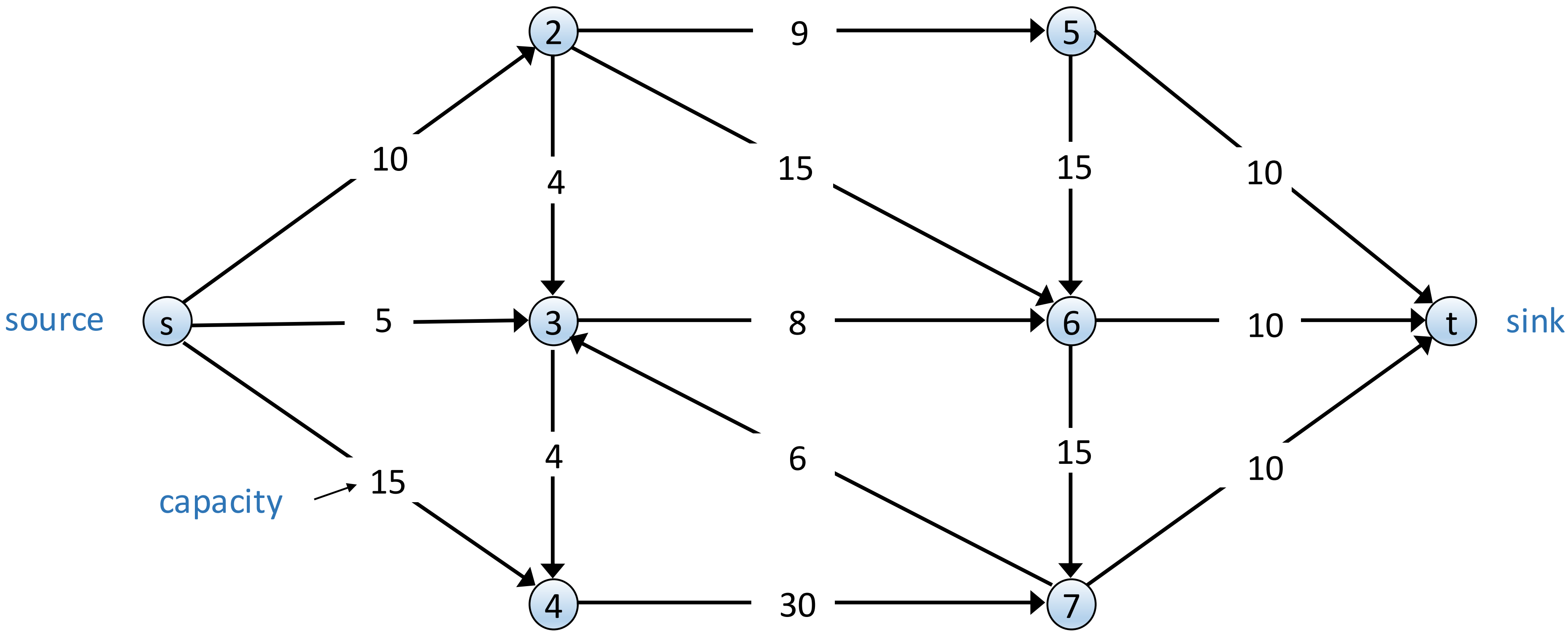
Network Flow

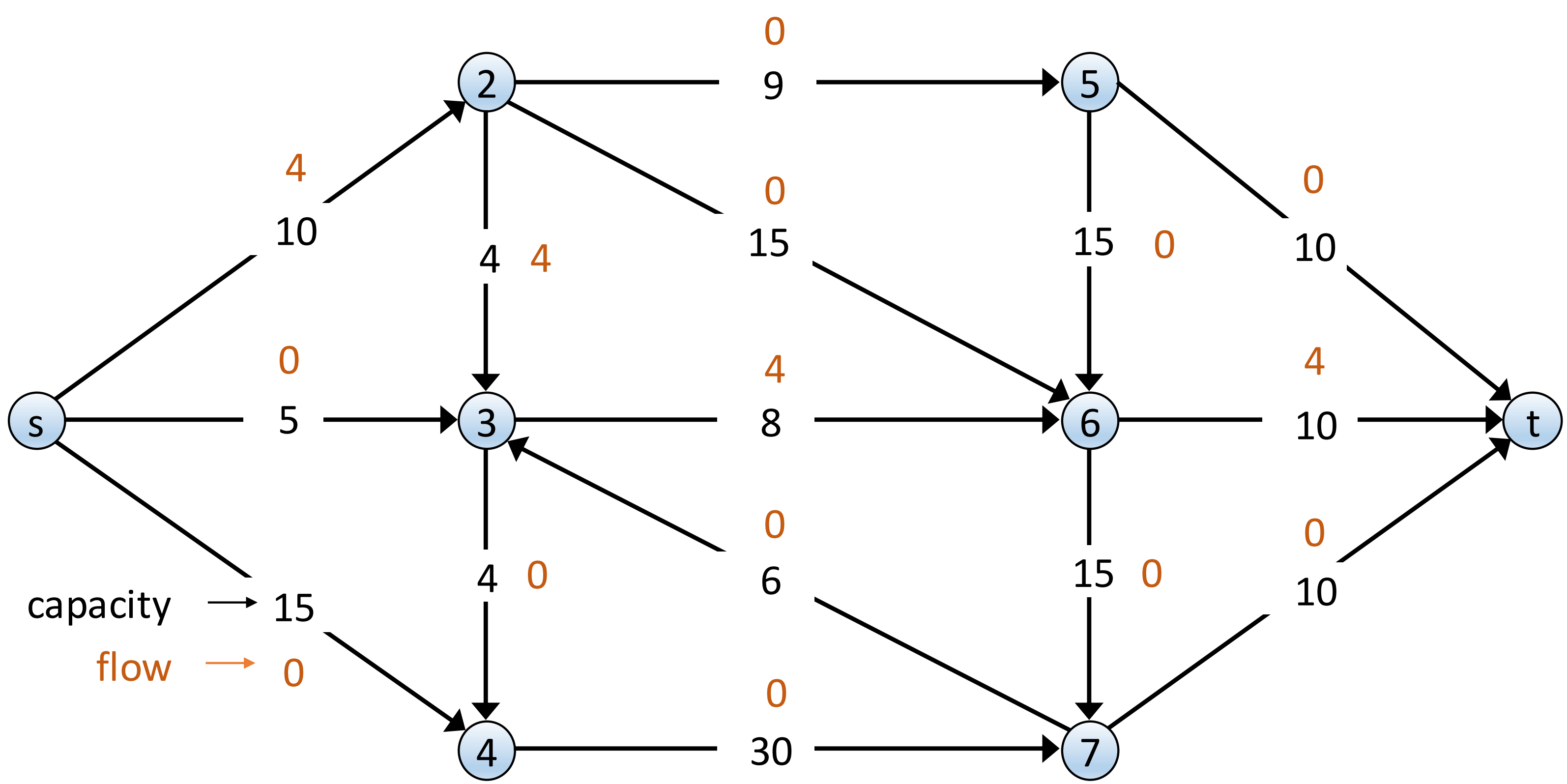
Flow Networks

- Directed graph $G = (V, E)$
- Two special nodes: source s and sink t
- Edge capacities $c(e)$
- Assume strongly connected (for simplicity)



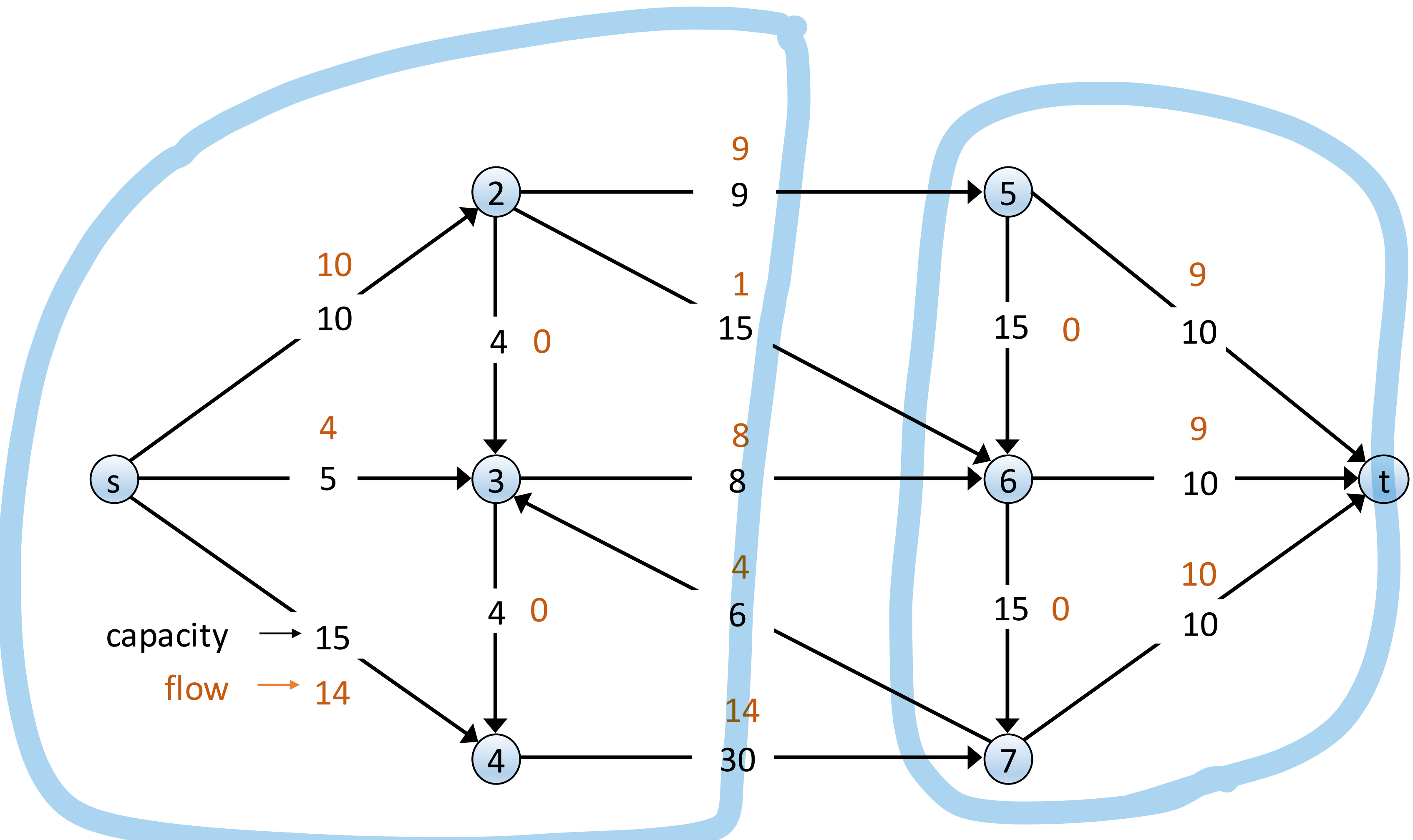
Flows

- An s-t flow is a function $f(e)$ such that
  - For every $e \in E$, $0 \le f(e) \le c(e)$ (capacity)
  - For every $v \in V \setminus \{s,t\}$, $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)
- The value of a flow is $val(f) = \sum_{e \text{ out of } s} f(e)$



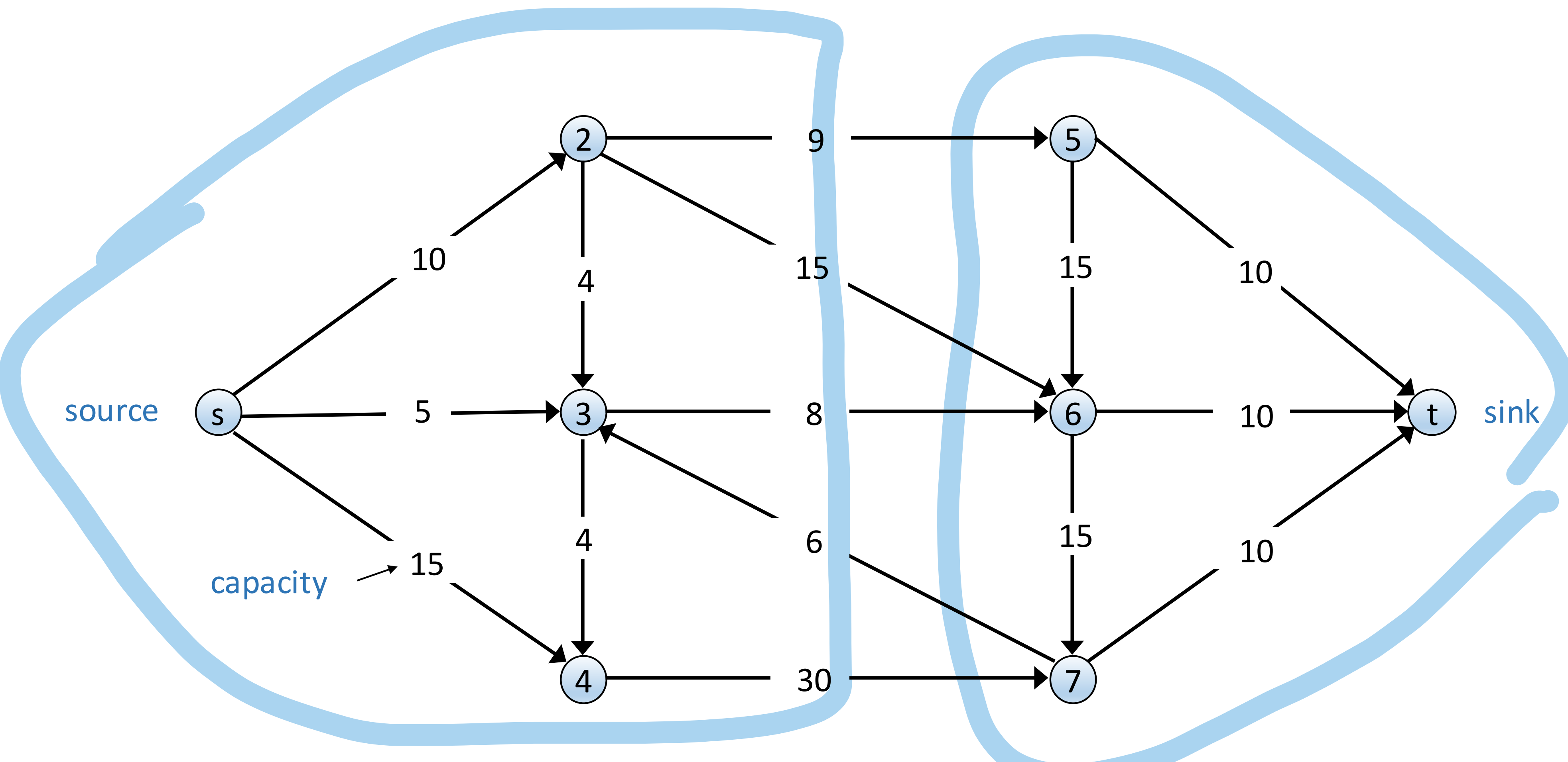
Maximum Flow Problem

- Given G = (V,E,s,t,{c(e)}), find an s-t flow of maximum value
- value(f) = 10 + 4 + 14 = 28



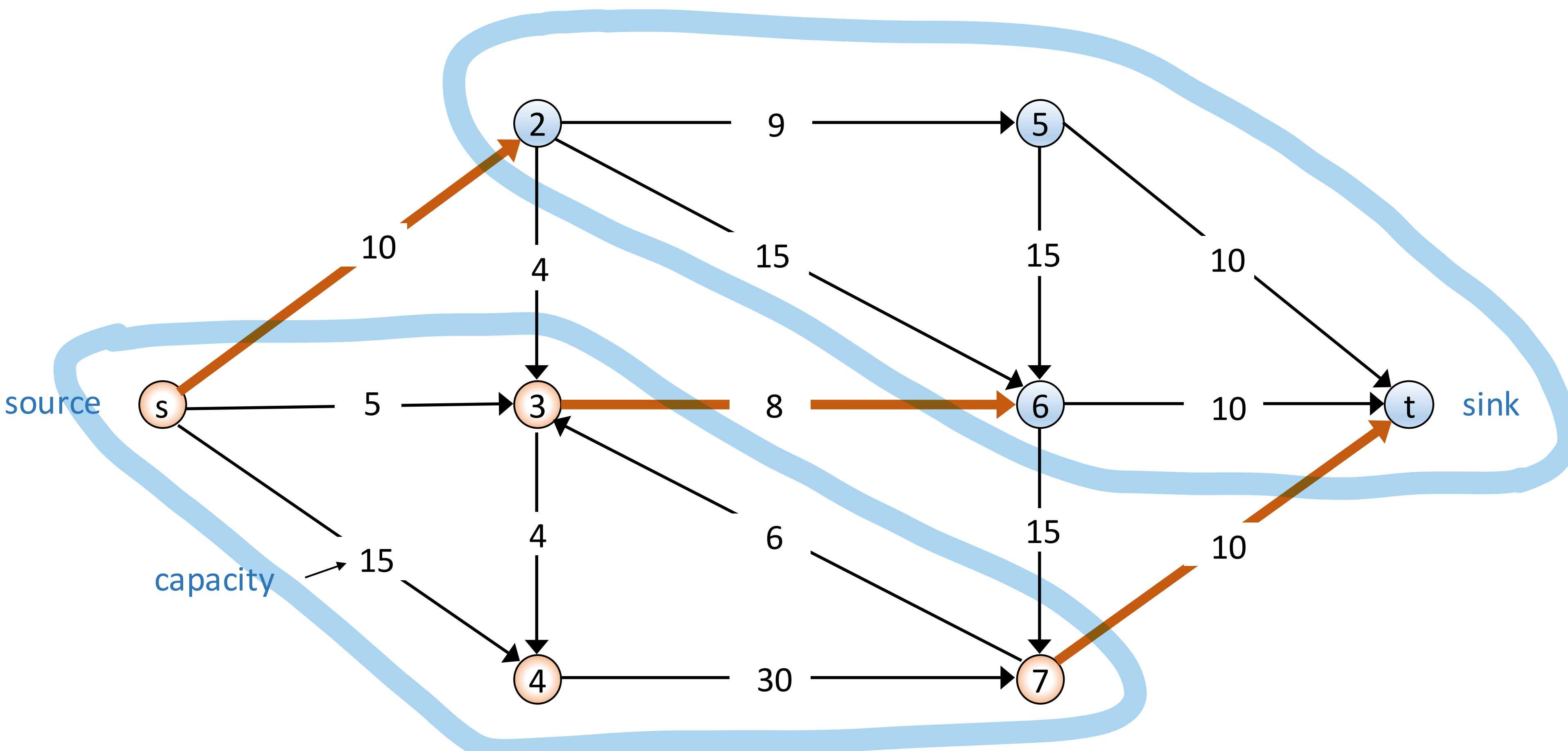
Cuts

- An s-t cut is a partition (A, B) of V with $s \in A$ and $t \in B$
- The capacity of a cut (A,B) is $cap(A,B) = \sum_{e \text{ out of } A} c(e)$



Minimum Cut problem

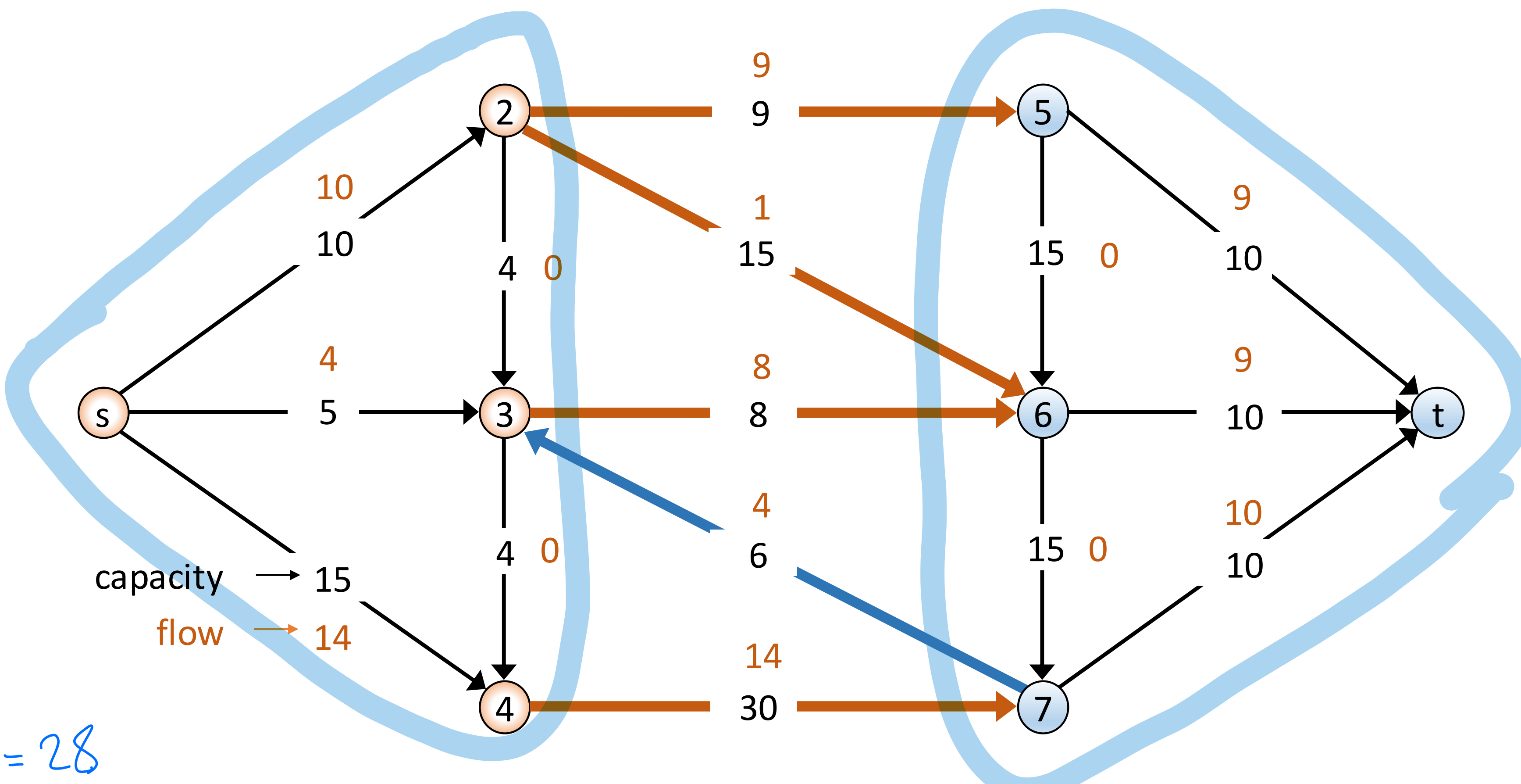
- Given G = (V,E,s,t,{c(e)}), find an s-t cut of minimum capacity
- cap({s,3,4,7}, {2,5,6,t}) = 28



Flows & Cuts: Closely Related

- **Fact:** If f is *any* s-t flow and (A, B) is any s-t cut, then the net flow across (A, B) is equal to the amount leaving s
- The net flow across any s-t cut is the same!

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = val(f)$$



val(f) = 28

Cuts & Flows

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

$$val(f) \leq cap(A, B) = \sum_{e \text{ out of } A} c_e$$

$$val(f) \overset{\text{Earlier Fact}}{=\!=} \sum_{e \text{ out of } A} f_e - \sum_{e \text{ into } A} f_e$$

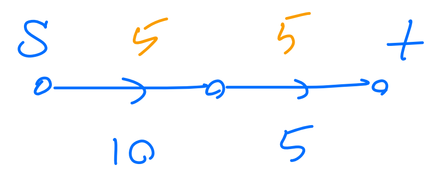
$$\leq \sum_{e \text{ out of } A} c_e - 0$$

$$= cap(A, B).$$

True or False?

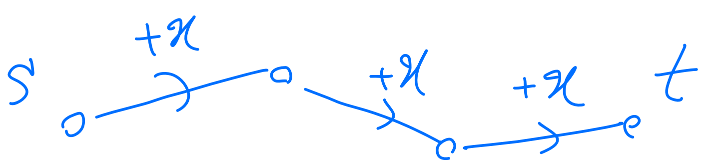
- The max flow always has an edge *e* leaving the source *s* such that *f(e) = c(e)* (is **saturated)?**

False



- The max flow always has an edge *e* such that *f(e) = c(e)* (is **saturated)?**

True: Take any path from s to t and increase the flow over the path.

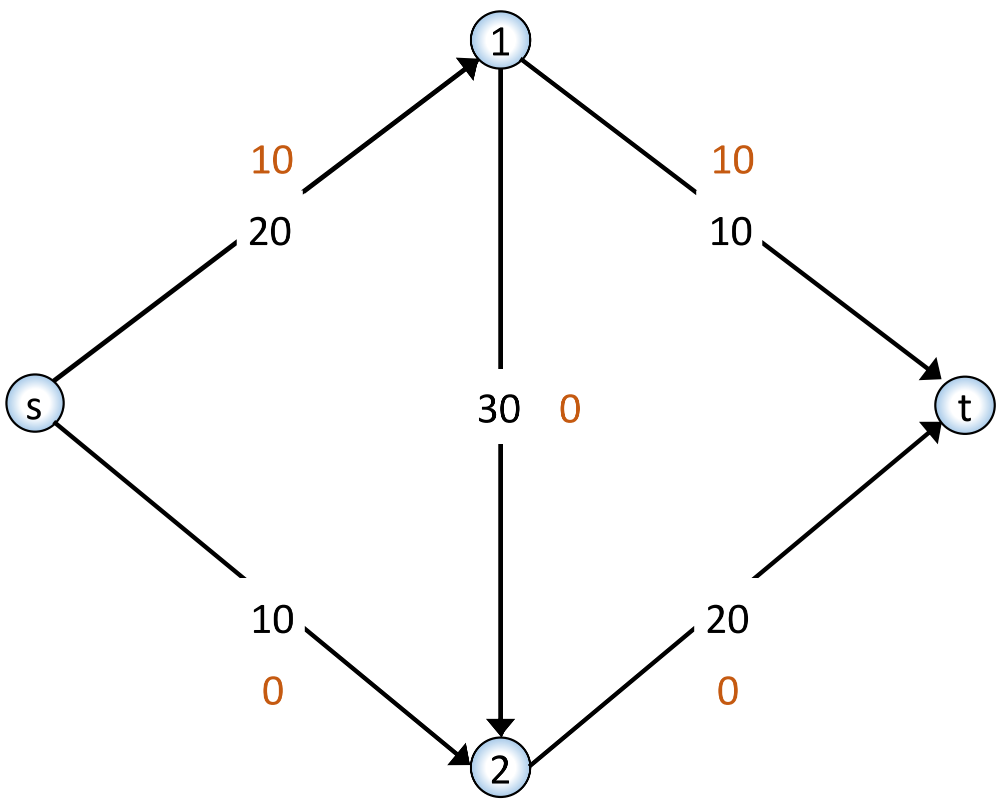


Network Flow

a. Key concepts and problem definitions
b. Augmenting paths nd greedy max flow

Augmenting Paths

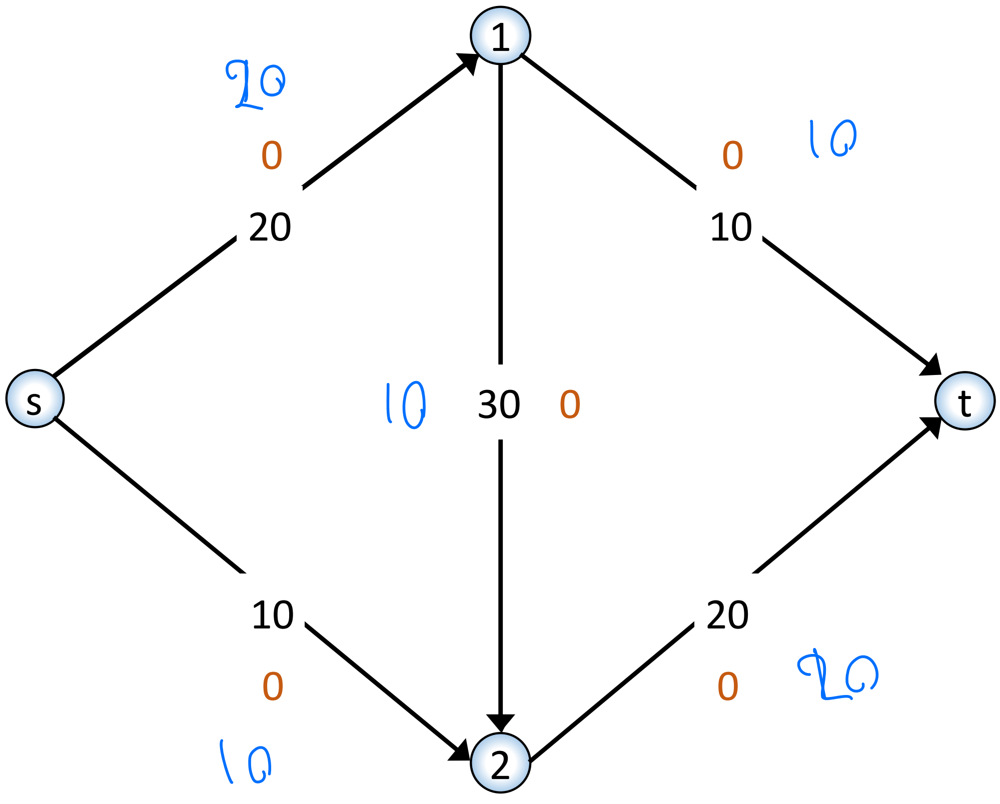
- Given a network $G = (V, E, s, t, \{c(e)\})$ and a flow f, an **augmenting path** P is a simple $s \to t$ path such that $f(e) < c(e)$ for every edge $e \in P$



- Are these augmenting paths?
  - ✗ s – 1 – t
  - ✓ s – 2 – t
  - ✓ s – 1 – 2 - t

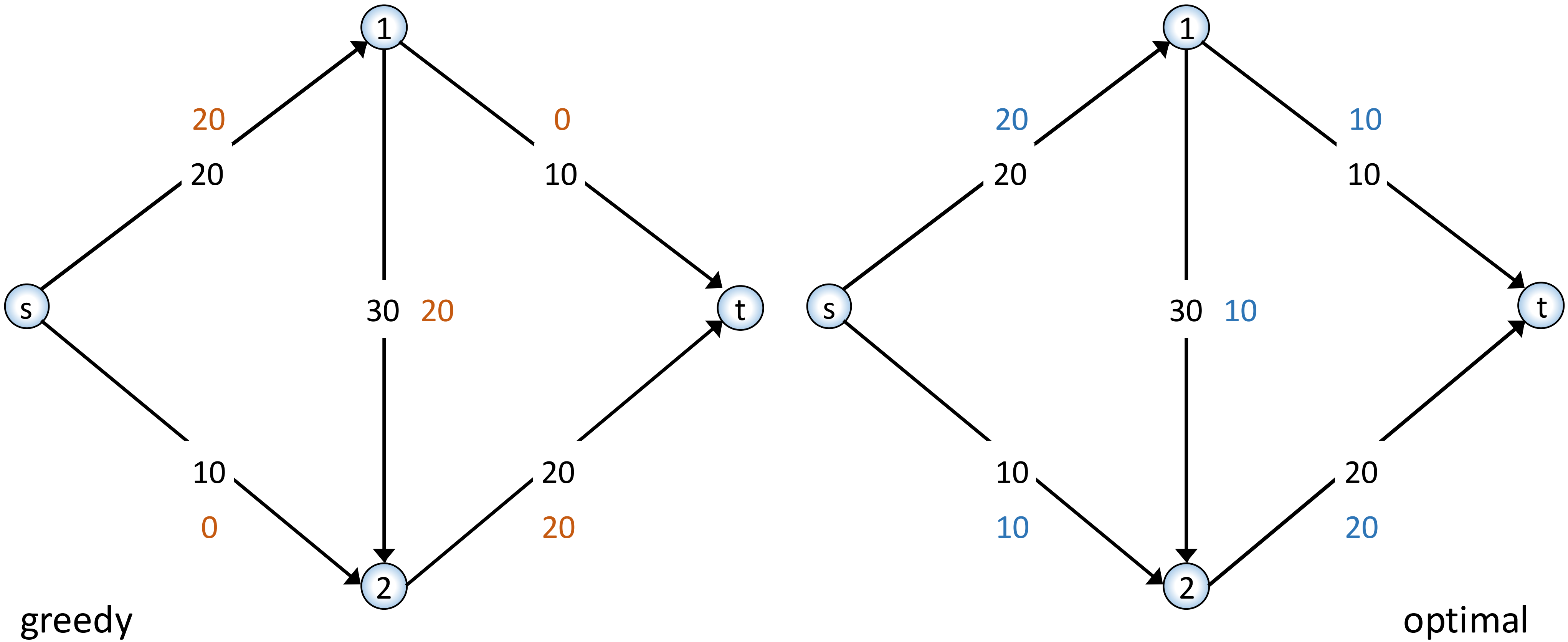
Greedy Max Flow

- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P & increase flow by max amount
- Repeat until you get stuck



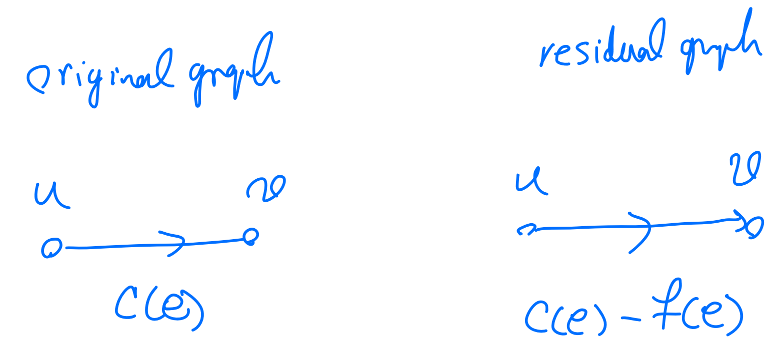
Does Greedy Work?

- Greedy gets stuck before finding a max flow
- How can we get from our solution to the max flow?



Residual Graphs

- Original edge: $e = (u, v) \in E$.
  - Flow $f(e)$, capacity $c(e)$
  - Residual capacity: $c(e) - f(e)$



- Residual edge
  - Allows "undoing" flow
  - $e = (u, v)$ and $e^R = (v, u)$.
  - cap(e^R) = $f(e)$

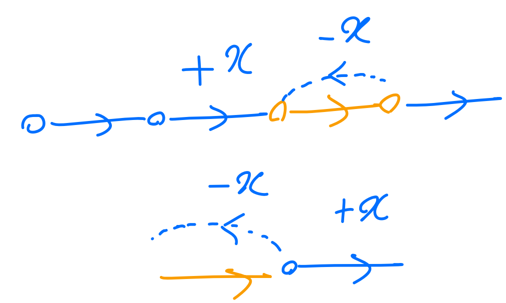
- Residual graph $G_f = (V, E_f)$
  - Original edges with positive residual capacity & residual edges with positive flow
  - $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

CS3000: Algorithms & Data

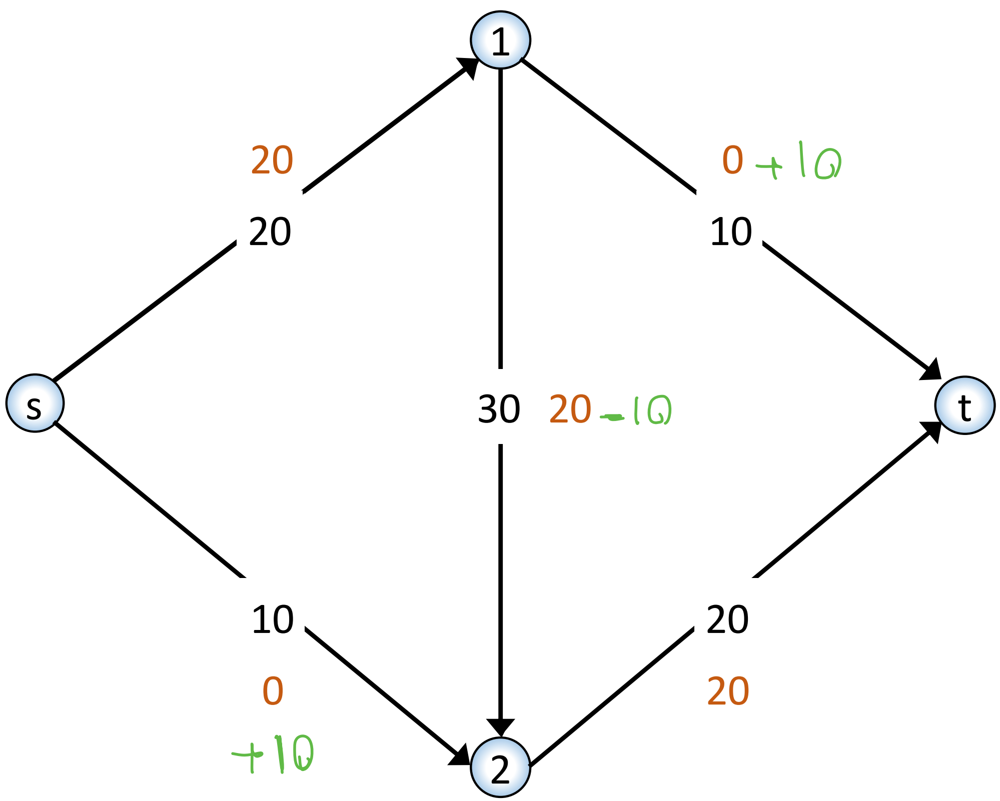
Unit 7: Network Flow

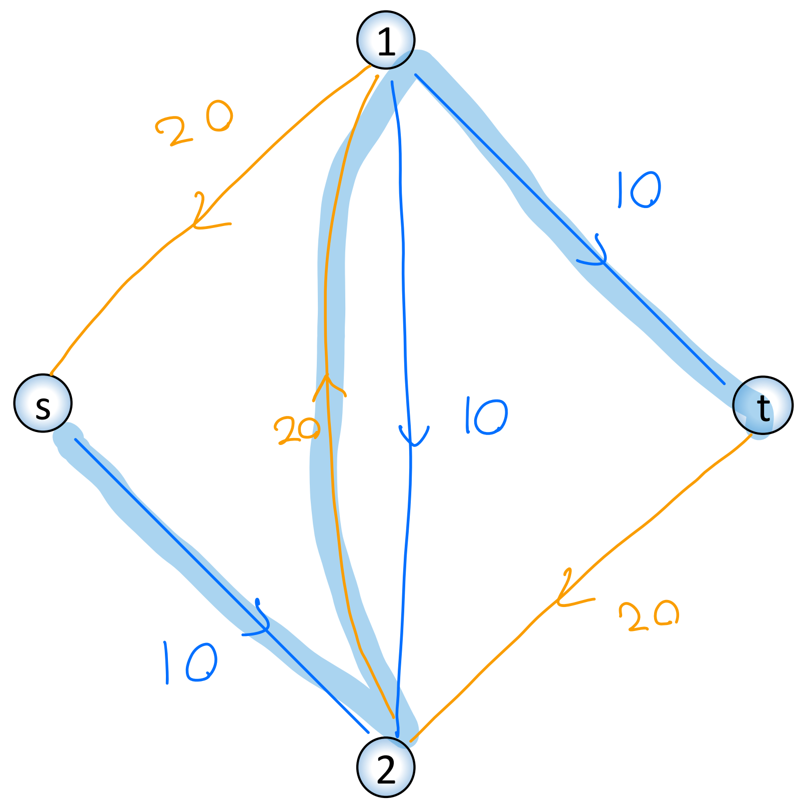
a. Key concepts and problem definitions
b. Augmenting paths and greedy max flow
c. The Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm



- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck





Augmenting Paths in Residual Graphs

- Let G_f be a **residual graph**
- Let P be an augmenting path in the **residual graph**
- **Fact:** $f' = \text{Augment}(G_f, P)$ is a valid flow

```
Augment(G_f, P)
    b ← the minimum capacity of an edge in P
    for e ∈ P
        if (e is an original edge):
            f(e) ← f(e) + b
        else:
            f(e^R) ← f(e^R) - b
    return f
```

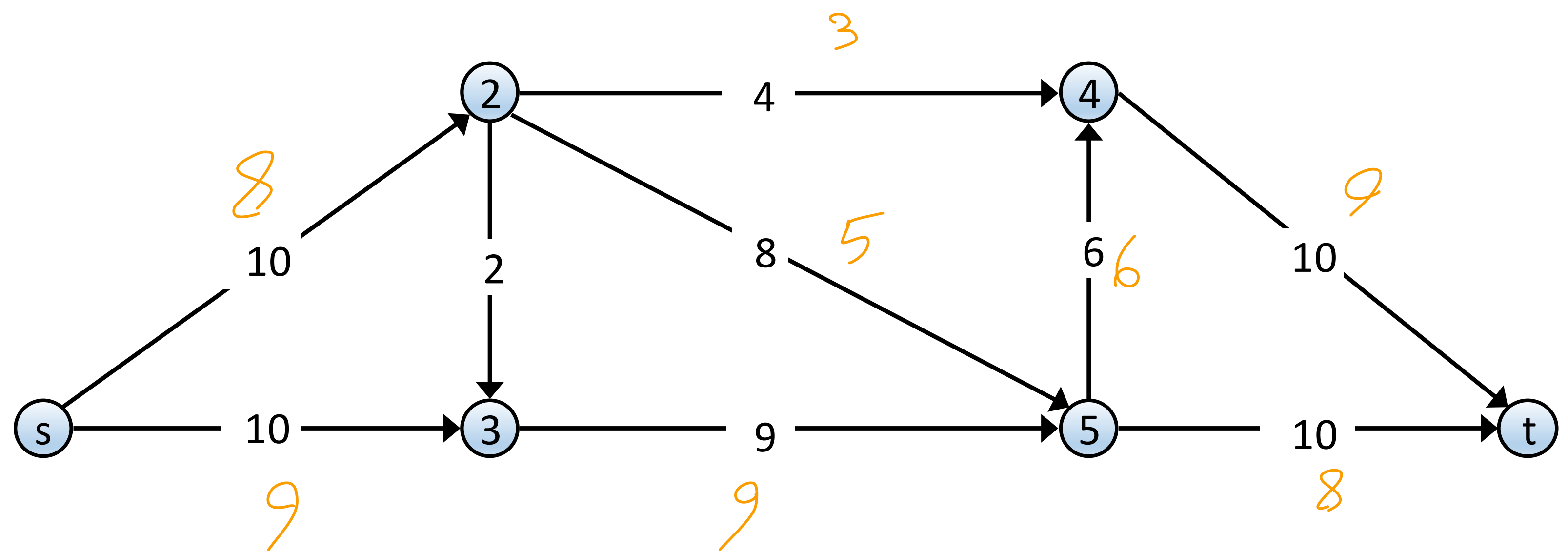
Ford-Fulkerson Algorithm

```
FordFulkerson(G,s,t,{c(e)})
    for e ∈ E: f(e) ← 0
    Gf is the residual graph

    while (there is an s-t path P in Gf)
        f ← Augment(Gf,P)
        update Gf

    return f
```

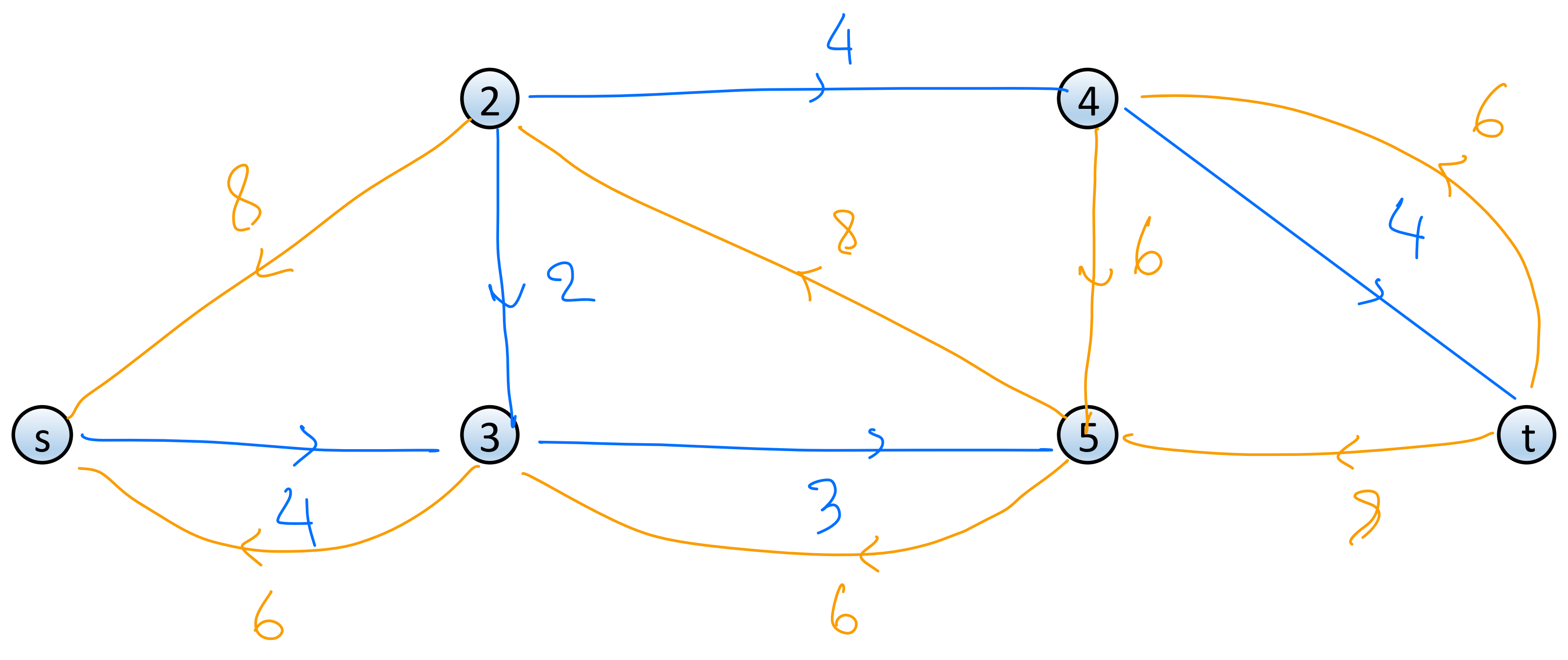
```
Augment(Gf, P)
    b ← the minimum capacity of an edge in P
    for e ∈ P
        if (e is an original edge): f(e) ← f(e) + b
        else:  f(e^R) ← f( e^R) - b
    return f
```

Ford-Fulkerson Demo

G:



G_f:



What do we want to prove?

Running Time of Ford-Fulkerson

- For **integer capacities**, $\leq val(f^*)$ augmentation steps

- Can perform each augmentation step in $O(m)$ time
  - find augmenting path in $O(m)$
  - augment the flow along path in $O(n)$
  - update the residual graph along the path in $O(n)$

- For integer capacities, FF runs in $O\big(m \cdot val(f^*)\big)$ time
  - $O(mn)$ time if all capacities are $c_e = 1$
  - $O(mnC_{\max})$ time for any integer capacities $\leq C_{\max}$
  - Problematic when capacities are large—more on this later!

Network Flow

a. Key concepts and problem definitions
b. Augmenting paths and greedy max flow
c. The Ford-Fulkerson Algorithm
d. Optimality of Ford-Fulkerson and Duality

Optimality of Ford-Fulkerson

- **Theorem:** f is a maximum s-t flow if and only if there is no augmenting s-t path in G_f

- **Strong MaxFlow-MinCut Duality:** The value of the max s-t flow equals the capacity of the min s-t cut

- We'll prove that the following are equivalent for all f
  1. There exists a cut (A, B) such that $val(f) = cap(A, B)$
  2. Flow f is a maximum flow
  3. There is no augmenting path in G_f

we proved last time that for any s-t flow f, and any s-t cut A, B,

$$val(f) \leq cap(A, B).$$

Optimality of Ford-Fulkerson

- **Theorem:** the following are equivalent for all f
  1. There exists a cut (A, B) such that $val(f) = cap(A, B)$
  2. Flow f is a maximum flow
  3. There is no augmenting path in G_f

Optimality of Ford-Fulkerson

- **(3 → 1)** If there is no augmenting path in G_f, then there is a cut (A, B) such that $val(f) = cap(A, B)$
  - Let A be the set of nodes reachable from s in G_f
  - Let B be all other nodes

Optimality of Ford-Fulkerson

- **(3 → 1)** If there is no augmenting path in G_f, then there is a cut (A, B) such that $val(f) = cap(A, B)$
  - Let A be the set of nodes reachable from s in G_f
  - Let B be all other nodes
  - **Key observation:** no edges in G_f go from A to B

residual graph

Take an edge e that crosses the cut

- If e is $A \to B$, then $f(e) = c(e)$
- If e is $B \to A$, then $f(e) = 0$

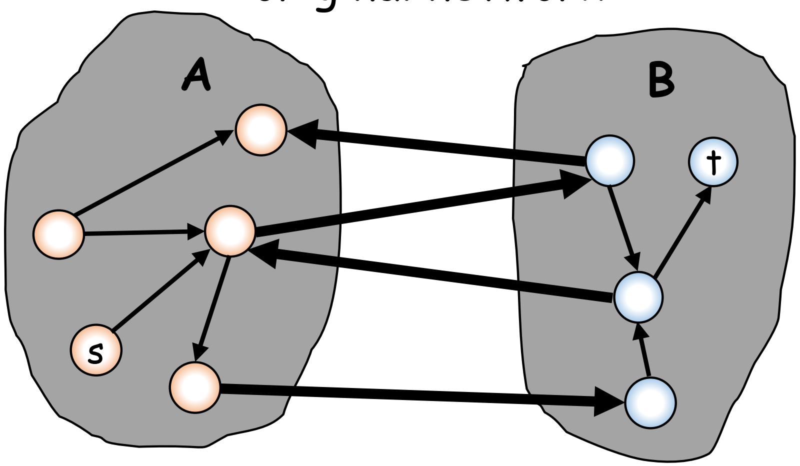
last session's Fact

$val(f) =$ net flow across (A,B) in G

$$= \sum_{e: A \to B} f_e - \sum_{e: B \to A} f_e$$

$$= cap(A,B) - 0 = cap(A,B)$$

original network



Construction of the residual graph:

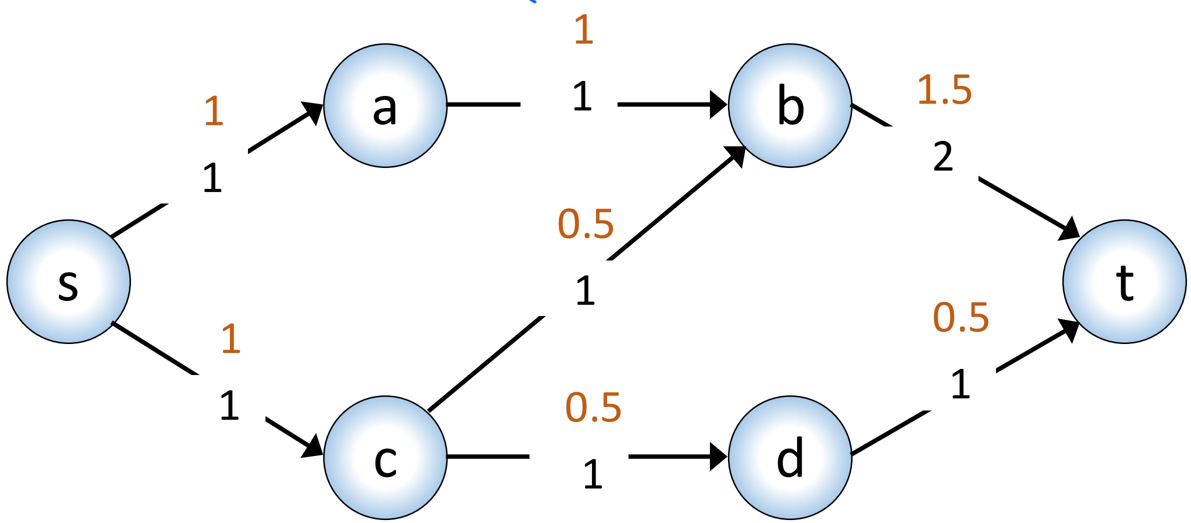
* include e in G_f if $f_e < c_e$ $\quad \forall e = (u,v)$ in G

* include e^R in G_f if $f_e > 0$ "

Ask the Audience

* Since for any flow f', $val(f') \leq cap(A,B)$, as discussed last time, f must be a max flow
* If there is another s-t cut A', B', with $cap(A',B') < cap(A,B)$ then $val(f) > cap(A',B')$ a contradiction.

- Is this a maximum flow?



- Is there an **integer maximum flow**? Yes
- Does every graph with **integer capacities** have an **integer maximum flow**? Yes

Summary

- **The Ford-Fulkerson Algorithm solves maximum s-t flow**
  - Running time $O\big(m \cdot val(f^*)\big)$ in networks with integer capacities

- **Strong MaxFlow-MinCut Duality: max flow = min cut**
  - The value of the maximum s-t flow equals the capacity of the minimum s-t cut
  - If f^* is a maximum s-t flow, then the set of nodes reachable from s in G_{f^*} gives a minimum cut
  - Given a max-flow, can find a min-cut in time $O(n+m)$

- **Every graph with integer capacities has an integer maximum flow**
  - Ford-Fulkerson will return an integer maximum flow

Network Flow

a. Key concepts and problem definitions
b. Augmenting paths and greedy max flow
c. The Ford-Fulkerson Algorithm
d. Optimality of Ford-Fulkerson and Duality
e. Choosing good augmenting paths

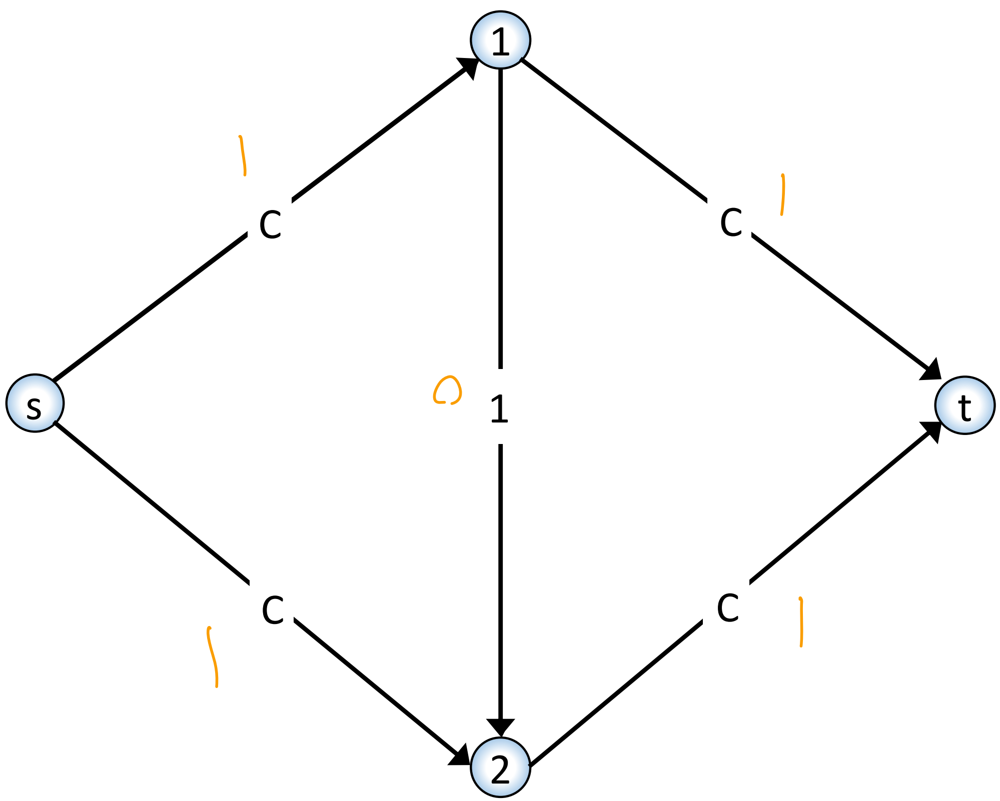
Speeding Up Ford-Fulkerson

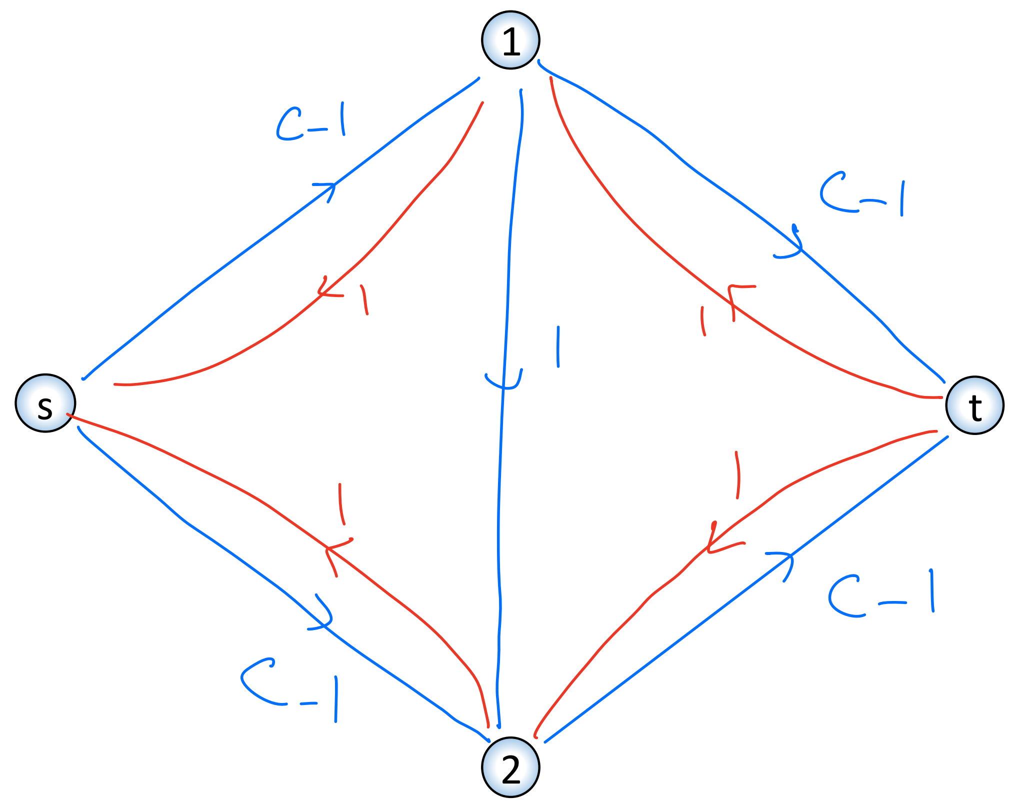
$val(f^*) = 2C$

- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph** G_f
- Repeat until you get stuck

$\Theta(val(f^*))$

with a bad choice of the augmenting path, the FF algorithm runs in $\Theta(m \cdot C)$ time





Choosing Good Augmenting Paths

- **Last time:** arbitrary augmenting paths
  - If Ford-Fulkerson terminates, then we have found a max flow
  - Can construct capacities where the algorithm never terminates
  - Can require many augmenting paths to terminate

- **Today:** clever augmenting paths
  - Maximum-capacity augmenting path (“fattest path”)
  - Shortest augmenting paths (“shortest path”)

Fattest Augmenting Path

- Maximum-capacity augmenting path

- Can find the fattest augmenting path in time $O(m \log C)$ in several different ways
  - Variants of Prim's or Kruskal's MST algorithm
  - BFS + binary search

Fattest Augmenting Path

Arbitrary Paths

- **Assume integer capacities**
- Value of maxflow: v^*
- Value of aug path: ≥ 1
- Flow remaining in G_f: $\leq v^* - 1$
- # of aug paths: $\leq v^*$

Maximum-Capacity Path

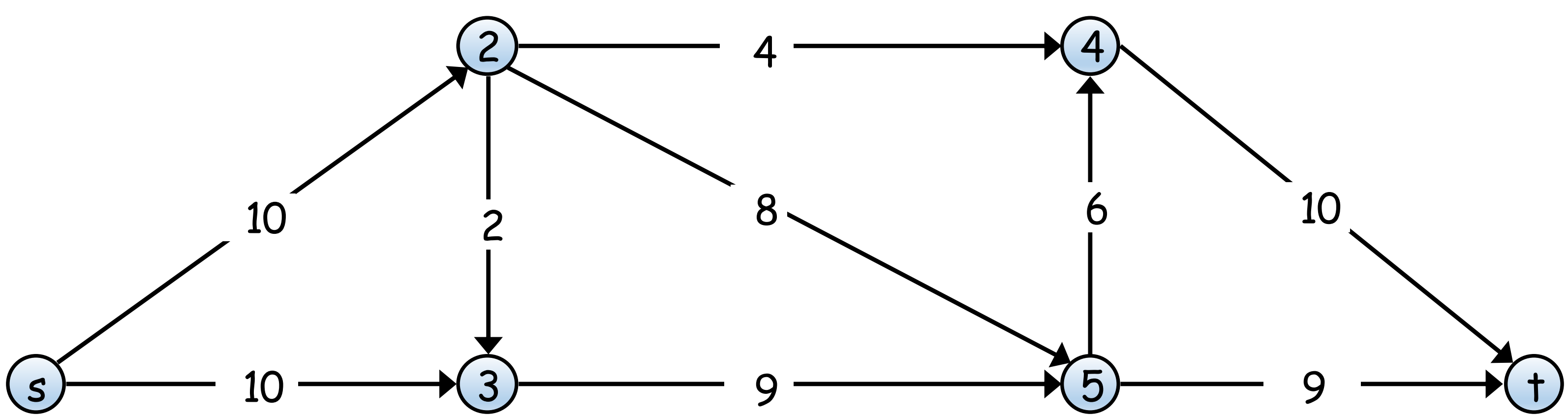
- **Assume integer capacities**
- Value of maxflow: v^*
- Value of aug path: $\frac{v^*}{m}$
- Flow remaining in G_f: $v^* - \frac{v^*}{m} = \left(1-\frac{1}{m}\right)v^*$
- # of aug paths:

$$v^* \longrightarrow \left(1-\frac{1}{m}\right)v^* \longrightarrow \left(1-\frac{1}{m}\right)^2 v^* \longrightarrow \cdots \longrightarrow \underbrace{\left(1-\frac{1}{m}\right)^m}\, v^* \longrightarrow \cdots \longrightarrow \left(1-\frac{1}{m}\right)^{\Theta(m \log v^*)} v^*$$

$$< 1.$$

Fattest Augmenting Path

- f^* is a maximum flow with value $v^* = val(f^*)$
- P is a fattest augmenting s-t path with capacity B
- **Key Claim:** $B \geq \frac{v^*}{m}$



Fattest Augmenting Path

- f^* is a maximum flow with value $v^* = val(f^*)$
- P is a fattest augmenting s-t path with capacity B
- **Key Claim:** $B \geq \frac{v^*}{m}$
- **Proof:**

Fattest Augmenting Path

Arbitrary Paths

- **Assume integer capacities**

- Value of maxflow: v^*
- Value of aug path: ≥ 1
- Flow remaining in G_f: $\leq v^* - 1$
- # of aug paths: $\leq v^*$

Maximum-Capacity Path

- **Assume integer capacities**

- Value of maxflow: v^*
- Value of aug path:
- Flow remaining in G_f:
- # of aug paths:

Choosing Good Paths

- **Last time:** arbitrary augmenting paths
  - If Ford-Fulkerson terminates, it has found a maximum flow

- **Today:** clever augmenting paths
  - Maximum-capacity augmenting path (“fattest augmenting path”)
    - $\leq m$ augmenting paths (assuming integer capacities)
    - $O(m^2 \ln C)$ total running time
      $\lg v^*$
  - Shortest augmenting paths (“shortest augmenting path”)

Shortest Augmenting Path & Improvements

- Find the augmenting path with the fewest hops
  - Can find shortest augmenting path in $O(m)$ time using BFS
- **Theorem:** for any capacities $nm/2$ augmentations suffice
  - Overall running time $O(m^2 n)$
  - Works for any capacities!
- **Warning:** the proof is challenging, so we will skip it
- **Better Theorem:** Max flow can be solved in $O(mn)$ time
  - You can use this fact for all future assignments/exams

$m^{1+o(1)}$ $\lg C$

Choosing Good Augmenting Paths

- **Last time:** arbitrary augmenting paths
  - If Ford-Fulkerson terminates, then we have found a max flow
  - Can construct capacities where the algorithm never terminates
  - Can require many augmenting paths to terminate

- **Today:** clever augmenting paths
  - Maximum-capacity augmenting path ("fattest path")
  - Shortest augmenting paths ("shortest path")

Fattest Augmenting Path

- Maximum-capacity augmenting path

- Can find the fattest augmenting path in time $O(m \log m)$ in several different ways
  - Use a variant of Dijkstra or combine BFS & BinarySearch

Fattest Augmenting Path

Arbitrary Paths

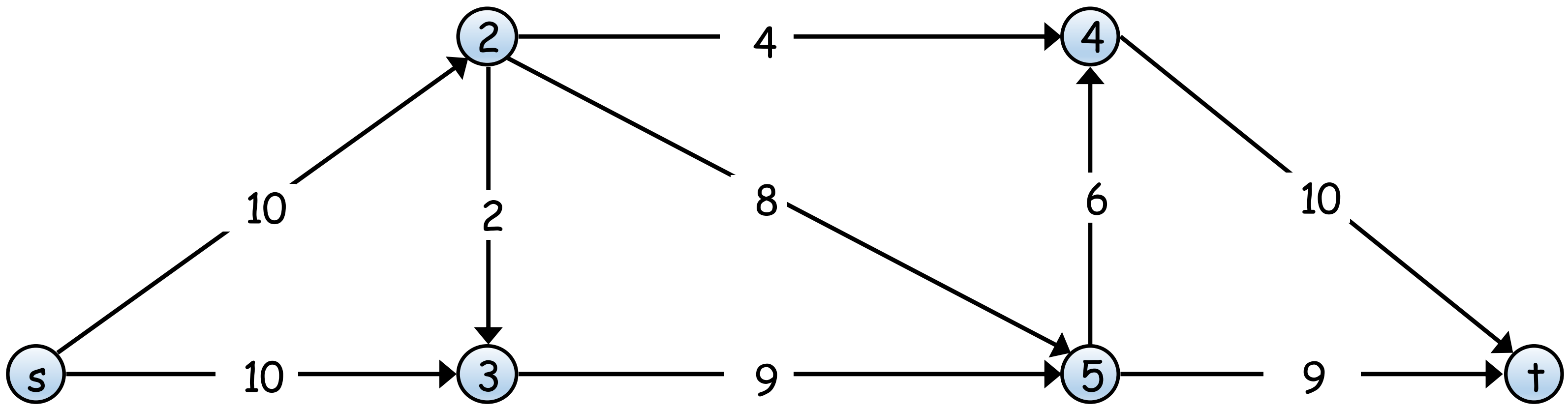
- **Assume integer capacities**
- Value of maxflow: v^*
- Value of aug path: ≥ 1
- Flow remaining in G_f: $\leq v^* - 1$
- # of aug paths: $\leq v^*$

Maximum-Capacity Path

- **Assume integer capacities**
- Value of maxflow: v^*
- Value of aug path:
- Flow remaining in G_f:
- # of aug paths:

Fattest Augmenting Path

- f^* is a maximum flow with value $v^* = val(f^*)$
- P is a fattest augmenting s-t path with capacity B
- **Key Claim:** $B \geq \frac{v^*}{m}$



Fattest Augmenting Path

- f^* is a maximum flow with value $v^* = val(f^*)$
- P is a fattest augmenting s-t path with capacity B
- **Key Claim:** $B \geq \frac{v^*}{m}$
- **Proof:**

Fattest Augmenting Path

Arbitrary Paths

- **Assume integer capacities**

- Value of maxflow: v^*
- Value of aug path: ≥ 1
- Flow remaining in G_f: $\leq v^* - 1$
- # of aug paths: $\leq v^*$

Maximum-Capacity Path

- **Assume integer capacities**

- Value of maxflow: v^*
- Value of aug path:
- Flow remaining in G_f:
- # of aug paths:

Choosing Good Paths

- **Last time:** arbitrary augmenting paths
  - If Ford-Fulkerson terminates, it has found a maximum flow

- **Today:** clever augmenting paths
  - Maximum-capacity augmenting path ("fattest augmenting path")
    - $\leq m \ln v^*$ augmenting paths (assuming integer capacities)
    - $O(m^2 \ln n \ln v^*)$ total running time
    - See KT for a faster variant ("fat-enough augmenting path"?)

  - Shortest augmenting paths ("shortest augmenting path")

Shortest Augmenting Path & Improvements

- Find the augmenting path with the fewest hops
  - Can find shortest augmenting path in $O(m)$ time using BFS

- **Theorem:** for any capacities $nm/2$ augmentations suffice
  - Overall running time $O(m^2 n)$
  - Works for any capacities!
- **Warning:** the proof is challenging, so we will skip it

- **Better Theorem:** Max flow can be solved in $O(mn)$ time
  - You can use this fact for all future assignments/exams

Applications of Network Flow

a. Reductions between computational problems

Applications of Network Flow

- Algorithms for maximum flow can be used to solve:
  - Bipartite Matching
  - Image Segmentation
  - Disjoint Paths
  - Survey Design
  - Matrix Rounding
  - Auction Design
  - Fair Division
  - Project Selection
  - Baseball Elimination
  - Airline Scheduling
  - …

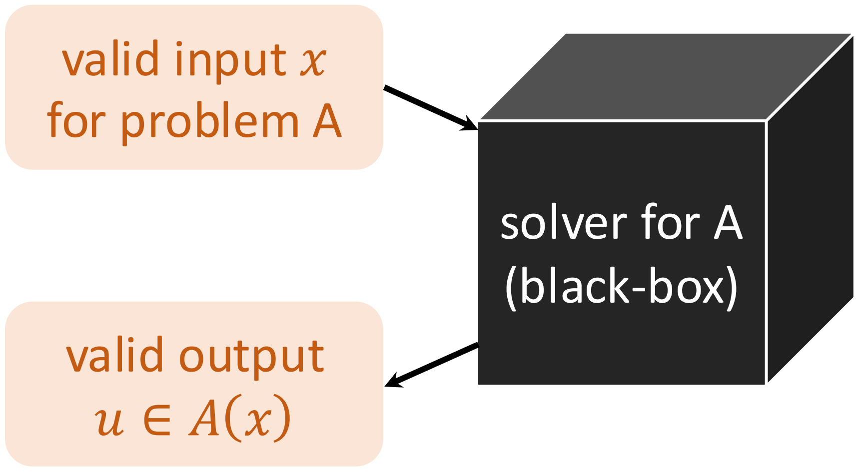
Mechanics of Reductions

- **Definition:** a computational **problem** is
  - a set of valid inputs $\boldsymbol{X}$ and
  - a set $\boldsymbol{A}(\boldsymbol{x})$ of valid outputs for each $\boldsymbol{x} \in \boldsymbol{X}$

- **Example:** integer maximum flow

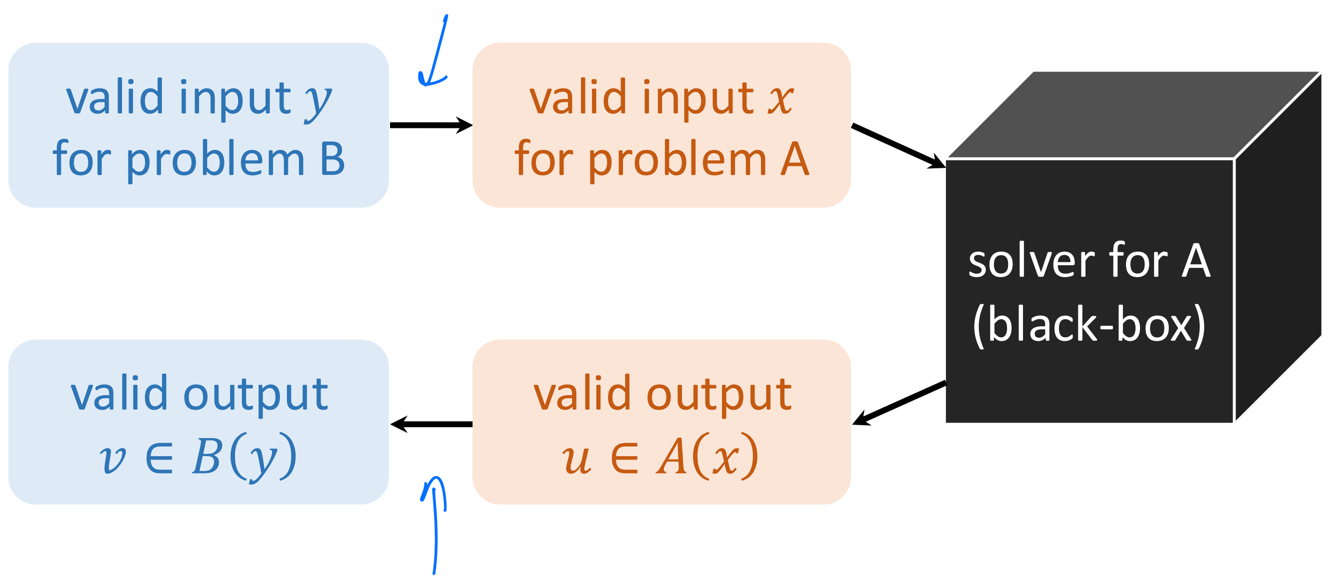
Mechanics of Reductions

- **Definition:** a **reduction** is an efficient algorithm that solves problem B using an algorithm that solves problem A as a **black-box**

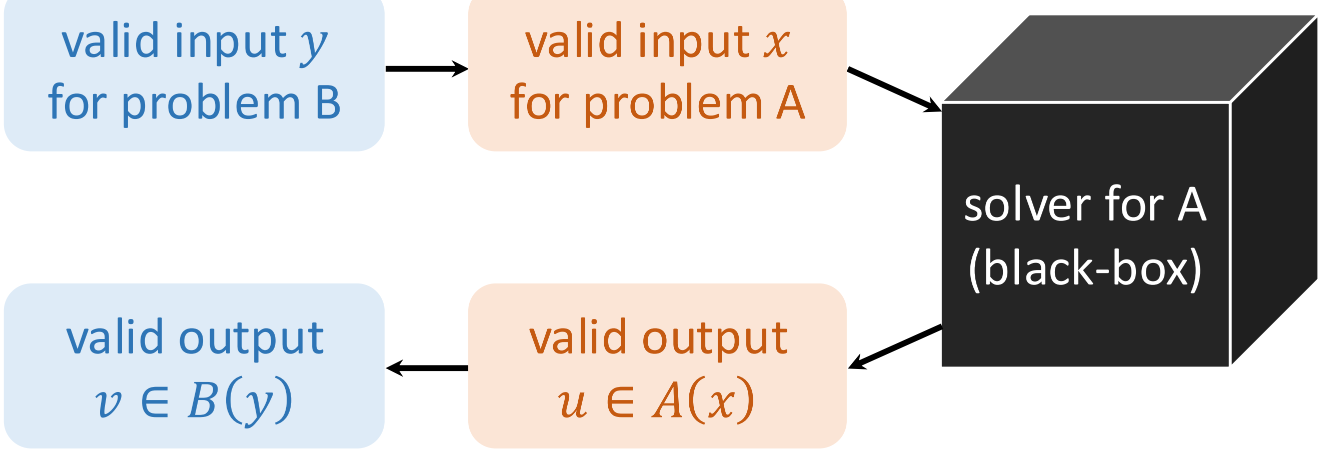


Mechanics of Reductions

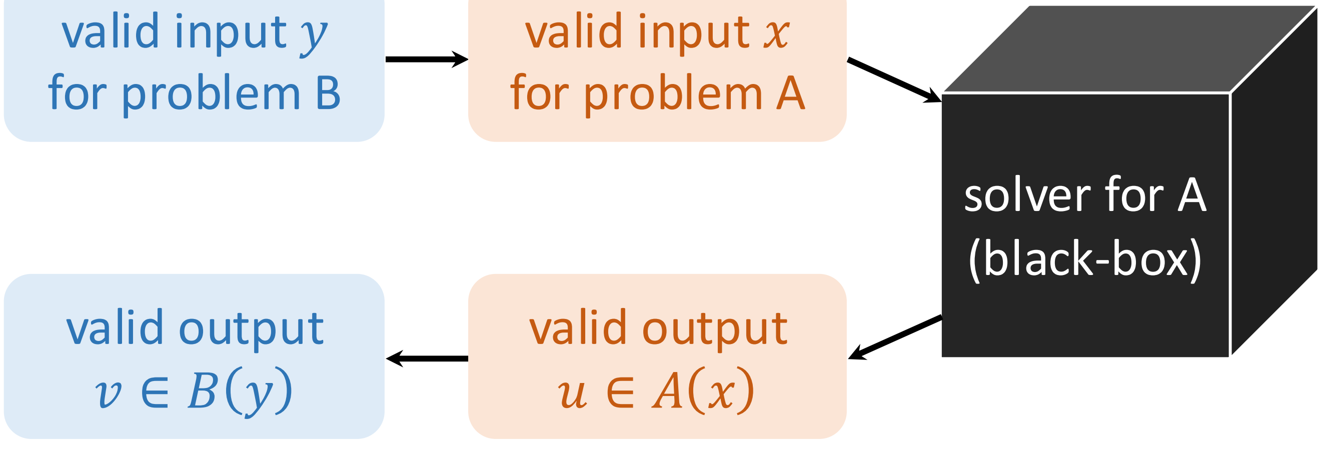
- **Definition:** a **reduction** is an efficient algorithm that solves problem B using an algorithm that solves problem A as a **black-box**



Correctness of Reductions



Running Time of Reductions

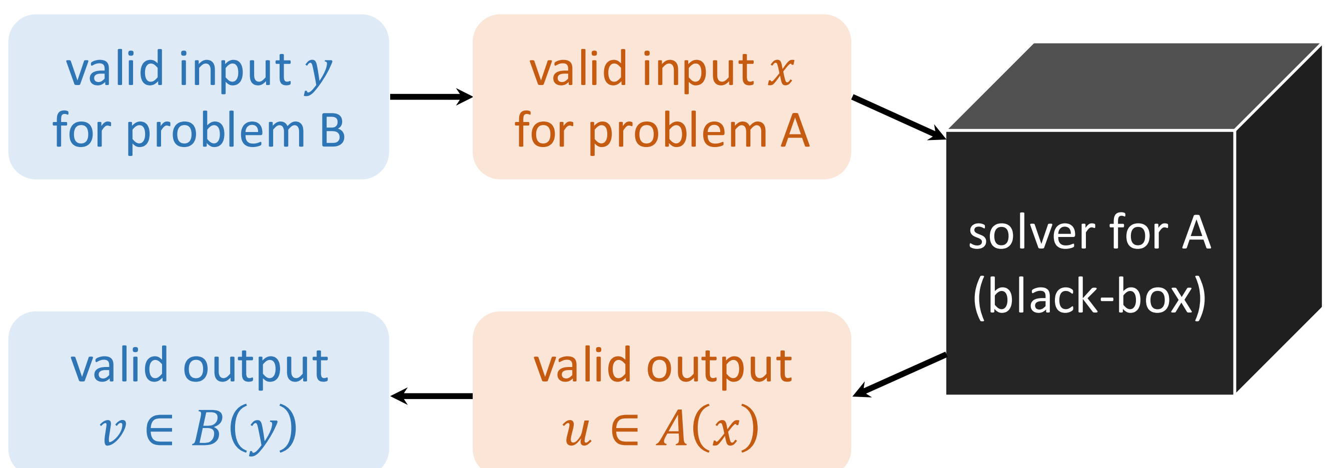


Example: Flows and Cuts

*

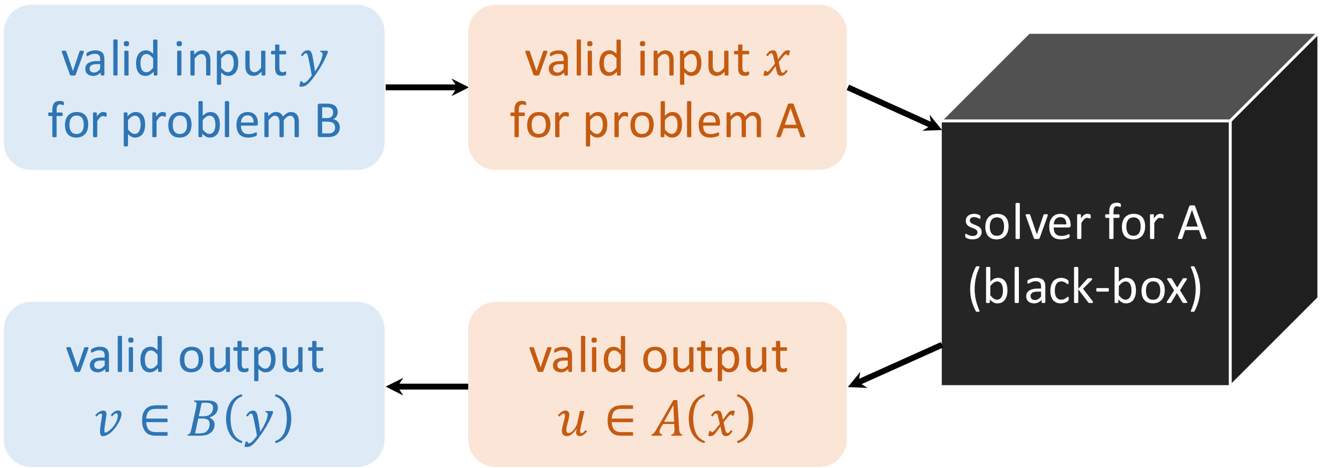
$O(m+n)$ + time to solve max flow

trivial



* Construct residual graph,
and take A = reachable(S)

Example: Sorting and Median

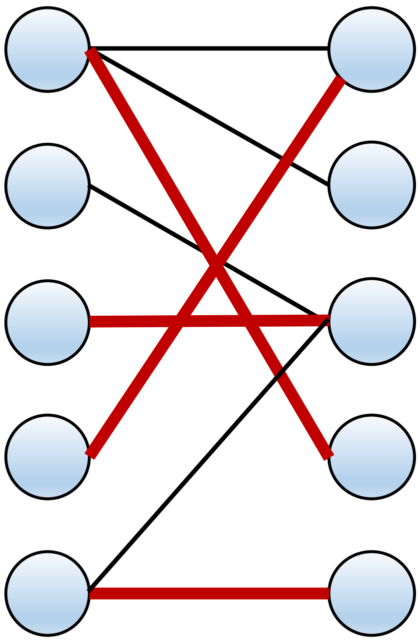


Maximum Bipartite Matching

- **Input:** bipartite graph $G = (V, E)$ with $V = L \cup R$
- **Output:** a matching of maximum size
  - A **matching** $M \subseteq E$ is a set of edges such that every node v is an endpoint of at most one edge in M
  - **Size** = $|M|$

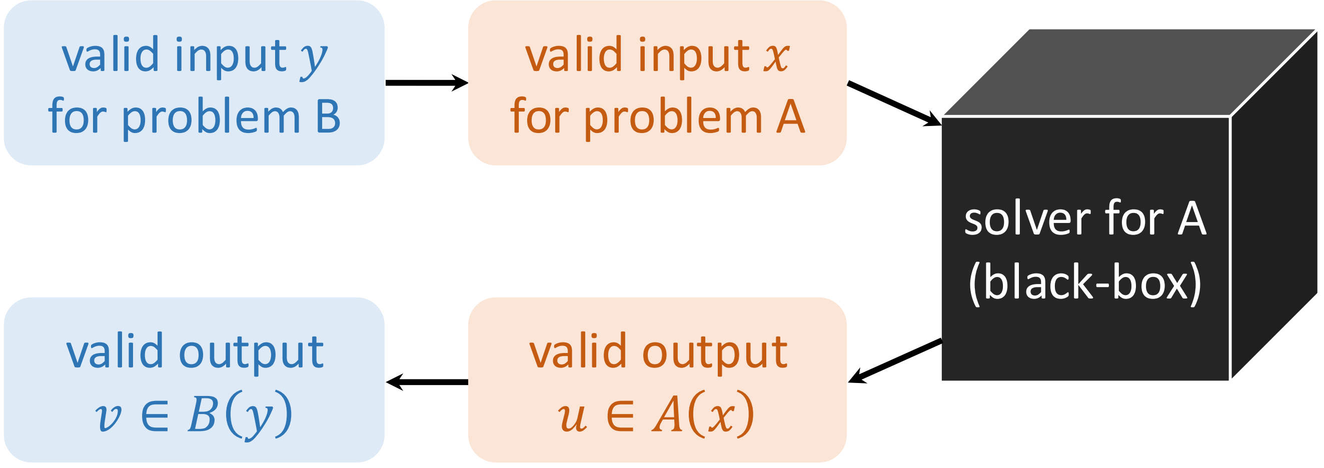
Models any problem where one type of object is assigned to another type:

- doctors to hospitals
- jobs to processors
- advertisements to websites



Mechanics of Reductions

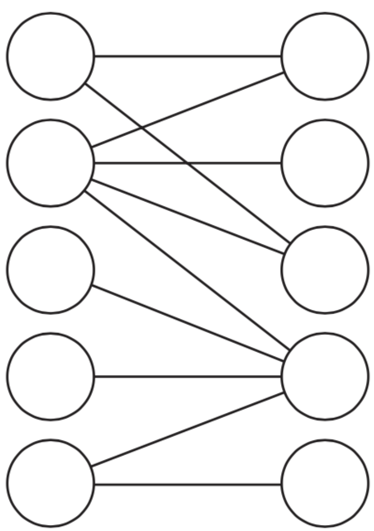
- **Theorem:** There is an efficient algorithm that solves maximum bipartite matching (MBM) using an algorithm that solves integer maximum s-t flow (MF)

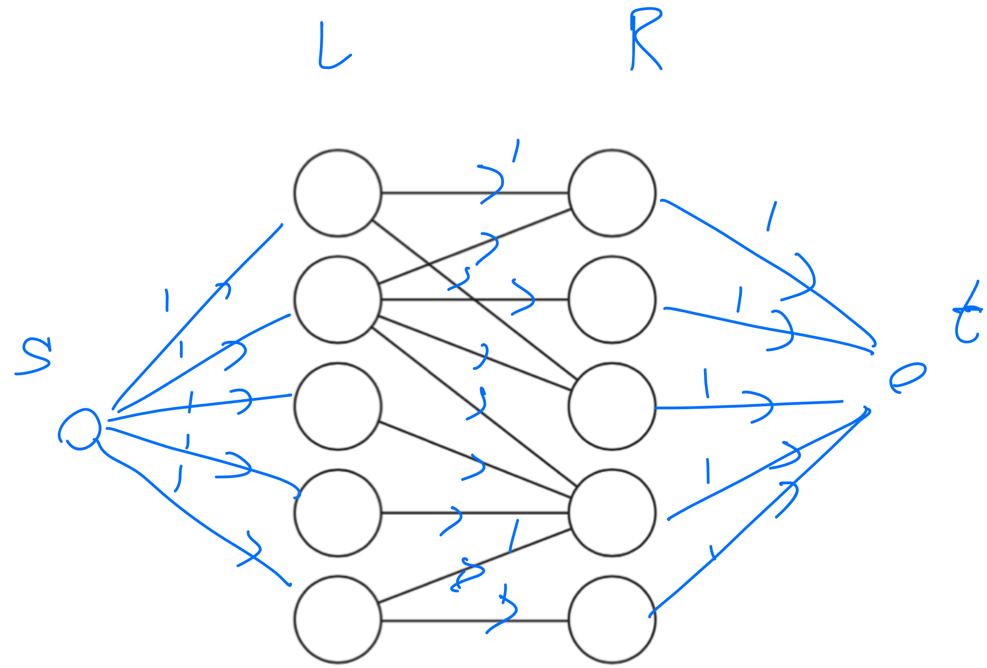


Step 1: Transform the Input

valid input G for MBM

valid network G' for MF

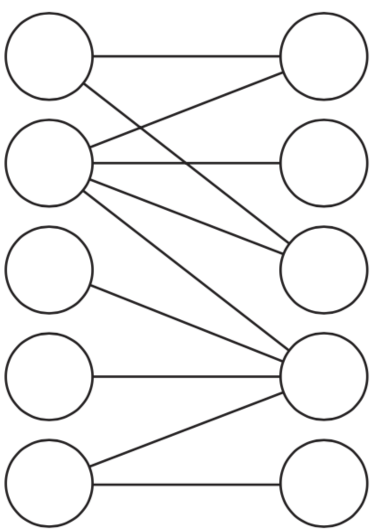


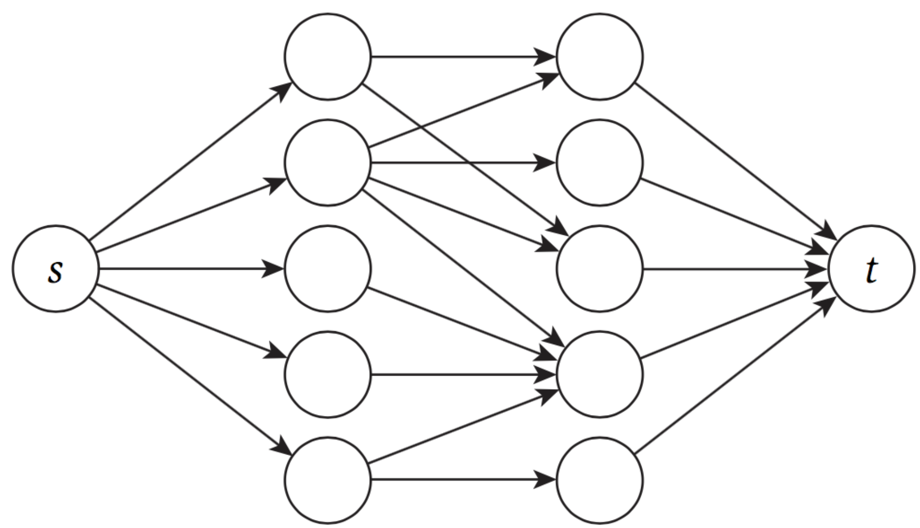


Step 1: Transform the Input

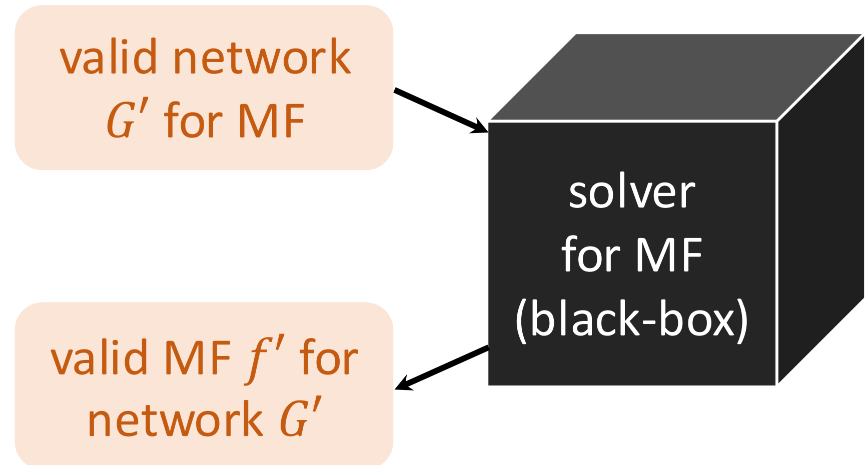
valid input G for MBM

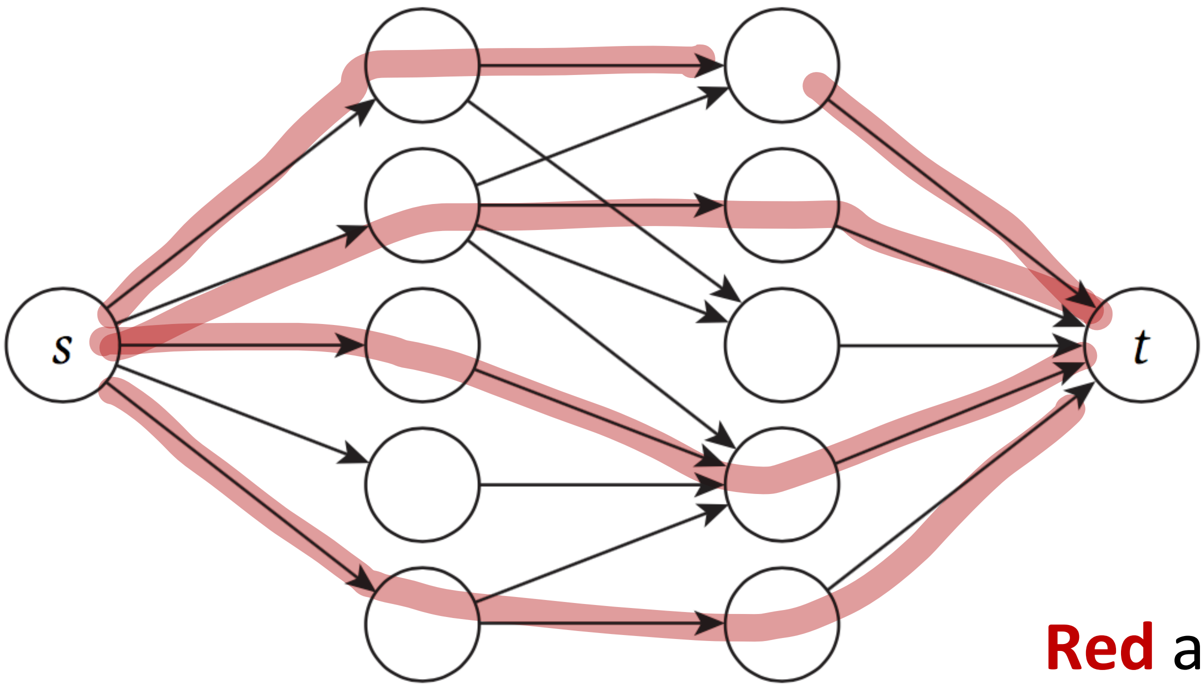
valid network G' for MF





Step 2: Receive the Output





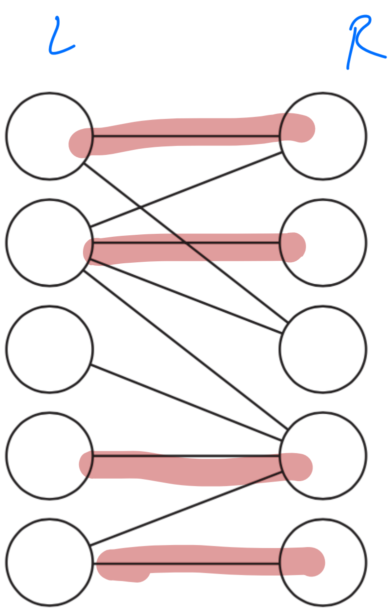
Red arrow means $f'(e) = 1$

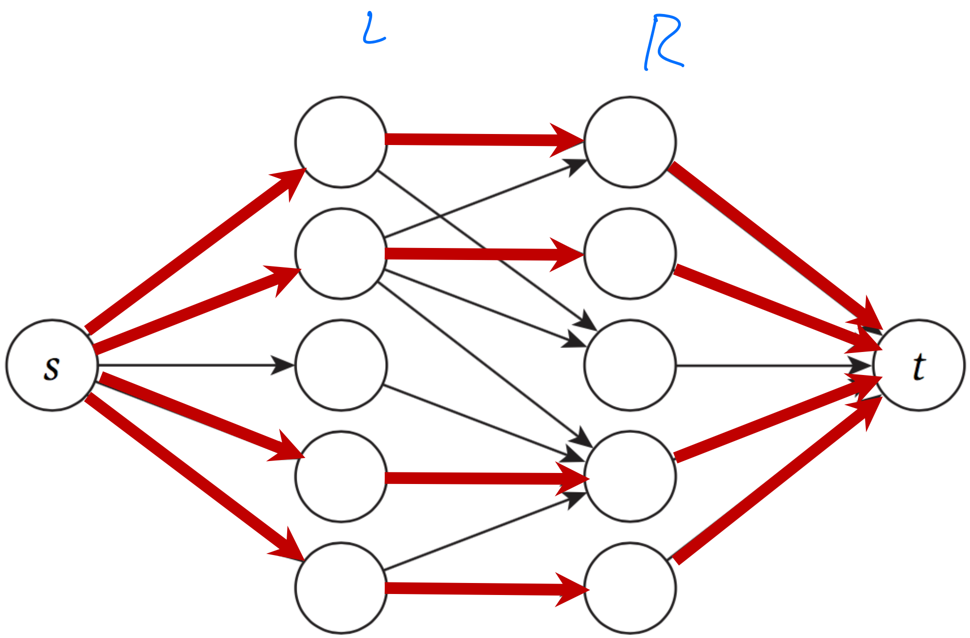
Black arrow means $f'(e) = 0$

Step 3: Transform the Output

valid MBM M for graph G

valid MF f' for network G'



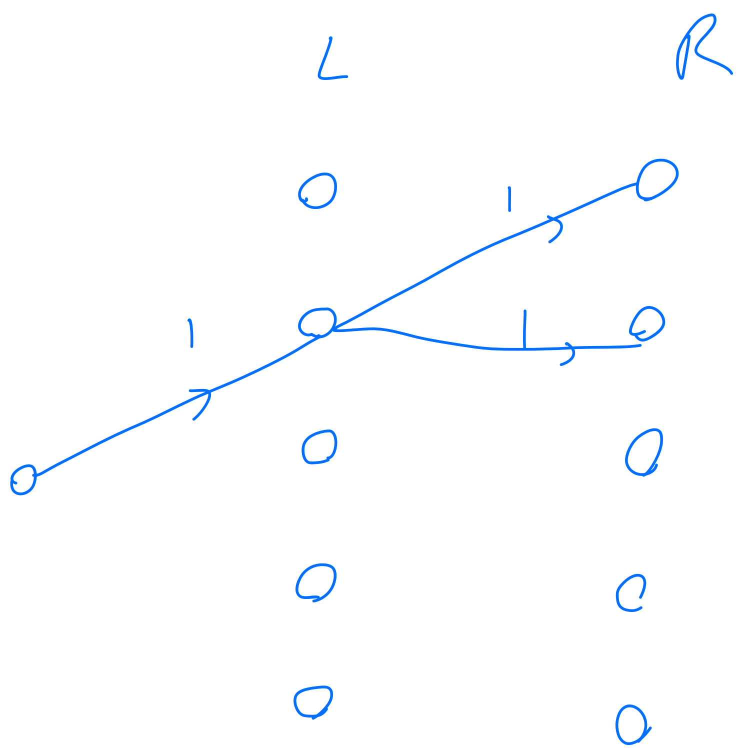


Reduction Recap

- Step 1: Transform the Input
  - Given bipartite graph $G = (L, R, E)$, produce flow network $G' = (V, E, \{c(e)\}, s, t)$ by:
    - orienting edges e from L to R
    - adding a node s with edges from s to every node in L
    - adding a node t with edges from every node in R to t
    - setting all capacities to 1
- Step 2: Receive the Output
  - Find an integer maximum s-t flow f' in G'
- Step 3: Transform the Output
  - Given an integer s-t flow $f'(e)$ let M be the set of edges e going from L to R that have $f'(e) = 1$

Correctness

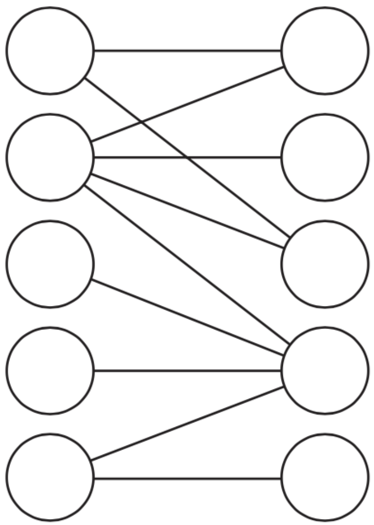
- Need to show:
  - ✓ (1) This algorithm returns a matching
  - (2) This matching is a maximum cardinality matching

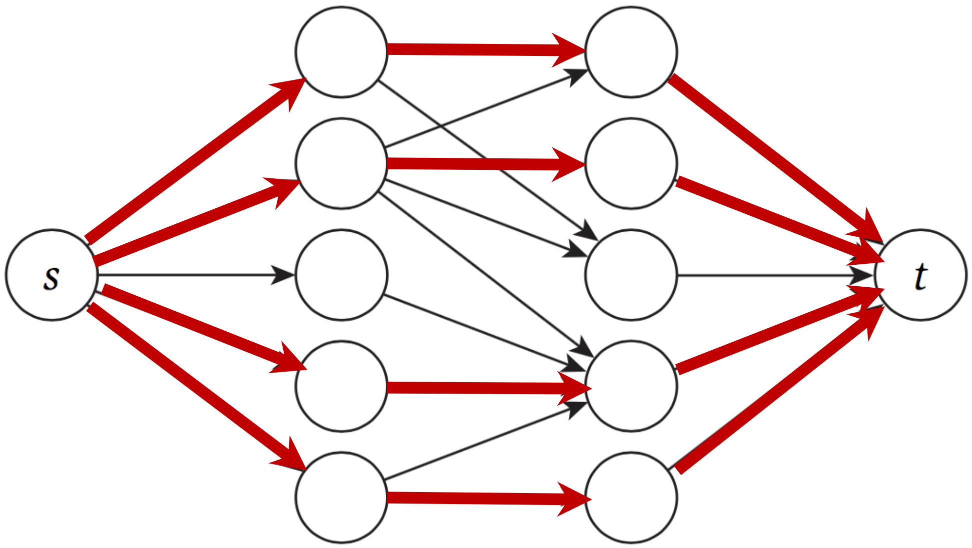


Every vertex in L has incoming capacity 1, therefore incoming flow at most one, therefore outgoing flow at most one $\Rightarrow$ vertices in L have at most one edge in the matching

Correctness

- This algorithm returns a matching



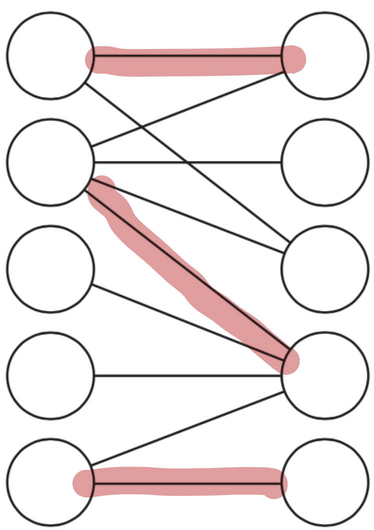


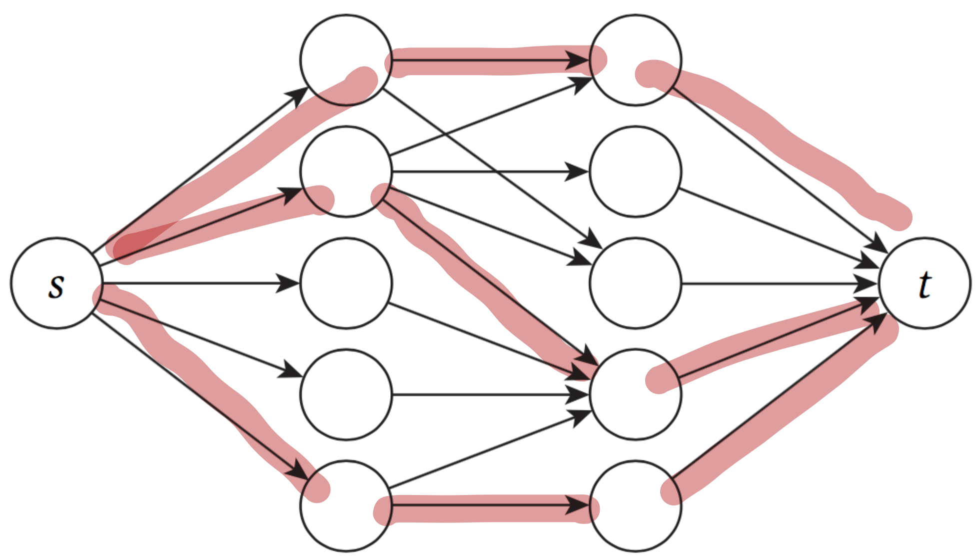
Correctness

- **Claim:** G has a matching of cardinality k if and only if G' has an s-t flow of value k

current fastest known algorithm for MBM runs in $O(m^{1+o(1)})$ time.

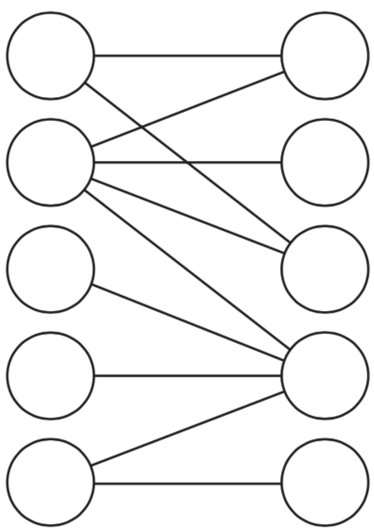
open: find max flow (or MBM) in $O(m \lg^{100} n)$

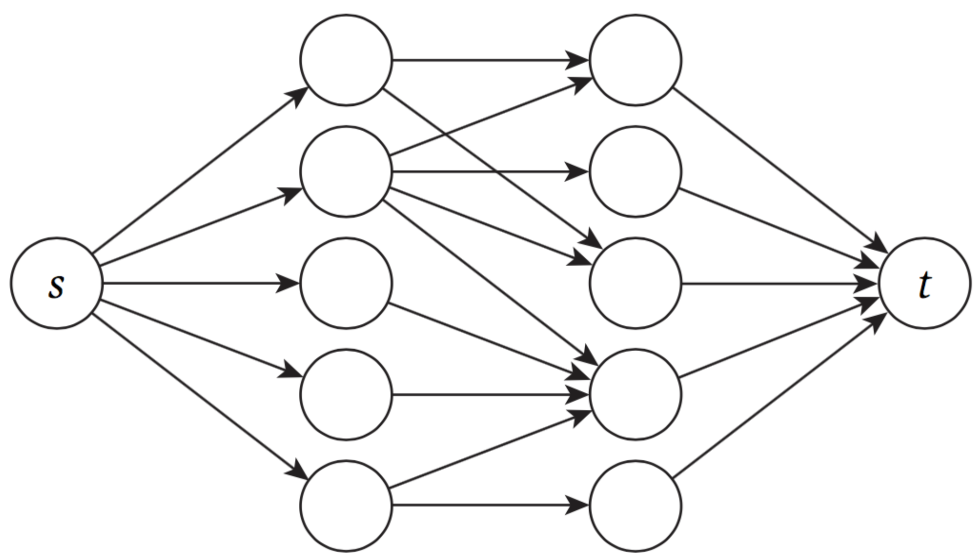




Correctness

- **Claim:** G has a matching of cardinality k if and only if G' has an s-t flow of value k





Running Time

- Need to analyze the time for:
  - (1) Producing G’ given G
  - (2) Finding a maximum flow in G’
  - (3) Producing M given G’

Maximum Bipartite Matching Summary

Solve maximum s-t flow in a graph with $n + 2$ nodes and $m + n$ edges and $c(e) = 1$ in time T

Solve maximum bipartite matching in a graph with n nodes and m edges in time $T + O(m + n)$

- Can solve max bipartite matching in time $O(nm)$ using Ford-Fulkerson
  - Improvement for maximum flow gives improvement for maximum bipartite matching!

Mechanics of Reductions

- **Definition:** a **reduction** is an efficient algorithm that solves problem B using an algorithm that solves problem A as a **black-box**