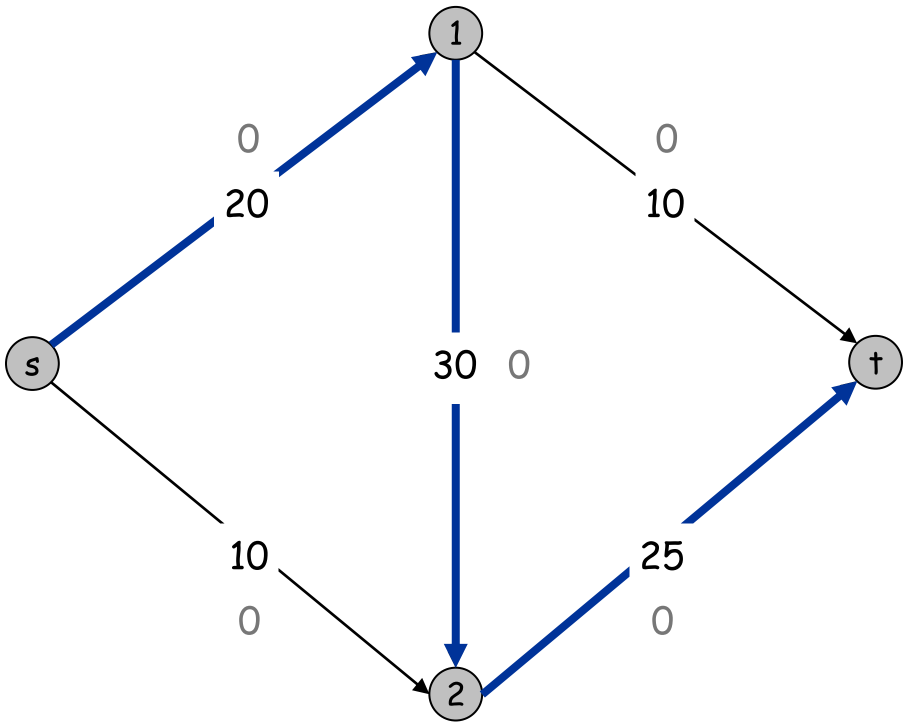
Also assume

- No edges going into s
- No edges going out of t

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.

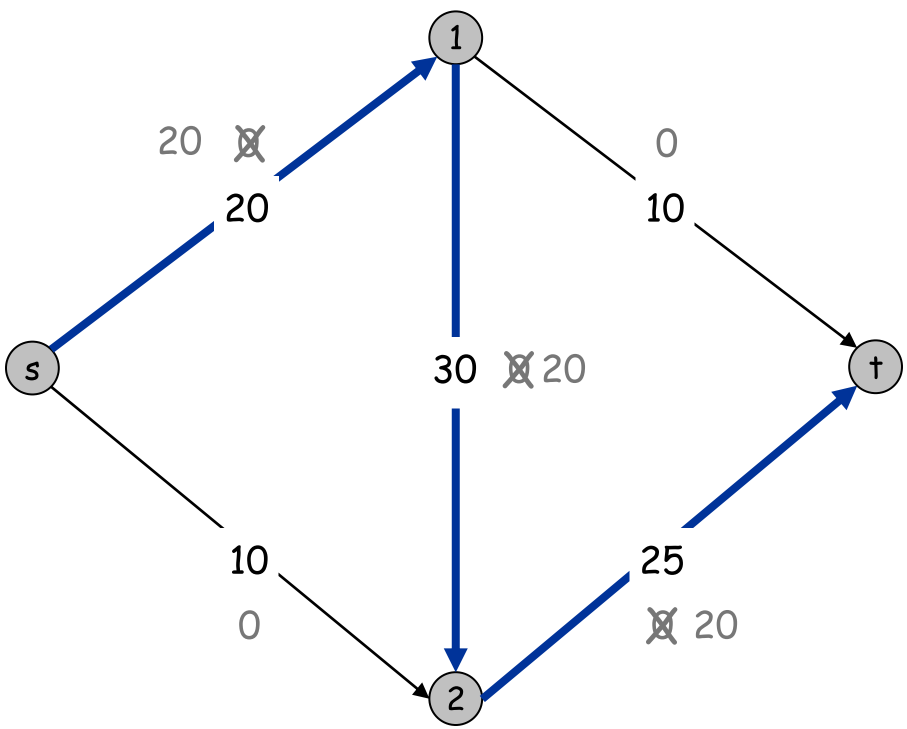


Flow value = 0

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.

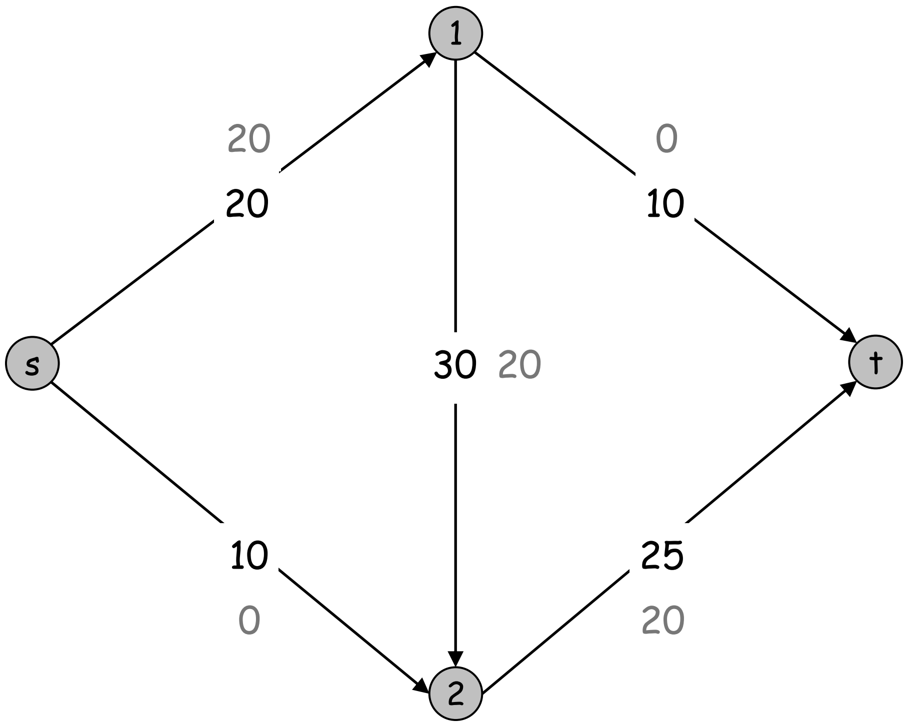


Flow value = 20

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.

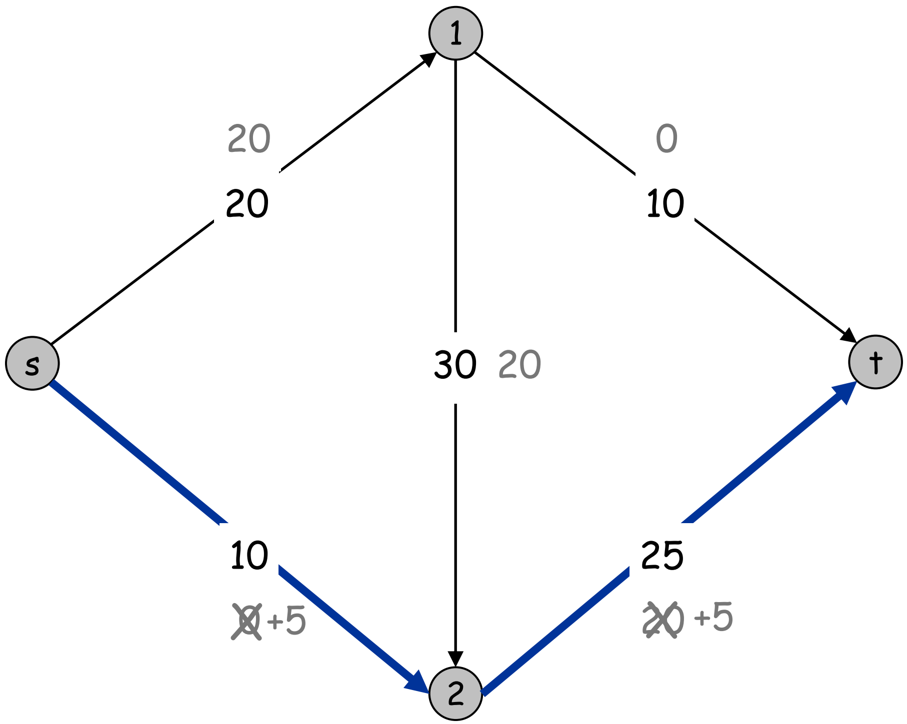


Flow value = 20

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.

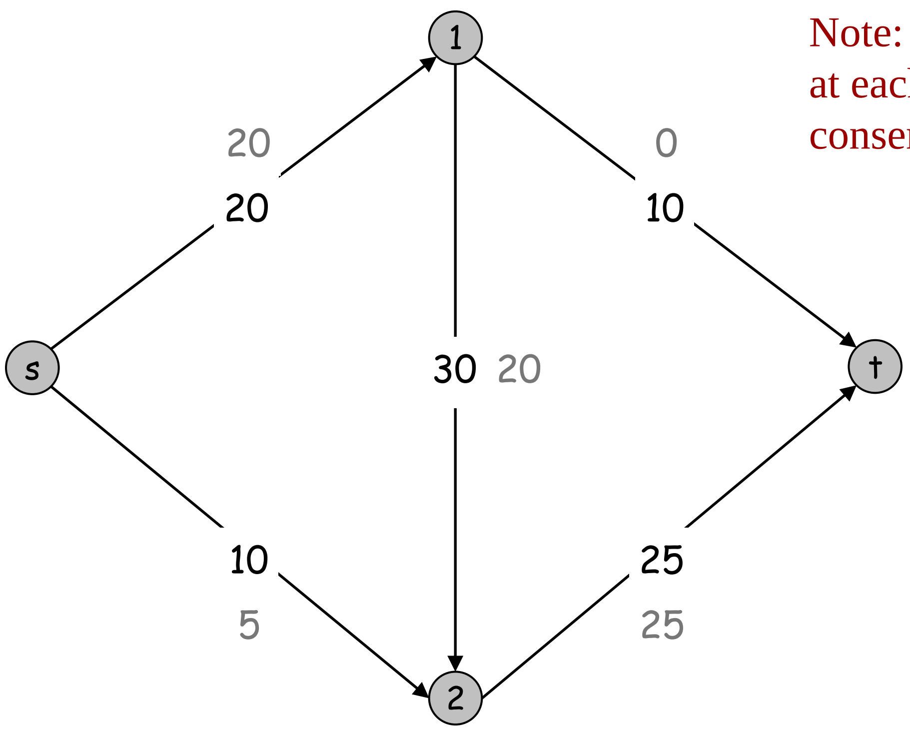


Flow value = 20

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.



Note: Adding flow along a *path* at each step maintains flow conservation at every vertex

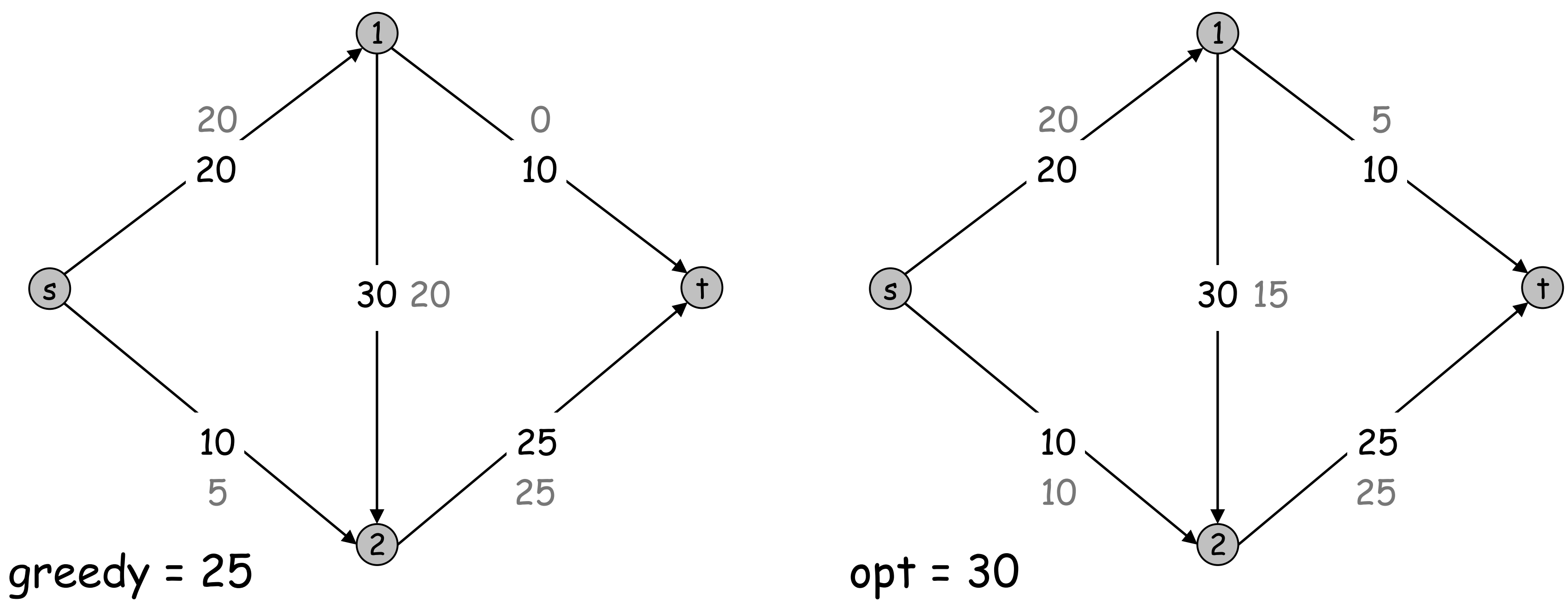
Flow value = 25

Towards a Max Flow Algorithm

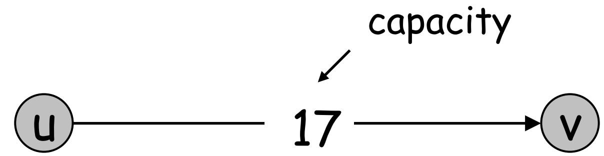
Greedy algorithm.

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

Doesn't Work:
local optimality $\neq$ global optimality

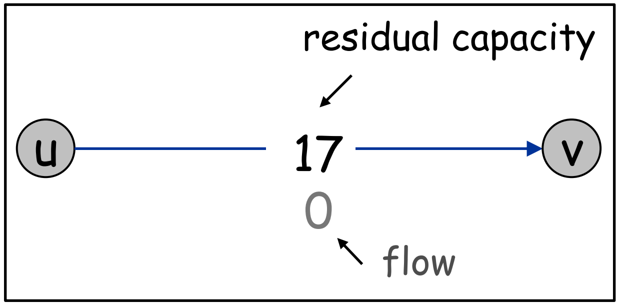


Residual Graph



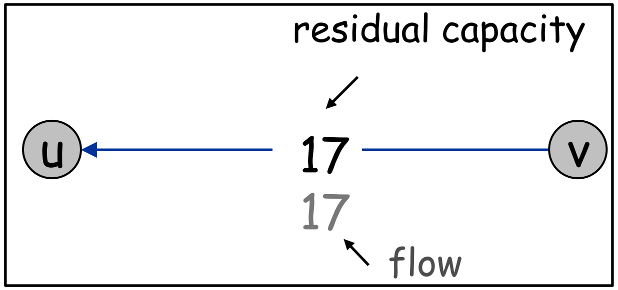
Original edge: $e = (u, v) \in E$.

- Flow $f(e)$, capacity $c(e)$.



Create (New) Residual edges:

a. If $f(u, v) = 0$, it has one residual edge (u, v) with residual capacity $c_f(u, v) = c(u, v)$

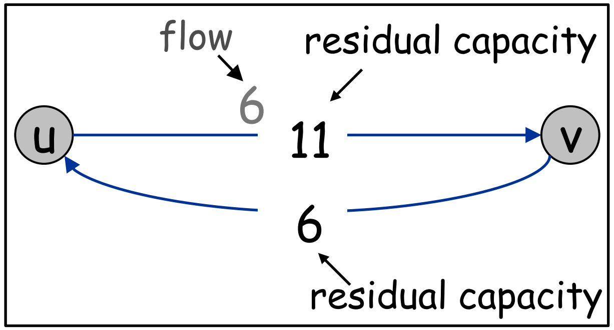


b. If $f(u, v) = c(u, v)$, it has one residual edge (v, u) with residual capacity $c_f(v, u) = f(u, v)$

c. If $0 < f(u, v) < c(u, v)$, it has two residual edges:

   i. (u, v) with $c_f(u, v) = c(u, v) - f(u, v)$

   ii. (v, u) with $c_f(v, u) = f(u, v)$



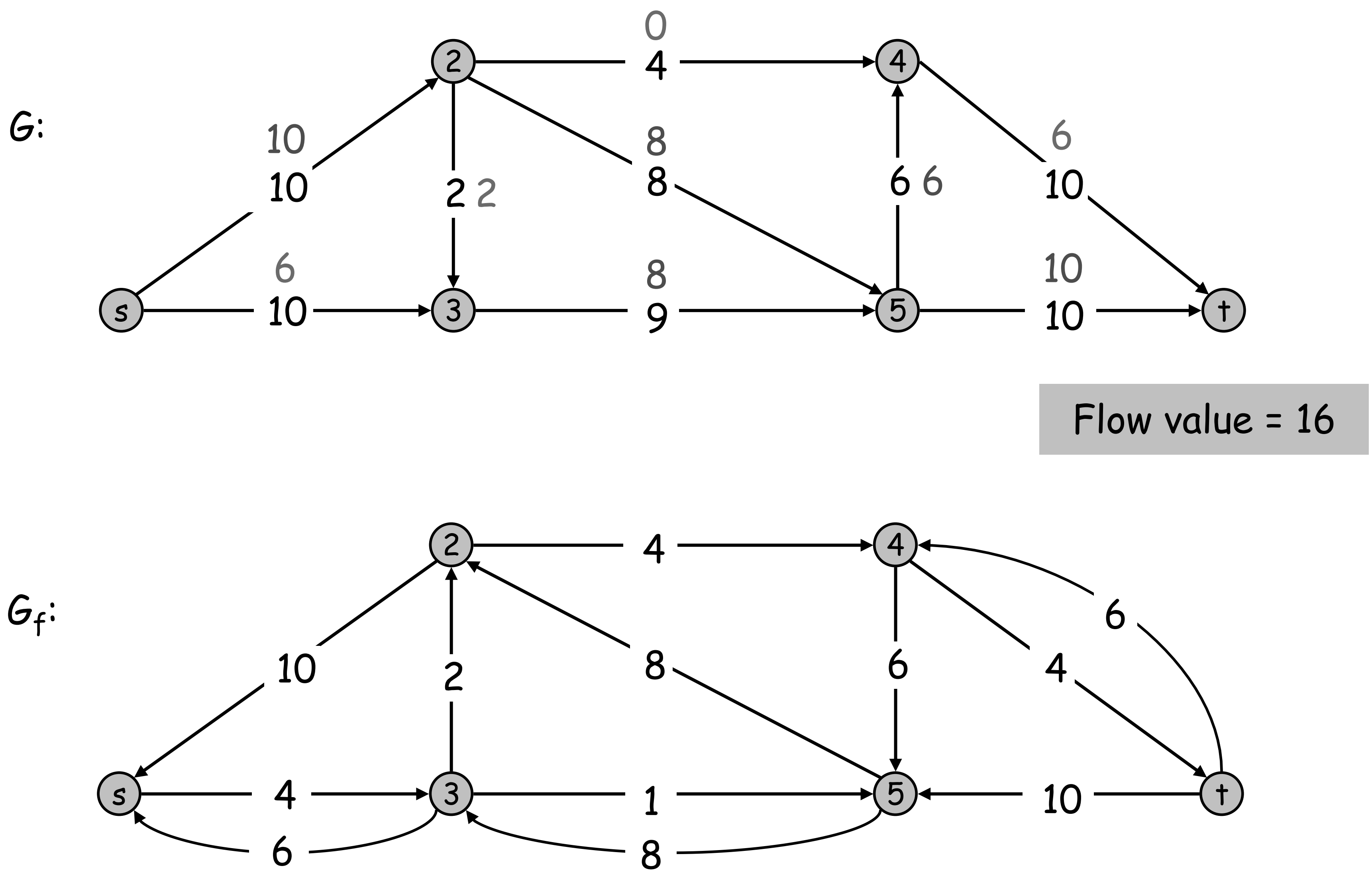
Residual graph: $G_f = (V, E_f)$.

- Vertices are the same vertices
- Edges are all the residual edges
- Residual capacity is "available remaining capacity"

A Graph G, flow f and associated residual Graph G_f

G:

G_f:



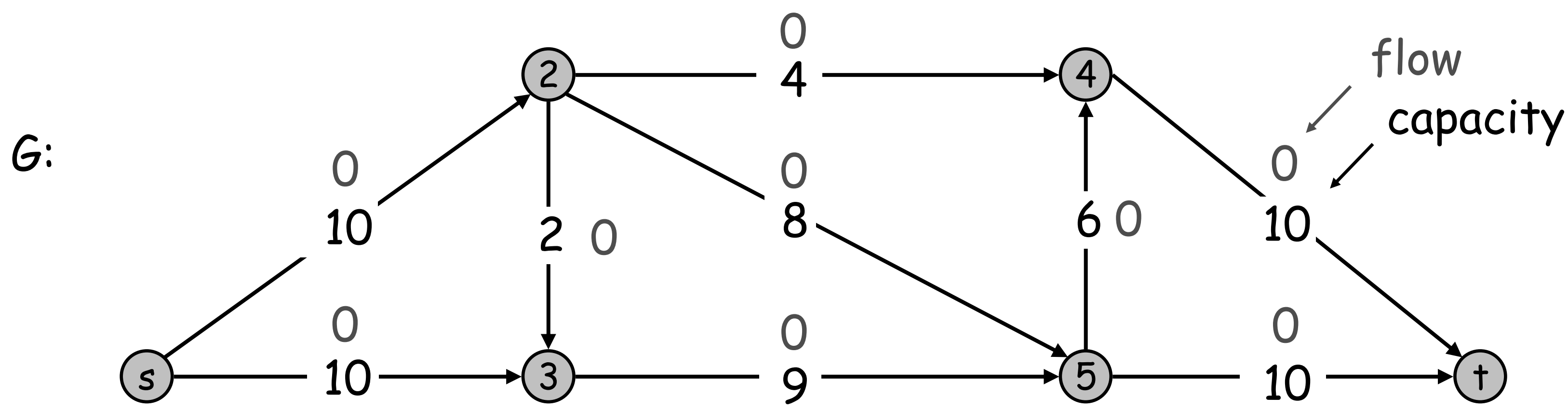
Ford-Fulkerson Algorithm

```
Ford-Fulkerson(G,s,t)
for each (u,v) ∈ E do
      f(u,v) ← 0
      c_f(u,v) ← c(e)
      c_f(v,u) ← 0
while there exists path p in residual graph G_f do
      c_f(p) ← min{c_f(e): e ∈ p}
      for each edge (u,v) ∈ p do
            if (u,v) ∈ E then
                  f(u,v) ← f(u,v) + c_f(p)
                  c_f(u,v) ← c_f(u,v) − c_f(p)
                  c_f(v,u) ← c_f(v,u) + c_f(p)
            else
                  f(v,u) ← f(v,u) − c_f(p)
                  c_f(v,u) ← c_f(v,u) + c_f(p)
                  c_f(u,v) ← c_f(u,v) − c_f(p)
```

Forward edge

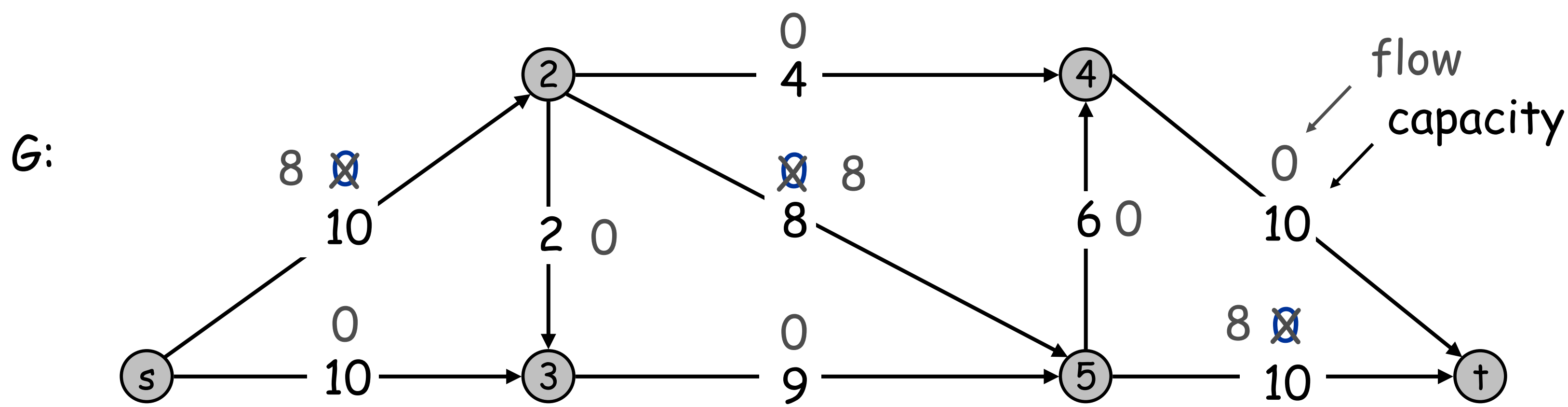
Backward edge

Ford-Fulkerson Algorithm

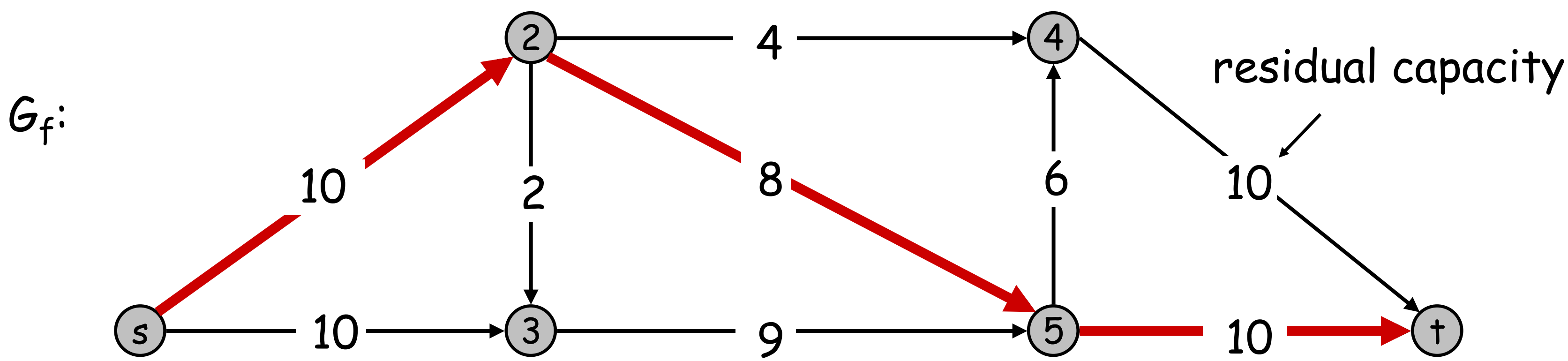


Flow value = 0

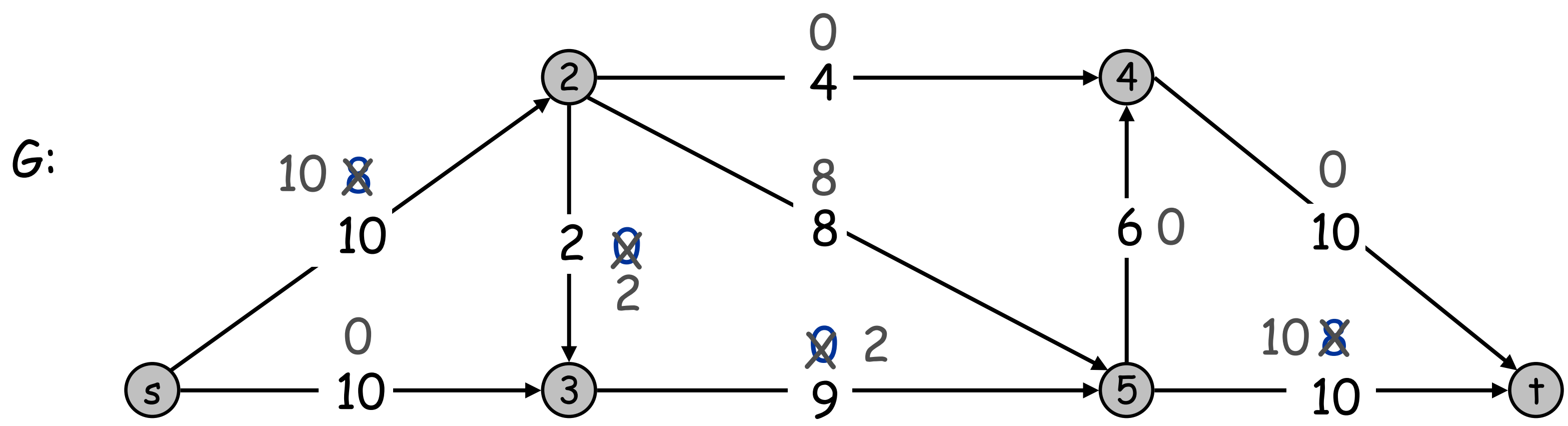
Ford-Fulkerson Algorithm



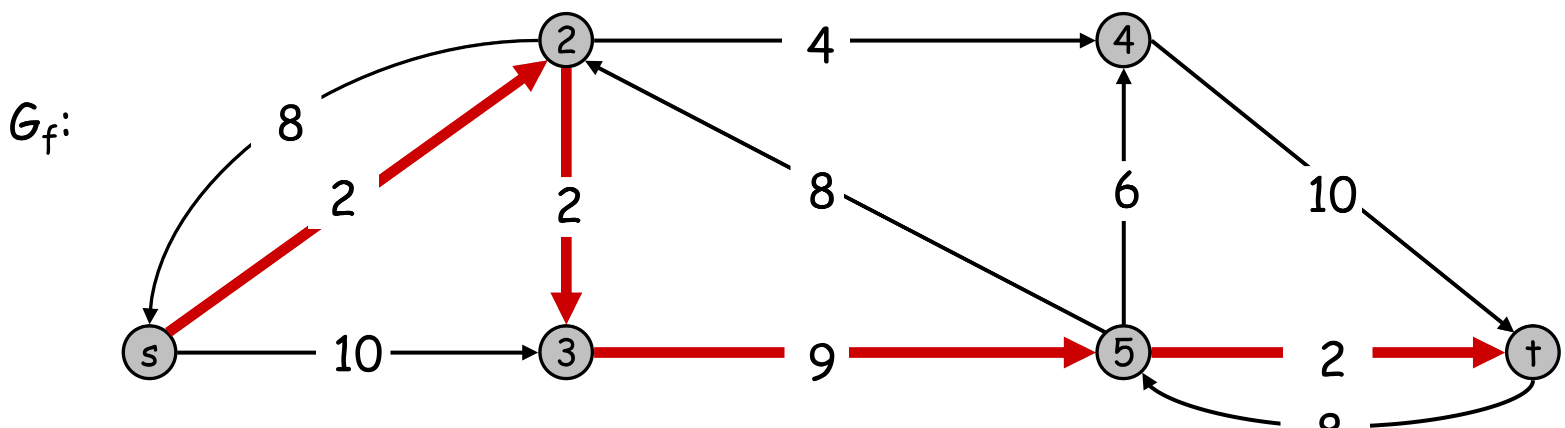
Flow value = 0



Ford-Fulkerson Algorithm

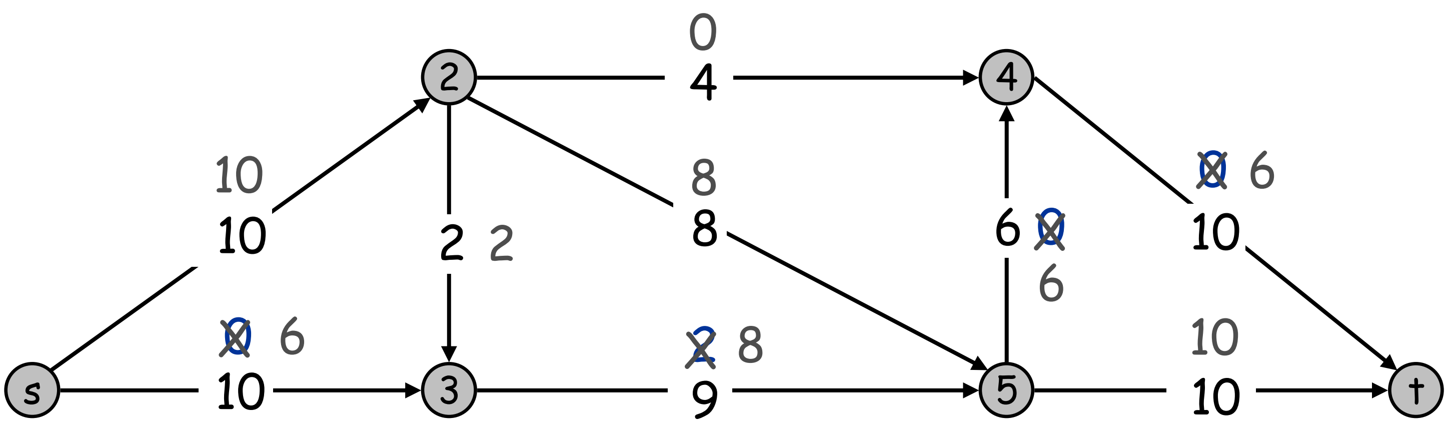


Flow value = 8



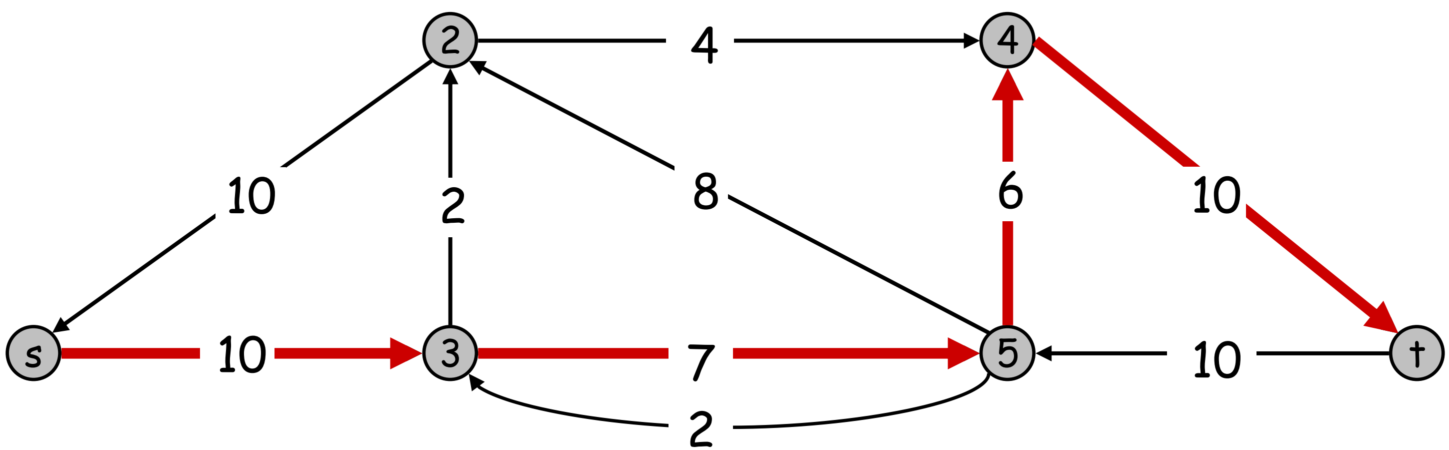
Ford-Fulkerson Algorithm

G:

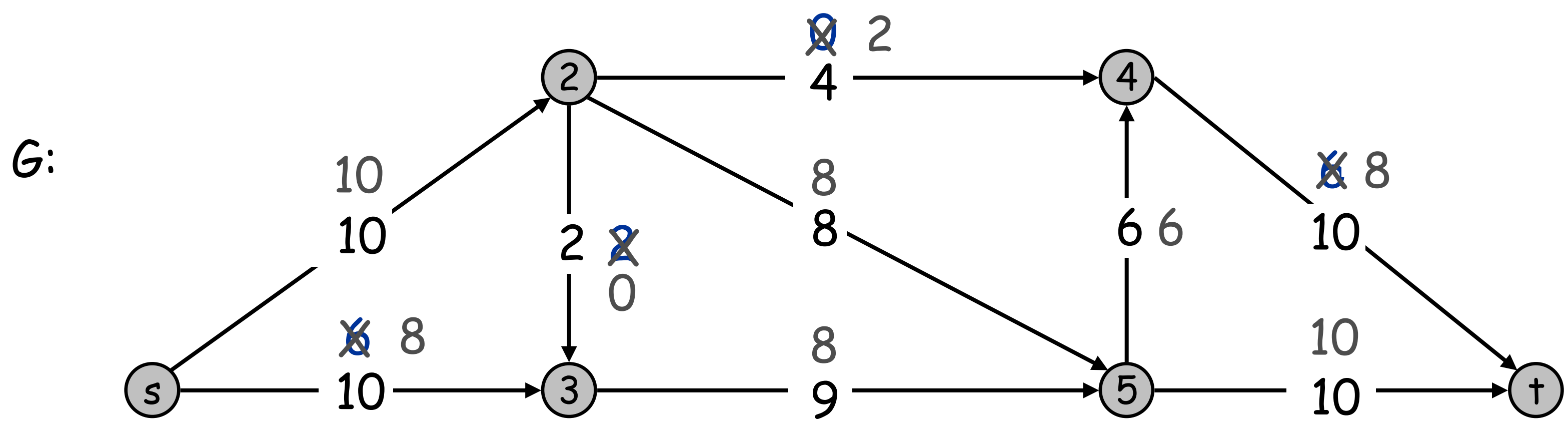


Flow value = 10

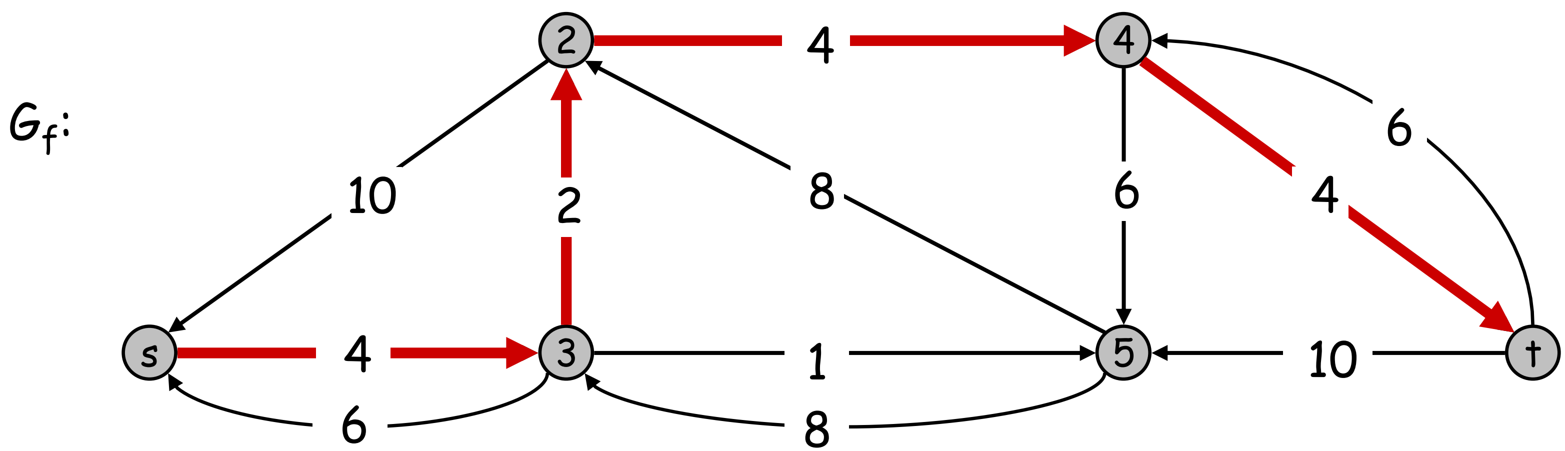
G_f:



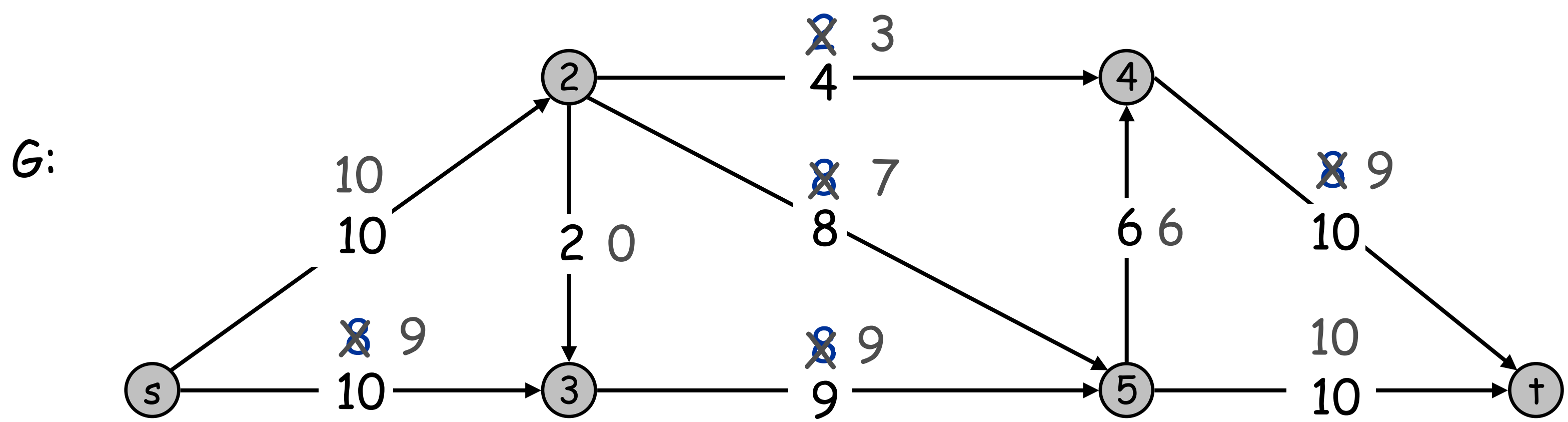
Ford-Fulkerson Algorithm



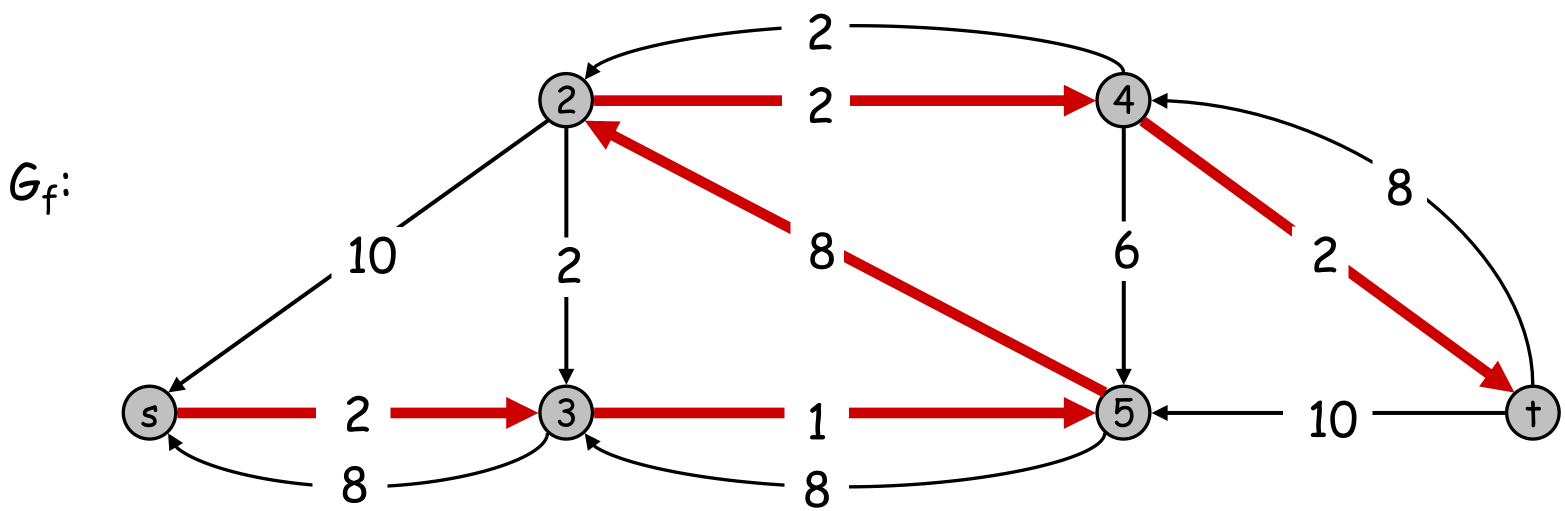
Flow value = 16



Ford-Fulkerson Algorithm

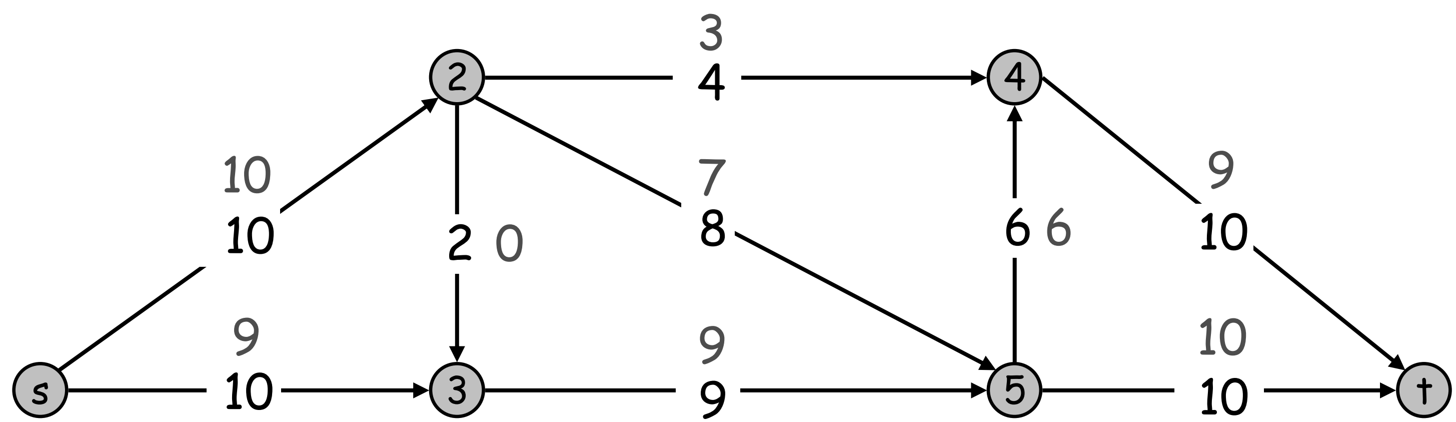


Flow value = 18



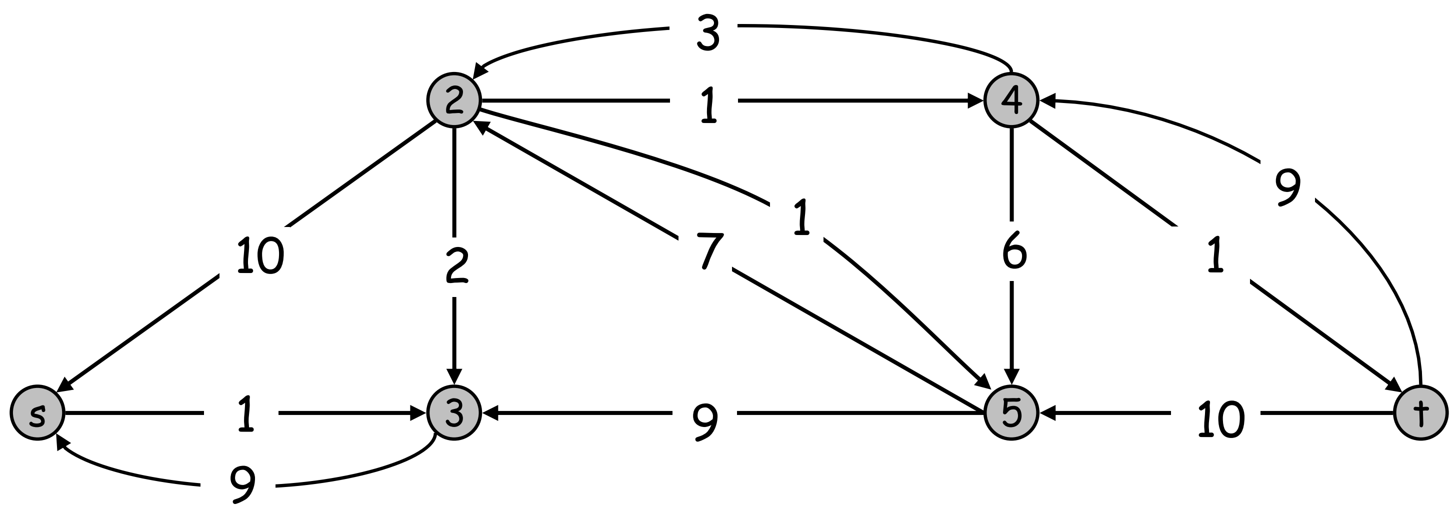
Ford-Fulkerson Algorithm

G:



Flow value = 19

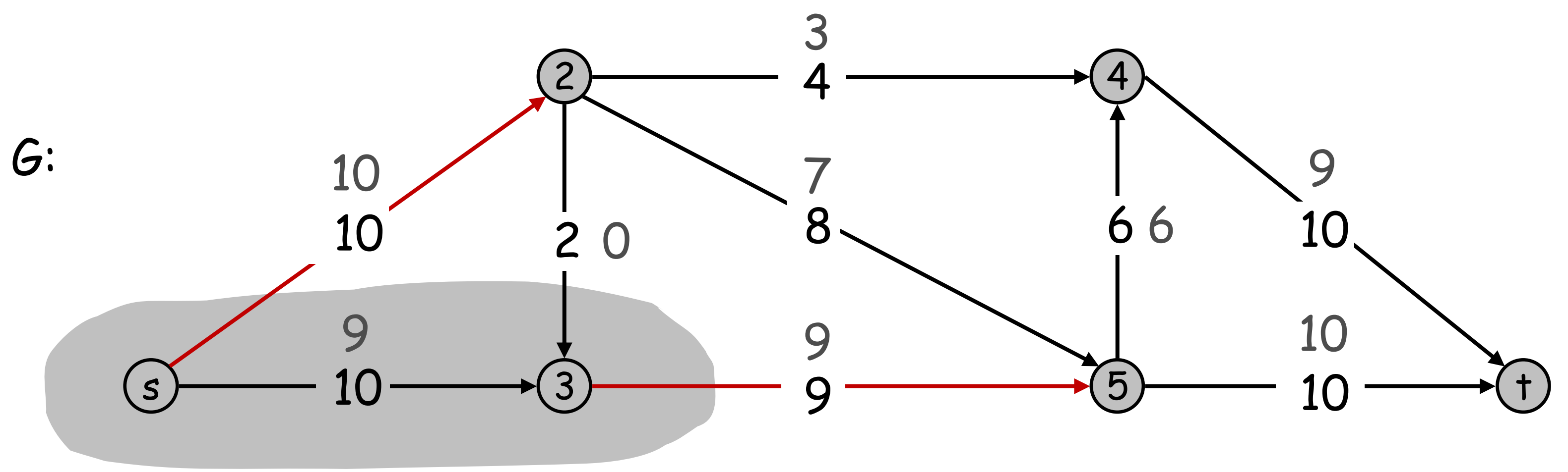
G_f:



No s-t path exists in G_f. Algorithm stops! Current flow is optimally maximal.

Ford-Fulkerson Algorithm

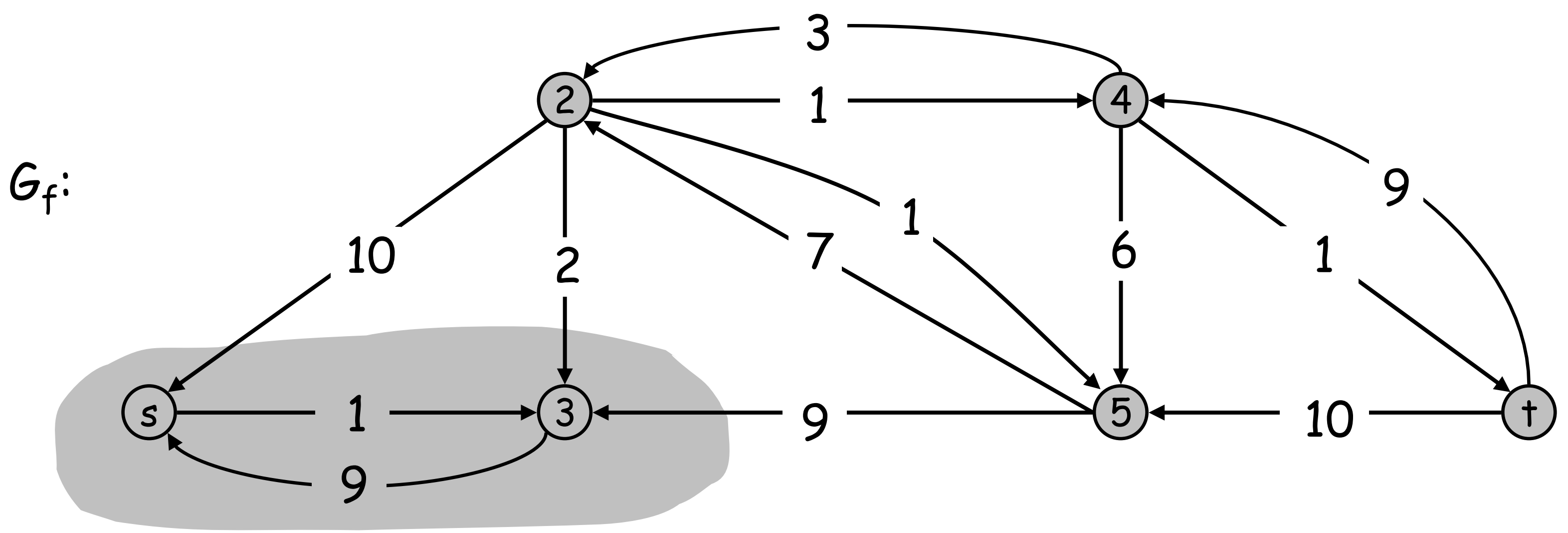
G:



Cut capacity = 19

Flow value = 19

G_f:

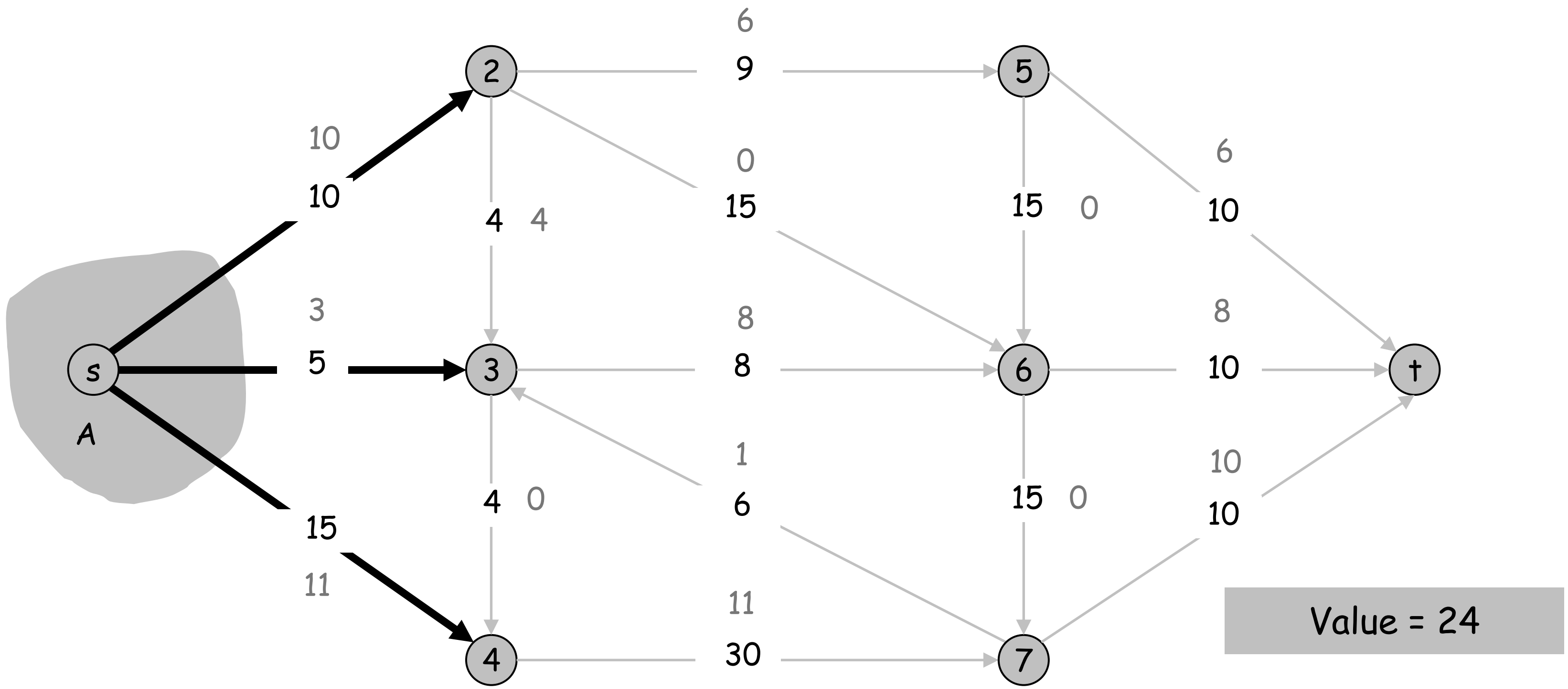


Net flow and cuts

Def: Let f be any flow, and let (S,T) be any s-t cut. Then, the net flow across the cut is

$$f(S,T) = \sum_{e \text{ from } S \text{ to } T} f(e) - \sum_{e \text{ from } T \text{ to } S} f(e)$$

Net flow lemma: For any s-t cut (S,T), $f(S,T) = |f|$.

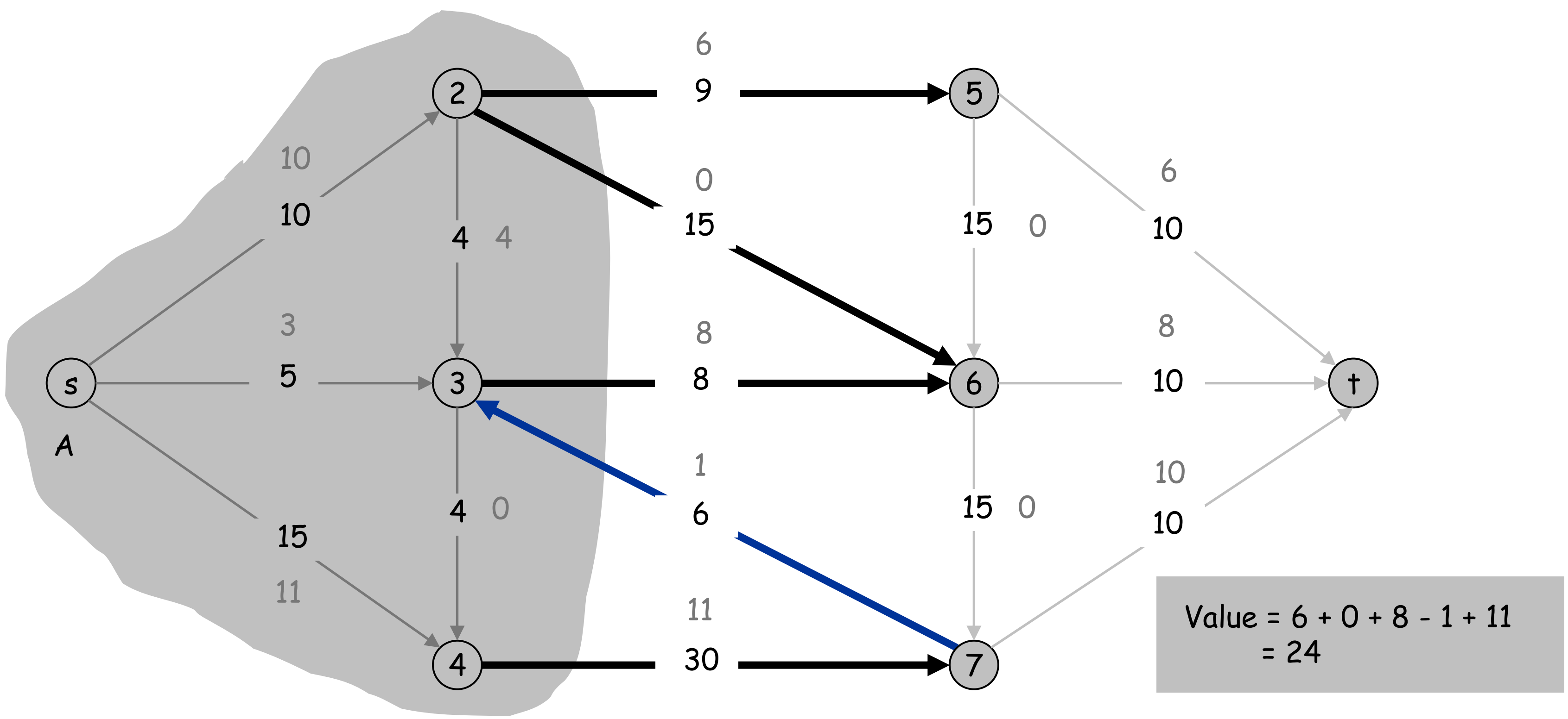


Net flow and cuts

Def: Let f be any flow, and let (S,T) be any s-t cut. Then, the net flow across the cut is

$$f(S,T) = \sum_{e \text{ from } S \text{ to } T} f(e) - \sum_{e \text{ from } T \text{ to } S} f(e)$$

Net flow lemma: For any s-t cut (S,T), $f(S,T) = |f|$.

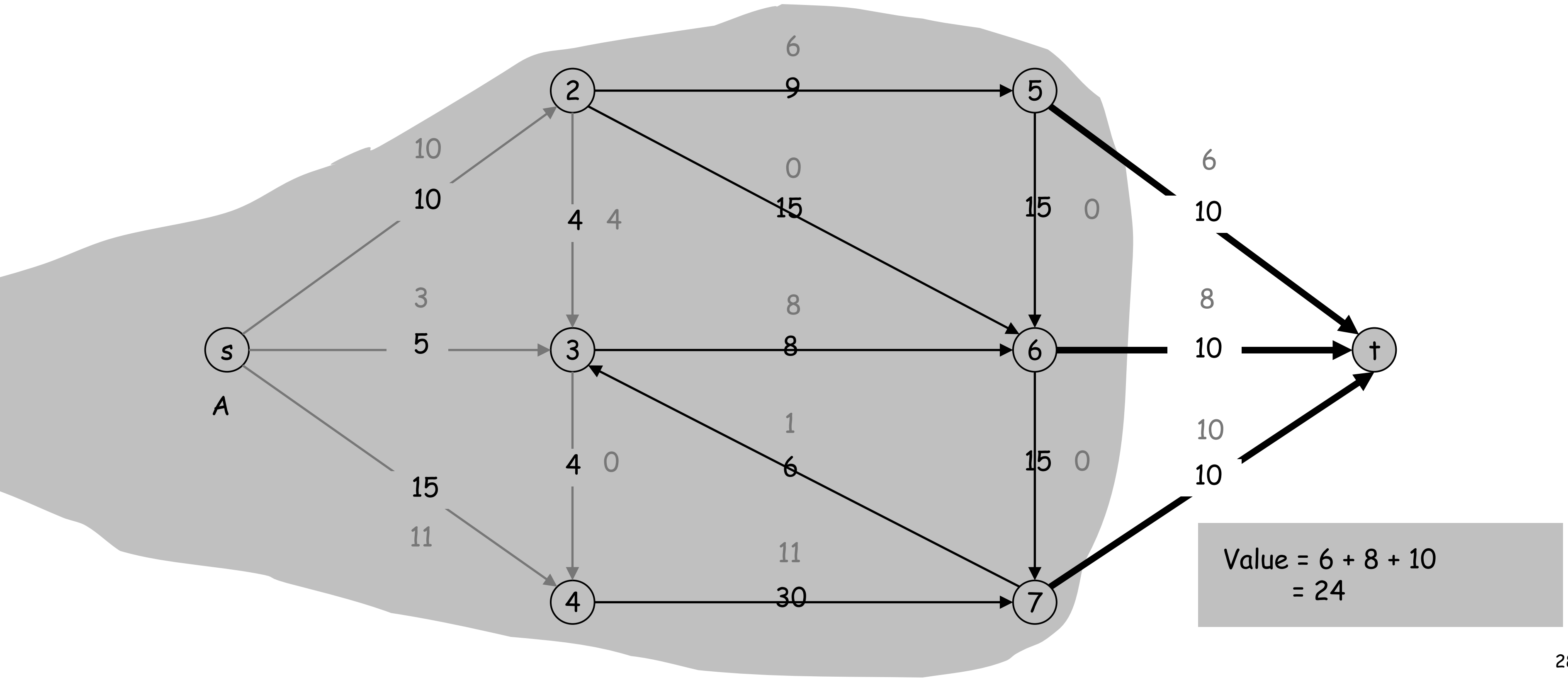


Net flow and cuts

Def: Let f be any flow, and let (S, T) be any s-t cut. Then, the net flow across the cut is

$$f(S,T) = \sum_{e \text{ from } S \text{ to } T} f(e) - \sum_{e \text{ from } T \text{ to } S} f(e)$$

Net flow lemma: For any s-t cut (S, T), $f(S,T) = |f|$.



Net flow and cuts

Net flow lemma: Let f be any flow, and let (S, T) be any s-t cut. Then,

$$f(S,T) = \sum_{e \text{ from } S \text{ to } T} f(e) - \sum_{e \text{ from } T \text{ to } S} f(e) = |f|$$

Proof:

$$\sum_{e \text{ out of } s} f(e) = |f| \qquad (1)$$

By flow conservation, for any vertex $v \in V - \{s, t\}$,

$$\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) = 0 \qquad (2)$$

Sum (2) over all $v \in S - \{s\}$, together with (1). We see that

- For every edge e inside S, both $f(e)$ and $-f(e)$ appear
- For every edge e from S to T, only $f(e)$ appear
- For every edge e from T to S, only $-f(e)$ appear

Lemma is thus proved!

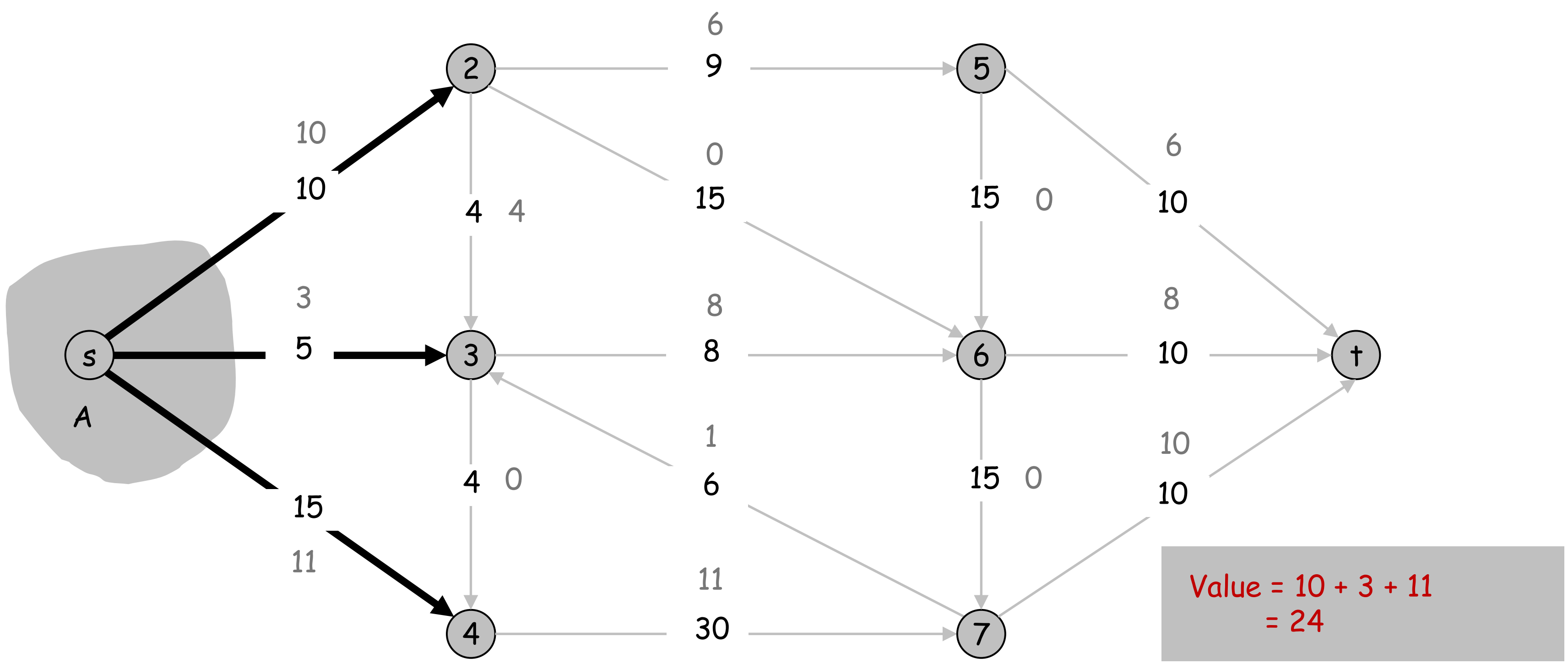
How Can we prove flows optimal (maximal)?

We just saw tools for calculating the value of a flow.

Given flow, how can we prove that the flow is optimal, or can it be improved?

For example, the flow below has value=24.

This can be improved to have value=28



How Can we prove flows optimal (maximal)?

We just saw tools for calculating the value of a flow.

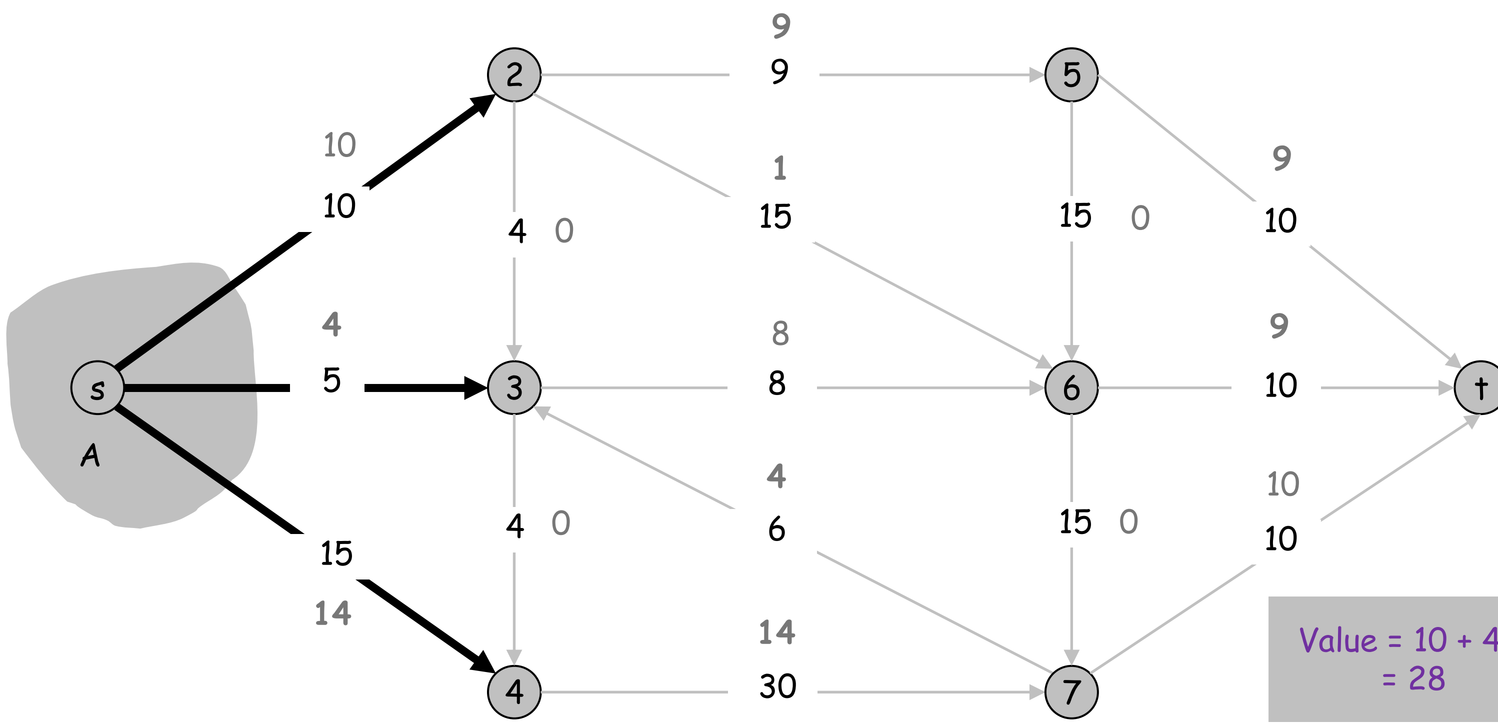
Given flow, how can we prove that the flow is optimal, i.e., can it be improved?

For example, the flow below has value=24.

This can be improved to have value=28

Is this the best?

How can we be sure?



Value = 10 + 4 + 14
= 28

Flow and Cuts

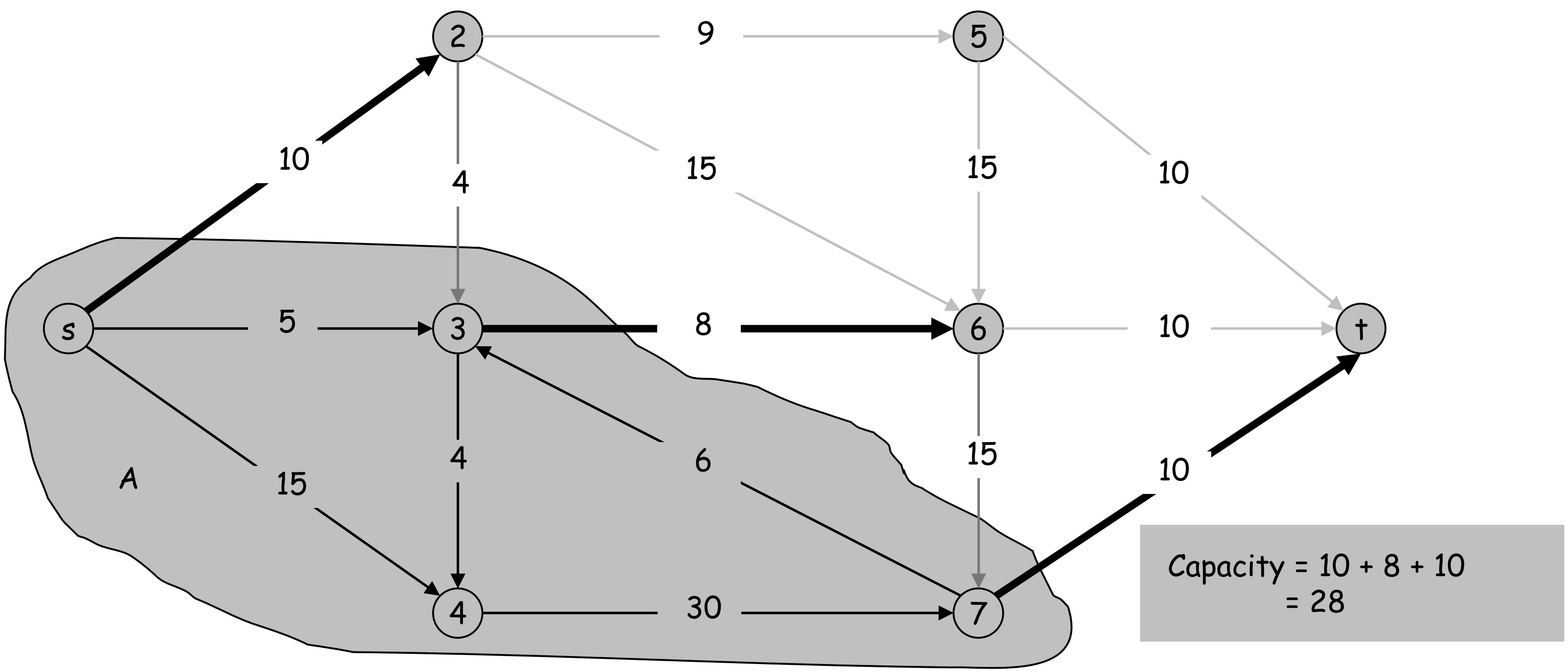
Def: The **capacity** of the cut (S,T) is $c(S,T) = \sum_{e \text{ from } S \text{ to } T} c(e)$

Claim: For any flow f and any s-t cut (S,T), $|f| \leq c(S,T)$.

Proof:

$$|f| = \sum_{e \text{ from } S \text{ to } T} f(e) - \sum_{e \text{ from } T \text{ to } S} f(e)$$

$$\leq \sum_{e \text{ from } S \text{ to } T} f(e) \leq \sum_{e \text{ from } S \text{ to } T} c(e) = c(S,T)$$



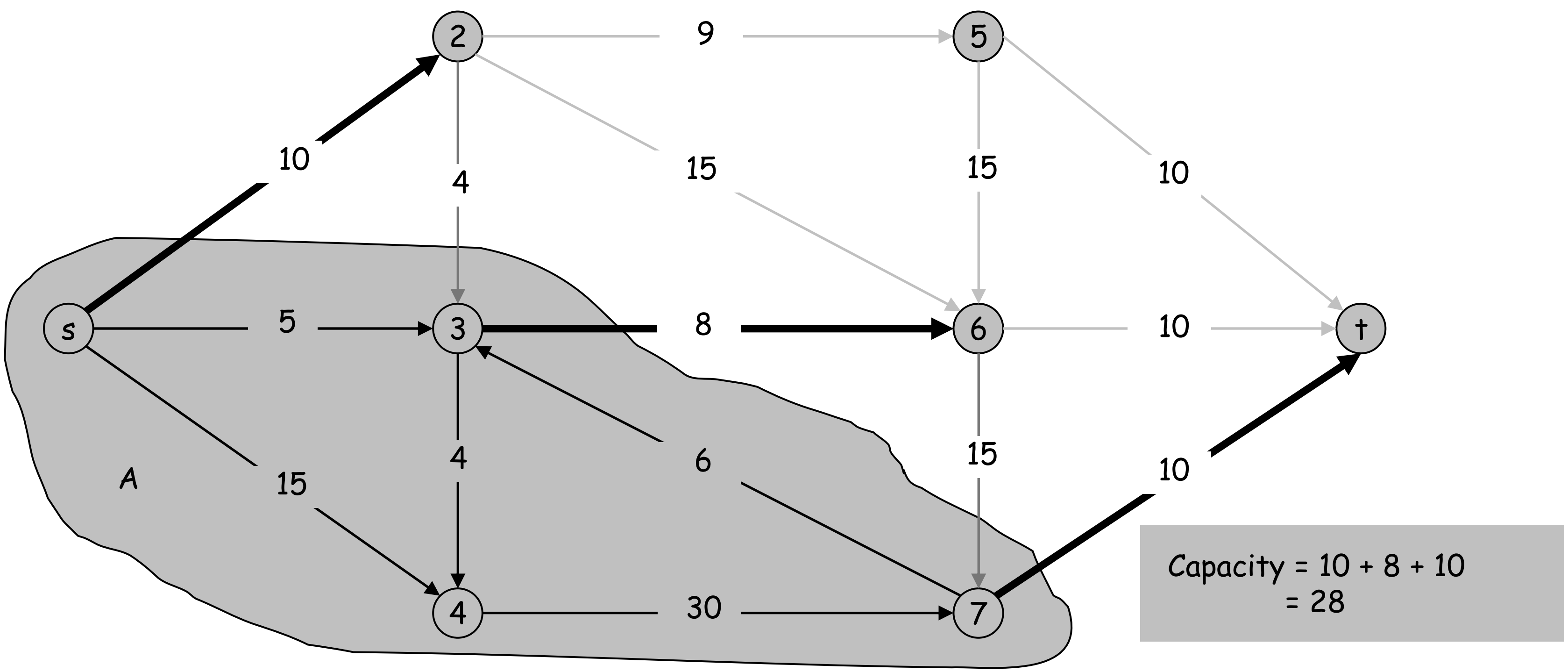
Flow and Cuts

Def: The capacity of the cut (S, T) is $c(S,T) = \sum_{e \text{ from } S \text{ to } T} c(e)$

Claim: For any flow f and any s-t cut (S, T), $|f| \leq c(S,T)$.

Example Usage: A few pages ago, we found a flow with value $|f| = 28$ for the graph below. The cut (S, T), with $S = \{s, 3,4,7\}$ has $c(S,T) = 28$ so that flow is maximum, since no flow can be better!

We now make this into a theorem!



Correctness of Ford-Fulkerson Algorithm

Max-Flow min-cut theorem: Let f be any flow.
Then the following three statements are equivalent:

(1) f is a maximum flow.

(2) The residual graph G_f has no path from s to t.

(3) $|f| = c(S,T)$ for some s-t cut (S,T).

Proof: (1) ⇒ (2), or ¬(2) ⇒ ¬(1): If there is a path in G_f, we can improve f.

(2) ⇒ (3):

- Need to find an s-t cut (S,T) such that $|f| = c(S,T)$
- By net flow lemma, $|f| = f(S,T)$, so must find a cut such that
  a) all edges e from S to T are full, i.e., $f(e) = c(e)$
  b) all edges e from T to S are empty, i.e., $f(e) = 0$
- Consider S = set of all nodes reachable from s in G_f.
- S cannot include t due to (2), so it is a valid s-t cut
- And this cut must meet the two conditions above

(3) ⇒ (1): By the claim from last page.

Ford-Fulkerson: Running time analysis

Q: Which path to choose in the residual graph?

A: Ford-Fulkerson doesn't specify.

- The choice does not affect correctness
- But it does affect running time
- Note that one iteration of the loop can find one augmenting path in $O(E)$ time using BFS or DFS so full run-time of FF is $O(\#\ iterations \cdot\ E)$

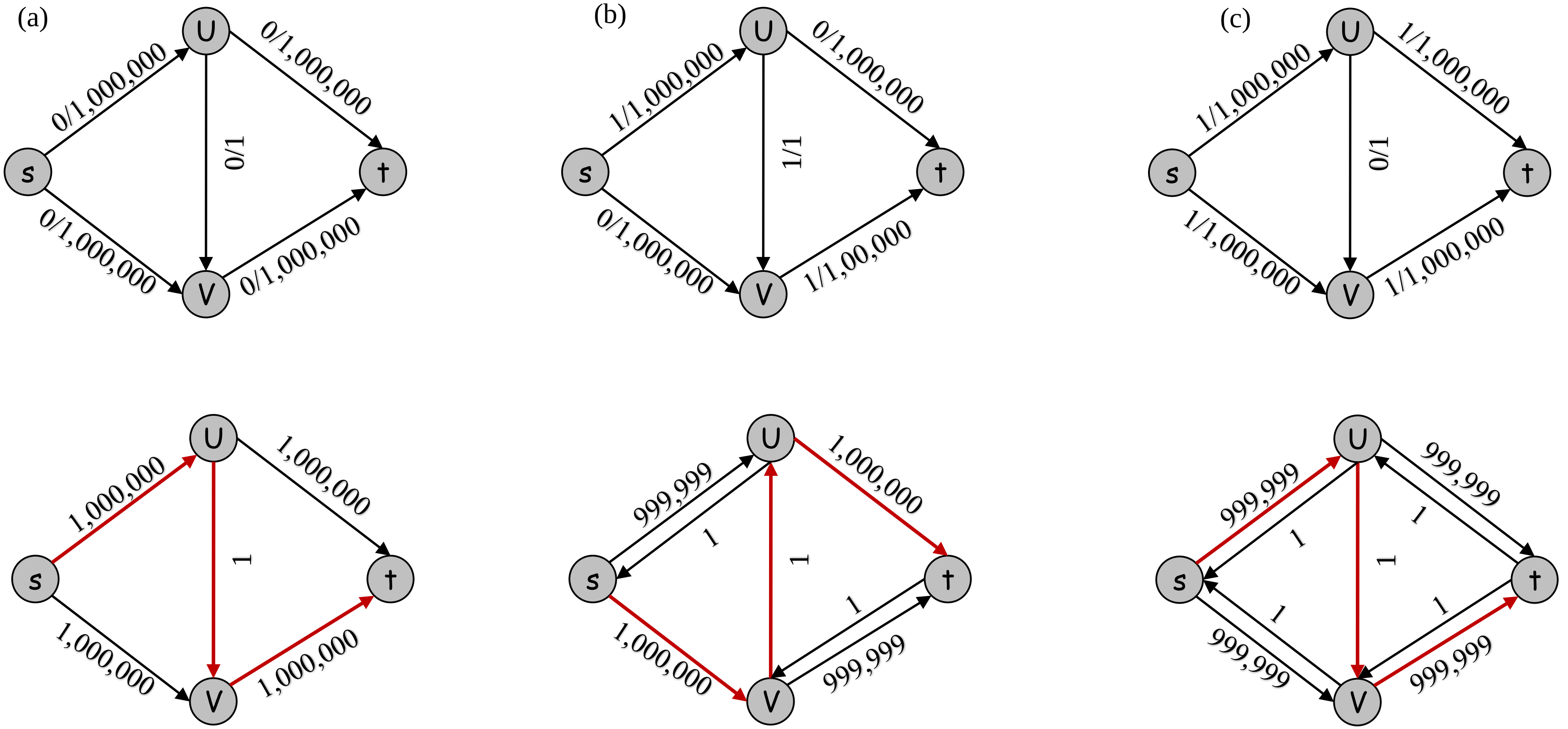
Claim: When all capacities are integers, Ford-Fulkerson takes at most $|f^*|$ iterations, where f^* is a maximum flow.

Proof: Each iteration increases $|f|$ by at least 1.

Integrality property: if all edge capacities are integers, then there exists a max flow for which every flow value is an integer and the F-F algorithm constructs such a flow.

Proof: The flow created by F-F is an integral flow since all (residual) capacities created are integral, so all changes to flows are additions/subtractions of integers.

Bad example



This up/down process will continue, adding only 1 unit of flow per augmenting path.
The final algorithm will require 1,000,000 augmenting steps!
If we had chosen s,u,t as first augmenting path, algorithm only uses 2 steps!

When capacities are irrational numbers, the algorithm might never terminate!

Edmonds-Karp: Choosing the shortest augmenting path

Idea: Choose the shortest (in terms of # edges) path in residual graph

- Can be done in $O(E)$ time using BFS.

Theorem: If we always choose the shortest path in the residual graph to augment the flow, then the Ford-Fulkerson algorithm terminates in $O(VE)$ iterations.

Proof: See textbook (not required).

Corollary: The Ford-Fulkerson algorithm can be implemented to run in $O(VE^2)$ time.

More advanced algorithms

- Push-relabel algorithms, $O(V^2E)$ time, and perform well in practice (see textbook for details)
- Theoretically better algorithms
  - $O(VE)$ [King, Rao, Tarjan, 1994] [Orlin, 2013]
  - $O\left(E^{1+o(1)} \log U\right)$ [Chen, Kyng, Liu, Peng, Gutenberg and Sachdeva, 2022]
    U: largest edge capacity (assuming integer capacities)