# Network flow

Reading: Kleinberg & Tardos
Ch. 7.1

# Save the lemmings!

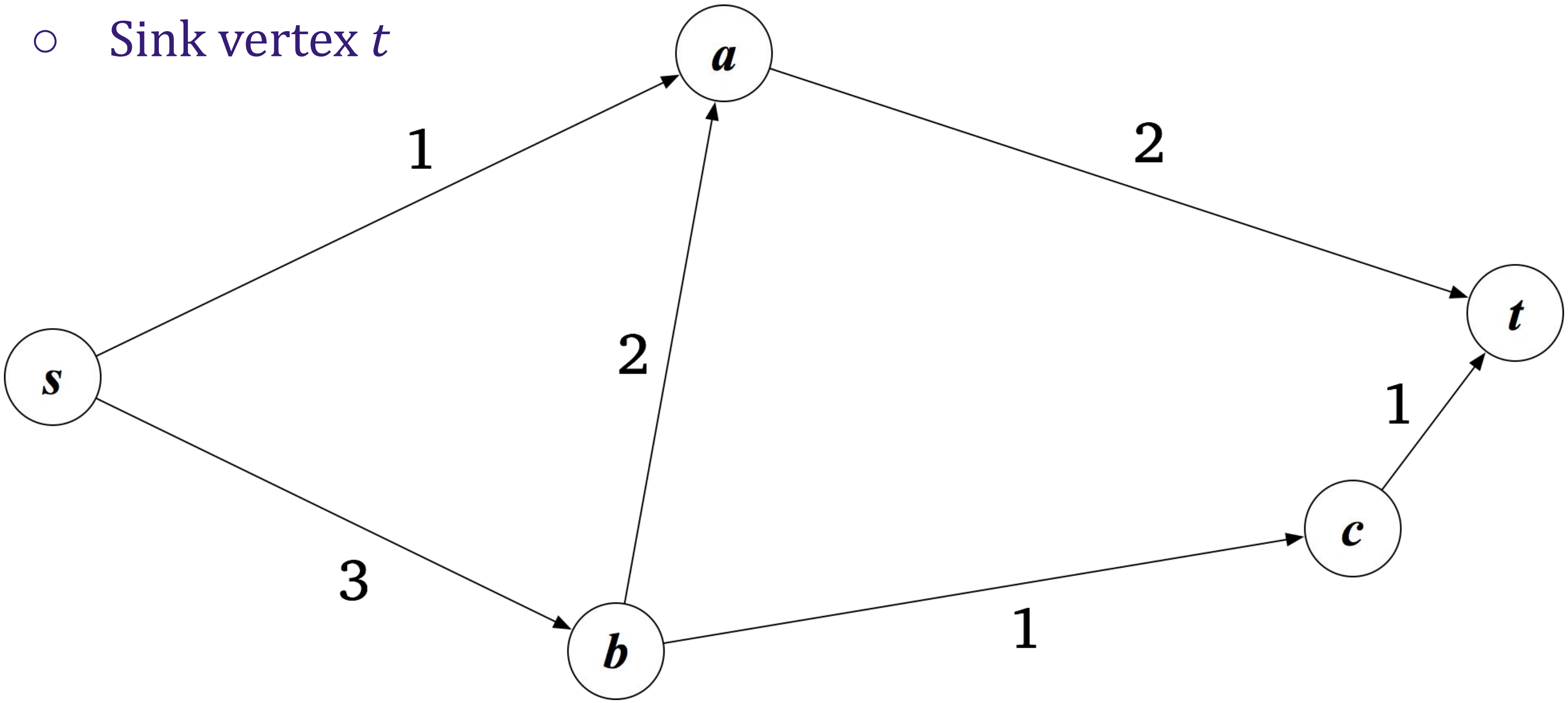
You’ve been chosen to guide lemmings to safety! They enter from one door and must leave out another. There may be multiple ways for them to get from the entrance to the exit


*How can you route the lemmings to save as many as possible?*

# “Demo”

# Step 1: Problem formulation

- Input:
  - Directed graph G = (V,E) with edge capacities $c_e$ for each edge $e$
  - Source vertex $s$
  - Sink vertex $t$



# Step 1: Problem formulation

- Input:
  - Directed graph G = (V,E) with edge capacities $c_e$ for each edge $e$
  - Source vertex $s$
  - Sink vertex $t$
- Find maximum “flow” from source to target
  - ***s*-*t* flow** $f$:$E \rightarrow \mathbf{R}^+$ such that
    - (capacity) $0 \leq f(e) \leq c_e$ for each edge $e$
    - (conservation) for each **internal** vertex $v$ (not $s$ or $t$)

# Step 1: Problem formulation

- Input:
  - Directed graph G = (V,E) with edge capacities $c_e$ for each edge $e$
  - Source vertex $s$
  - Sink vertex $t$
- Find maximum "flow" from source to target
  - ***s*-*t* flow** $f$:$E \rightarrow \mathbf{R}^+$ such that
    - (capacity) $0 \leq f(e) \leq c_e$ for each edge $e$
    - (conservation) for each **internal** vertex $v$ (not $s$ or $t$)

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

# Step 1: Problem formulation

- Input:
  - Directed graph G = (V,E) with edge capacities $c_e$ for each edge $e$
  - Source vertex $s$
  - Sink vertex $t$
- Find maximum "flow" from source to target
  - ***s-t* flow** $f$:$E \rightarrow \mathbf{R}^+$ such that
    - (capacity) $0 \leq f(e) \leq c_e$ for each edge $e$
    - (conservation) for each **internal** vertex $v$ (not $s$ or $t$)

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

  - Maximize **value** of s-t flow: flow out of source (= flow into target)

# Step 1: problem formulation

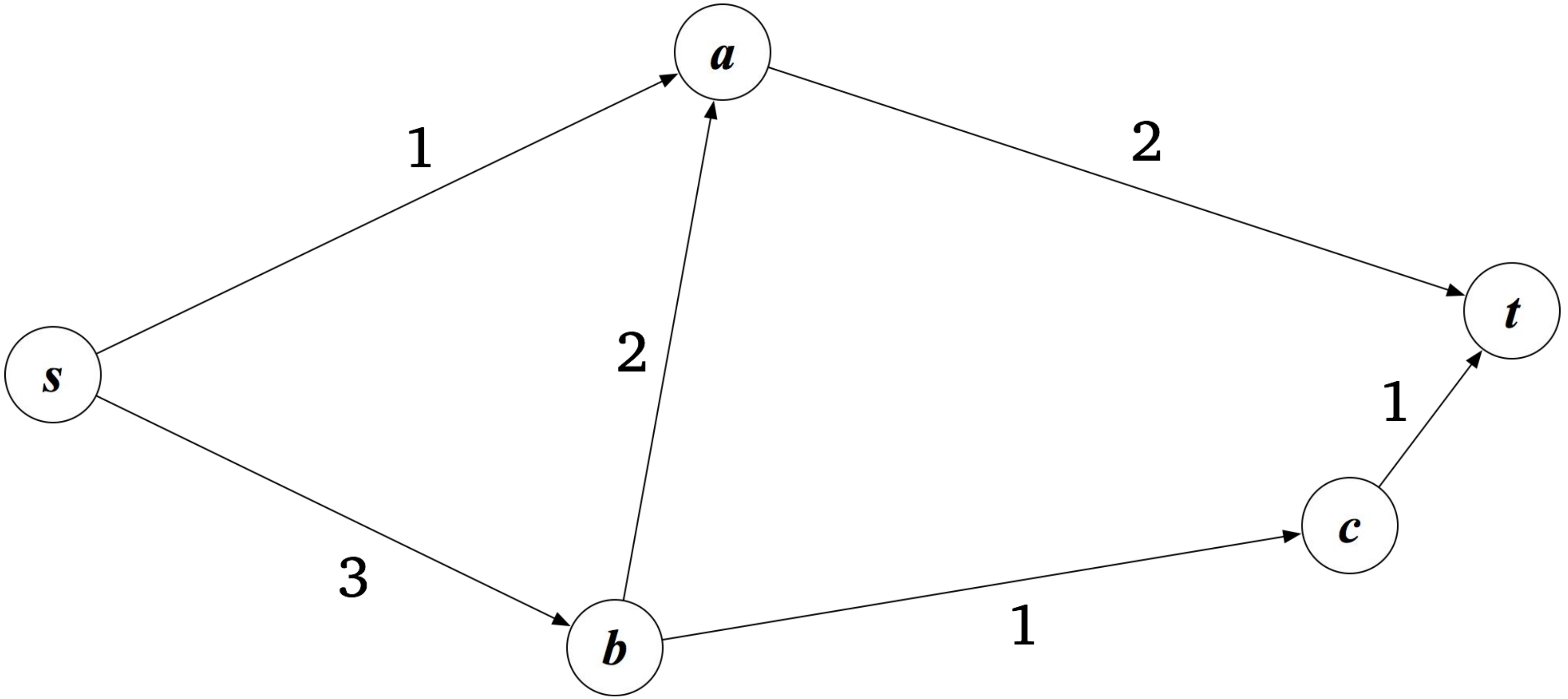
- Input:
  - Directed graph G = (V,E) with edge capacities $c_e$ for each edge $e$
  - Source vertex $s$
  - Sink vertex $t$
- Find maximum "flow" from source to target
  - ***s-t* flow** $f: E \to \mathbf{R}^+$ such that
    - (capacity) $0 \le f(e) \le c_e$ for each edge $e$
    - (conservation) for each **internal** vertex $v$ (not $s$ or $t$)

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

  - Maximize **value** of s-t flow: flow out of source (= flow into target)

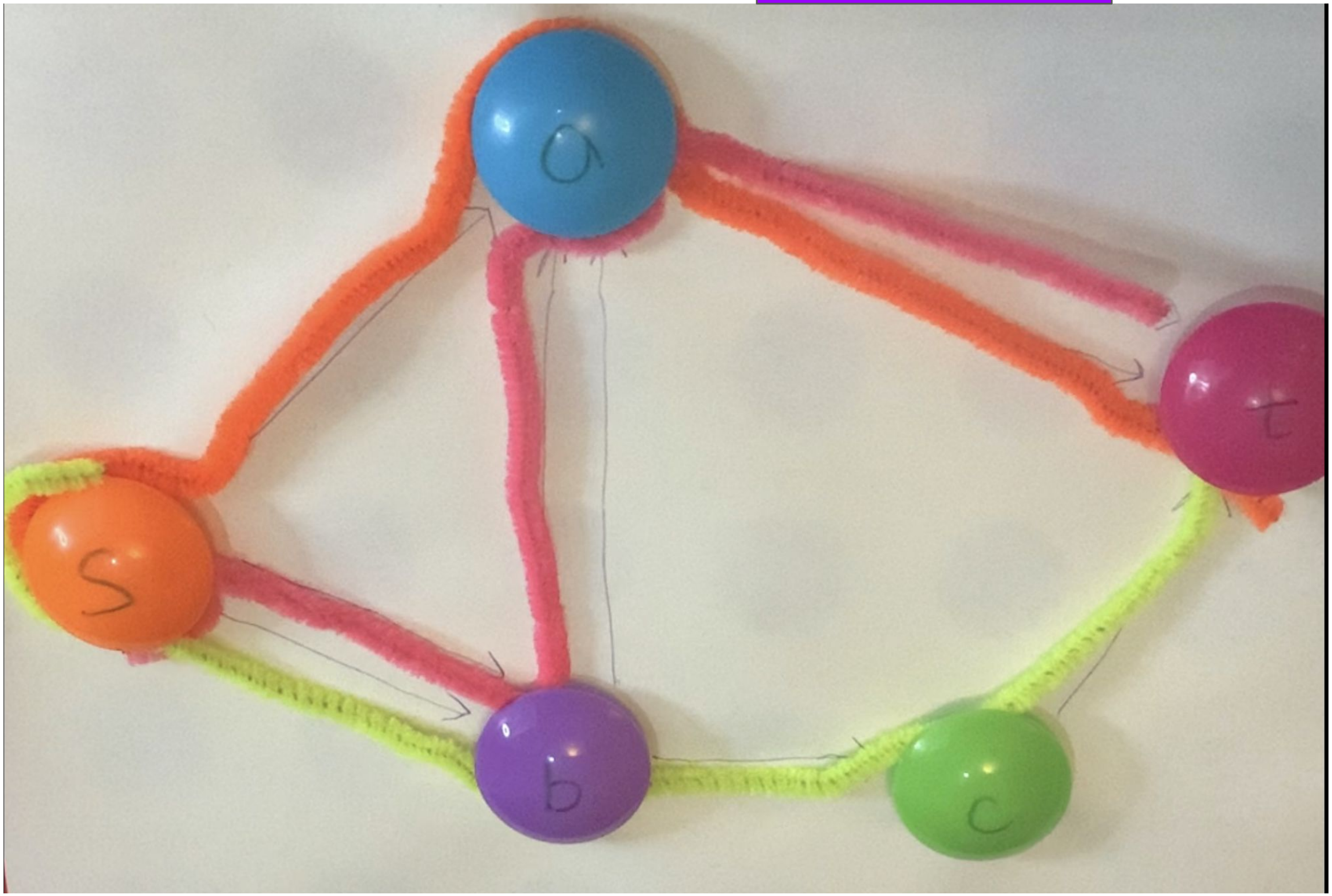
$$\sum_{e \text{ out of } s} f(e)$$

# Example input

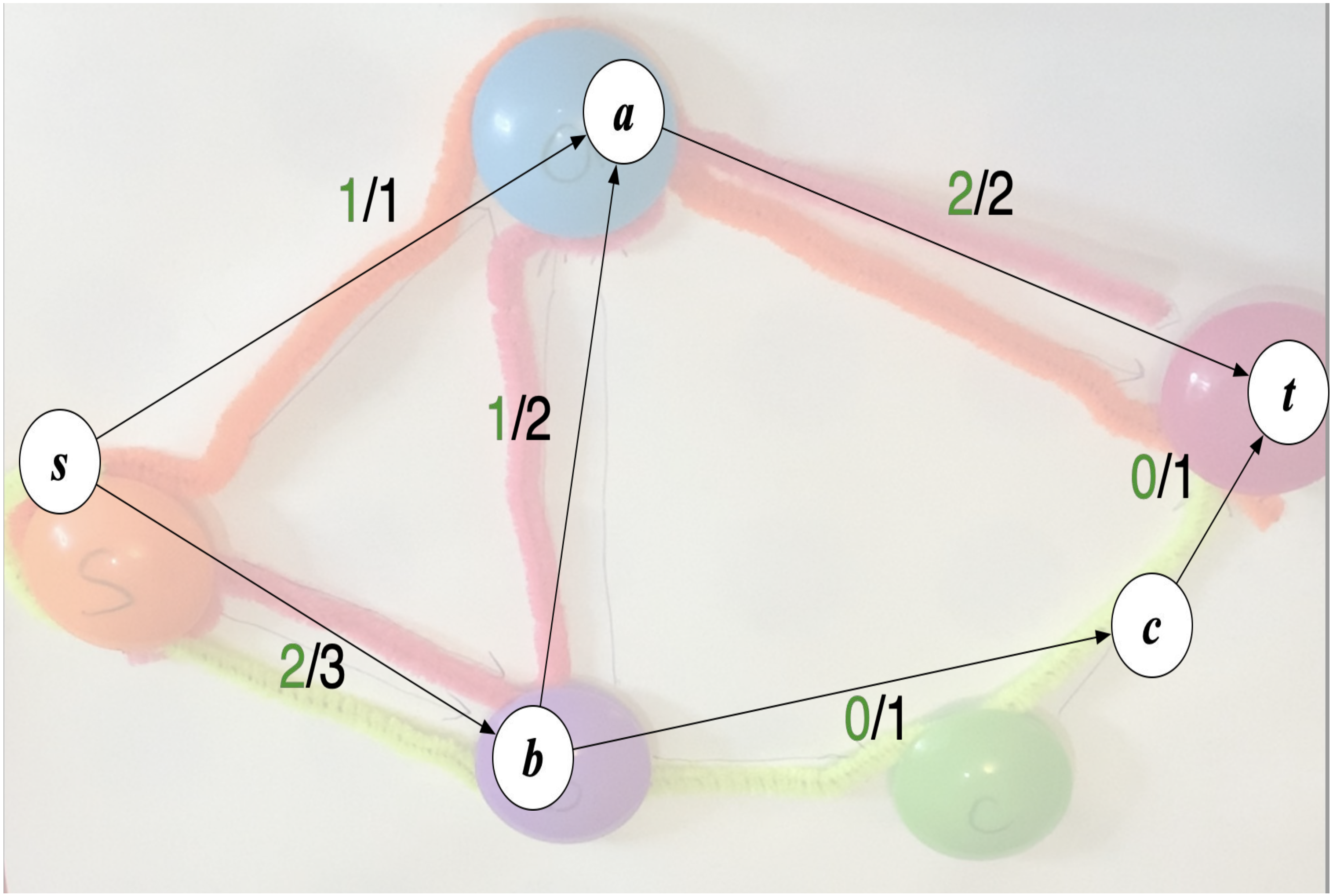


# Did we find a **flow**?

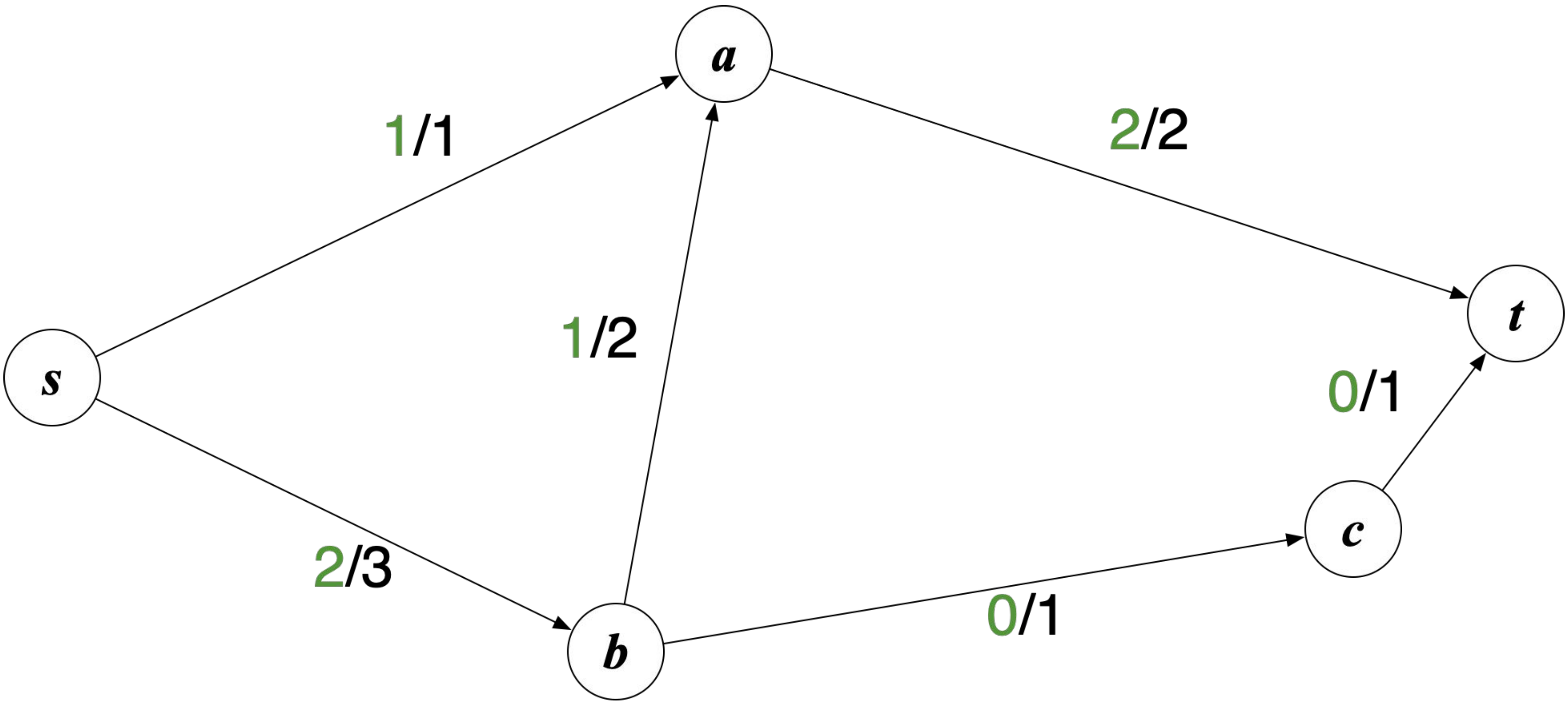
Capacity?
Conservation?



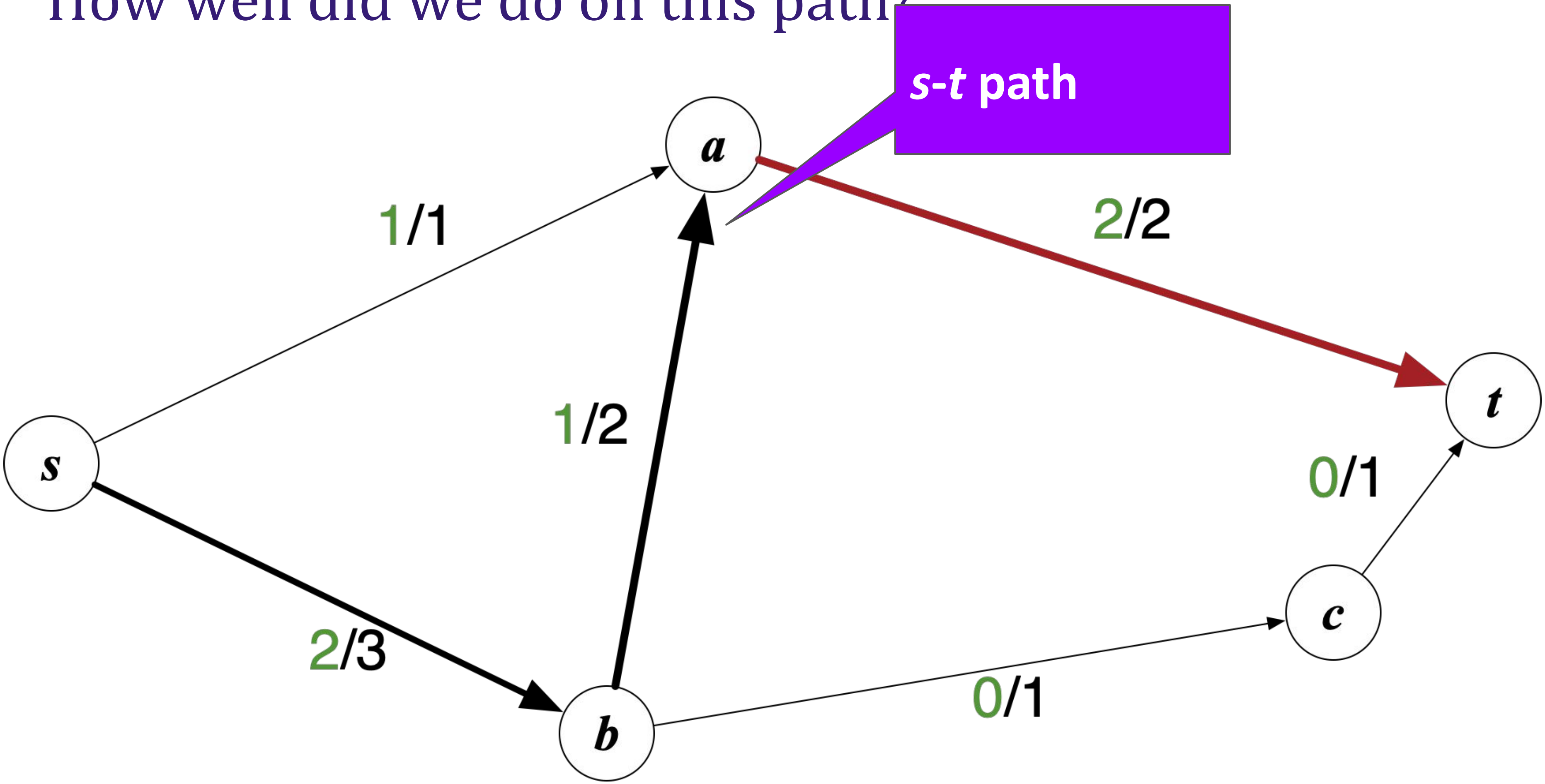
# Flow diagram



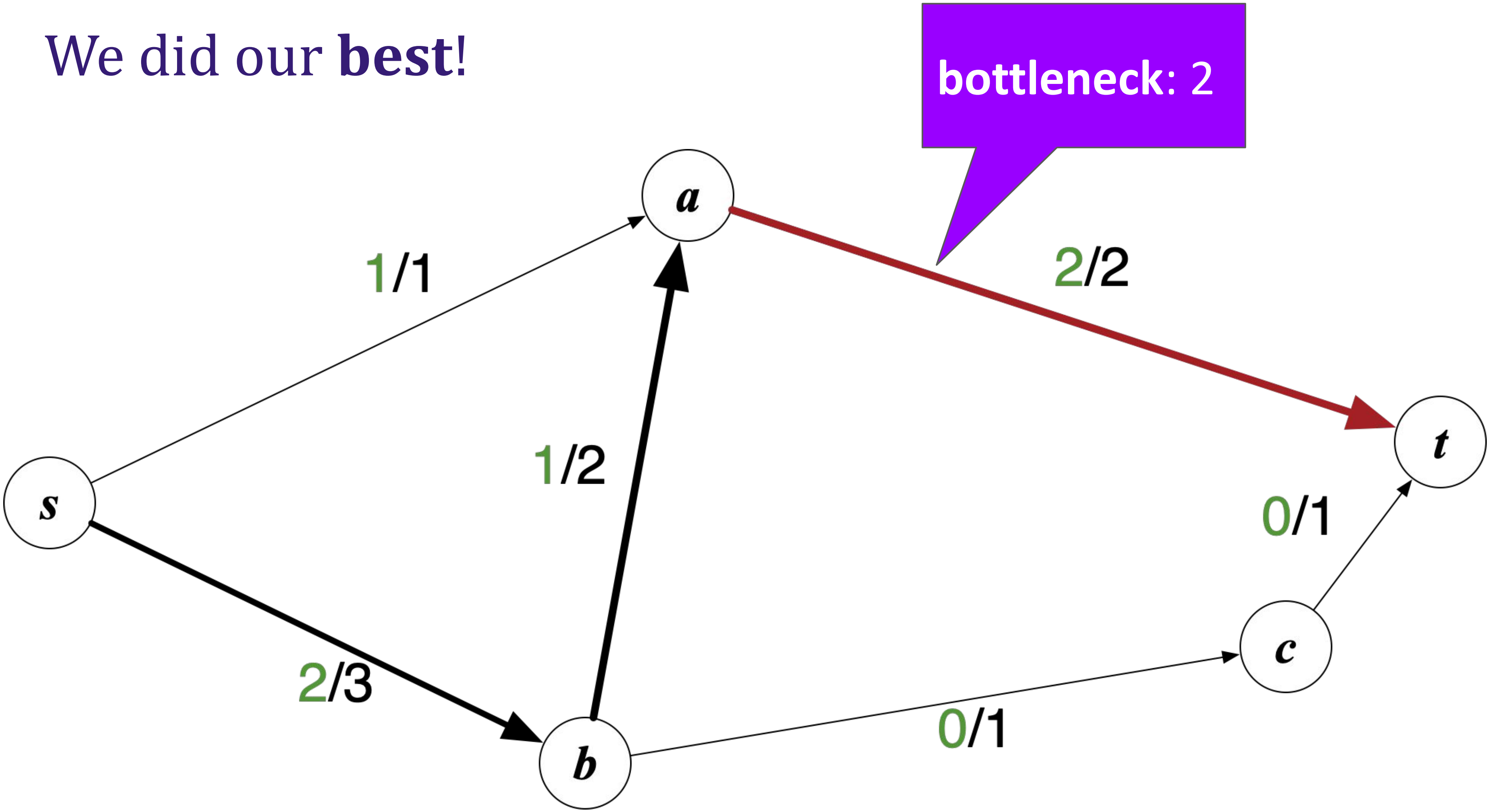
# Flow diagram



# How well did we do on this path?



We did our **best**!

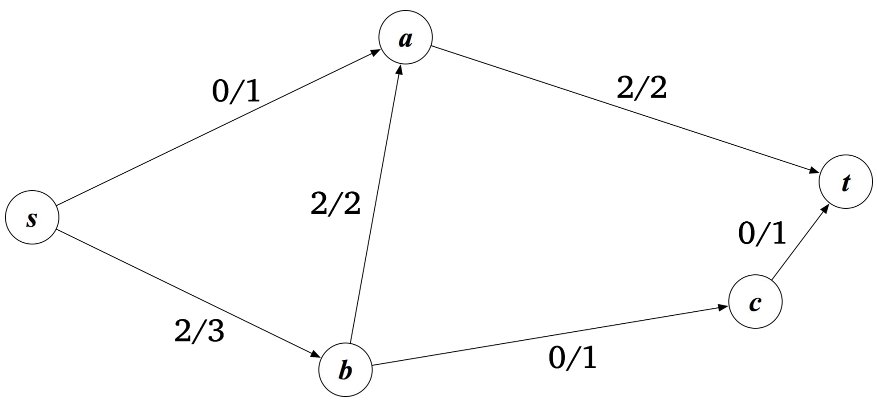


# Can we capture this idea?

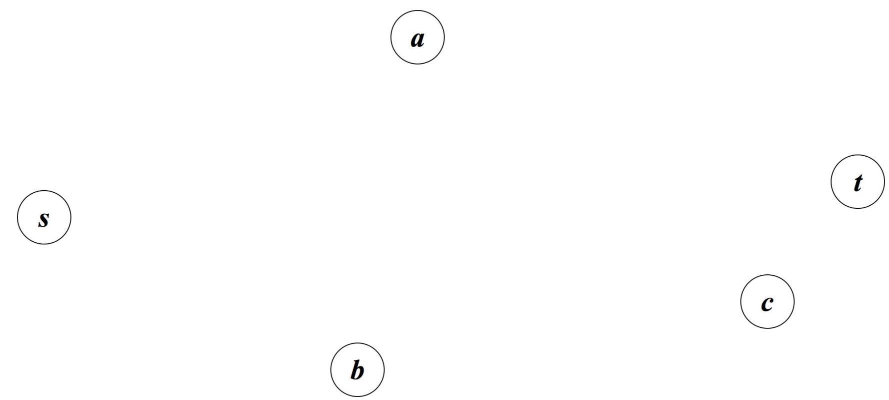
*with a residual graph*

# Residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e - f(e)$: value is unmet capacity $c_e - f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$



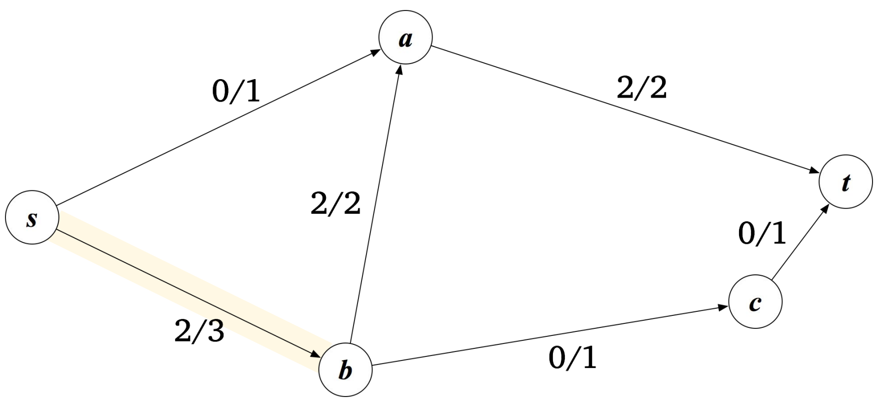

flow



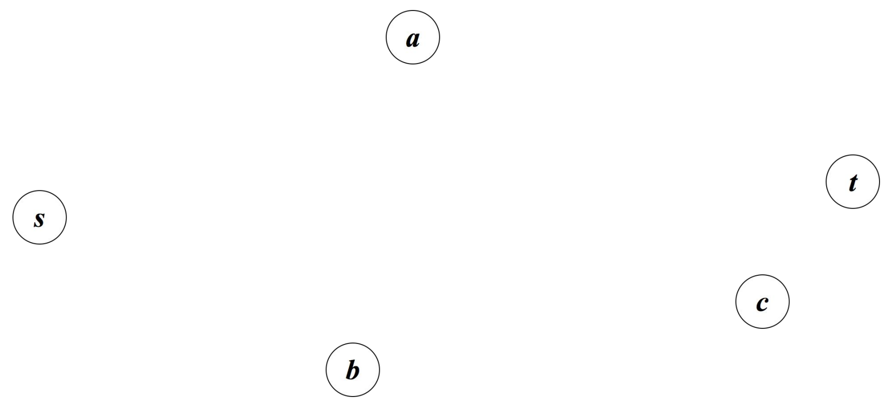

residual graph

# Residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e - f(e)$: value is unmet capacity $c_e - f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$



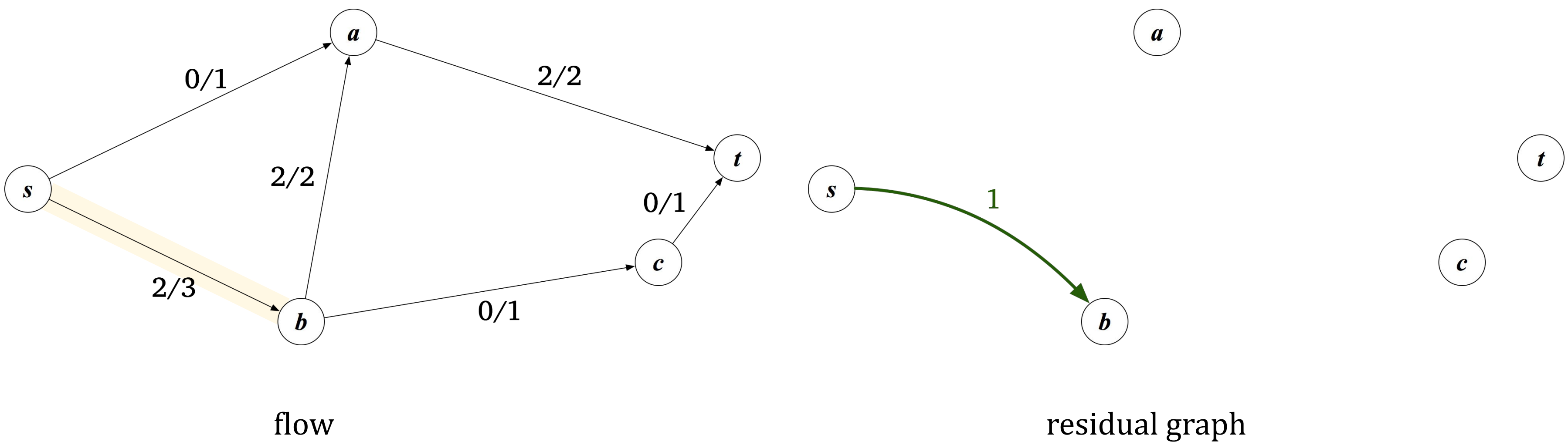

flow




residual graph

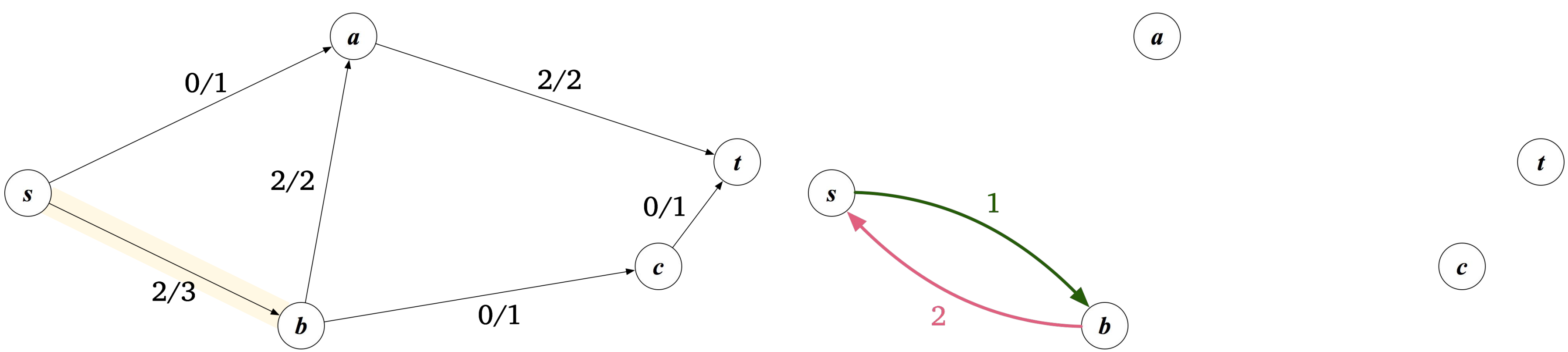
# Residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e - f(e)$: value is unmet capacity $c_e - f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$



# Residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e$ - $f(e)$: value is unmet capacity $c_e$ - $f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$

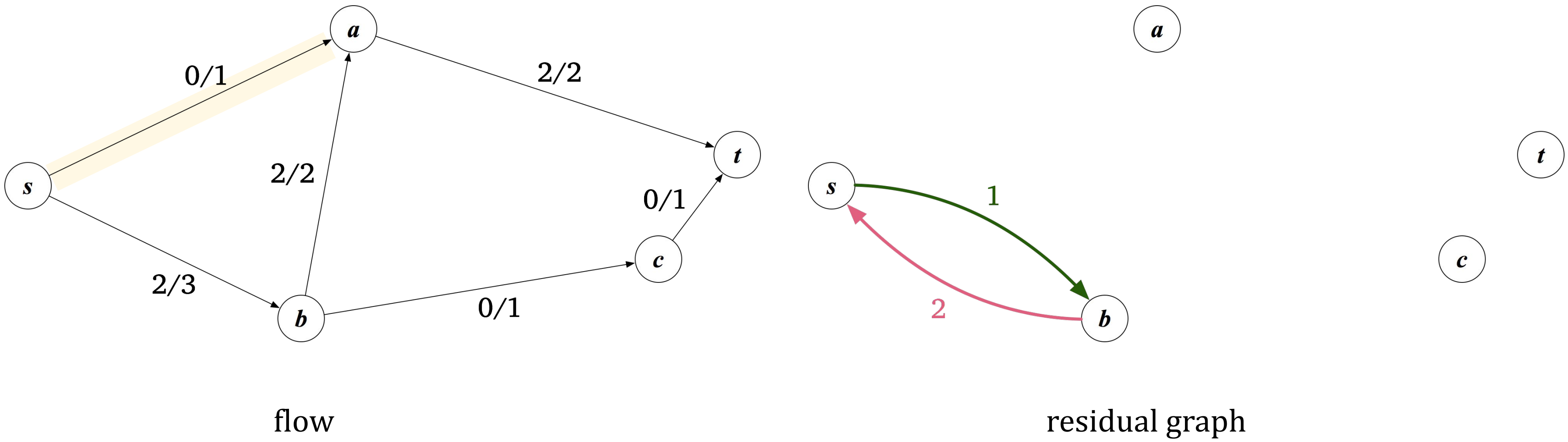



flow

residual graph

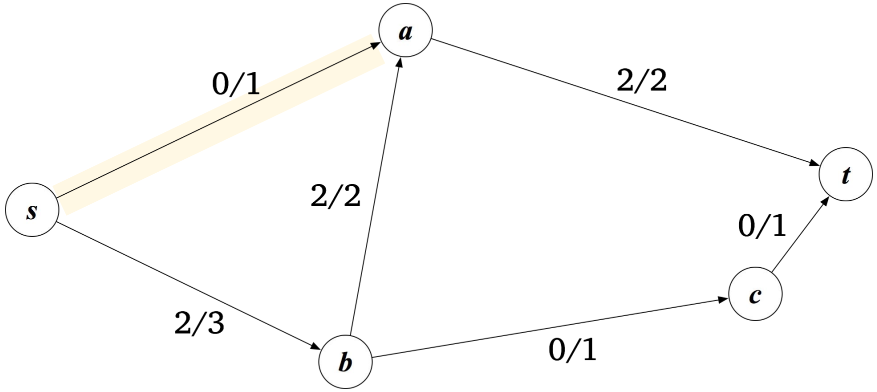
# Residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e - f(e)$: value is unmet capacity $c_e - f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$

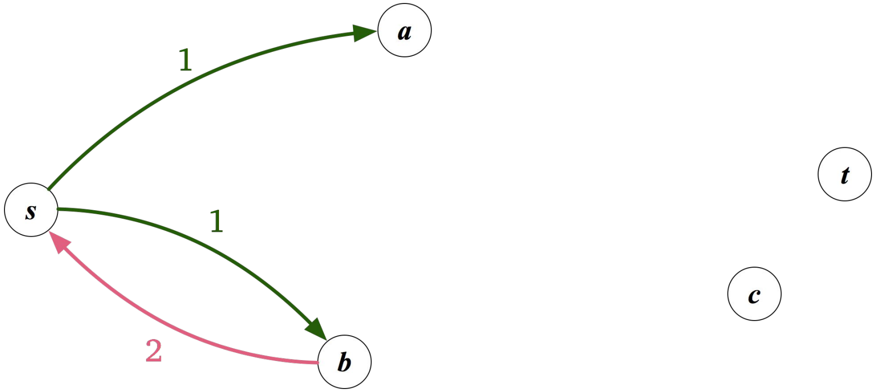


# Residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e$ - $f(e)$: value is unmet capacity $c_e$ - $f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$



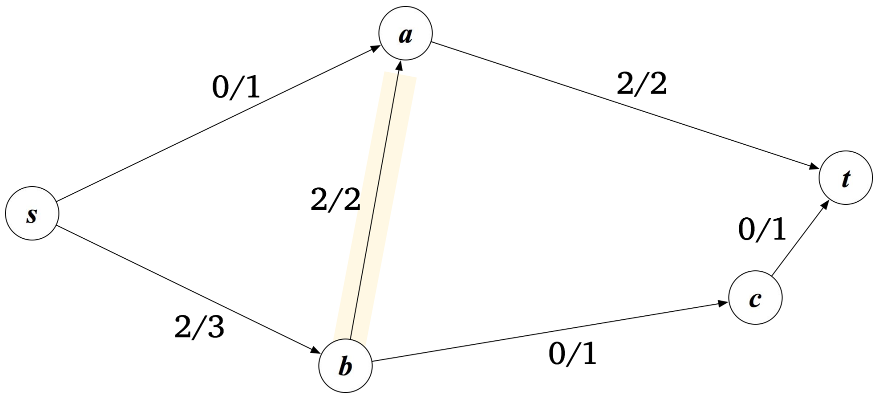

flow



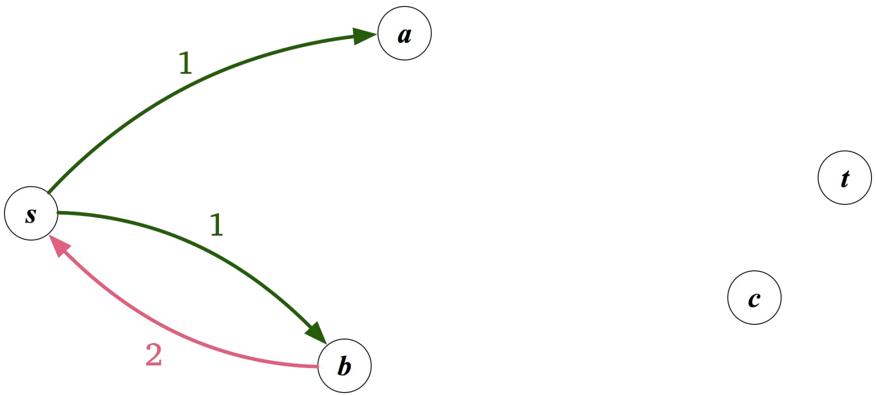

residual graph

# Residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e - f(e)$: value is unmet capacity $c_e - f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$



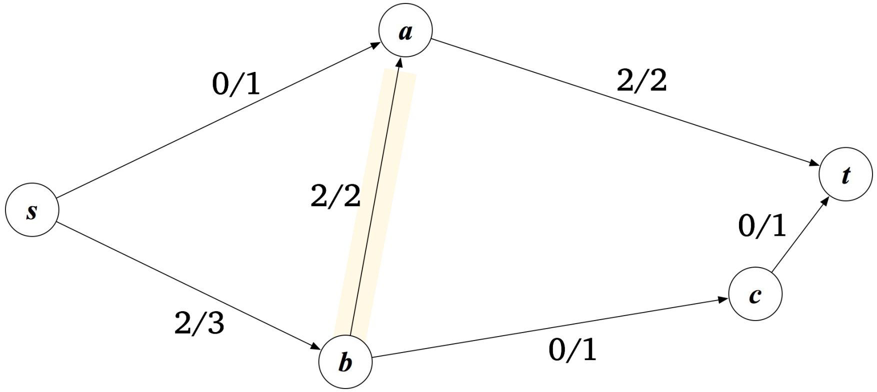

flow



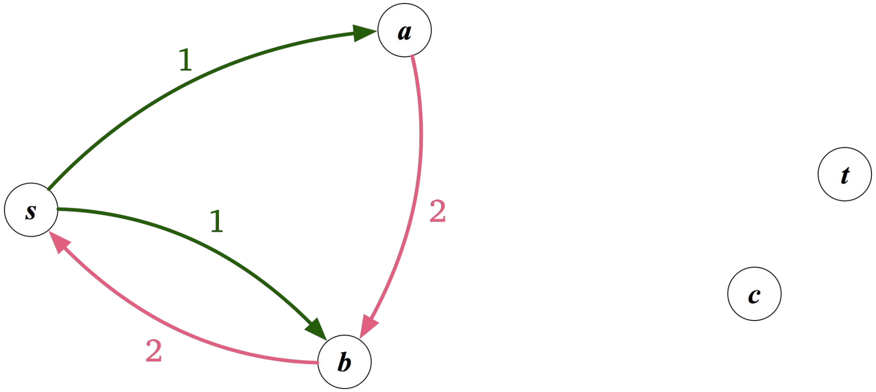

residual graph

# Residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e - f(e)$: value is unmet capacity $c_e - f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$



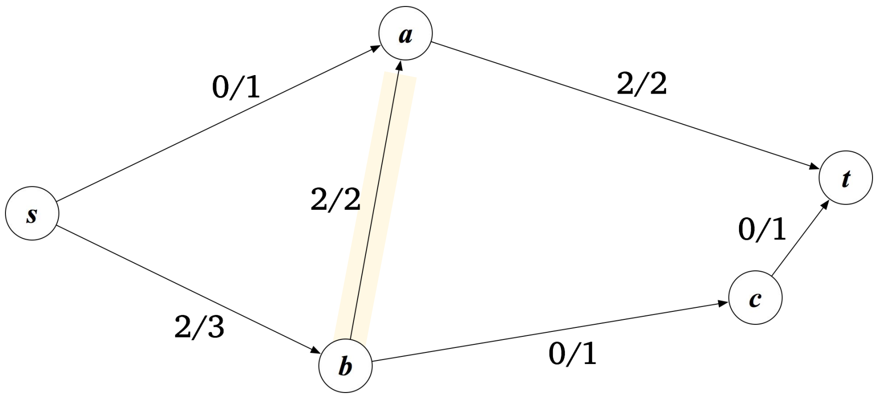

flow



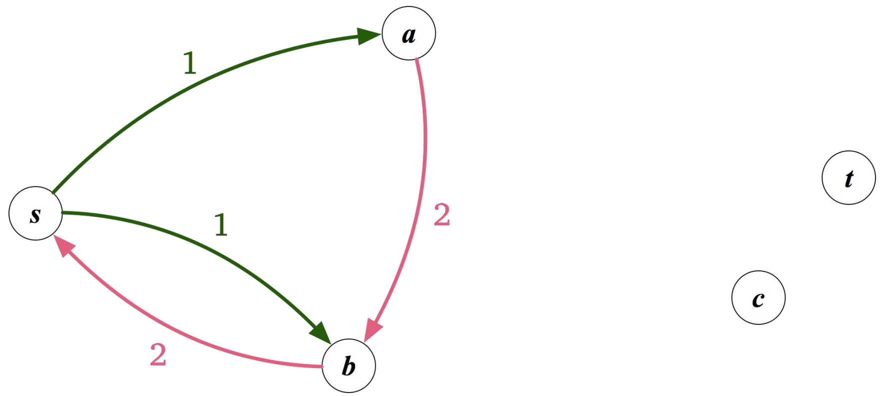

residual graph

# Residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e$ - $f(e)$: value is unmet capacity $c_e$ - $f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$



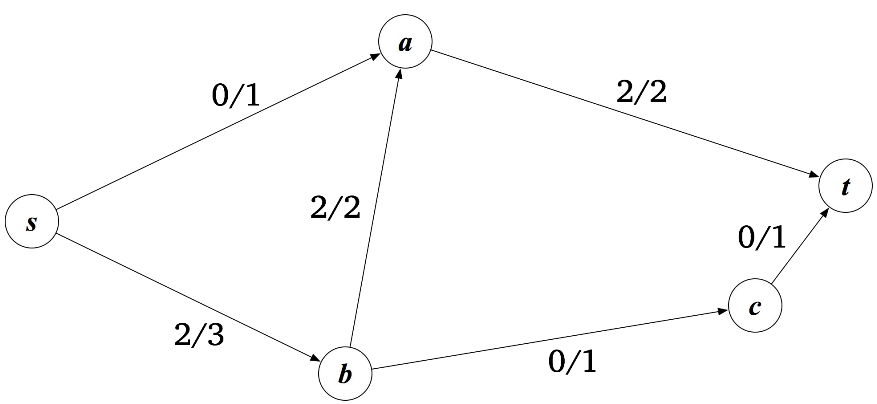

flow



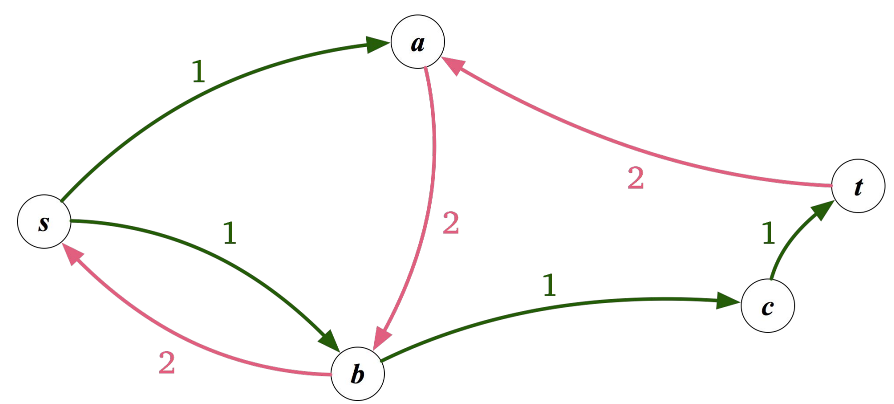

residual graph

# Residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e$ - $f(e)$: value is unmet capacity $c_e$ - $f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$




flow



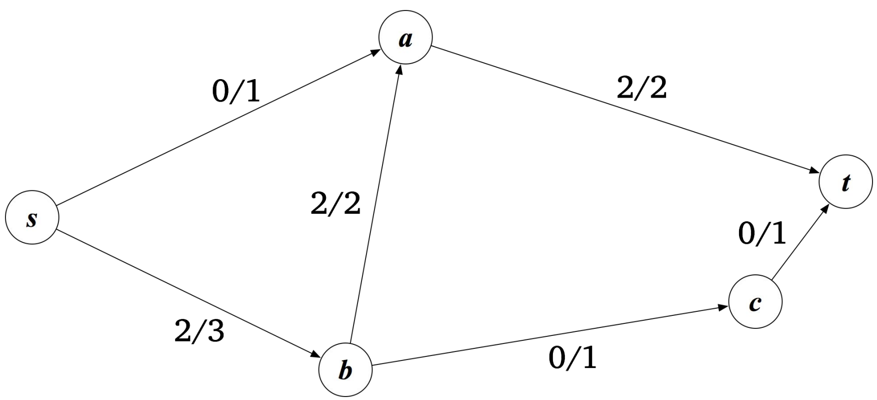

residual graph

# How does this help?

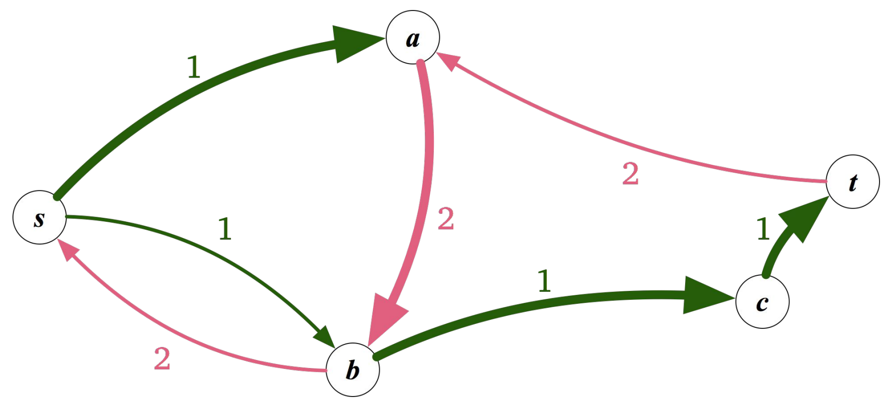
*s*-*t* path in residual graph => **augment** flow

# Residual graph: *s*-*t* path

- Find *s*-*t* path, edge types:
  - Forward edge if $c_e - f(e)$: value is unmet capacity $c_e - f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$



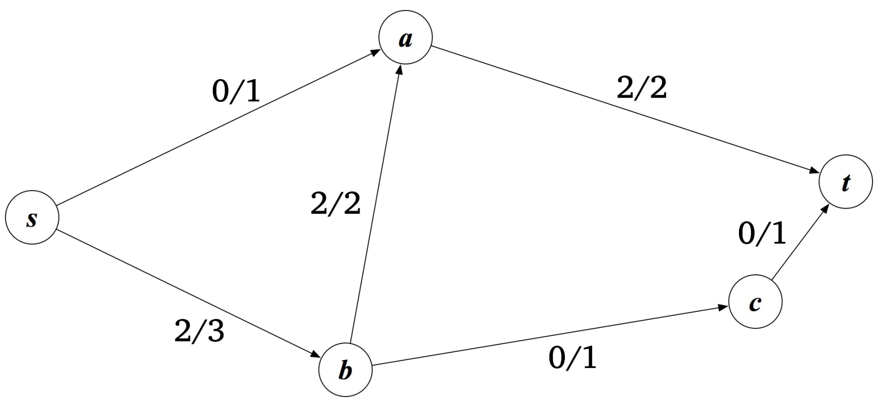

flow



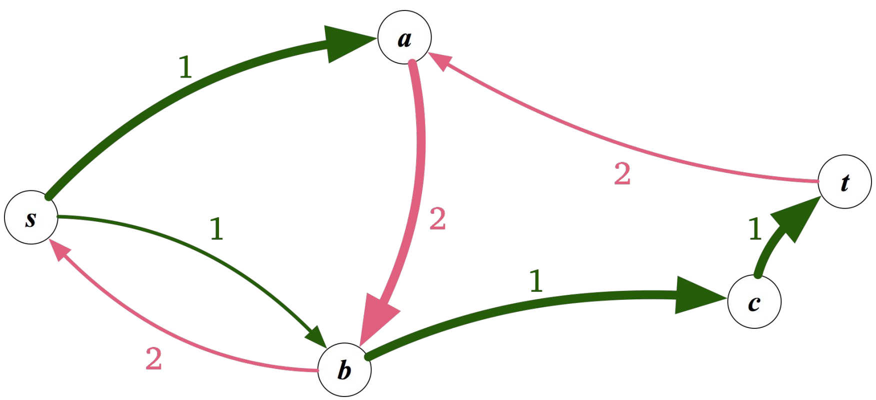

residual graph

# Residual graph: augmenting *s*-*t* path

- Bottleneck: min value
  - Example: 1
- Augmenting path
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow



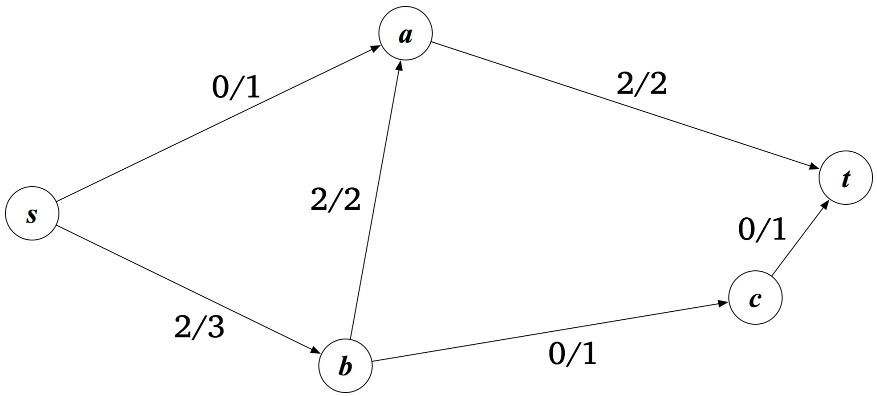

flow



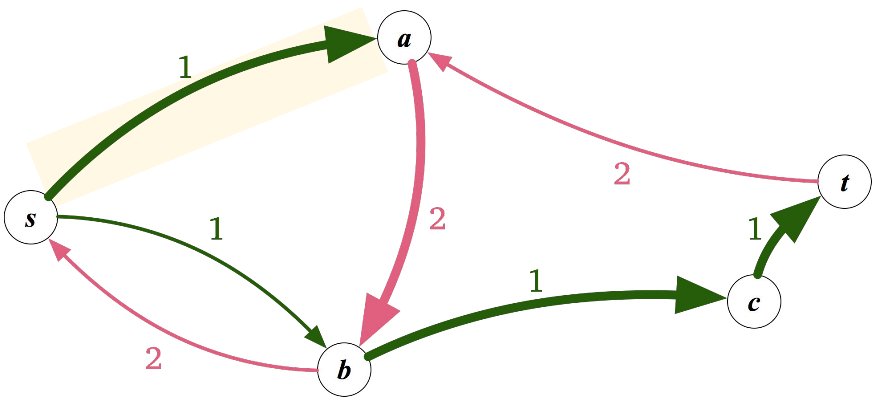

residual graph

# Residual graph: augmenting *s-t* path

- Bottleneck: min value
  - Example: 1
- Augmenting path
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow



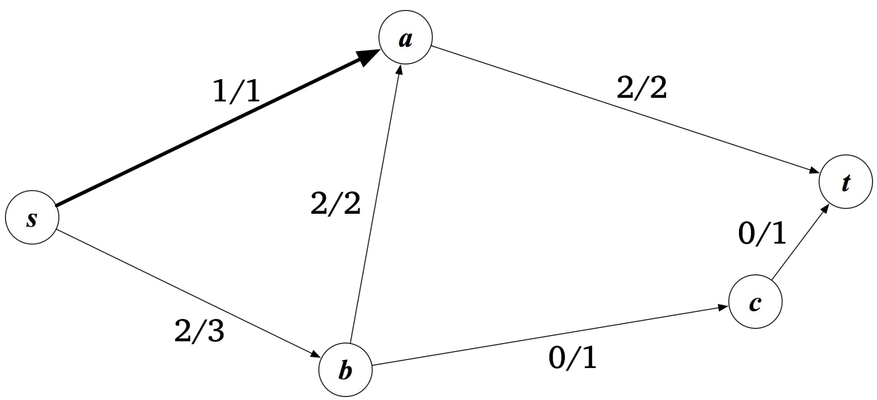

flow



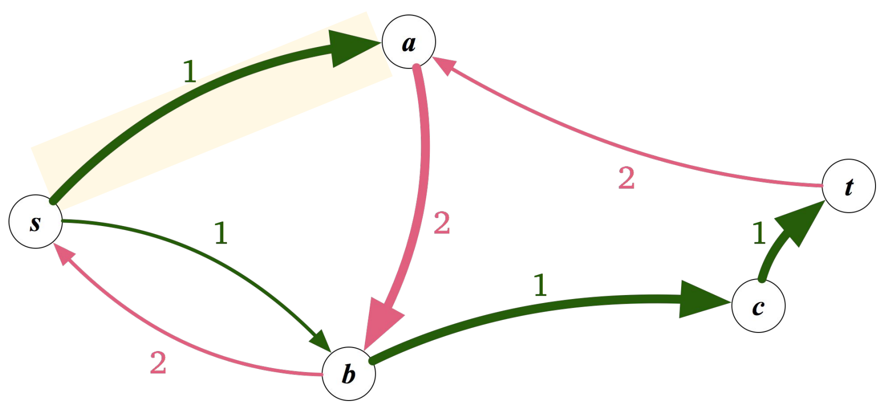

residual graph

# Residual graph: augmenting *s*-*t* path

- Bottleneck: min value
  - Example: 1
- Augmenting path
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow



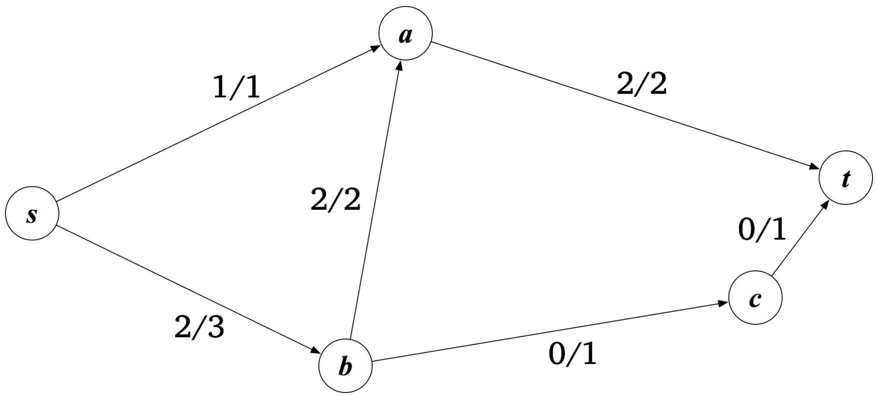

flow



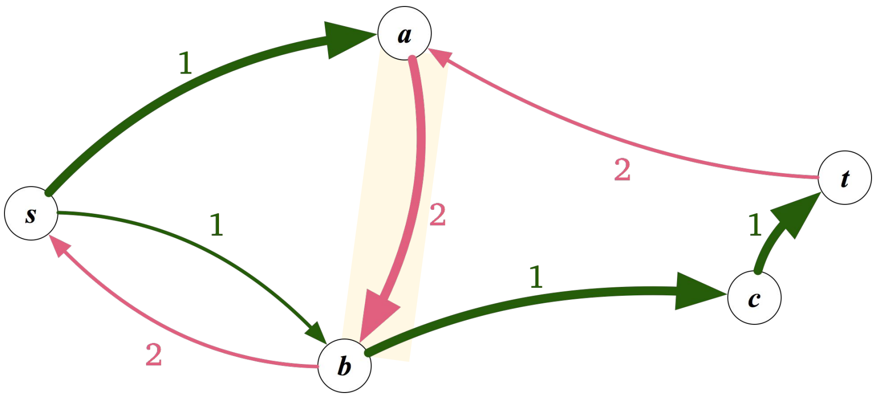

residual graph

# Residual graph: augmenting *s*-*t* path

- Bottleneck: min value
  - Example: 1
- Augmenting path
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow



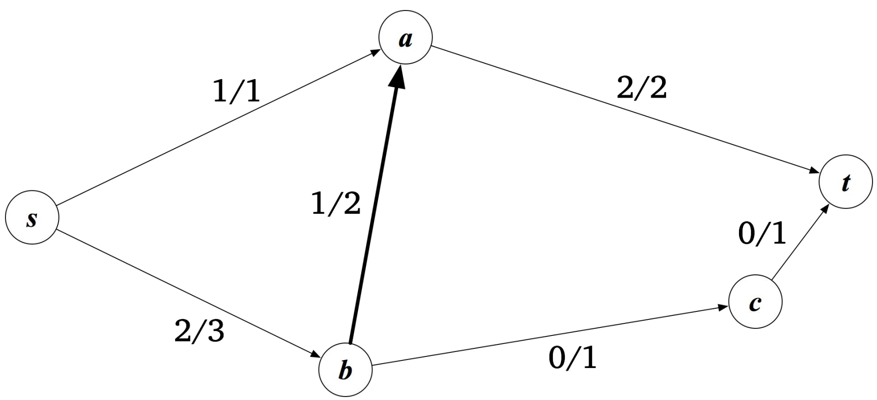

flow



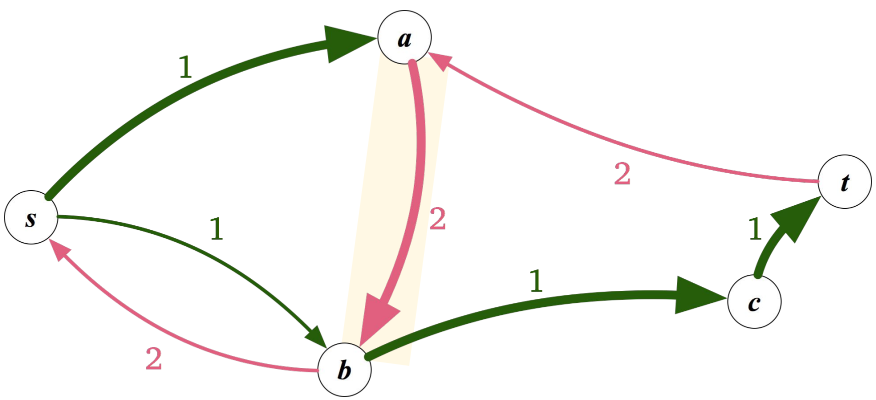

residual graph

# Residual graph: augmenting *s*-*t* path

- Bottleneck: min value
  - Example: 1
- Augmenting path
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow



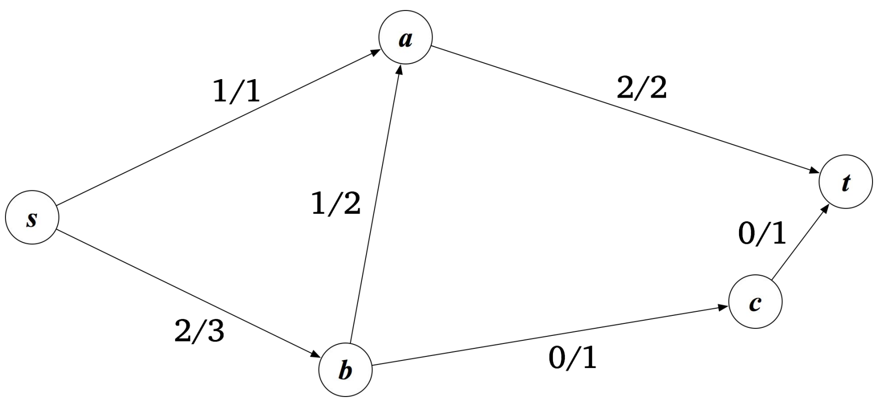

flow



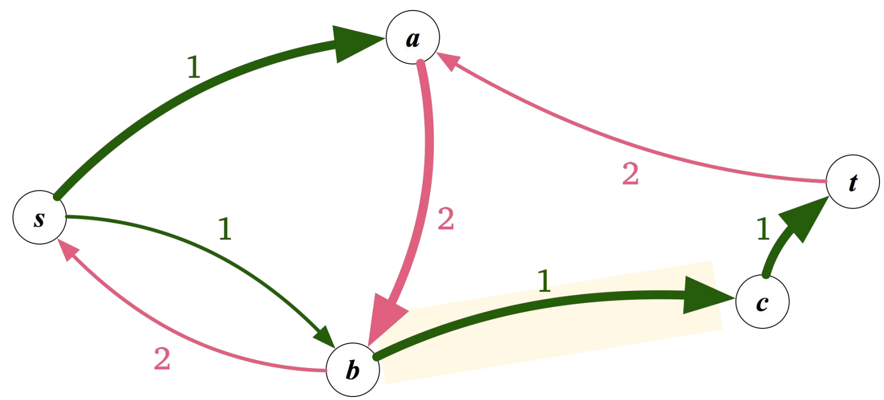

residual graph

# Residual graph: augmenting *s*-*t* path

- Bottleneck: min value
  - Example: 1
- Augmenting path
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow



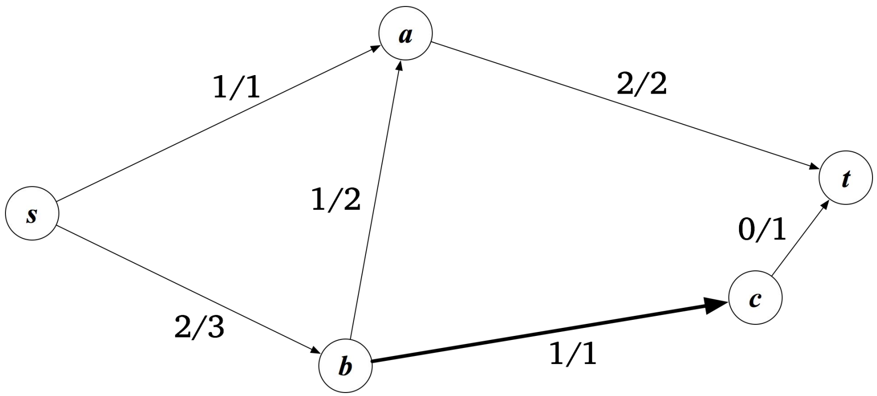

flow



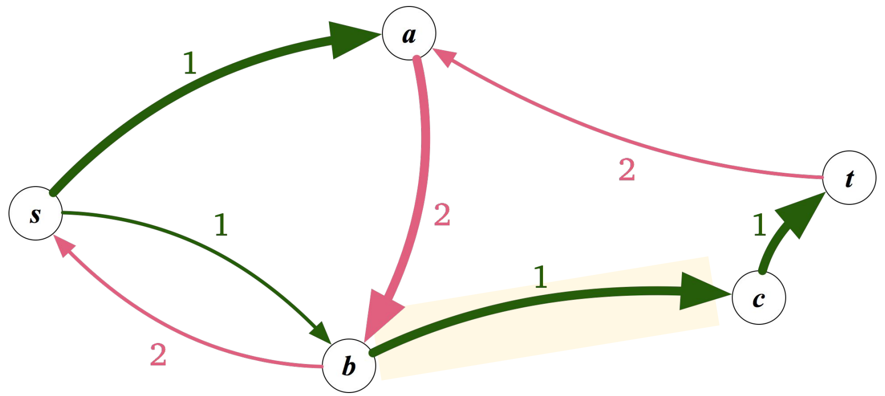

residual graph

# Residual graph: augmenting *s*-*t* path

- Bottleneck: min value
  - Example: 1
- Augmenting path
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow



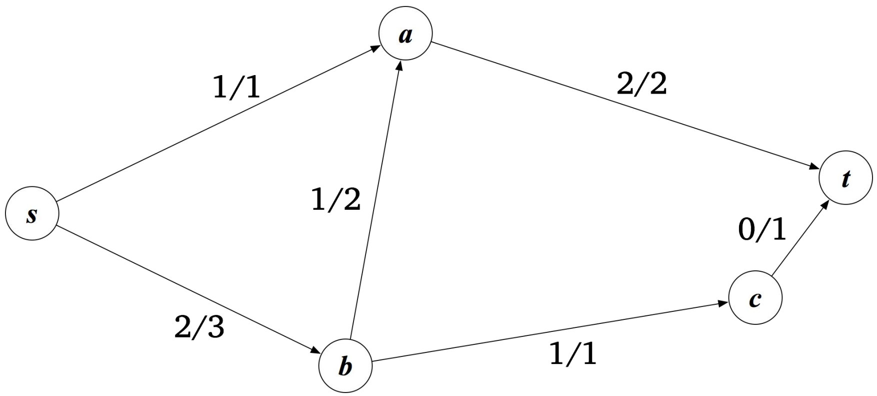

flow



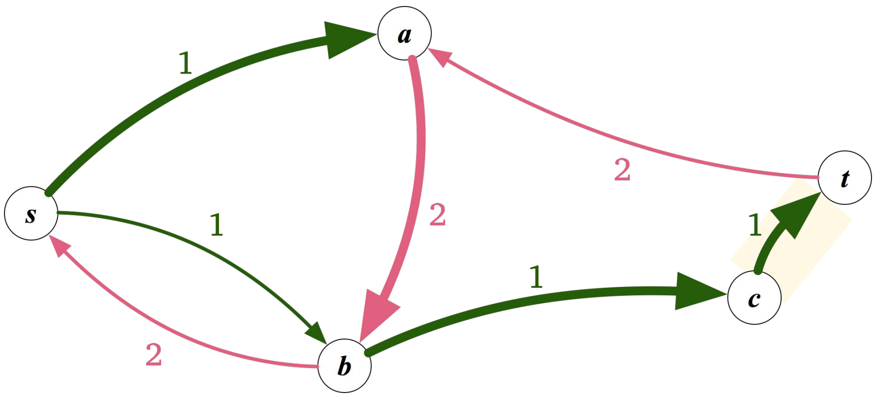

residual graph

# Residual graph: augmenting *s*-*t* path

- Bottleneck: min value
  - Example: 1
- Augmenting path
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow



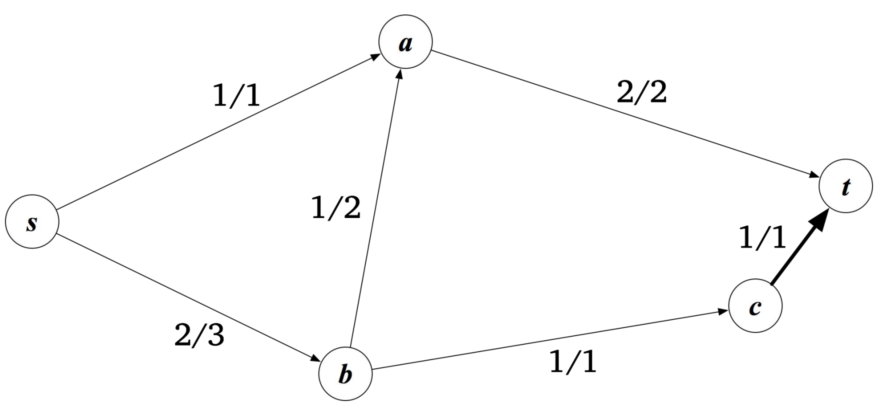

flow



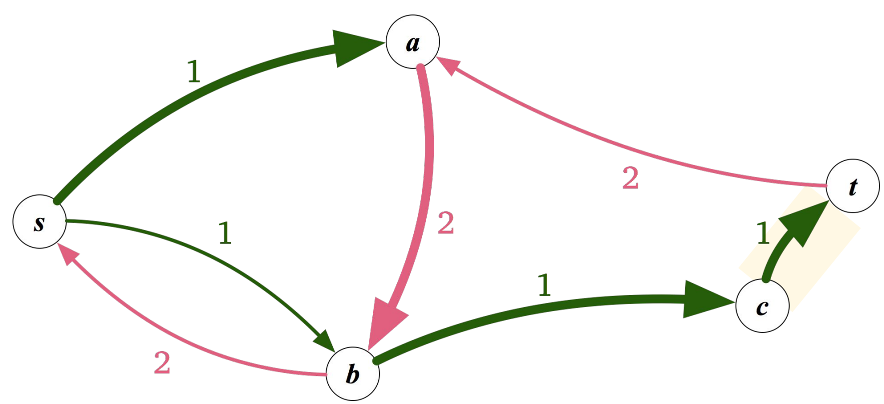

residual graph

# Residual graph: augmenting *s-t* path

- Bottleneck: min value
  - Example: 1
- Augmenting path
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow



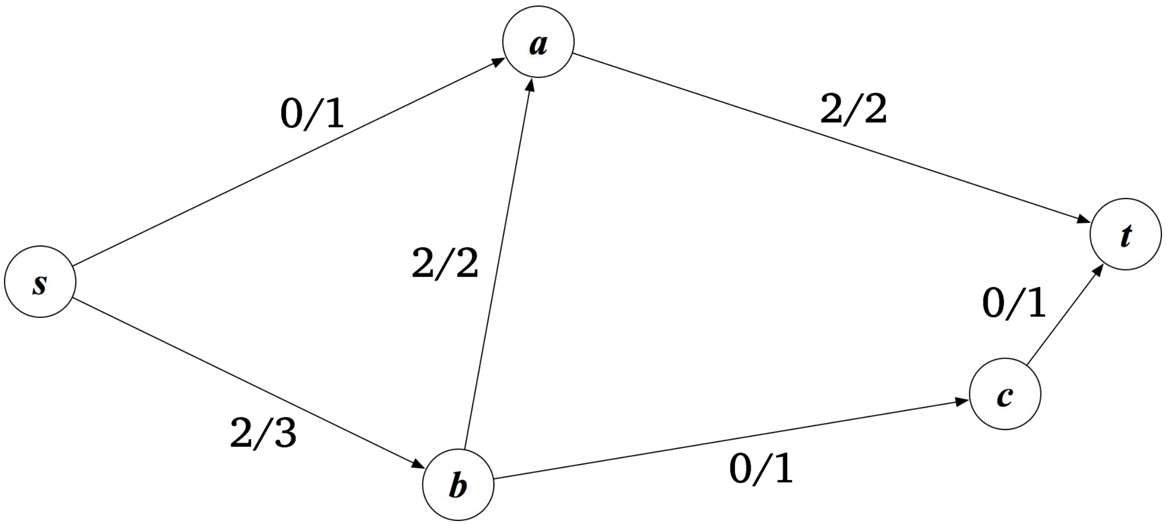

flow



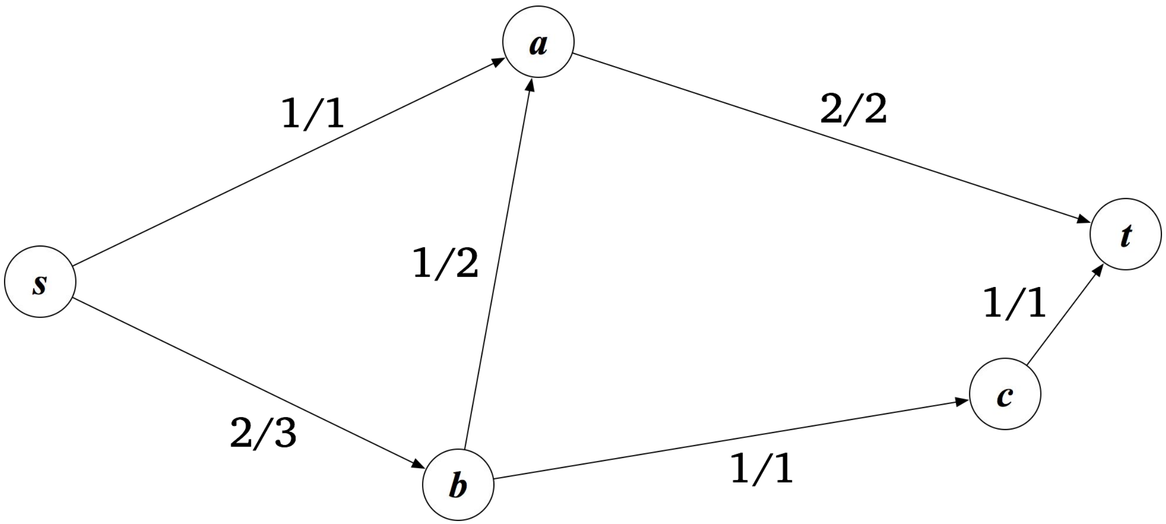

residual graph

# Residual graph: augmenting *s*-*t* path increases flow!

Flow value: 2




Flow value: 3

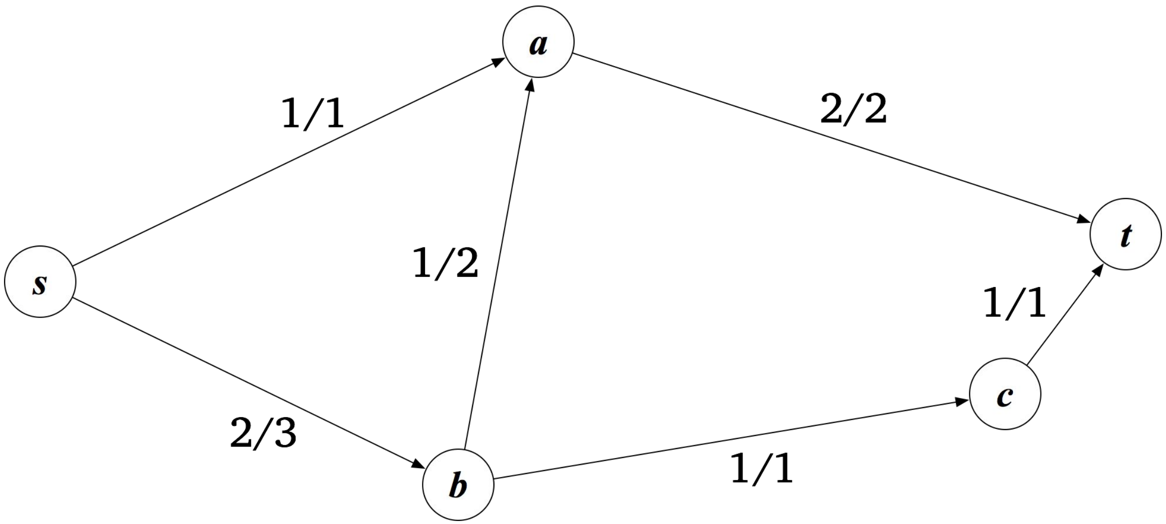


# Let's try that whole thing again!

Build residual graph:

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e - f(e)$: value is unmet capacity $c_e - f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$

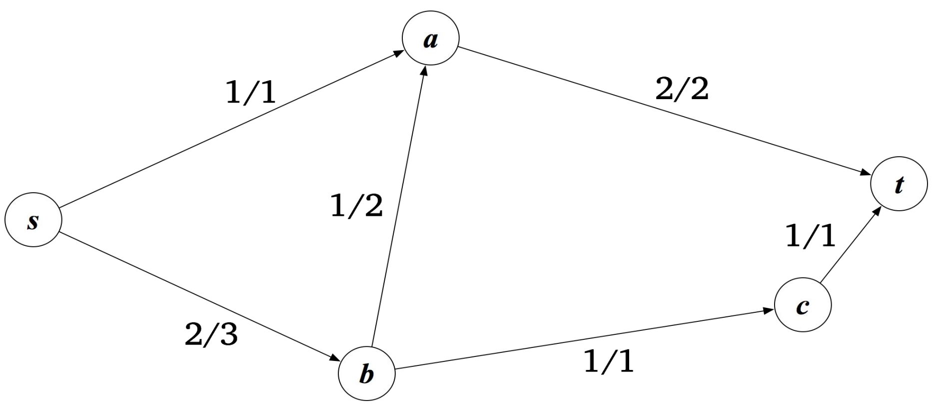
Flow value: 3

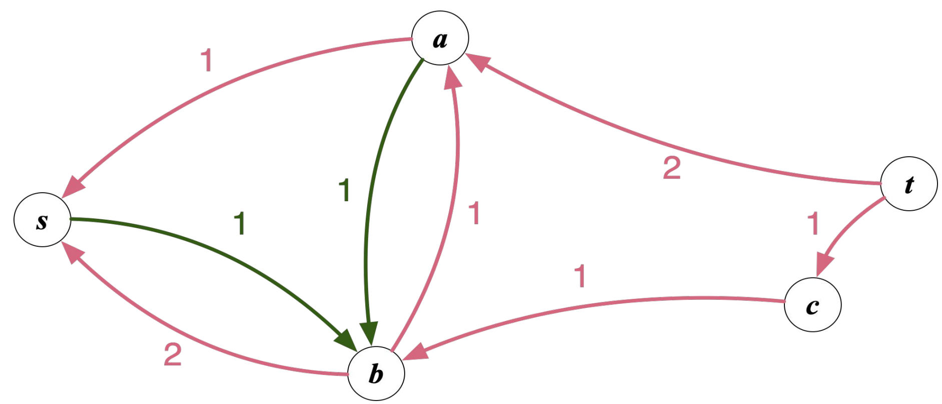


# Let’s try that whole thing again!

Build residual graph:

- For each edge, add up to two edges for *“residual capacity”*:
  - Forward edge if $c_e - f(e)$: value is unmet capacity $c_e - f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$

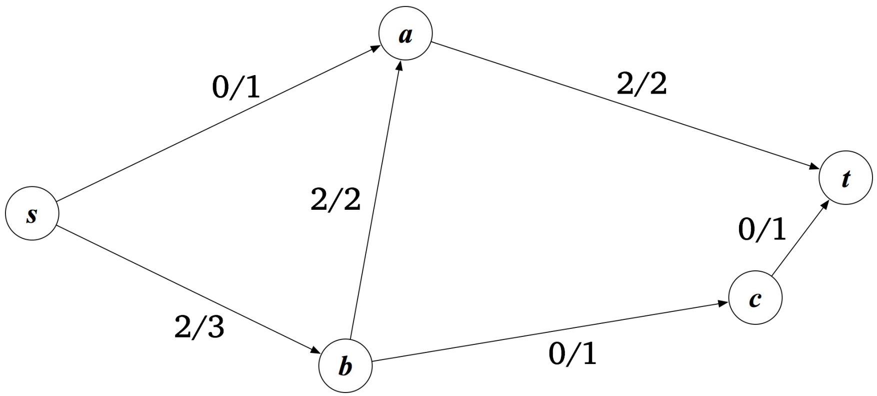


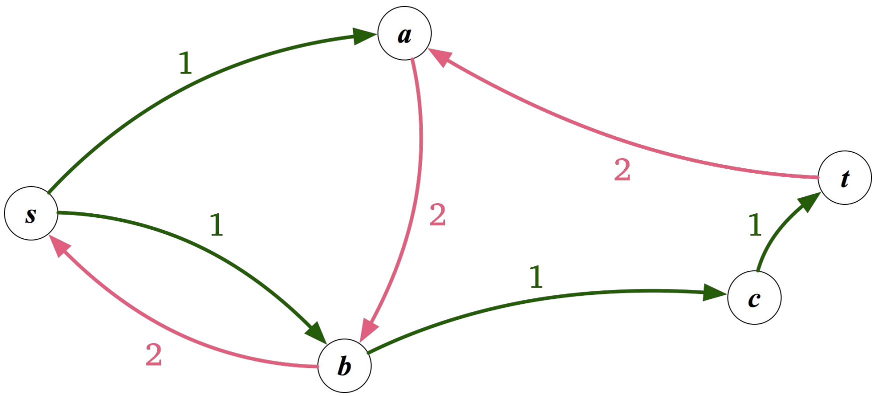

Flow value: 3

# Step 2: approach
# flow $f \longrightarrow$ residual graph

- For each edge, add up to two edges for *"residual capacity"*:
  - Forward edge if $c_e$ - $f(e)$: value is unmet capacity $c_e$ - $f(e)$
  - Backward edge if $f(e) > 0$: value is current flow $f(e)$



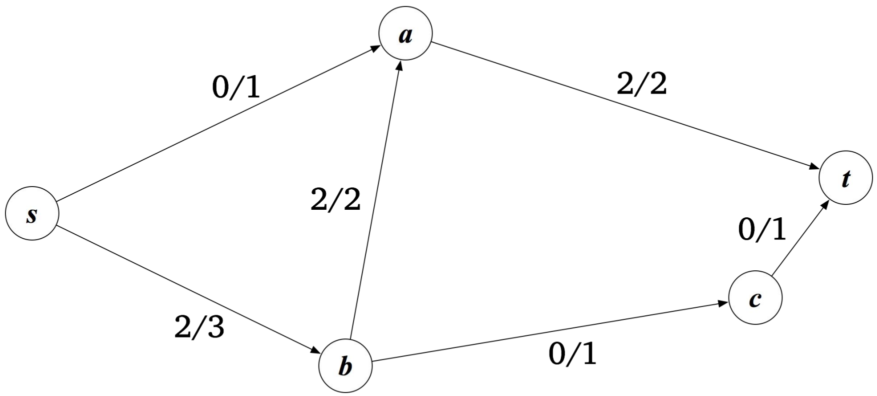

flow



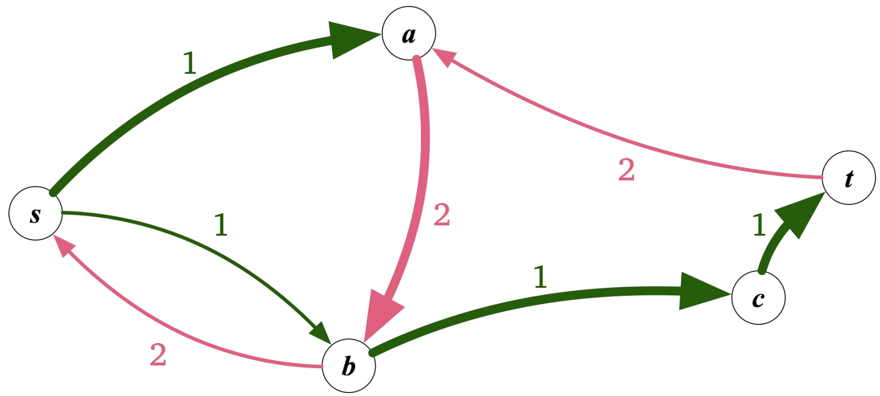

residual graph

# Step 2: approach
# residual graph: find simple *s*-*t* path

- Given a flow *f*, build residual graph
- Find a simple *s*-*t* path, if one exists



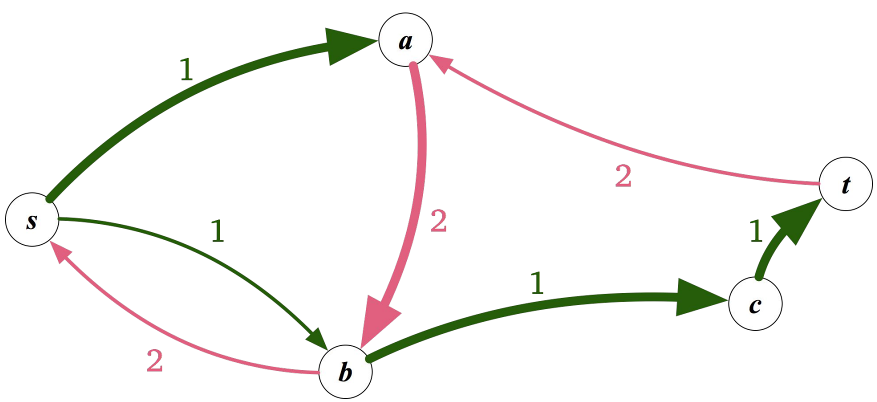

flow



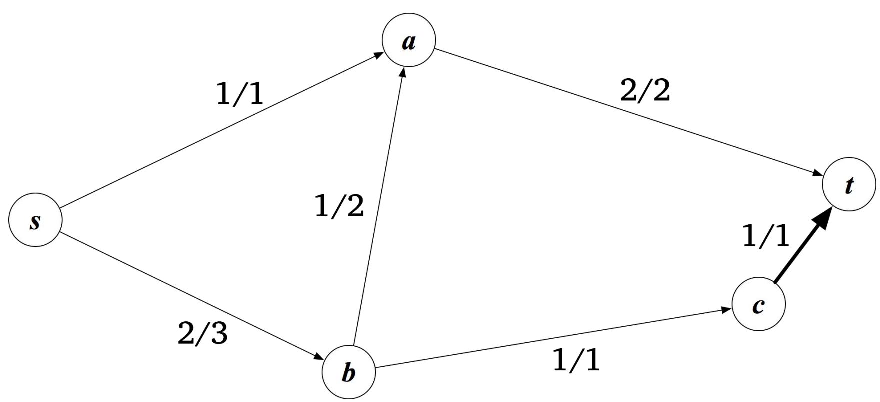

residual graph

# Step 2: approach
# *s*-*t* path → augmented flow

- Determine *bottleneck*: min value
  - Example: 1
- Use path to *augment* flow to new flow *f'*
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow




residual graph



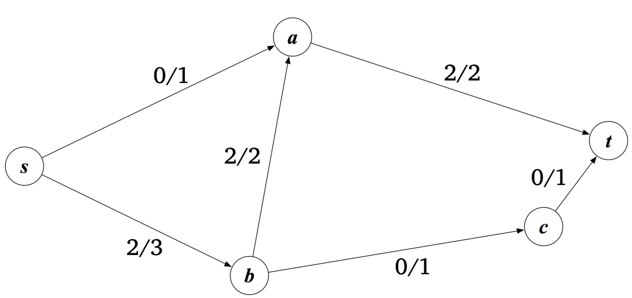

augmented flow

# Step 2: approach

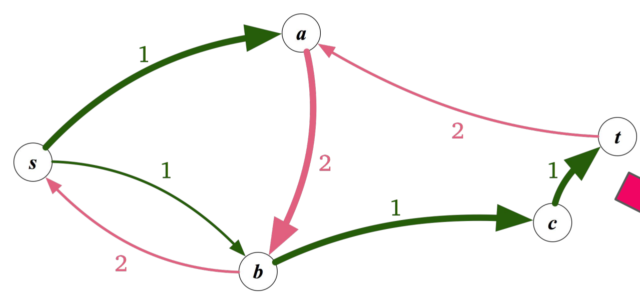
and...
repeat!

- Init flow $f(e) = 0$ for all $e$

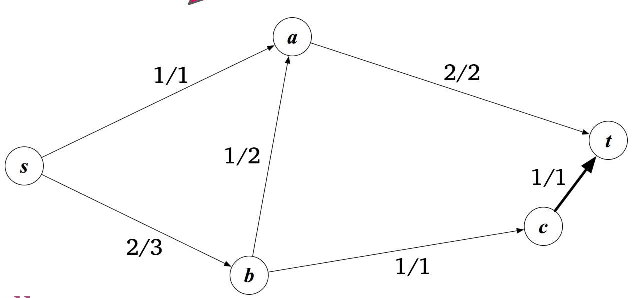
- Given a flow $f$, build residual graph
- Find a simple $s$-$t$ path, if one exists
- Augment using bottleneck to get new flow $f'$




flow




residual graph




augmented flow

# Ford-Fulkerson pseudocode

```
// G is a directed graph, c the capacity labels for edges,
// s and t source and sink (respectively)
findFlow( G = (V,E), c,  s, t ):
        init flow f to assign 0 to each edge
        f' =  augment( G, s, t, f )
        while ( f' != f )
                f = f'
         return f
// build residual graph
buildResidual( G = (V,E), f ):
        R = empty graph on V
        for each e=uv in G:
                // if unmet capacity, forward edge
                if f(e) < c_e:
                        add edge uv  with weight c_e - f(e)
                                        and label "forward" to R
                // if flow, backward edge
                if  f(e) > 0:
                        Add edge vu with weight f(e)
                                        and label "backward" to R
        return R
```

```
// find s-t path, if it exists, and augment flow
// return augmented flow or original f if no path
// assumes G is graph with weighted and labeled edges
augment( graph G, s, t, flow f ):
        R = buildResidual( G, f )
        find s-t path in R [DF or BF]
        bottleneck = min value of all edges on s-t path
        init f'(e) = f(e) for all e
        for each edge uv on s-t path:
                if uv is the "forward" version of e in R:
                        f'(e) += bottleneck
                else // backward edge
                        f'(e) -= bottleneck
        return f
```

# Resources

https://rosulek.github.io/vamonos/demos/max-flow.html

https://bl.ocks.org/estk/9629395

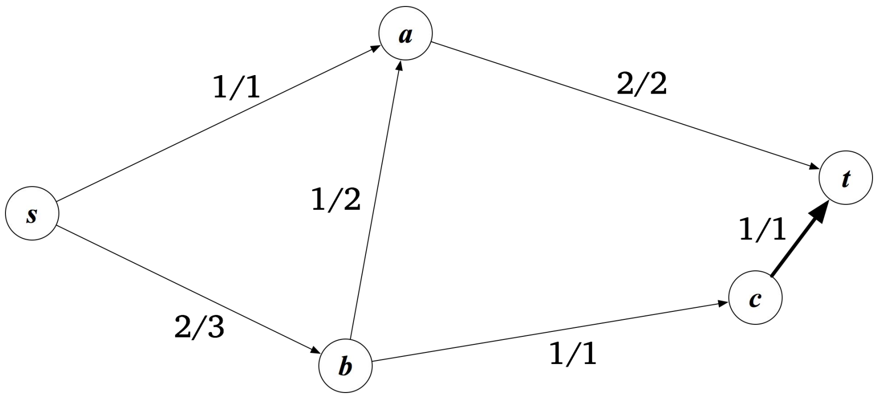
# Step 3: analysis -- correct?

1. At each step, $f'$ is a flow
2. After all iterations, final flow is maximum

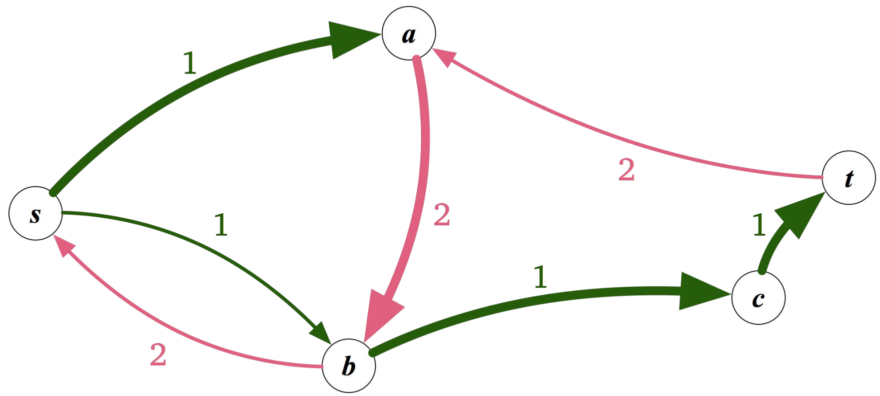
# Step 3: analysis -- correct?
# Residual graph: augmenting *s*-*t* path

We need to prove that it is a flow!

- Determine *bottleneck*: min value
  - Example: 1
- Use path to *augment* flow to new flow *f'*
  - Forward edge: add bottleneck to flow
  - Backward edge: subtract bottleneck from flow



flow



residual graph

# Claim: augmented $f'$ is a flow

Bottleneck: min value on path

Augment: add bottleneck for each forward edge
subtract bottleneck for each backward edge

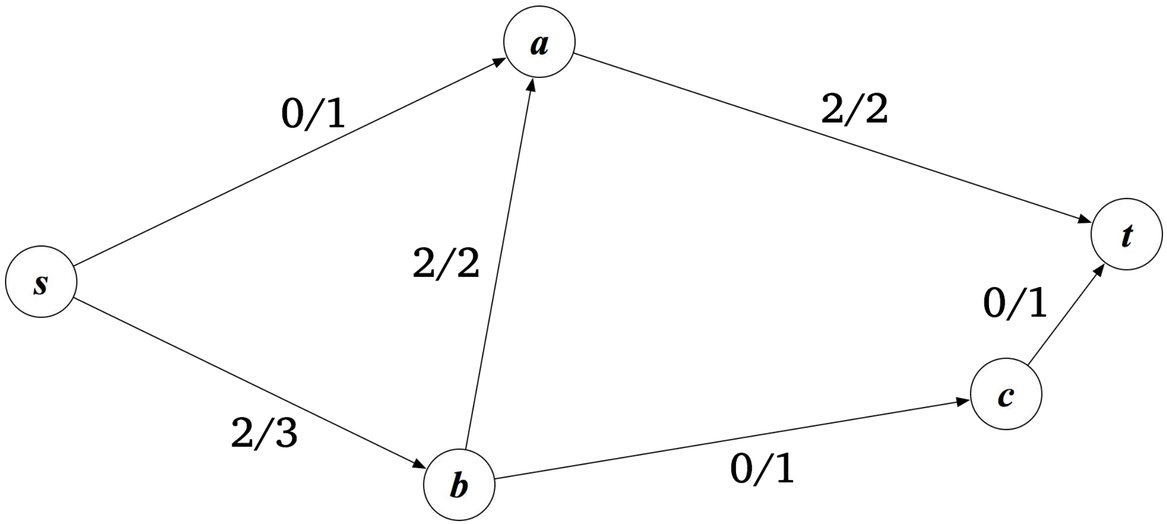
Need to **prove**:

1. (capacity) is $0 \leq f'(e) \leq c_e$ for each edge $e$?
2. (conservation) is flow conserved at each vertex?

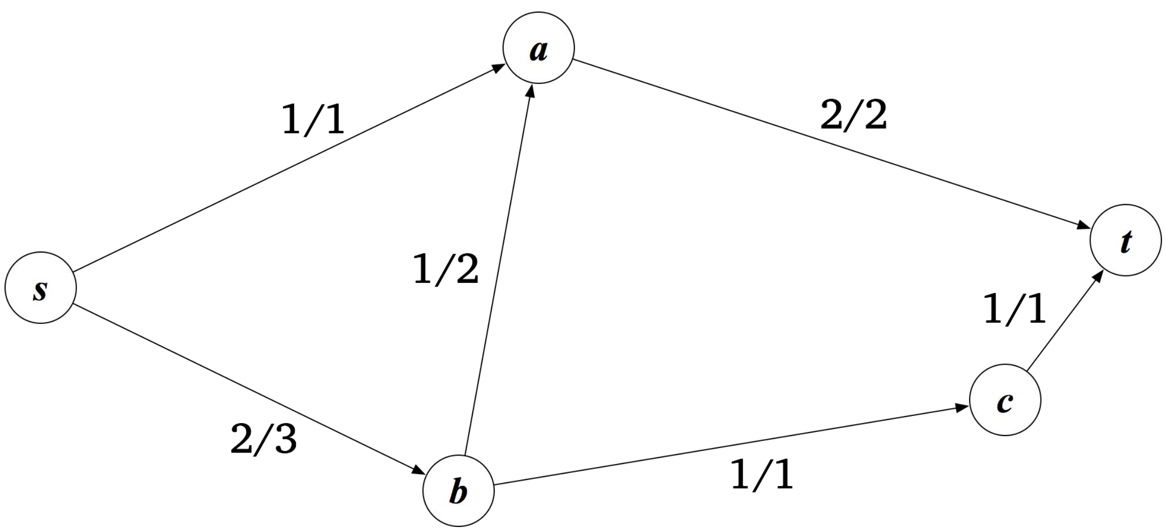
$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

# Augmenting path increases flow by bottleneck value!

Flow value: 2




Flow value: 3



# Step 3: analysis -- correct?

1. At each step, $f'$ is a flow
2. After all iterations, final flow is maximum

# Step 3: analysis -- correct?

1. At each step, *f'* is a flow
2. After all iterations, final flow is ~~maximum~~ maximal

# Step 3: analysis -- correct?

1. At each step, *f'* is a flow
2. After all iterations, final flow is ~~maximum~~ maximal
   a. Stop when we can't increase anymore (greedy algorithm)

# How do we prove it is **maximum**?

(maximum value over all possible flows)

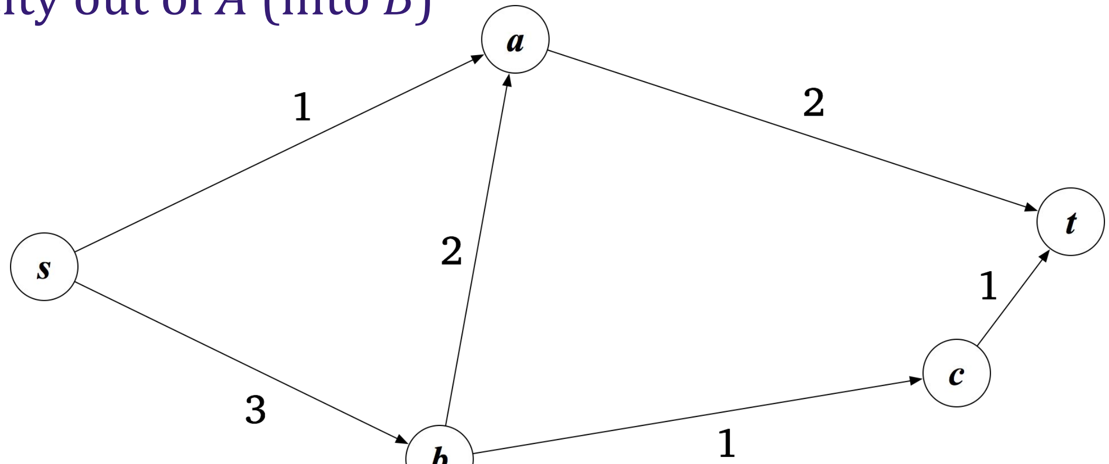
# Key proof ideas (K&T Ch. 7.2)

- Proof structure
  - Any flow is *lower bound* of max flow
  - Algorithm stops when meet *upper bound*
  - **Best we can do!**

# Key proof ideas (K&T Ch. 7.2)

## Terms

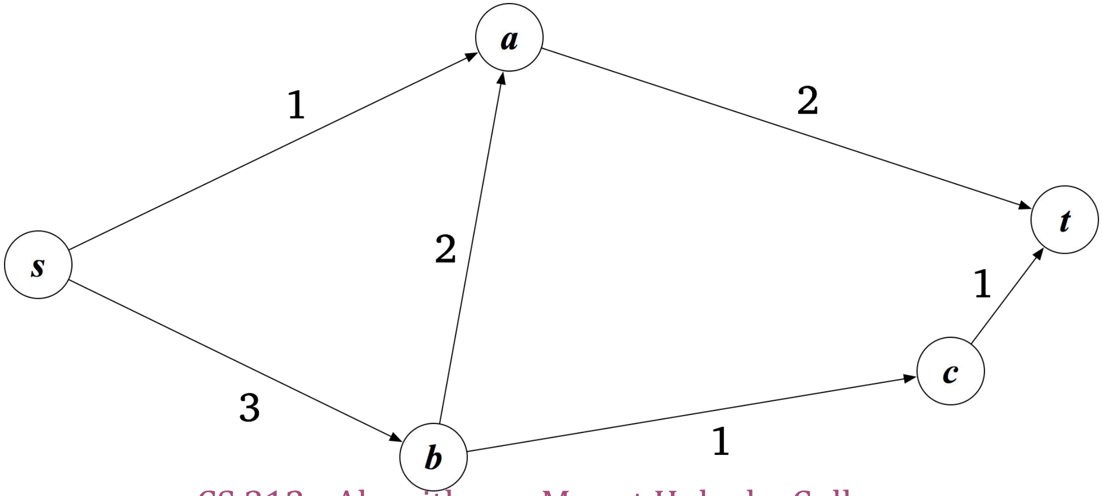
- Cut
  - ($A$,$B$) partition graph
- $s$-$t$ cut
  - $s$ in $A$ and $t$ in $B$
- Capacity of ($A$,$B$)
  - capacity out of $A$ (into $B$)



# Key proof ideas (K&T Ch. 7.2)

Key ideas

- Capacity of any cut gives *upper bound* on max flow
- Ford-Fulkerson stops at cut => lower + upper bounds meet
    - $A^*$ is set of vertices reachable from $s$ in residual graph
- In fact, max-flow = min-cut



# Step 4: running time analysis

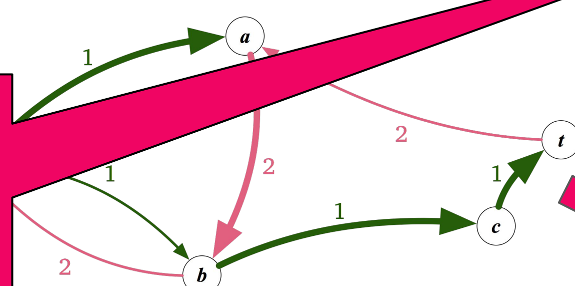
Total: O(Cm)

- Init flow *f(e)* = 0 for all *e*
- Given a flow *f*, build residual graph
- Find a simple *s*-*t* path, if one exists
- Augment using bottleneck to get new flow *f'*

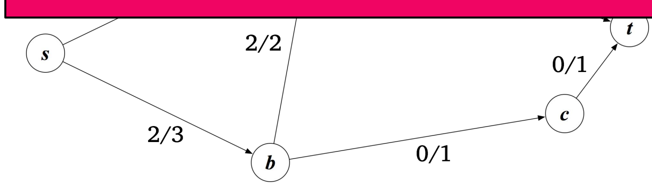
O($m$)

O($m$)

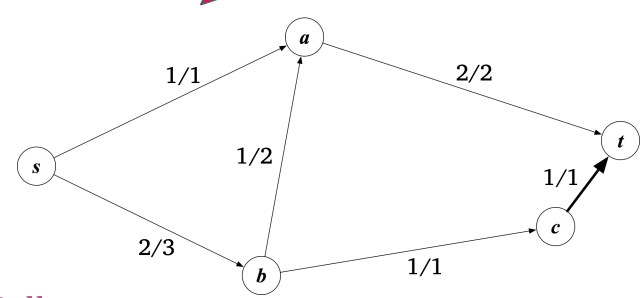
Repeat…
At most C times
(C = total capacity out of *s*)




residual graph




flow




augmented flow