CSE 421: Introduction to Algorithms

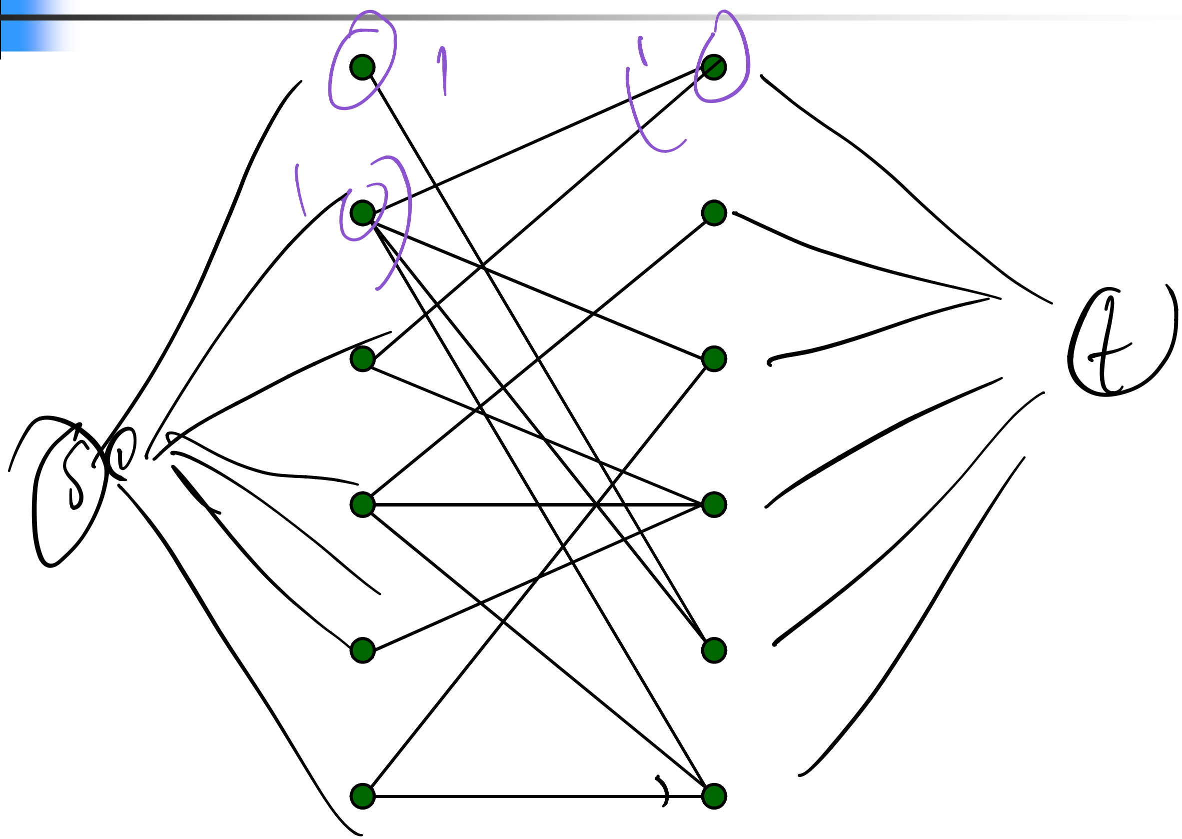
Network Flow

Paul Beame

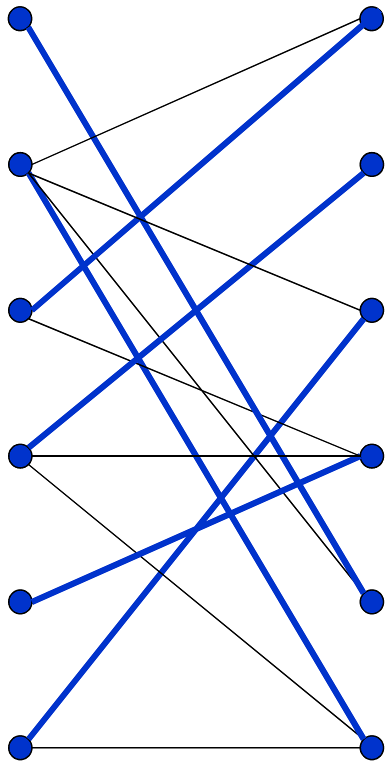
Bipartite Matching

- **Given:** A bipartite graph $G=(V,E)$
  - $M \subseteq E$ is a matching in **G** iff no two edges in **M** share a vertex
- **Goal:** Find a matching **M** in **G** of maximum possible size

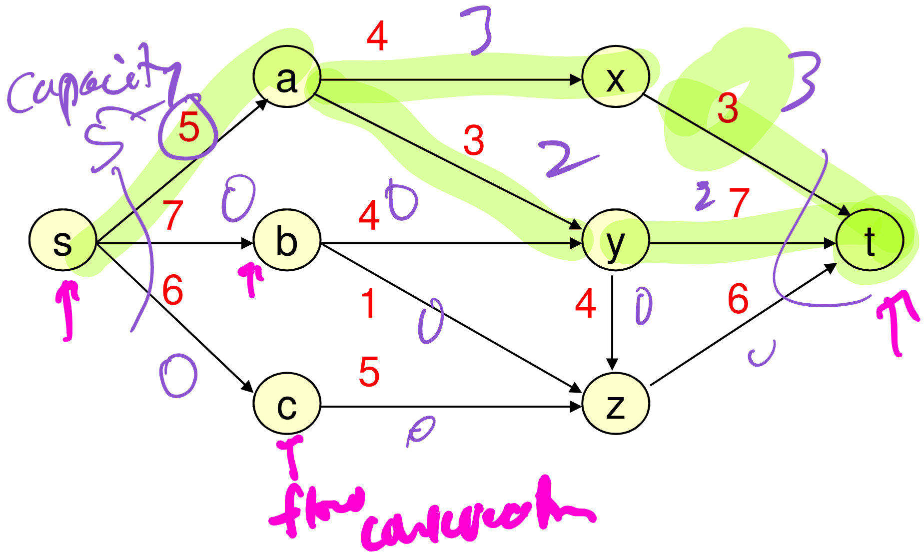
Bipartite Matching



Bipartite Matching



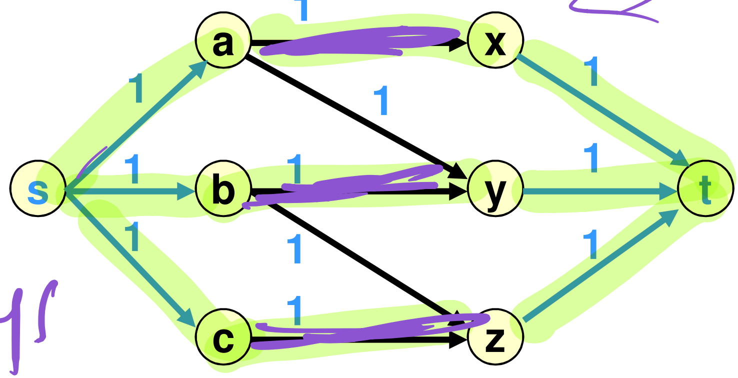
The Network Flow Problem



- How much stuff can flow from s to t?

Bipartite matching as a special case of flow

Suppose maximum size M is a matching of



Can always send at flow at least |M|

original graph direct edge from L to R

Net Flow: Formal Definition

Given:

A digraph $G = (V,E)$

Two vertices s,t in V (source & sink)

A *capacity* $c(u,v) \geq 0$ for each $(u,v) \in E$ (and $c(u,v) = 0$ for all non-edges (u,v))

Find:

A *flow function* $f: E \rightarrow R$ s.t., for all u,v:

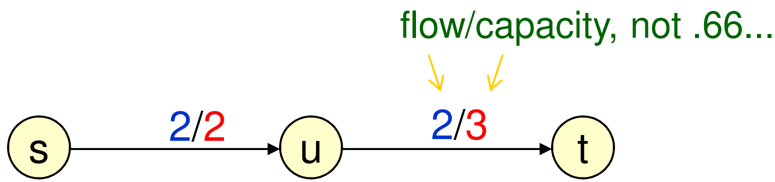
- $0 \leq f(u,v) \leq c(u,v)$ [Capacity Constraint]
- if $u \neq s,t$, i.e. $f^{out}(u)=f^{in}(u)$ [Flow Conservation]

Maximizing total flow $\nu(f) = f^{out}(s)$

Notation:

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

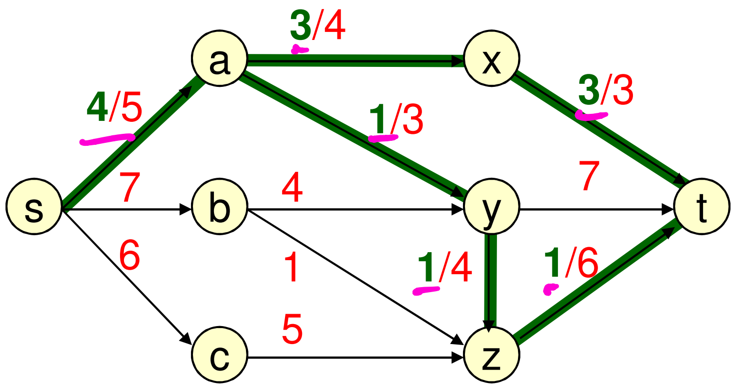
Example: A Flow Function



$$f^{in}(u)=f(s,u)=2=f(u,t)=f^{out}(u)$$

Example: A Flow Function

on every edge
f = 0
always legal



- Not shown: f(u,v) if = 0
- Note: max flow ≥ 4 since
  f is a flow function, with $\nu(\mathbf{f}) = 4$

Max Flow via a Greedy Alg?

While there is an $s \rightarrow t$ path in G

- Pick such a path, **p**
- Find **c**, the min capacity of any edge in **p**
- Count **c** towards the flow value
- Subtract **c** from all capacities on **p**
- Delete edges of capacity **0**

Max Flow via a Greedy Alg?

While there is an s → t path in G

Pick such a path, **p**

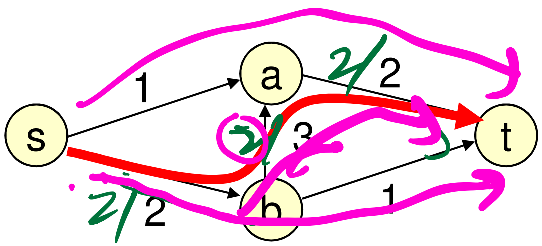
Find **c**, the min capacity of any edge in **p**

Count **c** towards the flow value

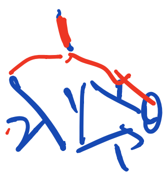
Subtract **c** from all capacities on **p**

Delete edges of capacity **0**

- This does NOT always find a max flow:



If pick s →b →a →t first, flow stuck at 2.
But flow 3 possible.



A Brief History of Flow

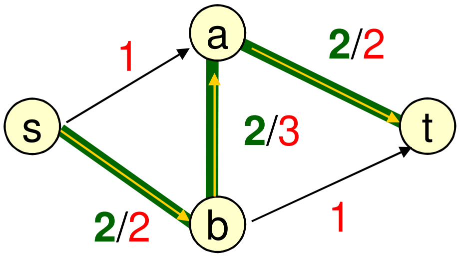
#	year	discoverer(s)	bound
1	1951	Dantzig	$O(n^2mU)$
2	1955	Ford & Fulkerson	$O(nmU)$
3	1970	Dinitz Edmonds & Karp	$O(nm^2)$
4	1970	Dinitz	$O(n^2m)$
5	1972	Edmonds & Karp Dinitz	$O(m^2 \log U)$
6	1973	Dinitz Gabow	$O(nm \log U)$
7	1974	Karzanov	$O(n^3)$
8	1977	Cherkassky	$O(n^2\sqrt{m})$
9	1980	Galil & Naamad	$O(nm \log^2 n)$
10	1983	Sleator & Tarjan	$O(nm \log n)$
11	1986	Goldberg & Tarjan	$O(nm \log(n^2/m))$
12	1987	Ahuja & Orlin	$O(nm + n^2 \log U)$
13	1987	Ahuja et al.	$O(nm \log(n\sqrt{\log U}/(m+2)))$
14	1989	Cheriyan & Hagerup	$E(nm + n^2 \log^2 n)$
15	1990	Cheriyan et al.	$O(n^3/\log n)$
16	1990	Alon	$O(nm + n^{8/3} \log n)$
17	1992	King et al.	$O(nm + n^{2+\epsilon})$
18	1993	Phillips & Westbrook	$O(nm(\log_{m/n} n + \log^{2+\epsilon} n))$
19	1994	King et al.	$O(nm \log_{m/(n \log n)} n)$
20	1997	Goldberg & Rao	$O(m^{3/2} \log(n^2/m) \log U)$ $O(n^{2/3} m \log(n^2/m) \log U)$
	2012	Orlin + King et al.	$O(nm)$

n = # of vertices
m= # of edges
U = Max capacity

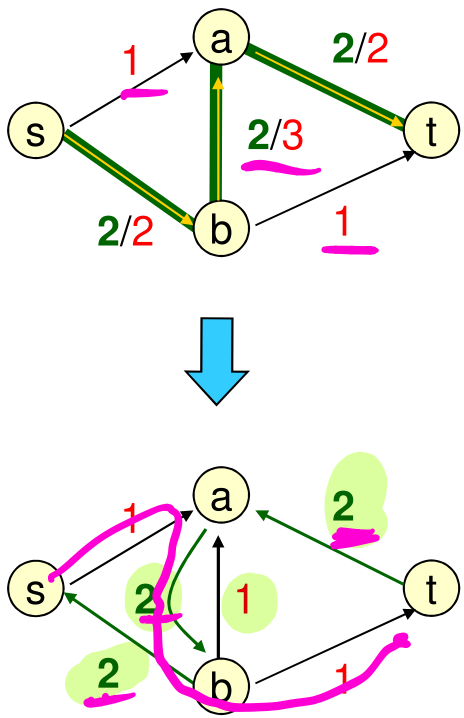
Source: Goldberg & Rao, FOCS '97

Yin Tat Lee ... even better

Greed Revisited: Residual Graph & Augmenting Path

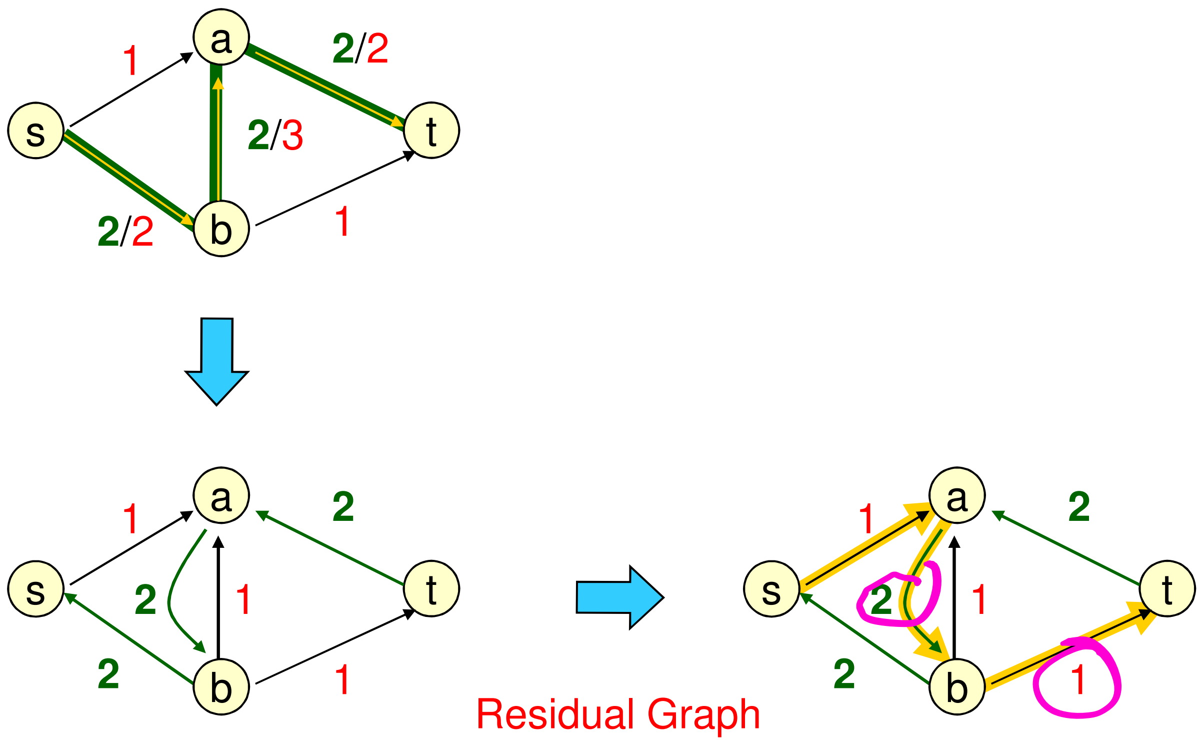


Greed Revisited: Residual Graph & Augmenting Path

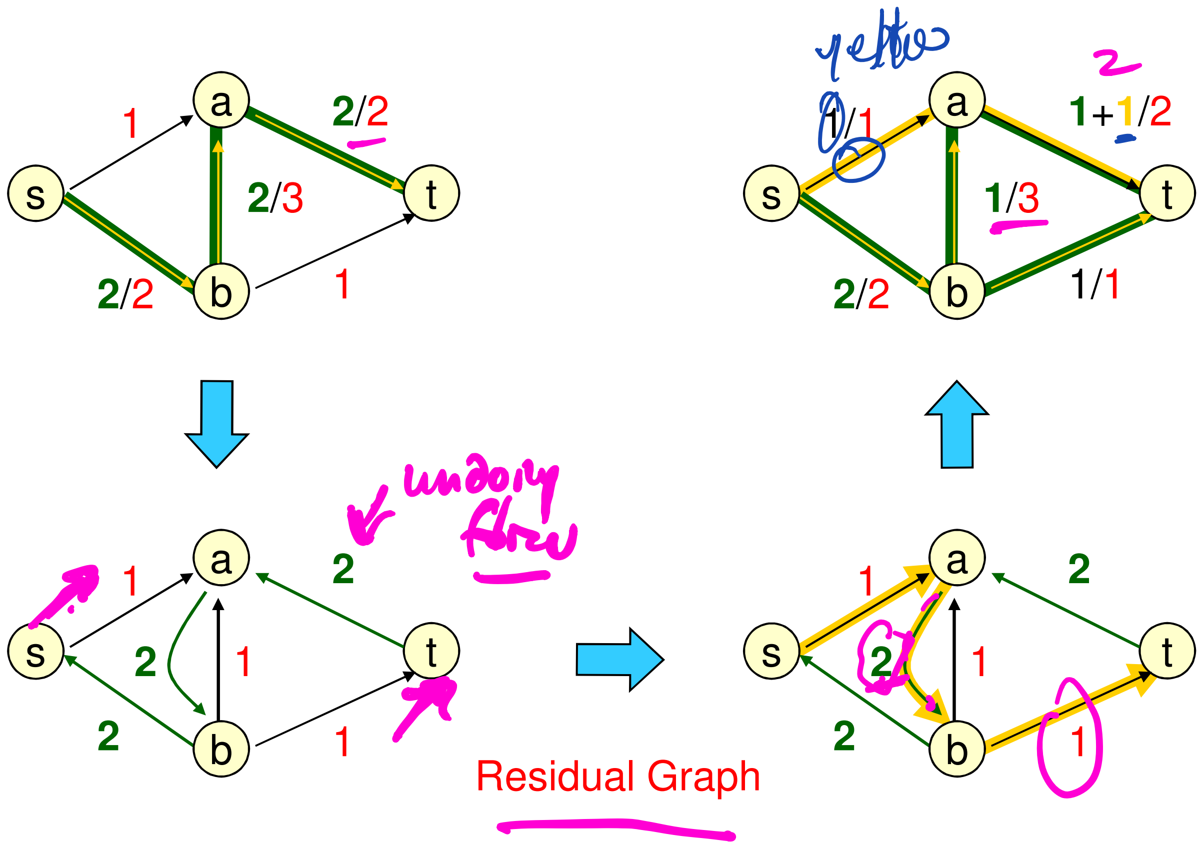


Residual Graph

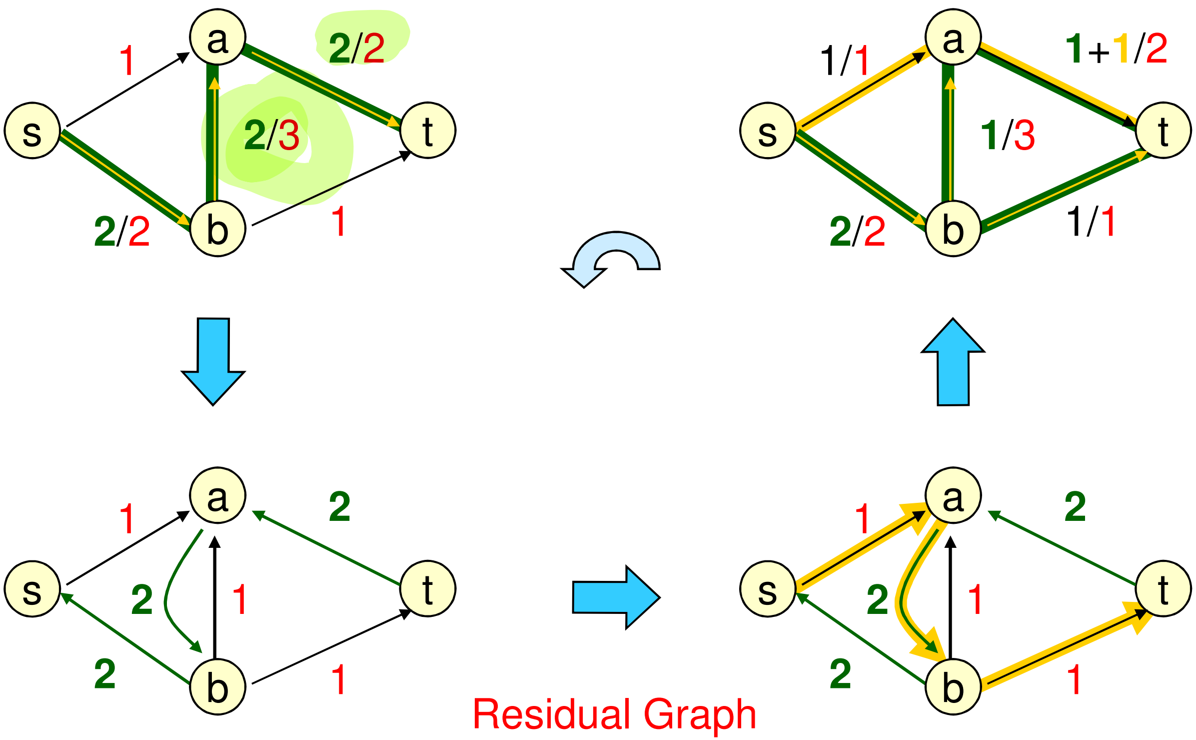
Greed Revisited:
Residual Graph & Augmenting Path



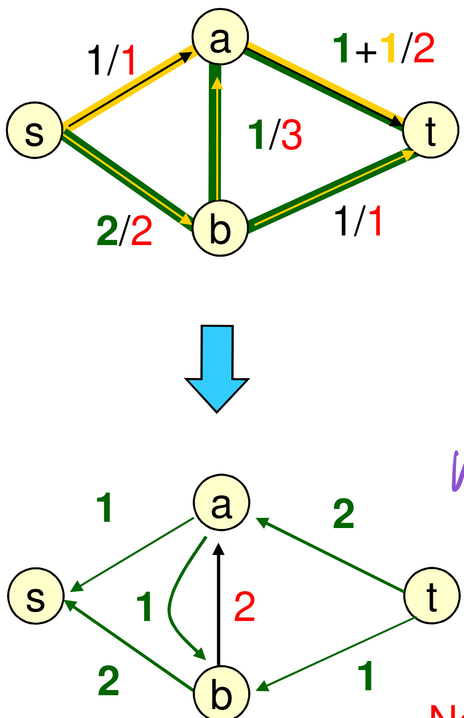
Greed Revisited: Residual Graph & Augmenting Path



Greed Revisited: Residual Graph & Augmenting Path



Greed Revisited: An Augmenting Path



no path from s to t

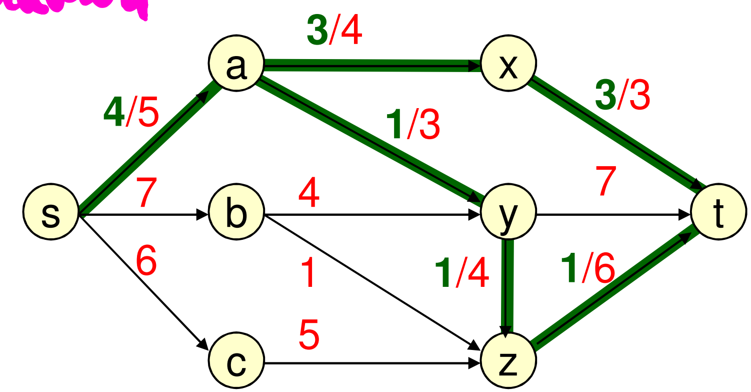
New Residual Graph

Residual Capacity

forward

- The *residual capacity* (w.r.t. **f**) of **(u,v)** is $\mathbf{c_f(u,v) = c(u,v) - f(u,v)}$ if $\mathbf{f(u,v) \leq c(u,v)}$ and $\mathbf{c_f(u,v)=f(v,u)}$ if $\mathbf{f(v,u)>0}$

backward



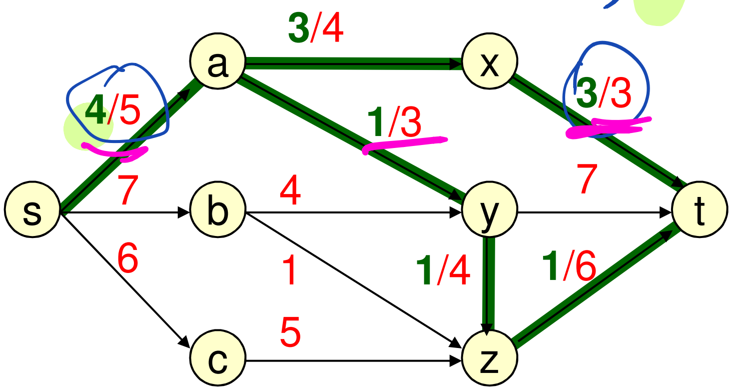
- e.g. $\mathbf{c_f(s,b)=7}$; $\mathbf{c_f(a,x) = 1}$; $\mathbf{c_f(x,a) = 3}$

Residual Graph & Augmenting Paths

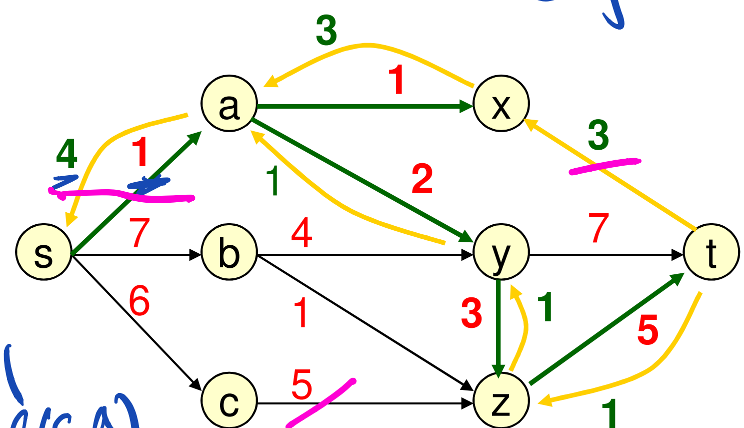
- The *residual graph* (w.r.t. $\mathbf{f}$) is the graph $\mathbf{G_f} = (\mathbf{V},\mathbf{E_f})$, where
  $$\mathbf{E_f} = \{ (\mathbf{u},\mathbf{v}) \mid \mathbf{c_f}(\mathbf{u},\mathbf{v}) > 0 \}$$
  - Two kinds of edges
    - Forward edges
      - $\mathbf{f}(\mathbf{u},\mathbf{v})<\mathbf{c}(\mathbf{u},\mathbf{v})$ so $\mathbf{c_f}(\mathbf{u},\mathbf{v}) = \mathbf{c}(\mathbf{u},\mathbf{v}) - \mathbf{f}(\mathbf{u},\mathbf{v}) > \mathbf{0}$
    - Backward edges
      - $\mathbf{f}(\mathbf{u},\mathbf{v})>\mathbf{0}$ so $\mathbf{c_f}(\mathbf{v},\mathbf{u}) = \mathbf{f}(\mathbf{u},\mathbf{v}) > \mathbf{0}$
- An *augmenting path* (w.r.t. $\mathbf{f}$) is a simple $\mathbf{s} \rightarrow \mathbf{t}$ path in $\mathbf{G_f}$.

A Residual Network

G, f

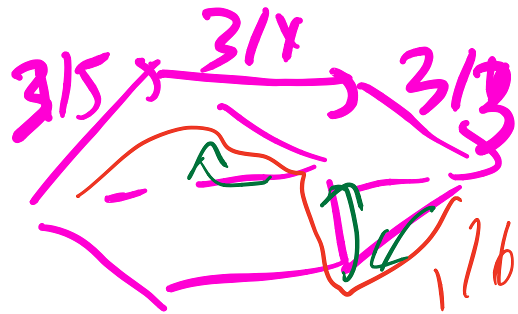


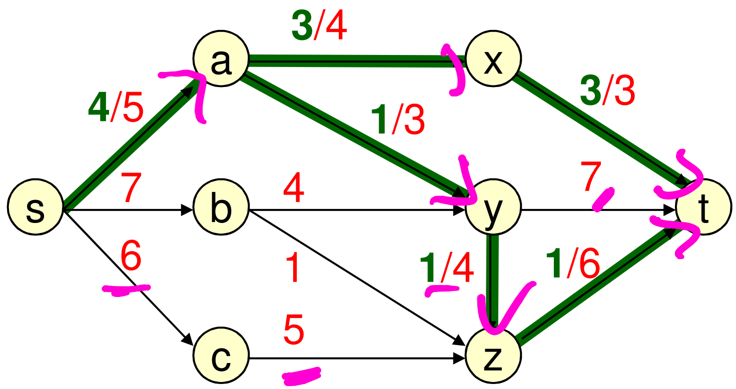
G_f

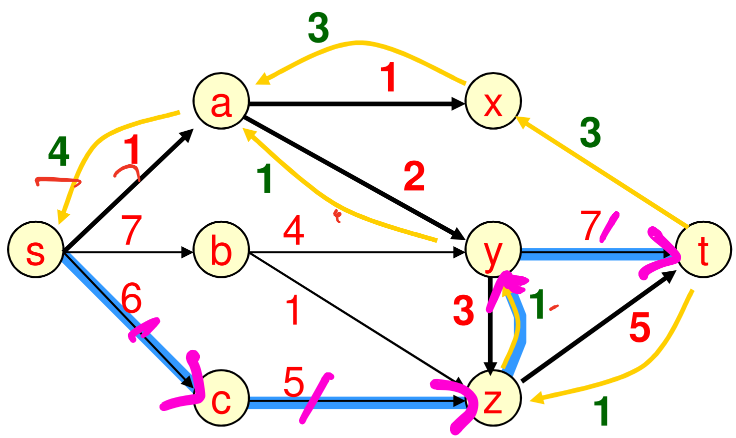


$c(s,a) = 5$
$f(s,a) = 4$
$c_f(s,a) = c(s,a) - f(s,a)$
$= 5 - 4 = 1$
$c_f(a,s) = f(s,a)$
$= 4$

An Augmenting Path







Augmenting A Flow

in residual graph

augment(**f**,**P**)

$\mathbf{c_P} \leftarrow \min_{(\mathbf{u},\mathbf{v})\in \mathbf{P}} \mathbf{c_f}(\mathbf{u},\mathbf{v})$ "bottleneck(P)"

for each $\mathbf{e} \in \mathbf{P}$

if **e** is a forward edge then

increase **f**(**e**) by $\mathbf{c_P}$

else (**e** is a backward edge)

decrease **f**(**e'**) by $\mathbf{c_P}$ where **e'** = reverse of **e**

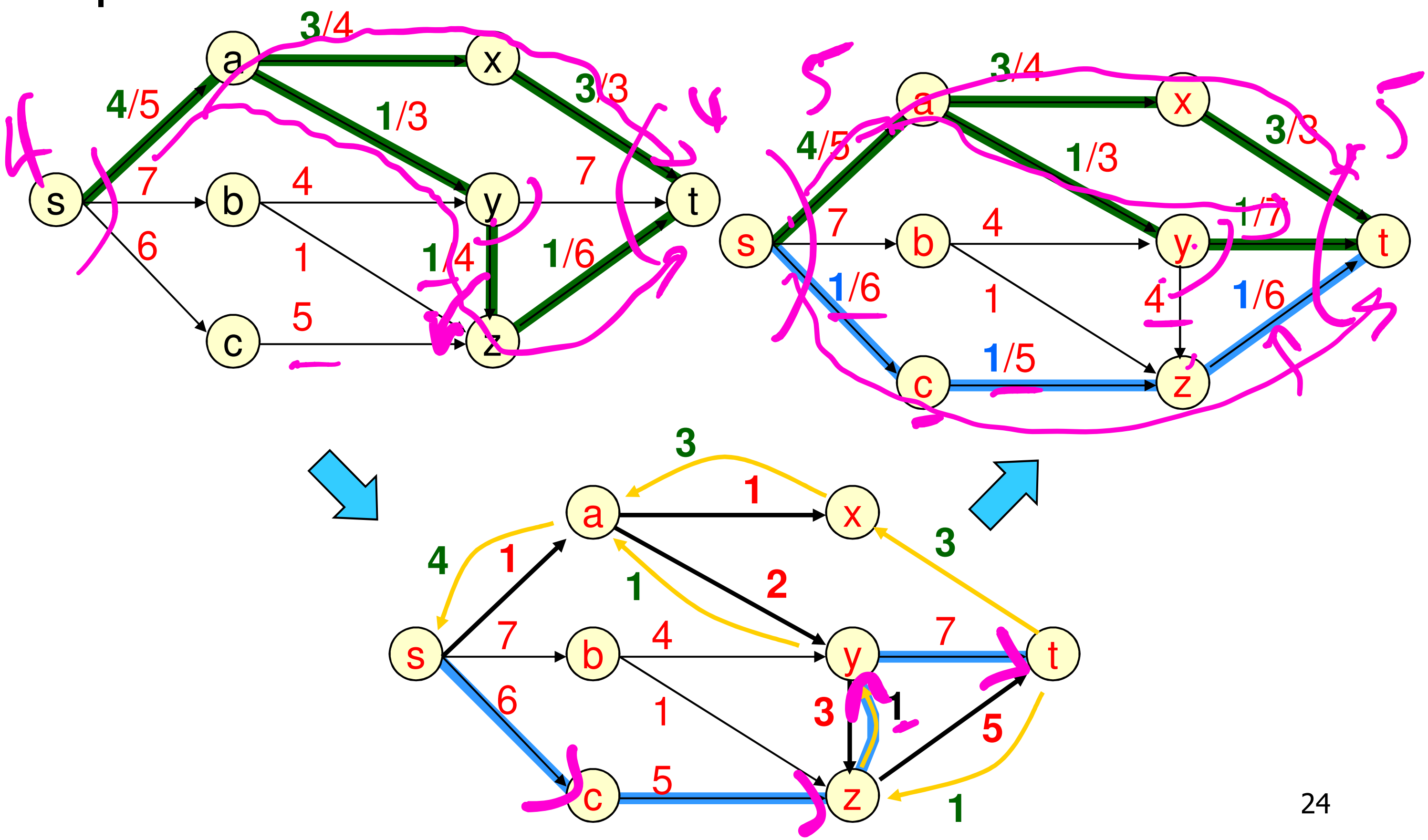
i.e. the original forward edge

endif

endfor

return(**f**)

Augmenting A Flow



Claim: Augmented flow is legal

If G_f has an augmenting path **P**, then the function **f'=**augment(**f**,**P**) is a legal flow.

Proof:

- **f'** and **f** differ only on the edges of **P** so only need to consider such edges (**u**,**v**)

Proof: Augmented flow is legal

- If (u,v) is a forward edge then
$$f'(u,v)=f(u,v)+c_P \le f(u,v)+c_f(u,v)$$
$$= f(u,v)+c(u,v)-f(u,v)$$
$$=c(u,v)$$
- If (u,v) is a backward edge then f and f' differ on flow along (v,u) instead of (u,v)
$$f'(v,u)=f(v,u)-c_P \ge f(v,u)-c_f(u,v)$$
$$= f(v,u)-f(v,u)=0$$
- Other conditions like flow conservation still met

Ford-Fulkerson Method

residual graph

Start with **f=0** for every edge

While **G_f** has an augmenting path,

augment

- Questions:
  - Does it halt?
  - Does it find a maximum flow?
  - How fast?

Observations about Ford-Fulkerson Algorithm

- At every stage the capacities and flow values are always integers (if they start that way)
- The flow value $\nu(\mathbf{f'})=\nu(\mathbf{f})+\mathbf{c_P}>\nu(\mathbf{f})$ for $\mathbf{f'}=\mathrm{augment}(\mathbf{f},\mathbf{P})$

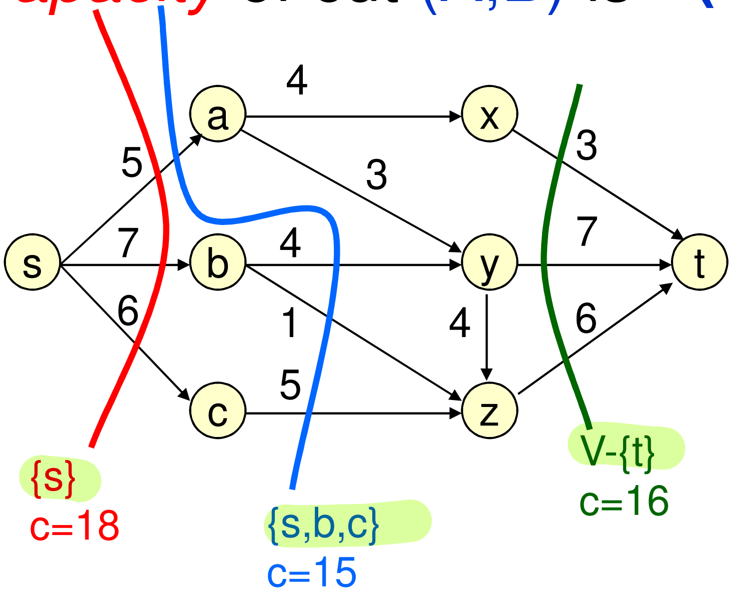
  >0
  - Since edges of residual capacity **0** do not appear in the residual graph
- Let $\mathbf{C}=\sum_{(\mathbf{s},\mathbf{u})\in \mathbf{E}} \mathbf{c}(\mathbf{s},\mathbf{u})$
  - $\nu(\mathbf{f})\le\mathbf{C}$
  - **F-F** does at most **C** rounds of augmentation since flows are integers and increase by at least **1** per step

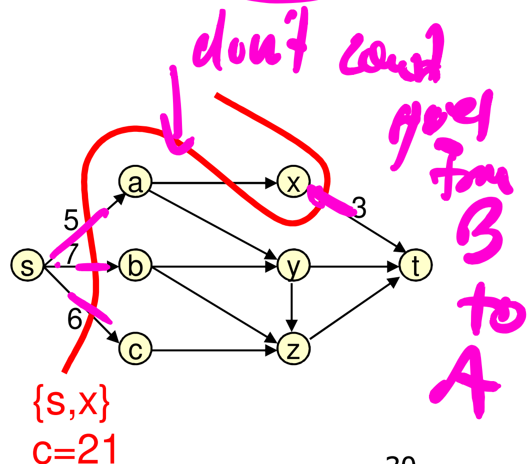
Running Time of Ford-Fulkerson

- For $\mathbf{f=0}$, $\mathbf{G_f=G}$
- Finding an augmenting path in G_f is graph search $O(\mathbf{n}+\mathbf{m})=O(\mathbf{m})$ time
- Augmenting and updating $\mathbf{G_f}$ is $O(\mathbf{n})$ time
- Total $O(\mathbf{mC})$ time
- Does it find a maximum flow?
  - Need to show that for every flow $\mathbf{f}$ that isn't maximum $\mathbf{G_f}$ contains an **s-t**-path

Cuts

- A partition (**A,B**) of **V** is an *s-t-cut* if
  - **s∈ A**, **t∈ B**
- *Capacity* of cut (A,B) is $\mathbf{c(A,B)} = \sum_{\substack{u\in A\\ v\in B}} \mathbf{c(u,v)}$





Convenient Definition

- $f^{out}(A)=\sum_{v\in A,\, w\notin A} f(v,w)$ = flow of cut (A,B) ≤ capacity of cut (A,B)

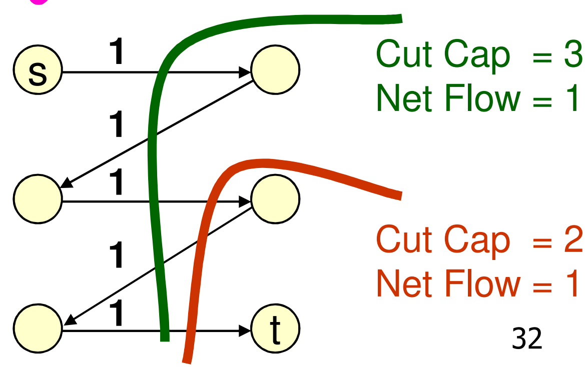
- $f^{in}(A)=\sum_{v\in A,\, u\notin A} f(u,v)$ total flow of edges from B to A

Two claims

- For any flow **f** and any cut (**A**,**B**), $f^{out}(s) - f^{in}(s)$ $=$ 0
  1) the net flow across the cut equals the total flow, i.e., $v(f) = f^{out}(A) - f^{in}(A)$, and
  2) the net flow across the cut cannot exceed the capacity of the cut, i.e. $f^{out}(A) - f^{in}(A) \leq c(A,B)$

  $v(f) \leq$ cut value

- Corollary :
  Max flow ≤ Min cut



Proof of Claim 1

- Consider a set **A** with $\mathbf{s} \in \mathbf{A}$, $\mathbf{t} \notin \mathbf{A}$
- $f^{out}(A)-f^{in}(A) = \sum_{v\in A,\, w\notin A} f(v,w) - \sum_{v\in A,\, u\notin A} f(u,v)$
- We can add flow values for edges with both endpoints in **A** to **both** sums and they would cancel out so
- $f^{out}(A)-f^{in}(A) = \sum_{v\in A,\, w\in V} f(v,w) - \sum_{v\in A,\, u\in V} f(u,v)$

$$= \sum_{v\in A} \left(\sum_{w\in V} f(v,w) - \sum_{u\in V} f(u,v)\right)$$

$$= \sum_{v\in A} \left(f^{out}(v) - f^{in}(v)\right)$$

$$= f^{out}(s) - f^{in}(s)$$

since all other vertices have $f^{out}(v)=f^{in}(v)$

- $\nu(f) = f^{out}(s)$ and $f^{in}(s)=0$

Proof of Claim 2

- $\nu(f)=f^{out}(A)-f^{in}(A)$ ≥ 0
  $\leq f^{out}(A)$
  $= \sum_{v\in A,\, w\notin A} f(v,w)$
  $\leq \sum_{v\in A,\, w\notin A} c(v,w)$
  $\leq \sum_{v\in A,\, w\in B} c(v,w)$
  $=c(A,B)$

$f^{out}(A) \leq c(A,B)$

Max Flow / Min Cut Theorem

Claim 3 For any flow $\mathbf{f}$, if G_f has no augmenting path then there is some $\mathbf{s}$-$\mathbf{t}$-cut $(\mathbf{A},\mathbf{B})$ such that $\nu(\mathbf{f})=\mathbf{c}(\mathbf{A},\mathbf{B})$ (proof on next slide)

- We know by Claims 1 & 2 that any flow $\mathbf{f'}$ satisfies $\nu(\mathbf{f'}) \leq \mathbf{c}(\mathbf{A},\mathbf{B})$ and we know that F-F runs for finite time until it finds a flow $\mathbf{f}$ satisfying conditions of Claim 3
  - Therefore by Claim 3 for any flow $\mathbf{f'}$, $\nu(\mathbf{f'}) \leq \nu(\mathbf{f})$
- Theorem (a) F-F computes a maximum flow in $\mathbf{G}$

(b) For any graph $\mathbf{G}$, the value $\nu(\mathbf{f})$ of a maximum flow = minimum capacity $\mathbf{c}(\mathbf{A},\mathbf{B})$ of any $\mathbf{s}$-$\mathbf{t}$-cut in $\mathbf{G}$

$\overset{1}{7.8}$, $\overset{2}{7.8}$, $\overset{3}{7.9}$ → K&T

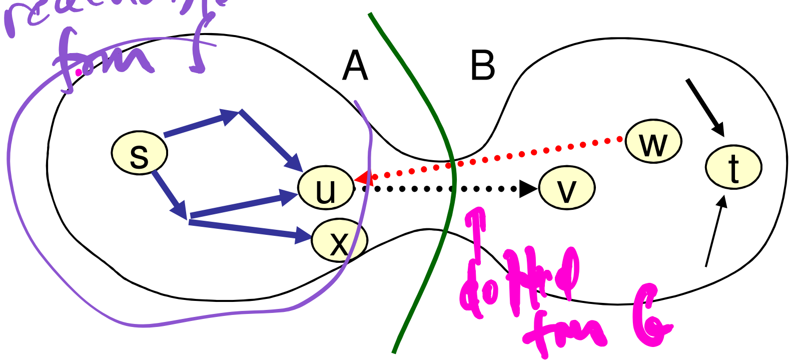
Claim 3

residual

Let $\mathbf{A} = \{ \mathbf{u} \mid \exists$ an path in $\mathbf{G_f}$ from $\mathbf{s}$ to $\mathbf{u} \}$

$\mathbf{B} = \mathbf{V} - \mathbf{A}$; $\mathbf{s} \in \mathbf{A}$, $\mathbf{t} \in \mathbf{B}$

reachable from s



forward

saturated

$\mathbf{f(u,v)=c(u,v)}$

$(u,v) \in G$

$(u,v) \notin G_f$

no flow

$\mathbf{f(w,u)=0}$

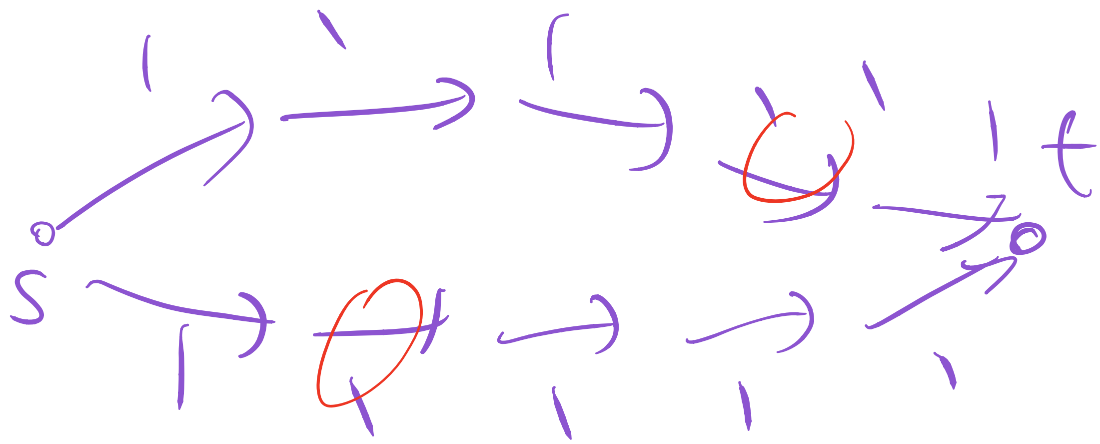
$(w,u) \in G$

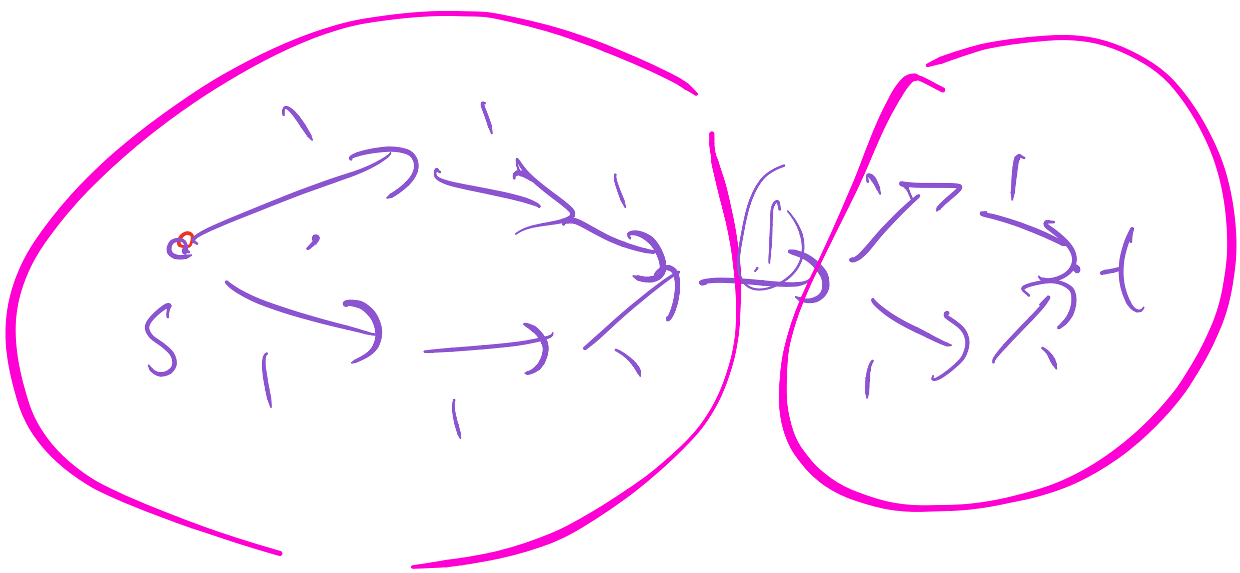
$(u,w) \notin G_f$

This is true for **every** edge crossing the cut, i.e.

$$\mathbf{f^{out}(A) = \sum_{\substack{u\in A\\ v\in B}} f(u,v) = \sum_{\substack{u\in A\\ v\in B}} c(u,v) = c(A,B)}$$ and $\mathbf{f^{in}(A)=0}$ so

$$\mathbf{v(f)=\underbrace{f^{out}(A)}_{c(A,B)}-\underbrace{f^{in}(A)}_{0}=c(A,B)}$$

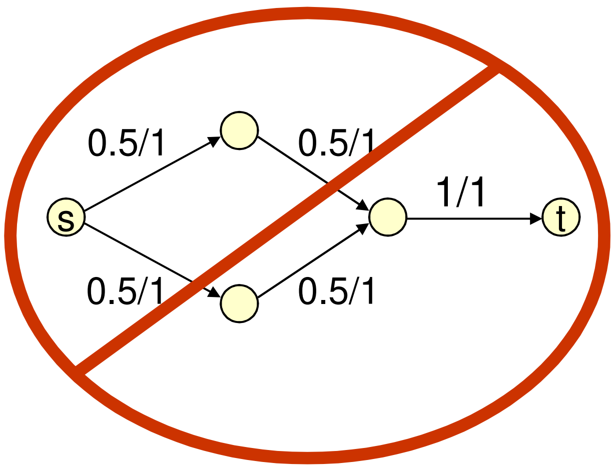

s
t
1


s
1

Flow Integrality Theorem

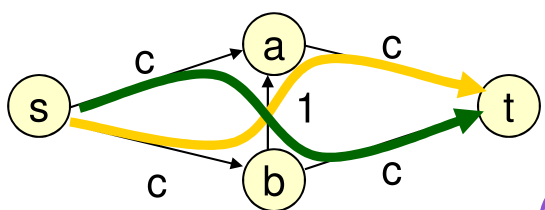
If all capacities are integers

- The max flow has an integer value
- Ford-Fulkerson method finds a max flow in which f(u,v) is an integer for all edges (u,v)

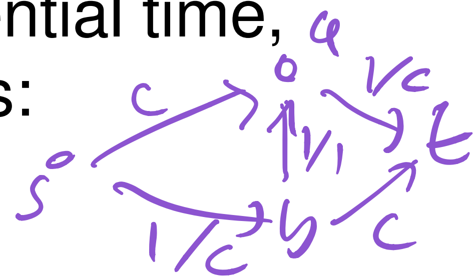


Corollaries & Facts

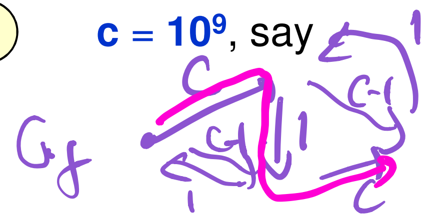
- If Ford-Fulkerson terminates, then it's found a max flow.
- It will terminate if **c(e)** integer or rational (but may not if they're irrational).
- However, may take exponential time, even with integer capacities:



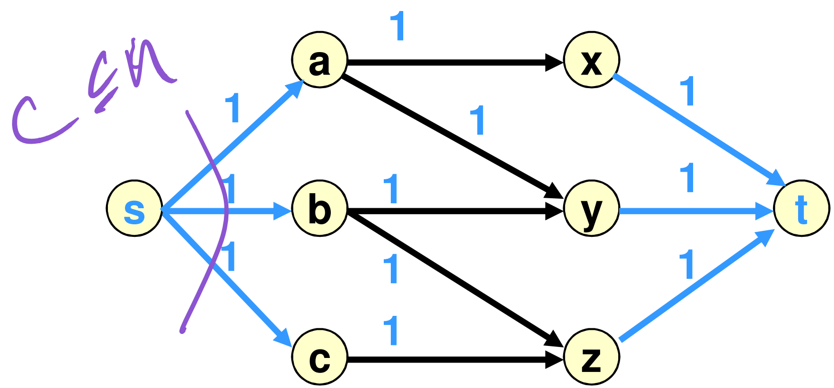
2c steps



c = **10**^{9}, say



Bipartite matching as a special case of flow



Integer flows implies each flow is just a subset of the edges

Therefore flow corresponds to a matching

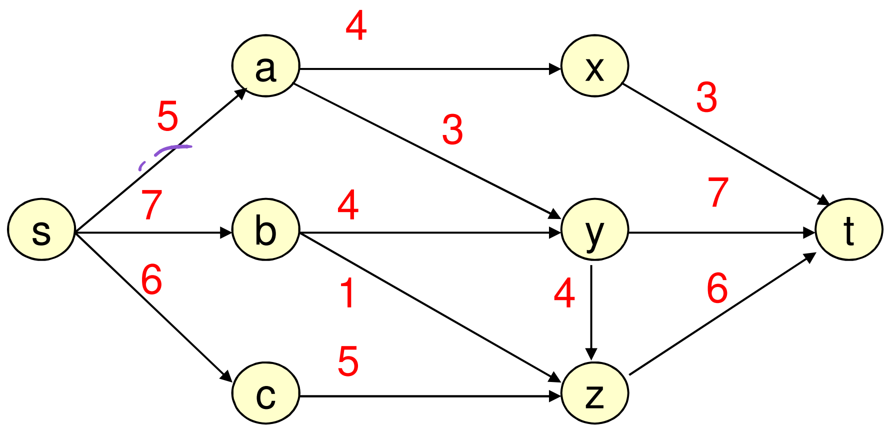
O(**mC**)=$\boxed{\text{O(}\textbf{nm}\text{)}}$ running time

n

Capacity-Scaling algorithm

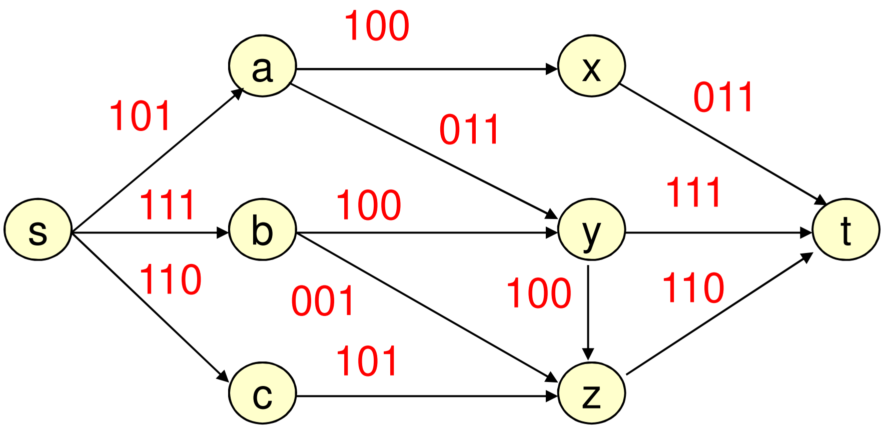
- General idea:
  - Choose augmenting paths **P** with 'large' capacity $\mathbf{c_P}$
  - Can augment flows along a path **P** by any amount $\Delta \leq \mathbf{c_P}$
    - Ford-Fulkerson still works
  - Get a flow that is maximum for the high-order bits first and then add more bits later

Capacity Scaling

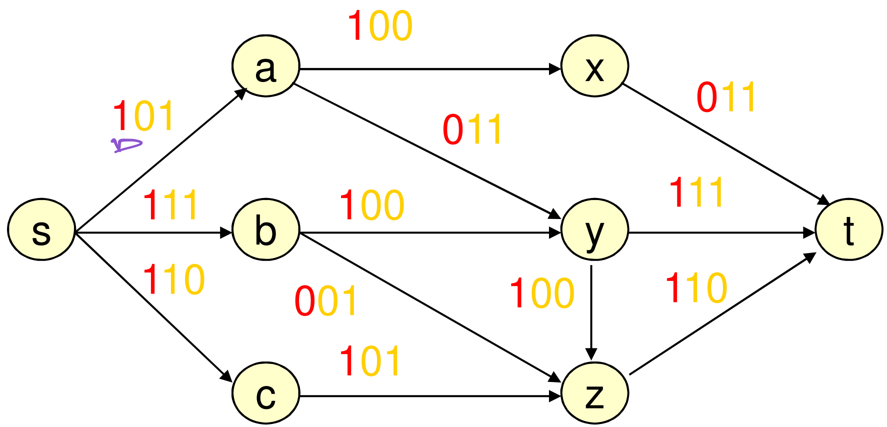


represent capacity in binary

Capacity Scaling



Capacity Scaling Bit 1

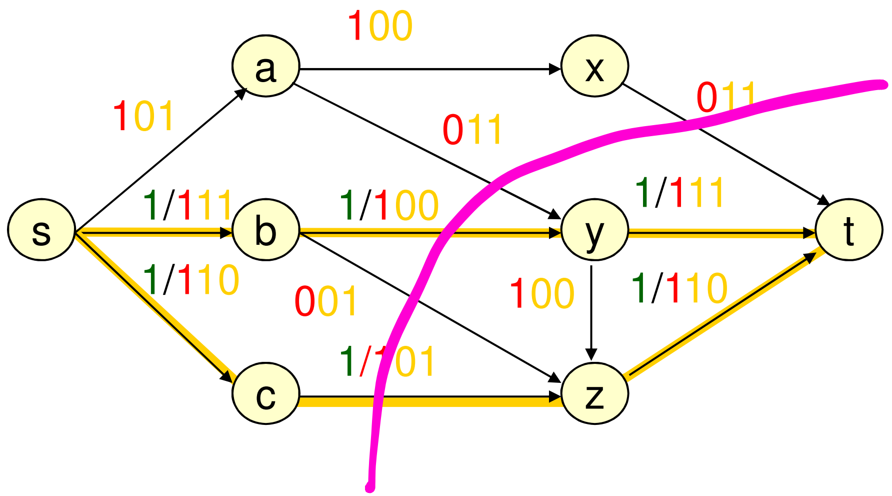


Capacity on each edge is at most **1**
(either **0** or **1** times **Δ=4**)

each iteration of FF augment
∴ O(m)
# iterations of FF augment
O(mn)

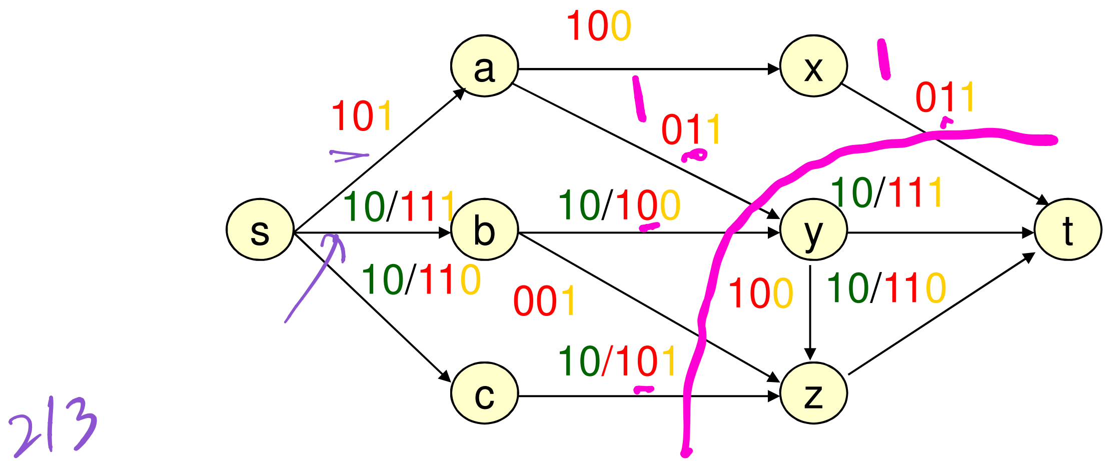
F.F with 0/1 capacity
0/4

Capacity Scaling Bit 1



O(nm) time

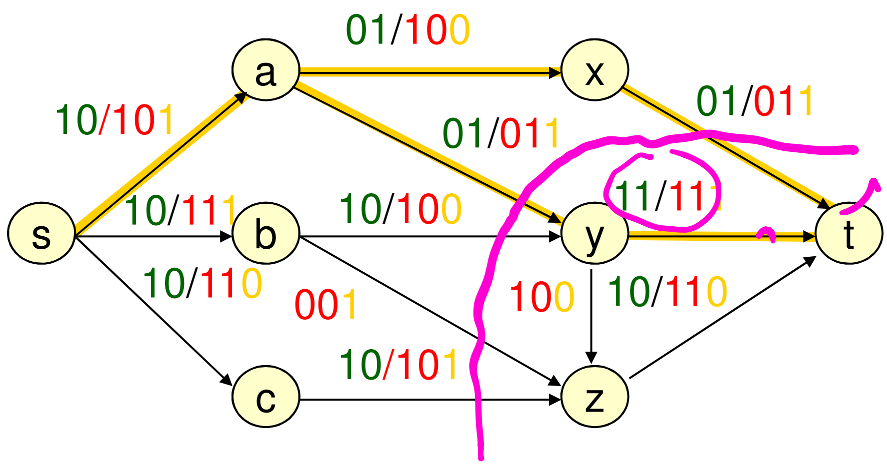
Capacity Scaling Bit 2



Residual capacity across min cut is at most **m**
(either **0** or **1** times **Δ=2**)

# of steps O(m²)

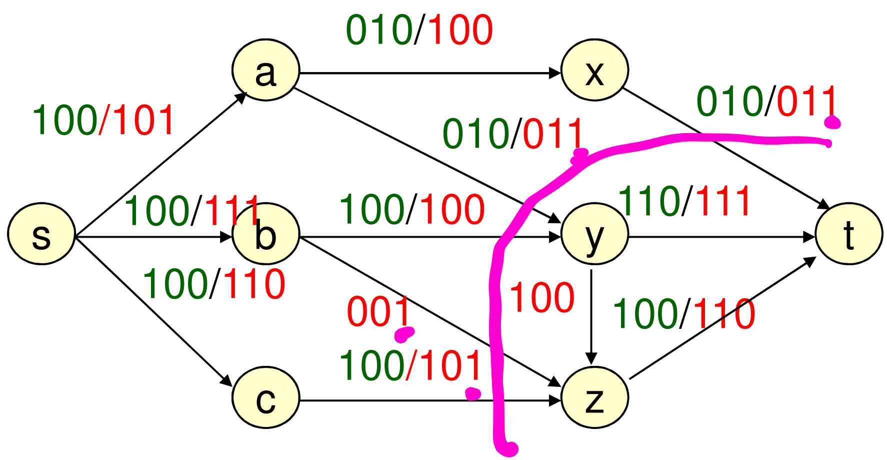
Capacity Scaling Bit 2



Residual capacity across min cut is at most **m**

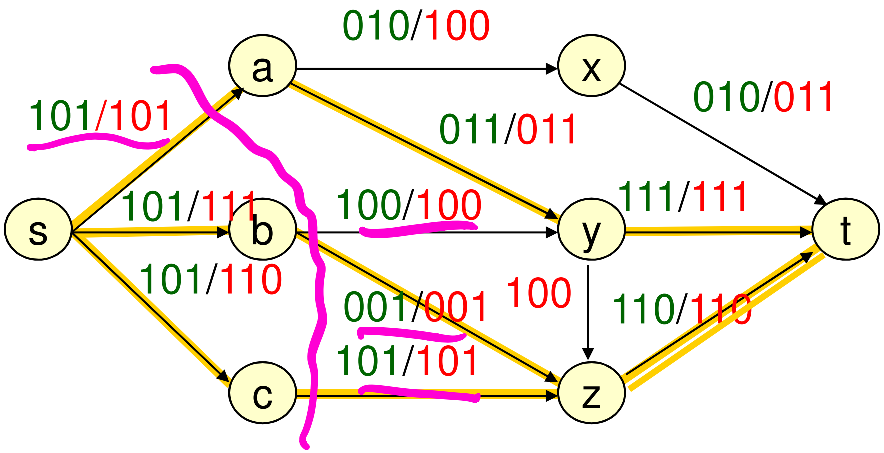
⇒ ≤ **m** augmentations

Capacity Scaling Bit 3



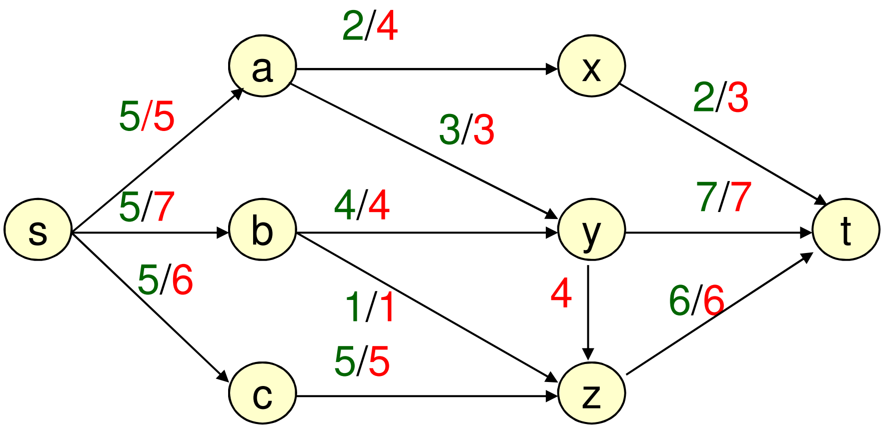
Residual capacity across min cut is at most **m**
(either **0** or **1** times **Δ=1**)

Capacity Scaling Bit 3

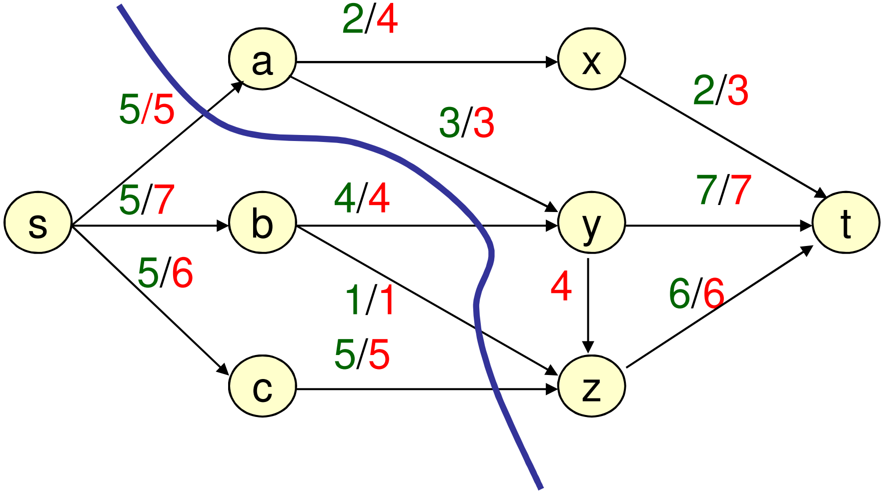


After $\leq$ **m** augmentations

Capacity Scaling Final



Capacity Scaling Min Cut



Total time for capacity scaling

total capacity cap({s}, rest)

- $\log_2 \mathbf{U}$ rounds where $\mathbf{U}$ is largest capacity
- At most $\mathbf{m}$ augmentations per round
  - Let $\mathbf{c_i}$ be the capacities used in the $\mathbf{i}^{th}$ round and $\mathbf{f}_i$ be the maxflow found in the $\mathbf{i}^{th}$ round
    - For any edge $(\mathbf{u},\mathbf{v})$, $\mathbf{c_{i+1}}(\mathbf{u},\mathbf{v}) \leq \mathbf{2c_i}(\mathbf{u},\mathbf{v})+\mathbf{1}$
  - $\mathbf{i+1}^{st}$ round starts with flow $\mathbf{f = 2\ f_i}$
  - Let $(\mathbf{A},\mathbf{B})$ be a min cut from the $\mathbf{i}^{th}$ round
    - $\nu(\mathbf{f_i})=\mathbf{c_i}(\mathbf{A},\mathrm{B})$ so $\nu(\mathbf{f})=\mathbf{2c_i}(\mathbf{A},\mathbf{B})$
  - $\nu(\mathbf{f_{i+1}}) \leq \mathbf{c_{i+1}}(\mathbf{A},\mathbf{B}) \leq \mathbf{2c_i}(\mathbf{A},\mathbf{B})+\mathbf{m} = \nu(\mathbf{f})+\mathbf{m}$
- $O(\mathbf{m})$ time per augmentation
- Total time $O(\mathbf{m^2} \log \mathbf{U})$

m log U vs C

mincut value

Edmonds-Karp Algorithm

- Use a shortest augmenting path (via Breadth First Search in residual graph)

- Time: $O(\mathbf{n\ m^2})$

worst case analysis

BFS/Shortest Path Lemmas

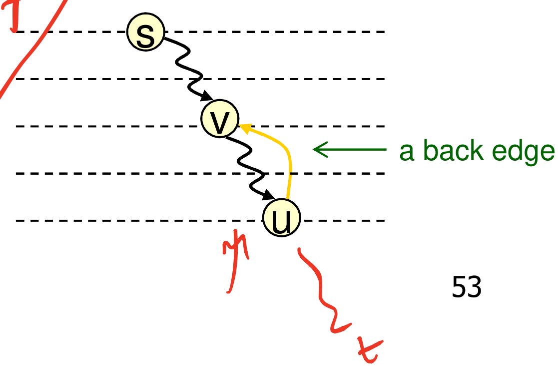
residual graph

Distance from **s** in G_f is never reduced by:

- Deleting an edge
  Proof: no new (hence no shorter) path created
- Adding an edge (u,v), provided v is nearer than u
  Proof: BFS is unchanged, since **v** visited before **(u,v)** examined

$G_f \rightsquigarrow G_f'$

delete an edge

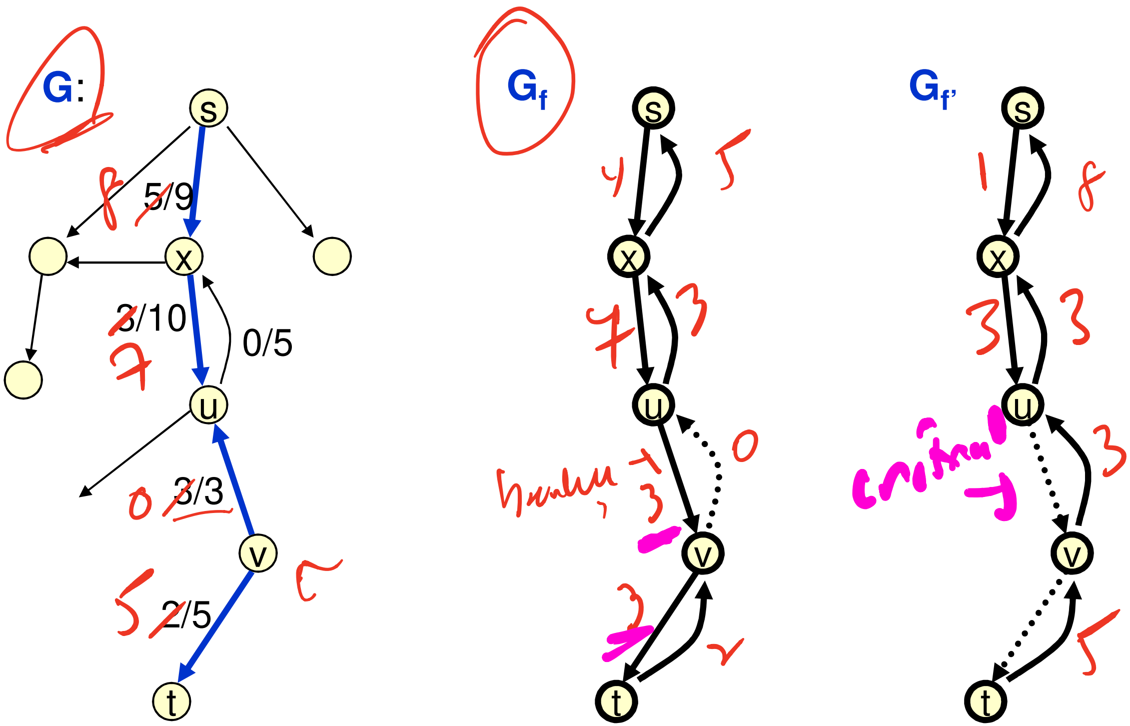


Key Lemma

Let **f** be a flow, $\mathbf{G_f}$ the residual graph, and **P** a shortest augmenting path. Then no vertex is closer to **s** after augmentation along **P**.

Proof: Augmentation along **P** only deletes forward edges, or adds back edges that go to previous vertices along **P**

Augmentation vs BFS



Theorem

The Edmonds-Karp Algorithm performs $O(mn)$ flow augmentations

Proof:

Call (u,v) critical for augmenting path P if it's closest to s having min residual capacity

It will disappear from G_f after augmenting along P

In order for (u,v) to be critical again the (u,v) edge must re-appear in G_f but that will only happen when the distance to u has increased by 2 (next slide)

It won't be critical again until farther from s so each edge critical at most $n/2$ times

Critical Edges in G_f

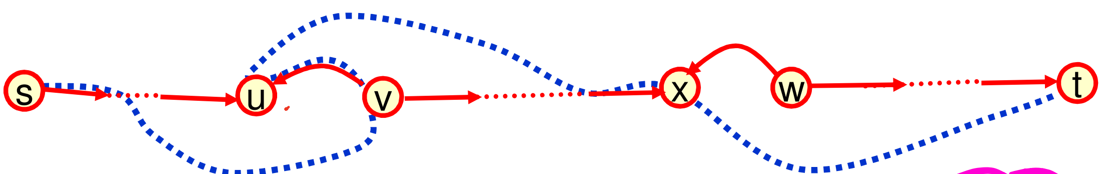
Shortest s-t path **P** in G_f

critical edge

$d_f(s,v)=d_f(s,u)+1$ since this is a shortest path

After augmenting along **P**

For **(u,v)** to be critical later for some flow **f'** it must be in $G_{f'}$
so must have augmented along a shortest path containing **(v,u)**



Then we must have $d_{f'}(s,u)=d_{f'}(s,v)+1 \ge d_f(s,v)+1=d_f(s,u)+2$

new

old

Corollary

- Edmonds-Karp runs in $O(\mathbf{nm}^2)$ time

at most $\frac{nm}{2}$ augmentations

* cost of BFS $O(m)$

$O(n^3)$

$O(nm)$

Problem 3

One max flow run plus a linear amount of extra work.

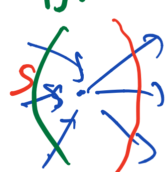
Applications of Network Flow

1. Edge-Disjoint Paths in Graphs:

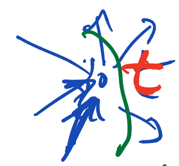
Given vertices s and t in G, find as many simple paths as possible from s to t that don't share any edges

(a) Directed Graphs:

ignore



ignore



min cut $C \le n-1$

capacity 1 on all edges

Compute a max flow using F-F

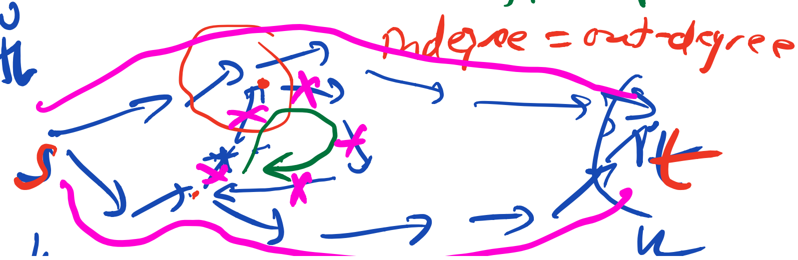
time $O(mn)$

suppose max flow value is k

F-F all flows are integers

flow ≡ set of edges where flow = 1

only draw edges with flow 1



Min cut

= min # of edges whose deletion disconnects s from t

flow may have a cycle

Repeat: find a cycle in flow & set value to 0.

Greedy: start at s take an outedge
repeat until (either t is reached
or repeated vertex v
if repeated vertex found
remove cycle and continue
until t

Repeat
Find one path
remove that path

O(m)

Menger's Thm for directed graphs
maxflow = min cut for directed graph

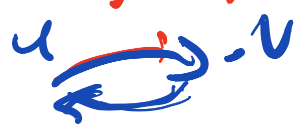
$\boxed{\text{max \# of edge-disjoint s-t paths} = \text{min \# of edges whose deletion removes all s-t paths}}$

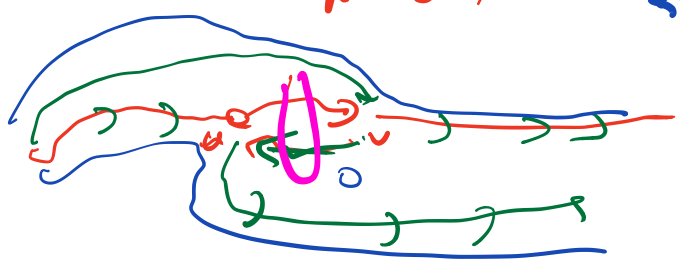
(b) Undirected graphs?

Solution:

run FF and thus directed simple path finding never get both

on a single path.





Menger's for Undirected Graphs:

min # of edge-disjoint paths between s and t

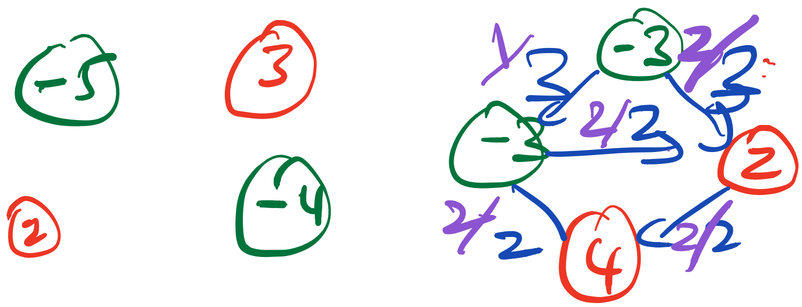
= min # of edges whose deletion disconnects s and t.

Directed Network with suppliers
and consumers
capacities on edges

vertex v : either a supplier or consumer

consumer v : demand $d_v \geq 0$

supplier : demand $d_v < 0$
supply $-d_v$ units

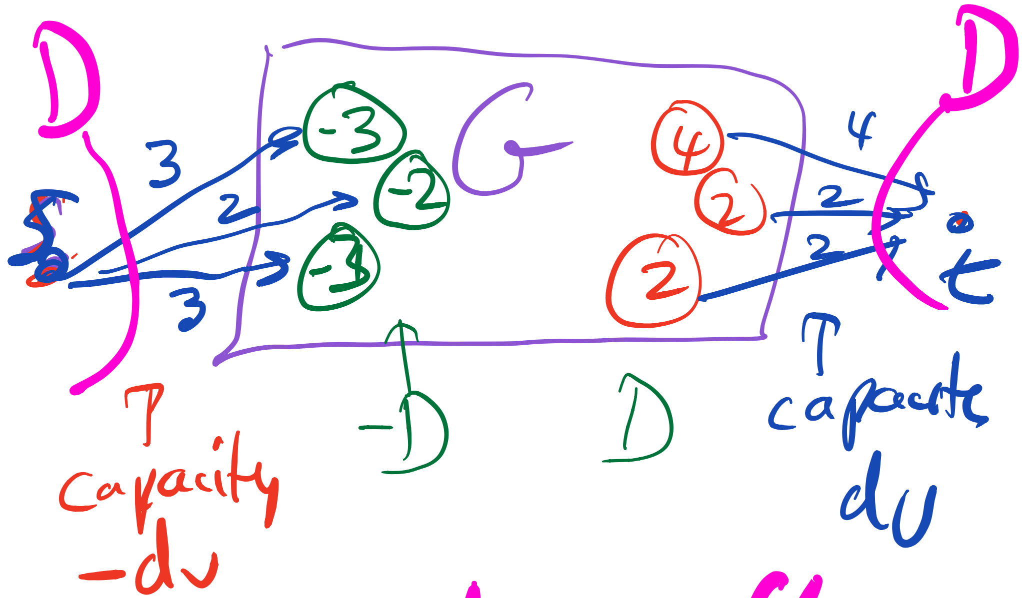


Circulation with demands $d_v = f^{in}(v) - f^{out}(v)$

Q: can we meet all the demands
i.e. suppliers send out all
their suppliers and
consumers gets all
their needs filled?

G

Need: $\sum_v d_v = 0$



compute maxflow

FF alg with actually compute values

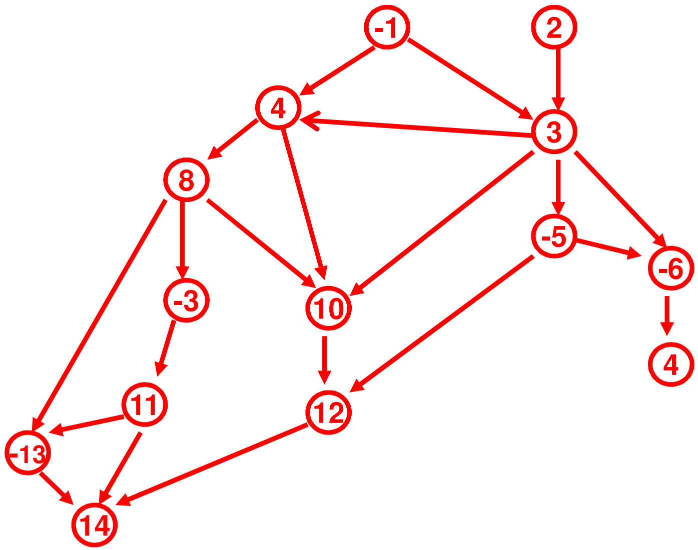
if $= D$ yes get circulation ✓

if $< D$ can't satisfy

Project Selection a.k.a. The Strip Mining Problem

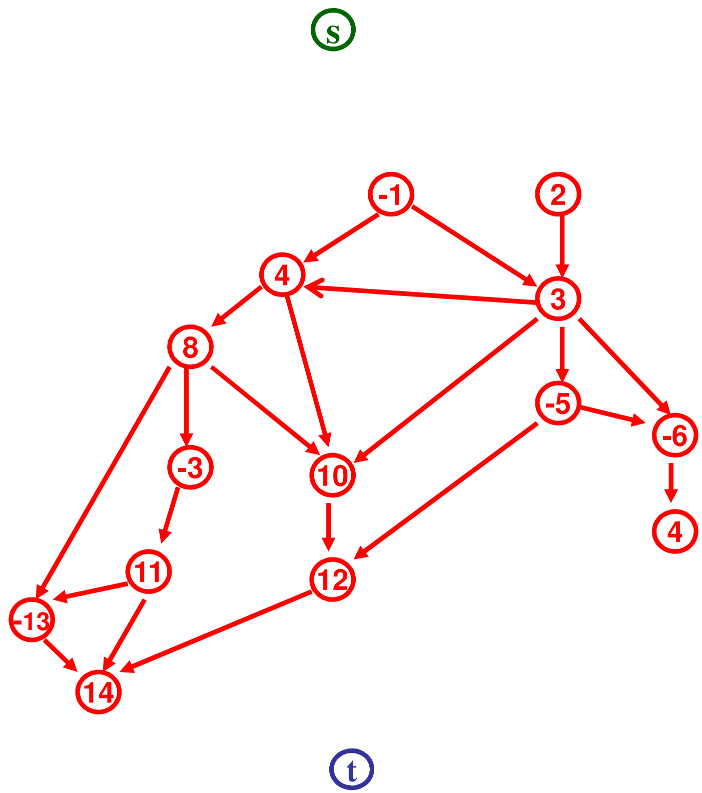
- Given
  - a directed acyclic graph $\mathbf{G}=(\mathbf{V},\mathbf{E})$ representing precedence constraints on tasks (a task points to its **predecessors**)
  - a profit value $\mathbf{p}(\mathbf{v})$ associated with each task $\mathbf{v} \in \mathbf{V}$ (may be positive or negative)
- Find
  - a set $\mathbf{A} \subseteq \mathbf{V}$ of tasks that is closed under predecessors, i.e. if $(\mathbf{u},\mathbf{v}) \in \mathbf{E}$ and $\mathbf{u} \in \mathbf{A}$ then $\mathbf{v} \in \mathbf{A}$, that maximizes $\text{Profit}(\mathbf{A})=\Sigma_{\mathbf{v} \in \mathbf{A}} \mathbf{p}(\mathbf{v})$

Project Selection Graph



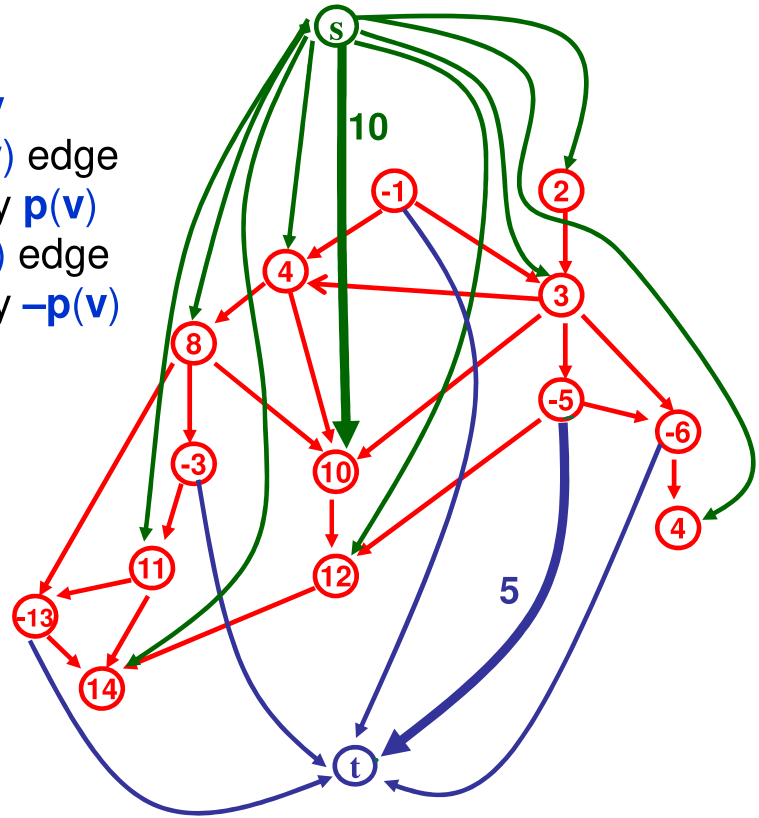
Each task points to its predecessor tasks

Extended Graph



Extended Graph G’

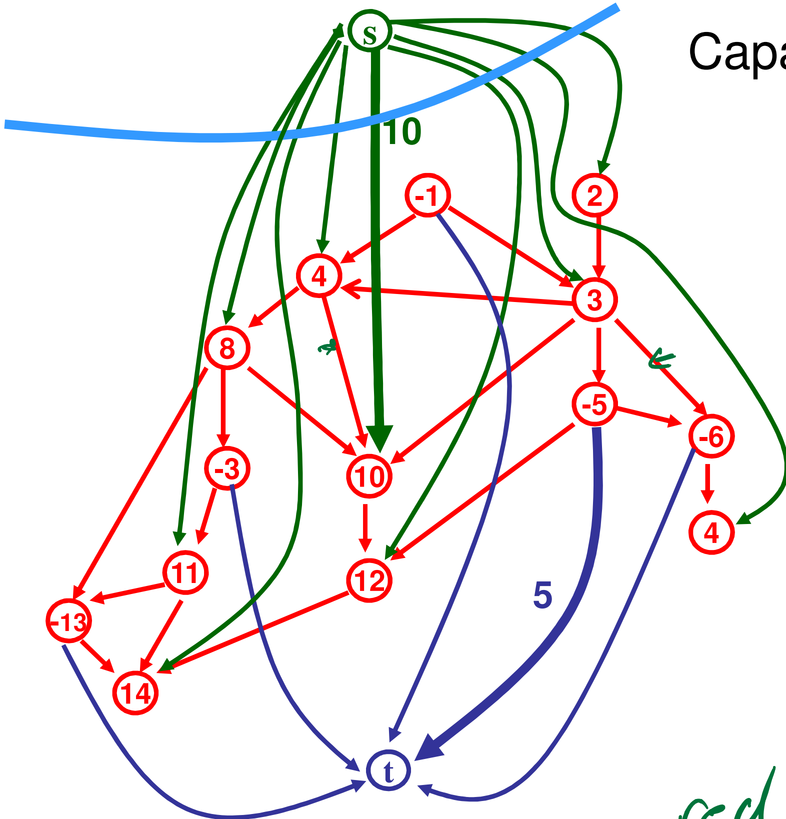
For each vertex **v**
If **p**(**v**)≥**0** add (**s**,**v**) edge
with capacity **p**(**v**)
If **p**(**v**)<**0** add (**v**,**t**) edge
with capacity **–p**(**v**)



Extended Graph G'

- Want to arrange capacities on edges of **G** so that for minimum **s**-**t**-cut (**S**,**T**) in **G'**, the set **A**=**S**-{**s**}
  - satisfies precedence constraints
  - has maximum possible profit in **G**
- Cut capacity with **S**={**s**} is just $\mathbf{C}=\sum_{\mathbf{v}:\ \mathbf{p(v)}\geq 0} \mathbf{p(v)}$
  - Profit(**A**) ≤ **C** for any set **A**
- To satisfy precedence constraints don't want any original edges of **G** going forward across the minimum cut
  - That would correspond to a task in **A**=**S**-{**s**} that had a predecessor not in **A**=**S**-{**s**}
- Set capacity of each of the edges of **G** to **C+1**
  - The minimum cut has size at most **C**

Extended Graph G'

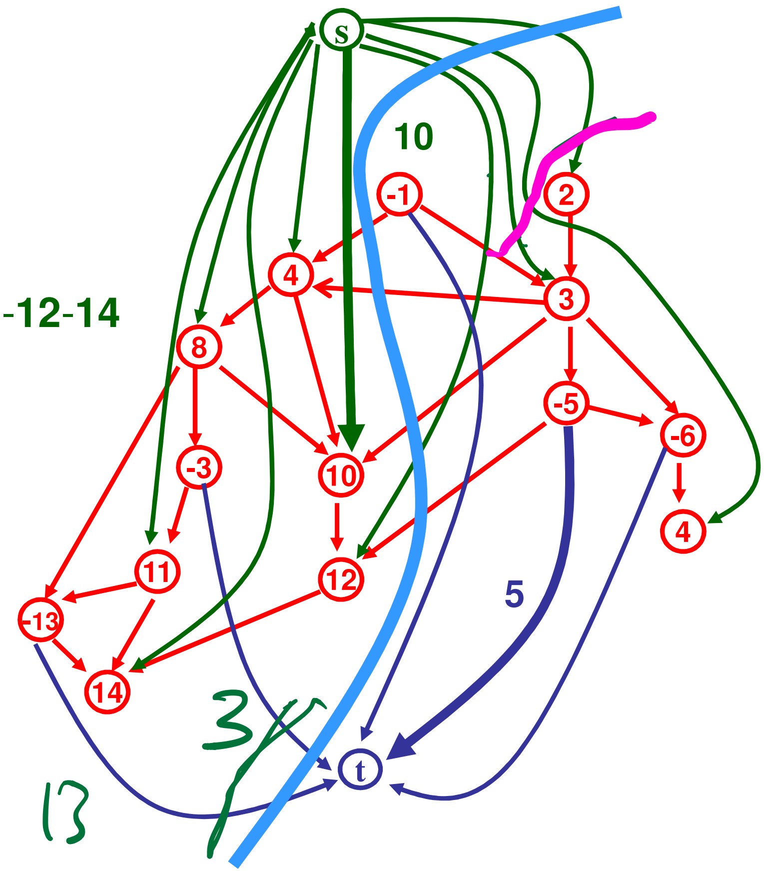


Capacity **C**

red edges capacity C+1
never part of a cut [illegible]

Extended Graph G'

Cut value
=13+3+2+3+4
=13+3
+C-4-8-10-11-12-14



Project Selection

- **Claim** Any s-t-cut (S,T) in G' such that $A=S-\{s\}$ satisfies precedence constraints has capacity

$$c(S,T)=C - \sum_{v\in A} p(v) = C - \text{Profit}(A)$$

- **Corollary** A minimum cut (S,T) in G' yields an optimal solution $A=S-\{s\}$ to the profit selection problem

- **Algorithm** Compute maximum flow f in G', find the set S of nodes reachable from s in G'_f and return $S-\{s\}$

Proof of Claim

- $A=S-\{s\}$ satisfies precedence constraints
  - No edge of G crosses forward out of A since those edges have capacity $C+1$
  - Only forward edges cut are of the form (v,t) for $v\in A$ or (s,v) for $v\notin A$
  - The (v,t) edges for $v\in A$ contribute
    $$\sum_{v\in A:p(v)<0} -p(v) = -\sum_{v\in A:p(v)<0} p(v)$$
  - The (s,v) edges for $v\notin A$ contribute
    $$\sum_{v\notin A:\ p(v)\geq 0} p(v)=C-\sum_{v\in A:\ p(v)\geq 0} p(v)$$
  - Therefore the total capacity of the cut is
    $$c(S,T) = C - \sum_{v\in A} p(v) = C-\text{Profit}(A)$$