CSE 421
Introduction to Algorithms

Lecture 16: Maxflow/MinCut
Ford-Fulkerson

Announcements

See EdStem Announcement/Email posted/sent on Sunday/Monday.

Midterm next **Monday, November 4, 6:00 – 7:30 pm in this room**

    - Exam designed for a regular class time-slot but this includes extra time to finish.

- Coverage:
    - Up to the end of last Thursday's section on Dynamic Programming

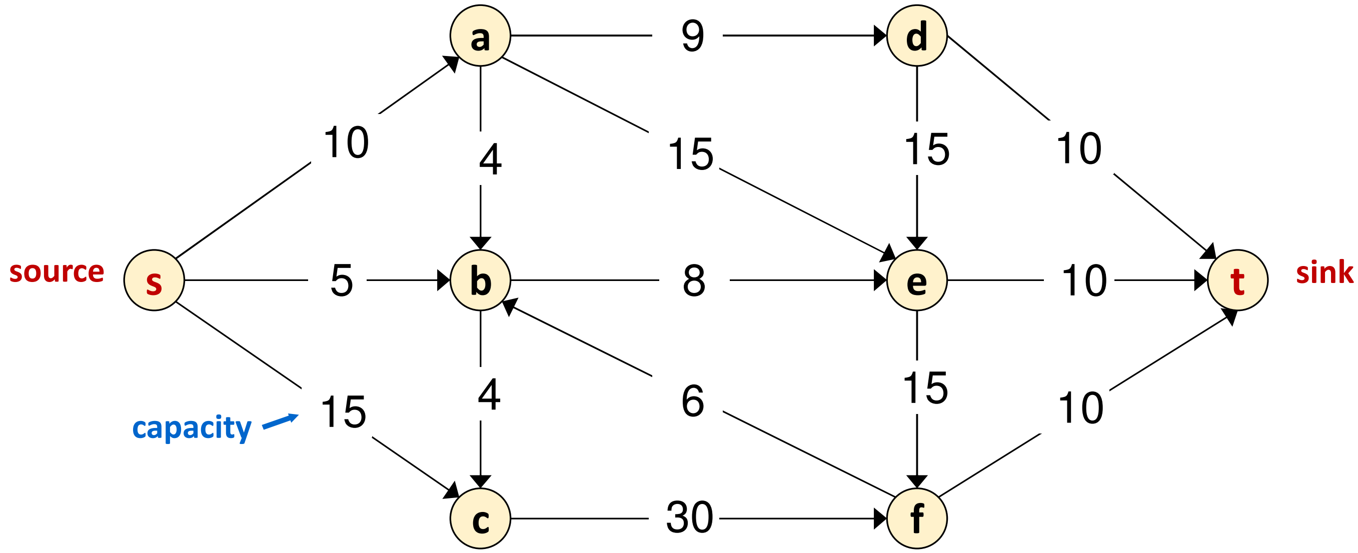
- See important details in two Ed posts. Sample midterm for practice problems.
    - Includes 2-page "reference sheet" available to you on the midterm.
- Tomorrow's section will focus on review problems.

- Zoom review session for Q&A on Sunday Nov 3 at 4:45 pm.

Last time: Flow Network

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 $e \geq 0$.



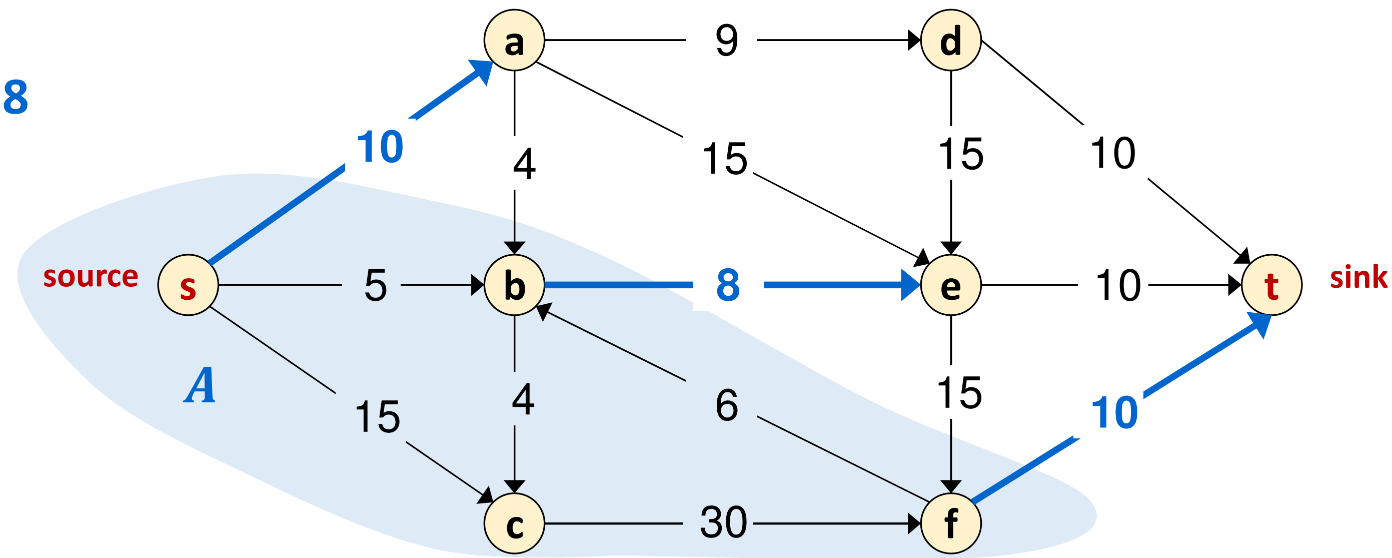
Last time: Minimum Cut Problem

Minimum s-t cut problem:

Given: a flow network

Find: an s-t cut (A, B) of minimum capacity $c(A,B) = \sum_{e \text{ out of } A} c(e)$

capacity **28**



Last time: Flows

Defn: An s-t flow in a flow network is a function $f: E \to \mathbb{R}$ that satisfies:

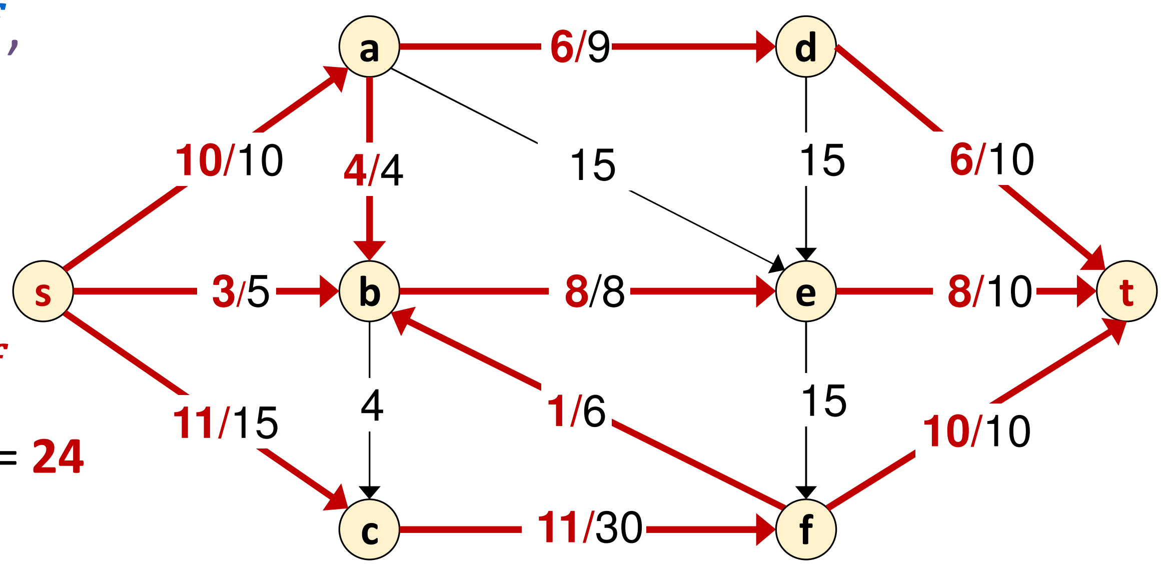
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity constraints]
- For each $v \in V - \{s,t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

Defn: The value of flow f,

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

Only show non-zero values of f

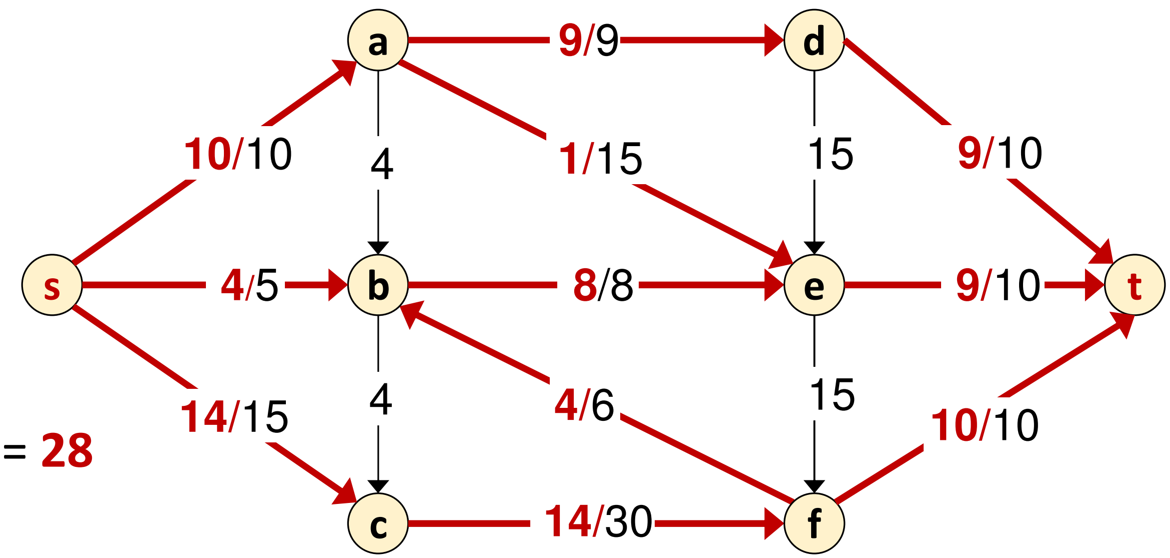
value = **24**



Last time: Maximum Flow Problem

Given: a flow network

Find: an s-t flow of maximum value



value = **28**

Last time: Certificate of Optimality

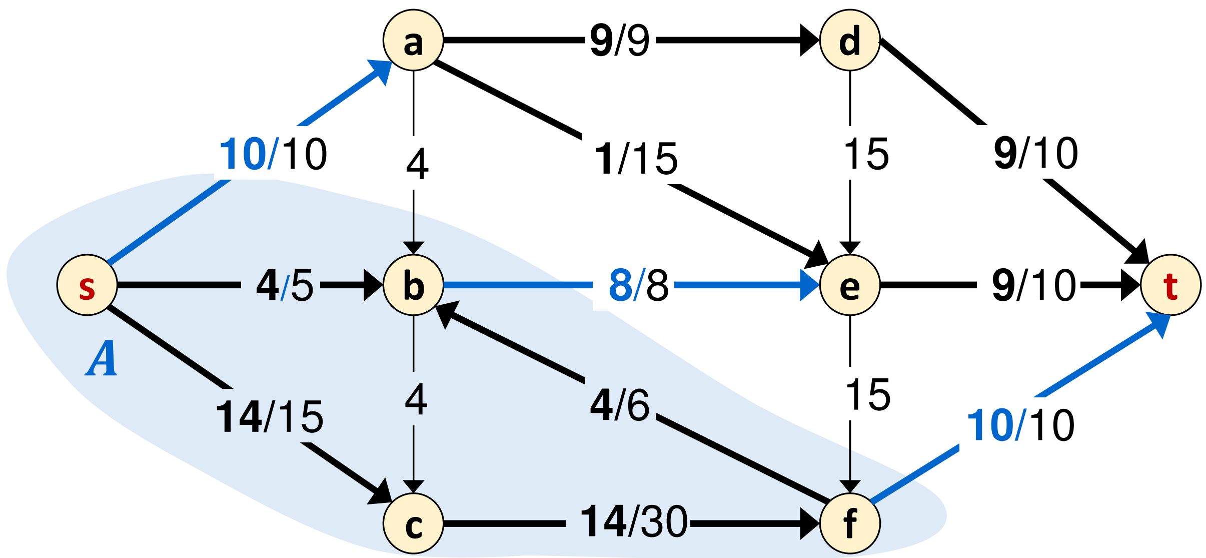
Corollary: Let f be any s-t flow and (A, B) be any s-t cut.

If $v(f) = c(A, B)$ then f is a max flow and (A, B) is a min cut.

Value of flow = **28**

Capacity of cut = **28**

Both are optimal!



Last time: Towards a Max Flow Algorithm

What about the following greedy algorithm?

- Start with $\boldsymbol{f(e) = 0}$ for all edges $\boldsymbol{e \in E}$.
- While there is an $\boldsymbol{s}$-$\boldsymbol{t}$ path $\boldsymbol{P}$ where each edge has $\boldsymbol{f(e) < c(e)}$.
  - "Augment" flow along $\boldsymbol{P}$; that is:
    - Let $\boldsymbol{\alpha} = \min_{\boldsymbol{e \in P}}(\boldsymbol{c(e) - f(e)})$
    - Add $\boldsymbol{\alpha}$ to flow on every edge $\boldsymbol{e}$ along path $\boldsymbol{P}$. (Adds $\boldsymbol{\alpha}$ to $\boldsymbol{v(f)}$.)

But this can get stuck...

Flows and cuts so far

Let $\boldsymbol{f}$ be any $\boldsymbol{s}$-$\boldsymbol{t}$ flow and $(\boldsymbol{A}, \boldsymbol{B})$ be any $\boldsymbol{s}$-$\boldsymbol{t}$ cut:

Flow Value Lemma: The net value of the flow sent across $(\boldsymbol{A}, \boldsymbol{B})$ equals $\boldsymbol{v}(\boldsymbol{f})$.

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

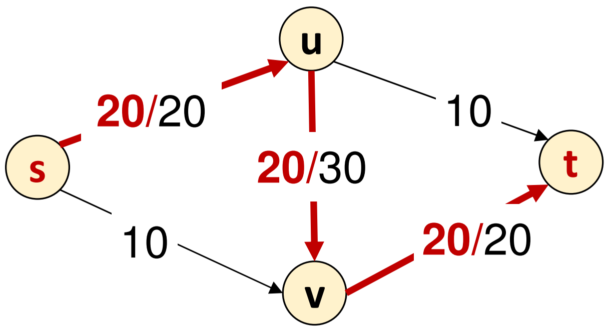
Weak Duality: The value of the flow is at most the capacity of the cut;
i.e., $\boldsymbol{v}(\boldsymbol{f}) \leq \boldsymbol{c}(\boldsymbol{A}, \boldsymbol{B})$. **"Maxflow ≤ Mincut"**

Corollary: If $\boldsymbol{v}(\boldsymbol{f}) = \boldsymbol{c}(\boldsymbol{A}, \boldsymbol{B})$ then $\boldsymbol{f}$ is a maximum flow and $(\boldsymbol{A}, \boldsymbol{B})$ is a minimum cut.

Augmenting along paths using a greedy algorithm can get stuck.

Today: Ford-Fulkerson Algorithm, which applies greedy ideas to a "residual graph" that lets us reverse prior flow decisions from the basic greedy approach.

Greed Revisited: Residual Graph & Augmenting Paths

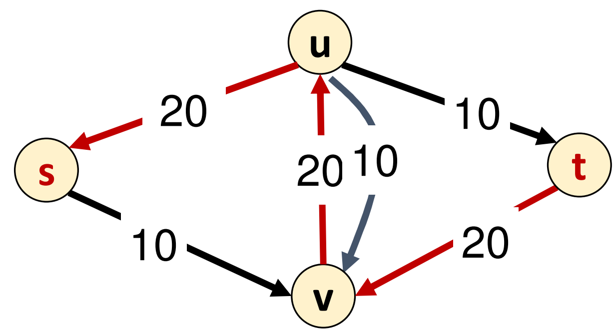


The only way we could route more flow from **s** to **t** would be to reduce the flow from **u** to **v** to make room for that amount of extra flow from **s** to **v**.
But to conserve flow we also would need to increase the flow from **u** to **t** by that same amount.

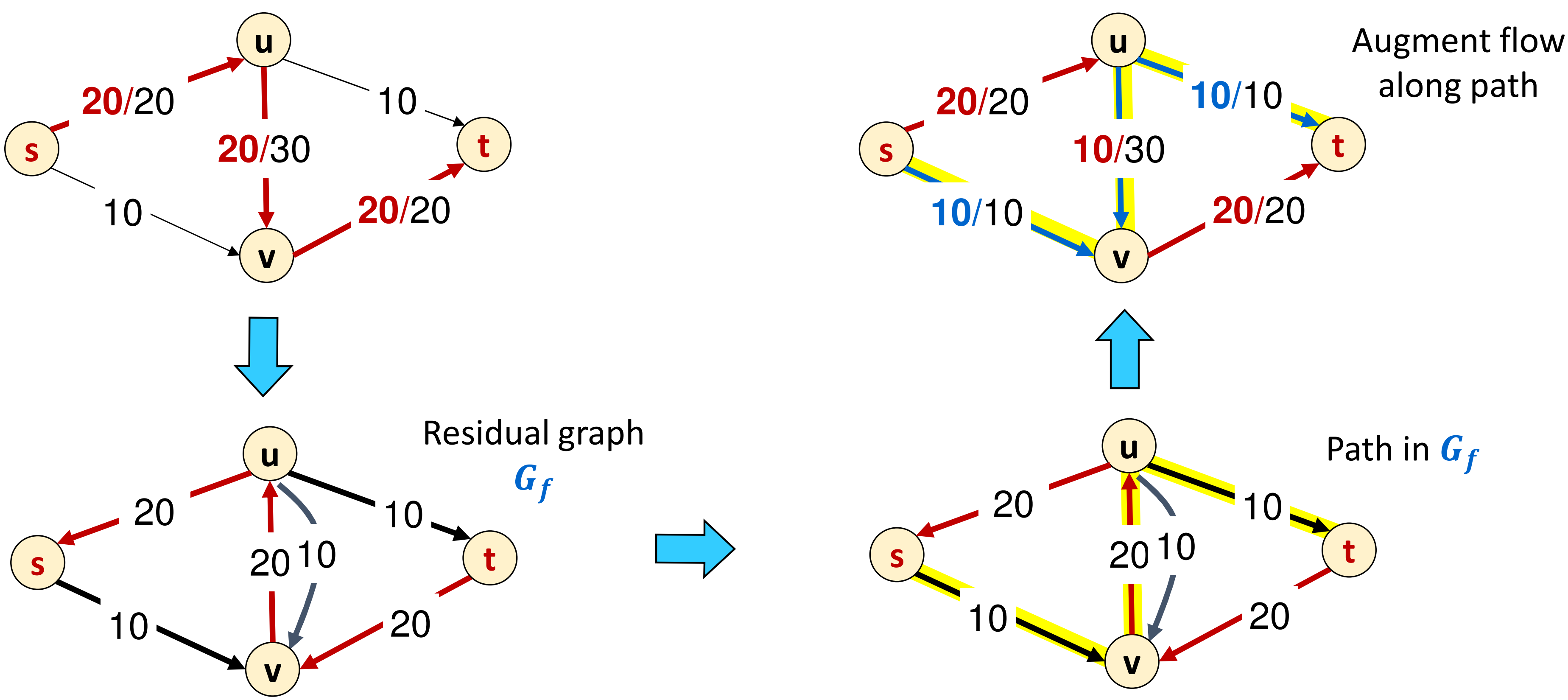
Suppose that we took this flow $\boldsymbol{f}$ as a baseline, what changes could each edge handle?

- We could add up to 10 units along **sv** or **ut** or **uv**
- We could reduce by up to 20 units from **su** or **uv** or **vt**

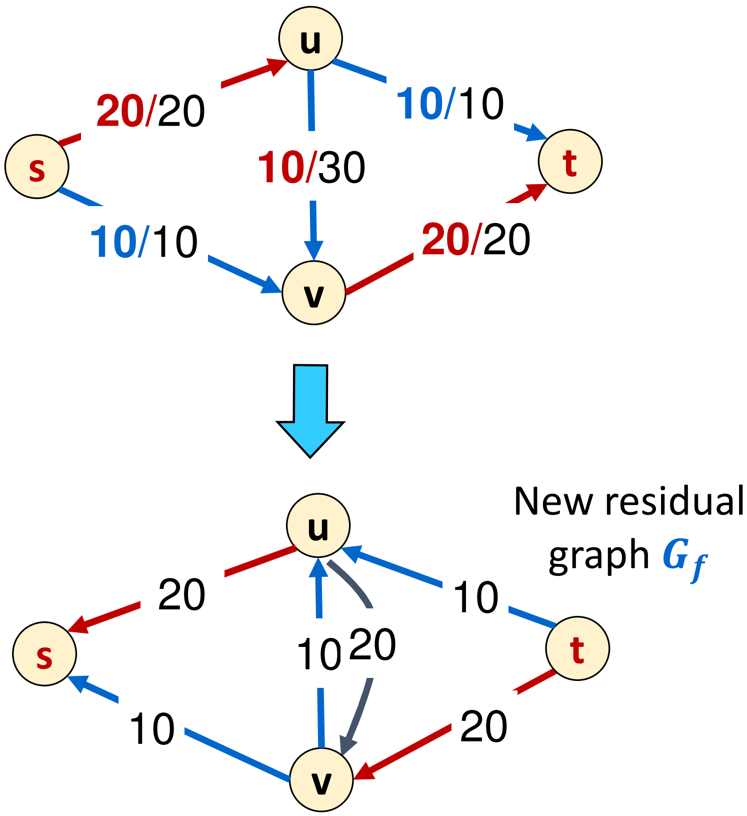
This gives us a residual graph $\boldsymbol{G_f}$ of possible changes where we draw reducing as "sending back".



Greed Revisited: Residual Graph & Augmenting Paths



Greed Revisited: Residual Graph & Augmenting Paths



No s-t path

BTW: Flow is optimal

Residual Graphs

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

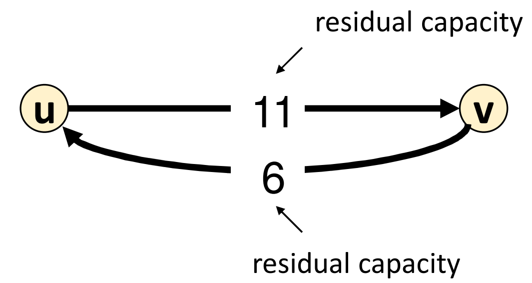
- Flow $f(e)$, capacity $c(e)$.

Residual edges of two kinds:

- Forward: $e = (u, v)$ with capacity $c_f(e) = c(e) - f(e)$
  - Amount of extra flow we can add along e
- Backward: $e^{\mathrm{R}} = (v, u)$ with capacity $c_f(e) = f(e)$
  - Amount we can reduce/undo flow along e

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

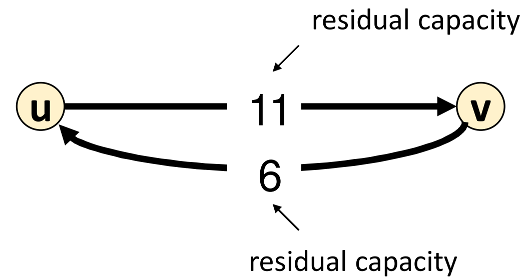
- Residual edges with residual capacity $c_f(e) > 0$.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^{\mathrm{R}} : f(e) > 0\}$.



Residual Graphs and Augmenting Paths

Residual edges of two kinds:

- Forward: $e = (u, v)$ with capacity $c_f(e) = c(e) - f(e)$
  - Amount of extra flow we can add along e
- Backward: $e^{\mathrm{R}} = (v, u)$ with capacity $c_f(e) = f(e)$
  - Amount we can reduce/undo flow along e



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

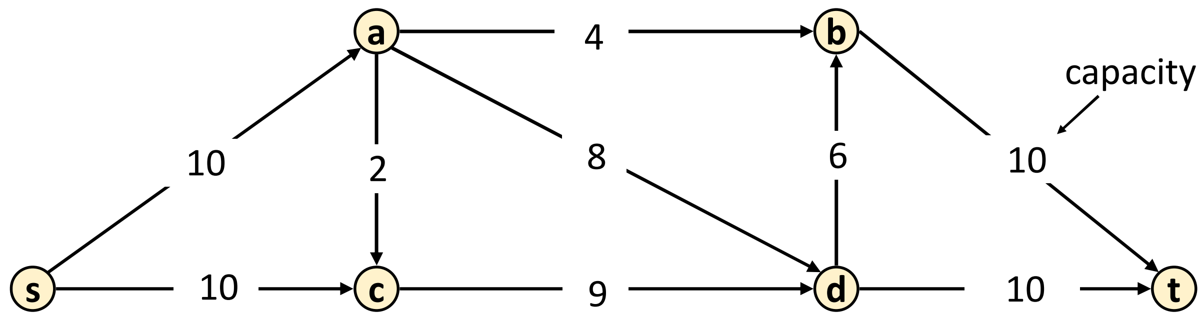
- Residual edges with residual capacity $c_f(e) > 0$.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^{\mathrm{R}}: f(e) > 0\}$.

Augmenting Path: Any s-t path P in G_f. Let **bottleneck**(P)= $\min_{e \in P} c_f(e)$.

Ford-Fulkerson idea: Repeat "find an augmenting path P and increase flow by **bottleneck**(P)" until none left.

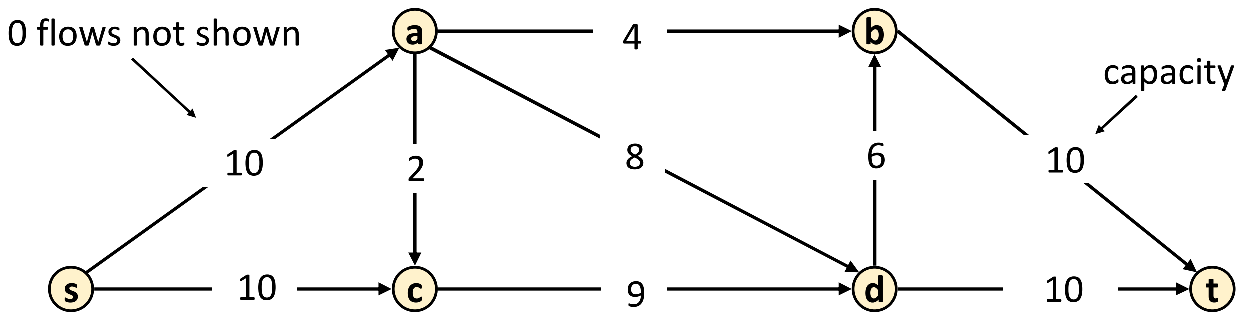
Ford-Fulkerson Algorithm

G:



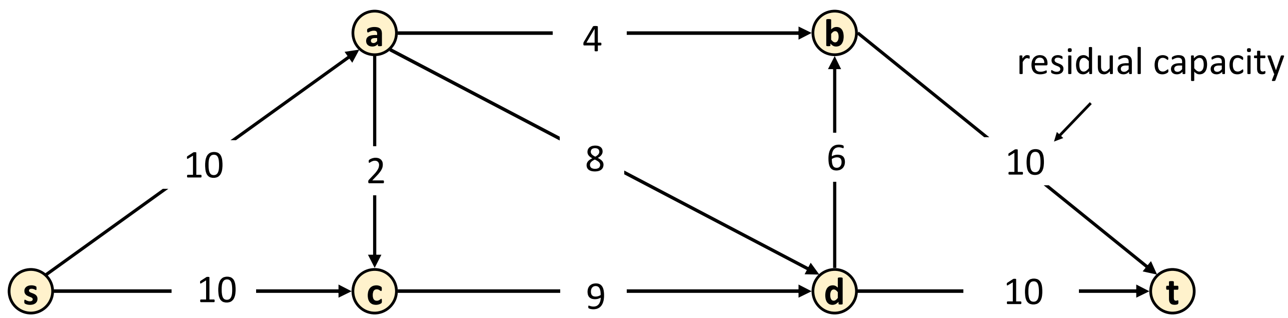
Ford-Fulkerson Algorithm

G:

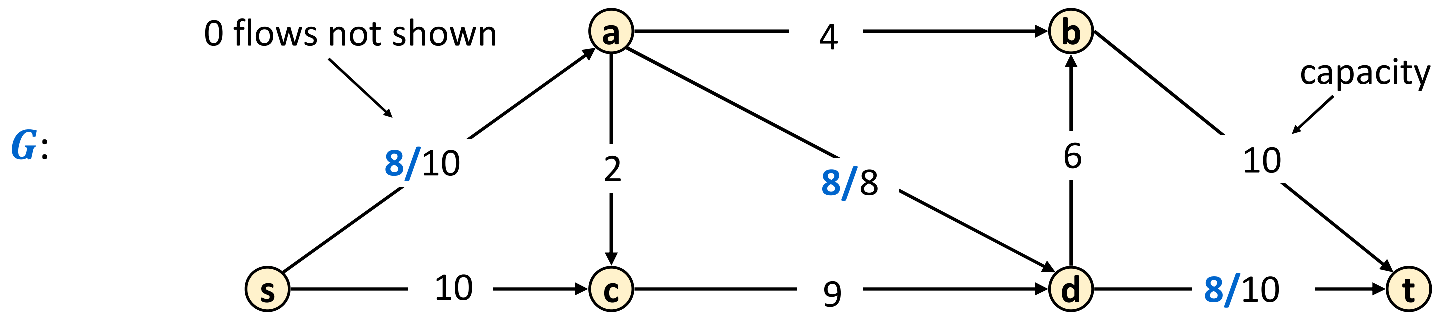


Flow value = **0**

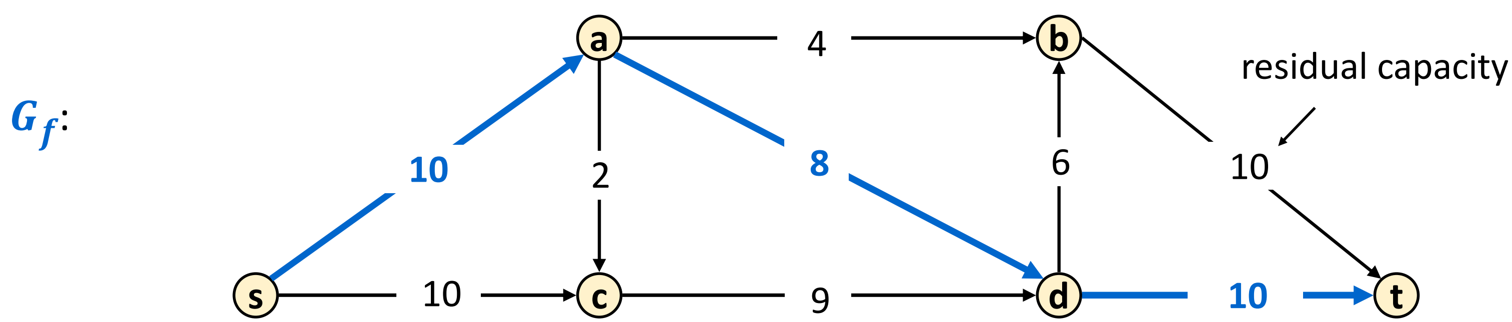
G_f:



Ford-Fulkerson Algorithm

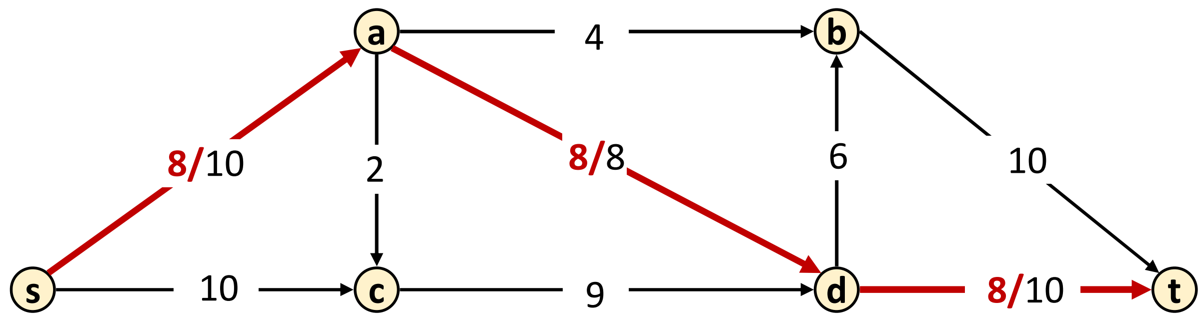


Flow value = 0 +8=8



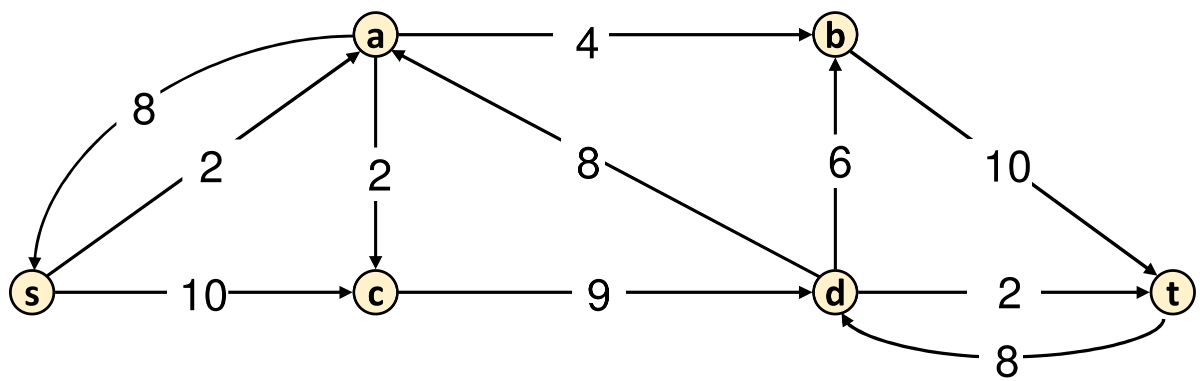
Ford-Fulkerson Algorithm

G:



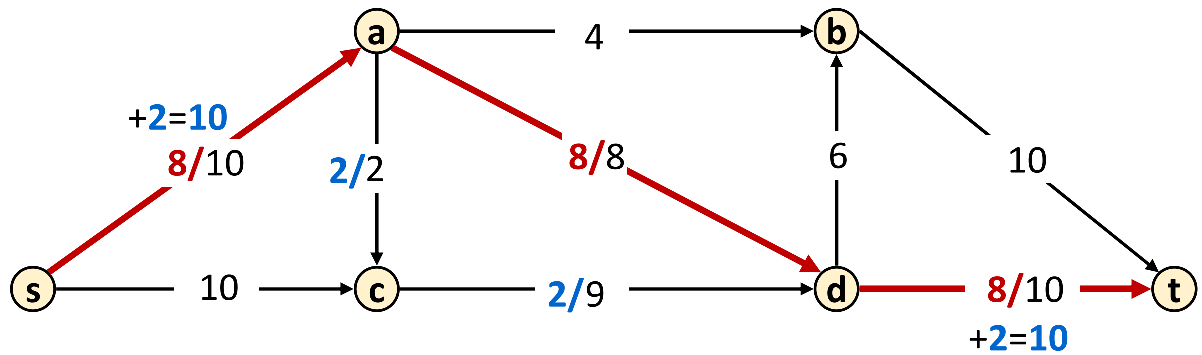
Flow value = **8**

G_f:



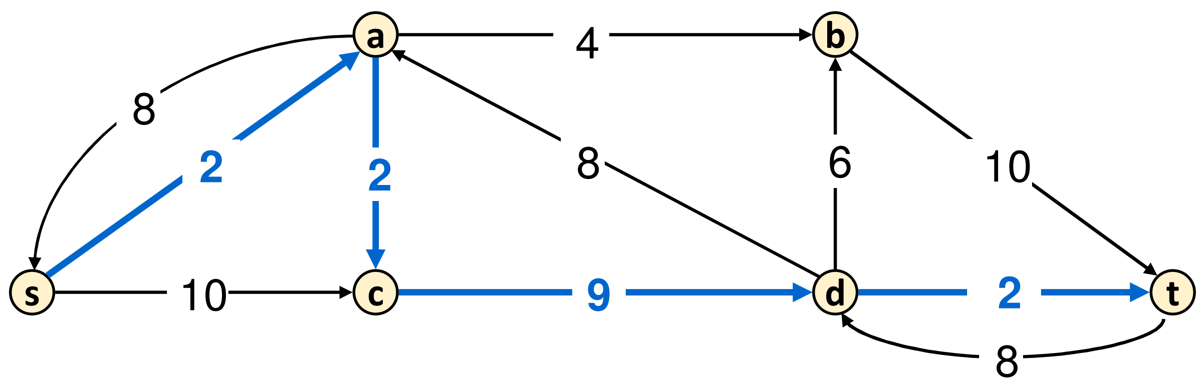
Ford-Fulkerson Algorithm

G:



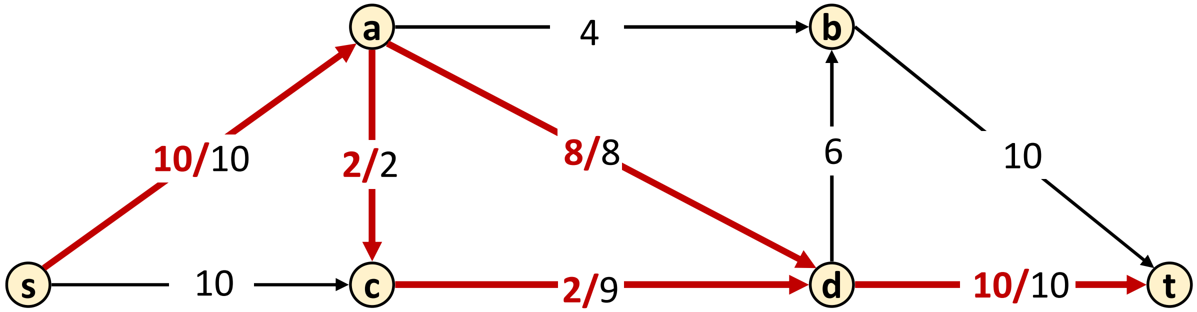
Flow value = 8
+2=10

G_f:



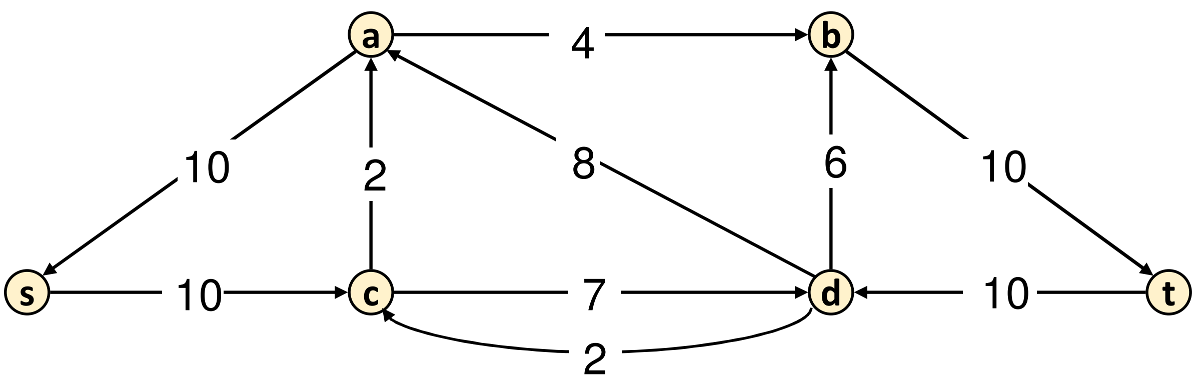
Ford-Fulkerson Algorithm

G:



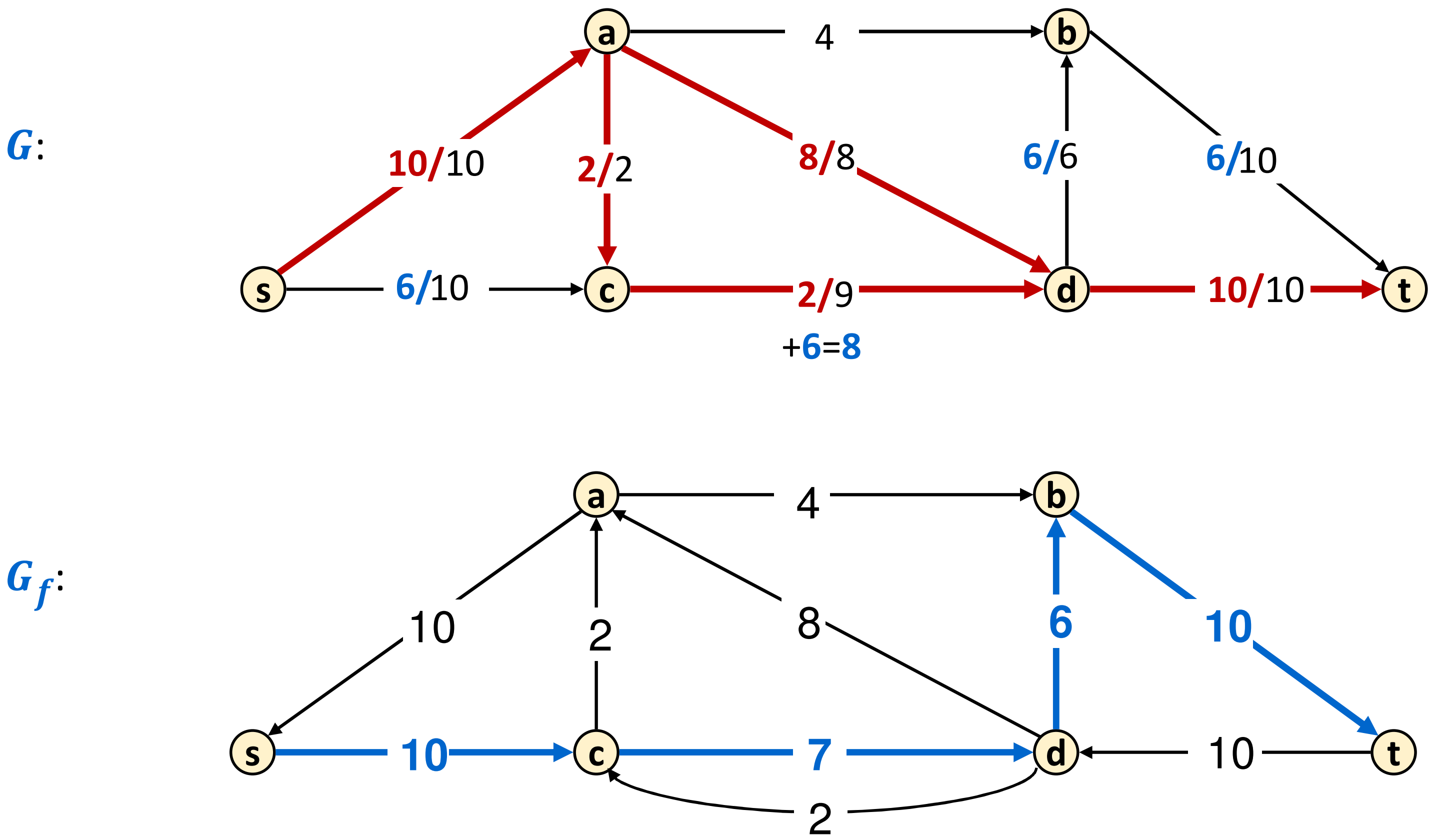
Flow value = **10**

G_f:



20

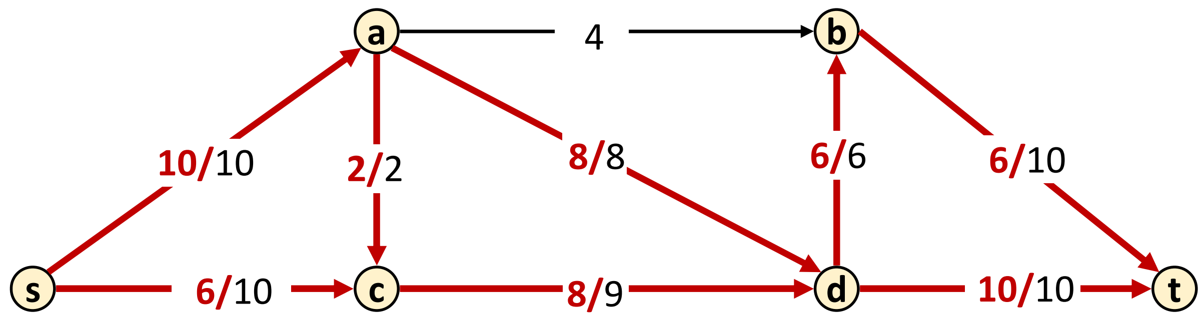
Ford-Fulkerson Algorithm



Flow value = 10
+6=16

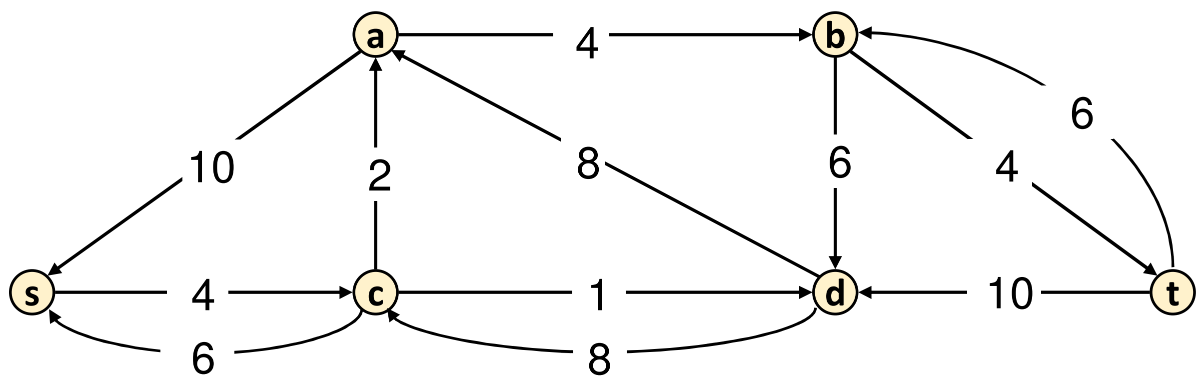
Ford-Fulkerson Algorithm

G:

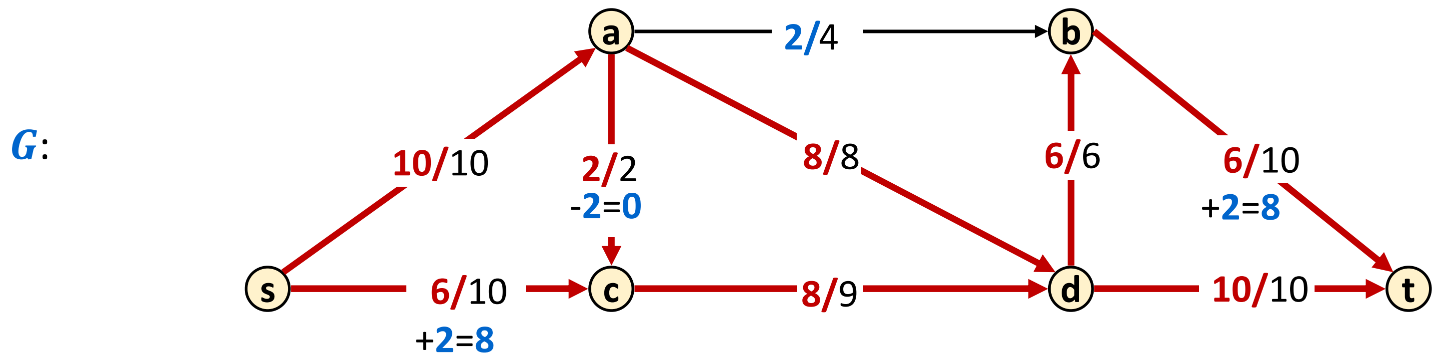


Flow value = **16**

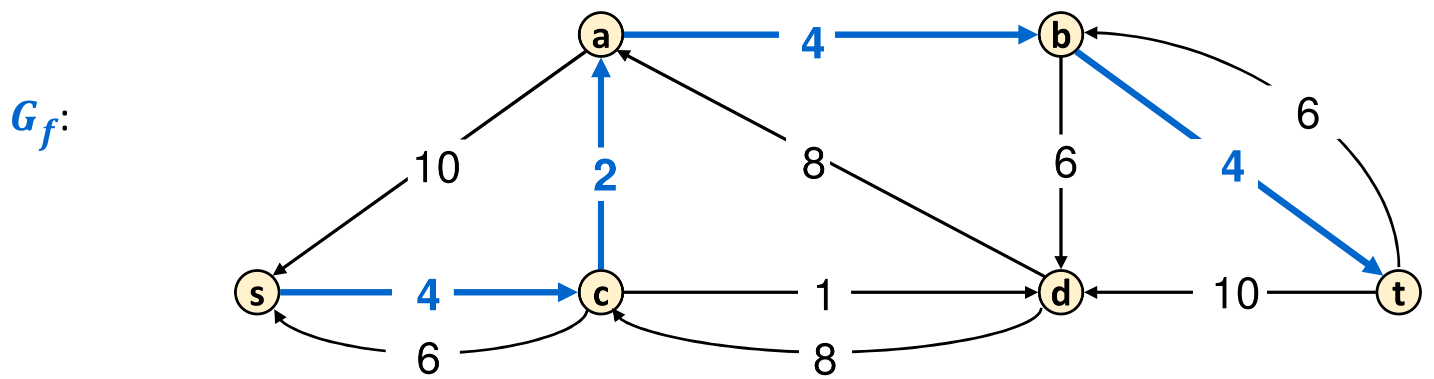
G_f:



Ford-Fulkerson Algorithm

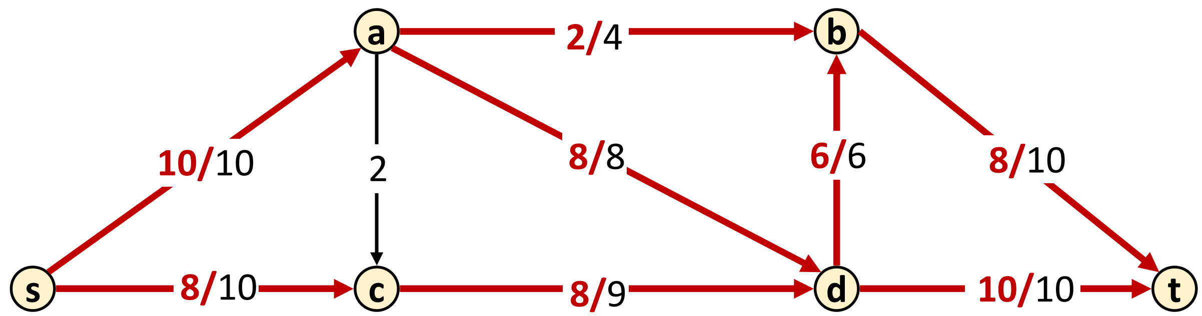


Flow value = 16
+2=18



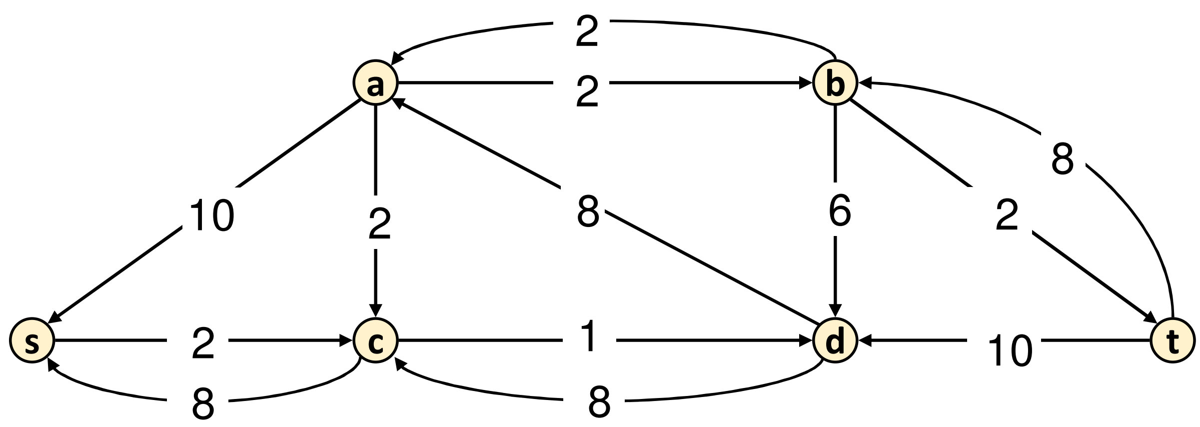
Ford-Fulkerson Algorithm

G:



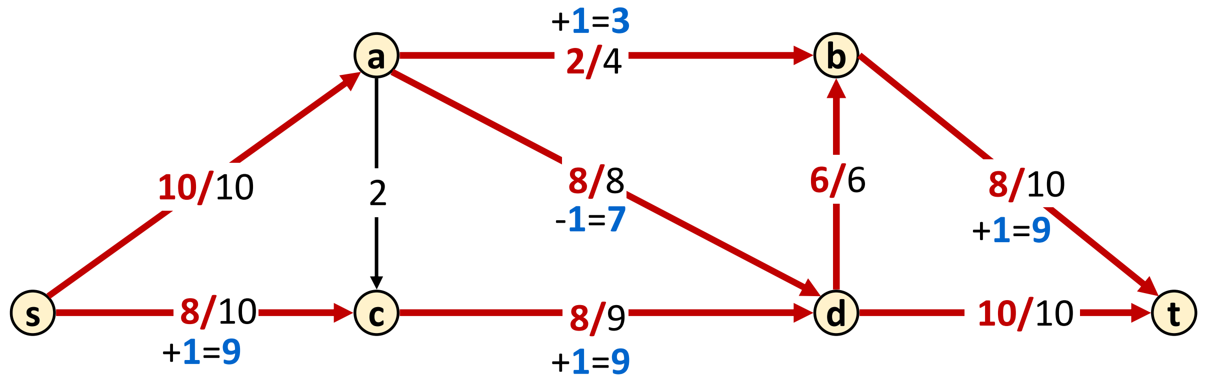
Flow value = **18**

G_f:



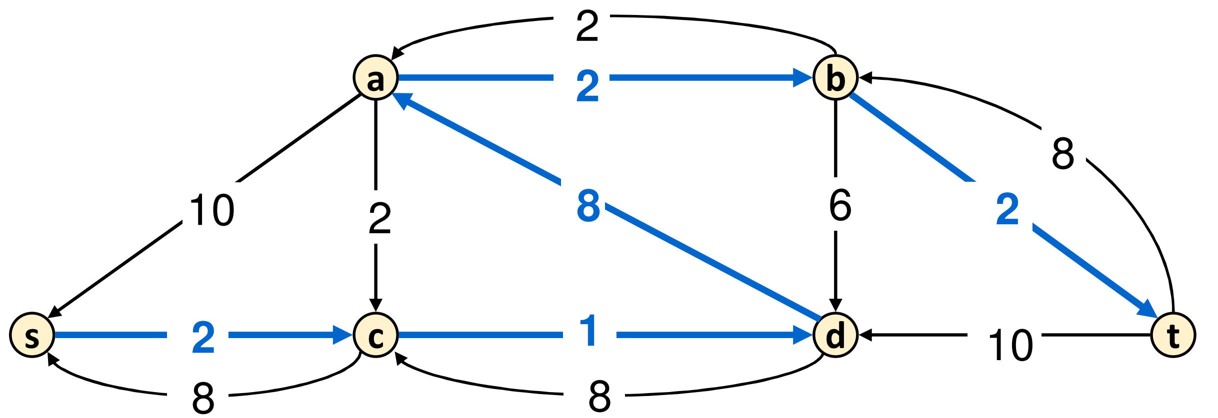
Ford-Fulkerson Algorithm

G:



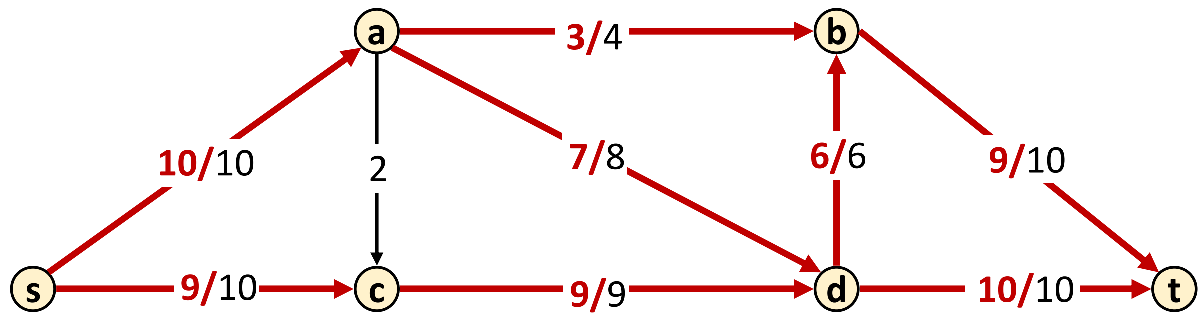
Flow value = 18
+1=19

G_f:



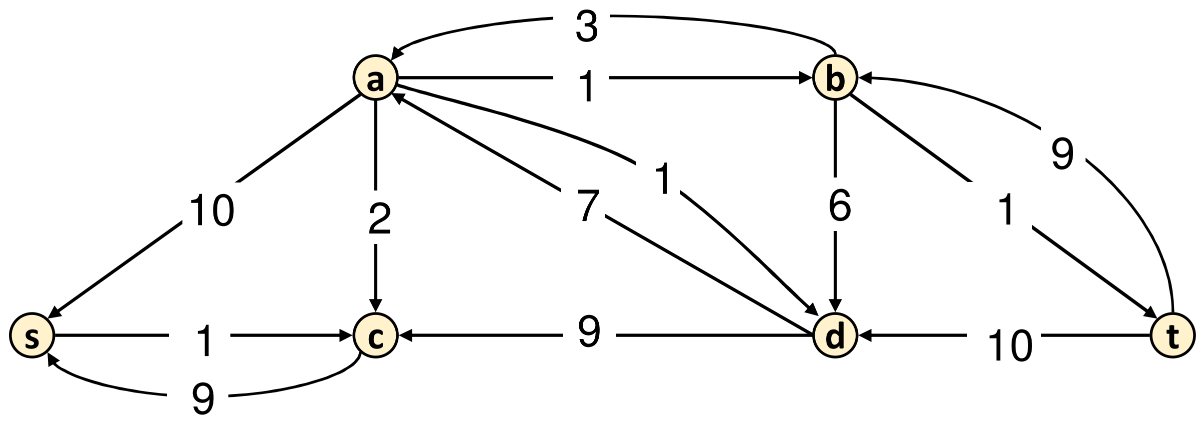
Ford-Fulkerson Algorithm

G:



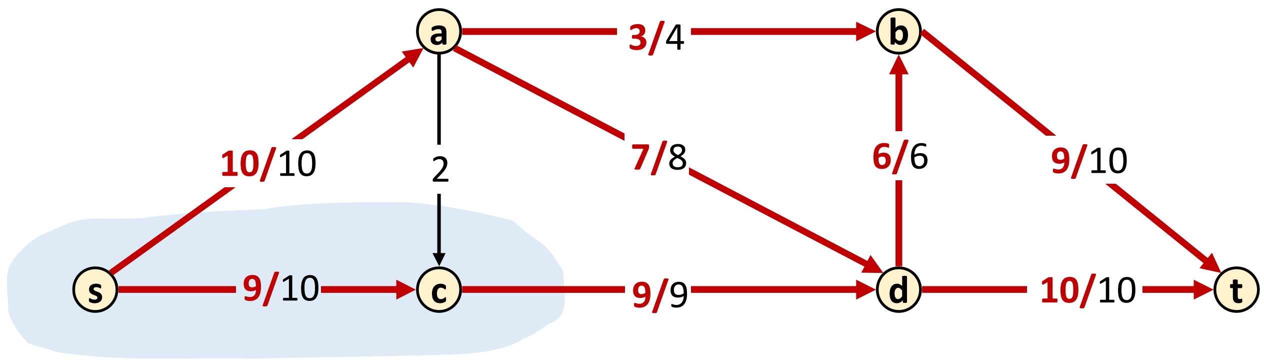
Flow value = **19**

G_f:



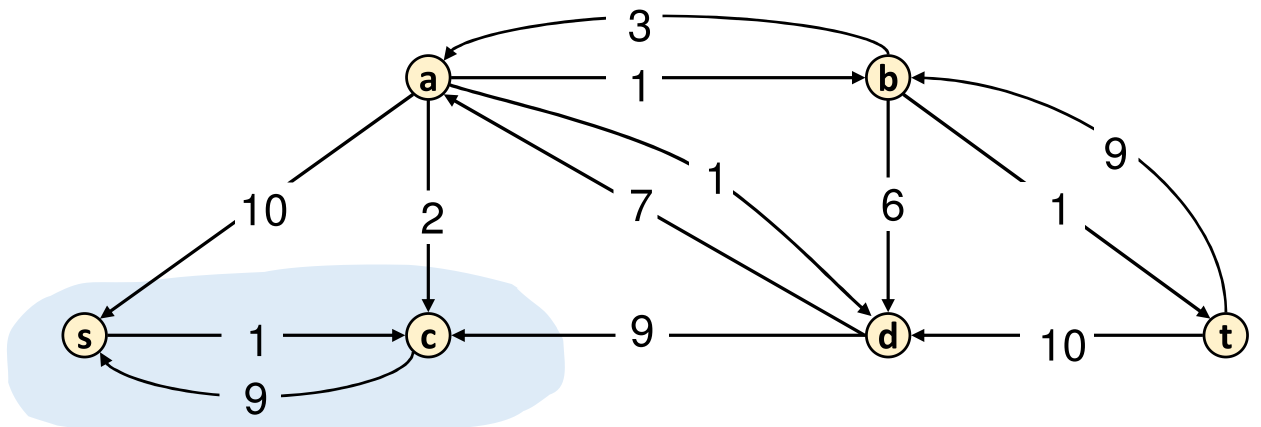
Ford-Fulkerson Algorithm

G:



Flow value = **19**

G_f:



Cut capacity = **19**

Augmenting Path Algorithm

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

   while (Gf has an s-t path P) {
      f ← Augment(f, c, P)
      update Gf
   }
   return f
}
```

```
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
}
```

Max-Flow Min-Cut Theorem

Augmenting Path Theorem: Flow f is a max flow $\Leftrightarrow$ there are no augmenting paths wrt f

Max-Flow Min-Cut Theorem: The value of the max flow equals the value of the min cut.
[Elias-Feinstein-Shannon 1956, Ford-Fulkerson 1956] **"Maxflow = Mincut"**

Proof: We prove both together by showing that all of these are equivalent:

(i) There is a cut (A, B) such that $v(f) = c(A, B)$.

(ii) Flow f is a max flow.

(iii) There is no augmenting path w.r.t. f.

Only (iii) $\Rightarrow$ (i) remaining

(i) $\Rightarrow$ (ii): We already know this by the corollary to weak duality lemma.

(ii) $\Rightarrow$ (iii): (by contradiction)
If there is an augmenting path w.r.t. flow f then we can improve f. Therefore f is not a max flow.

Proof of Max-Flow Min-Cut Theorem

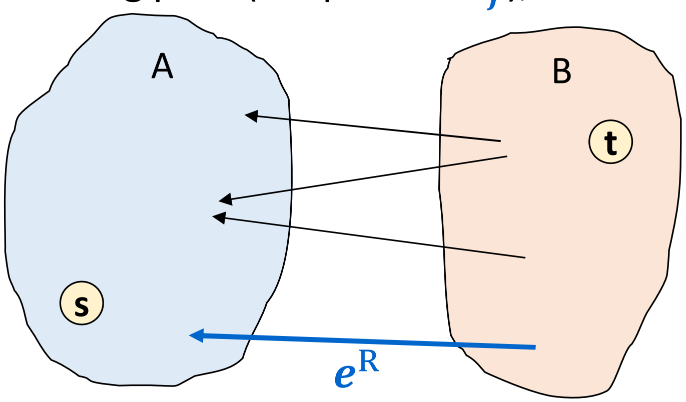
(iii) ⇒ (i):

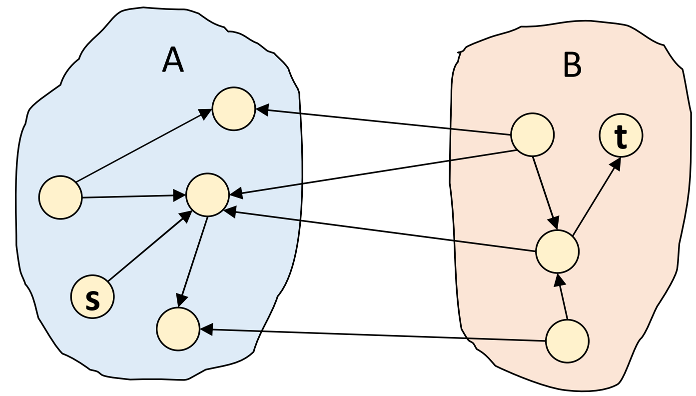
Claim: If there is no augmenting path w.r.t. f, there is a cut (A, B) s.t. $v(f) = c(A, B)$.

Proof of Claim: Let f be a flow with no augmenting paths.

Let A be the set of vertices reachable from s in residual graph G_f.

- By definition of A, $s \in A$.
- Since no augmenting path (s-t path in G_f), $t \notin A$.





residual graph

Proof of Max-Flow Min-Cut Theorem

(iii) ⇒ (i):

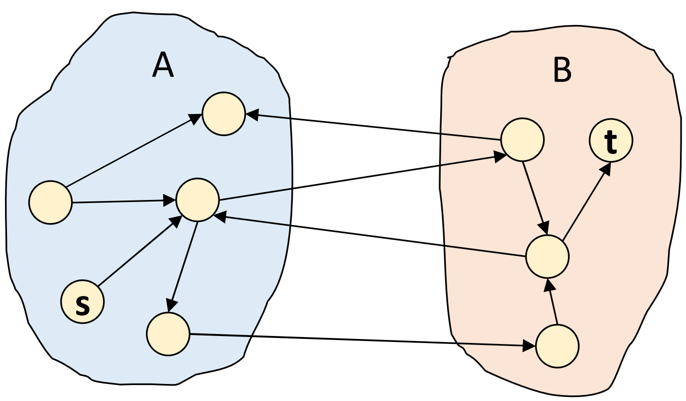
Claim: If there is no augmenting path w.r.t. f, there is a cut (A, B) s.t. $v(f) = c(A, B)$.

Proof of Claim: Let f be a flow with no augmenting paths.

Let A be the set of vertices reachable from s in residual graph G_f.

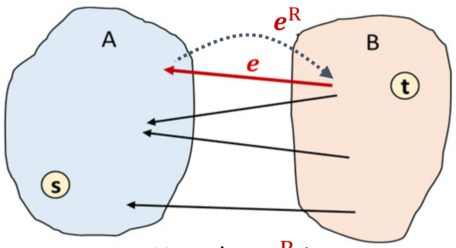
- By definition of A, $s \in A$.
- Since no augmenting path (s-t path in G_f), $t \notin A$.

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



original network

Proof of Max-Flow Min-Cut Theorem



No edge e^{R} in residual graph

(iii) ⇒ (i):

Claim: If there is no augmenting path w.r.t. f, there is a cut (A, B) s.t. $v(f) = c(A, B)$.

Proof of Claim: Let f be a flow with no augmenting paths.

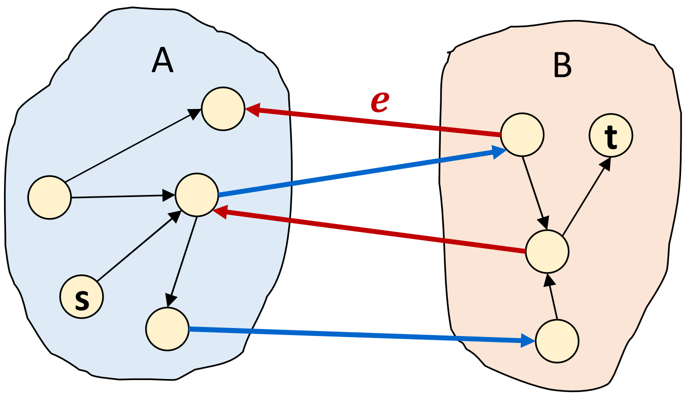
Let A be the set of vertices reachable from s in residual graph G_f.

- By definition of A, $s \in A$.
- Since no augmenting path (s-t path in G_f), $t \notin A$.

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

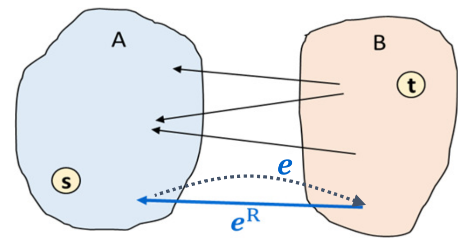
So no flow on e

$$f(e) = c_f(e^{\mathrm{R}}) = 0$$



original network

Proof of Max-Flow Min-Cut Theorem



No edge e in residual graph

(iii) ⇒ (i):

Claim: If there is no augmenting path w.r.t. f, there is a cut (A, B) s.t. $v(f) = c(A, B)$.

Proof of Claim: Let f be a flow with no augmenting paths.

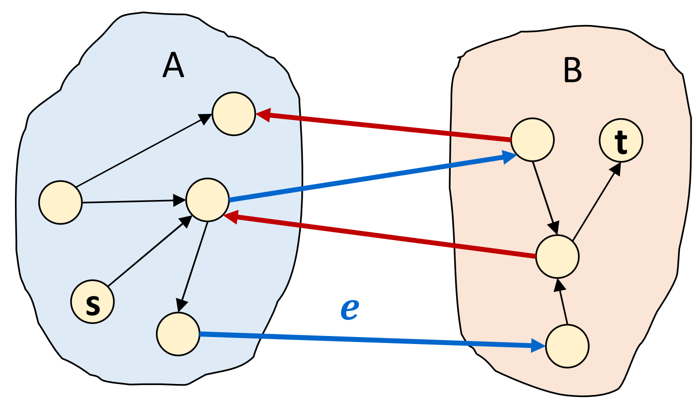
Let A be the set of vertices reachable from s in residual graph G_f.

- By definition of A, $s \in A$.
- Since no augmenting path (s-t path in G_f), $t \notin A$.

No unused capacity on e

$$0 = c_f(e) = c(e) - f(e)$$

Then
$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$$
$$= \sum_{e \text{ out of } A} f(e)$$
$$= \sum_{e \text{ out of } A} c(e)$$



original network

$$f(e) = c(e)$$

"e is saturated"

Proof of Max-Flow Min-Cut Theorem

(iii) ⇒ (i):

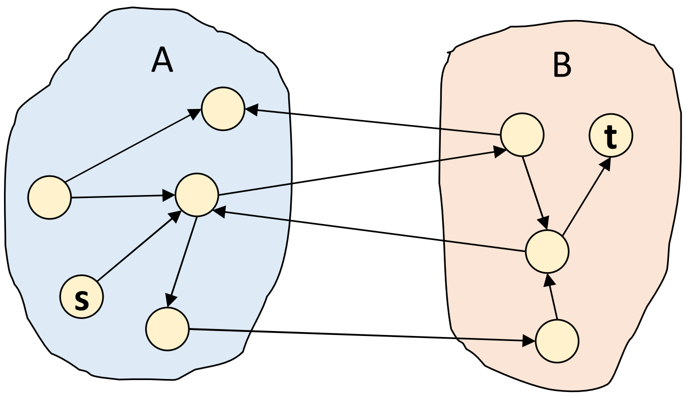
Claim: If there is no augmenting path w.r.t. f, there is a cut (A, B) s.t. $v(f) = c(A, B)$.

Proof of Claim: Let f be a flow with no augmenting paths.

Let A be the set of vertices reachable from s in residual graph G_f.

- By definition of A, $s \in A$.
- Since no augmenting path (s-t path in G_f), $t \notin A$.

Then
$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$$
$$= \sum_{e \text{ out of } A} f(e)$$
$$= \sum_{e \text{ out of } A} c(e) \quad = c(A, B)$$
■



original network

Running Time

- Computing first G_f takes $O(n+m)$ time. (Can ignore disconnected bits so $m \geq n-1$.)
- Finding each augmenting path (graph search in G_f) takes $O(m)$ time.
- Updating f and G_f takes $O(n)$ time.

Total $O(m)$ per iteration.

Assumption: All capacities are integers between 1 and C.

Ford-Fulkerson Invariant: Every flow value $f(e)$ and every residual capacity $c_f(e)$ remains an integer throughout the algorithm. So there is a maximum flow with only integer flows.

Theorem: The Ford-Fulkerson algorithm terminates in $\leq$ Maxflow $< nC$ iterations.

Proof: Capacity of cut with $A = \{s\}$ is $\leq (n-1)C$. Each augmentation increases flow value by at least 1. ■

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

Bipartite Matching

A graph $G = (V, E)$ is bipartite iff

- Set V of vertices has two disjoint parts X and Y
- Every edge in E joins a vertex from X and a vertex from Y

Set $M \subseteq E$ is a matching in G iff no two edges in M share a vertex

Goal: Find a matching M in G of maximum size.

Differences from stable matching

- limited set of possible partners for each vertex
- sides may not be the same size
- no notion of stability; matching everything may be impossible.

Bipartite Matching

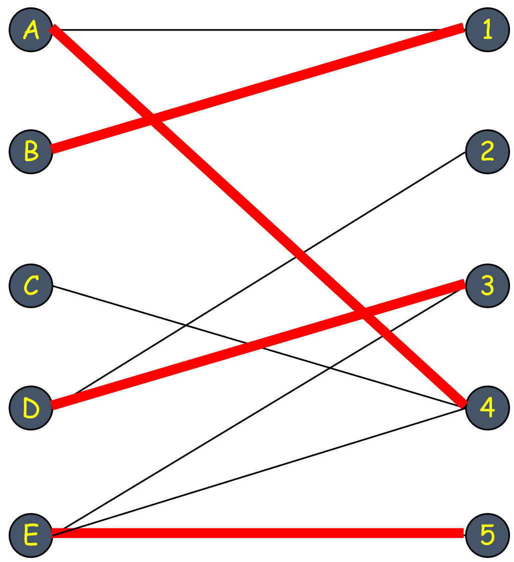
- Models assignment problems
  - $\boldsymbol{X}$ represents customers, $\boldsymbol{Y}$ represents salespeople
  - $\boldsymbol{X}$ represents professors, $\boldsymbol{Y}$ represents courses

- If $|\boldsymbol{X}| = |\boldsymbol{Y}| = \boldsymbol{n}$
  - $\boldsymbol{G}$ has perfect matching iff maximum matching has size $\boldsymbol{n}$

Bipartite Matching

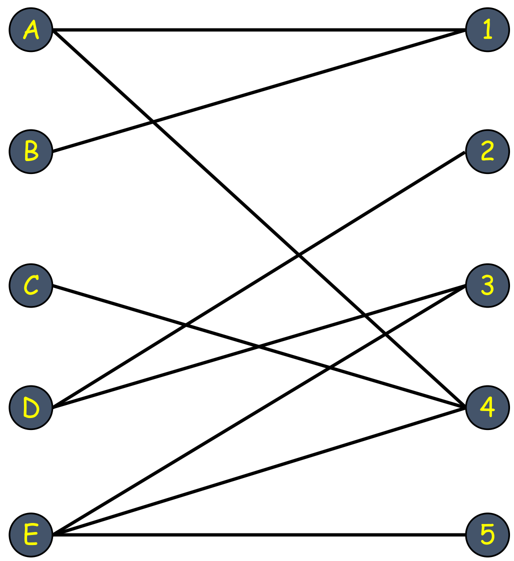
Input: Bipartite graph

Goal: Find maximum size matching.



Bipartite Matching as a special case of Flow

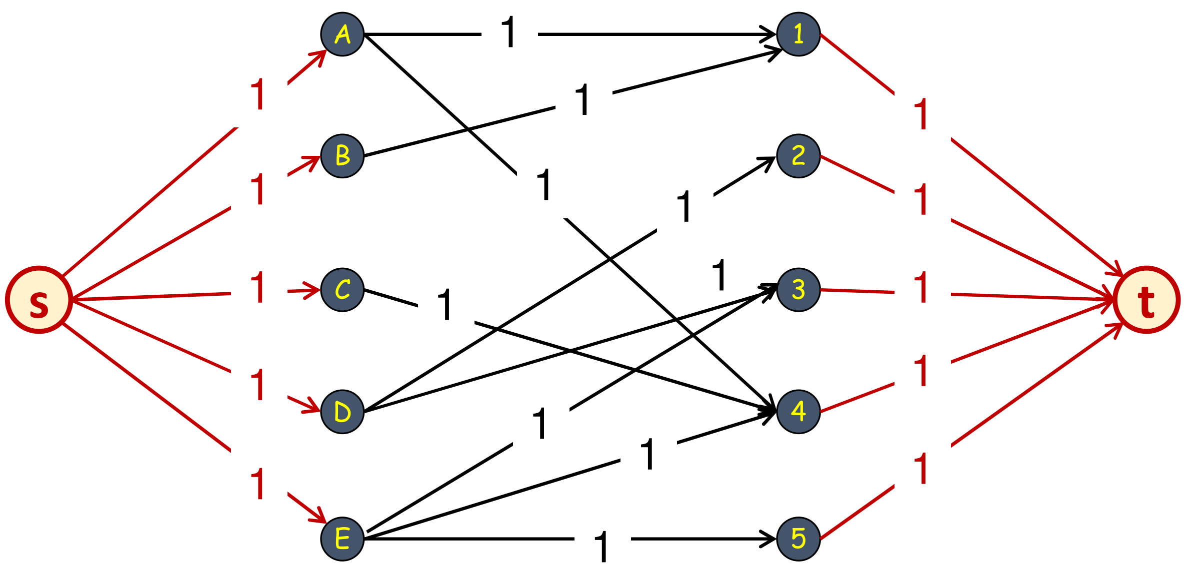
Input: Bipartite graph



Bipartite Matching as a special case of Flow

Add new source **s** pointing to left set, new sink **t** pointed to by right set.

Direct all edges from left to right with capacity 1. Compute MaxFlow.



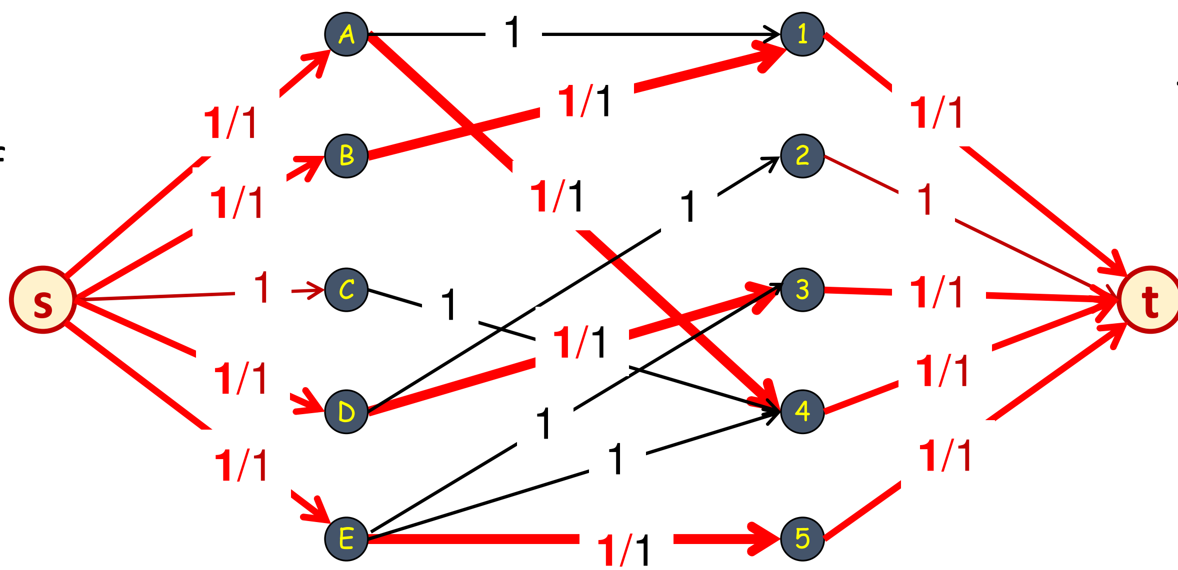
Bipartite Matching as a special case of Flow

Add new source **s** pointing to left set, new sink **t** pointed to by right set.

Direct all edges from left to right with capacity 1. Compute MaxFlow.

Correctness:
Integer flow just gives a subset of edges.

Source and sink edges imply it is a matching

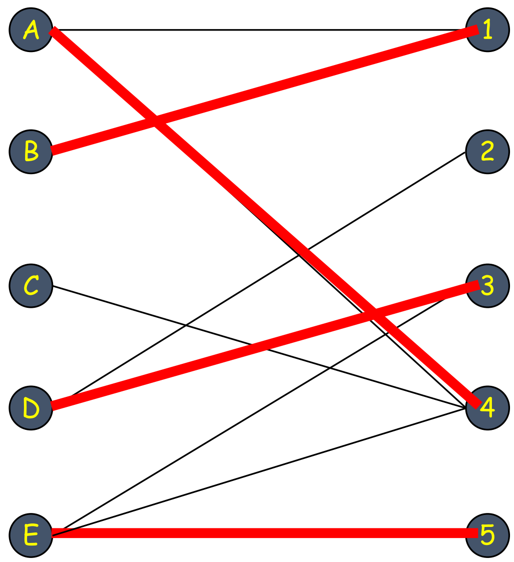


Time $O(mn)$

Bipartite Matching

Input: Bipartite graph

Goal: Find maximum size matching.



Optimality

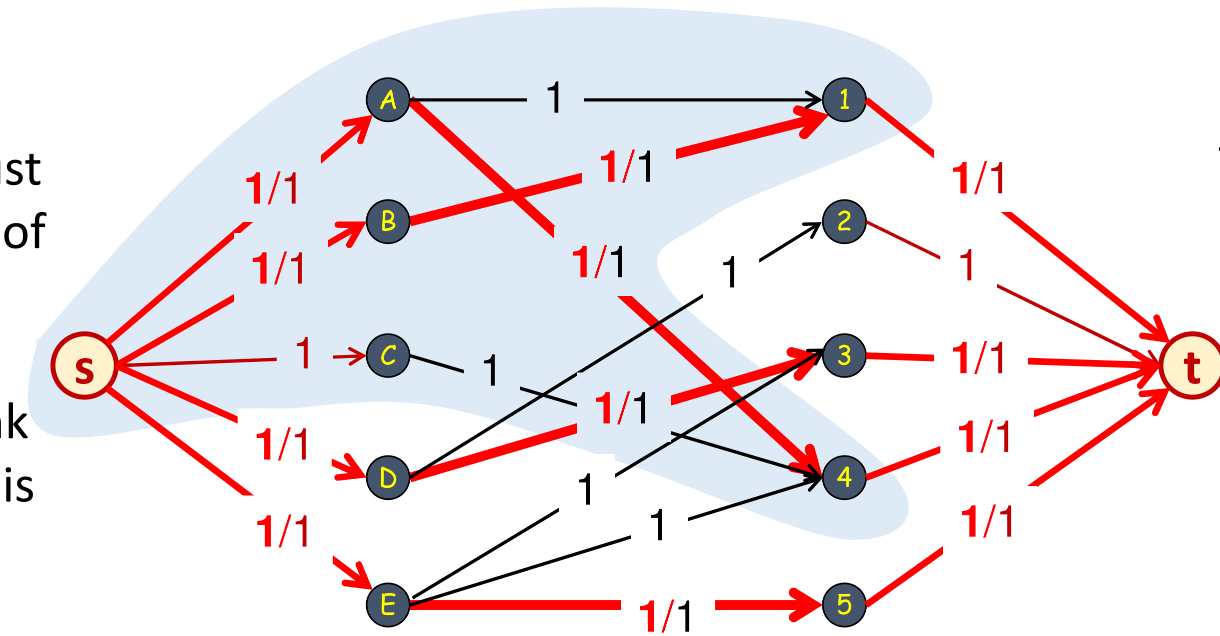
Bipartite Matching as a special case of Flow

Add new source **s** pointing to left set, new sink **t** pointed to by right set.

Direct all edges from left to right with capacity 1. Compute MaxFlow.

Correctness:
Integer flow just gives a subset of edges.

Source and sink edges imply it is a matching



Time $O(mn)$

Optimality

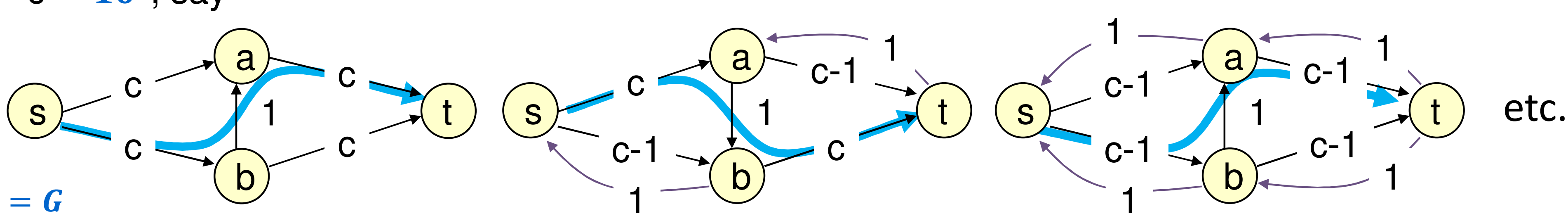
Ford-Fulkerson Efficiency

Worst case runtime $O(mnC)$ with integer capacities $\leq C$.

- $O(m)$ time per iteration.
- At most nC iterations.
- This is “pseudo-polynomial” running time.

- May take exponential time, even with integer capacities:

c $= 10^9$, say



$G_f = G$