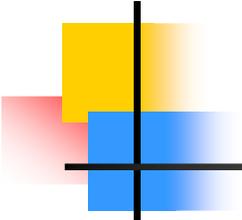


# **CSE 521: Design & Analysis of Algorithms I**

---

## **Network Flow**

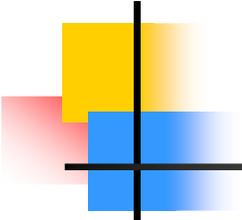
Paul Beame



# Bipartite Matching

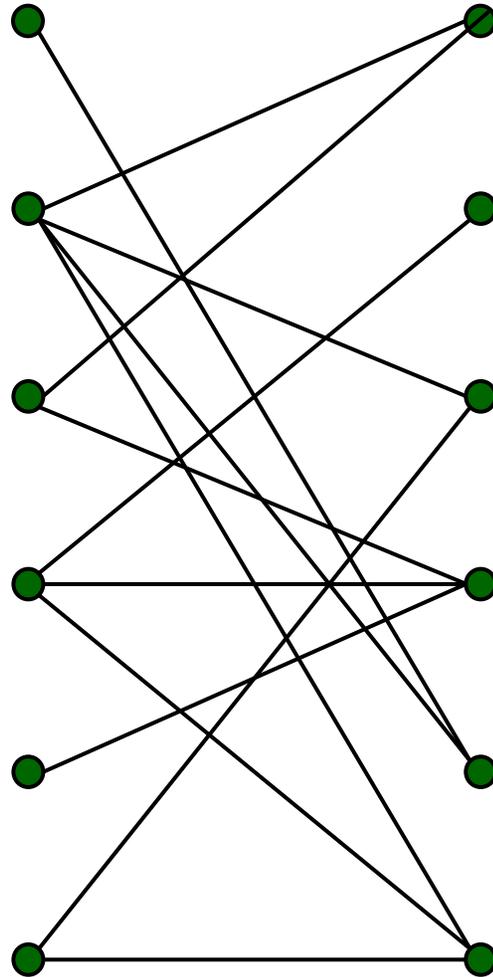
---

- **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

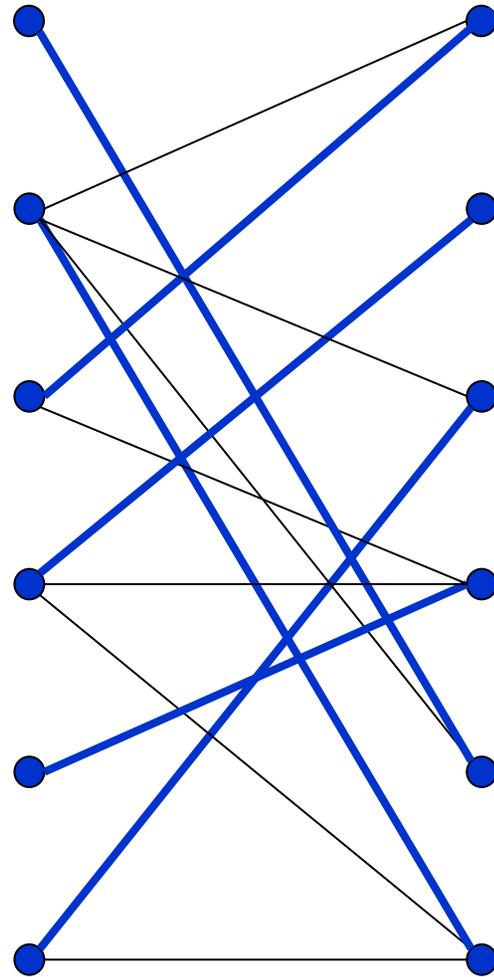


# Bipartite Matching

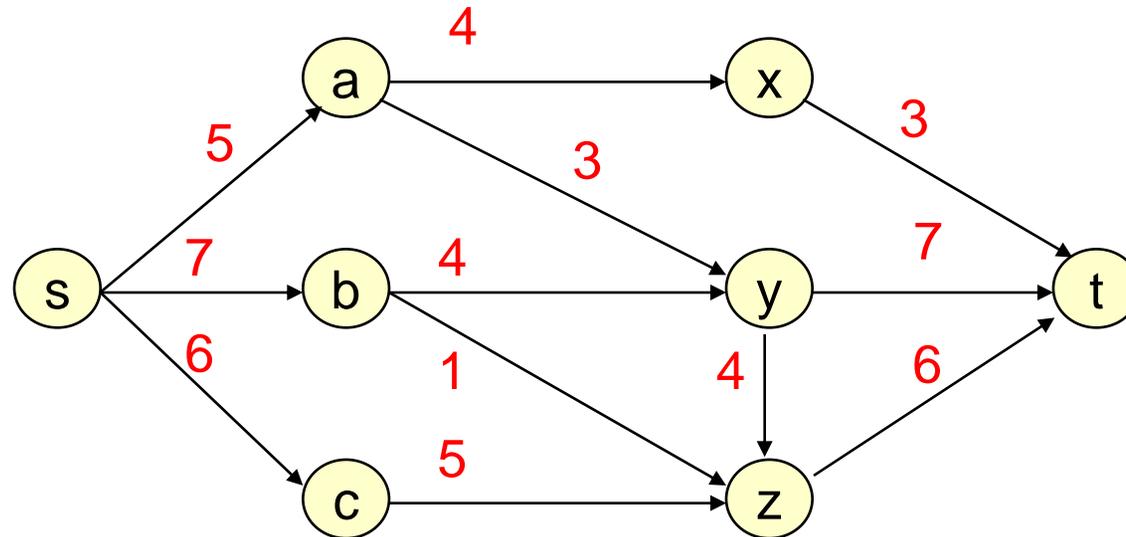
---



# Bipartite Matching

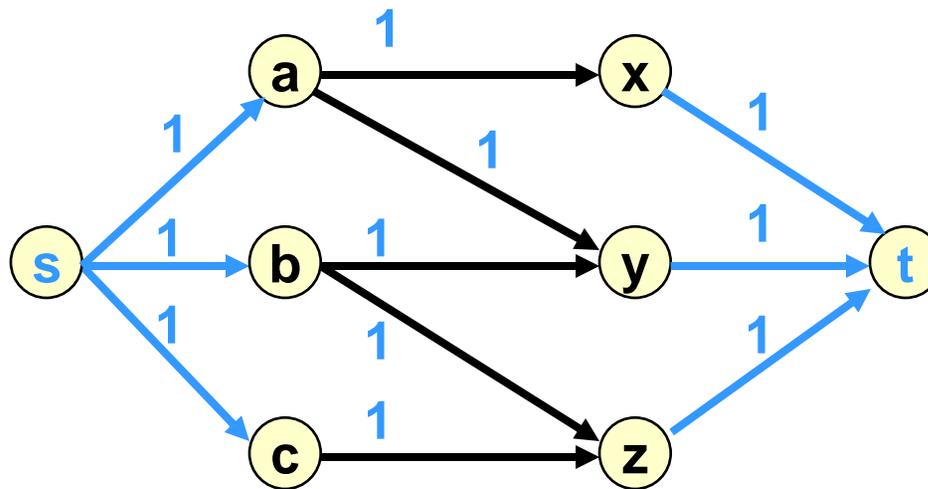


# The Network Flow Problem



- How much stuff can flow from **s** to **t**?

# Bipartite matching as a special case of flow



# Net Flow: Formal Definition

## Given:

A digraph  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

Two vertices  $\mathbf{s}, \mathbf{t}$  in  $\mathbf{V}$   
(**source** & **sink**)

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

## Find:

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

- $\mathbf{0} \leq \mathbf{f}(\mathbf{u}, \mathbf{v}) \leq \mathbf{c}(\mathbf{u}, \mathbf{v})$

[Capacity Constraint]

- if  $\mathbf{u} \neq \mathbf{s}, \mathbf{t}$ , i.e.  $\mathbf{f}^{\text{out}}(\mathbf{u}) = \mathbf{f}^{\text{in}}(\mathbf{u})$

[Flow Conservation]

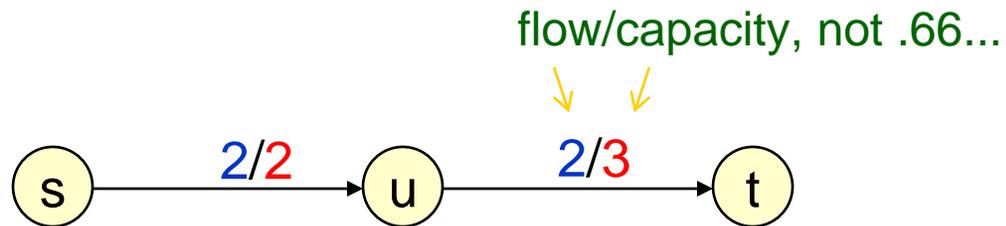
Maximizing total flow  $\mathbf{v}(\mathbf{f}) = \mathbf{f}^{\text{out}}(\mathbf{s})$

Notation:

$$\mathbf{f}^{\text{in}}(\mathbf{v}) = \sum_{\mathbf{e}=(\mathbf{u},\mathbf{v}) \in \mathbf{E}} \mathbf{f}(\mathbf{u}, \mathbf{v})$$

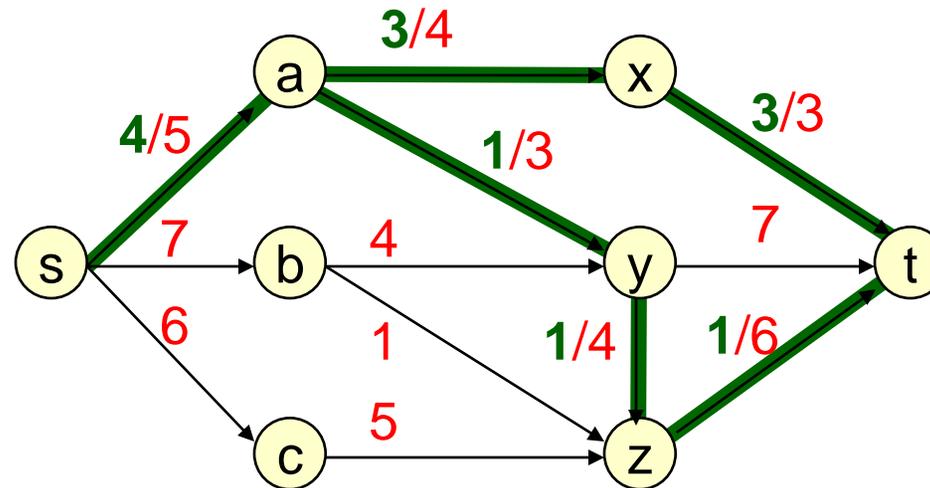
$$\mathbf{f}^{\text{out}}(\mathbf{v}) = \sum_{\mathbf{e}=(\mathbf{v},\mathbf{w}) \in \mathbf{E}} \mathbf{f}(\mathbf{v}, \mathbf{w})$$

# Example: A Flow Function



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

# Example: A Flow Function



- Not shown:  $f(u,v)$  if  $= 0$
- Note: **max flow**  $\geq 4$  since  $f$  is a flow function, with  $v(f) = 4$

# Max Flow via a Greedy Alg?

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

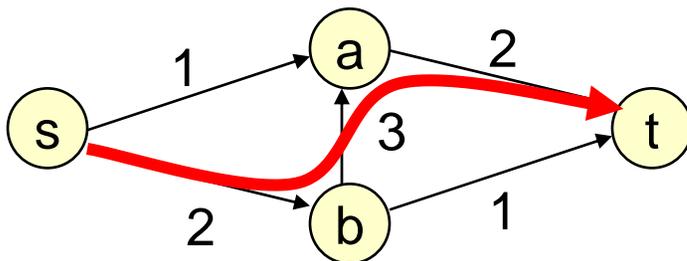
Pick such a path,  $p$

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

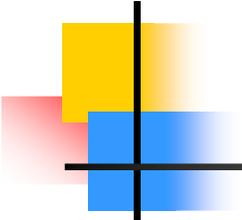
Subtract  $c$  from all capacities on  $p$

Delete edges of capacity  $0$

- This does **NOT** always find a max flow:



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



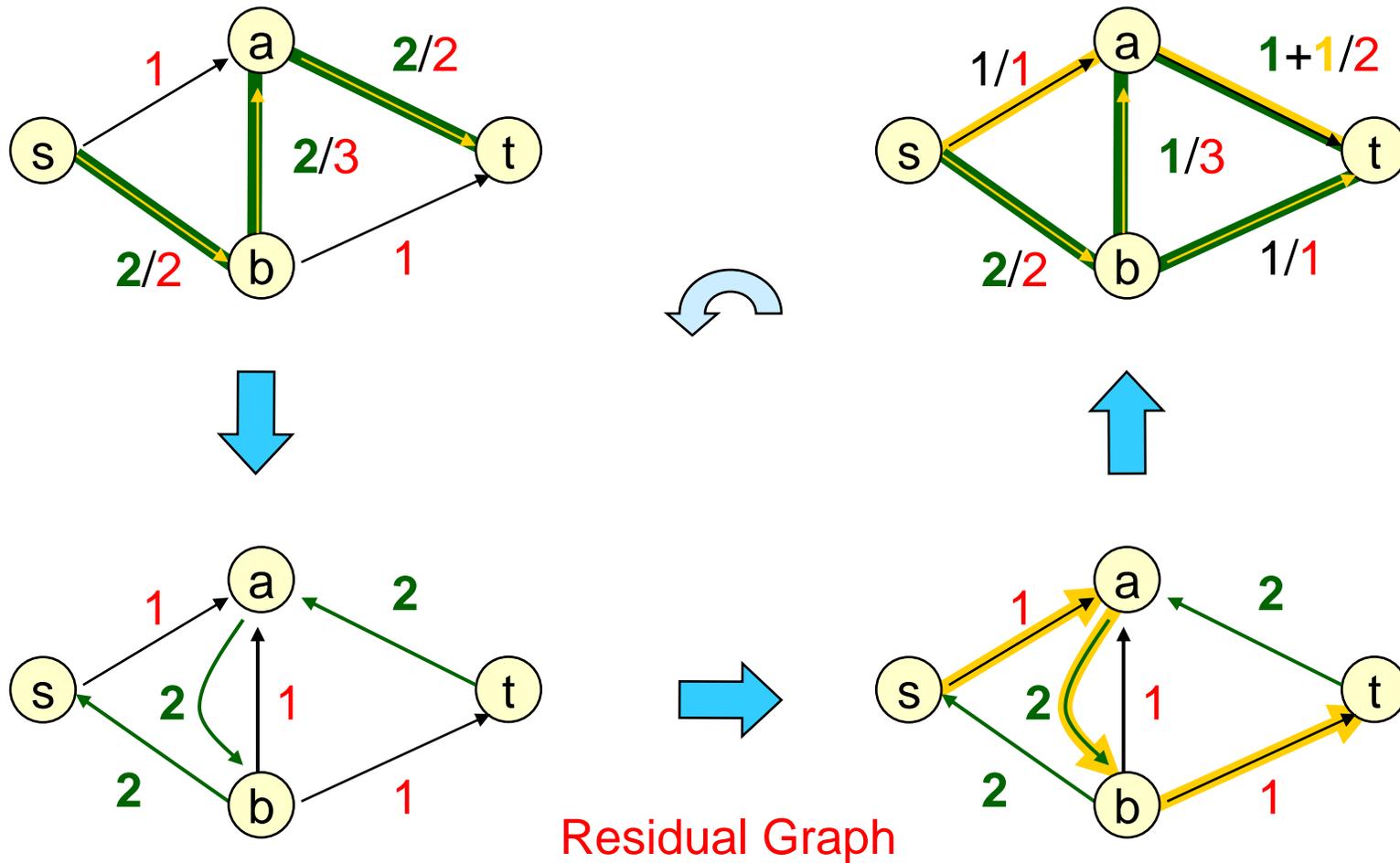
# A Brief History of Flow

#	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	Cheriyani & Hagerup	$E(nm + n^2 \log^2 n)$
15	1990	Cheriyani 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	$O(m^{3/2} \log(n^2/m) \log U)$ $O(n^{2/3} m \log(n^2/m) \log U)$

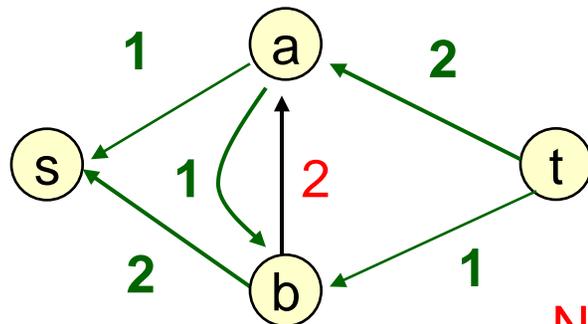
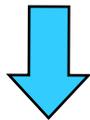
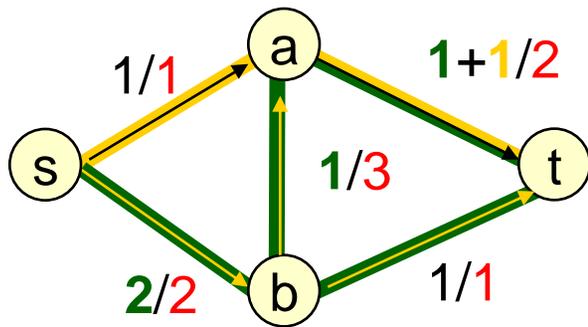
$n$  = # of vertices  
 $m$  = # of edges  
 $U$  = Max capacity

Source: Goldberg & Rao, FOCs '97

# Greed Revisited: Residual Graph & Augmenting Path



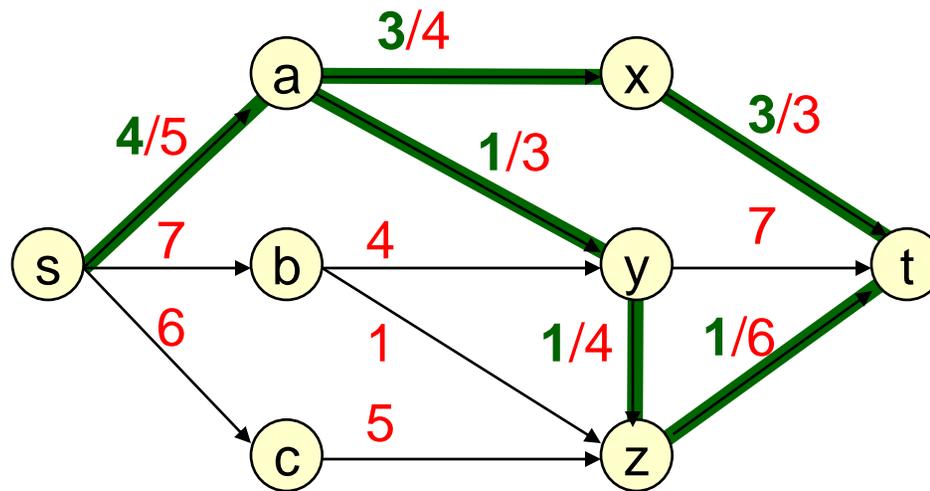
# Greed Revisited: An Augmenting Path



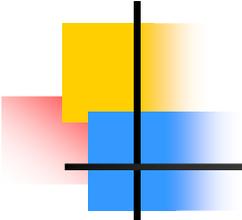
New Residual Graph

# Residual Capacity

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



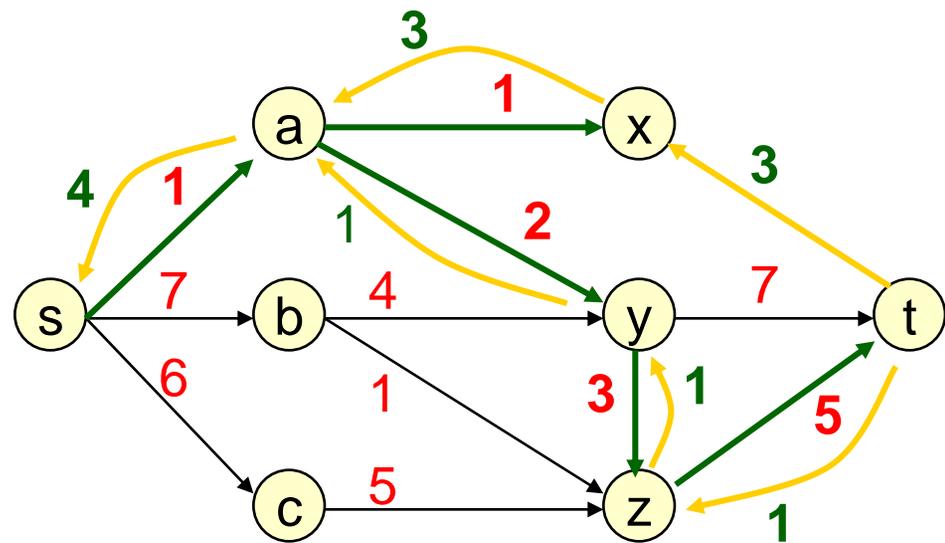
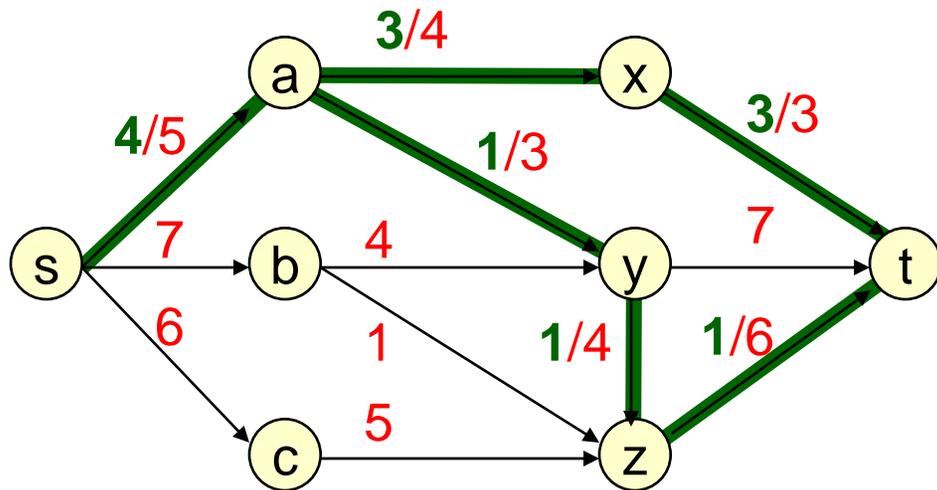
- e.g.  $c_f(s,b)=7$ ;  $c_f(a,x) = 1$ ;  $c_f(x,a) = 3$



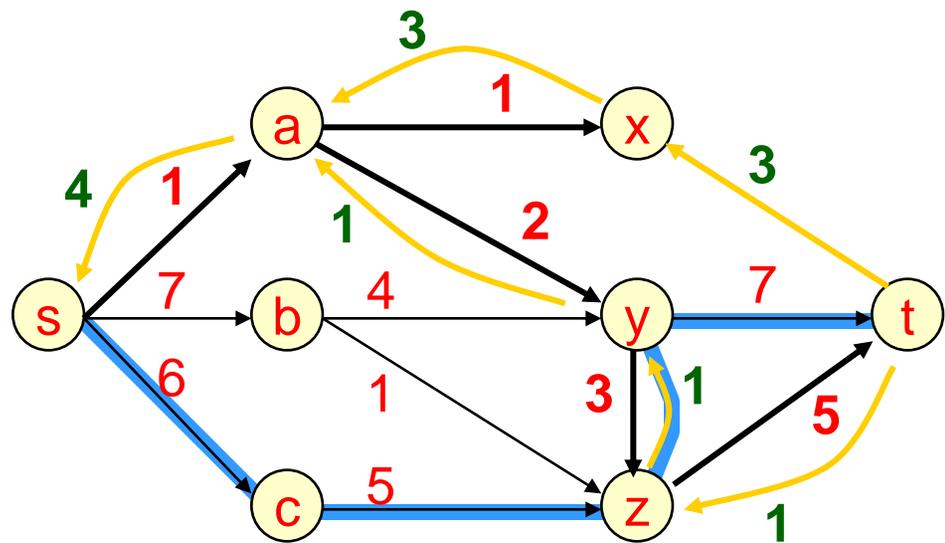
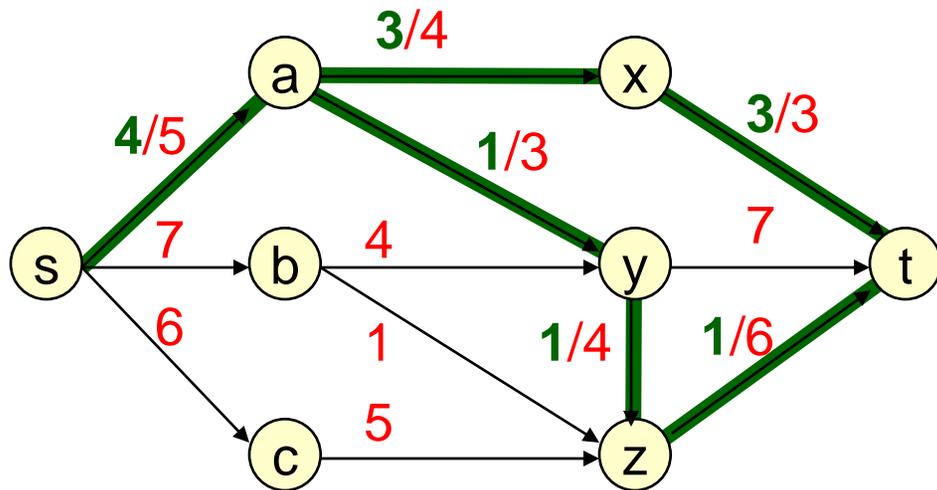
# Residual Graph & Augmenting Paths

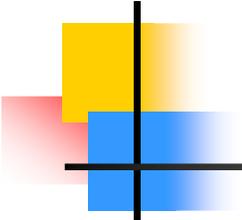
- The *residual graph* (w.r.t.  $\mathbf{f}$ ) is the graph  $\mathbf{G}_f = (\mathbf{V}, \mathbf{E}_f)$ , where
$$\mathbf{E}_f = \{ (\mathbf{u}, \mathbf{v}) \mid \mathbf{c}_f(\mathbf{u}, \mathbf{v}) > 0 \}$$
  - Two kinds of edges
    - **Forward** edges
      - $\mathbf{f}(\mathbf{u}, \mathbf{v}) 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

# A Residual Network



# An Augmenting Path





# Augmenting A Flow

---

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

$\mathbf{c}_P \leftarrow \min_{(u,v) \in P} \mathbf{c}_f(u,v)$  “bottleneck(**P**)”

for each **e**  $\in$  **P**

if **e** is a forward edge then

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

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

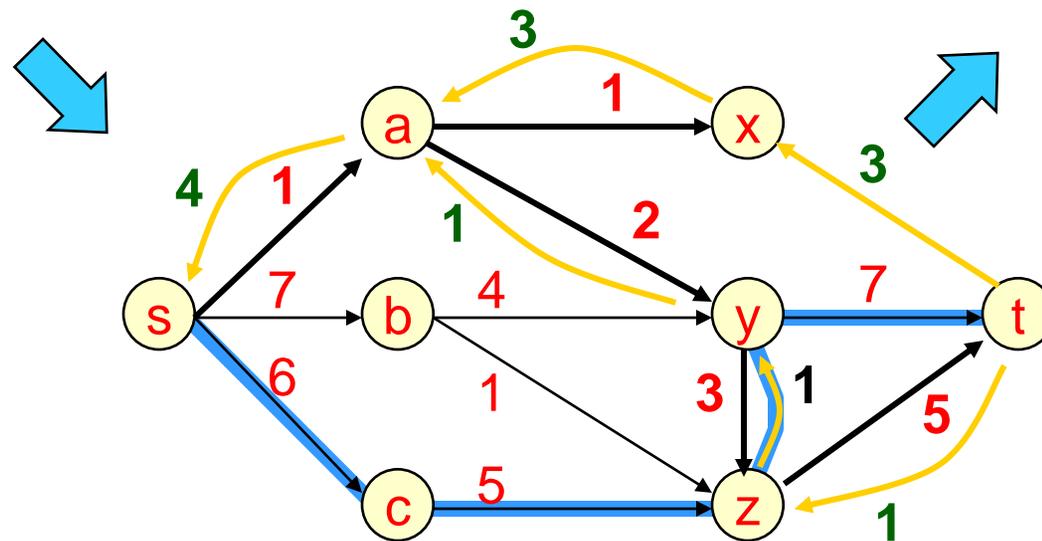
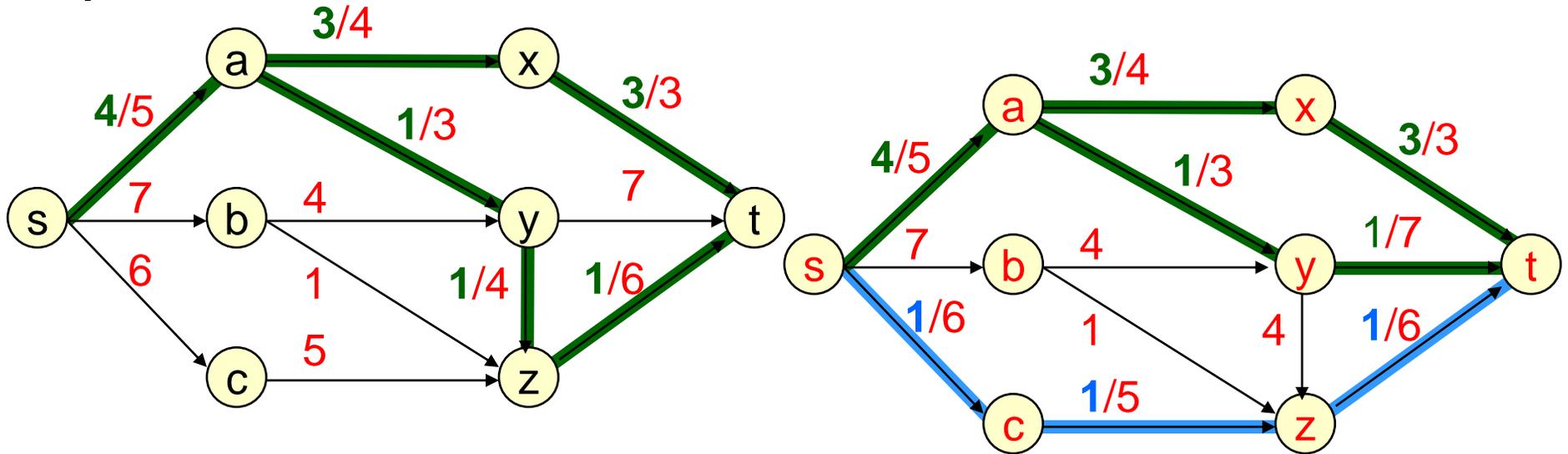
decrease **f**(**e**) by  $\mathbf{c}_P$

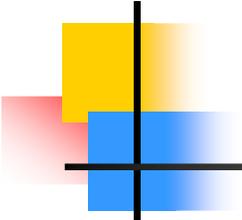
endif

endfor

return(**f**)

# Augmenting A Flow





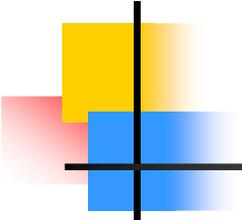
## Claim 7.1

---

If  $G_f$  has an augmenting path  $P$ , then the function  $f' = \text{augment}(f, P)$  is a legal flow.

**Proof:**

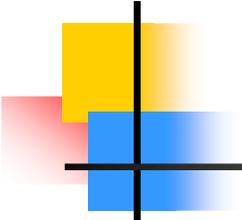
- $f'$  and  $f$  differ only on the edges of  $P$  so only need to consider such edges  $(u, v)$



## Proof of Claim 7.1

---

- If  $(u,v)$  is a forward edge then
$$\begin{aligned}f'(u,v) &= f(u,v) + c_p \leq f(u,v) + c_f(u,v) \\ &= f(u,v) + c(u,v) - f(u,v) \\ &= c(u,v)\end{aligned}$$
- If  $(u,v)$  is a backward edge then  $f$  and  $f'$  differ on flow along  $(v,u)$  instead of  $(u,v)$ 
$$\begin{aligned}f'(v,u) &= f(v,u) - c_p \geq f(v,u) - c_f(u,v) \\ &= f(v,u) - f(v,u) = 0\end{aligned}$$
- Other conditions like flow conservation still met



# Ford-Fulkerson Method

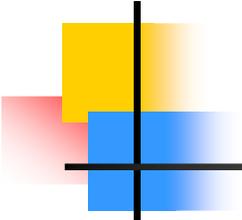
---

Start with  $f=0$  for every edge

While  $G_f$  has an augmenting path,  
augment

## ■ Questions:

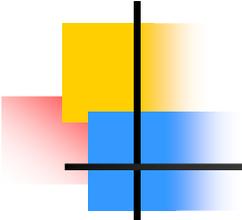
- Does it halt?
- Does it find a maximum flow?
- How fast?



# Observations about Ford-Fulkerson Algorithm

---

- At every stage the capacities and flow values are always integers (if they start that way)
- The flow value  $v(\mathbf{f}') = v(\mathbf{f}) + c_p > v(\mathbf{f})$  for  $\mathbf{f}' = \text{augment}(\mathbf{f}, \mathbf{P})$ 
  - Since edges of residual capacity  $0$  do not appear in the residual graph
- Let  $\mathbf{C} = \sum_{(s,u) \in E} \mathbf{c}(s,u)$ 
  - $v(\mathbf{f}) \leq \mathbf{C}$
  - $\mathbf{F}-\mathbf{F}$  does at most  $\mathbf{C}$  rounds of augmentation since flows are integers and increase by at least  $1$  per step



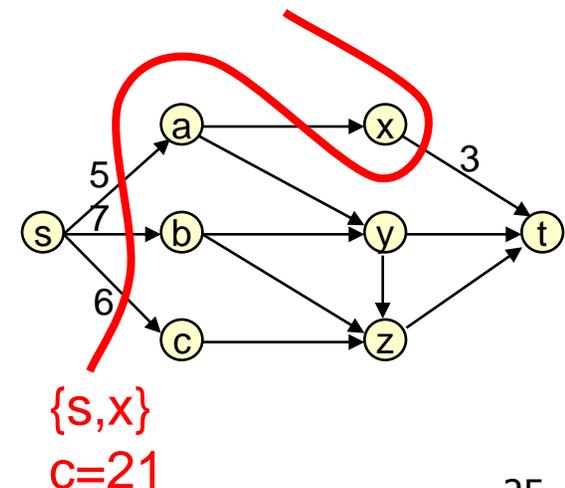
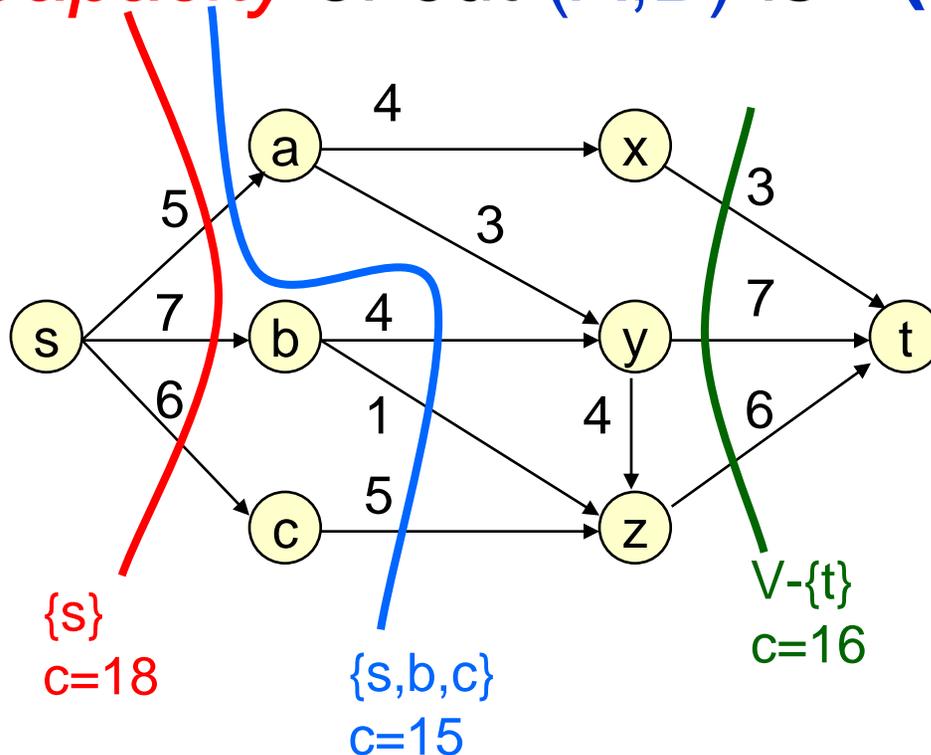
# Running Time of Ford-Fulkerson

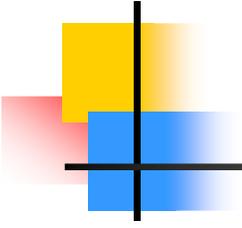
---

- For  $\mathbf{f=0}$ ,  $\mathbf{G_f=G}$
- Finding an augmenting path in  $\mathbf{G_f}$  is graph search  $\mathbf{O(n+m)=O(m)}$  time
- Augmenting and updating  $\mathbf{G_f}$  is  $\mathbf{O(n)}$  time
- Total  $\mathbf{O(mC)}$  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  $\mathbf{s-t}$ -path

# Cuts

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





# Convenient Definition

---

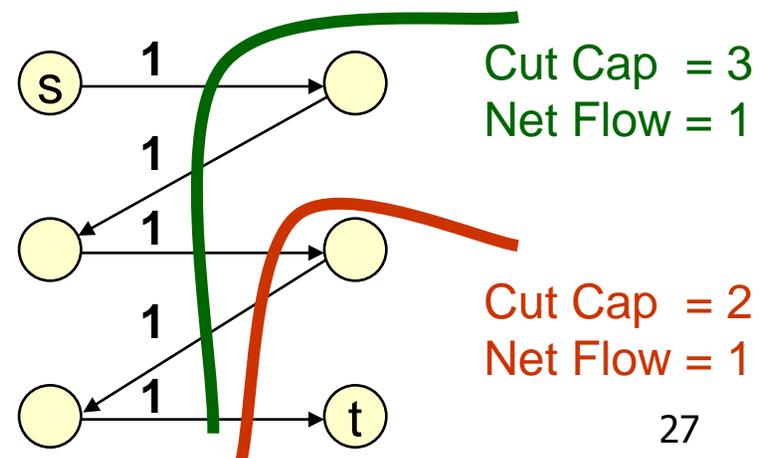
- $f^{\text{out}}(\mathbf{A}) = \sum_{v \in A, w \notin A} \mathbf{f}(v, w)$

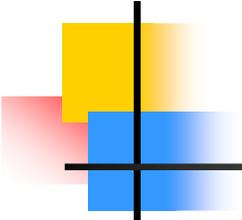
- $f^{\text{in}}(\mathbf{A}) = \sum_{v \in A, u \notin A} \mathbf{f}(u, v)$

# Claims 7.6 and 7.8

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

- **Corollary :**  
Max flow  $\leq$  Min cut



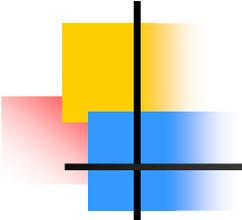


## Proof of Claim 7.6

---

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

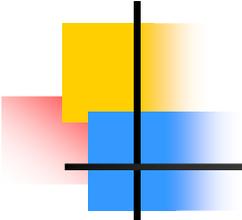
since all other vertices have  $f^{\text{out}}(v) = f^{\text{in}}(v)$
- $v(f) = f^{\text{out}}(s)$  and  $f^{\text{in}}(s) = 0$



## Proof of Claim 7.8

---

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



# Max Flow / Min Cut Theorem

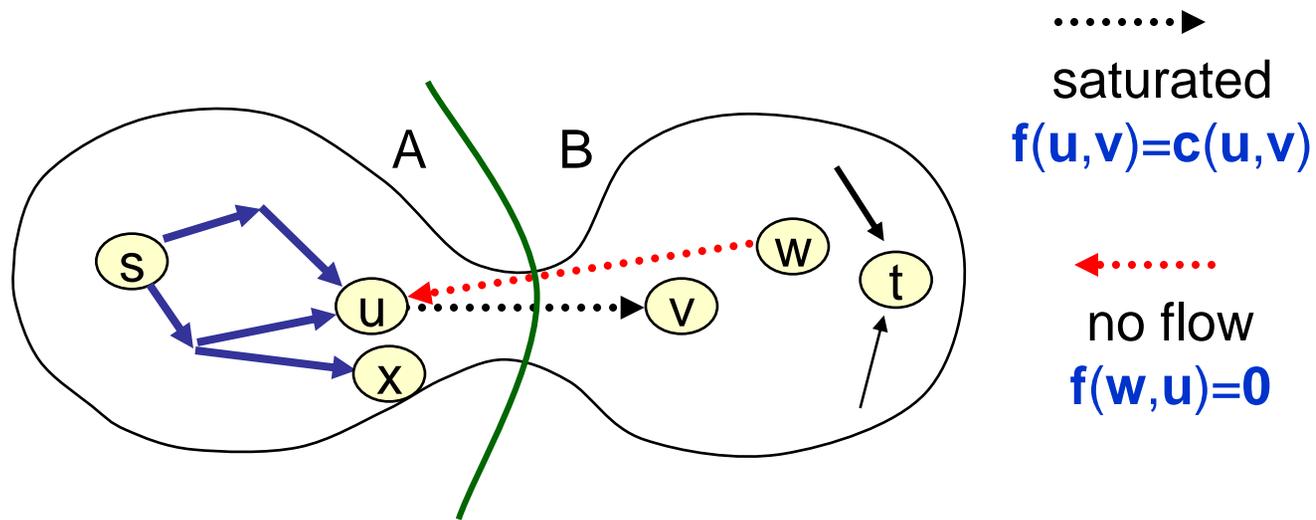
---

**Claim 7.9** For any flow  $f$ , if  $G_f$  has no augmenting path then there is some  $s$ - $t$ -cut  $(A, B)$  such that  $v(f) = c(A, B)$  (proof on next slide)

- We know by **Claims 7.6 & 7.8** that any flow  $f'$  satisfies  $v(f') \leq c(A, B)$  and we know that F-F runs for finite time until it finds a flow  $f$  satisfying conditions of **Claim 7.9**
  - Therefore by 7.9 for any flow  $f'$ ,  $v(f') \leq v(f)$
- **Corollary (1)** F-F computes a maximum flow in  $G$   
**(2)** For any graph  $G$ , the value  $v(f)$  of a maximum flow = minimum capacity  $c(A, B)$  of any  $s$ - $t$ -cut in  $G$

# Claim 7.9

Let  $A = \{ u \mid \exists \text{ an path in } G_f \text{ from } s \text{ to } u \}$   
 $B = V - A; s \in A, t \in B$



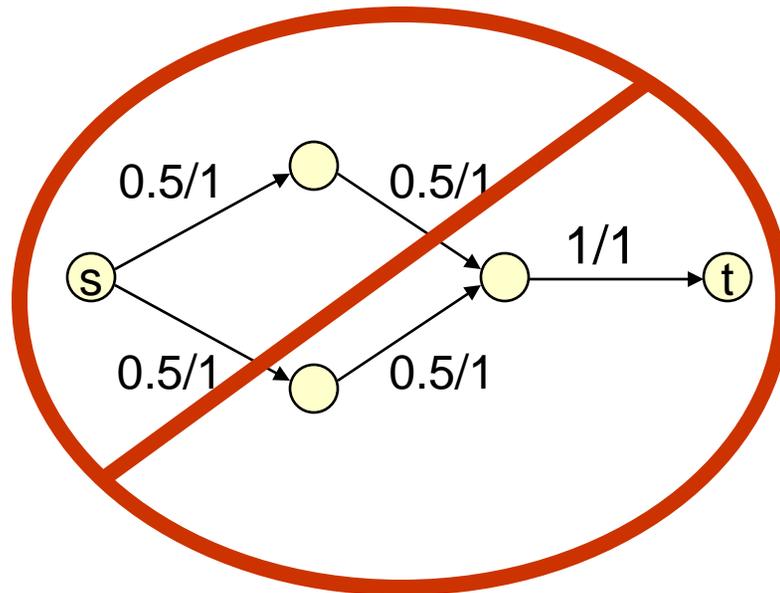
This is true for **every** edge crossing the cut, i.e.

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

# Flow Integrality Theorem

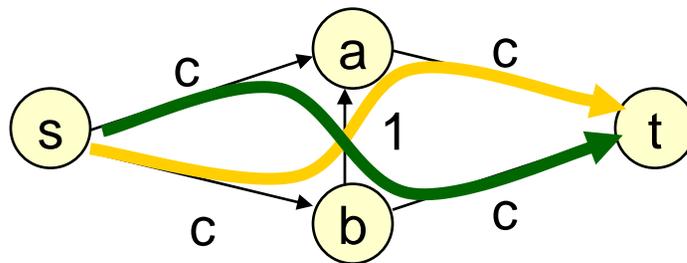
If all capacities are integers

- 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  $(u,v)$



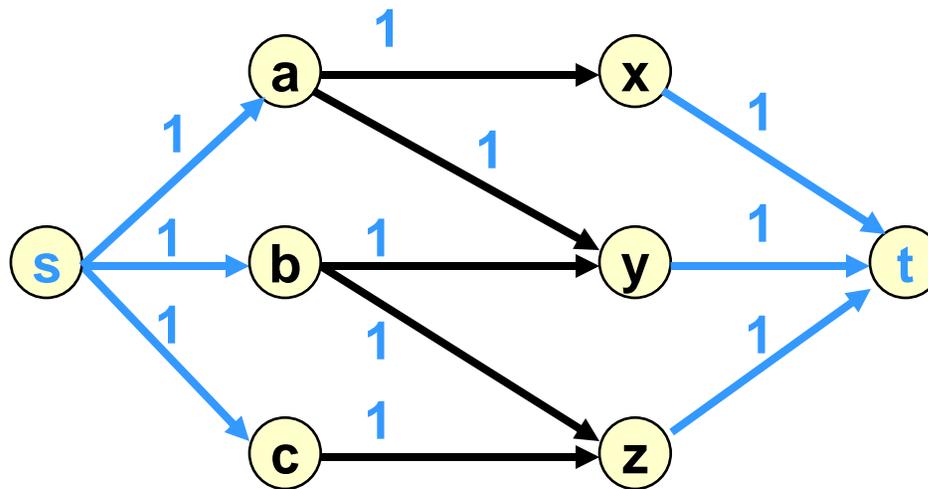
# Corollaries & Facts

- 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:



$c = 10^9$ , say

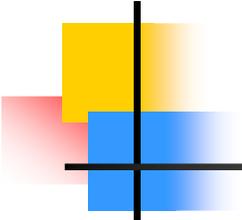
# Bipartite matching as a special case of flow



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

Therefore flow corresponds to a matching

$O(mC) = O(nm)$  running time



# Consequences of Ford-Fulkerson: Hall's Theorem

---

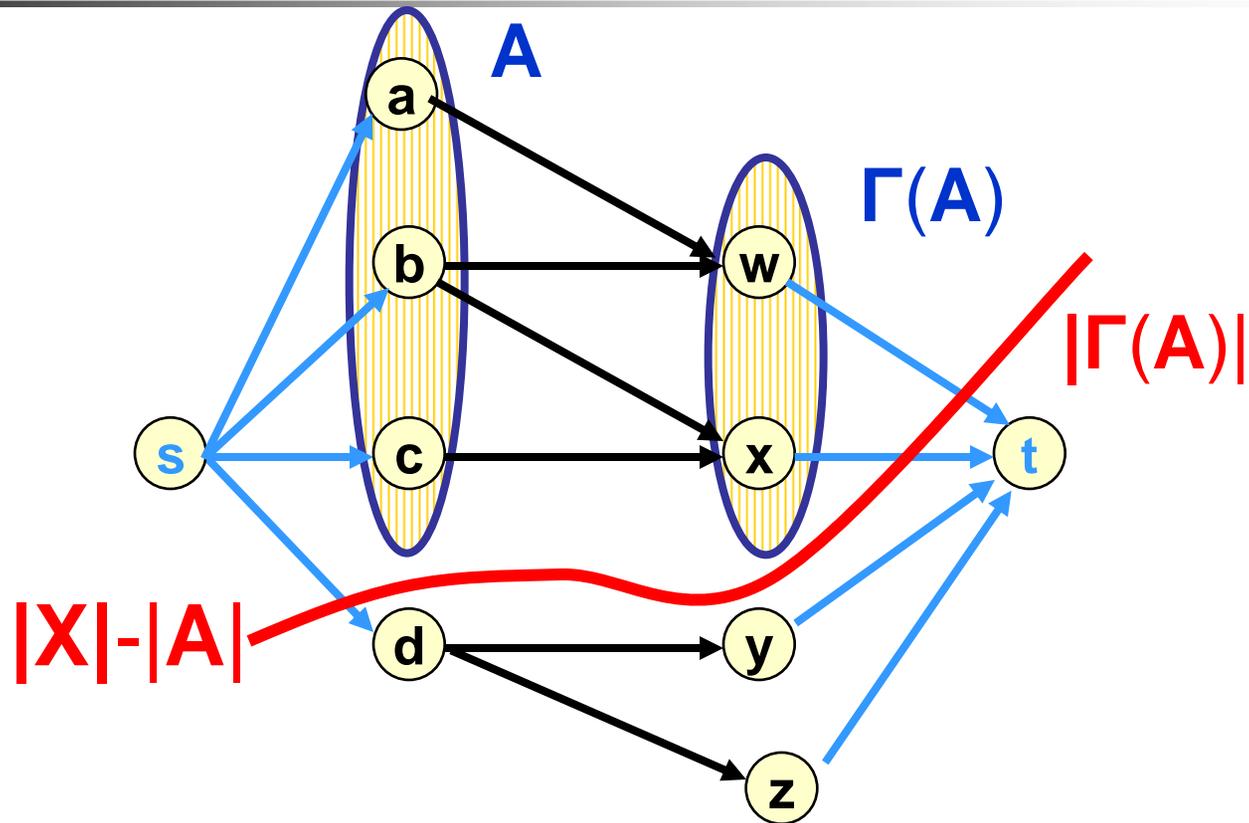
- **Def:** For a graph  $G=(V,E)$  and  $A \subseteq V$  let the neighborhood of  $A$  be

$$\Gamma(A) = \{w \in V \mid (v,w) \in E \text{ for some } v \in A\}$$

- **Hall's Theorem:** A graph  $G=(V,E)$  with bipartition  $V=X \cup Y$  where  $|X|=|Y|$  has a perfect matching if and only if for every set  $A \subseteq X$  we have  $|\Gamma(A)| \geq |A|$ .

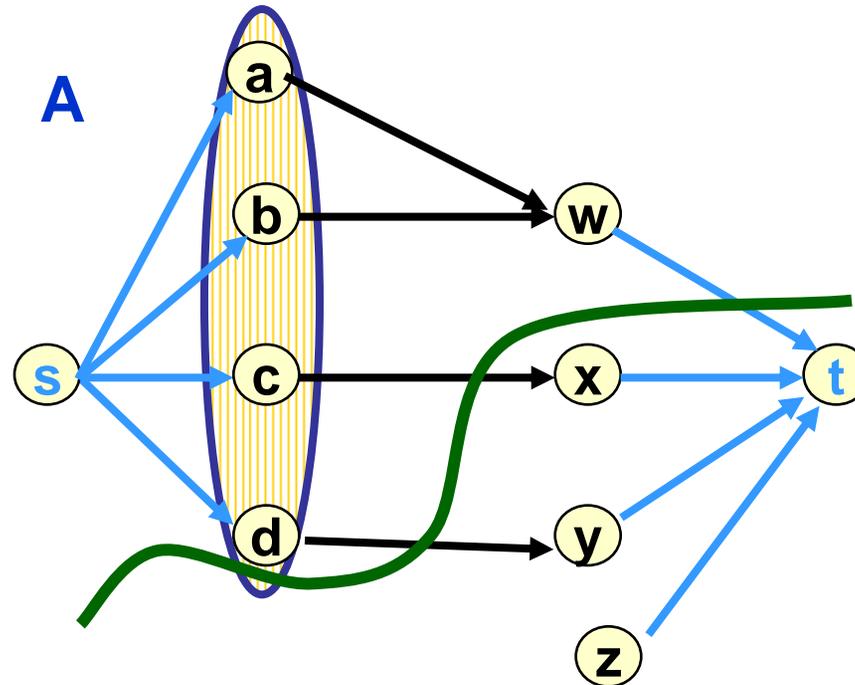
- **Proof:**  $\rightarrow$  If  $|\Gamma(A)| < |A|$  for some set  $A$

# Hall's Theorem Proof: Part 1



Min cut  $\leq |X|-|A|+|\Gamma(A)| < |X|$   
so matching is not perfect

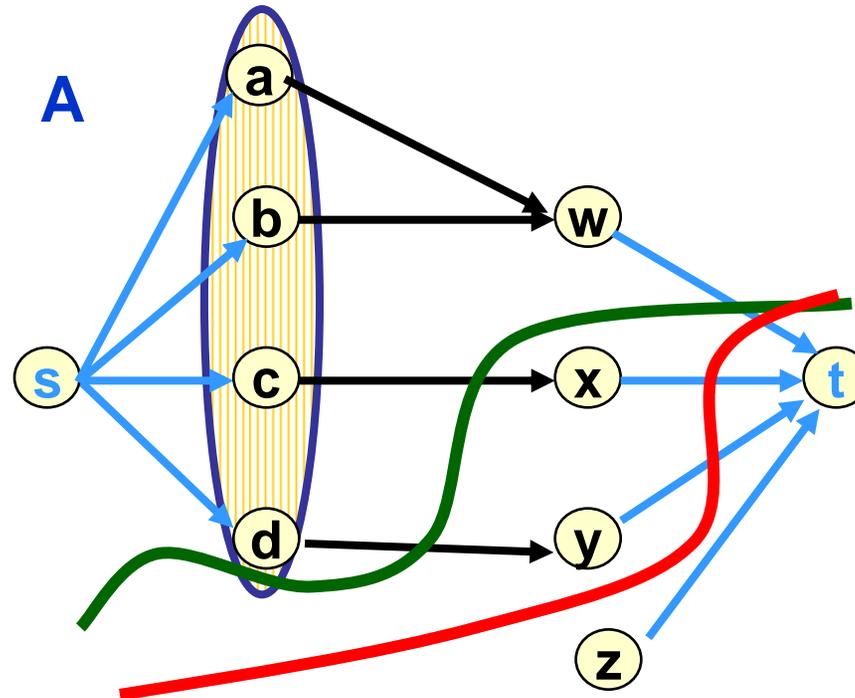
## Hall's Theorem Proof: Part 2



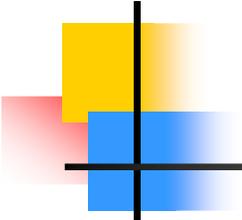
If no perfect matching there is some **st-cut**  $(A', B)$  with cut value  $< |X|$

- Let  $A$  = part of  $A'$  in  $X$

## Hall's Theorem Proof: Part 2



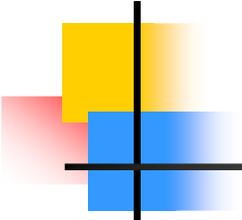
- Modify  $A'$  to include all vertices in  $\Gamma(A)$
- Value of mincut cannot go up so still  $< |X|$
- # of edges from  $s$  cut is  $|X| - |A|$
- # of edges to  $t$  cut is  $\geq |\Gamma(A)|$
- Total is  $\geq |X| - |A| + |\Gamma(A)|$  but  $< |X|$  so  $|\Gamma(A)| < |A|$



## Consequences of Ford-Fulkerson: Edge-disjoint paths

---

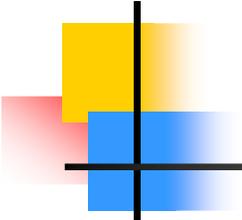
- Given a **directed** graph  $G=(V,E)$  and vertices  $s$  and  $t$  find a maximum set of edge-disjoint (simple) paths from  $s$  to  $t$ .
- **Lemma:** Any **0-1** flow  $f$  in the network flow graph on  $G=(V,E)$ ,  $s$ ,  $t$  with all capacities=1 contains a set of  $v(f)$  edge-disjoint simple paths from  $s$  to  $t$ .
- **Proof:** Follow the edges of positive flow through the graph. Flow conservation



# Edge-disjoint paths in directed graphs

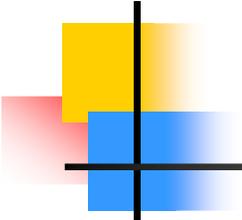
---

- Since a collection of  $k$  edge-disjoint paths is always a legal 0-1 flow of value  $k$ , by **Maxflow=Mincut** we have:
- **Directed Menger's Theorem:** The maximum number of edge-disjoint paths from  $s$  to  $t$  is equal to the minimum number of edges that need to be cut to separate  $s$  from  $t$ .
- We can use Ford-Fulkerson to find the flow in  $O(mn)$  time (since capacities are all  $1$ ) and then prune the flow as on previous slide in  $O(m+n)$  extra time.



# Edge-disjoint paths in undirected graphs

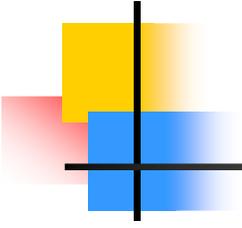
- Given an undirected graph  $G=(V,E)$  and nodes  $s$  and  $t$ , we can modify  $G$  to create a directed graph  $G'=(V,E')$  where we include both  $(u,v)$  and  $(v,u)$  in  $E'$  whenever  $\{u,v\}$  is in  $E$ .
- **Observation:** If in a flow  $f$  we have  $f(v,u)=f(u,v) > 0$  then we can set  $f(v,u)=f(u,v)=0$  and keep the same flow value
- **Algorithm:** Run F-F on  $G'$  in  $O(mn)$  time and then remove flow cycles in  $O(m+n)$  time.
- **Undirected Menger's Theorem:** The maximum number of edge-disjoint paths from  $s$  to  $t$  is equal to the minimum number of edges that need to be cut to separate  $s$  from  $t$ .



# Polynomial running time?

---

- F-F uses  $O(mC)$  time where  $C$  is the total capacity leaving  $s$ .
  - If all capacities are at most some upper bound  $U$  then  $C < nU$  but each input capacity takes only  $\log_2 U$  bits to represent so the running time is not polynomial in the input size
- We will give two modifications that yield polynomial-time algorithms for general network flow problems

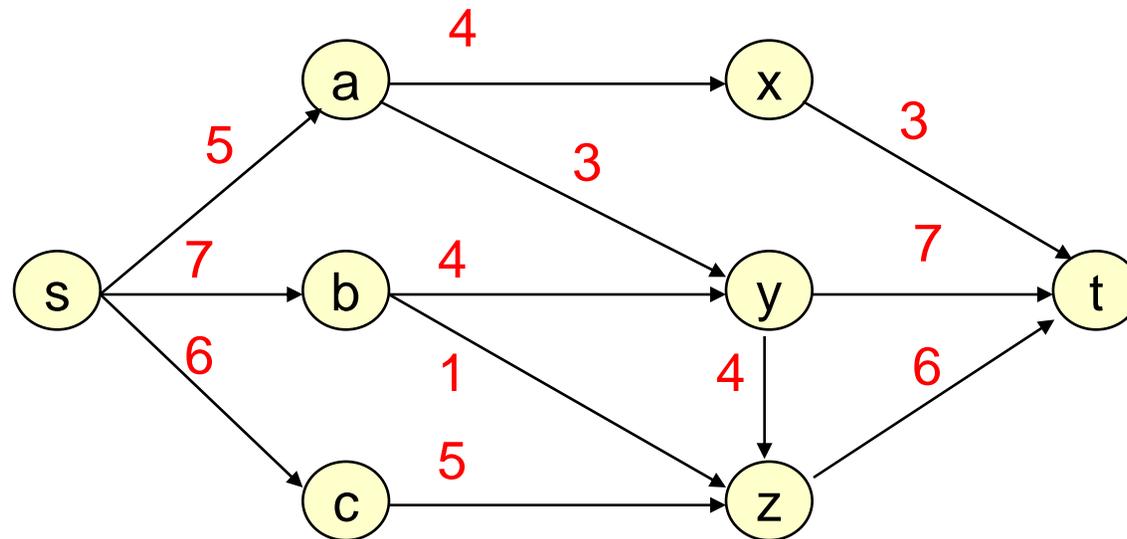


# Capacity-scaling algorithm

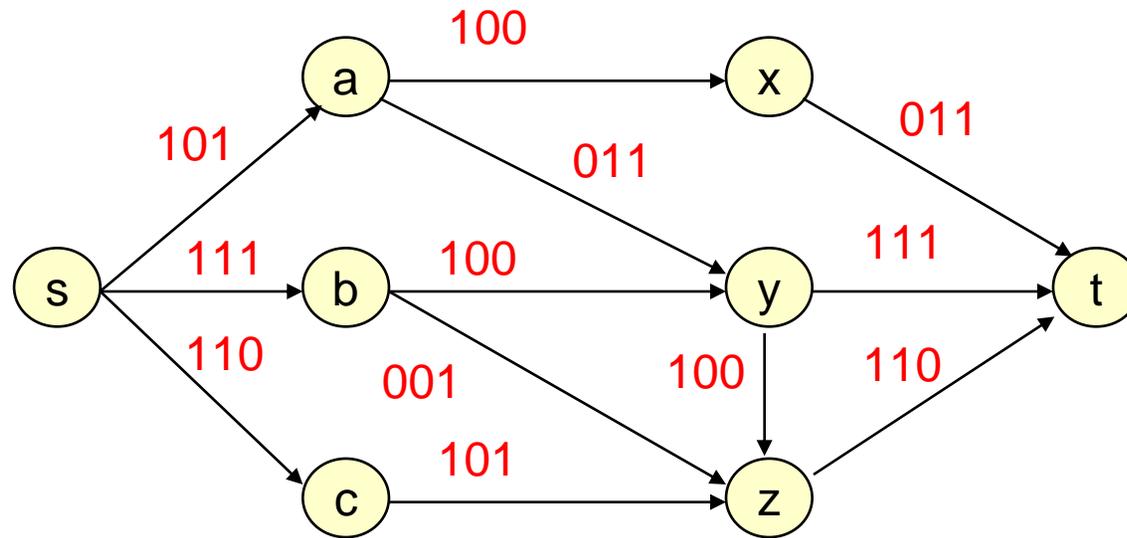
---

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

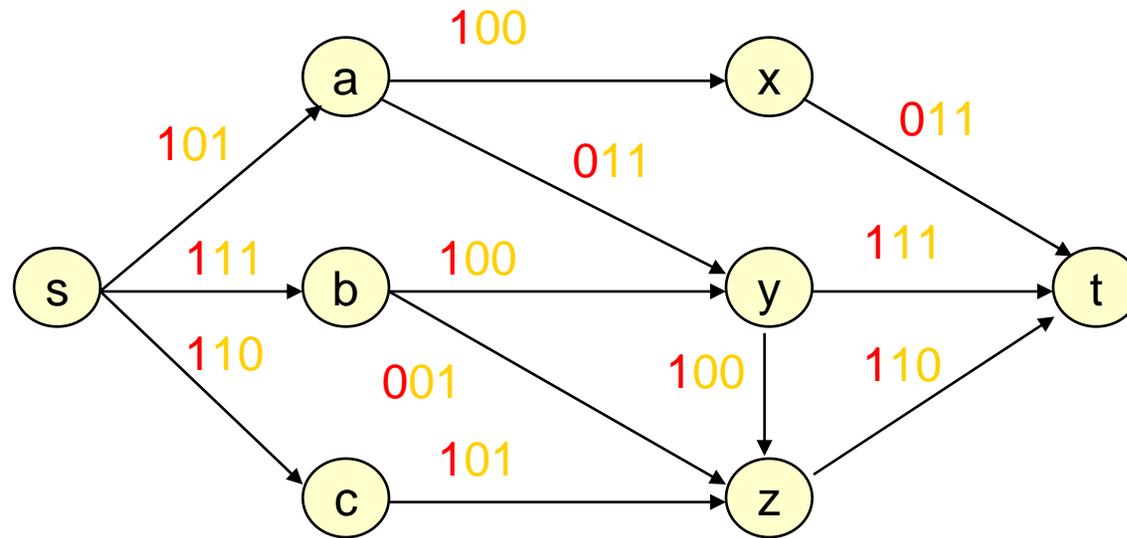
# Capacity Scaling



# Capacity Scaling

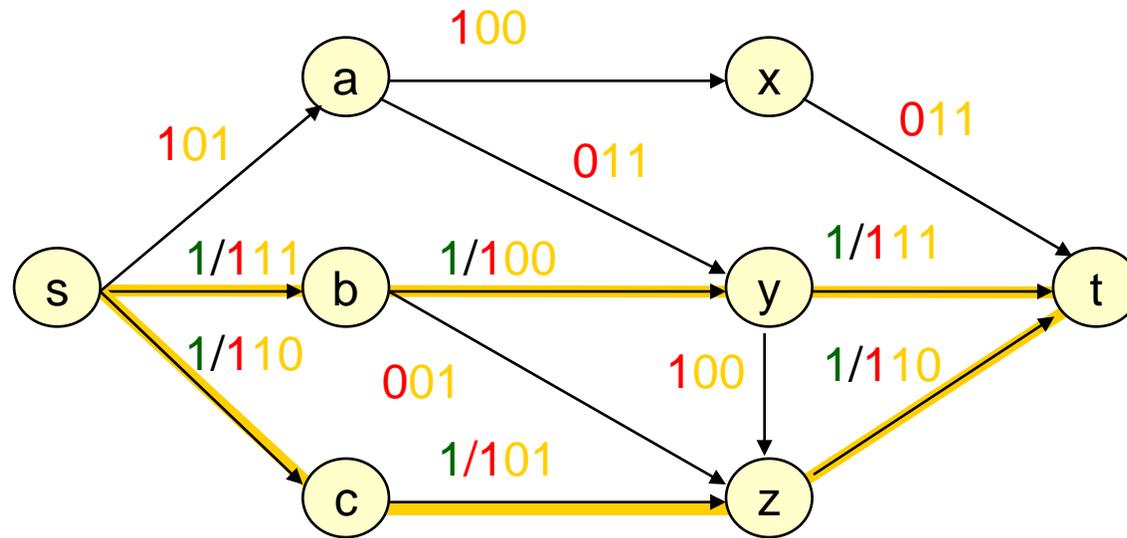


# Capacity Scaling Bit 1



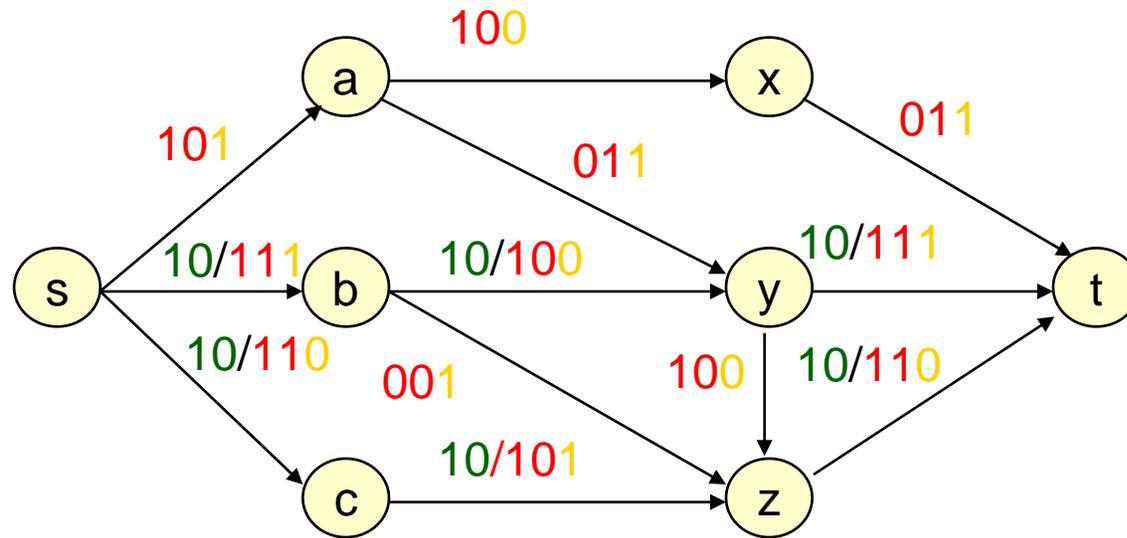
Capacity on each edge is at most **1**  
(either **0** or **1** times  $\Delta=4$ )

# Capacity Scaling Bit 1



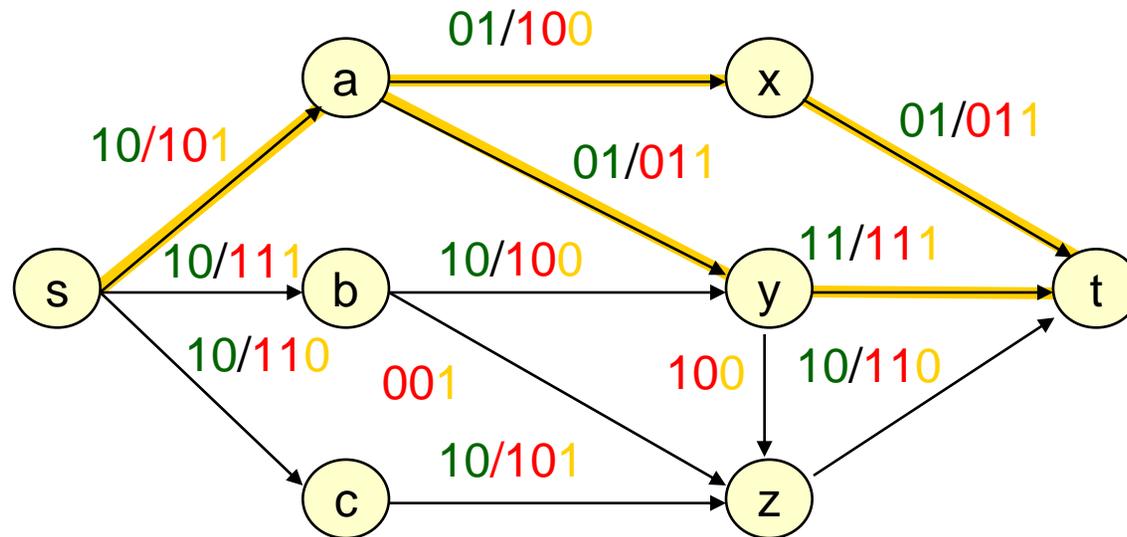
$O(nm)$  time

# Capacity Scaling Bit 2



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

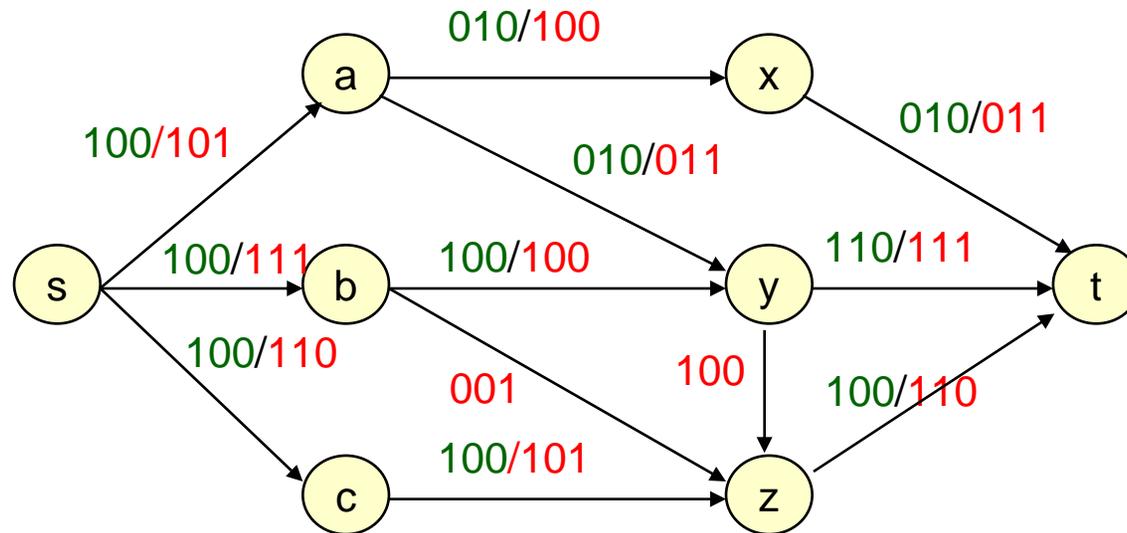
# Capacity Scaling Bit 2



Residual capacity across min cut is at most  $m$

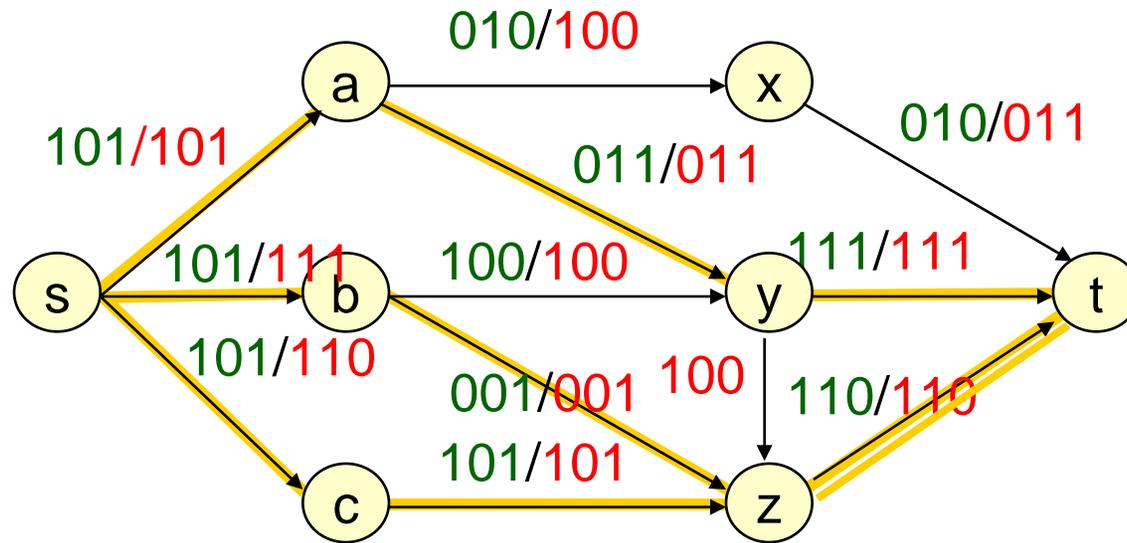
$\Rightarrow \leq m$  augmentations

# Capacity Scaling Bit 3



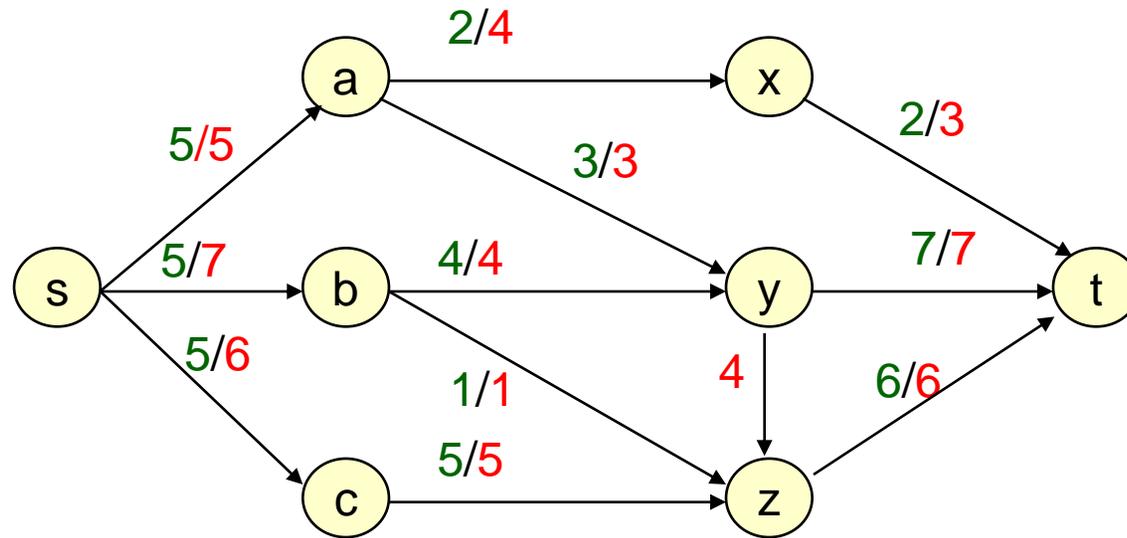
Residual capacity across min cut is at most **m**  
(either **0** or **1** times  $\Delta=1$ )

# Capacity Scaling Bit 3

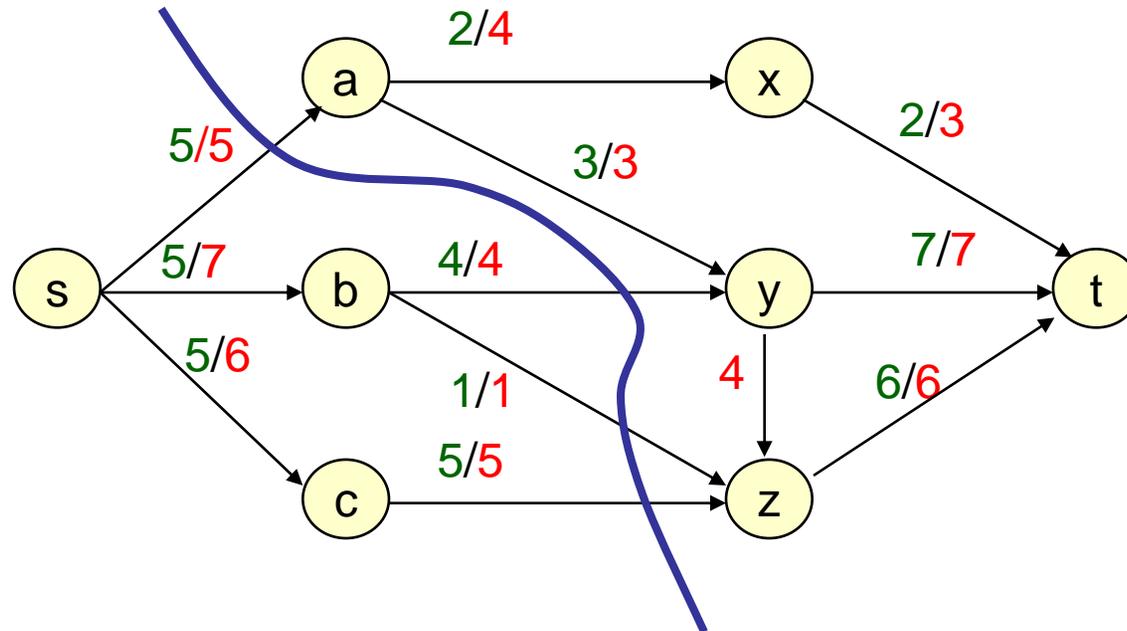


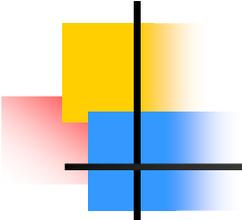
After  $\leq m$  augmentations

# Capacity Scaling Final



# Capacity Scaling Min Cut

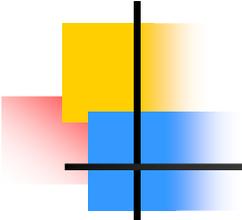




# Total time for capacity scaling

---

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



# Edmonds-Karp Algorithm

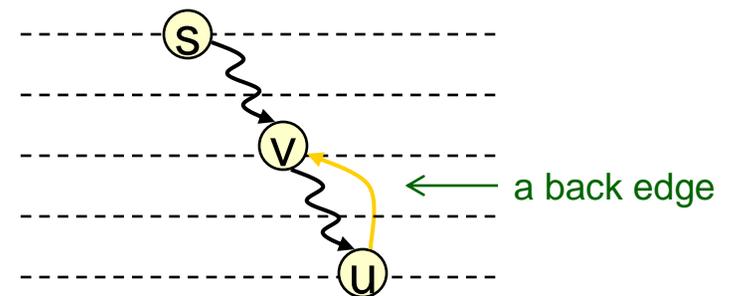
---

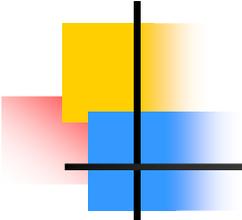
- Use a **shortest** augmenting path  
(via Breadth First Search in residual graph)
- Time:  $O(n m^2)$

# BFS/Shortest Path Lemmas

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





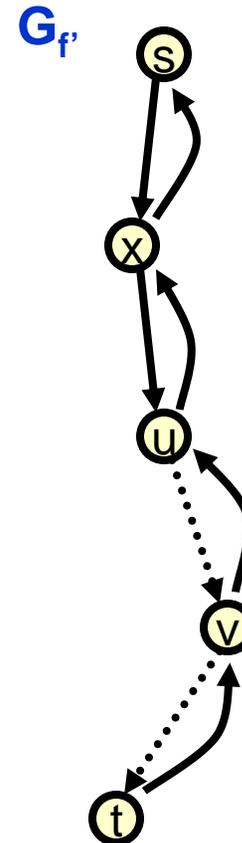
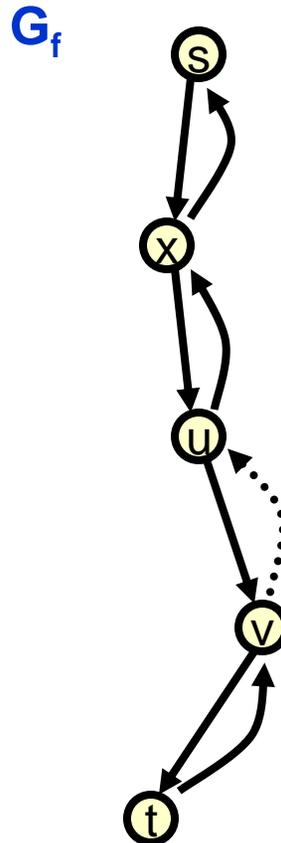
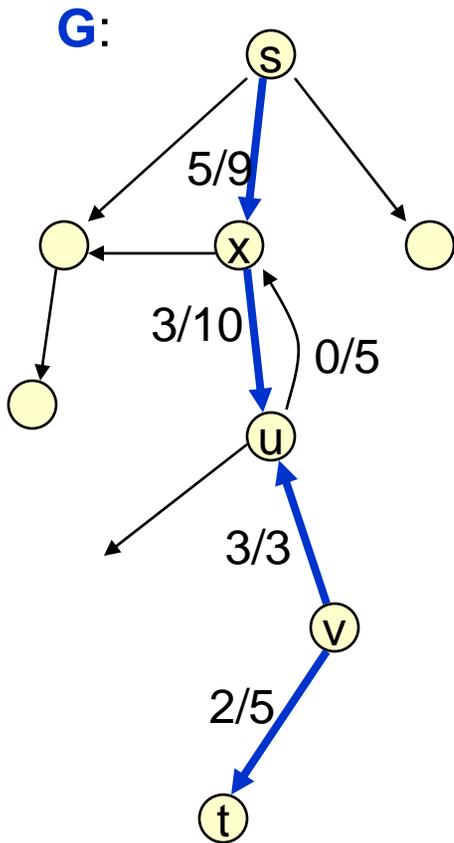
## Key Lemma

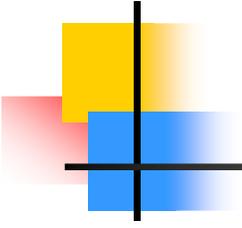
---

Let  $f$  be a flow,  $G_f$  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:

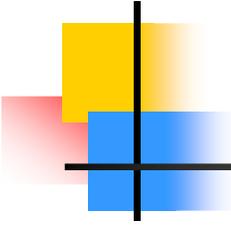
Call  $(u,v)$  **critical** for augmenting path  $P$  if it's closest to  $s$  having min residual capacity

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

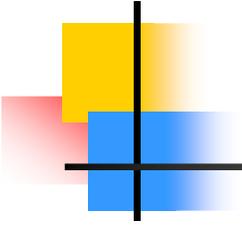




## Corollary

---

- Edmonds-Karp runs in  $O(nm^2)$  time



# Project Selection

## a.k.a. The Strip Mining Problem

---

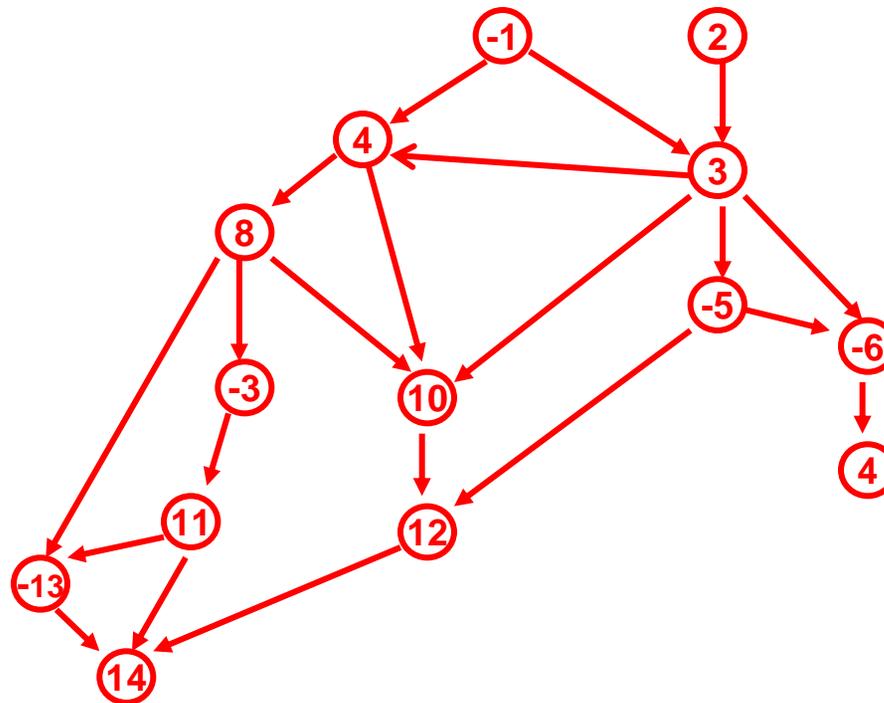
### ■ Given

- 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  $\mathbf{Profit}(\mathbf{A})=\sum_{\mathbf{v}\in\mathbf{A}} \mathbf{p}(\mathbf{v})$

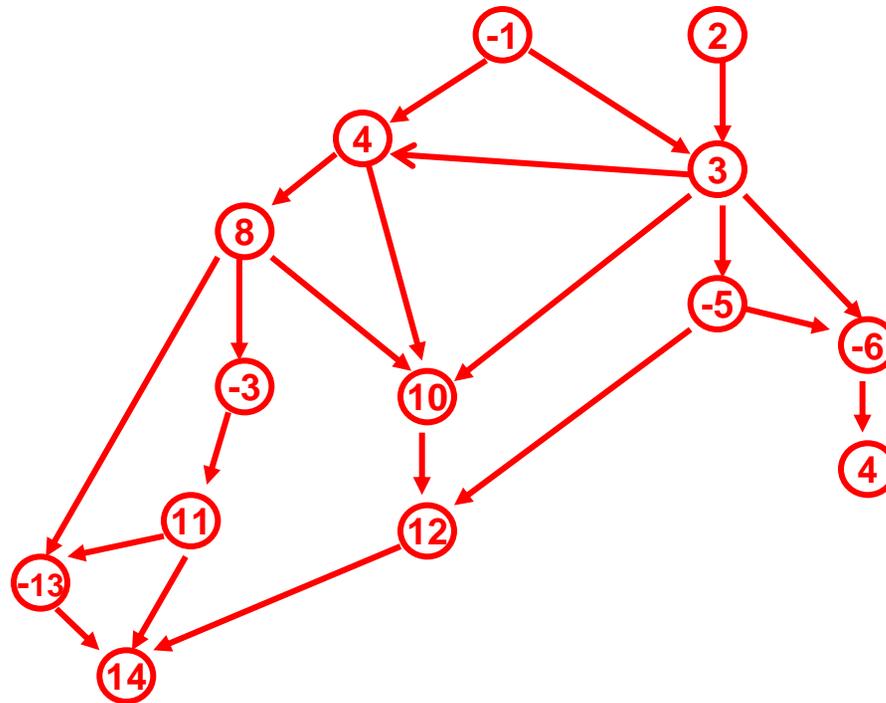
# Project Selection Graph



Each task points to its predecessor tasks

# Extended Graph

(s)



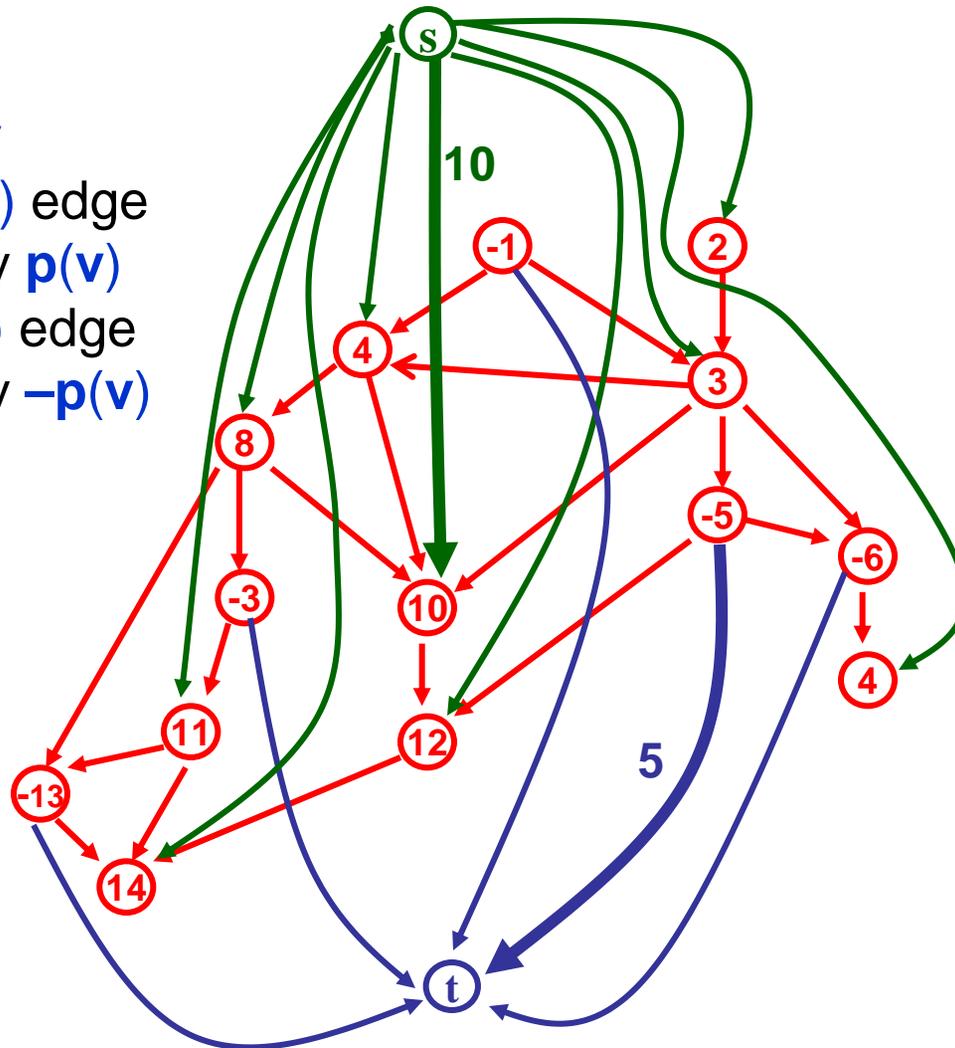
(t)

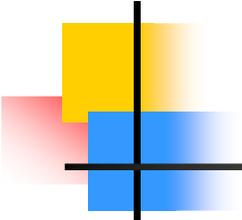
# Extended Graph $G'$

For each vertex  $v$

If  $p(v) \geq 0$  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  $C = \sum_{v: p(v) \geq 0} p(v)$ 
  - $\text{Profit}(A) \leq C$  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  $C$



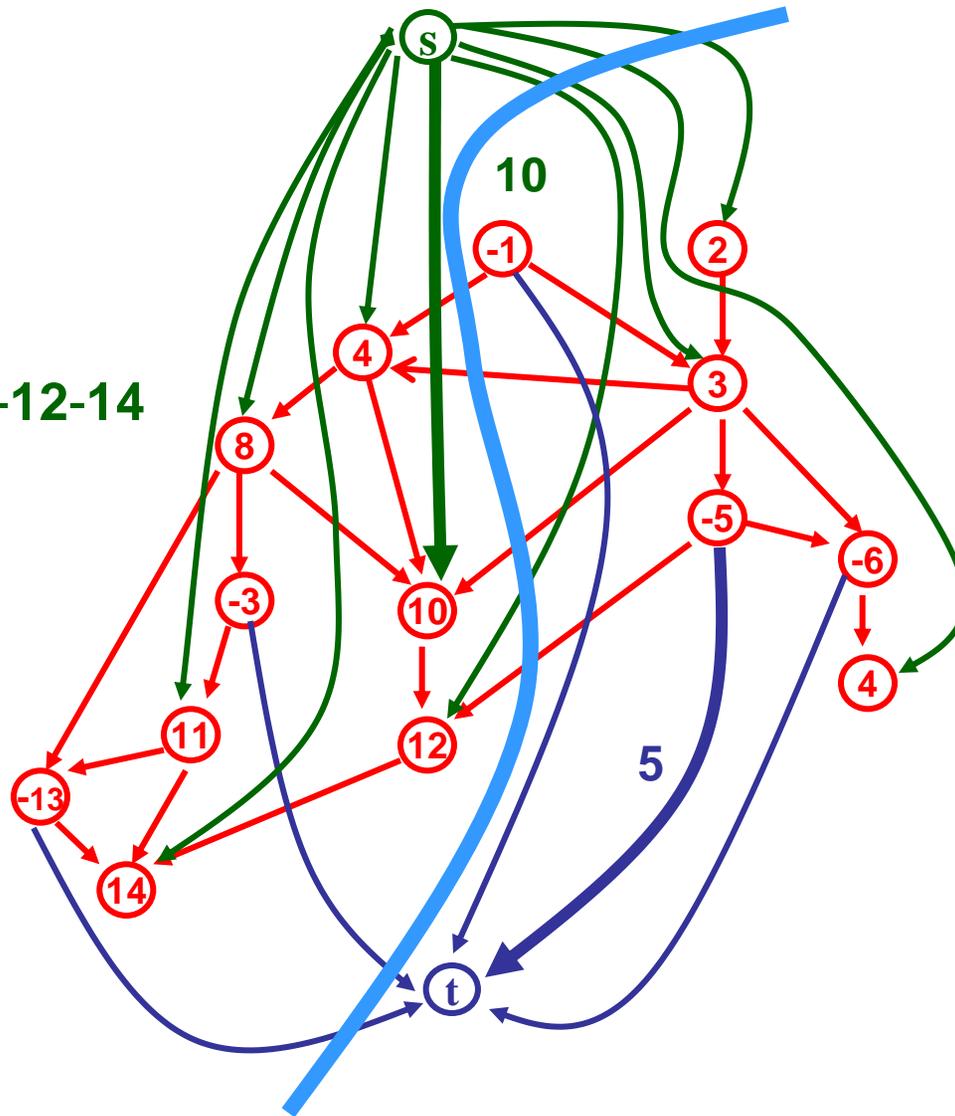
# Extended Graph G'

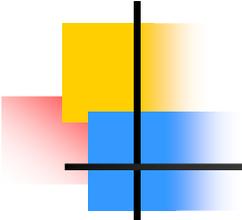
Cut value

$$= 13 + 3 + 2 + 3 + 4$$

$$= 13 + 3$$

$$+ C - 4 - 8 - 10 - 11 - 12 - 14$$





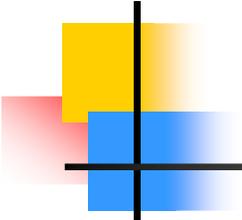
# Project Selection

---

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

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

- **Corollary** A minimum cut  $(\mathbf{S}, \mathbf{T})$  in  $\mathbf{G}'$  yields an optimal solution  $\mathbf{A} = \mathbf{S} - \{\mathbf{s}\}$  to the profit selection problem
- **Algorithm** Compute maximum flow  $\mathbf{f}$  in  $\mathbf{G}'$ , find the set  $\mathbf{S}$  of nodes reachable from  $\mathbf{s}$  in  $\mathbf{G}'_{\mathbf{f}}$  and return  $\mathbf{S} - \{\mathbf{s}\}$



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