Graph Algorithms 2

COMP 6651 – Algorithm Design Techniques

Denis Pankratov

Last time

- Basic graph terminology
- Representations of graphs: adjacency matrix and adjacency lists
- BFS, DFS
- Topological sort
- Strongly connected components
- MST: generic algorithm, Kruskal’s and Prim’s algorithms
- Shortest paths: types of problems, single-source shortest paths with non-negative weights, Dijkstra’s algorithm

Shortest paths

- Edge-weighted graph $G = (V, E), w : E \to \mathbb{R}$
- **Weight of path** $p = \langle v_0, v_1, \dots, v_k \rangle$ is

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) = \text{sum of edge weights on } p$$

- **Shortest-path weight** u to v:

$$\delta(u, v) = \begin{cases} \min\left(w(p) : u \overset{p}{\to} v\right) & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

- Can think of weights as representing any measure that accumulates linearly along a path and we wish to minimize it

Variants of shortest paths problems

- **Single-source**
  - Find shortest paths from a given source vertex $s \in V$ to every vertex $v \in V$
- **Single-destination**
  - Find shortest paths to a given destination vertex
- **Single-pair**
  - Find shortest path from u to v. Not known how to do it faster than single-source.
- **All-pairs**
  - Find shortest path from u to v for all $u, v \in V$.

Negative-weight edges

Some algorithms will not work when negative-weight edges are present

Other algorithms will work with negative-weight edges so long as there are no negative-weight cycles reachable from the source

If we have a negative-weight cycle, we can just keep going around it, and get $\delta(s, v) = -\infty$ for all v on the cycle

Some algorithms allow one to detect presence of negative-weight cycles

Some properties of shortest paths

- **Optimal substructure property**

  Any subpath of a shortest path is a shortest path itself

- **No cycles property**

  Shortest paths do not contain cycles without loss of generality

- **Triangle inequality**

  For all $(u, v) \in E$ we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$

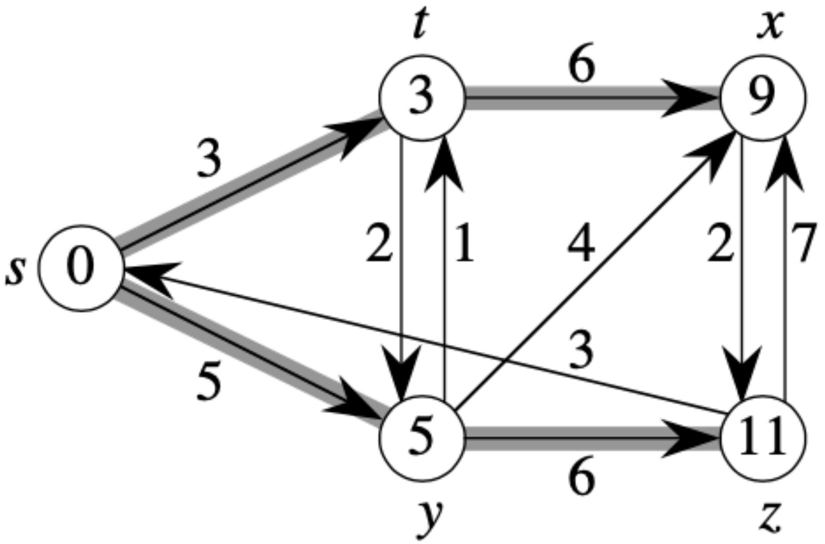
Single-source shortest paths (CLRS 24)

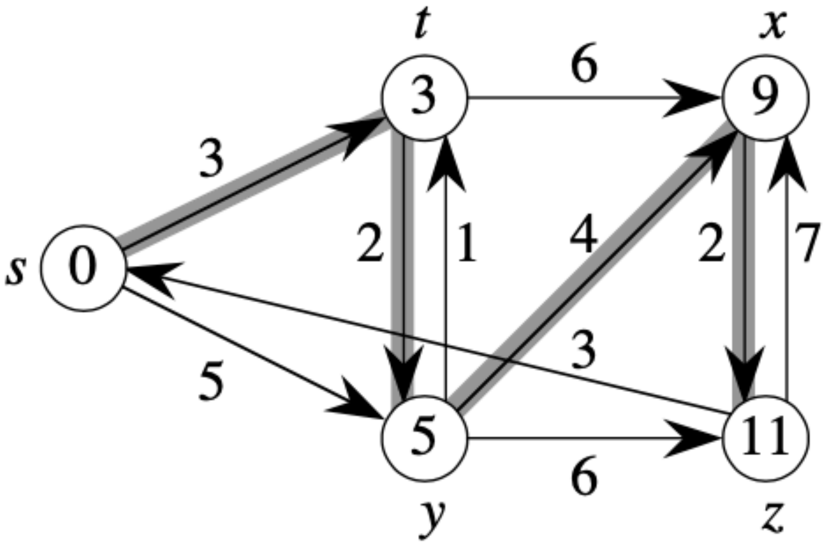
Input: $G = (V, E), w : E \to \mathbb{R}$

source vertex $s \in V$

Output: for each vertex v populate attribute $v.d = \delta(s, v)$

for each vertex v populate attribute $v.\pi$ = predecessor of v on shortest path from s





Generic algorithm

- Initially set $v.d \leftarrow \infty$
- As an algorithm progresses, $v.d$ reduces but satisfies $v.d \geq \delta(s, v)$
- Call $v.d$ a **shortest path estimate**
- Initially set $v.\pi \leftarrow NIL$
- The predecessor graph $\{(v.\pi, v)\}$ forms a tree called **shortest-path tree**
- Shortest path estimate is improved by **relaxing an edge**

Generic algorithm

$InitSingleSource(G = (V, E), s)$

  $\boldsymbol{for}\ v \in V$

    $v.d \leftarrow \infty$

    $v.\pi \leftarrow NIL$

  $s.d \leftarrow 0$

$Relax(u, v, w)$

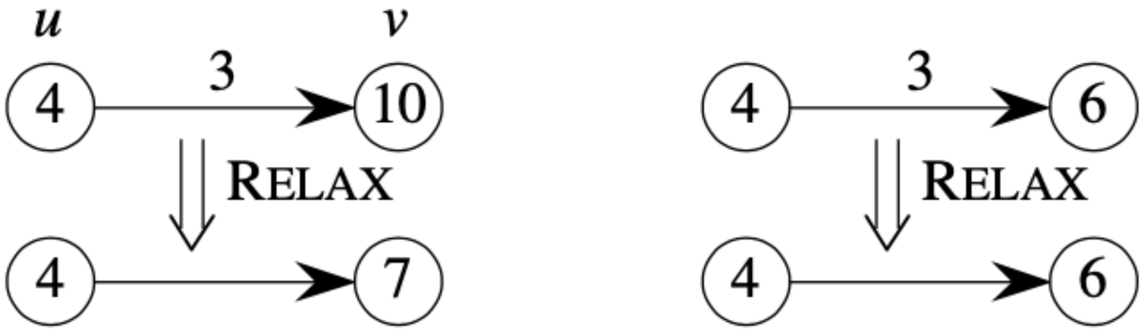
  // (u, v) is an edge

  // w is the weight function

  $if\ v.d > u.d + w(u, v)$

    $v.d \leftarrow u.d + w(u, v)$

    $v.\pi \leftarrow u$



- All single-source shortest paths algorithms we consider
  - Start by calling $InitSingleSource$
  - Then relax edges
- Algorithms differ in the order and number of times edges are relaxed

- **Upper bound property**
  - Always have $v.d \geq \delta(s, v)$ for all $v \in V$
- **Path relaxation property**
  - If $p = \langle v_0, v_1, \dots, v_k \rangle$ is a shortest path from $v_0 = s$ to $v = v_k$. If we relax edges in order $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$ even intermixed with other relaxations then we get $v.d = \delta(s, v)$

Bellman-Ford Algorithm

- Allows negative-weight edges
- Returns $true$ if no negative-weight cycles are reachable from s, $false$ otherwise

$BellmanFord(G = (V, E), w, s)$

$\quad InitSingleSource(G, s)$

$\quad \boldsymbol{for}\ i = 1\ \boldsymbol{to}\ |V| - 1$

$\quad\quad \boldsymbol{for}\ (u, v) \in E$

$\quad\quad\quad Relax(u, v, w)$

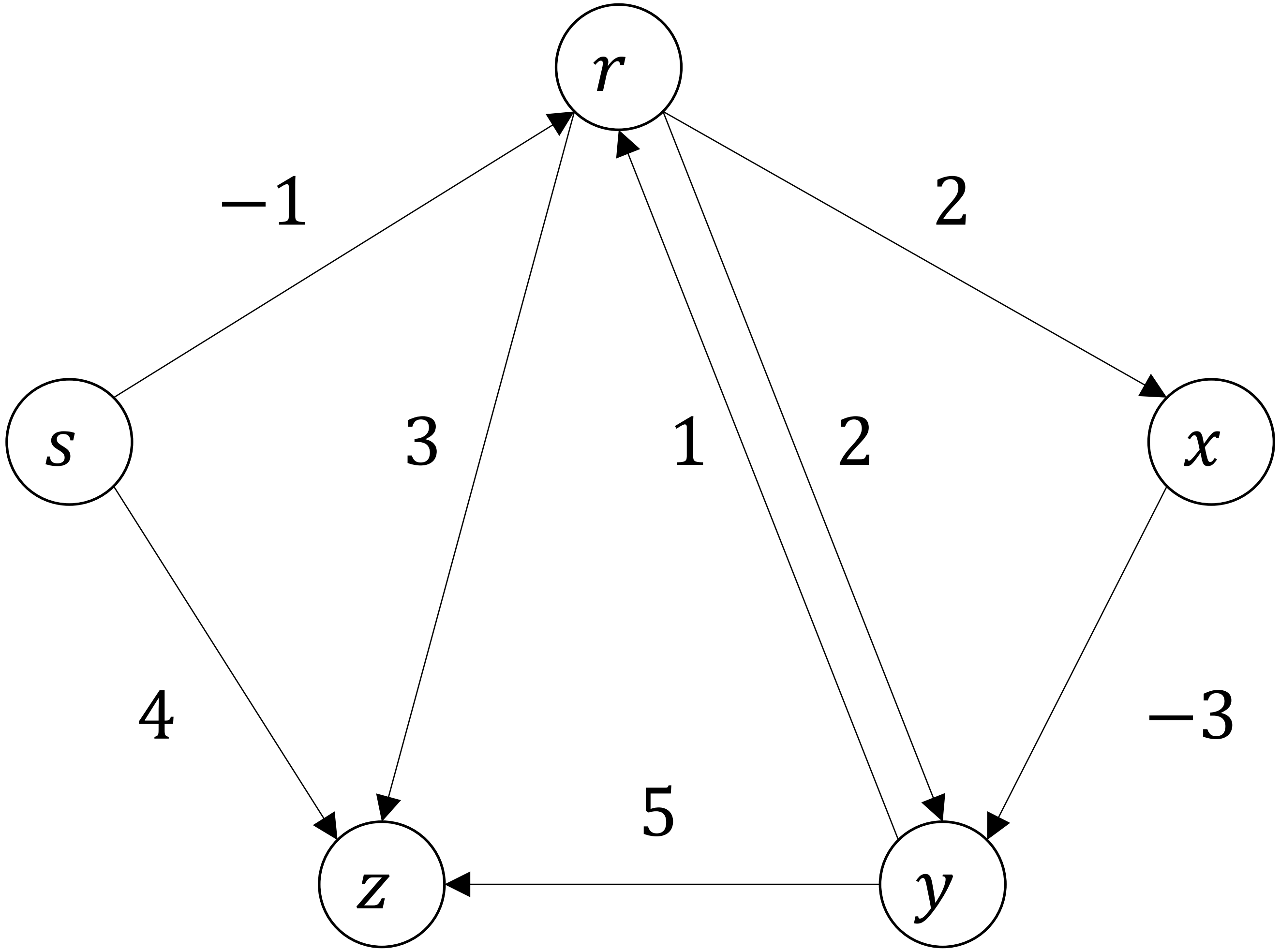
$\quad \boldsymbol{for}\ (u, v) \in E$

$\quad\quad \boldsymbol{if}\ v.d > u.d + w(u, v)$

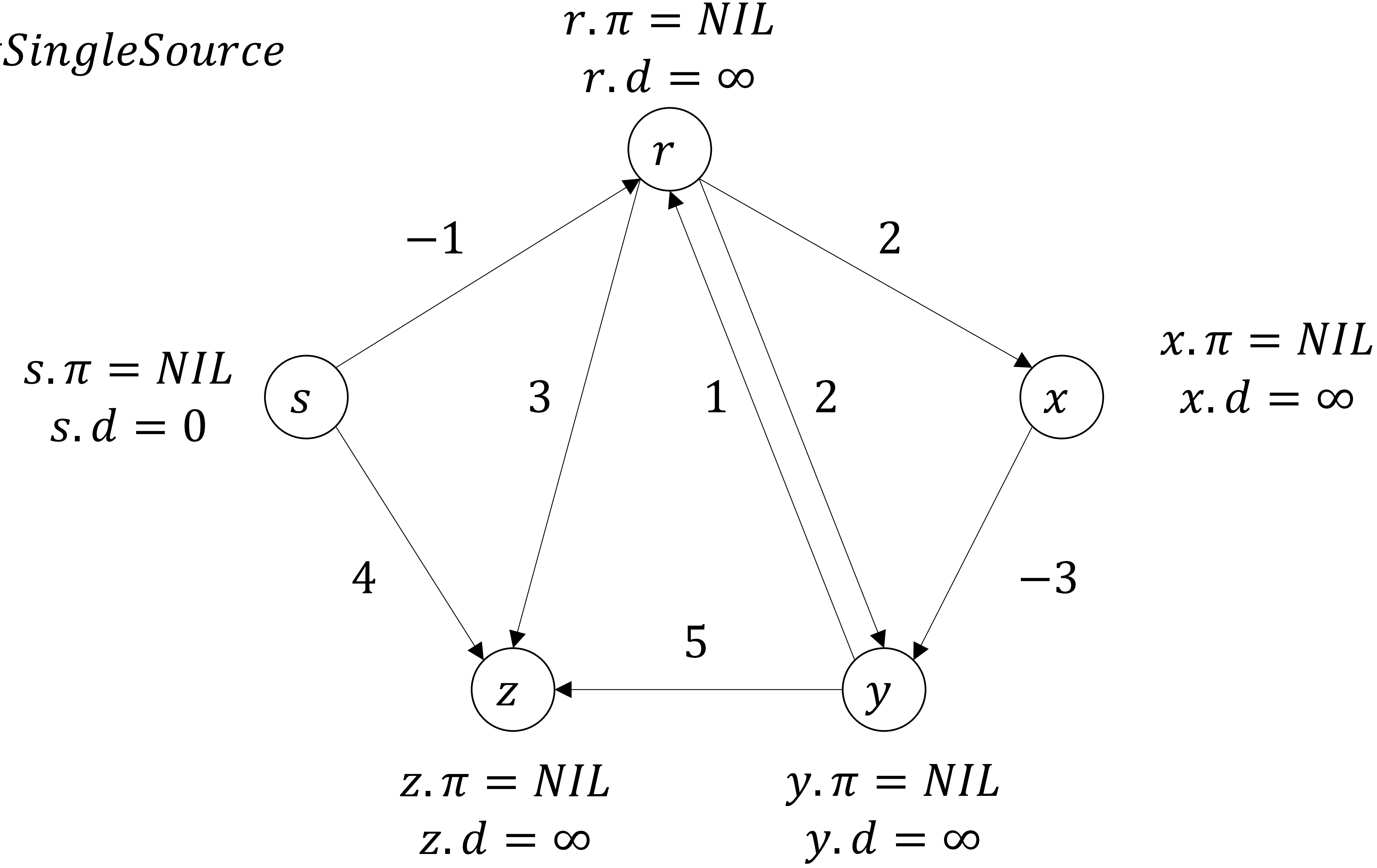
$\quad\quad\quad \boldsymbol{return}\ false$

$\quad \boldsymbol{return}\ true$

Main idea: relax each edge $|V| - 1$ times
Running time: $\Theta(|V| \cdot |E|)$

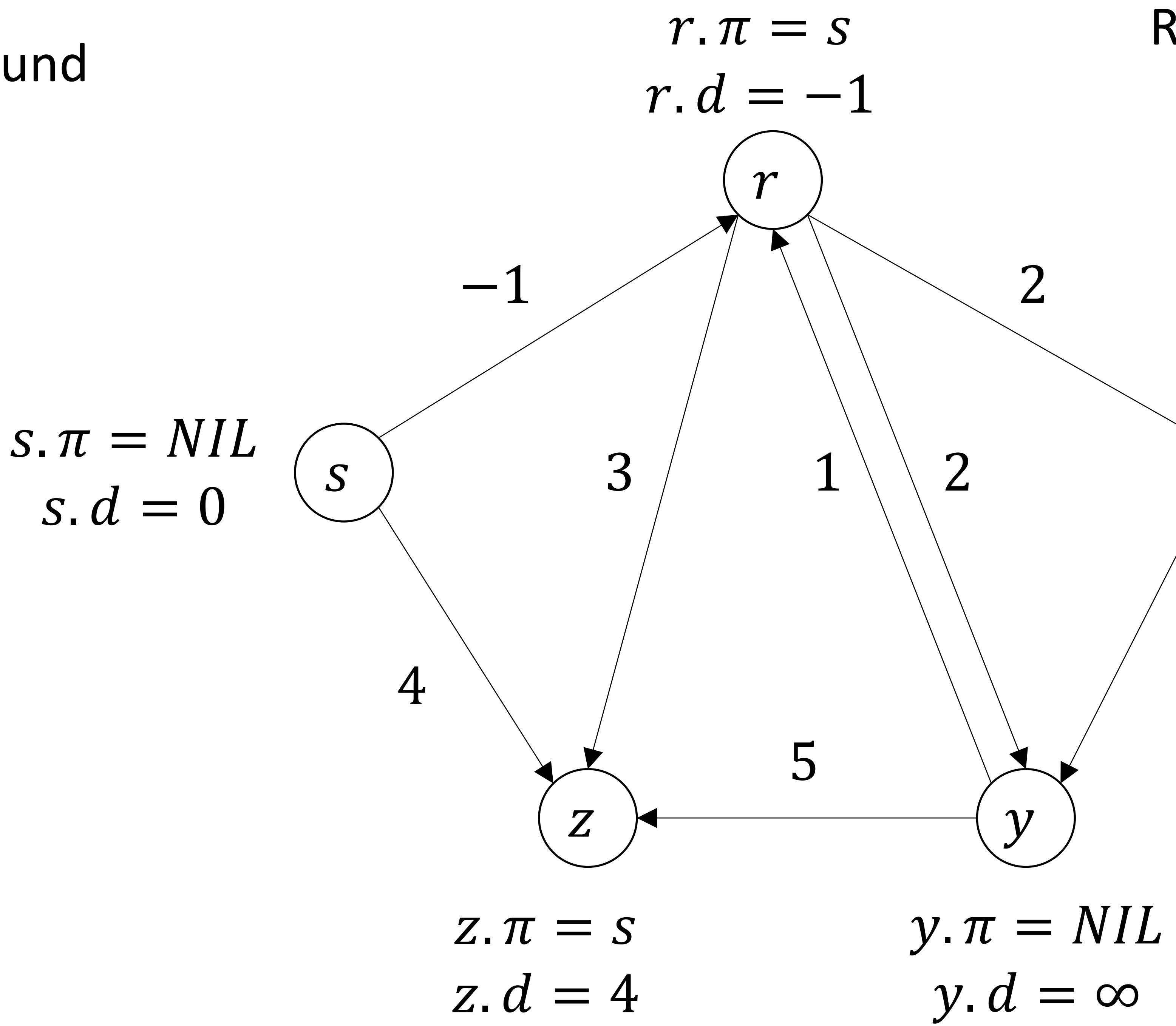

r
s
x
z
y
−1
2
3
1
2
4
−3
5

InitSingleSource



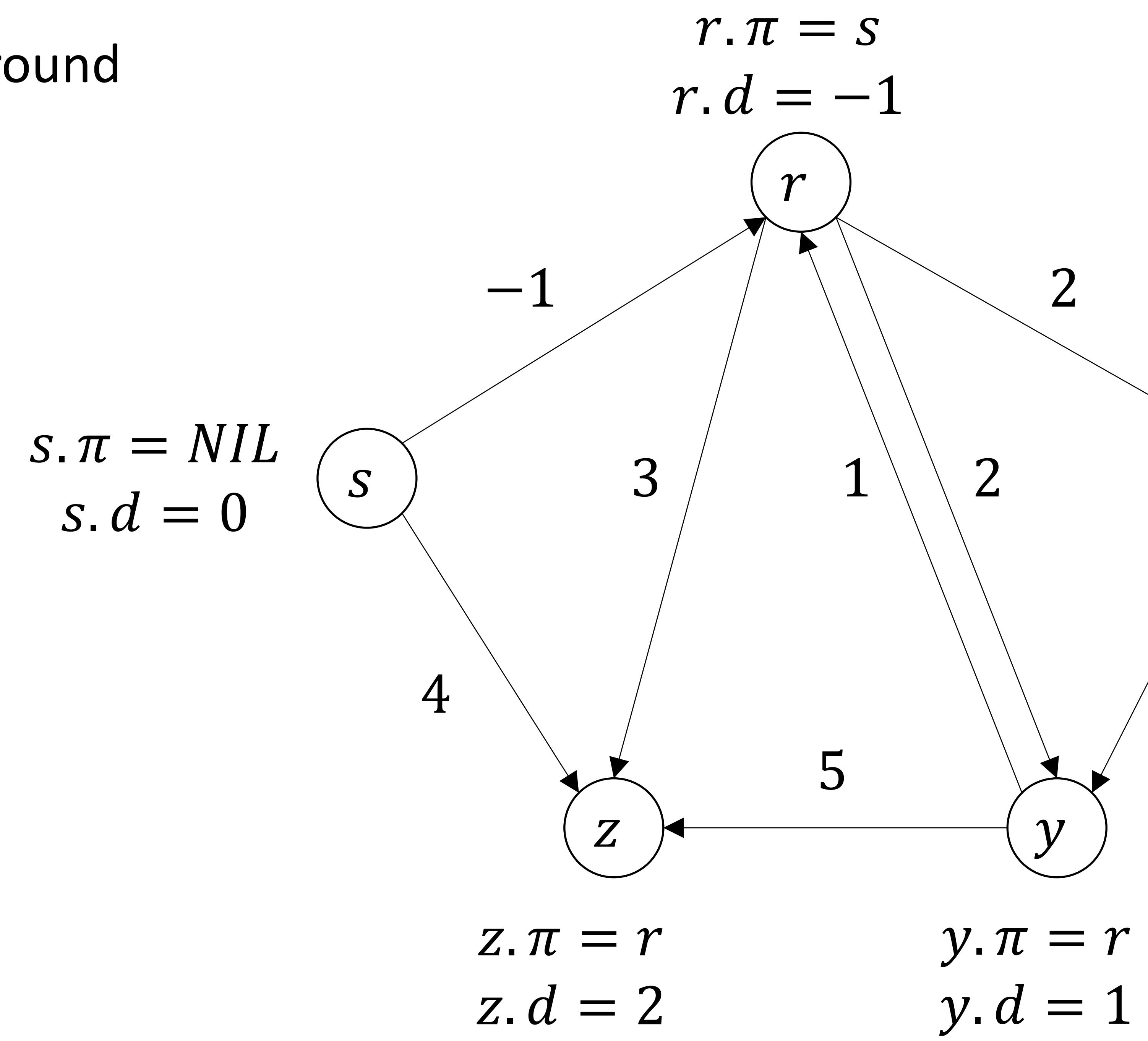
1st round

Relax edges in order:
$(y, r), (y, z), (x, y)$
$(r, x), (r, y), (r, z)$
$(s, r), (s, z)$



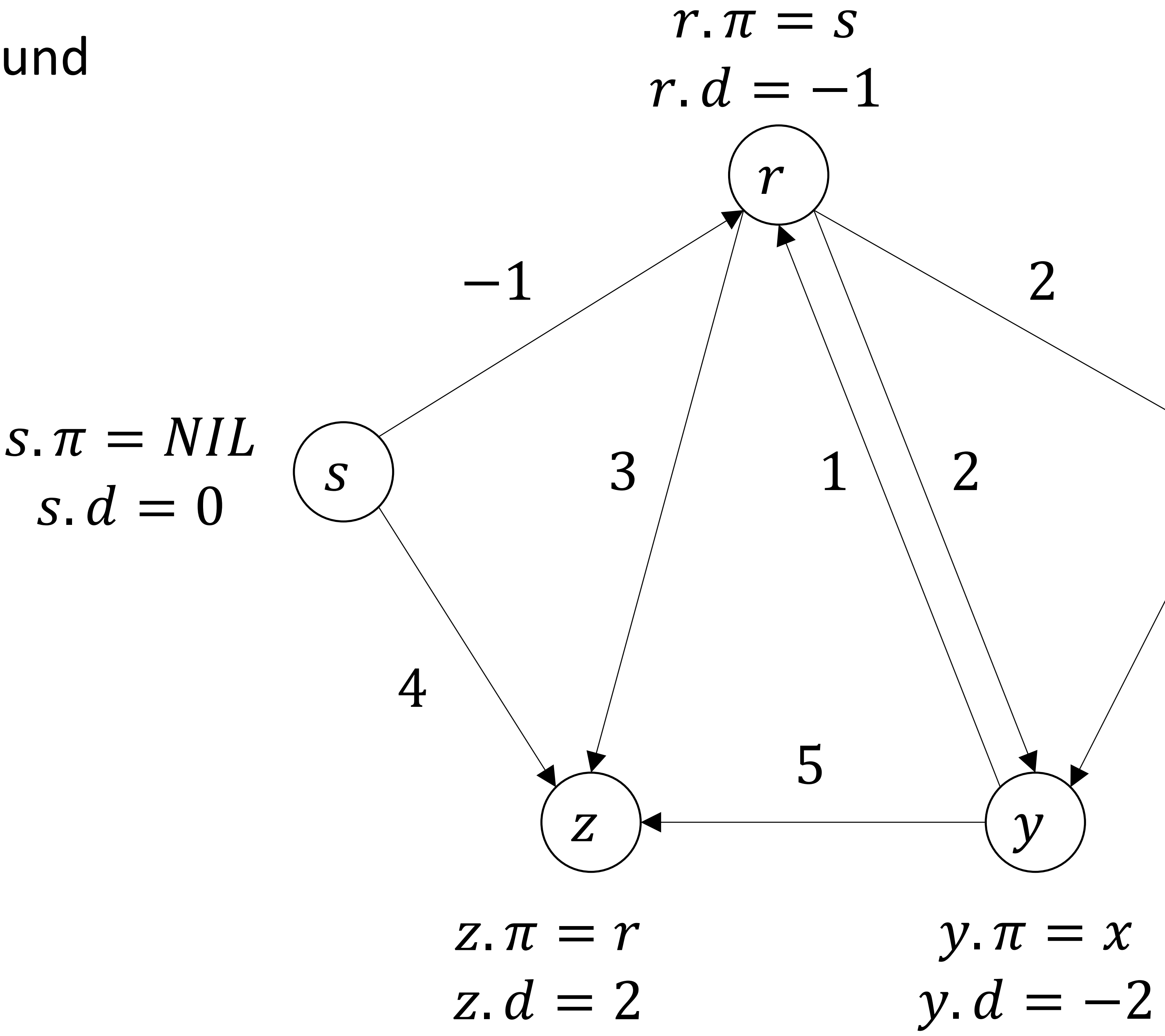
2nd round

Relax edges in order:
$(y,r),(y,z),(x,y)$
$(r,x),(r,y),(r,z)$
$(s,r),(s,z)$



3rd round

Relax edges in order:
$(y, r), (y, z), (x, y)$
$(r, x), (r, y), (r, z)$
$(s, r), (s, z)$



Bellman-Ford algorithm solves single-source shortest paths correctly

Proof

Suppose $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to $v = v_k$

By the **no cycles property**, p is acyclic and therefore has $\leq |V| - 1$ edges

Each iteration of the outer $\boldsymbol{for}$ loop relaxes all edges:

- First iteration guarantees to relax (v_0, v_1)
- Second iteration guarantees to relax (v_1, v_2)
- kth iteration guarantees to relax (v_{k-1}, v_k)

By the **path relaxation property**, $v.d = \delta(s, v)$

proof continued

What about $true/false$ values returned by the algorithm?

1. Suppose there is no negative-weight cycle reachable from s

   At termination for all $(u, v) \in E$

   $$v.d = \delta(s, v) \leq \delta(s, u) + w(u, v) \text{ (by triangle inequality)}$$
   $$= u.d + w(u, v)$$

2. Suppose there is a negative-weight cycle reachable from s

   Let the cycle be $\langle v_0, v_1, \dots, v_k \rangle$

   Assume for contradiction that Bellman-Ford returns $true$

$$\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} \left(v_{i-1}.d + w(v_{i-1}, v_i) \right) = \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Observe that $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$, therefore $\sum_{i=1}^{k} w(v_{i-1}, v_i) \geq 0$

Contradiction.

All-pairs shortest paths (CLRS 25)

Input: $G = (V, E), w : E \to \mathbb{R}$

Output: $|V|\times|V|$ matrix $D = \left(d_{ij}\right)$ of shortest distances $d_{ij} = \delta(i, j)$

- Could run Bellman-Ford from each vertex: running time is $O(|V|^2|E|)$ which is $O(|V|^4)$ for dense graphs, i.e., $|E| = \Theta(|V|^2)$
- If weights are non-negative, could run Dijkstra's from each vertex: running time is $O(|V| \cdot |E| \log|V|)$ with binary heap which is $O(|V|^3 \log|V|)$ if dense
- We can achieve $O(|V|^3)$ in all cases with no fancy data structures

Shortest paths and matrix multiplication

Record input weights in a matrix $W = \left(w_{ij}\right)$

$$w_{ij} = \begin{cases} 0 & i = j \\ w(i,j) & i \neq j, (i,j) \in E \\ \infty & i \neq j, (i,j) \notin E \end{cases}$$

This matrix has interpretation:

$w_{ij} =$ weight of a shortest path from i to j that uses at most 1 edge

To compute weights of shortest paths that use 2 edges

Shortest path from i to j using 2 edges:

- either uses a shortest path from i to j with at most 1 edge
- or uses an intermediate node k and uses shortest path from i to k with at most 1 edge and shortest path from k to j with at most 1 edge

Denote the resulting matrix by $W^{(2)} = \left(w_{ij}^{(2)}\right)$

Can be computed as

$$w_{ij}^{(2)} = \min(w_{ij}, \min_k(w_{ik} + w_{kj}))$$

Note: this is like matrix-multiplication with $\cdot$ replaced by $+$ and $\sum$ replaced by $\min$

Similarly can compute shortest paths that use at most $3, 4, \ldots$ edges

By the no cycle property, it suffices to compute shortest paths that use at most $|V| - 1$ edges

Need to compute $W^{(|V|-1)}$. First attempt:

$APSP - MM(W)$

$W^{(1)} \leftarrow W$

$\boldsymbol{for}\ p = 2\ to\ |V| - 1$

$W^{(p)} \leftarrow$ new $|V|\times|V|$ matrix, initially filled with ∞

$\boldsymbol{for}\ i \in V$

$\boldsymbol{for}\ j \in V$

$\boldsymbol{for}\ k \in V$

$W_{ij}^{(p)} \leftarrow \min\left(W_{ij}^{(p)}, W_{ik}^{(p-1)} + W_{kj}^{(p-1)}\right)$

$\boldsymbol{return}\ W^{(|V|-1)}$

Running time: $\Theta(|V|^4)$

Can speed it up with repeated squaring

Can compute $W^{(1)}, W^{(2)}, W^{(4)}, W^{(8)}, W^{(16)}, \dots, W^{(m)}$

Until $m > |V| - 1$

It's okay to overshoot, since $W^{(m)}$ doesn't change after $m \geq |V| - 1$

Since m is doubled every time, the outer loop is executed at most $\log |V|$ times

Overall running time becomes $O(|V|^3 \log|V|)$

Can speed it up with repeated squaring

$Faster - APSP - MM(W)$

$\quad W^{(1)} \leftarrow W$

$\quad m \leftarrow 1$

$\quad \boldsymbol{while}\ m < |V| - 1$

$\quad\quad W^{(2m)} \leftarrow$ new $|V| \times |V|$ matrix, initially filled with ∞

$\quad\quad \boldsymbol{for}\ i \in V$

$\quad\quad\quad \boldsymbol{for}\ j \in V$

$\quad\quad\quad\quad \boldsymbol{for}\ k \in V$

$\quad\quad\quad\quad\quad W_{ij}^{(2m)} \leftarrow \min\left(W_{ij}^{(2m)}, W_{ik}^{(m)} + W_{kj}^{(m)}\right)$

$\quad\quad m \leftarrow 2m$

$\quad \boldsymbol{return}\ W^{(m)}$

Floyd-Warshall algorithm

A different dynamic programming algorithm

Assume $V = [n]$ for simplicity

For a path $p = \langle v_0, v_1, \dots, v_k \rangle$ an **intermediate vertex** is any vertex except for v_0 and v_k

Define

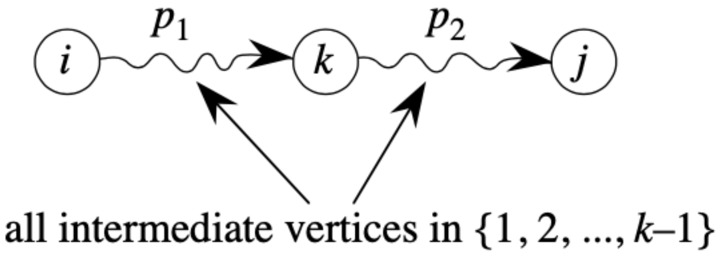
$$d_{ij}^{(k)} = \text{shortest path weight of any path } i \to j \text{ with all intermediate vertices in } \{1, 2, \dots k\} \subseteq V$$

This is the semantic array for the DP

The overall answer is $D^{(n)} = \left(d_{ij}^{(n)}\right)$

Consider a shortest path $i \overset{p}{\rightarrow} j$ with all intermediate vertices in $\{1, 2, \ldots, k\}$

- If k is not an intermediate vertex then all intermediate vertices are in $\{1, 2, \ldots, k-1\}$
- If k is an intermediate vertex:



Computational array:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & k \geq 1 \end{cases}$$

Pseudocode for Floyd-Warshall

$Floyd - Warshall(W, n)$

$\quad D^{(0)} \leftarrow W$

$\quad \boldsymbol{for}\ k = 1\ \boldsymbol{to}\ |V|$

$\quad\quad D^{(k)} \leftarrow$ new $|V| \times |V|$ matrix

$\quad\quad \boldsymbol{for}\ i = 1\ \boldsymbol{to}\ |V|$

$\quad\quad\quad \boldsymbol{for}\ j = 1\ \boldsymbol{to}\ |V|$

$\quad\quad\quad\quad d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$

$\quad \boldsymbol{return}\ D^{(n)}$

Running time: $\Theta(|V|^3)$

Maximum flow

$G = (V, E)$ is directed

Source vertex s and **sink** vertex t

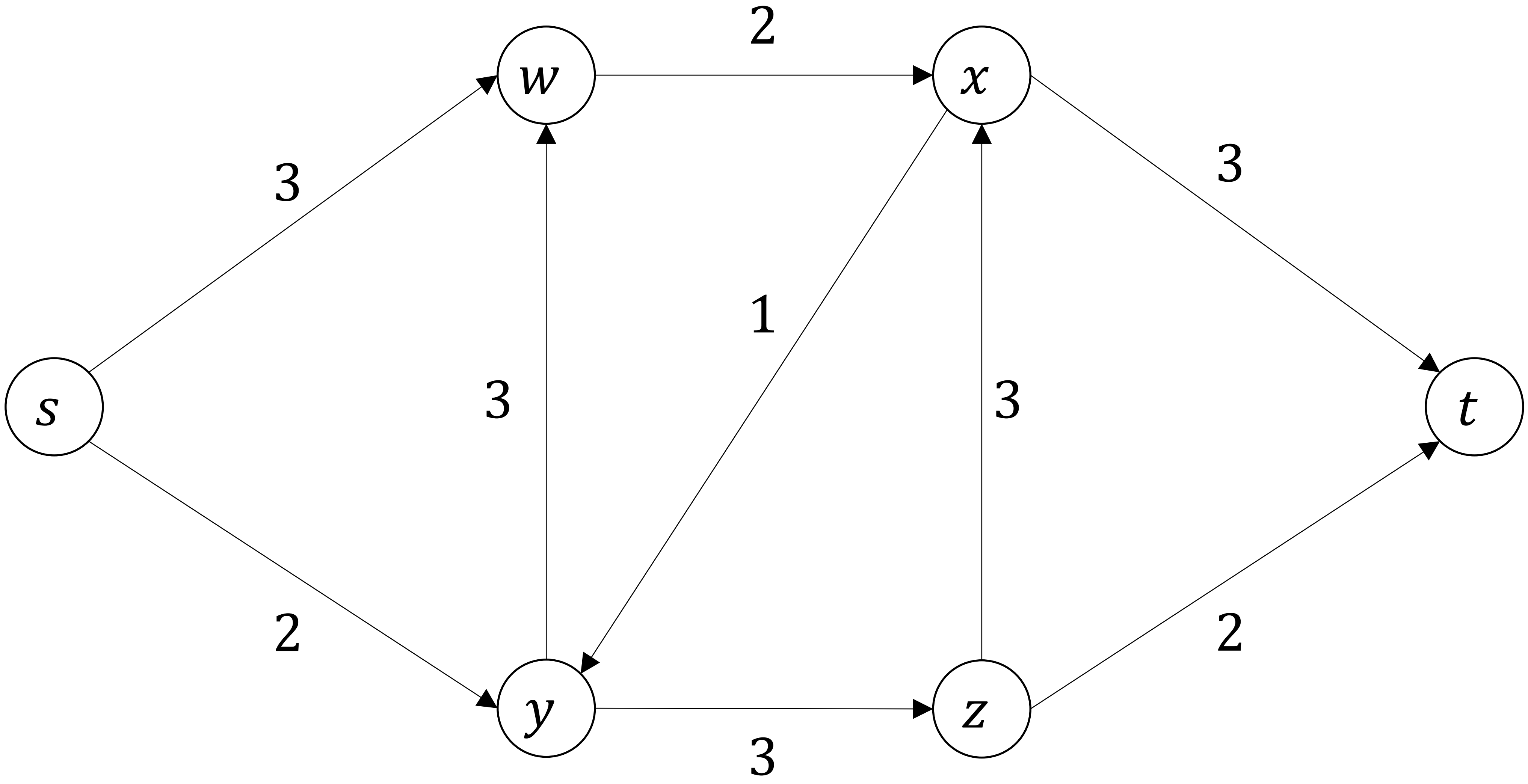
Each edge (u, v) has capacity $c(u, v) \geq 0$

If $(u, v) \notin E$ then $c(u, v) = 0$

If $(u, v) \in E$ then $(v, u) \notin E$ (can work around this restriction)

Assume for all $v \in V$ there exists a path from s to v and from v to t

The goal is to send maximum amount of **flow** from s to t



s
w
x
y
z
t
3
2
2
3
1
3
3
3
2
3

Flow

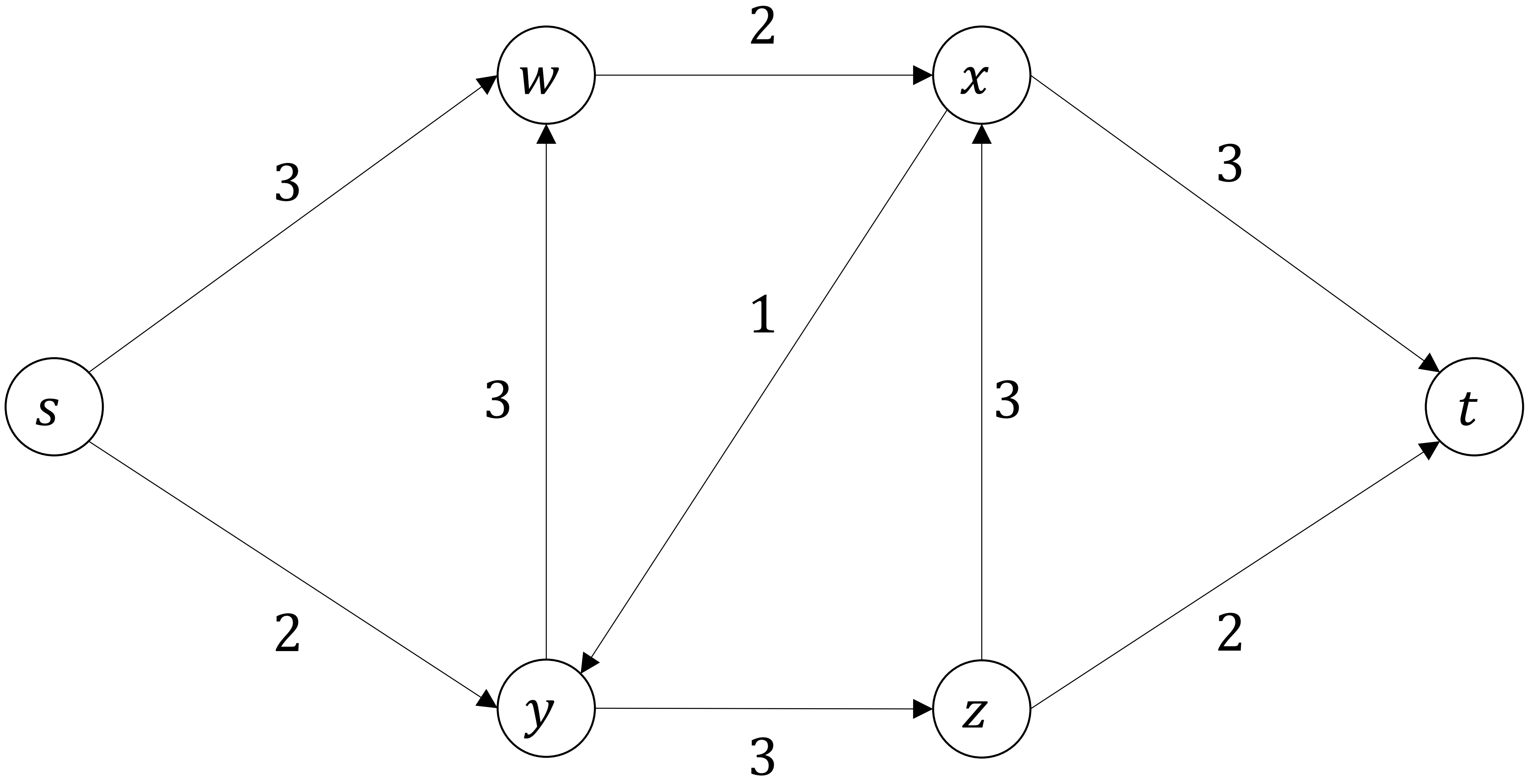
A function $f : V \times V \to \mathbb{R}$

Capacity constraint: for all $u, v \in V$

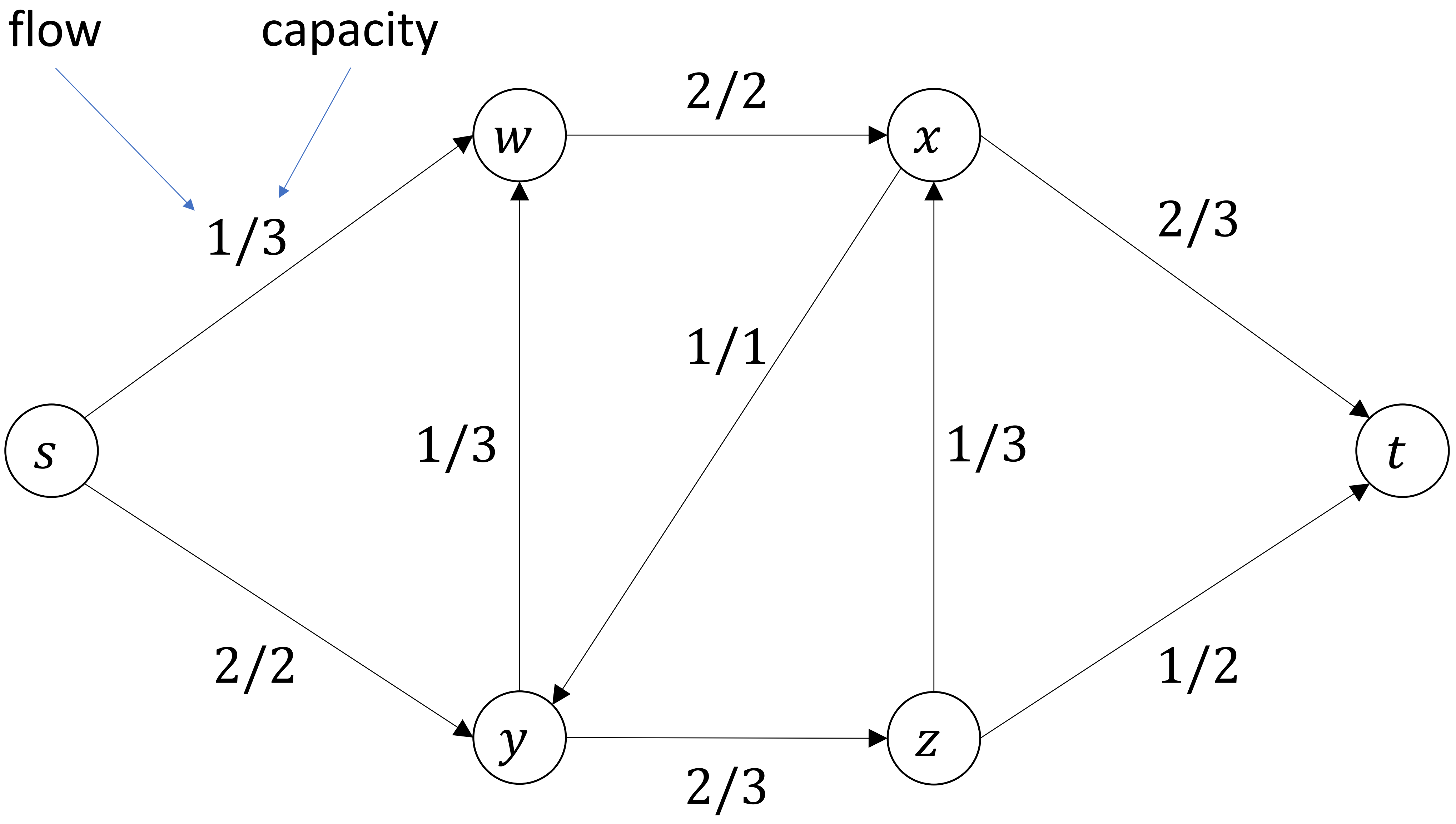
$$0 \leq f(u,v) \leq c(u,v)$$

Flow conservation: for all $u \in V - \{s,t\}$

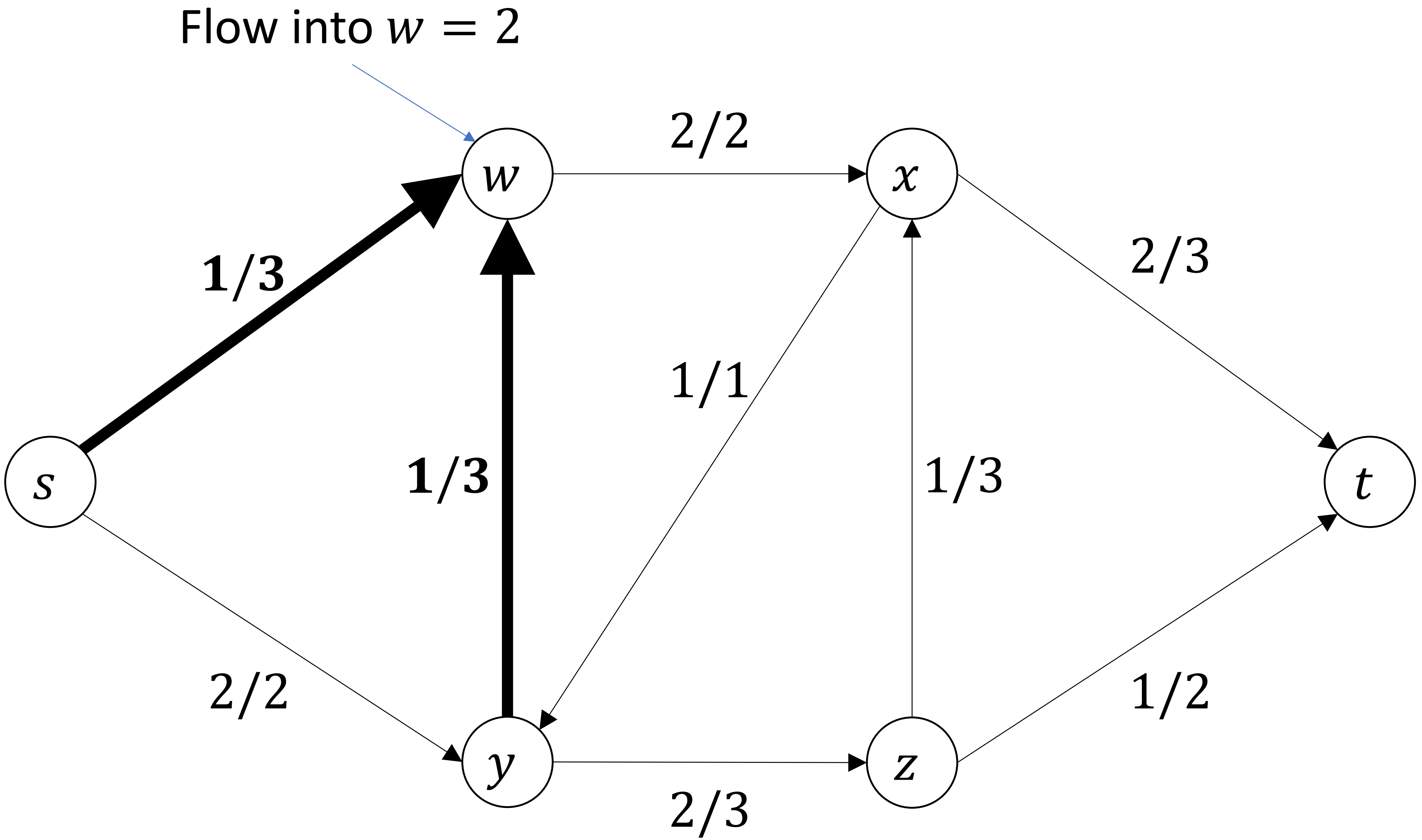
$$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$$



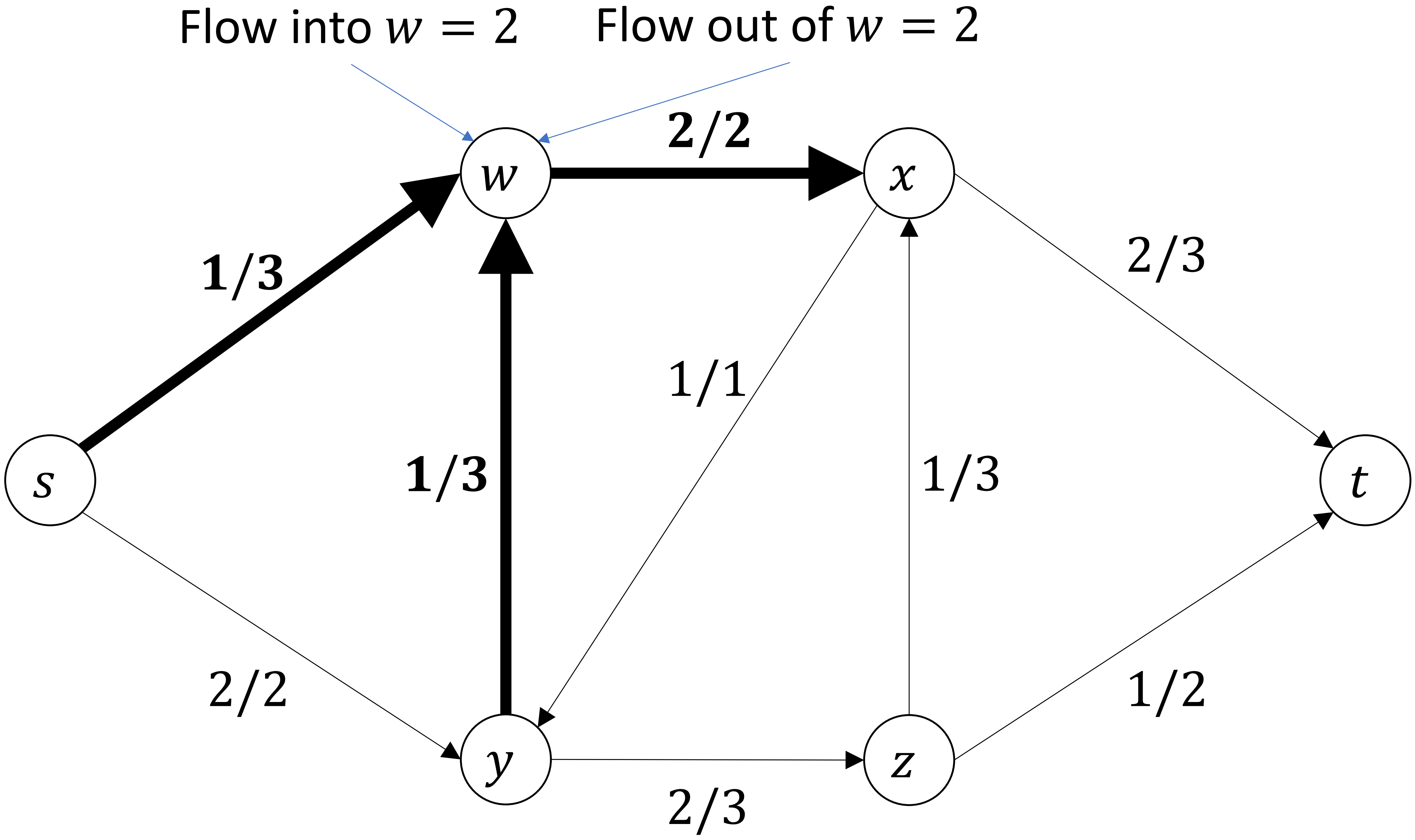
s
w
x
y
z
t
3
2
3
3
1
3
2
3
2



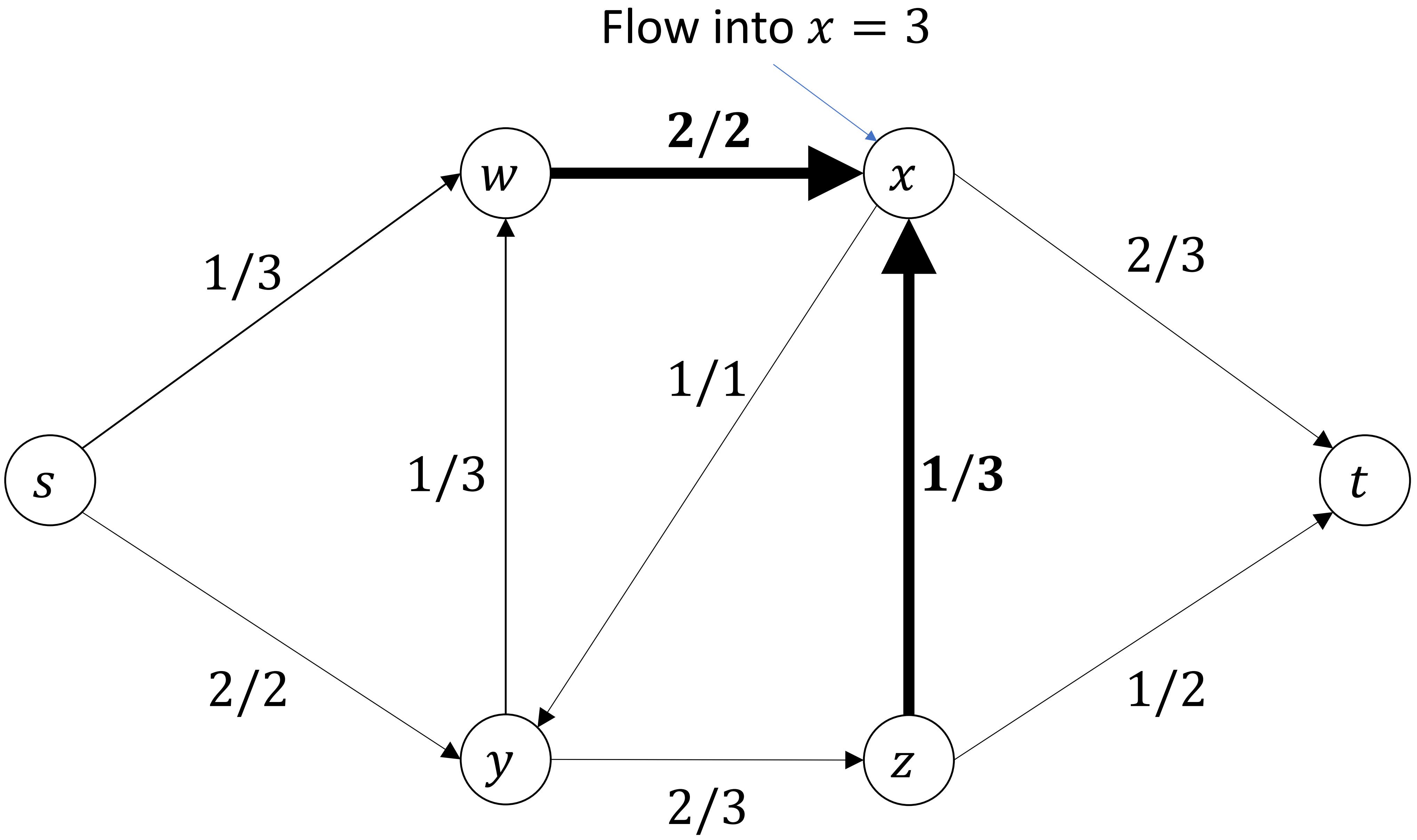
flow
capacity
1/3
2/2
2/2
1/3
1/1
1/3
2/3
2/3
1/2
s
w
x
y
z
t



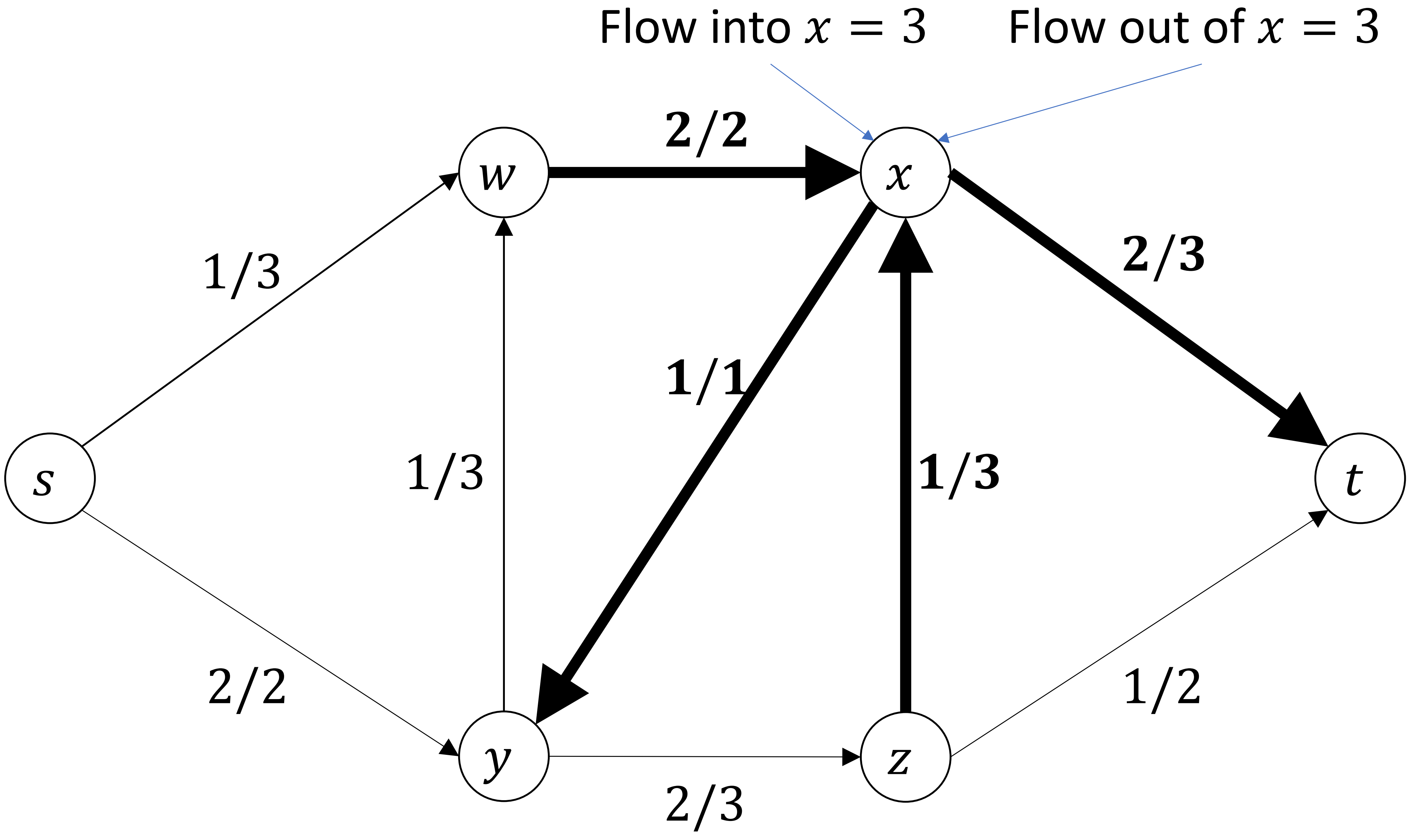
Flow into $w = 2$
w
x
s
t
y
z
2/2
1/3
1/3
1/1
2/3
1/3
2/2
1/2
2/3

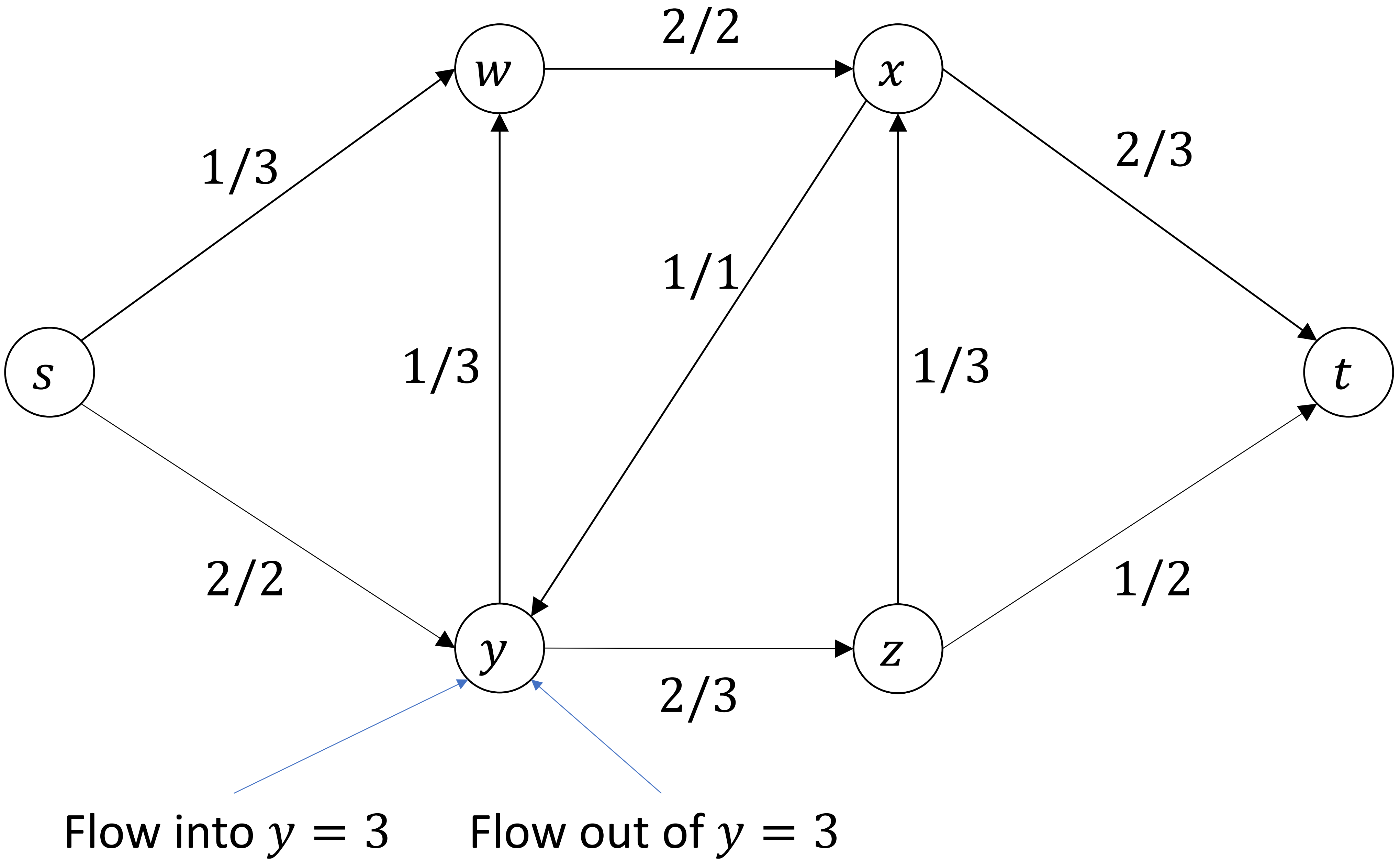


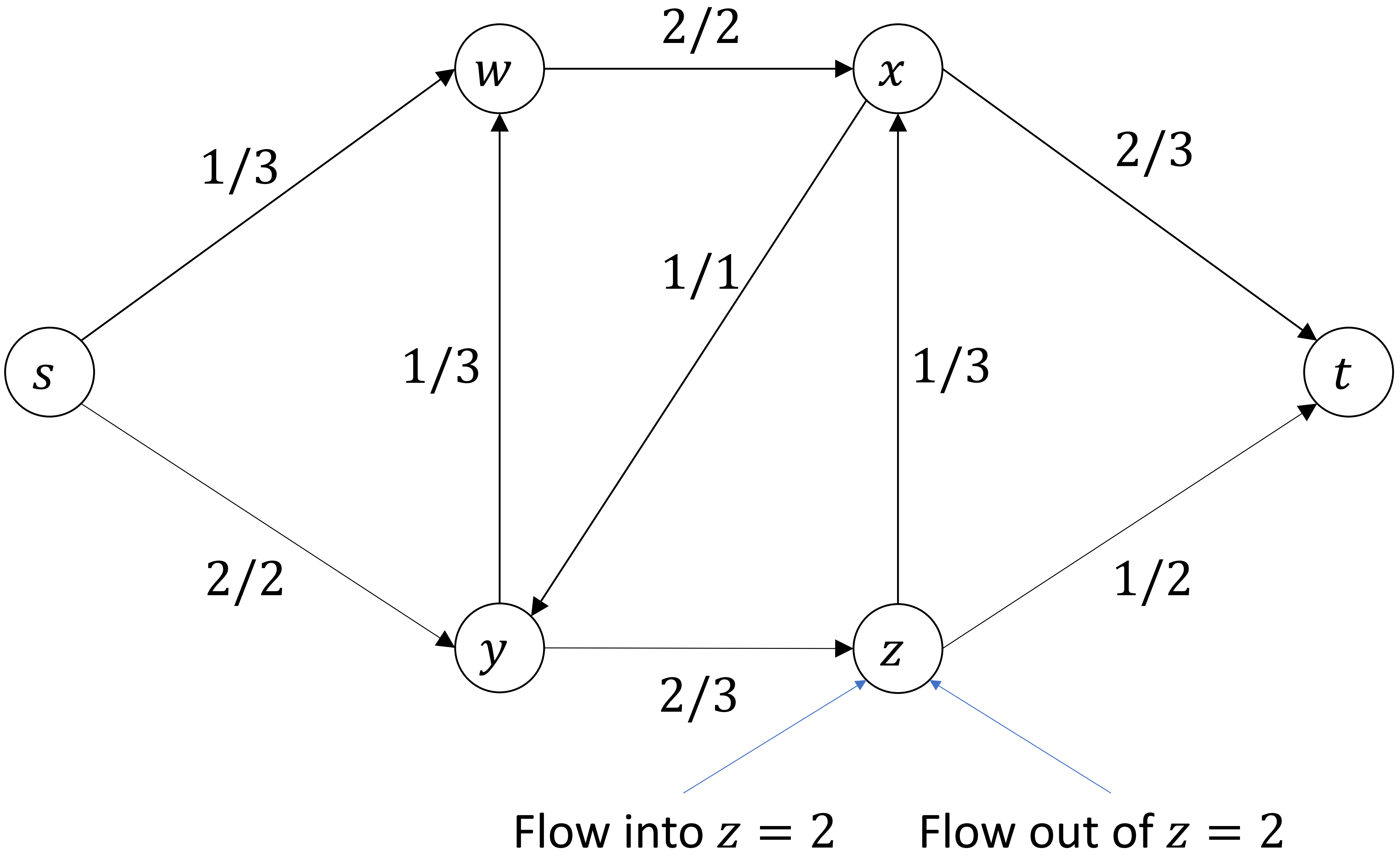
Flow into $w = 2$
Flow out of $w = 2$
w
x
s
t
y
z
2/2
1/3
1/3
2/3
1/1
1/3
2/2
1/2
2/3



Flow into $x = 3$
w
x
s
t
y
z
2/2
1/3
2/3
1/1
1/3
1/3
2/2
1/2
2/3


Flow into $x = 3$
Flow out of $x = 3$
w
x
2/2
2/3
1/3
1/1
s
1/3
1/3
t
2/2
1/2
y
z
2/3


w
x
s
t
y
z
2/2
1/3
2/3
1/1
1/3
1/3
2/2
1/2
2/3
Flow into $y = 3$
Flow out of $y = 3$



2/2
w
x
1/3
2/3
1/1
s
1/3
1/3
t
2/2
1/2
y
z
2/3
Flow into $z = 2$
Flow out of $z = 2$

Value of flow and maximum flow problem

Value of flow $= | f | = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$

$=$ flow out of source $-$ flow into source

Input: $G = (V, E), s \in V, t \in V, c : V \times V \rightarrow \mathbb{R}_{\geq 0}$

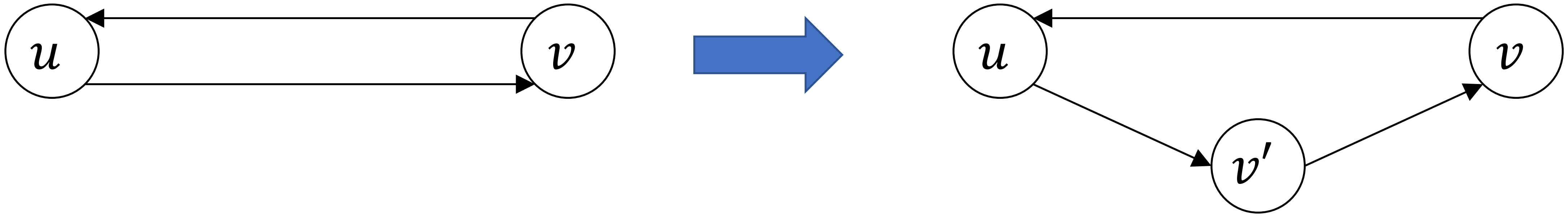
Output: flow $f : V \times V \rightarrow \mathbb{R}$ such that $| f |$ is maximized

Antiparallel edges

Edges (u, v) and (v, u) are called antiparallel

If $G = (V, E)$ contains antiparallel edges we can modify it into an equivalent graph (preserving maximum flow value) without antiparallel edges:

Introduce new node v'



Cuts

A cut (S, T) of flow network G is a partition of V into S and $T = V - S$ such that $s \in S$ and $t \in T$

- Similar to a cut in MSTs, except G is directed and we require $s \in S$ and $t \in T$

Given $f : V \times V \to \mathbb{R}$ the **net flow across cut** (S, T) is

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u)$$

Capacity of the cut is

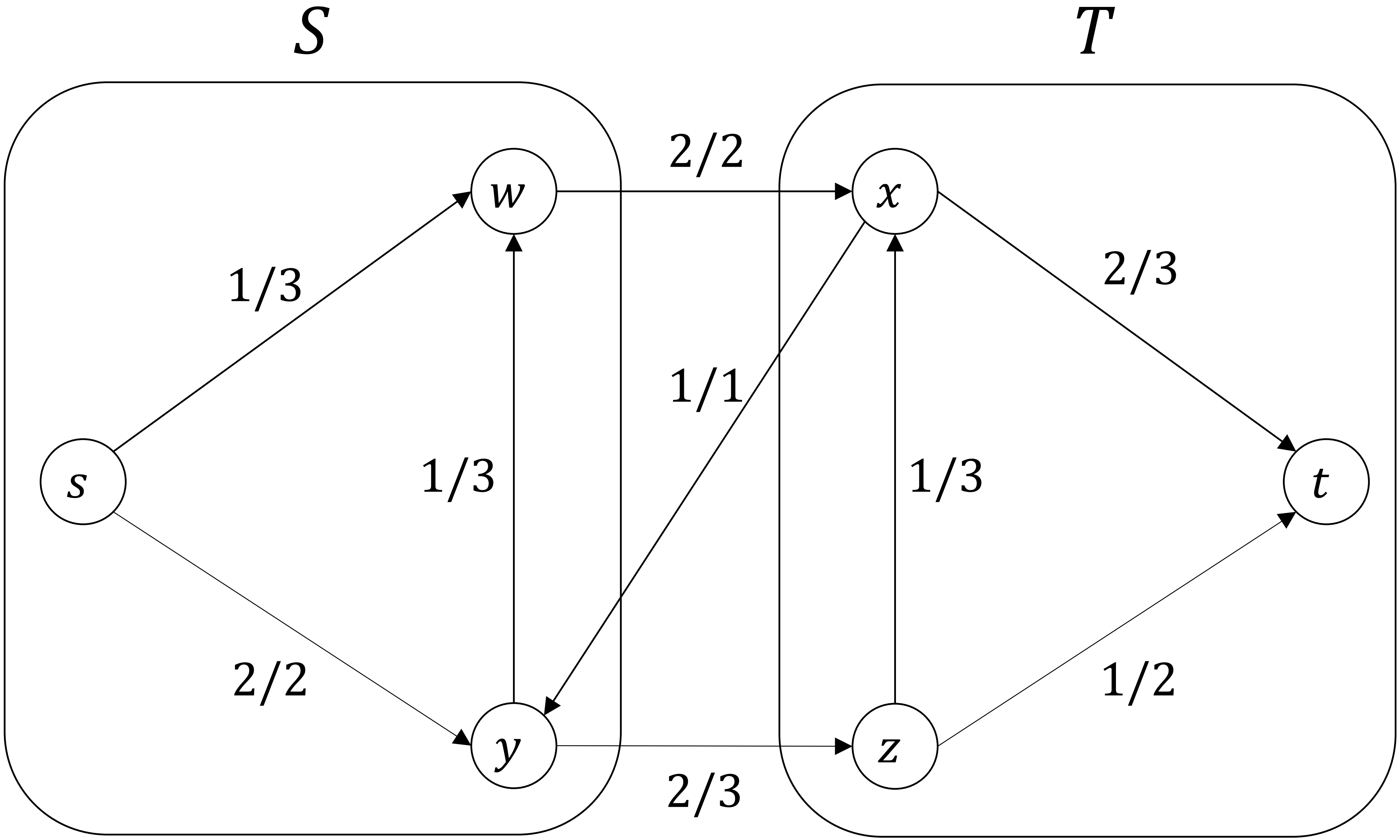
$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

Minimum cut is a cut whose capacity is minimum over all cuts

Asymmetry between flow and cut

Given cut (S, T) note that

- for net flow take flow of edges from S to T and subtract flow of edges from T to S
- for capacity of the cut take capacity of edges **only going from $\boldsymbol{S}$ to $\boldsymbol{T}$**



$$f(S,T) = f(w,x) + f(y,z) - f(x,y) = 3 \qquad c(S,T) = c(w,x) + c(y,z) = 5$$

Lemma: For any cut (S, T) we have

$$f(S,T) = |f|$$

Proof:

We need to show that

$$f(S,T) = \sum_{u \in S, v \in T} f(u,v) - \sum_{u \in S, v \in T} f(v,u)$$

is equal to

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$$

Conservation constraint: $u \in S - \{s\}$: $\sum_{v \in V} f(u,v) - \sum_{v \in V} f(v,u) = 0$

Therefore:

$$\sum_{u \in S - \{s\}} \left(\sum_{v \in V} f(u,v) - \sum_{v \in V} f(v,u) \right) = 0$$

Add it to the definition of $|f|$:

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) + \sum_{u \in S - \{s\}} \left(\sum_{v \in V} f(u,v) - \sum_{v \in V} f(v,u) \right)$$

$$= \sum_{u \in S, v \in V} f(u,v) - \sum_{u \in S, v \in V} f(v,u)$$

$$= \sum_{u \in S, v \in T} f(u,v) - \sum_{u \in S, v \in T} f(v,u) \qquad = f(S,T)$$

$$+ \left(\sum_{u \in S, v \in S} f(u,v) - \sum_{u \in S, v \in S} f(v,u) \right) \qquad = 0$$

QED

Corollary: The value of **any** flow is at most the capacity of **any** cut.

Proof:

Let f be an arbitrary flow, and (S, T) be an arbitrary cut.
By previous lemma we have:

$$|f| = f(S,T) = \sum_{u \in S, v \in T} f(u,v) - \sum_{u \in S, v \in T} f(v,u)$$

$$\leq \sum_{u \in S, v \in T} f(u,v)$$

$$\leq \sum_{u \in S, v \in T} c(u,v) = c(S,T)$$

QED

Ford-Fulkerson method

A framework for solving the maximum flow problem

Not a specific algorithm

A famous algorithm based on this framework is Edmonds-Karp

The method builds on two types of objects:

Residual network

and

Augmenting paths

Ford-Fulkerson method

Idea: start with all 0-flow and increase it as follows:

- Find a path p from s to t that allows addition flow to be sent along p without violating capacity constraints
- Send additional flow along p
- Repeat as long as possible

Residual network: helps identify possible paths along which to send additional flow

Augmenting path: a specific path p as above

Residual network

Given current flow f and a pair of vertices u and v, how much additional flow can we push directly from u to v?

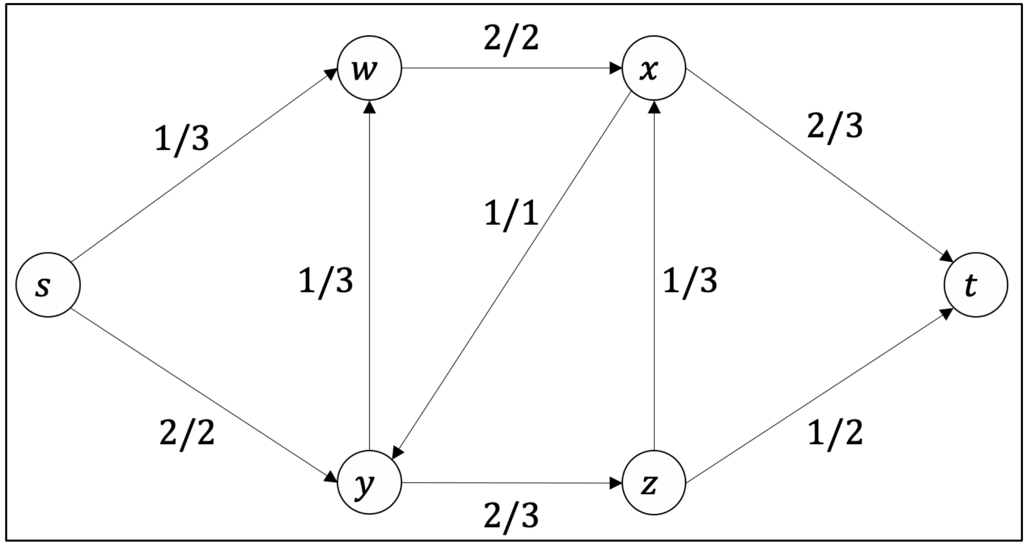
This is called **residual capacity**, denoted by $c_f(u, v)$:

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & (u,v) \in E \\ f(v,u) & (v,u) \in E \\ 0 & (u,v),(v,u) \notin E \end{cases}$$

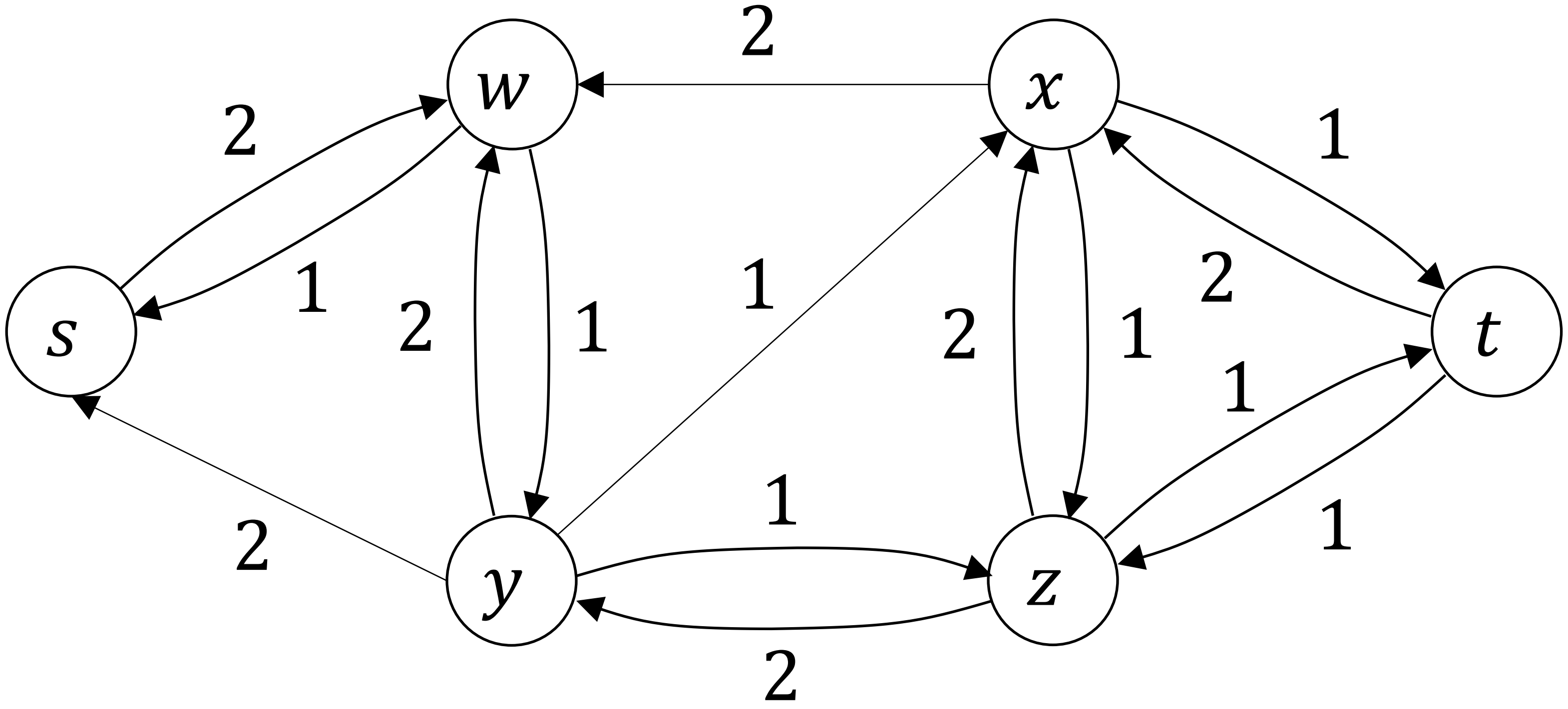
The **residual network** is $G = (V, E_f)$ where

$$E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$

G with flow f:



Residual network G_f:



Residual network properties

Each edge $(u, v) \in E_f$ corresponds to $(u, v) \in E$ or $(v, u) \in E$, thus:

$$\left|E_f\right| \leq 2\ |E|$$

Residual network can contain antiparallel edges $(u, v), (v, u) \in E_f$

Can **define a flow** in residual network that satisfies the definition but **with respect to residual capacities** $c_f(u, v)$

Augmentation

Given flows f in G and f' in G_f define augmentation of f by f', denoted by $(f \uparrow f')$:

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & (u,v) \in E \\ 0 & (u,v) \notin E \end{cases}$$

Intuition:

- Increase $f(u,v)$ by $f'(u,v)$
- Decrease it by $f'(v,u)$ because pushing flow in reverse **cancels** some of the flow in the original network

Augmenting path

Simple path p from s to t in G_f

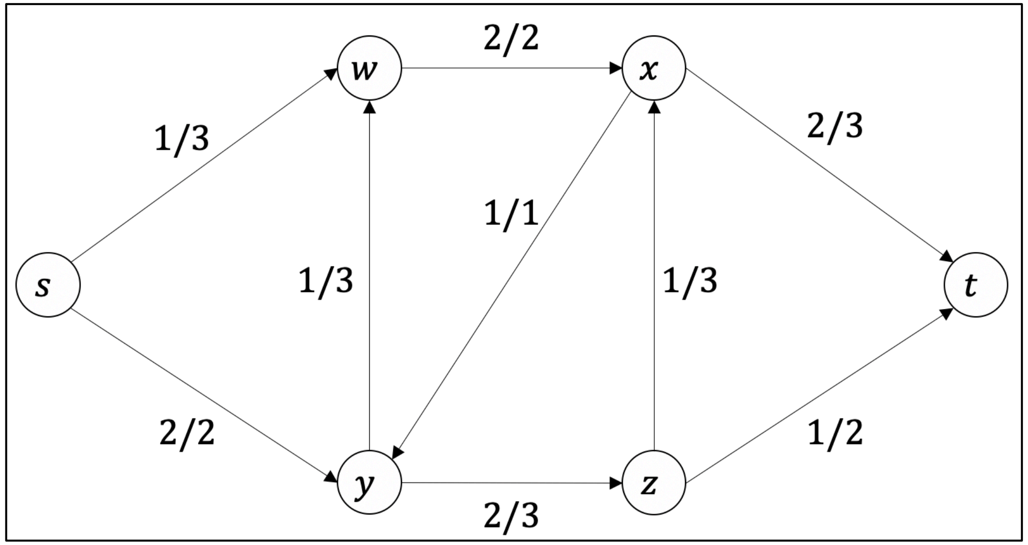
It allows us to push more flow from s to t

How much more flow?

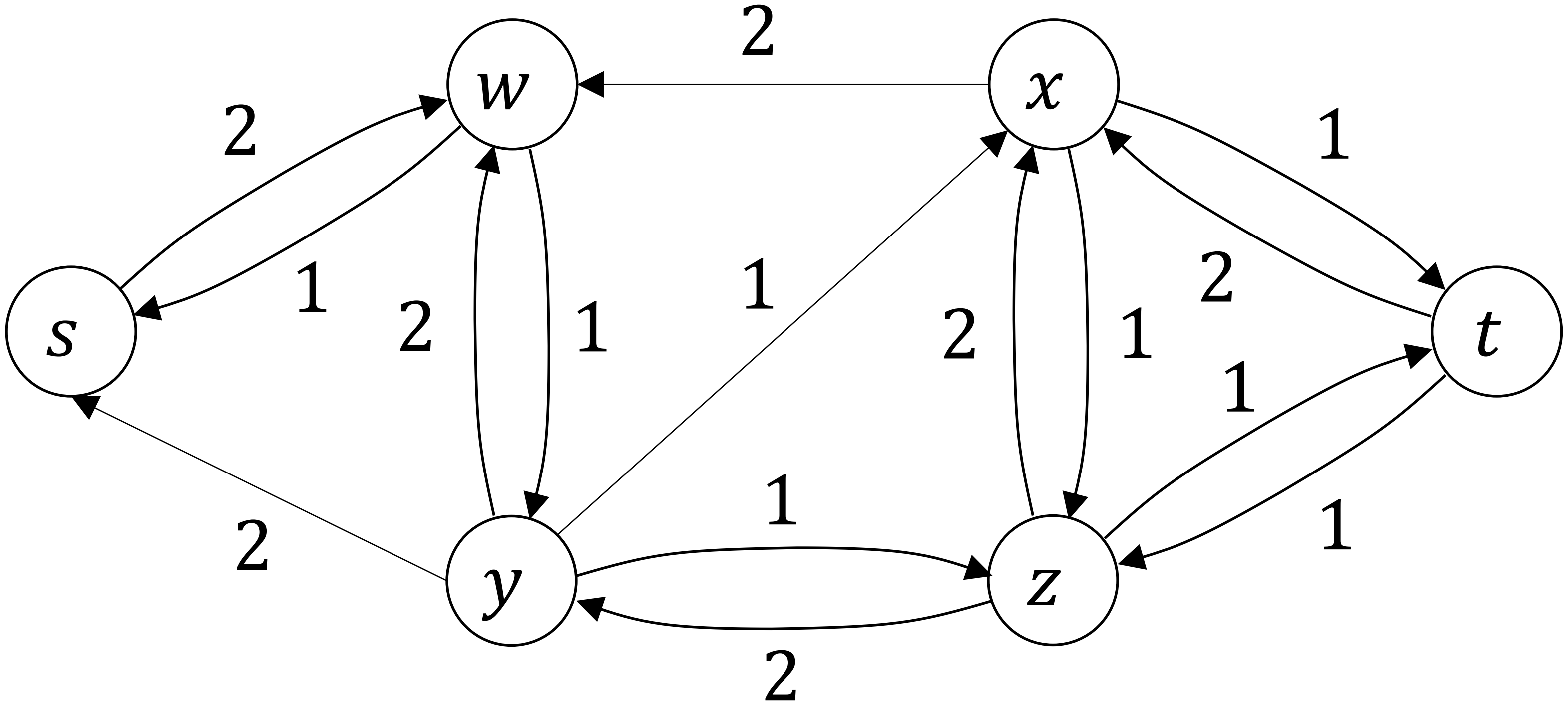
It is restricted by the **minimum residual capacity along the path**, i.e.

$$c_f(p) = \min\{c_f(u, v) : (u, v) \text{ is on path } p\}$$

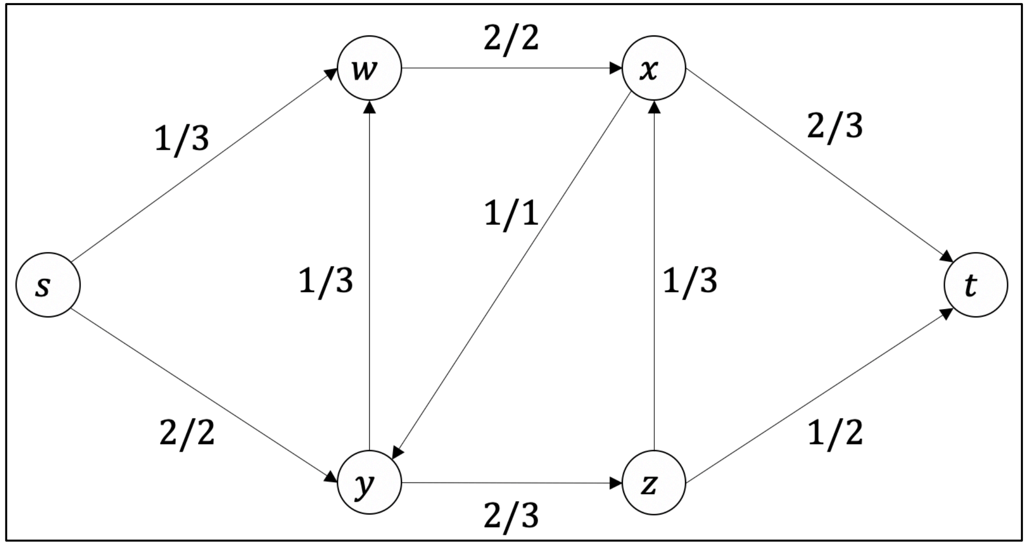
G with flow f:



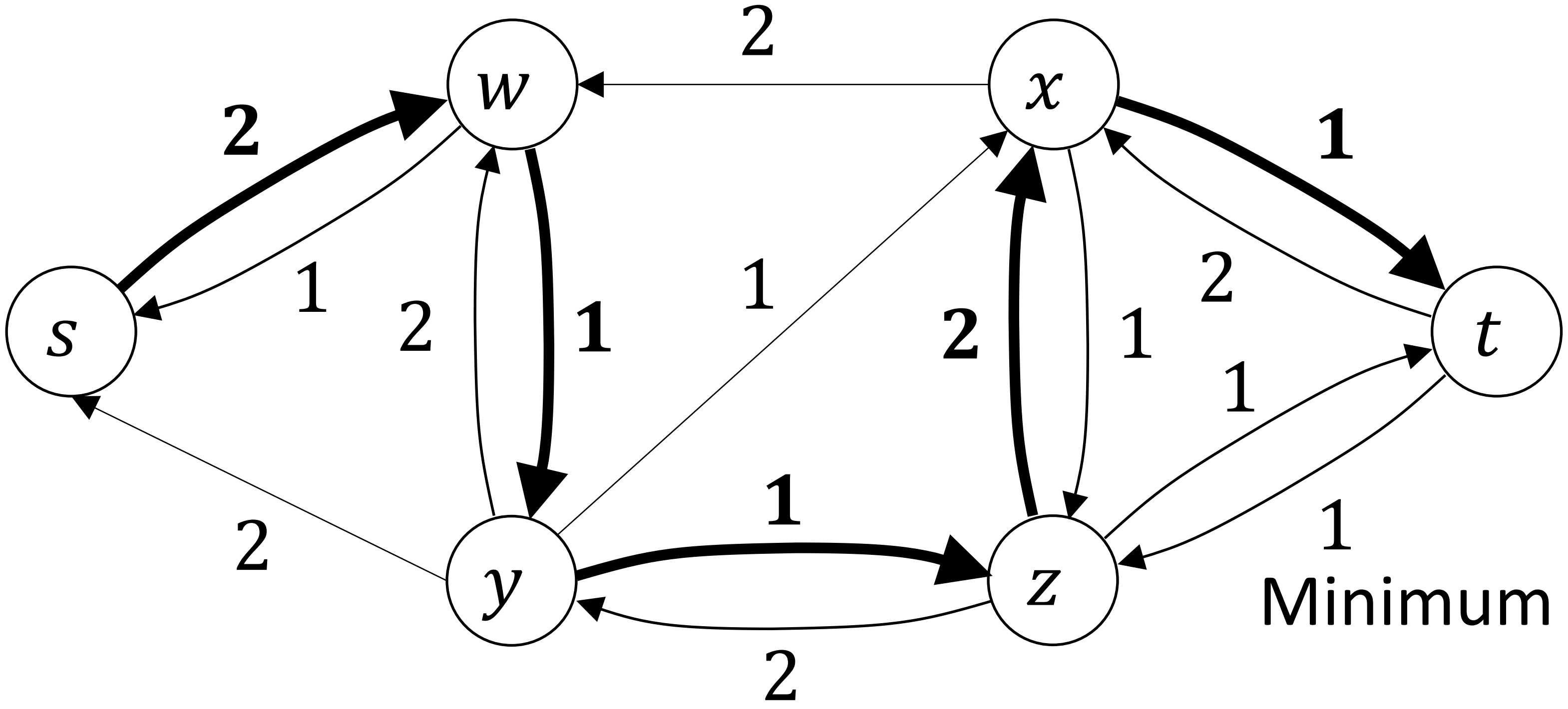
Residual network G_f:



G with flow f:



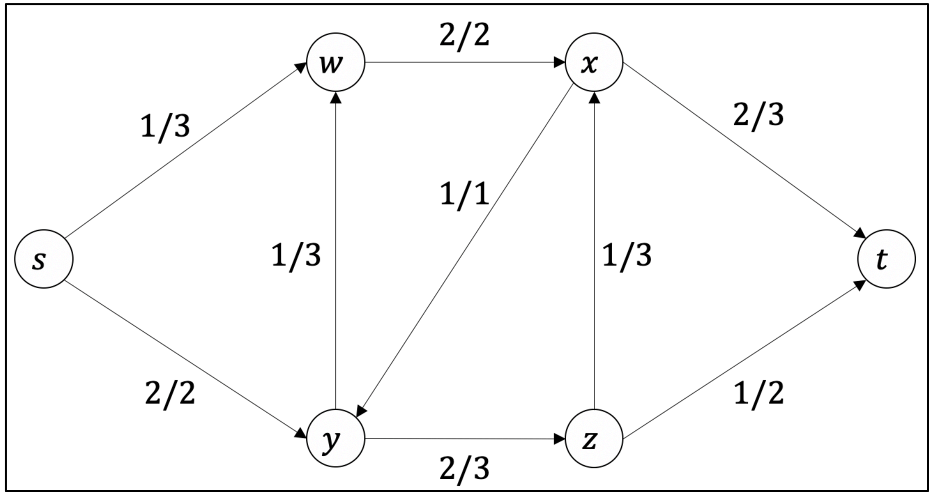
Residual network G_f:



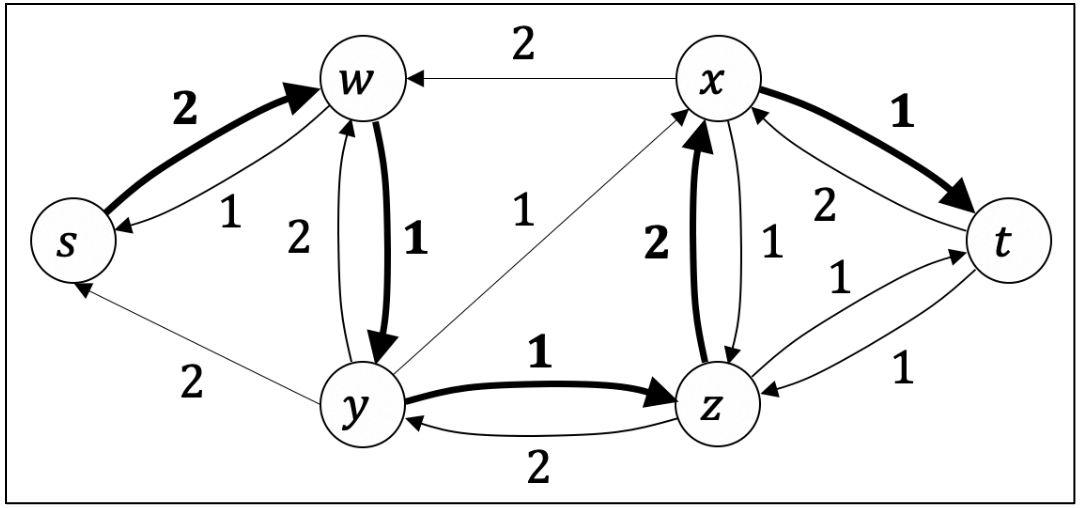
Augmenting path
$p = \langle s, w, y, z, x, t \rangle$

Minimum residual capacity along p
$c_f(p) = 1$

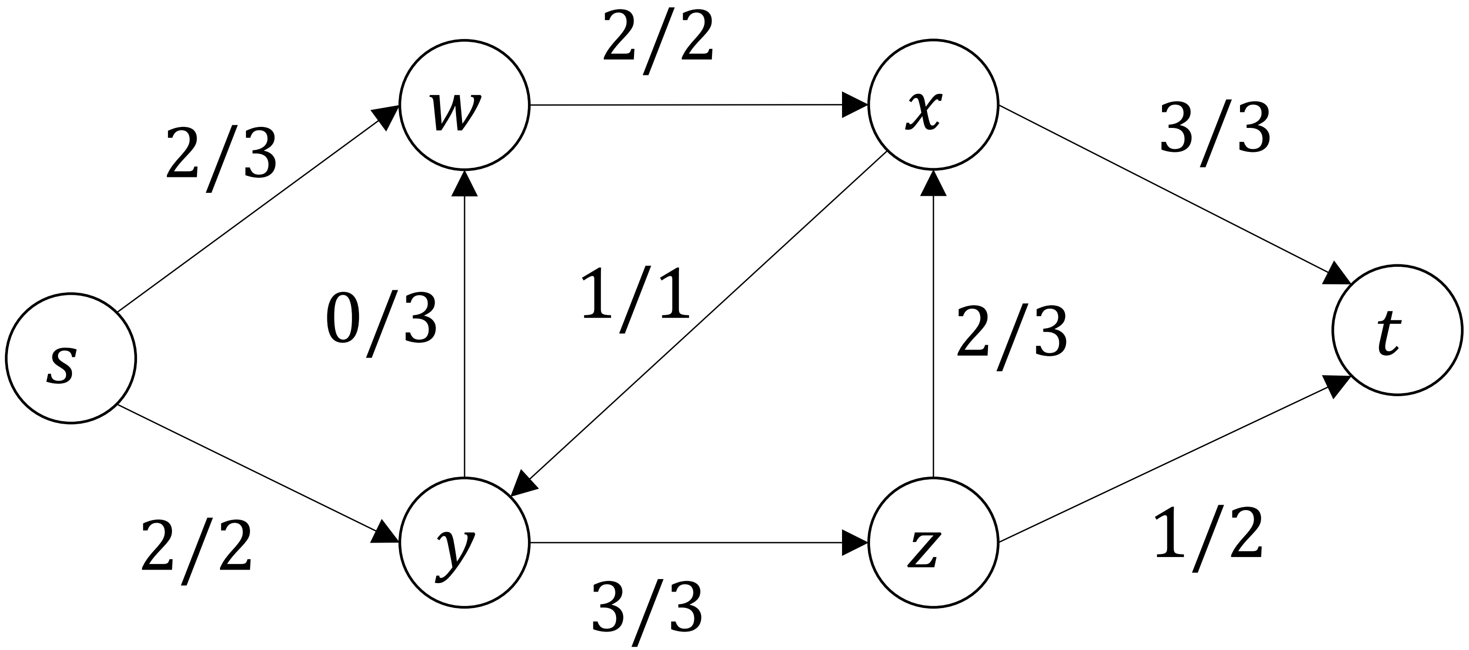
G with flow f:

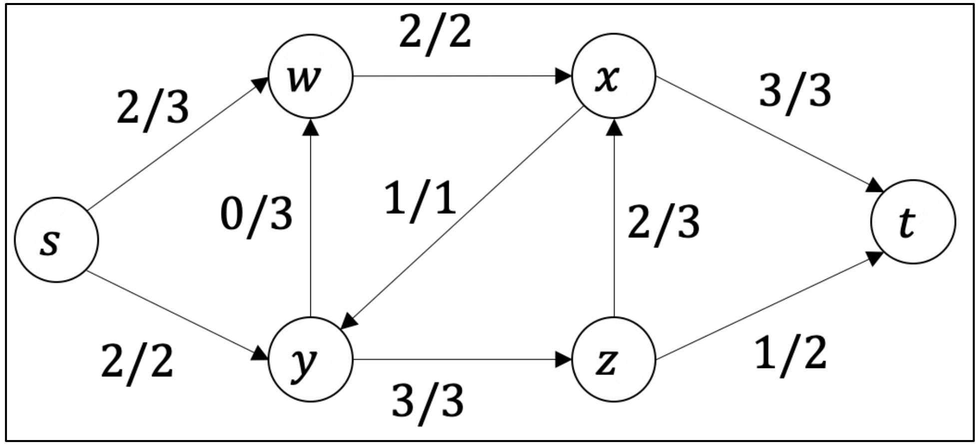


Residual network G_f and augmenting path

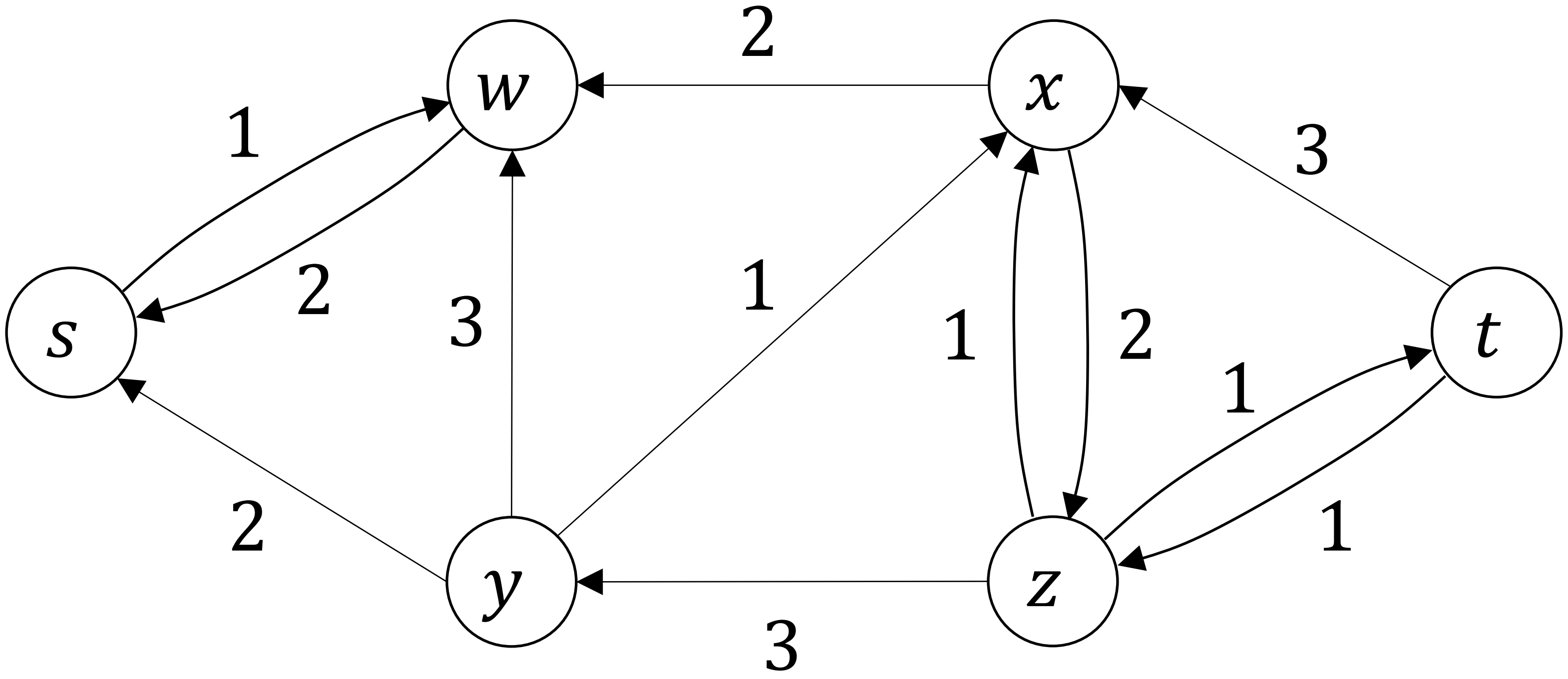


After pushing 1 unit of flow along the augmenting path:





Residual graph after pushing 1 unit of flow along the augmenting path:



No augmenting path. We claim that the flow is maximum

Lemma: given flow network G, flow f in G, and an augmenting path p in residual graph G_f. Define $f_p : V \times V \to \mathbb{R}$

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \text{ is on } p \\ 0 & \text{otherwise} \end{cases}$$

Then f_p is a flow in G_f with value $|f_p| = c_f(p) > 0.$

Corollary: $(f \uparrow f_p)$ is a flow in G with value

$$|f \uparrow f_p| = |f| + |f_p| > |f|$$

Max-flow min-cut theorem

The following are equivalent:

1. f is a maximum flow
2. G_f has no augmenting path
3. $|f| = c(S,T)$ for some cut (S,T)

Proof:

$1 \Rightarrow 2$: show contrapositive if G_f has an augmenting path p then f is not maximum, since $f \uparrow f_p$ is a flow of bigger value than f.

Max-flow min-cut theorem

The following are equivalent:

1. f is a maximum flow
2. G_f has no augmenting path
3. $|f| = c(S, T)$ for some cut (S, T)

Proof:

$2 \Rightarrow 3$: suppose G_f has no augmenting path. Define:

$$S = \{v \in V : \text{there exists a path from } s \text{ to } v \text{ in } G_f\}$$

$$T = V - S$$

We have $t \in T$ since otherwise there is an augmenting path

Therefore (S, T) is a valid cut

Max-flow min-cut theorem

The following are equivalent:

1. f is a maximum flow
2. G_f has no augmenting path
3. $|f| = c(S,T)$ for some cut (S,T)

Proof:

$2 \Rightarrow 3$: … continued. For $u \in S$ and $v \in T$

- $(u,v) \in E$ implies that $f(u,v) = c(u,v)$ (why?)
- $(v,u) \in E$ implies that $f(v,u) = 0$ (why?)
- $(u,v),(v,u) \notin E$ implies that $f(u,v) = f(v,u) = 0$

Max-flow min-cut theorem

The following are equivalent:

1. f is a maximum flow
2. G_f has no augmenting path
3. $|f| = c(S, T)$ for some cut (S, T)

Proof:

$2 \Rightarrow 3$: … continued.

$$|f| = f(S, T) = \sum_{u \in S, v \in T} f(u, v) - \sum_{v \in T, u \in S} f(v, u)$$

$$= \sum_{u \in S, v \in T} c(u, v) - \sum_{v \in T, u \in S} 0 = c(S, T)$$

Max-flow min-cut theorem

The following are equivalent:

1. f is a maximum flow
2. G_f has no augmenting path
3. $|f| = c(S,T)$ for some cut (S,T)

Proof:

$3 \Rightarrow 1$: If $|f| = c(S,T)$ and for any flow f' we have $|f'| \leq c(S,T)$ (by one of the previous corollaries). Therefore f is maximum flow.

Ford-Fulkerson method

- Keep augmenting flow along an augmenting path until there is no augmenting path.
- Represent flow in an attribute $(u, v).f$

$Ford - Fulkerson(G = (V, E), s, t)$

    $\boldsymbol{for}\ all\ (u, v) \in E$

        $(u, v).f \leftarrow 0$

    $\boldsymbol{while}$ there is an augmenting path p in G_f

        augment f by f_p

- If capacities are all integers each augmenting path raises the value of flow by at least 1.
- Let f^* denote a max flow then Ford-Fulkerson method needs at most $|f^*|$ iterations.
- The running time is $O(|E| \cdot |f^*|)$.
- This running time is not polynomial since $|f^*|$ is not a function of $|V|$ and $|E|$.
- If capacities are rational then they can be scaled to integers.
- If capacities are irrational then this method might never terminate!

Integrality theorem

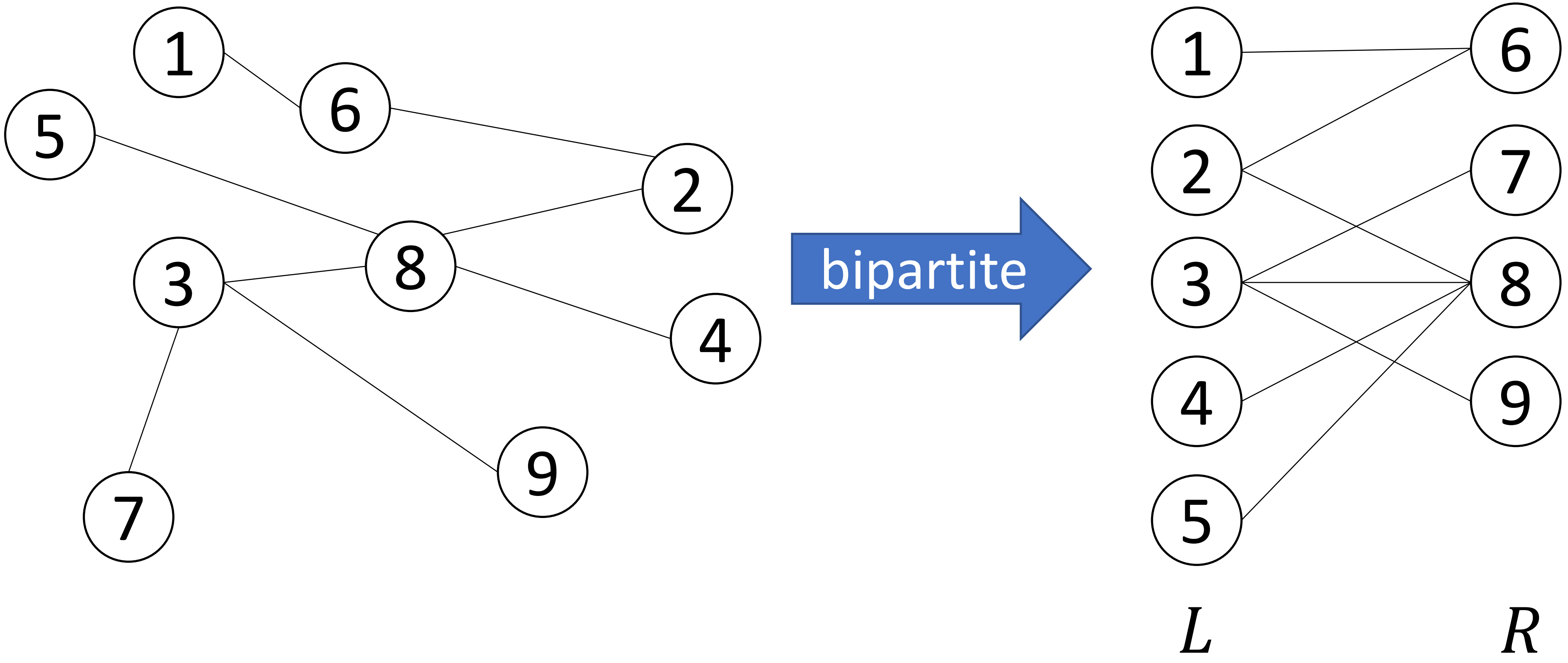
If the capacity function c takes on only **integer** values then the maximum flow produced by the Ford-Fulkerson method has the property that $|f|$ is an **integer**. Moreover for all vertices (u, v) the value $f(u, v)$ is an **integer**.

Edmonds-Karp algorithm

- Perform $Ford - Fulkerson$ but to compute augmenting paths run **BFS** in G_f
- Augmenting paths are shortest unweighted paths in G_f
- Theorem: Edmonds-Karp performs $O(|V| \cdot |E|)$ augmentations
- Thus, the overall running time is $O(|V| \cdot |E|^2)$
- See the book for details.

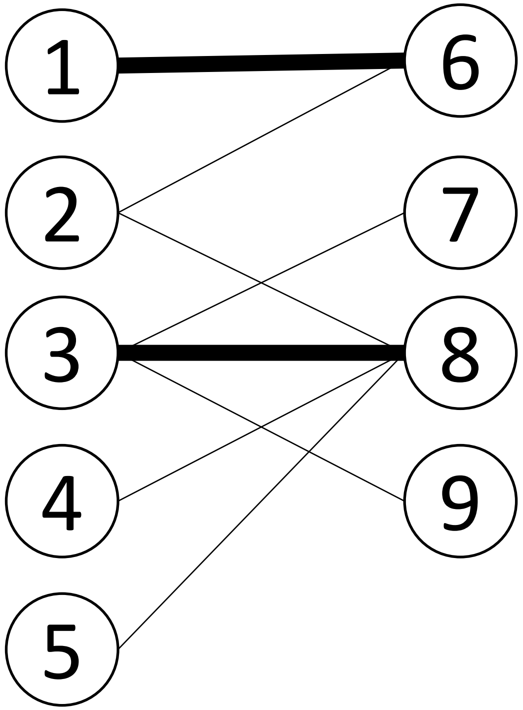
Bipartite graphs

An undirected graph $G = (V, E)$ is **bipartite** if we can partition the set of vertices $V = L \cup R$ such that all edges go between L and R

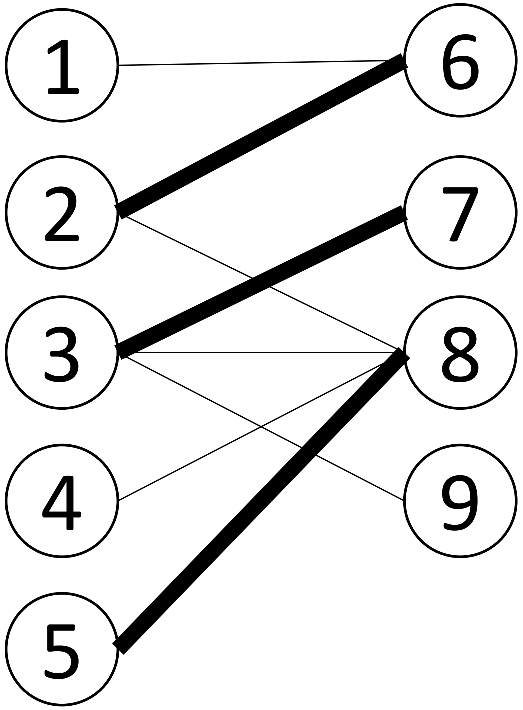


A **matching** is a subset of edges $M \subseteq E$ such that no two edges from M share a common vertex, i.e., for all $e_1, e_2 \in M$ we have $e_1 \cap e_2 = \emptyset$

A matching of maximum size is called a **maximum matching**.



A matching



A maximum matching

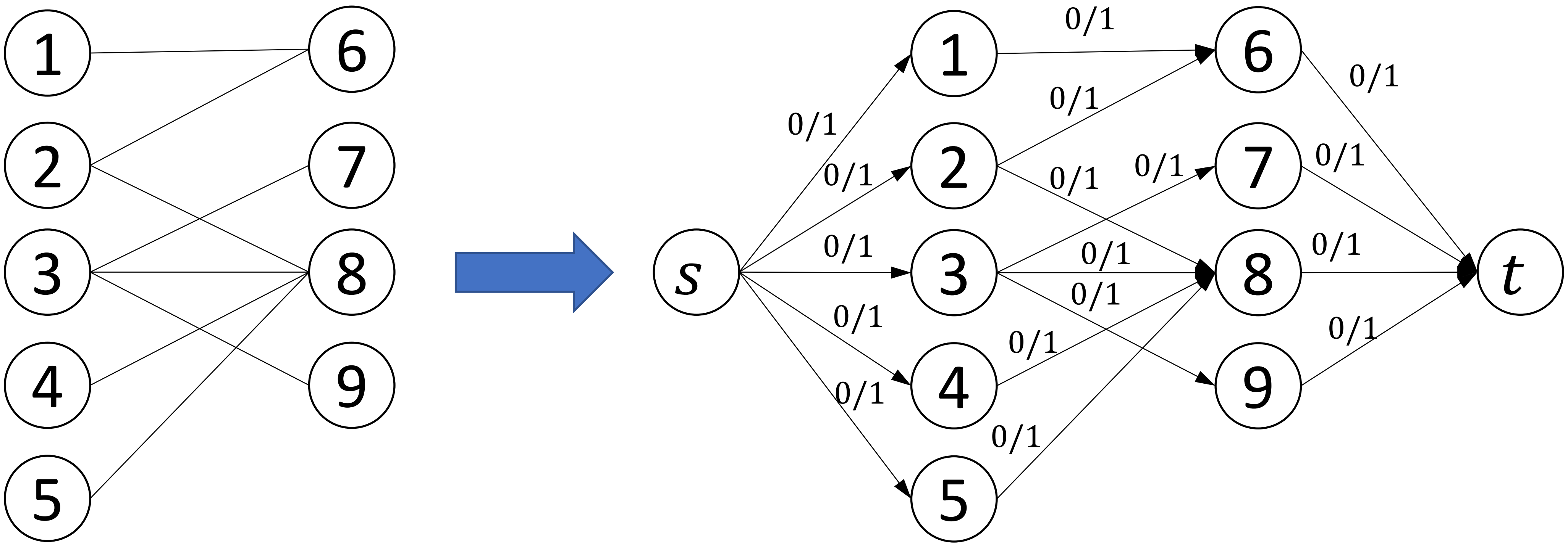
Maximum matching problem

Input: given undirected bipartite graph $G = (V, E)$ with bipartition $V = L \cup R$

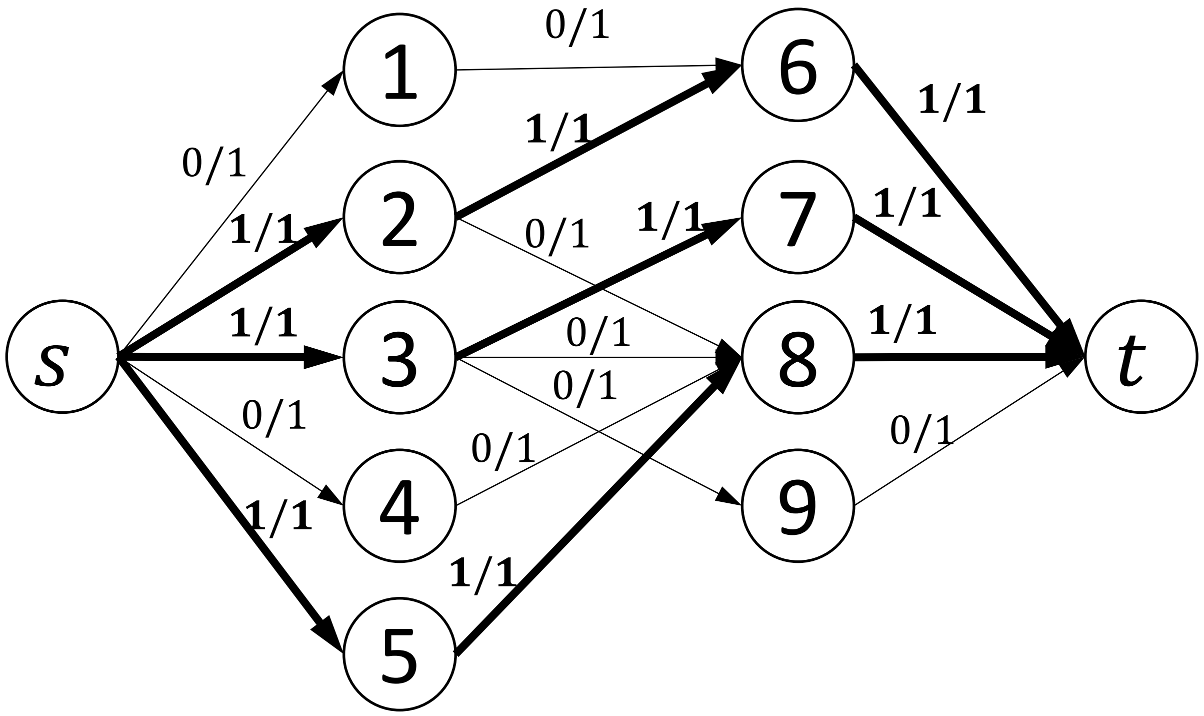
Output: a maximum matching $M \subseteq E$

Given a bipartite G define flow network $G' = (V', E')$ as follows:

- $V' = V \cup \{s, t\}$
- $E' = \{(s,u) : u \in L\} \cup \{(u,v) : u \in L, v \in R, \{u,v\} \in E\} \cup \{(v,t) : v \in R\}$
- $c(u,v) = 1$ for all $(u,v) \in E'$



- Find an integral max flow f
- Include those edges (u, v) that have $u \in L$ and $v \in R$ and
$$f(u, v) = 1$$



- Running time is $O(|V| \cdot |E|)$
- See the book for details

Now you should be able to…

- Solve single-source shortest paths on general weighted graphs (with negative edges and possibly even negative cycles)
- Solve all-pairs shortest paths using either matrix multiplication or Floyd-Warshall. Understand how to speed up matrix multiplication approach using repeated squaring
- Describe flow networks, state defining properties of flows, residual graphs, augmenting paths
- Explain $Ford - Fulkerson$ method and Edmonds-Karp algorithm. Explain the difference between the two
- Apply network flow algorithms to solve the maximum matching problem

Review questions

- Write down pseudocode for Bellman-Ford, APSP matrix multiplication, Floyd-Warshall
- Write down pseudocode for Edmonds-Karp
- Write down pseudocode for solving the maximum matching algorithm
- State the max-flow and min-cut theorem and the integrality theorem