Lecture 18: Maximum Flow

Version of April 15, 2017

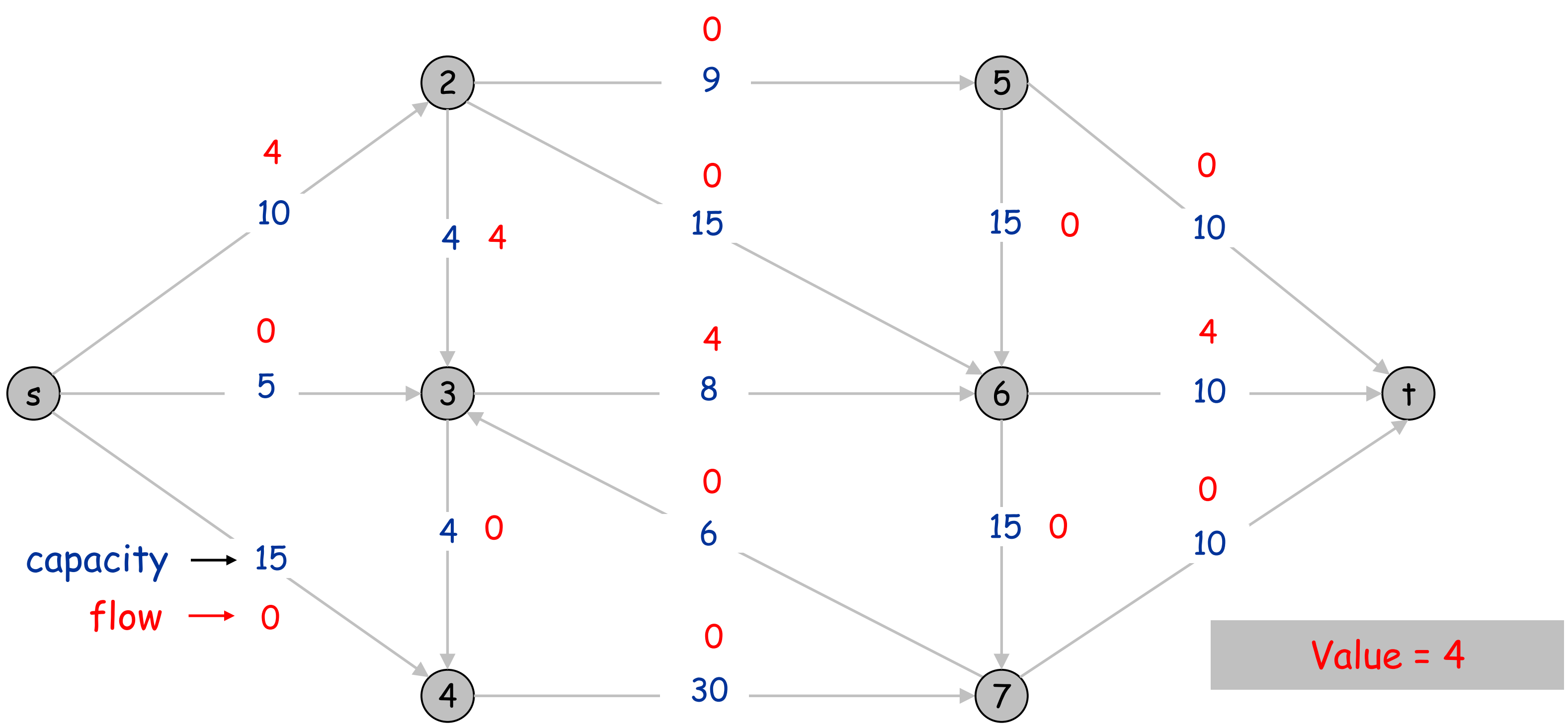
Flow

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** $\boldsymbol{v \in V - \{s, t\}}$, $\sum_{\boldsymbol{e} \text{ out of } \boldsymbol{v}} \boldsymbol{f(e)} = \sum_{\boldsymbol{e} \text{ into } \boldsymbol{v}} \boldsymbol{f(e)}$ (conservation)



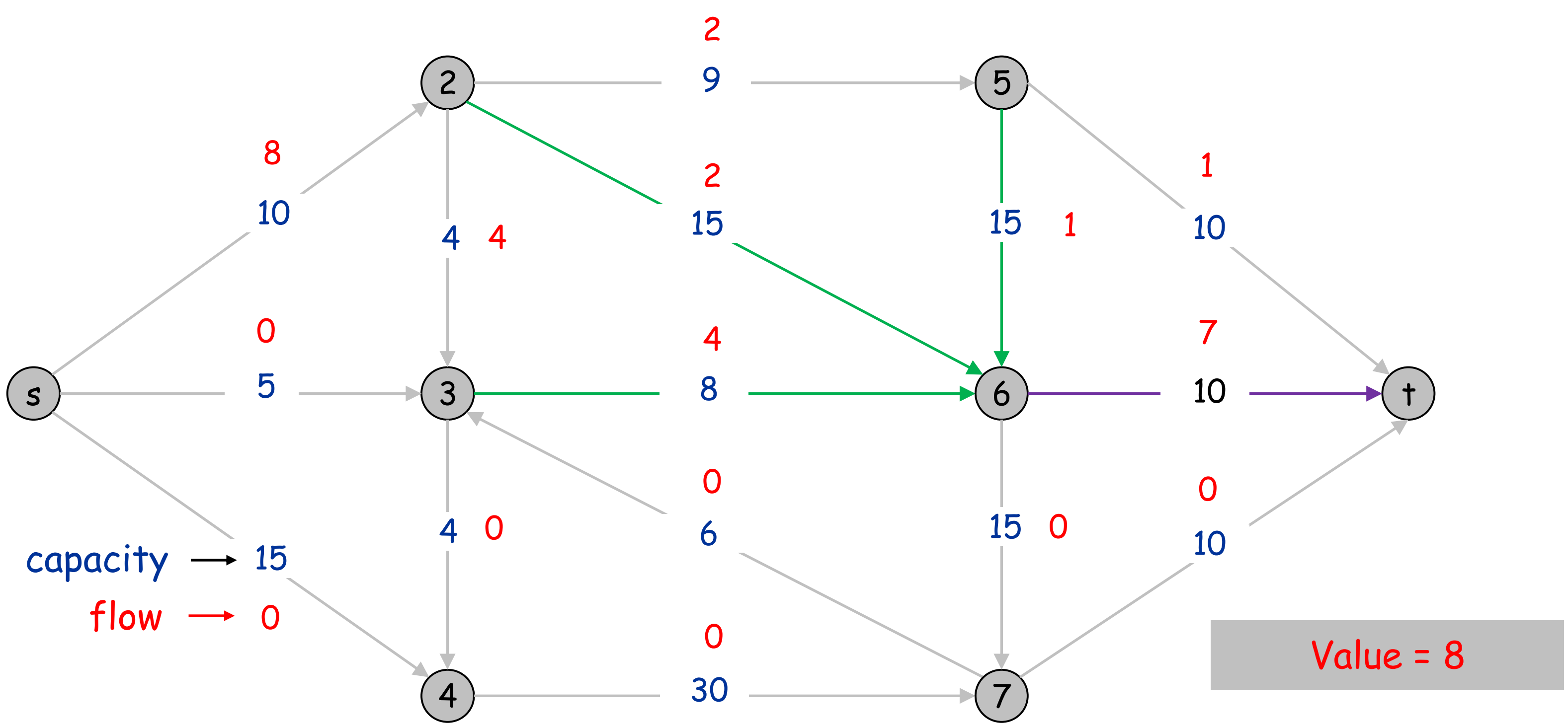
Flow

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



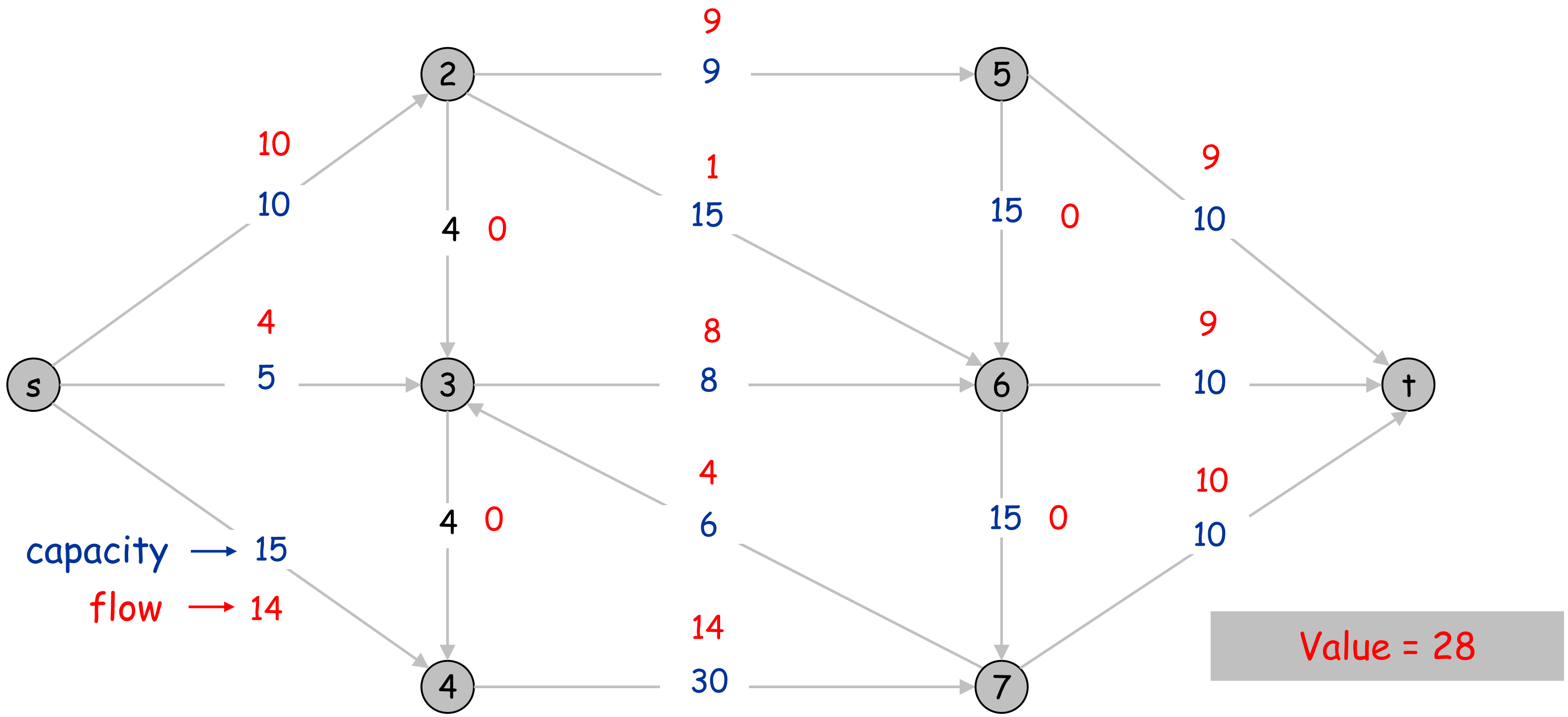
Maximum Flow

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?



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)
- …

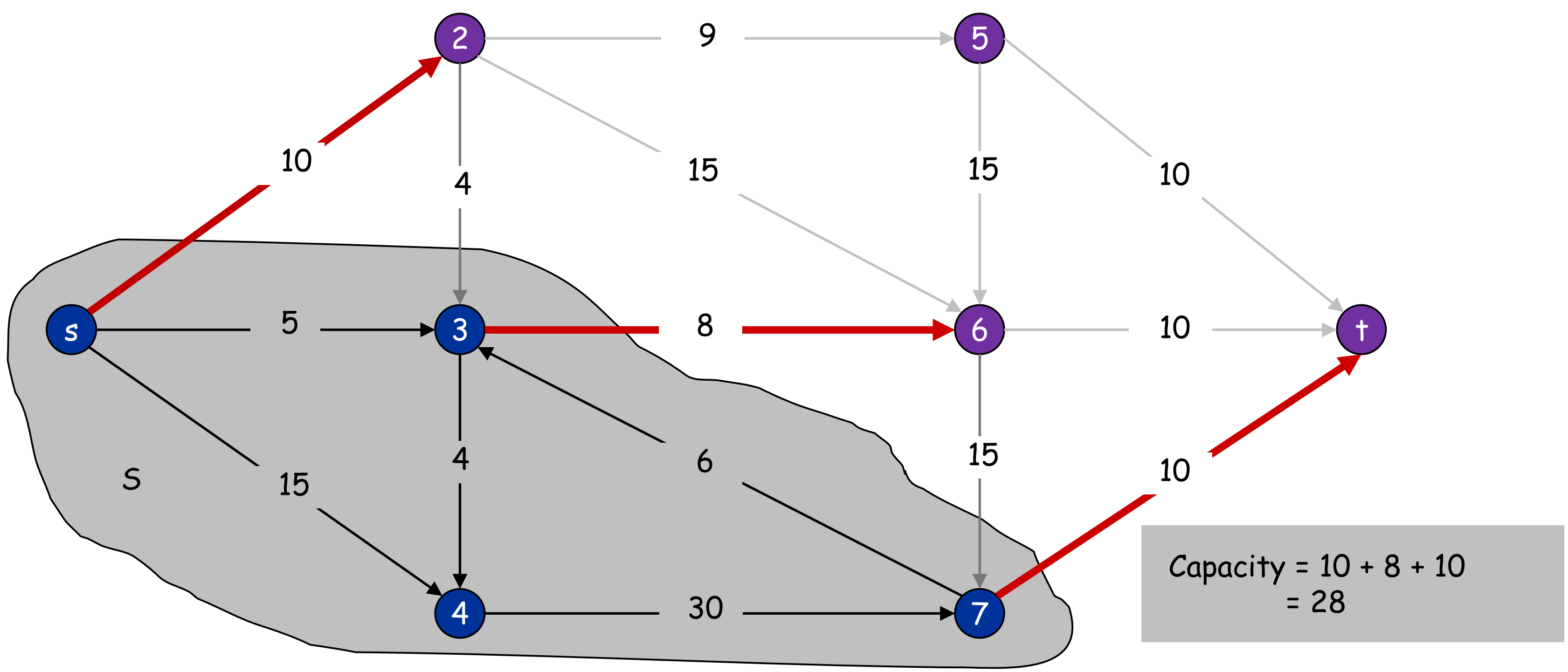
s-t Cut

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.

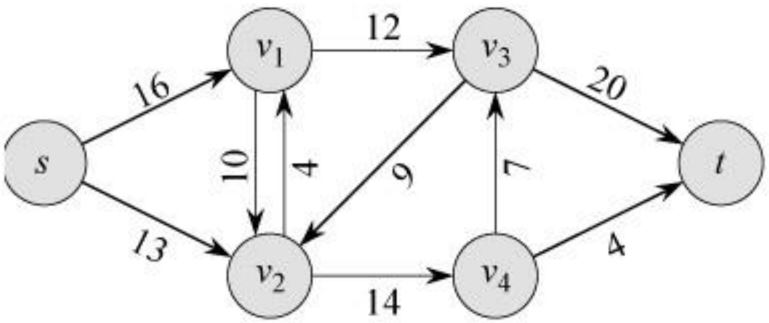
Observation: An s-t cut with capacity matching the value of a flow is a "proof" that the flow is a max flow.



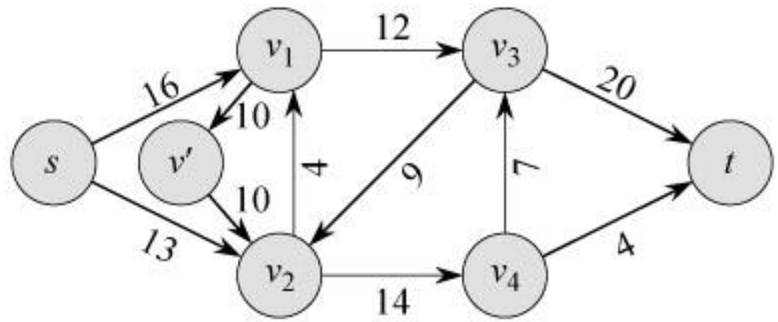
Assumptions

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)

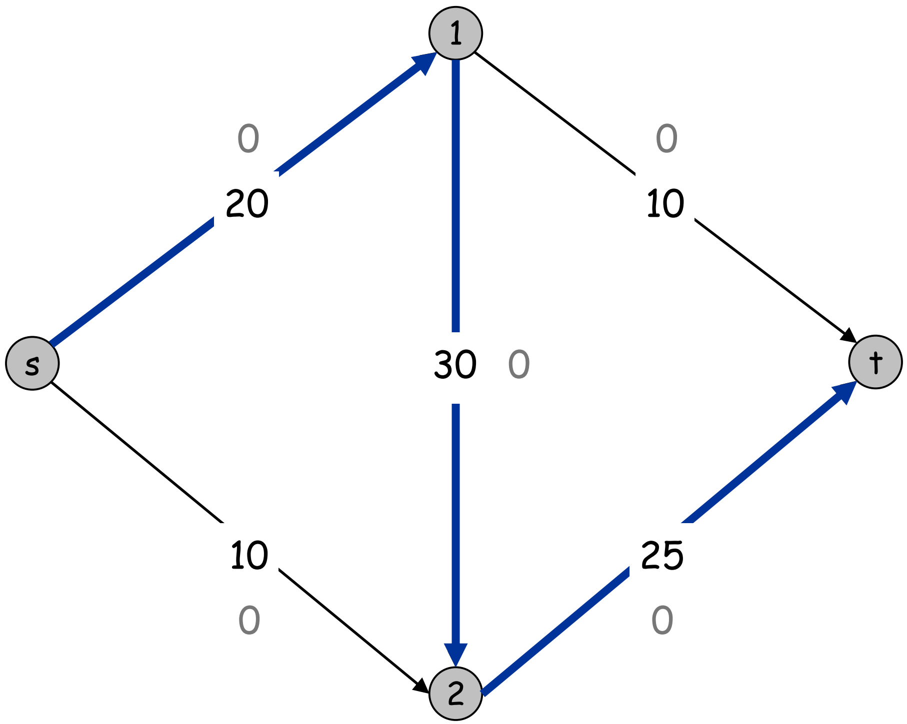
Also assume

- 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.

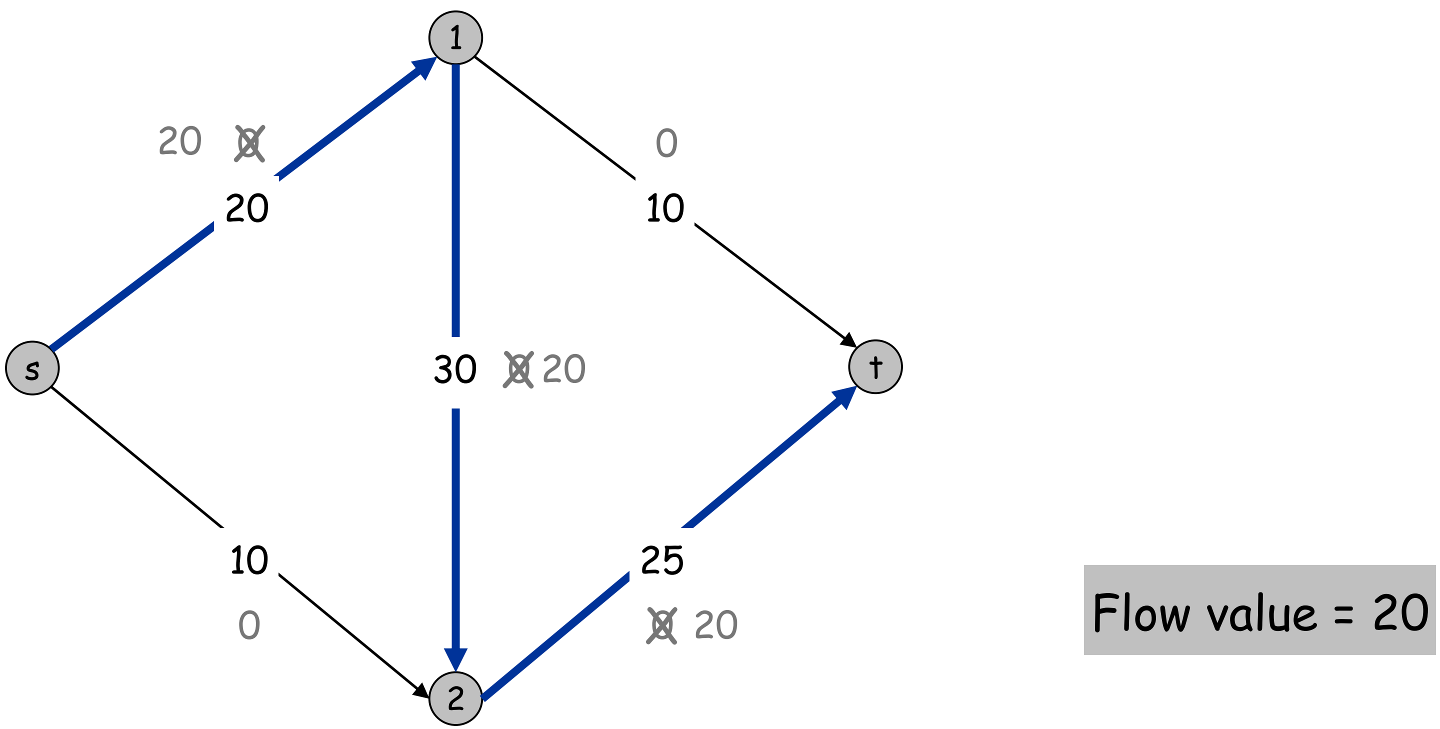


Flow value = 0

Towards a Max Flow Algorithm

Greedy algorithm.

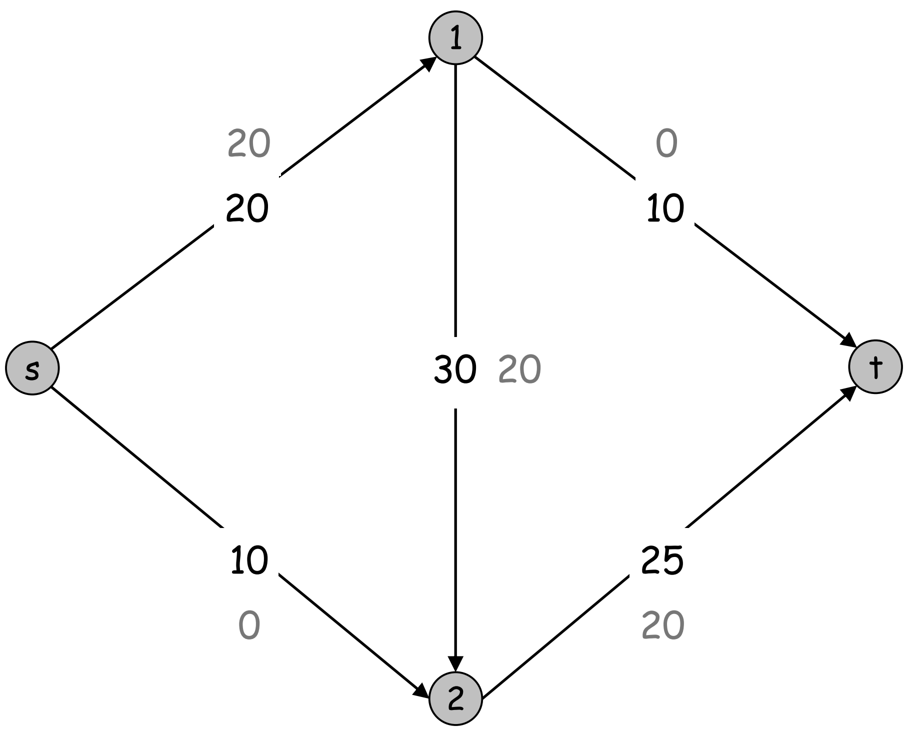
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s-t path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



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.

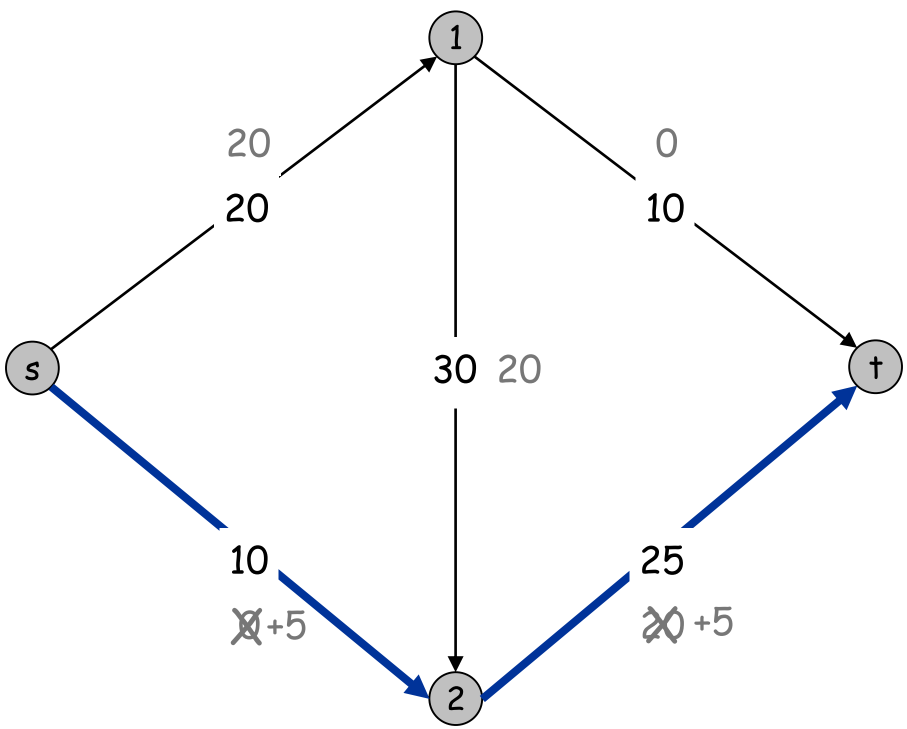


Flow value = 20

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.

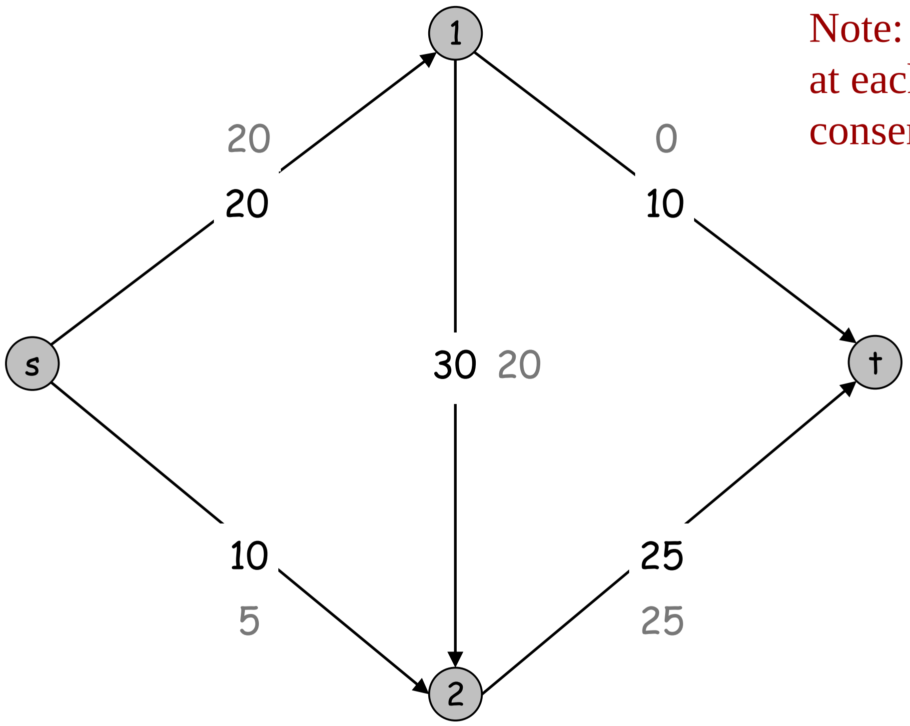


Flow value = 20

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

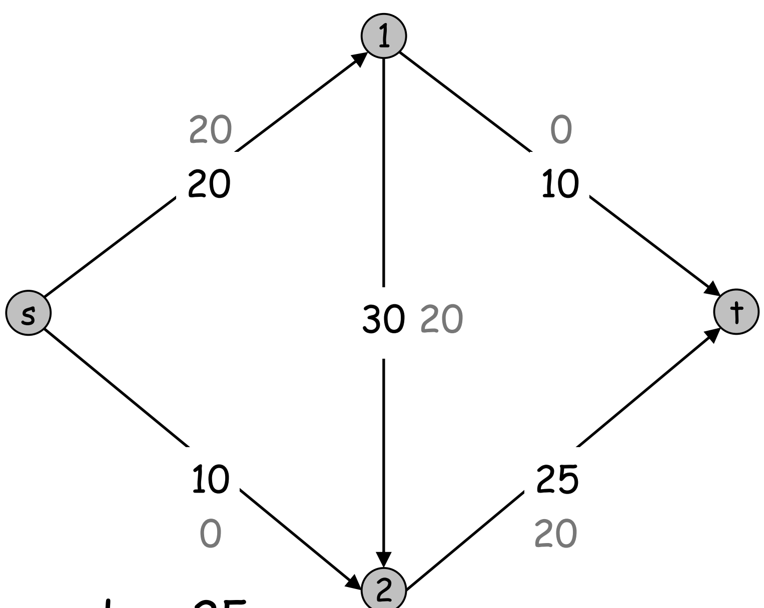
Flow value = 25

Towards a Max Flow Algorithm

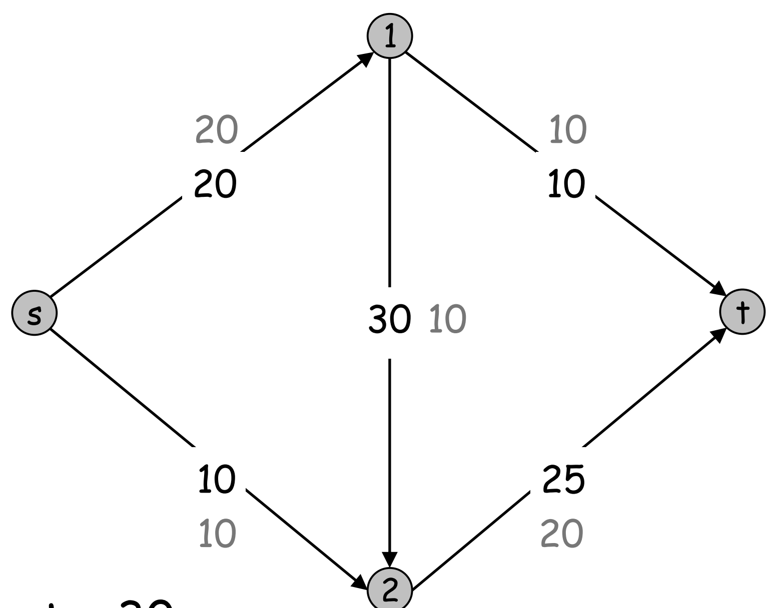
Greedy algorithm.

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

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

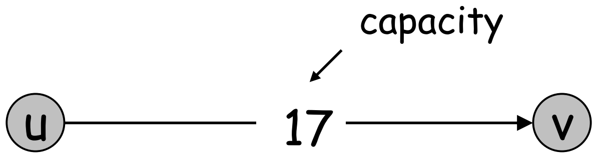


greedy = 25



opt = 30

Residual Graph

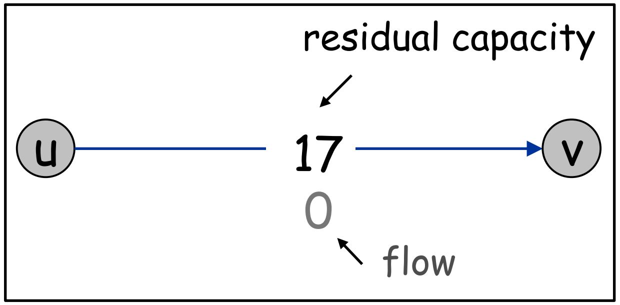


Original edge: $e = (u, v) \in E$.

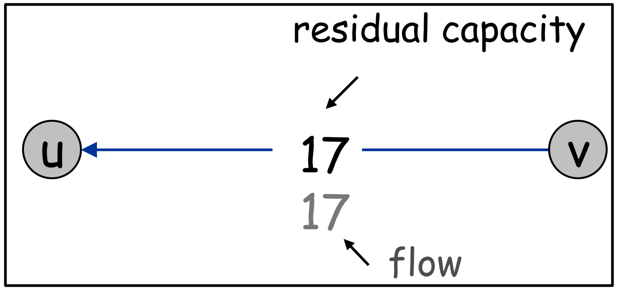
- Flow $f(e)$, capacity $c(e)$.

Create (New) Residual edges:

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



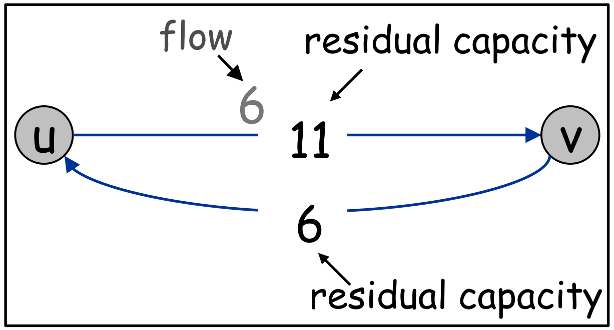
b. If $f(u, v) = c(u, v)$, it has one residual edge (v, u) with residual capacity $c_f(v, u) = f(u, v)$



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 $c_f(v, u) = f(u, v)$

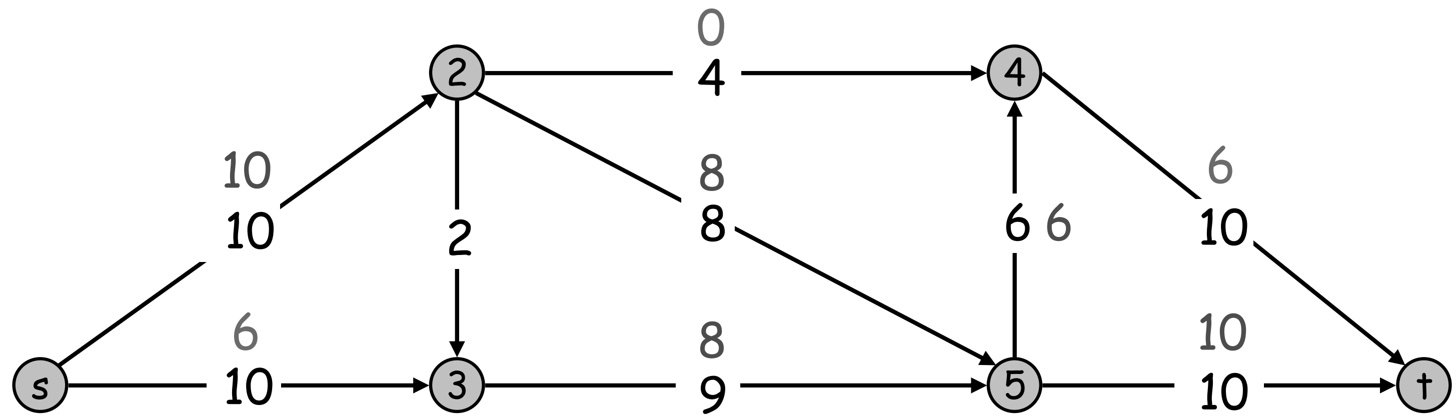


Residual graph: $G_f = (V, E_f)$.

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

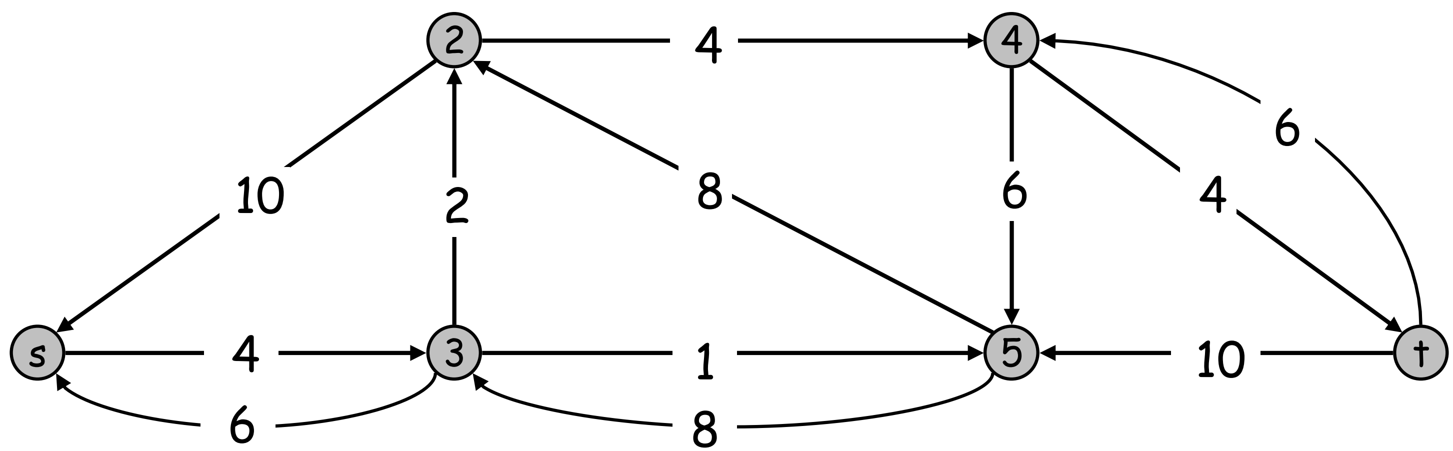
A Graph G, flow f and associated residual Graph G_f

G:



Flow value = 16

G_f:



Ford-Fulkerson Algorithm

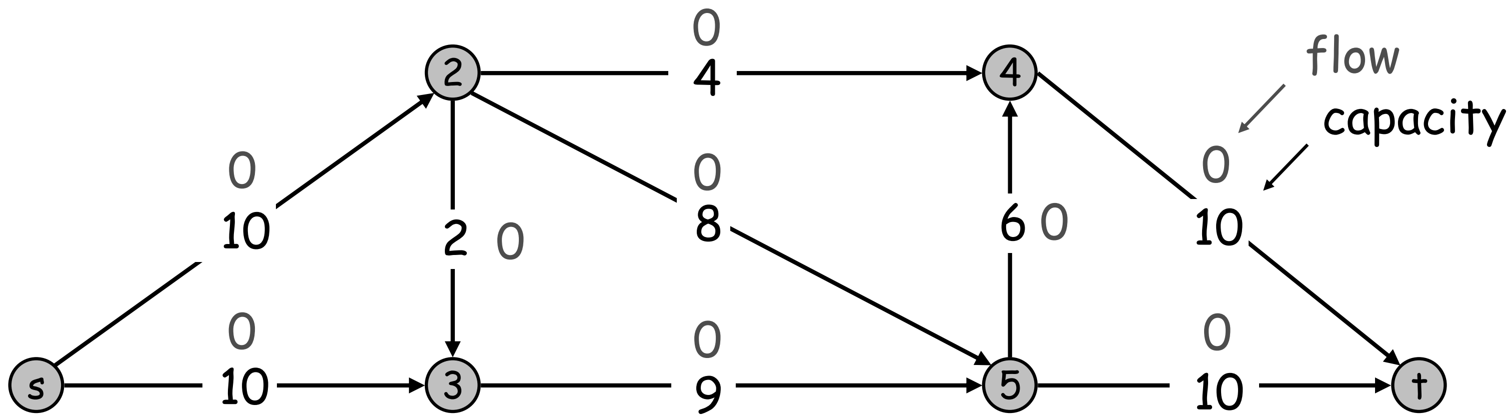
Greedy algorithm.

1. Start with $f(e) = 0$ for all edge $e \in E$.
2. Construct Residual Graph G_f for current flow f(e)=0
3. While there exists some s-t path P IN G_f
4. Let $c_f(p) \leftarrow \min\{c_f(e) : e \in P\}$
   This is the maximum amount of flow that can be pushed through residual capacity of P's edges
5. Push $c_f(p)$ units of flow along the edges $e \in P$
   by adding $c_f(p)$ to f(e) for every $e \in P$
6. Construct Residual Graph G_f for new current flow f(e)

Claim: When algorithm gets stuck, current flow is maximal!

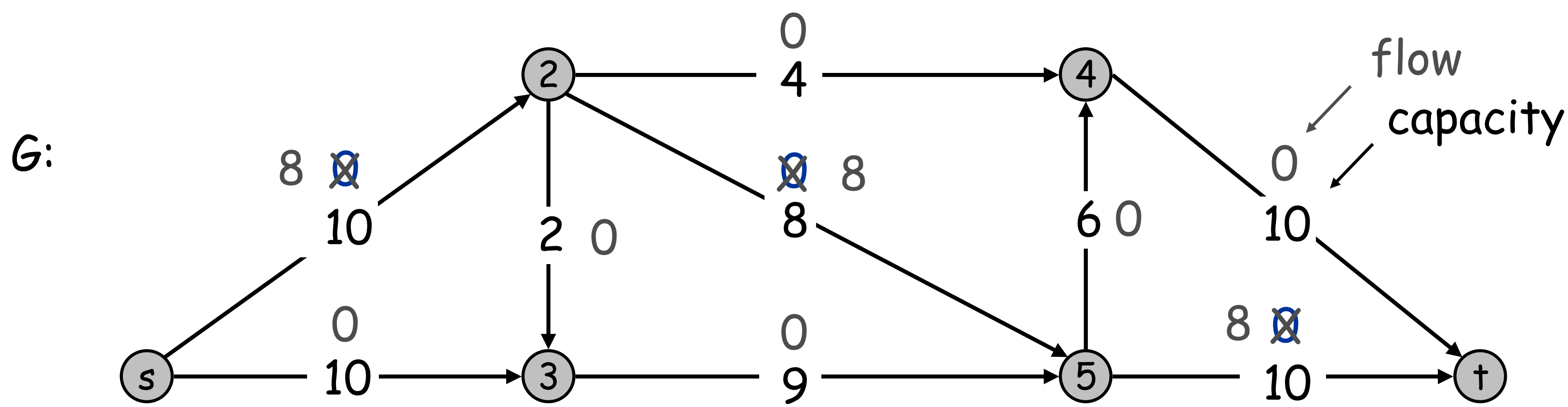
Ford-Fulkerson Algorithm

G:

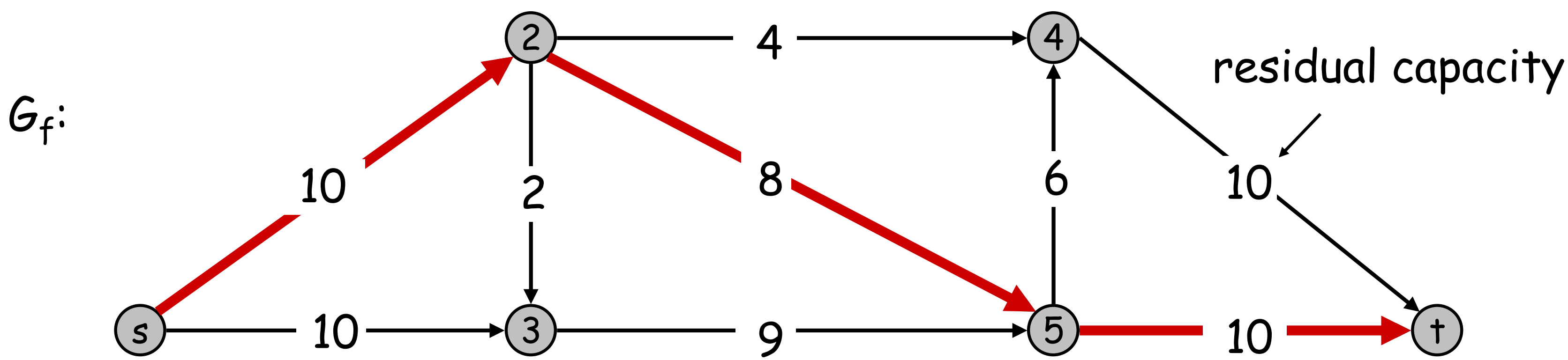


Flow value = 0

Ford-Fulkerson Algorithm

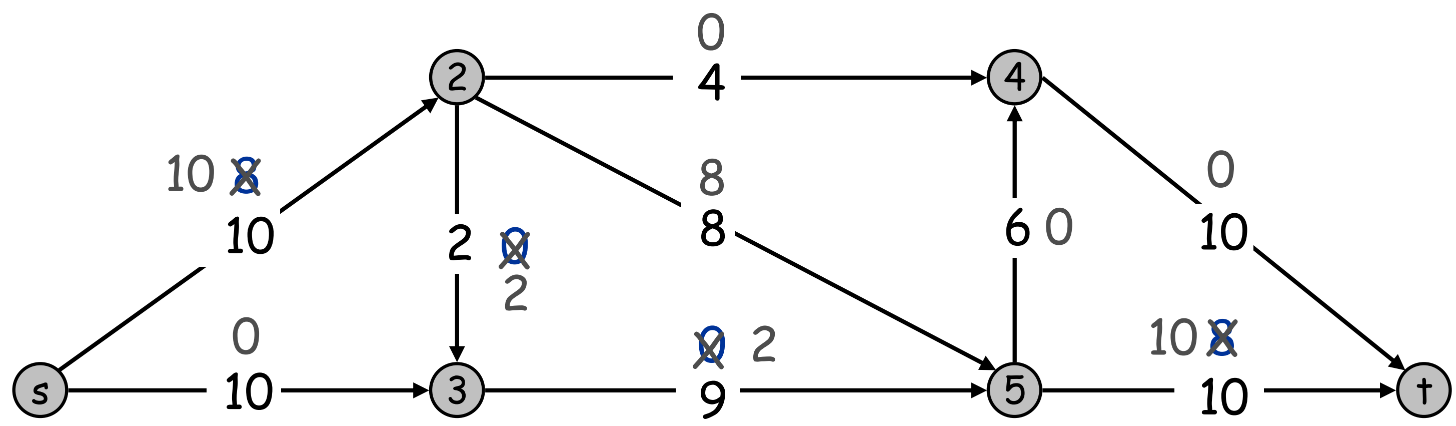


Flow value = 0



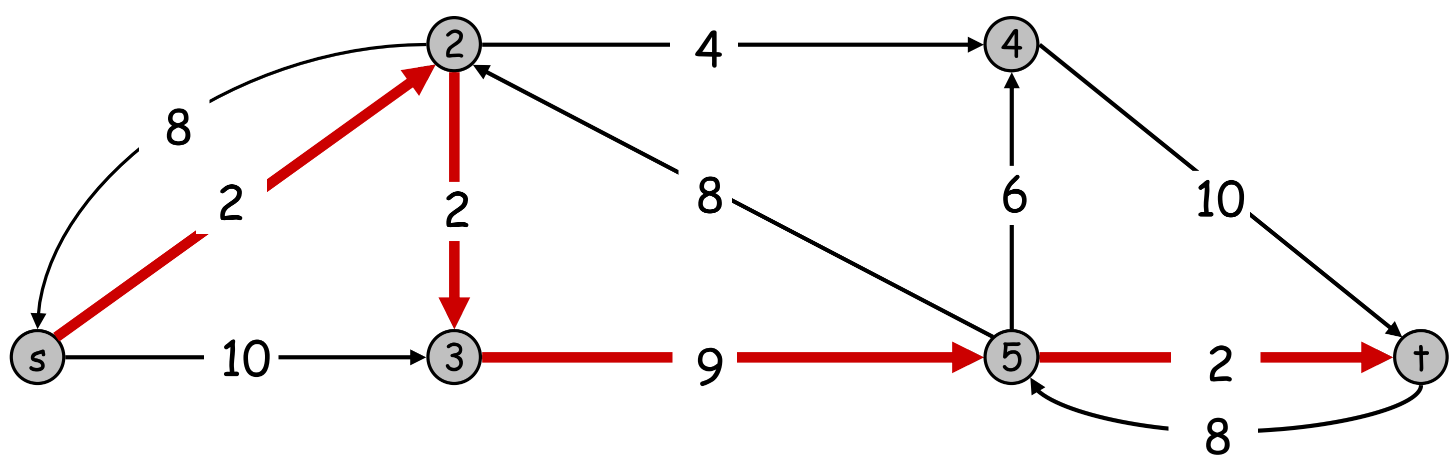
Ford-Fulkerson Algorithm

G:



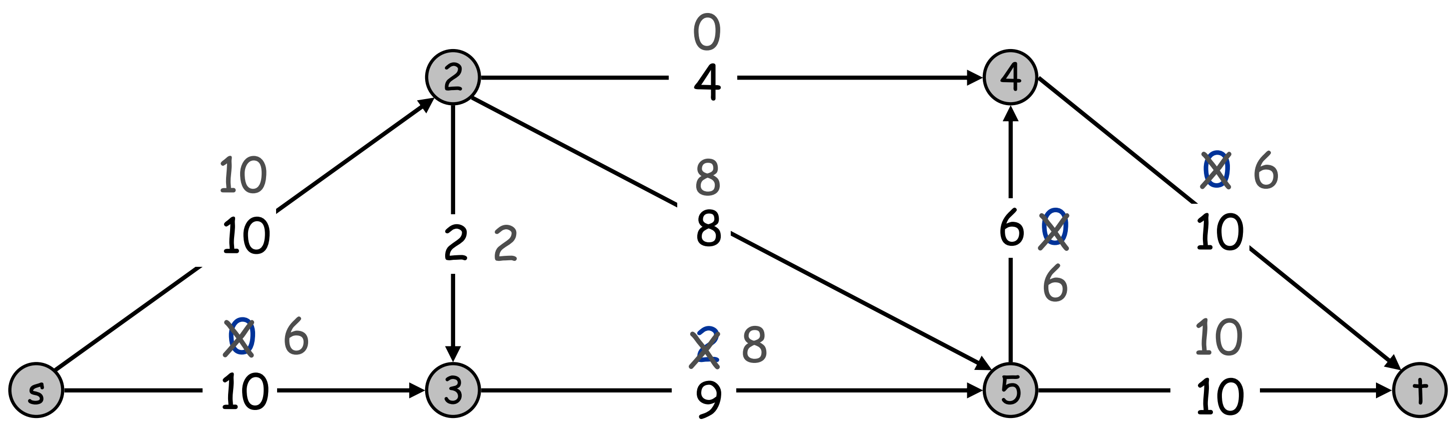
Flow value = 8

G_f:



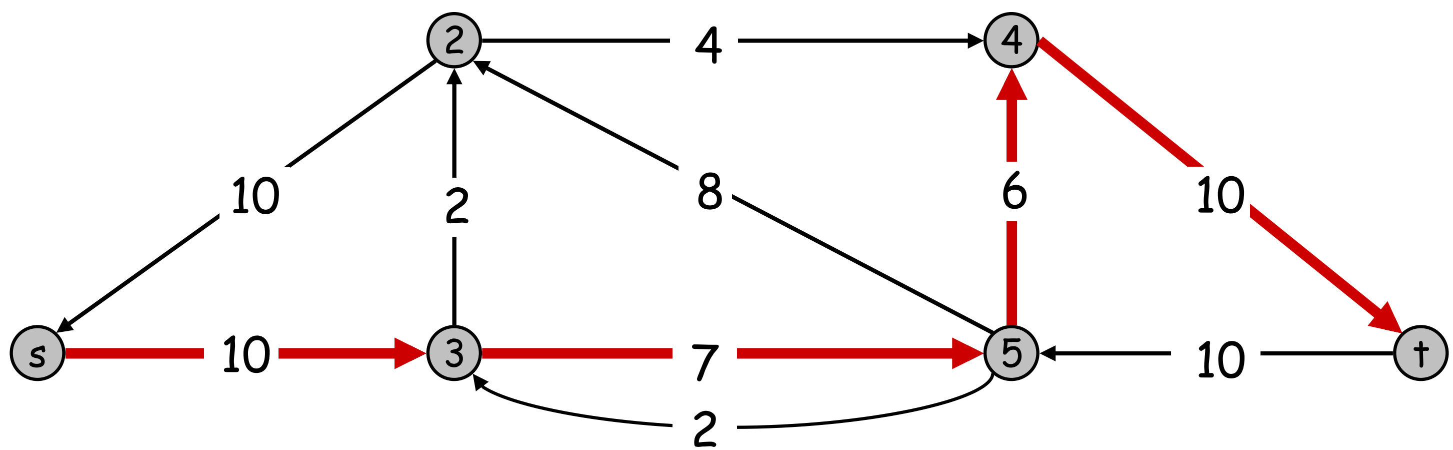
Ford-Fulkerson Algorithm

G:

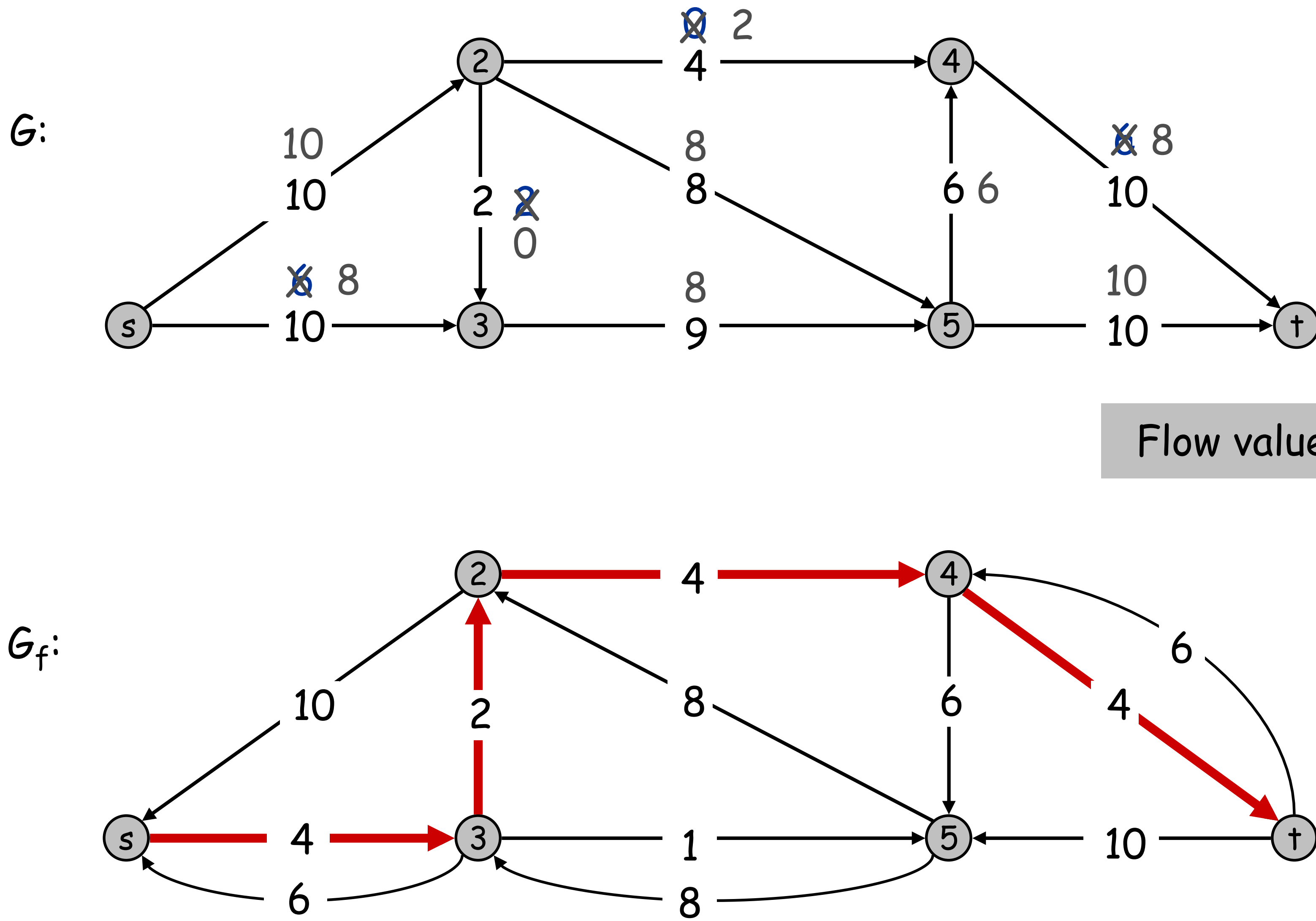


Flow value = 10

G_f:

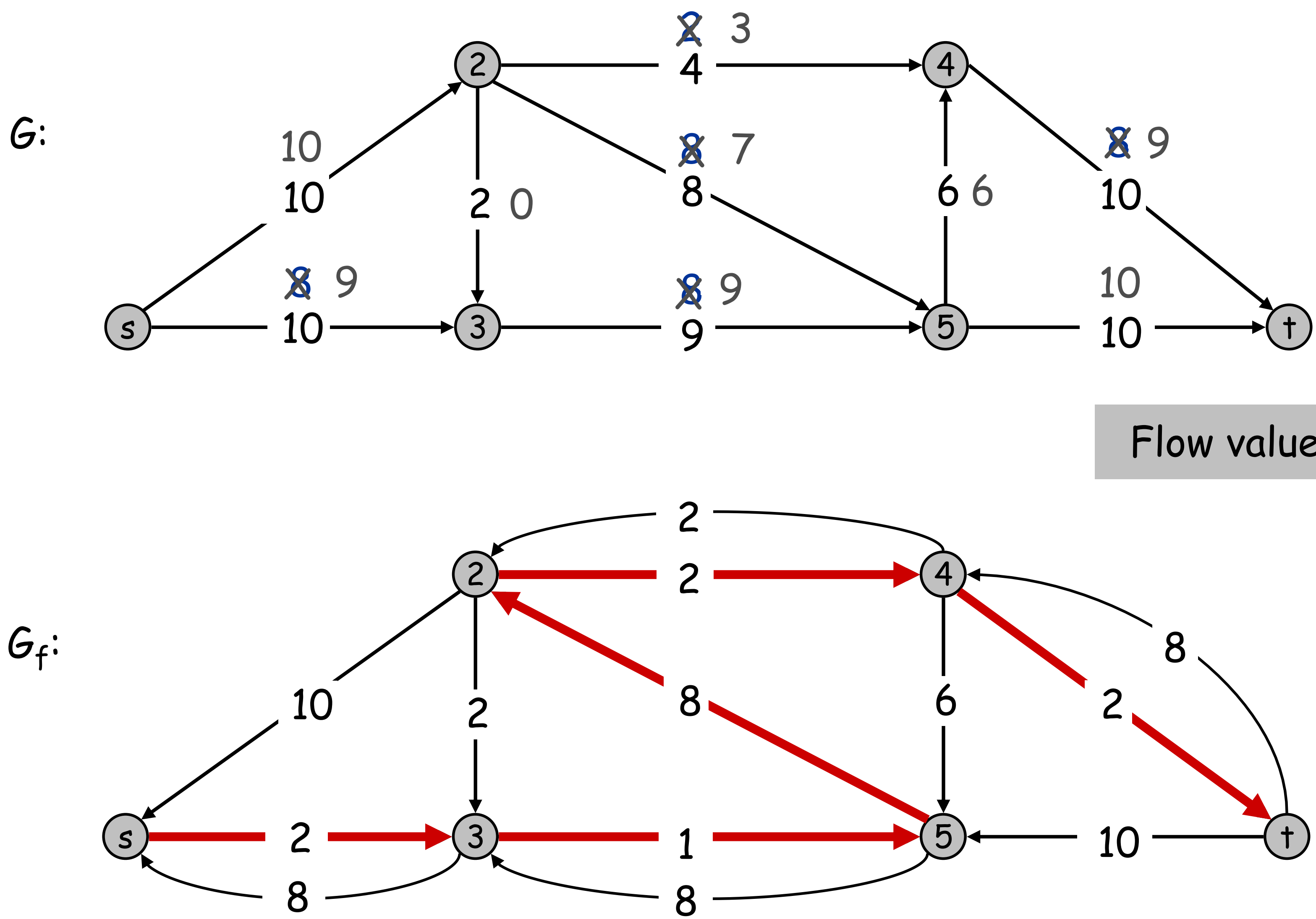


Ford-Fulkerson Algorithm



Flow value = 16

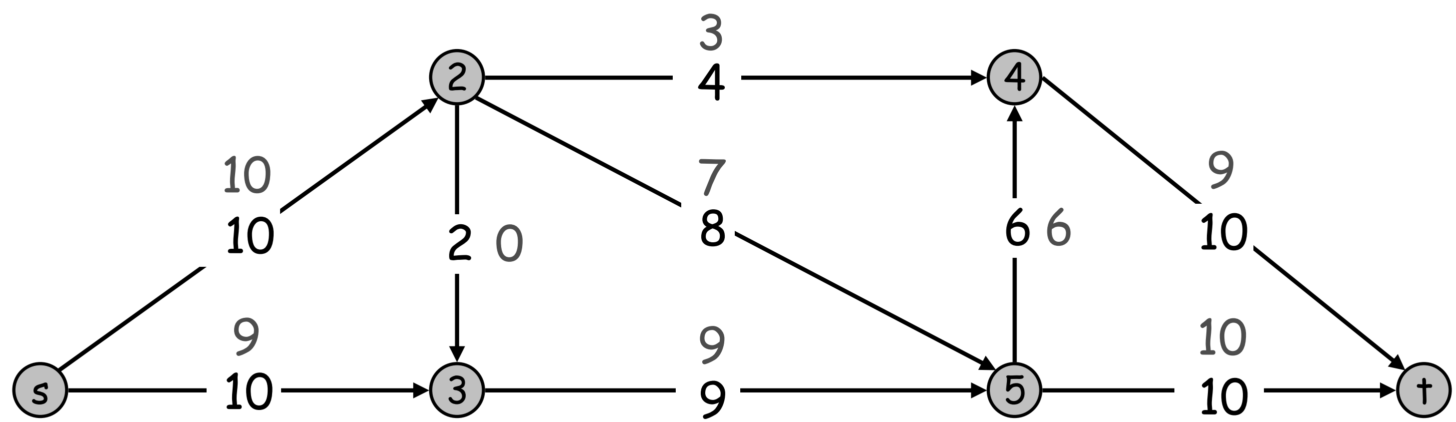
Ford-Fulkerson Algorithm



Flow value = 18

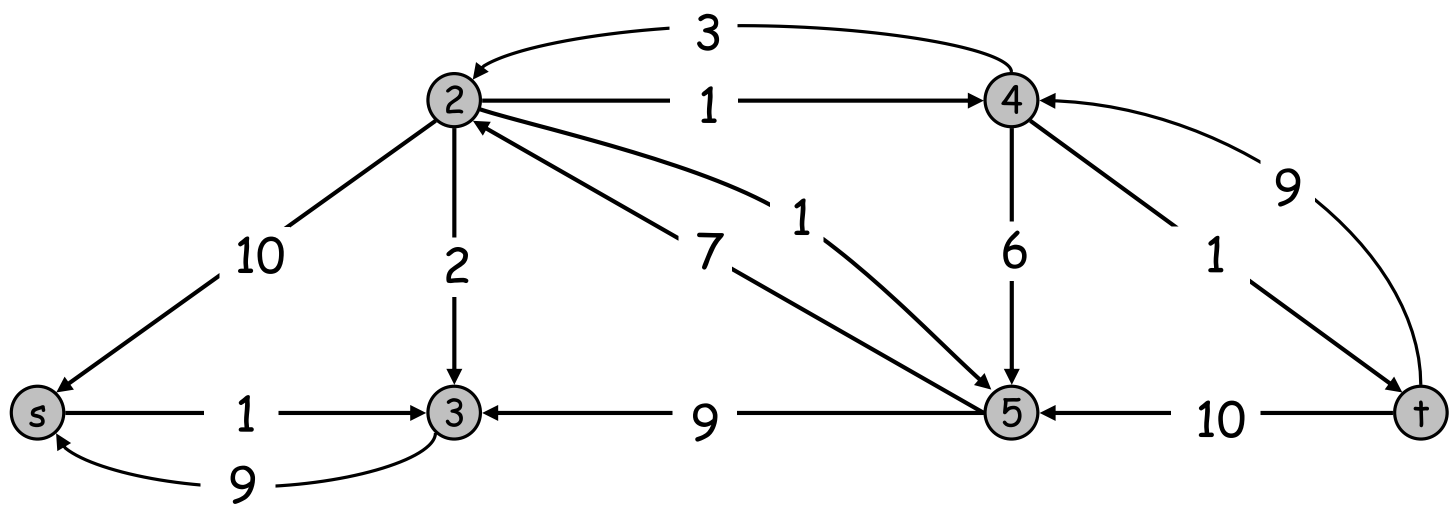
Ford-Fulkerson Algorithm

G:



Flow value = 19

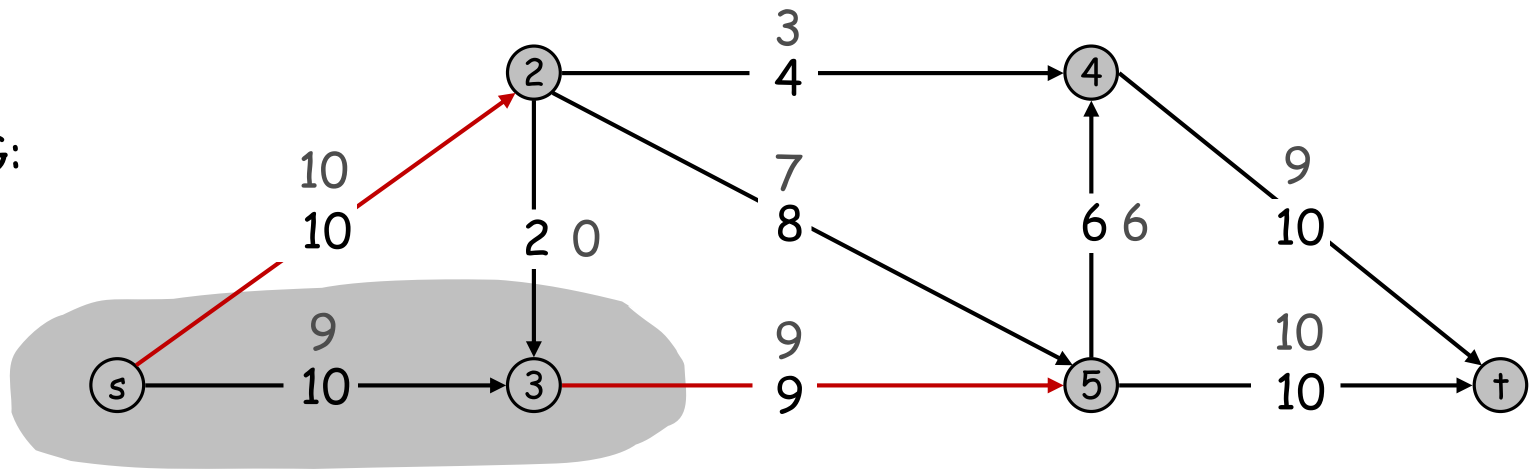
G_f:



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

Ford-Fulkerson Algorithm

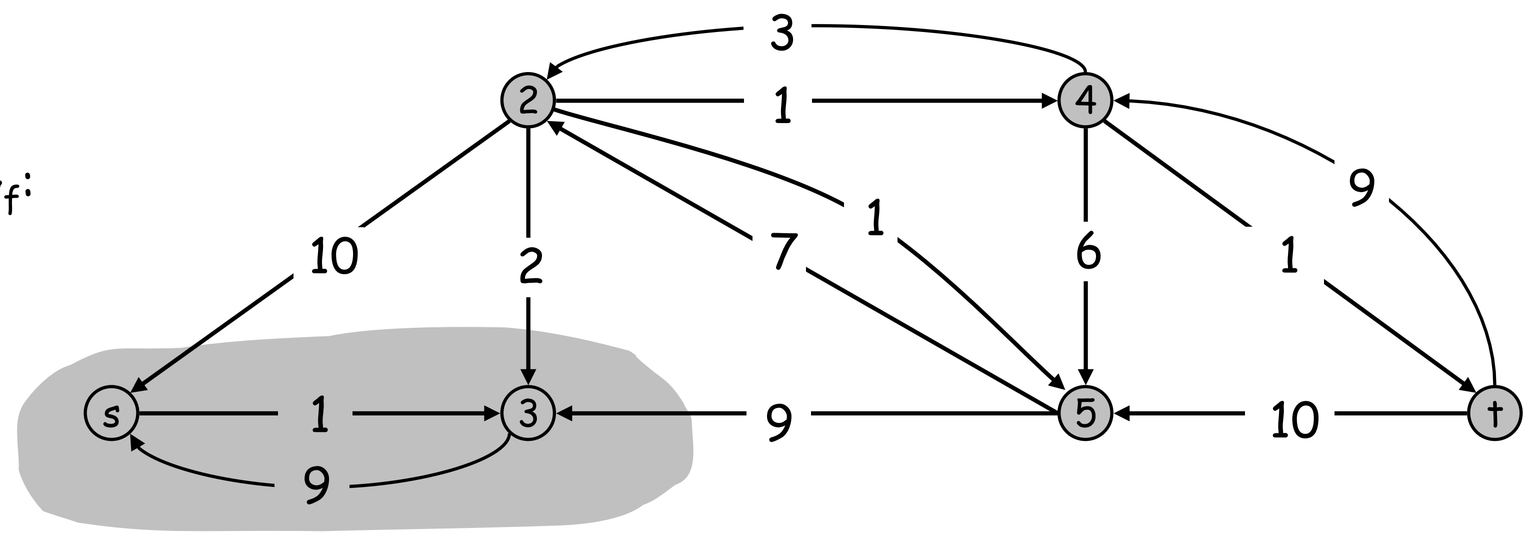
G:



Cut capacity = 19

Flow value = 19

G_f:



Ford-Fulkerson Algorithm

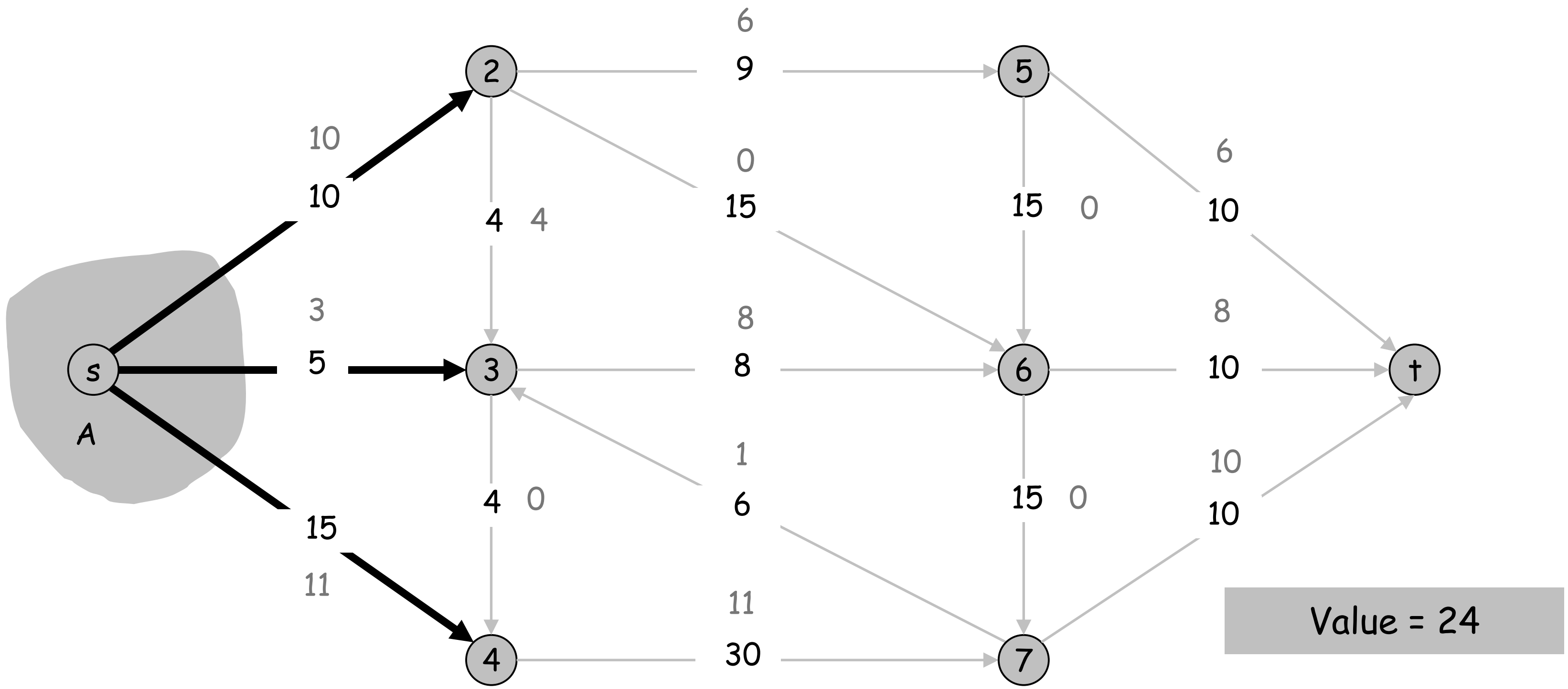
```
Ford-Fulkerson(G,s,t) {
for each (u,v) ∈ E do
      f(u,v) ← 0
      c_f(u,v) ← c(e)
      c_f(v,u) ← 0
while there exists path P in residual graph G_f do
      c_f(p) ← min{c_f(e): e ∈ P}
      for each edge (u,v) ∈ P do
            if (u,v) ∈ E then
                  f(u,v) ← f(u,v) + c_f(p)
 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

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), $f(S,T) = |f|$.

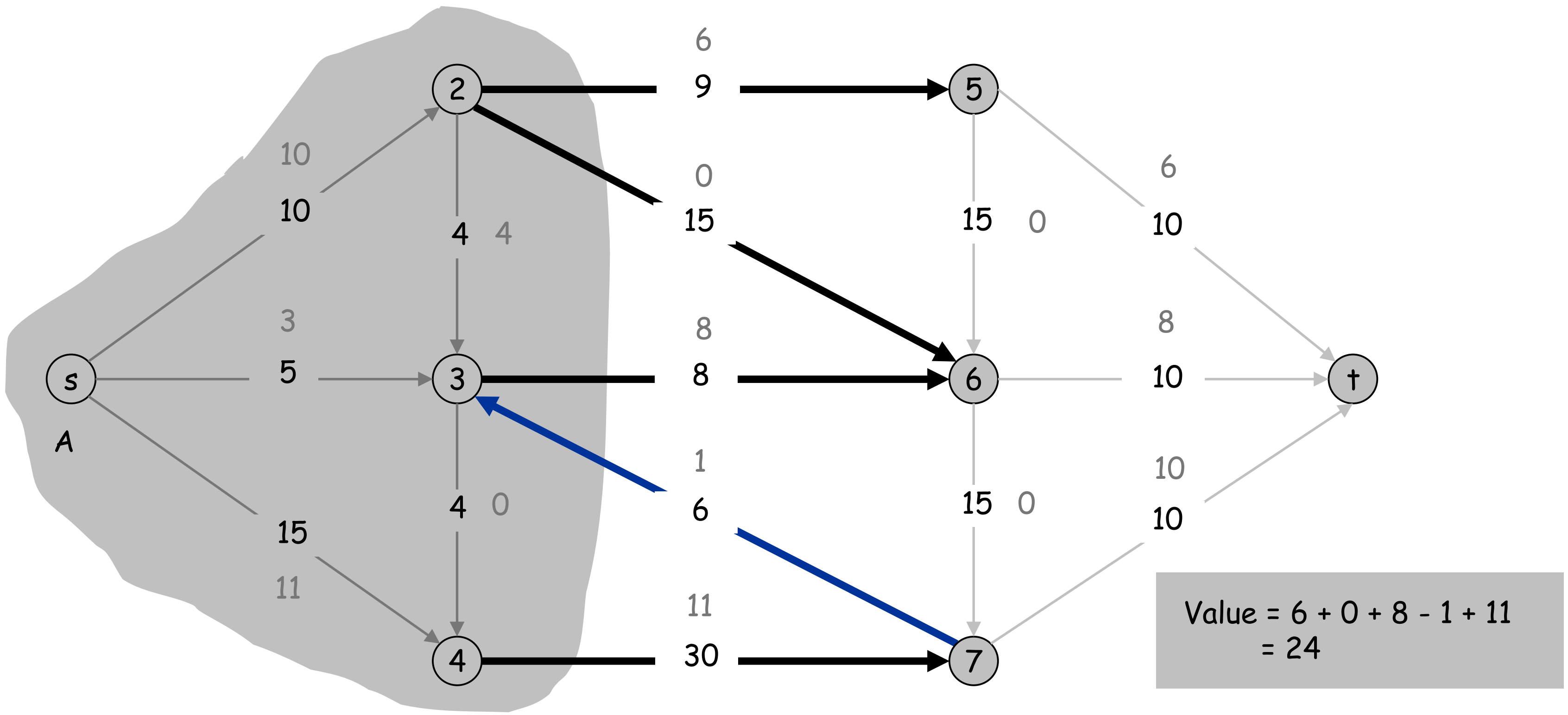


Net flow and cuts

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), $f(S,T) = |f|$.

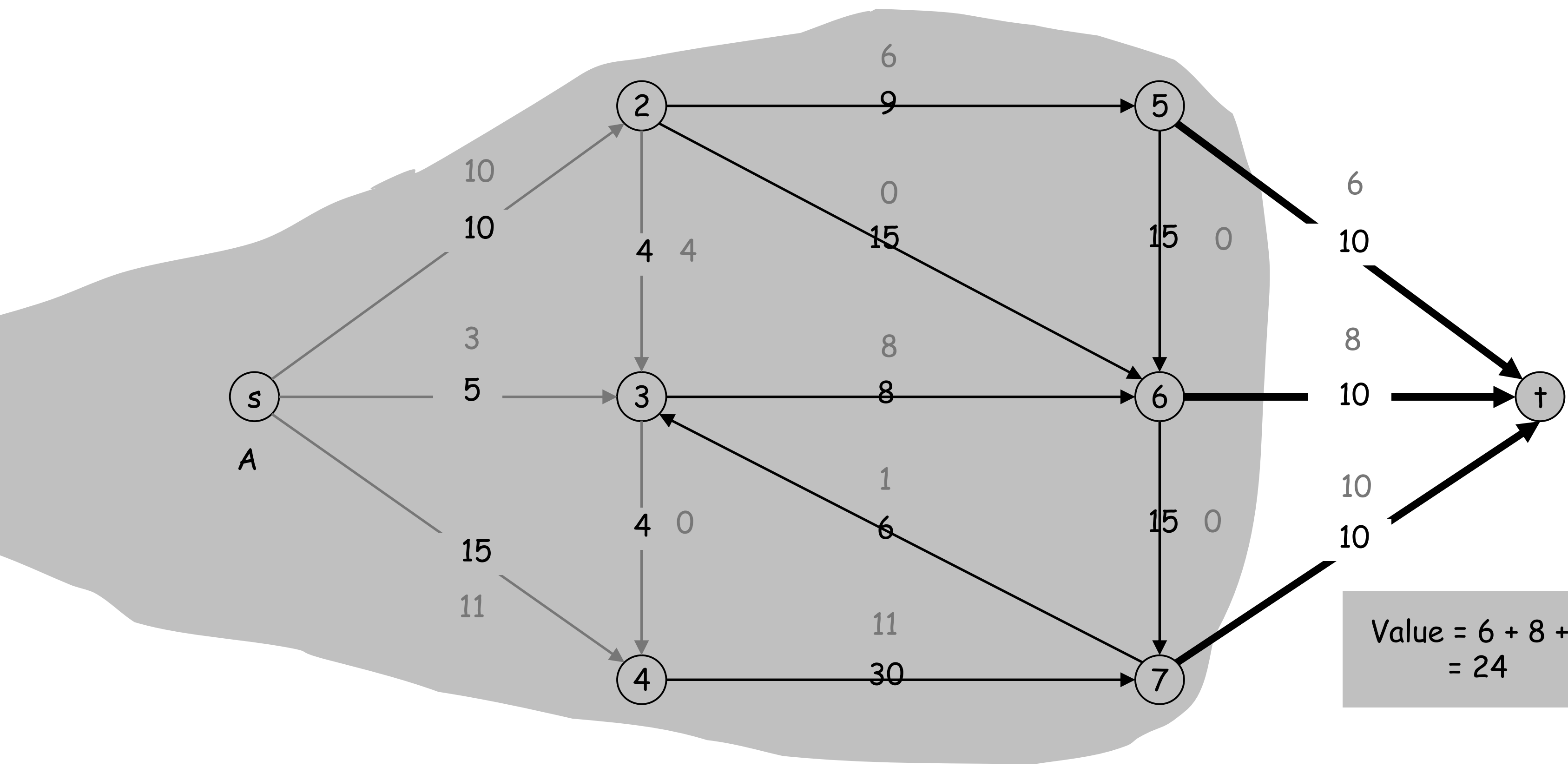


Net flow and cuts

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), $f(S,T) = |f|$.



Value = 6 + 8 + 10
= 24

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.

Flow and Cuts

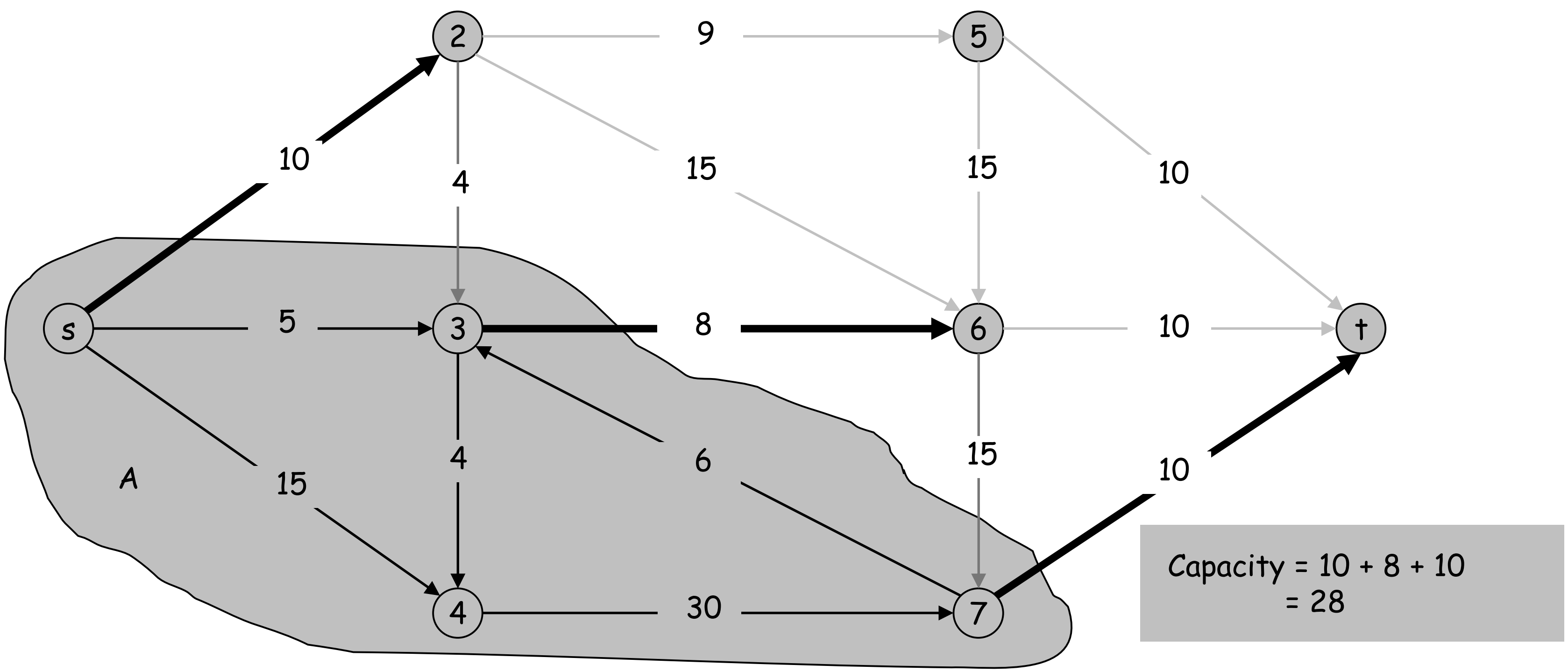
Def: The capacity of the cut (S, T) is $c(S,T) = \sum_{e \text{ from } S \text{ to } T} c(e)$

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

Proof:

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

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



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
  - all edges e from S to T are full, i.e., $f(e) = c(e)$
  - 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

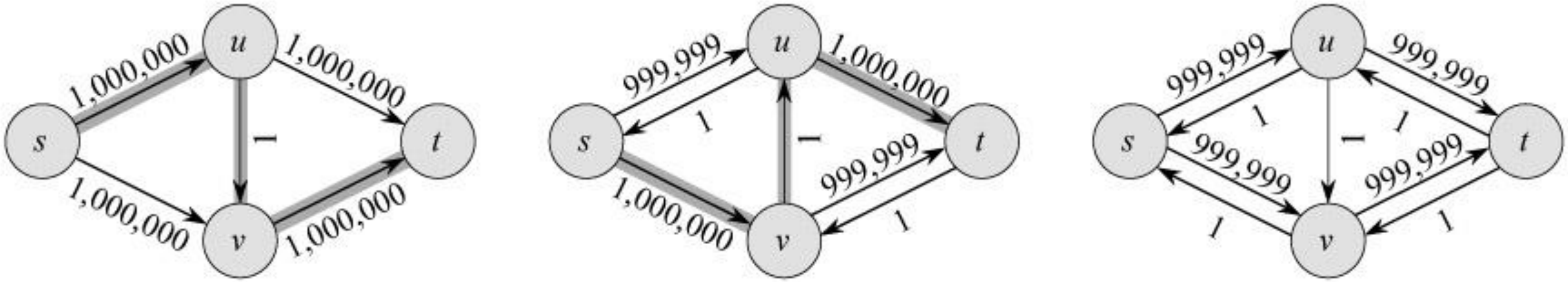
Claim: When all capacities are integers, Ford-Fulkerson takes at most $|f^*|$ iterations, where f^* is a maximum flow.

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



(a) (b) (c)

When capacities are irrational numbers, the algorithm may 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 best algorithm: $O(VE)$ time [King, Rao, Tarjan, 1994] [Orlin, 2013]

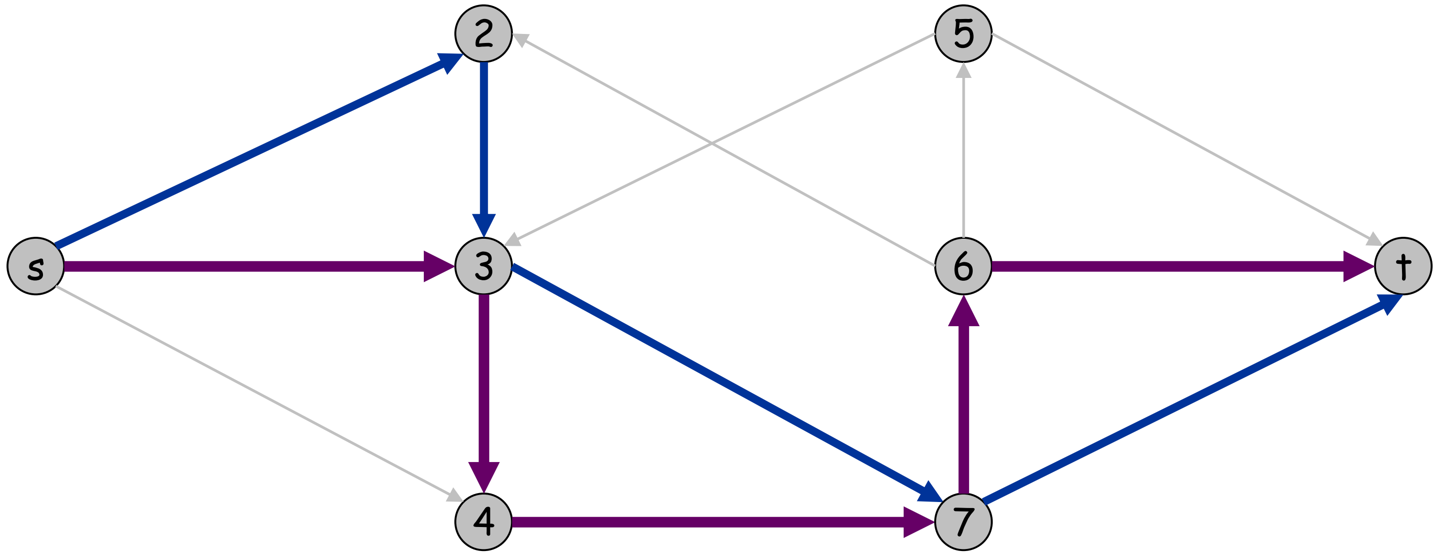
Applications of Max Flow

Edge Disjoint Paths

Disjoint path problem. Given a directed graph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint s-t paths.

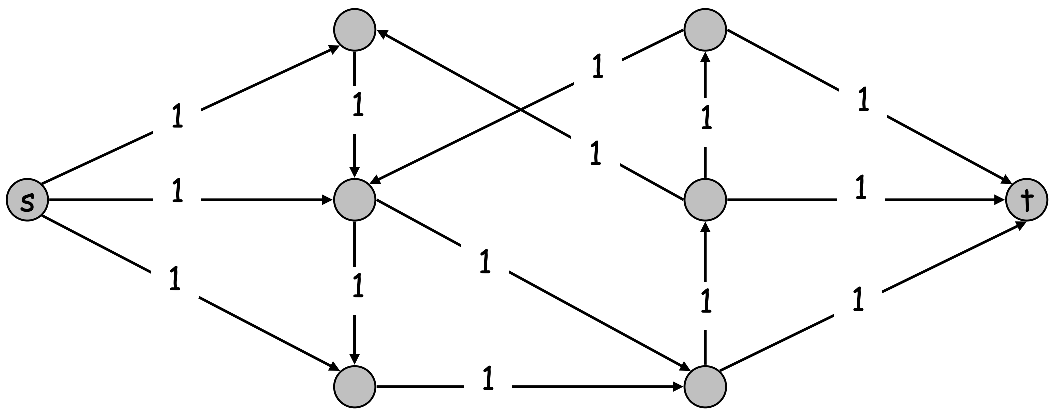
Def. Two paths are edge-disjoint if they have no edge in common.

Application: Communication networks.



Edge Disjoint Paths

Max flow formulation: assign unit capacity to every edge.



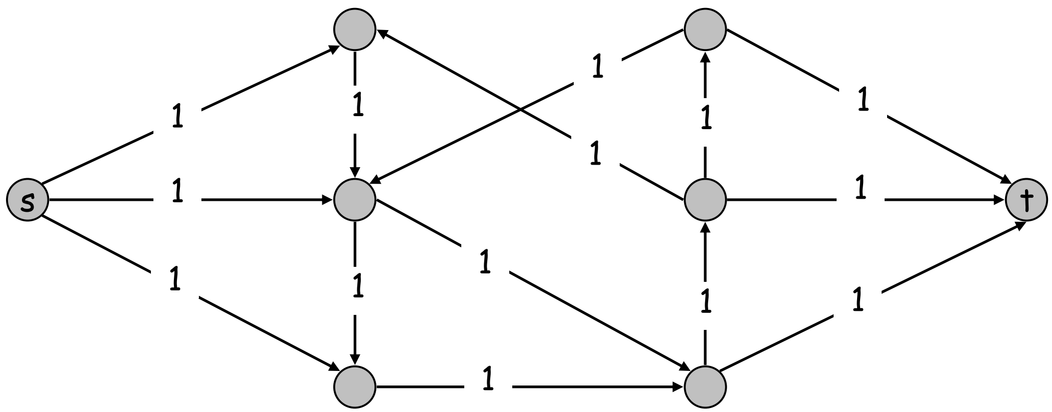
Theorem. Max number edge-disjoint s-t paths equals max flow value.

Proof. $\leq$

- Suppose there exists k edge-disjoint paths $P_1, \ldots, P_k$.
- Set $f(e) = 1$ if e participates in some path P_i; else set $f(e) = 0$.
- Since paths are edge-disjoint, f is a flow of value k.

Edge Disjoint Paths

Max flow formulation: assign unit capacity to every edge.



Proof. $\geq$

- Let f be a max flow in G' of value k computed by Ford-Fulkerson
- $f(e) = 1$ or 0 for every edge e (integrality property).

- Consider any edge (s, u) with $f(s, u) = 1$.
  - By conservation, there exists edge (u, v) with $f(u, v) = 1$
  - Continue to find the next unused edge out of v until reaching t.

- After finding one path, flow value decreases by 1.
- Repeat the process k times to find k edge-disjoint paths.
- The proof above also provides an algorithm.

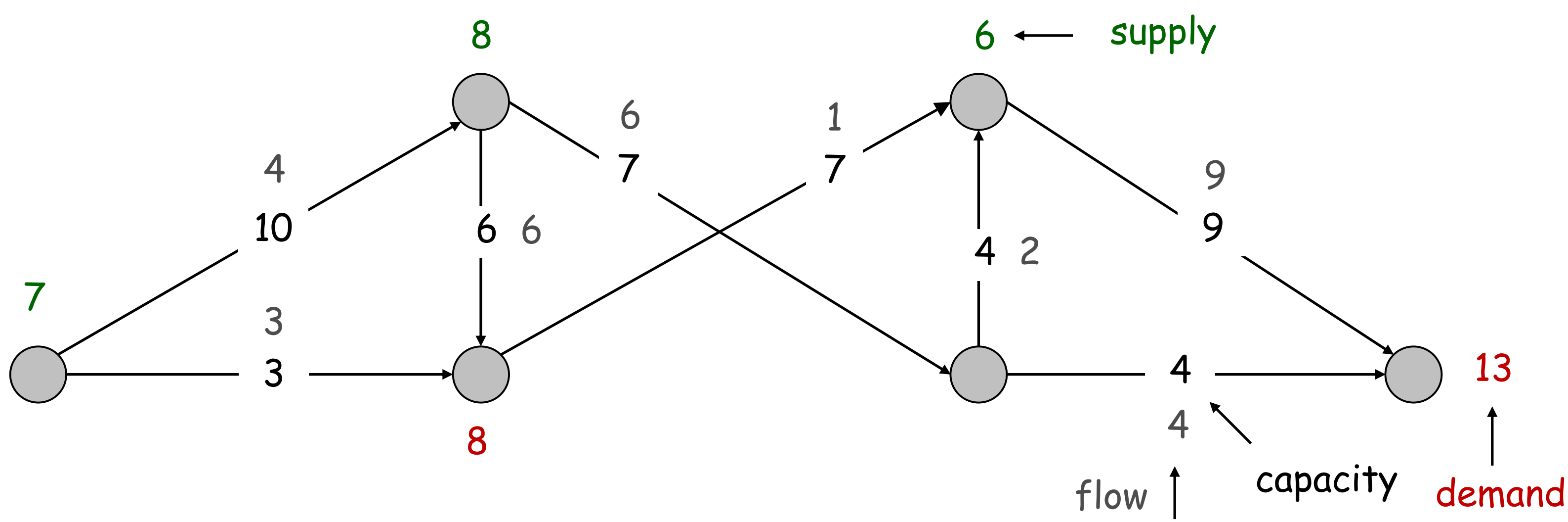
Circulation with Demands

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

- every edge $e \in E$ has a capacity $c(e)$;
- a number of source vertices $s_1, s_2, \ldots$, each with a supply of $sup(s_i)$ and a number of target vertices $t_1, t_2, \ldots$, each with a demand of $dem(t_i)$;
- $\sum_i sup(s_i) \geq \sum_i dem(t_i)$

Output: A flow f that meets capacity and conservation conditions, and

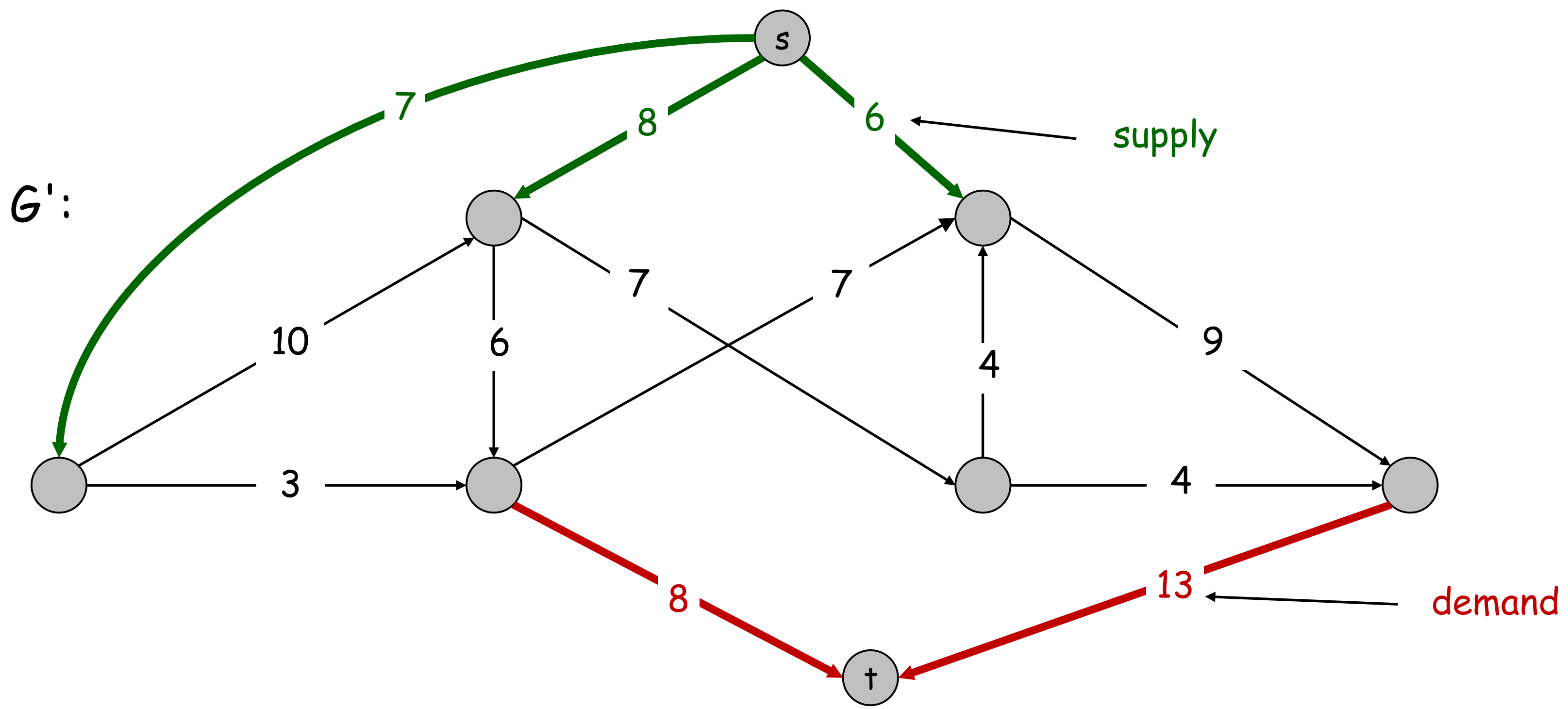
- At each source vertex s_i, $\sum_{e \text{ out of } s_i} f(e) - \sum_{e \text{ into } s_i} f(e) \leq sup(s_i)$;
- At each target vertex t_i, $\sum_{e \text{ into } t_i} f(e) - \sum_{e \text{ out of } t_i} f(e) = dem(s_i)$.



Solving Circulation with Demands using Max Flow

Algorithm:

- Add a "super source" s and a "super target" t.
- Add an edge from s to each s_i with capacity $sup(s_i)$.
- Add an edge from each t_i to t with capacity $dem(t_i)$.
- Compute the max flow f.
- If $|f| = \sum_i dem(t_i)$, then return f; else return "no solution".



Baseball Elimination

Team i	Wins w_i	To play r_i	Remaining Against = r_{ij}			
			1	2	3	4
1	3	2	-	1	1	0
2	2	3	1	-	1	1
3	2	3	1	1	-	1
4	0	2	0	1	1	-

Rule: Order teams by the number of wins.

Q: Does Team 4 still have a chance to finish in the first place (tie is OK)?

A: No, obviously.

Baseball Elimination

Team i	Wins w_i	To play r_i	Remaining Against = r_{ij}			
			1	2	3	4
1	3	2	-	1	1	0
2	2	3	1	-	1	1
3	2	3	1	1	-	1
4	1	2	0	1	1	-

Q: Does Team 4 still have a chance to finish in the first place (tie is OK)?

A: No, because

- Team 4 has to win both remaining games against team 2 and 3.
- Team 1 has to lose both remaining games against team 2 and 3.
- Then 2 and 3 will both have 3 wins.
- The game between team 2 and 3 will give one of them one more win.

Suppose you need to do this for MLB / Premier League…

Baseball Elimination: Formal Definition

Input:

- n teams: $1, 2, \ldots, n$
- One particular team, say n (without loss of generality)
- Team i has won w_i games already
- Team i and j still need to play r_{ij} games, $r_{ij} = 0$ or 1.
- Team i has a total of $r_i = \sum_j r_{ij}$ games to play

Output:

- "Yes", if there is an outcome for each remaining game such that team n finishes with the most wins (tie is OK).
- "No", if no such possibilities.

Brute-force algorithm:

- For each remaining game, consider two possible outcomes.
- Try all 2^r possible combinations, where $r = \sum_{i,j} r_{ij}$

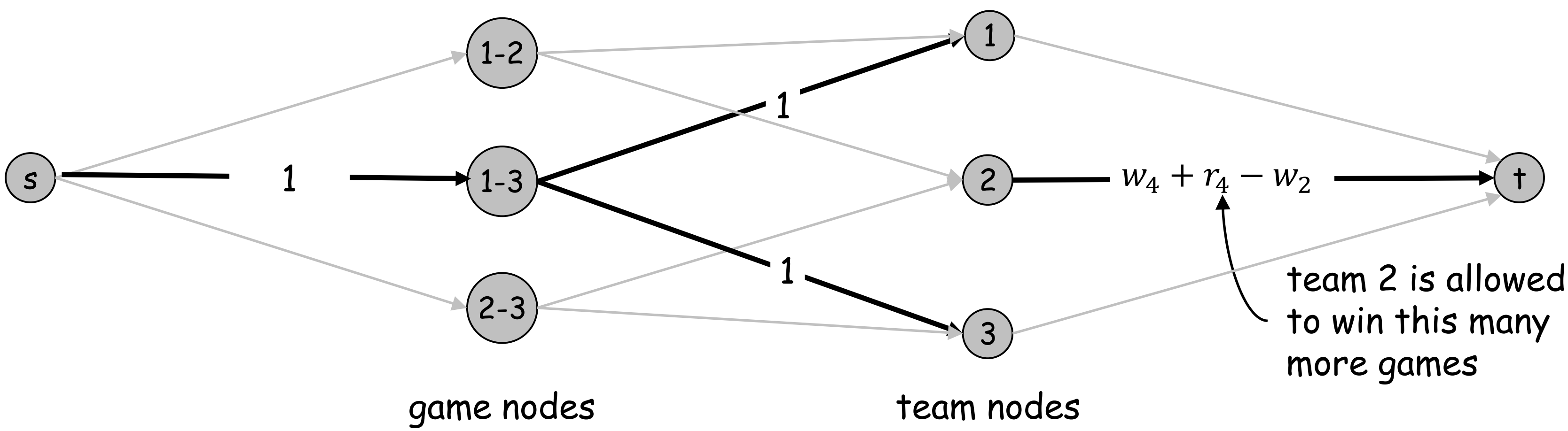
Baseball Elimination: Max Flow Formulation

Can team n finish with most wins?

- Assume team n wins all remaining games $\Rightarrow w_n + r_n$ wins.
- All other teams must have $\leq w_n + r_n$ wins.

Flow network construction:

- A source s and a target t
- A node for each remaining game (i, j); and an edge from s to it with capacity 1
- A node for each team $i = 1, 2, \dots, n-1$; and an edge from it to t with capacity $w_n + r_n - w_i$
- Game node (i, j) has edges to team node i and j, with capacity 1



Baseball Elimination: Max Flow Formulation

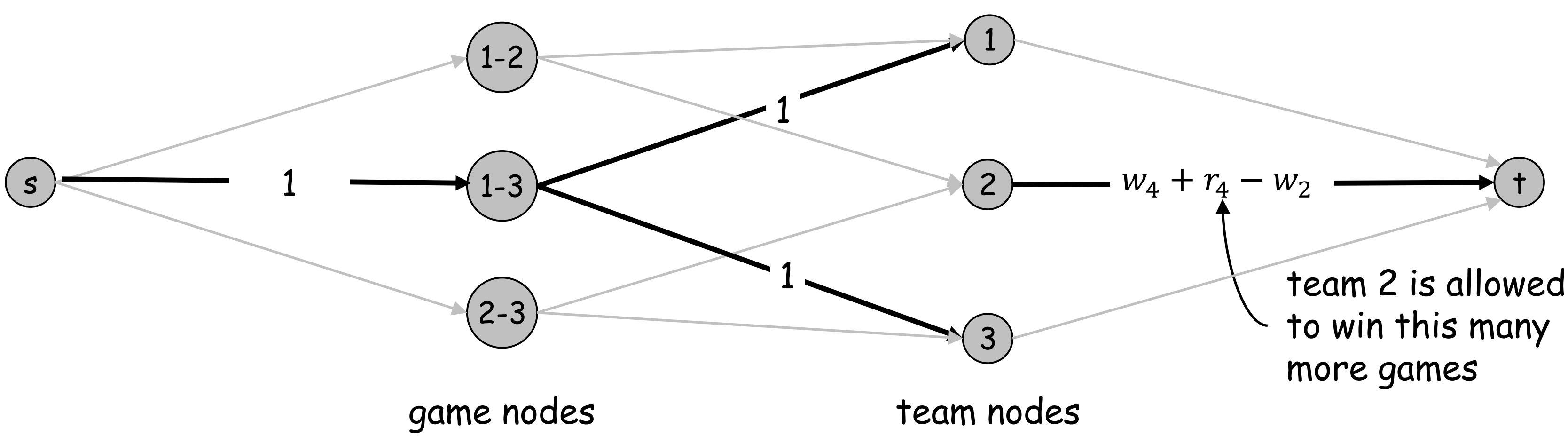
Claim: There is a way for team n to finish in the first place iff the max flow has value $r = \sum_{i,j} r_{ij}$.

Proof: "⇒": Suppose there is an outcome for each remaining game such that team n finishes the first. First set $f\big(s, (i,j)\big) = 1$ for all (i,j).

For each remaining game (i,j):

- if i wins, set $f\big((i,j), i\big) = 1$ and $f\big((i,j), j\big) = 0$;
- if j wins, set $f\big((i,j), j\big) = 1$ and $f\big((i,j), i\big) = 0$.

Team i wins $\leq w_n + r_n - w_i$ games, so it can send all incoming flow to t.



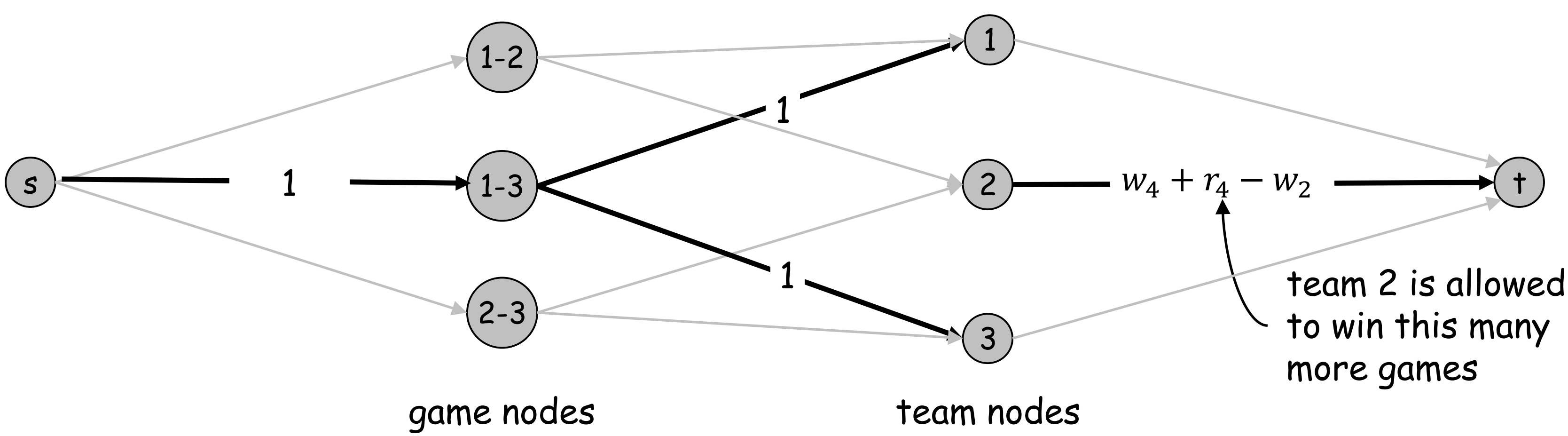
Baseball Elimination: Max Flow Formulation

Proof: "$\Leftarrow$": Suppose the max flow f has $|f| = r$. It must saturate all edges out of s.

Look at each game node (i, j). Exactly one of its outgoing edges must have 1 unit of flow (integrality property):

- If $f\big((i,j), i\big) = 1$, let i win the game;
- If $f\big((i,j), j\big) = 1$, let j win the game.

Team node i receives $\leq w_n + r_n - w_i$ units of flow, each corresponding to one win, so it cannot beat team n.



Baseball Elimination: Extensions

Q: What if r_{ij} can be more than 1?

Q: Can this be used for football (soccer) leagues?

- Using the old rule: Winner takes 2 points, loser 0 point; each team gets 1 point in case of a tie.