Lecture 12

Network flows, Max flow, Min-cut

CS 161 Design and Analysis of Algorithms

Ioannis Panageas

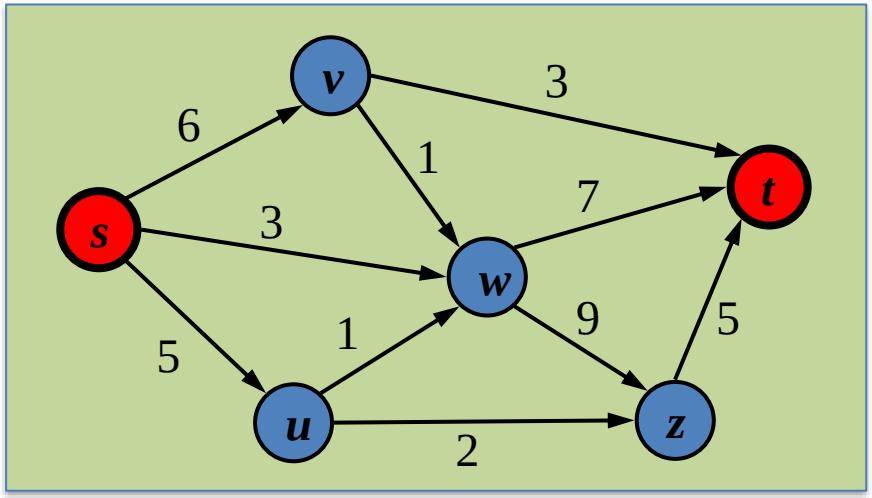
Flow Networks

Definition: A flow network consists of

- A weighted directed graph G with non-negative integer edge weights called capacities and denoted by $\boldsymbol{c(e)}$.
- Vertices, $\boldsymbol{s}$ and $\boldsymbol{t}$ of G, called the source and sink; $\boldsymbol{s}$ has no incoming edges and $\boldsymbol{t}$ has no outgoing edges.

Example:

(w, z) has capacity $c(w, z) = 9$.



Flow of a Network

Definition: Function $f : E \to \mathbb{N}$ from edges to non-negative integers so that for each edge e it holds

$$0 \leq f(e) \leq c(e) \quad \text{Capacity constraint}$$

$$\sum_{e \in \mathbf{outgoing}(u)} f(e) = \sum_{e \in \mathbf{incoming}(u)} f(e) \quad \text{Conservation rule}$$

for all $u \neq s, t$

Flow of a Network

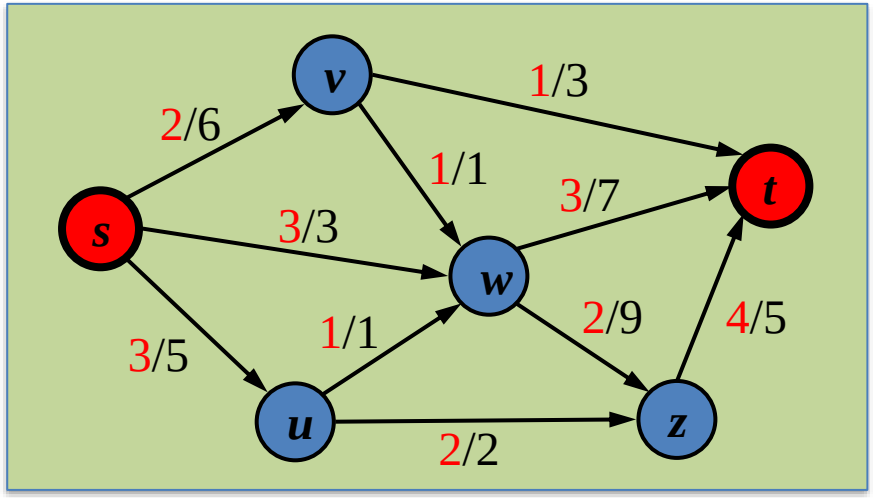
Definition: Function $f : E \to \mathbb{N}$ from edges to non-negative integers so that for each edge e it holds

$$0 \leq f(e) \leq c(e) \quad \text{Capacity constraint}$$

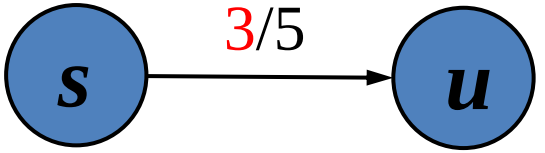
$$\sum_{e \in \textbf{outgoing}(u)} f(e) = \sum_{e \in \textbf{incoming}(u)} f(e) \quad \text{Conservation rule}$$

for all $u \neq s, t$

Example:



Capacity constraint:



$$0 \leq 3 \leq 5$$

Flow of a Network

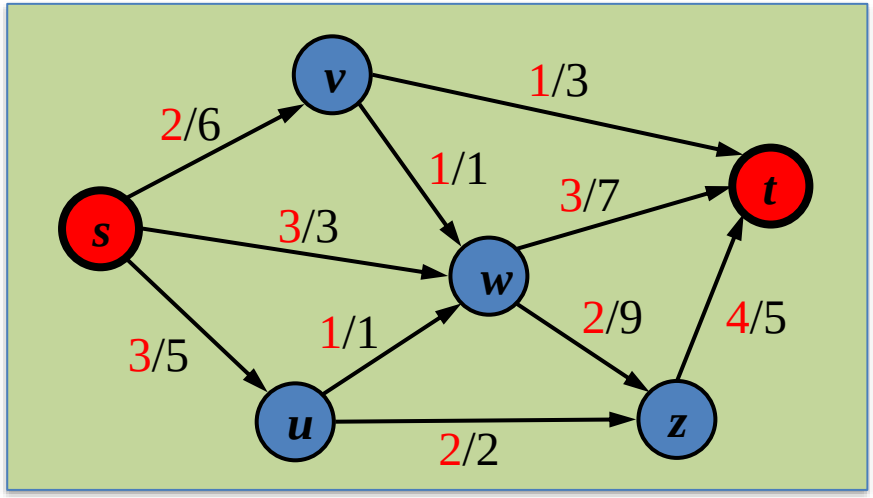
Definition: Function $f : E \to \mathbb{N}$ from edges to non-negative integers so that for each edge e it holds

$$0 \leq f(e) \leq c(e) \quad \text{Capacity constraint}$$

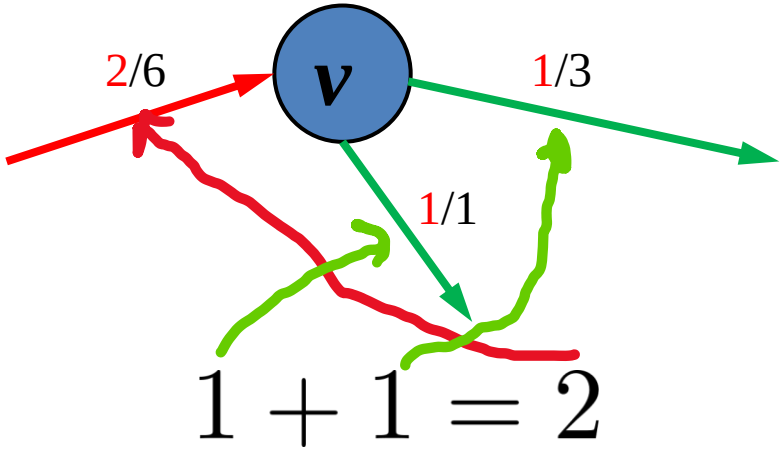
$$\sum_{e \in \mathbf{outgoing}(u)} f(e) = \sum_{e \in \mathbf{incoming}(u)} f(e) \quad \text{Conservation rule}$$

for all $u \neq s, t$

Example:



Conservation rule:



Flow of a Network

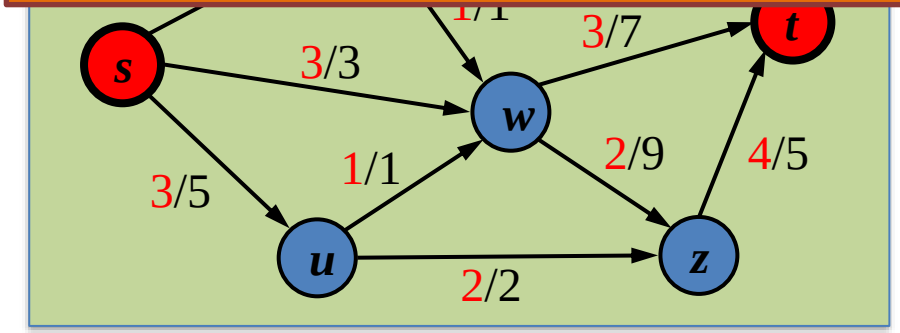
Definition: Function $f : E \to \mathbb{N}$ from edges to non-negative integers so that for each edge e it holds

$$0 \leq f(e) \leq c(e) \quad \text{Capacity constraint}$$

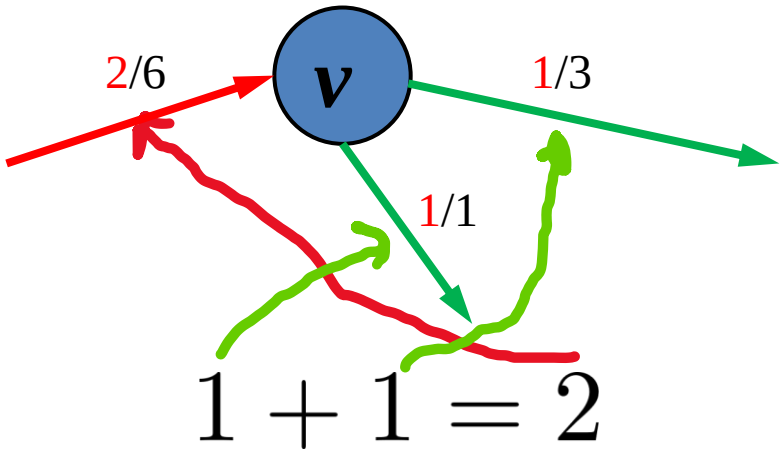
$$\sum_{e \in \mathbf{outgoing}(u)} f(e) = \sum_{e \in \mathbf{incoming}(u)} f(e) \quad \text{Conservation rule}$$

for all $u \neq s, t$

Question: if $f(e) = 0$ for all e, is it a flow? Yes.



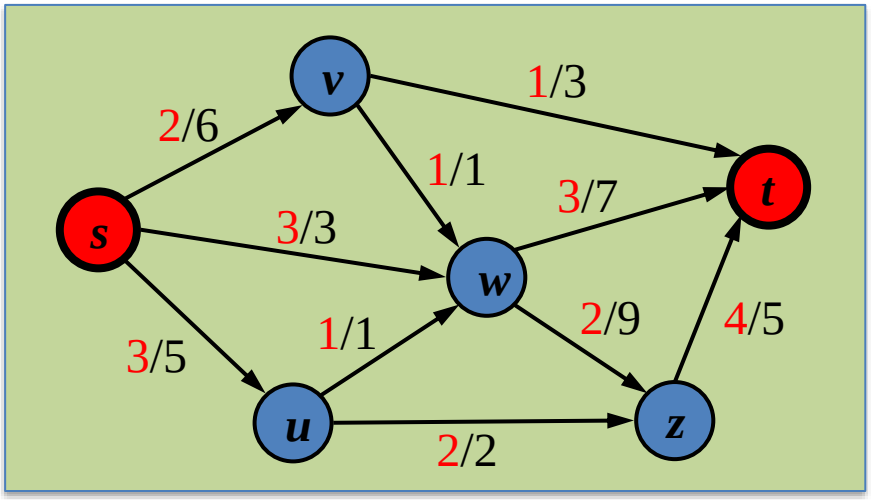
Conservation rule:



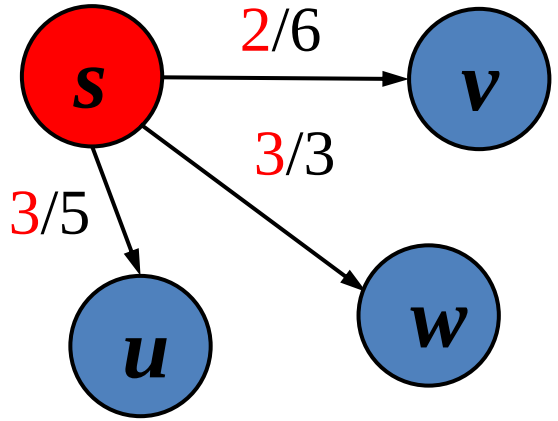
Value of a flow

Definition: Given a flow f, the value of flow $|f|$ is the total flow from source s, which is the same as the total flow into sink t.

Example:



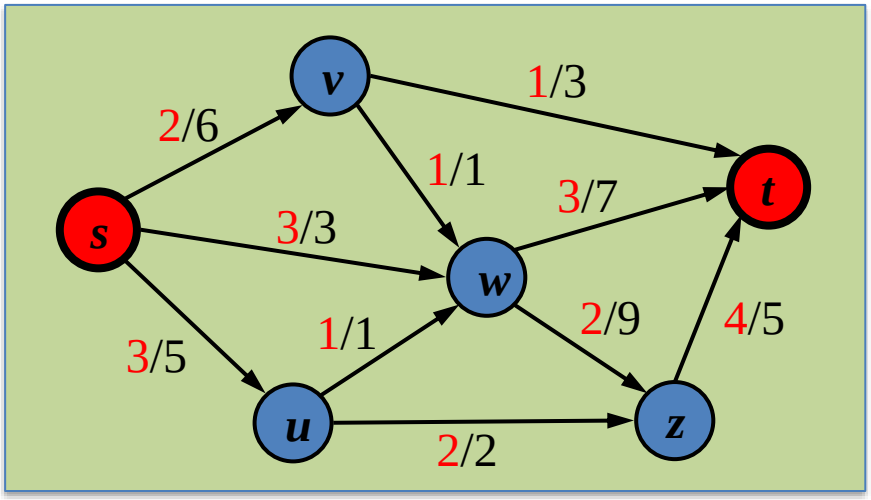
Total Flow $|f| = 3 + 3 + 2 = 8$



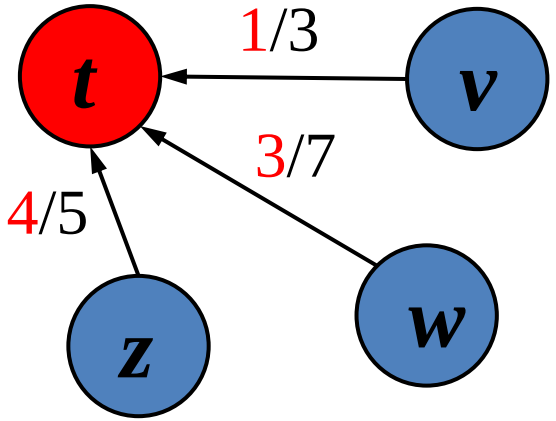
Value of a flow

Definition: Given a flow f, the value of flow $|f|$ is the total flow from source s, which is the same as the total flow into sink t.

Example:



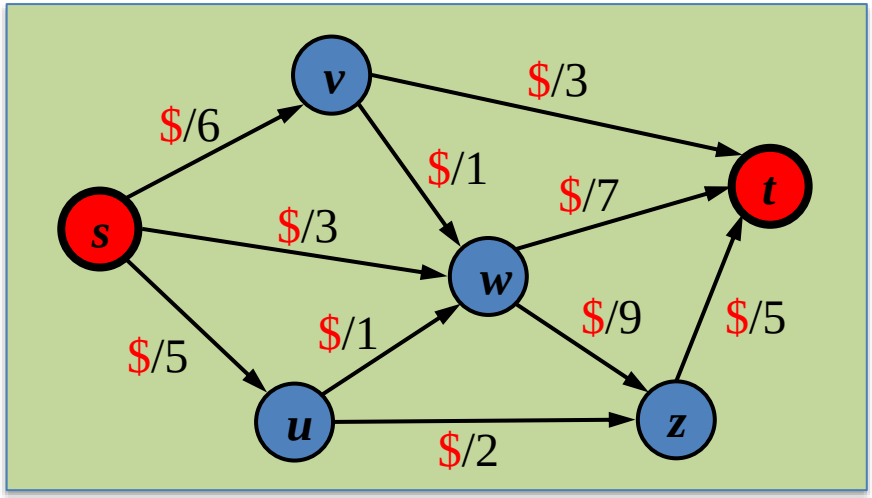
Total Flow $|f| = 4 + 3 + 1 = 8$



Maxflow Problem

Problem: Given a network G, a source s and a sink t, and capacities on the edges, compute the maximum possible flow value $|f^*|$.

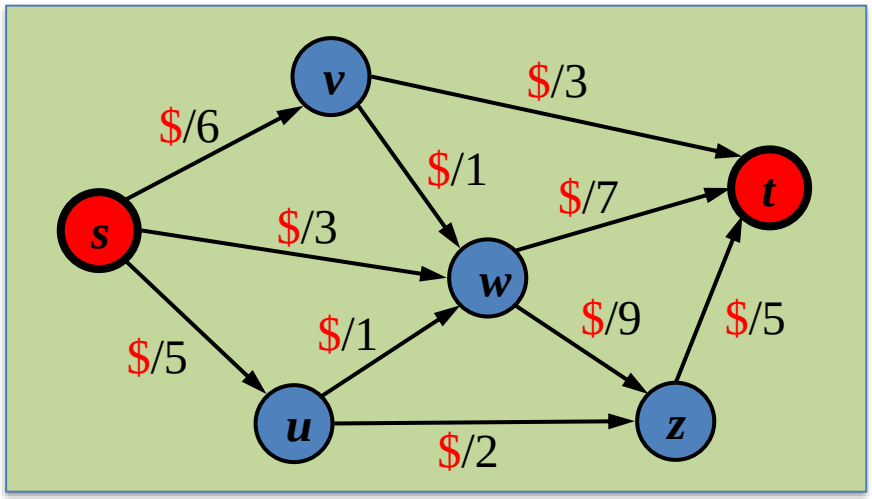
Example:



Find the $ to get maxflow $|f^*|$

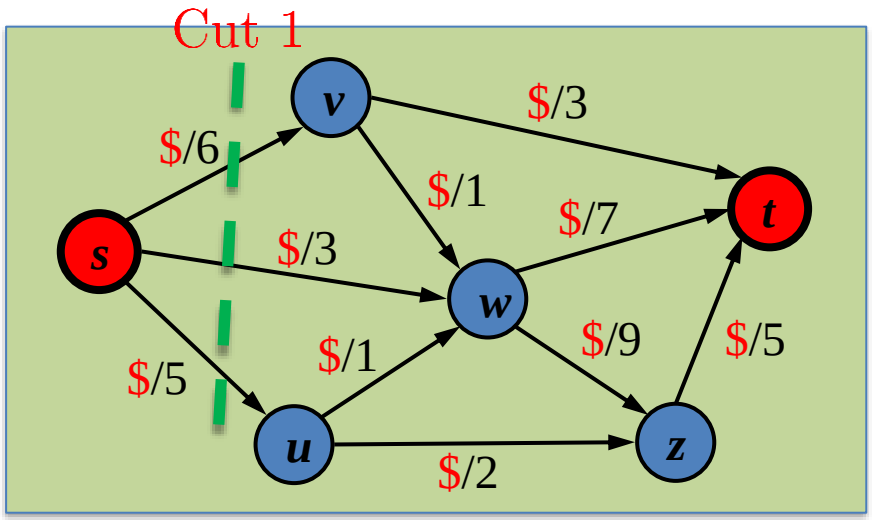
Maxflow Problem

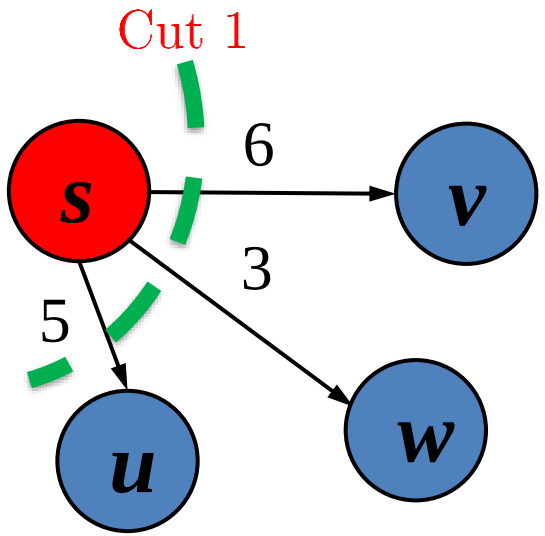
Question: How large can $|f|$ be in terms of the capacities?



Maxflow Problem

Question: How large can $|f|$ be in terms of the capacities?



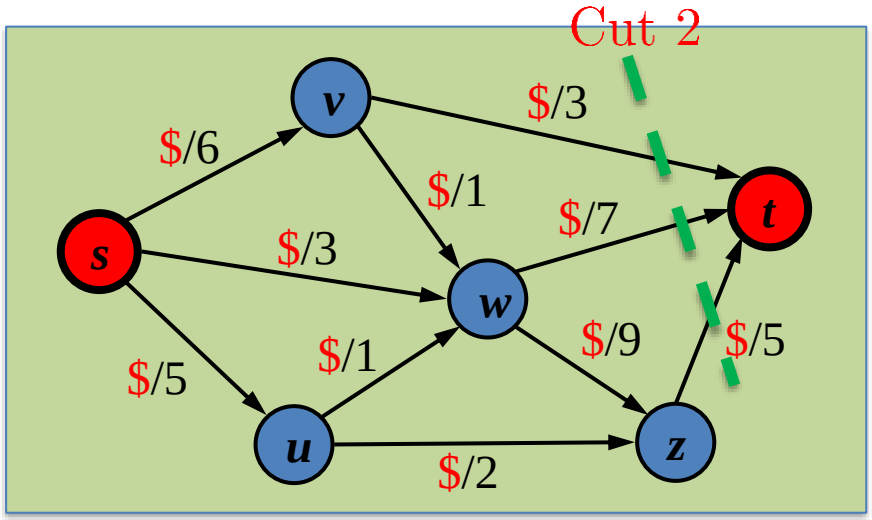


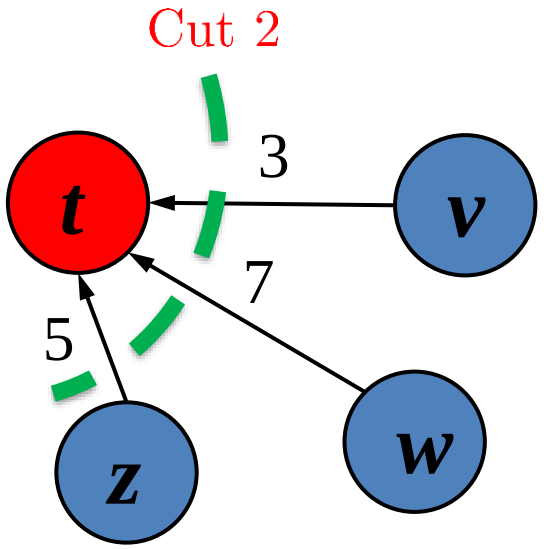
Answer

- Focusing on Cut 1, it should be **at most** 6+3+5=**14**.

Maxflow Problem

Question: How large can $|f|$ be in terms of the capacities?

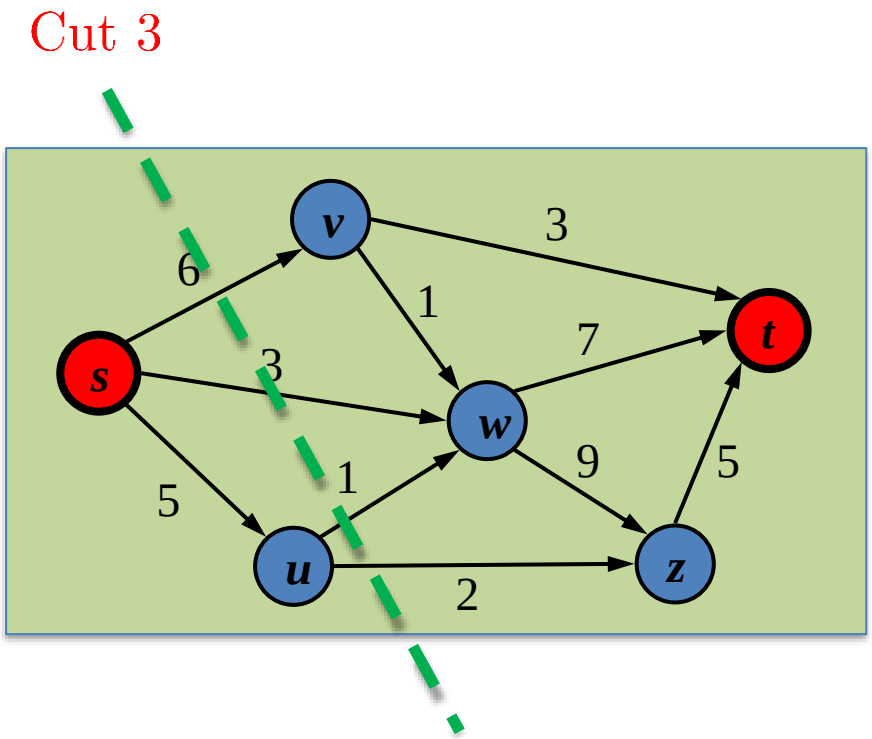




Answer

- Focusing on Cut 1, it should be **at most** 6+3+5=**14**.
- Focusing on Cut 2, it should be **at most** 3+7+5=**15**.

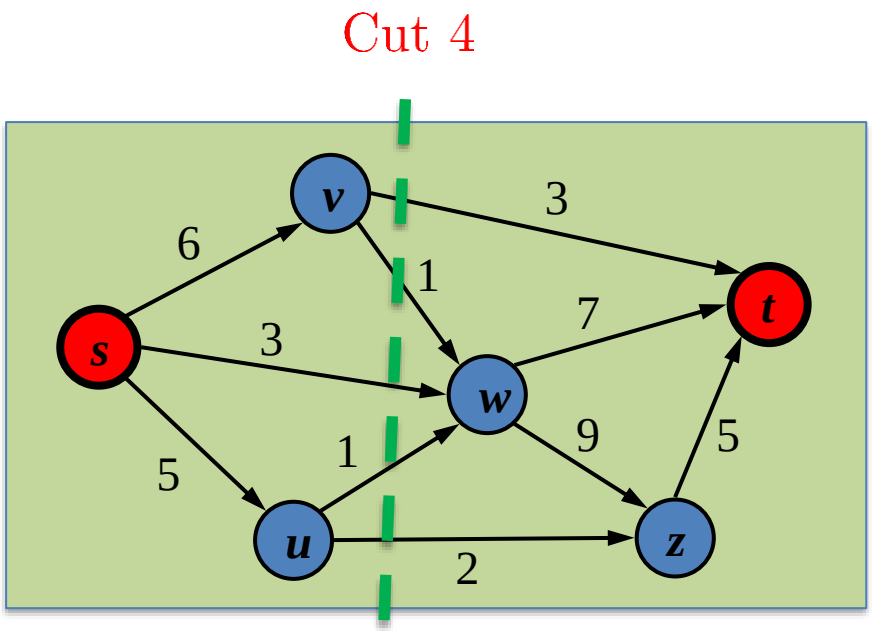
Maxflow Problem



Answer

- Focusing on Cut 3, it should be at most 6+3+1+2=**12**.

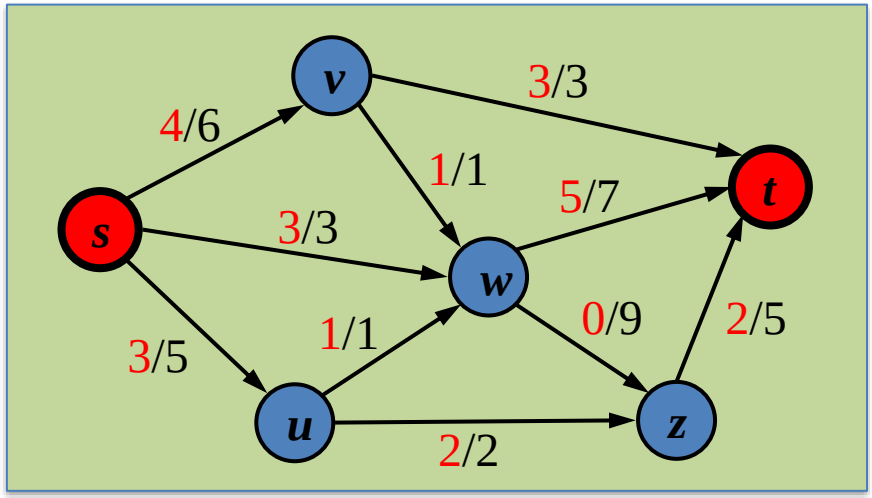
Maxflow Problem



Answer

- Focusing on Cut 3, it should be at most 6+3+1+2=**12**.
- Focusing on Cut 4, it should be at most 3+1+3+1+2=**10**.

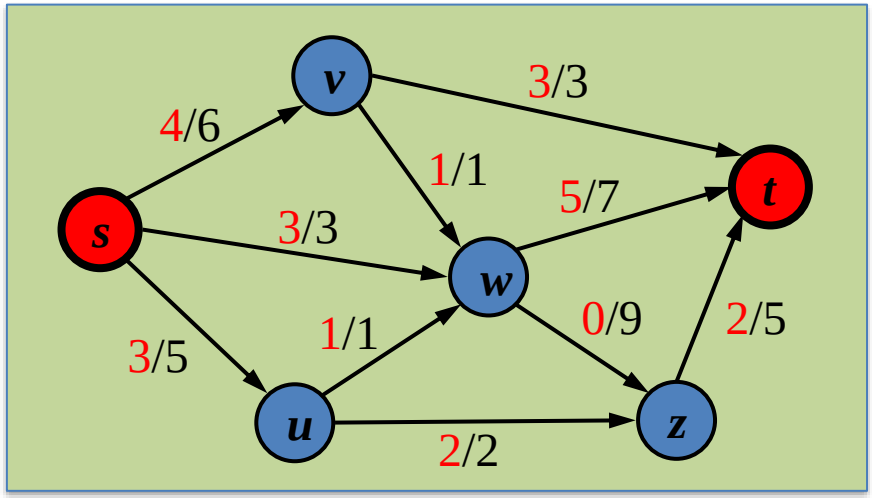
Maxflow Problem



Answer

- The above is a flow function (satisfies all conditions)
- The value of the flow is 4+3+3 = 10.

Maxflow Problem



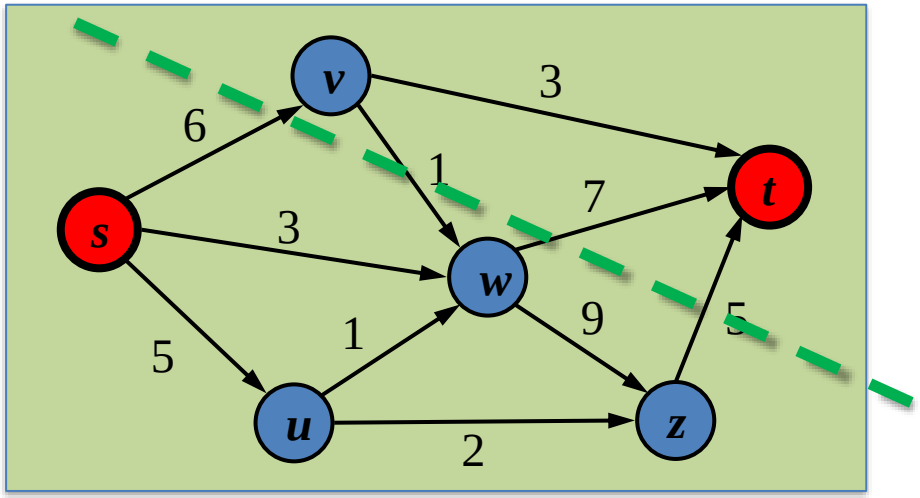
Answer

- The above is a flow function (satisfies all conditions)
- The value of the flow is 4+3+3 = 10.

Maxflow should be $\leq \mathbf{10}$ and found a flow with $|f| = \mathbf{10}$

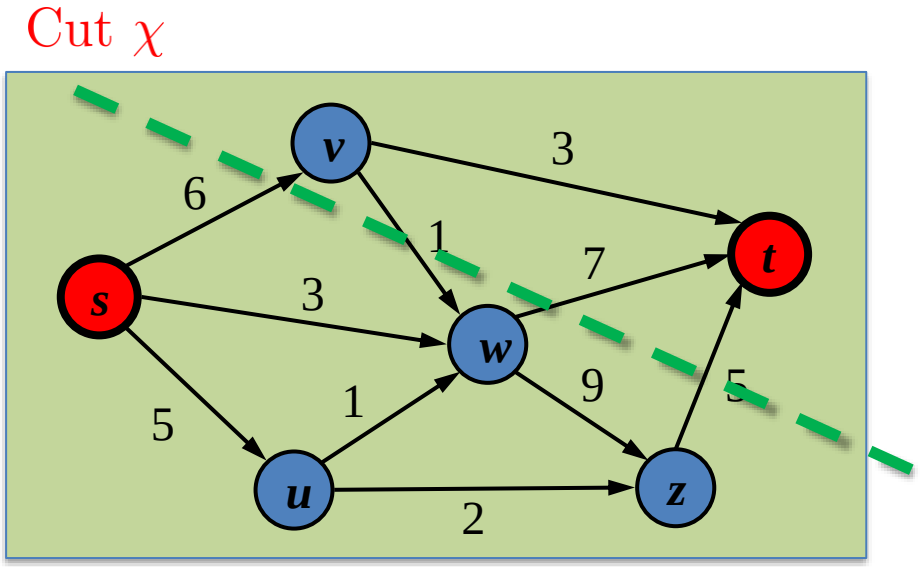
Cut

Definition: Given a network G, source $\boldsymbol{s}$, sink $\boldsymbol{t}$ and capacities on the edges, a cut is a partition of vertices in two parts V_s, V_t with $\boldsymbol{s}$ in V_s and $\boldsymbol{t}$ in V_t.



$V_s = \{s, u, w, z\}$ and $V_t = \{v, t\}$

Cut

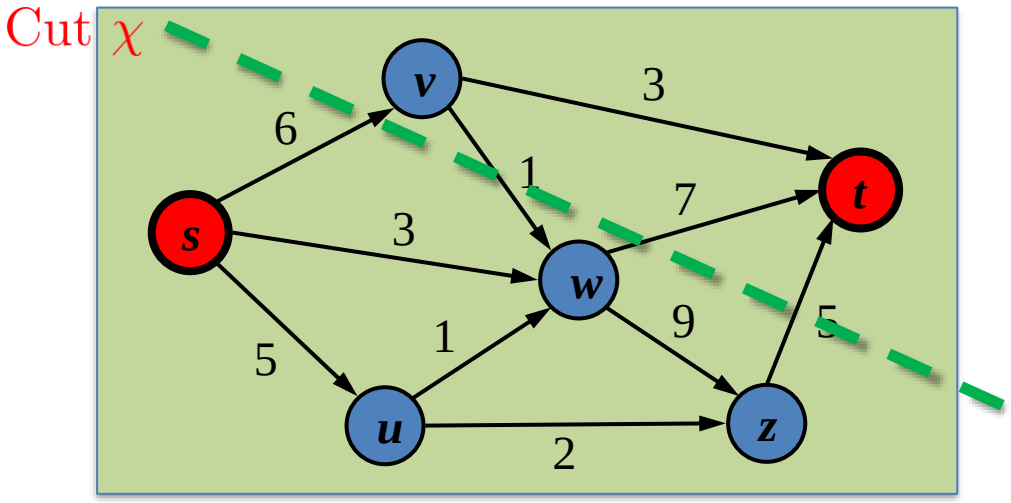


$V_s = \{s, u, w, z\}$ and $V_t = \{v, t\}$

Forward edge: origin in V_s and destination in V_t
Forward Edges in χ: $(s, v), (w, t), (z, t)$.

Backward edge: origin in V_t and destination in V_s
Backward Edges in χ: (v, w).

Cut



$V_s = \{s, u, w, z\}$ and $V_t = \{v, t\}$

Forward edge: origin in V_s and destination in V_t

Forward Edges in χ: $(s, v), (w, t), (z, t)$.

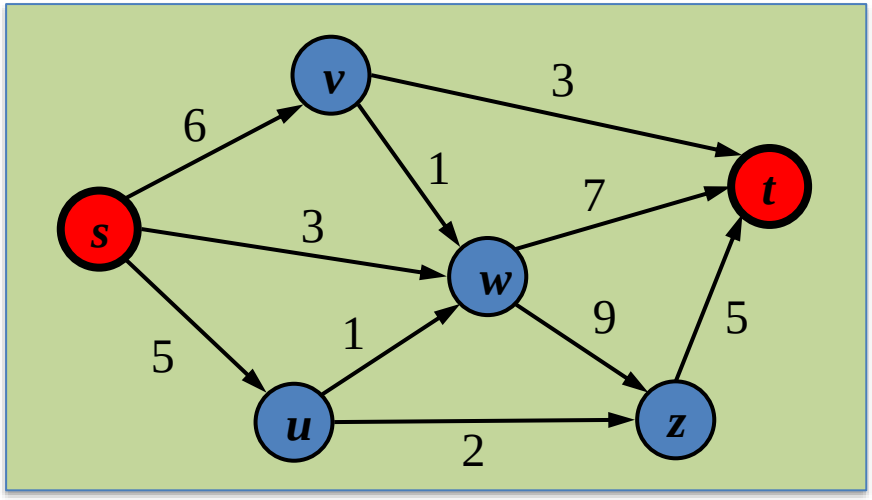
Backward edge: origin in V_t and destination in V_s

Backward Edges in χ: (v, w).

Capacity of a cut: Total capacity of forward edges.

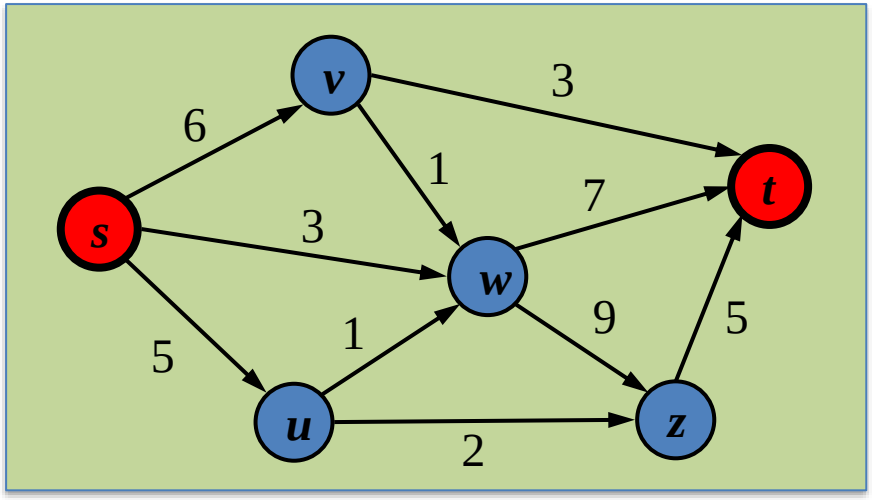
$\boldsymbol{c}(\chi) = 5 + 6 + 7 = 18$

Maxflow = Min Cut



There are 16 cuts (why)?

Maxflow = Min Cut



There are 16 cuts (why)?

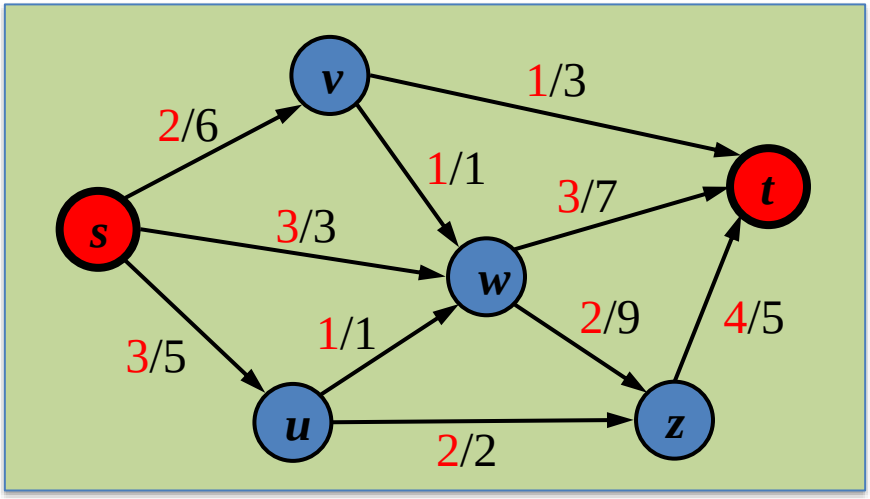
Theorem: The minimum capacity cut **equals** the maxflow value.

Augmenting paths

We are given a network G with edge capacities c and a flow f. Let (u, v) be an edge from u to v.

Residual capacity from u to v is $\Delta_f(u, v) = c(u, v) - f(u, v)$.

Residual capacity from v to u is $\Delta_f(v, u) = f(u, v)$



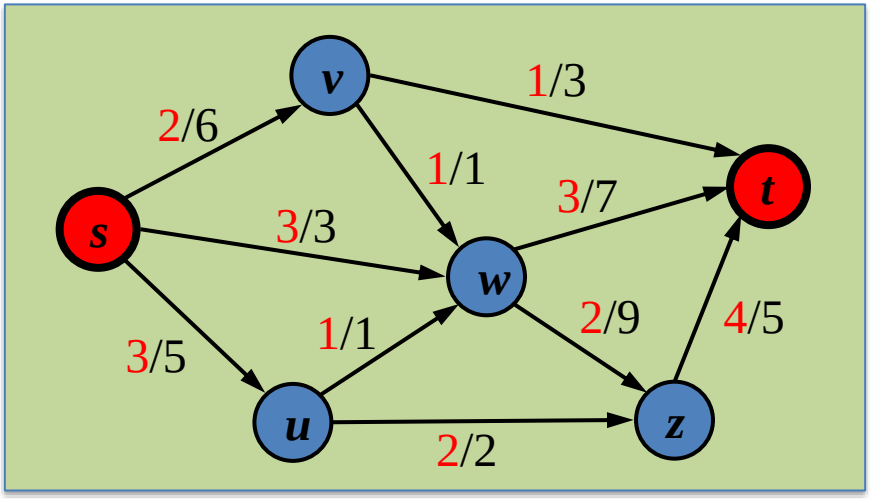
$$\Delta_f(s, v) = ?$$

Augmenting paths

We are given a network G with edge capacities c and a flow f. Let (u, v) be an edge from u to v.

Residual capacity from u to v is $\Delta_f(u, v) = c(u, v) - f(u, v)$.

Residual capacity from v to u is $\Delta_f(v, u) = f(u, v)$



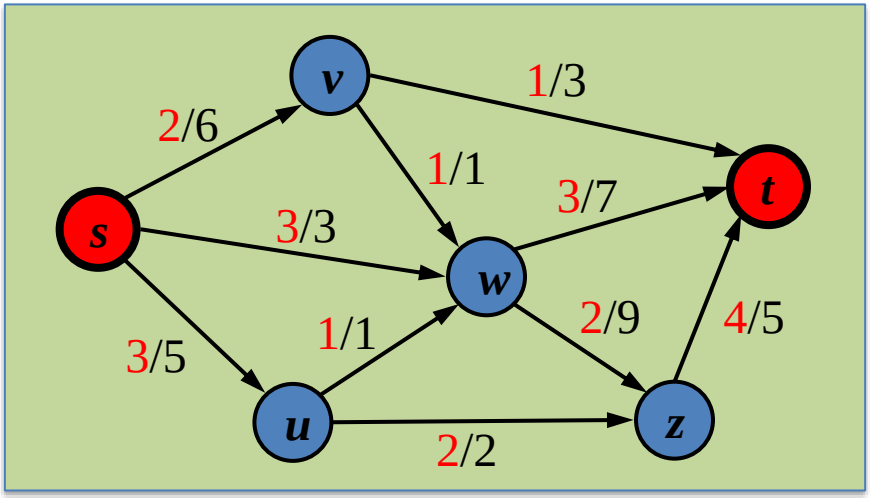
$$\Delta_f(s, v) = 4$$
$$\Delta_f(v, w) = ?$$

Augmenting paths

We are given a network G with edge capacities c and a flow f.
Let (u, v) be an edge from u to v.

Residual capacity from u to v is $\Delta_f(u, v) = c(u, v) - f(u, v)$.

Residual capacity from v to u is $\Delta_f(v, u) = f(u, v)$



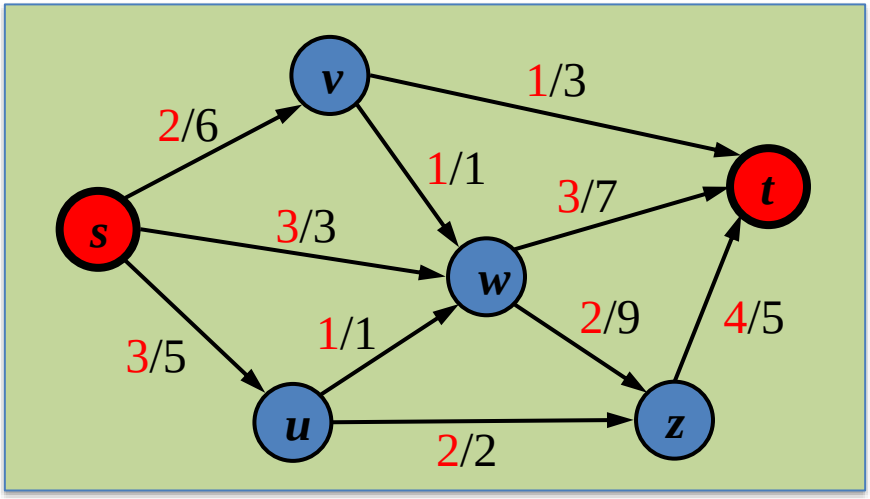
$$\Delta_f(s, v) = 4$$
$$\Delta_f(v, w) = 0$$
$$\Delta_f(w, u) = ?$$

Augmenting paths

We are given a network G with edge capacities c and a flow f.
Let (u, v) be an edge from u to v.

Residual capacity from u to v is $\Delta_f(u, v) = c(u, v) - f(u, v)$.

Residual capacity from v to u is $\Delta_f(v, u) = f(u, v)$



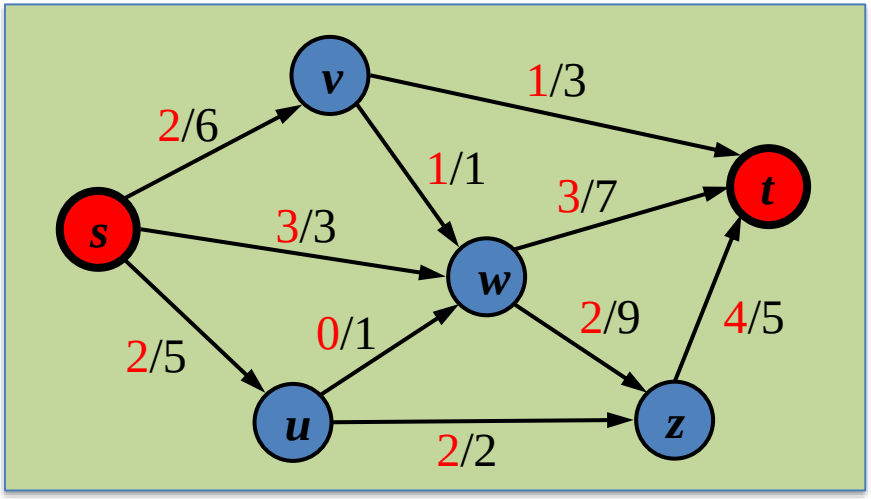
$$\Delta_f(s, v) = 4$$
$$\Delta_f(v, w) = 0$$
$$\Delta_f(w, u) = 1$$

Augmenting paths

We are given a network G with edge capacities c and a flow f.
Let (u, v) be an edge from u to v.

Residual capacity from u to v is $\Delta_f(u, v) = c(u, v) - f(u, v)$.

Residual capacity from v to u is $\Delta_f(v, u) = f(u, v)$



Augmenting path: Path from s to t with positive residual capacities.

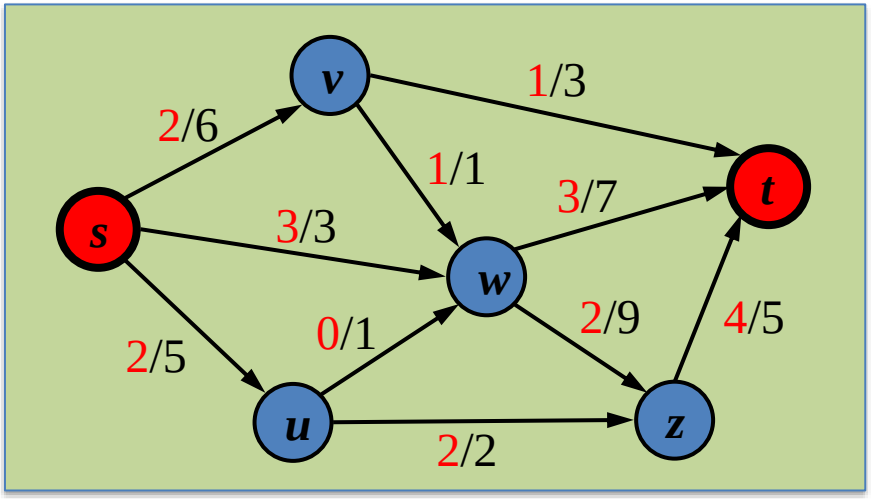
$s \to v \to t$ augmenting path
$s \to u \to w \to v \to t$ augmenting path

Augmenting paths

We are given a network G with edge capacities c and a flow f.
Let (u, v) be an edge from u to v.

Residual capacity from u to v is $\Delta_f(u, v) = c(u, v) - f(u, v)$.

Residual capacity from v to u is $\Delta_f(v, u) = f(u, v)$



Augmenting path: Path from s to t with positive residual capacities.

$s \to v \to t$ augmenting path
$s \to u \to w \to v \to t$ augmenting path

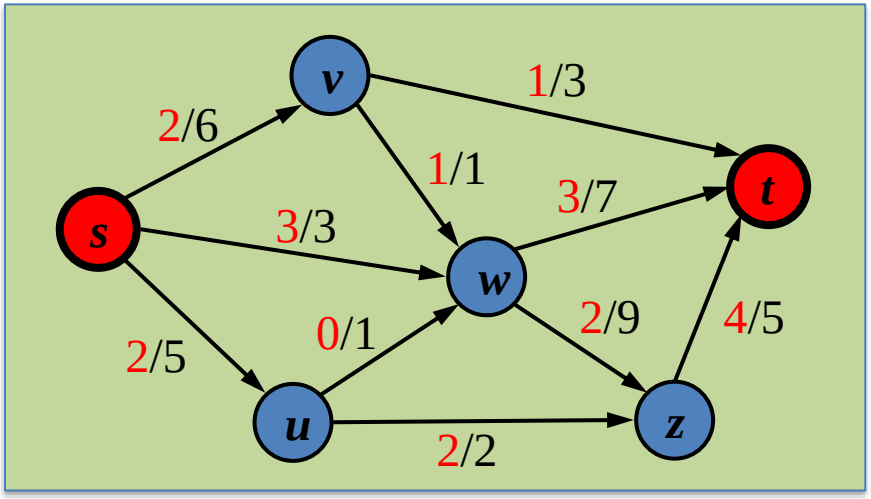
$s \to u \to z \to t$ is **not**

Augmenting paths

We are given a network G with edge capacities c and a flow f.
Let (u, v) be an edge from u to v.

Residual capacity from u to v is $\Delta_f(u, v) = c(u, v) - f(u, v)$.

Residual capacity from v to u is $\Delta_f(v, u) = f(u, v)$



$s \to v \to t$: 2 units of flow can be pushed (min over residual capacities).

$s \to u \to w \to v \to t$: 1 unit of flow can be pushed

$s \to u \to z \to t$: No flow can be pushed

The Ford-Fulkerson Algorithm

Main idea: Repeatedly search for an augmenting path π:

- If there is an augmenting path, augment flow with $\Delta_f(\pi)$ (minimum residual capacity among the edges of π) along the edges of π.

The Ford-Fulkerson Algorithm

Main idea: Repeatedly search for an augmenting path π:

- If there is an augmenting path, augment flow with $\Delta_f(\pi)$ (minimum residual capacity among the edges of π) along the edges of π.
- If there is no augmenting path, terminate.

Remark: You can use DFS (or BFS) to search for an augmenting path.

Running time: ?

The Ford-Fulkerson Algorithm

Main idea: Repeatedly search for an augmenting path π:

- If there is an augmenting path, augment flow with $\Delta_f(\pi)$ (minimum residual capacity among the edges of π) along the edges of π.
- If there is no augmenting path, terminate.

Remark: You can use DFS (or BFS) to search for an augmenting path.

Running time:

Time to search for an augmenting path $\times$ number of updates.

The Ford-Fulkerson Algorithm

Main idea: Repeatedly search for an augmenting path π:

- If there is an augmenting path, augment flow with $\Delta_f(\pi)$ (minimum residual capacity among the edges of π) along the edges of π.
- If there is no augmenting path, terminate.

Remark: You can use DFS (or BFS) to search for an augmenting path.

Running time:

Time to search for an augmenting path $\times$ number of updates.

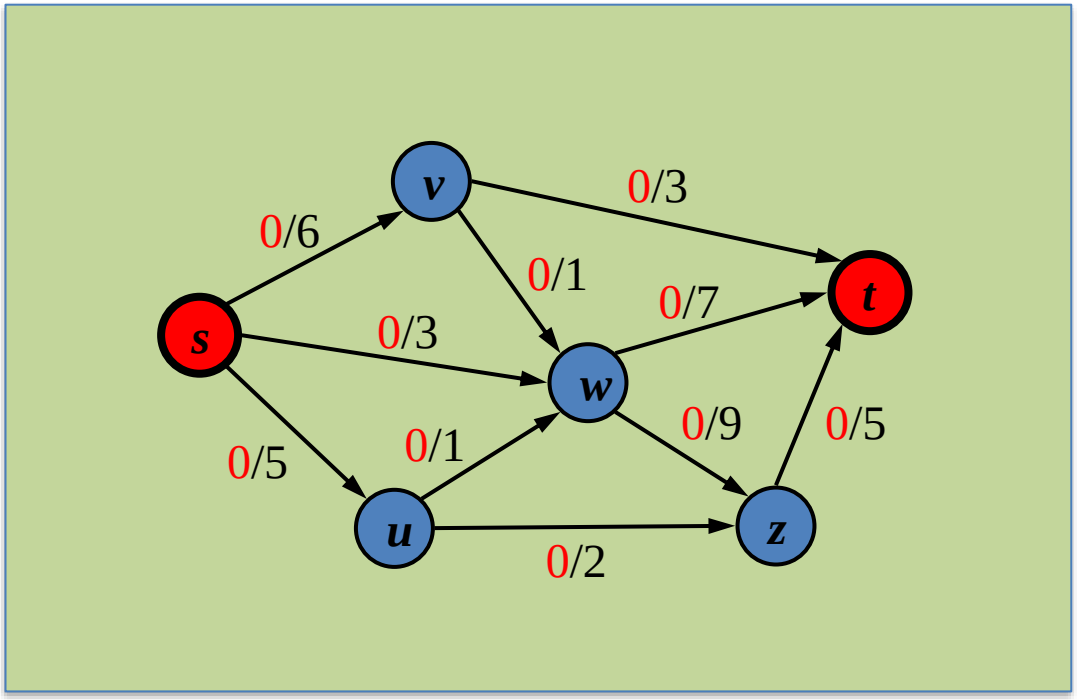
$$\Theta(|V| + |E|) \cdot |f^*|$$

Running time of DFS or BFS

Updates increase flow by 1 unit only

The Ford-Fulkerson Algorithm

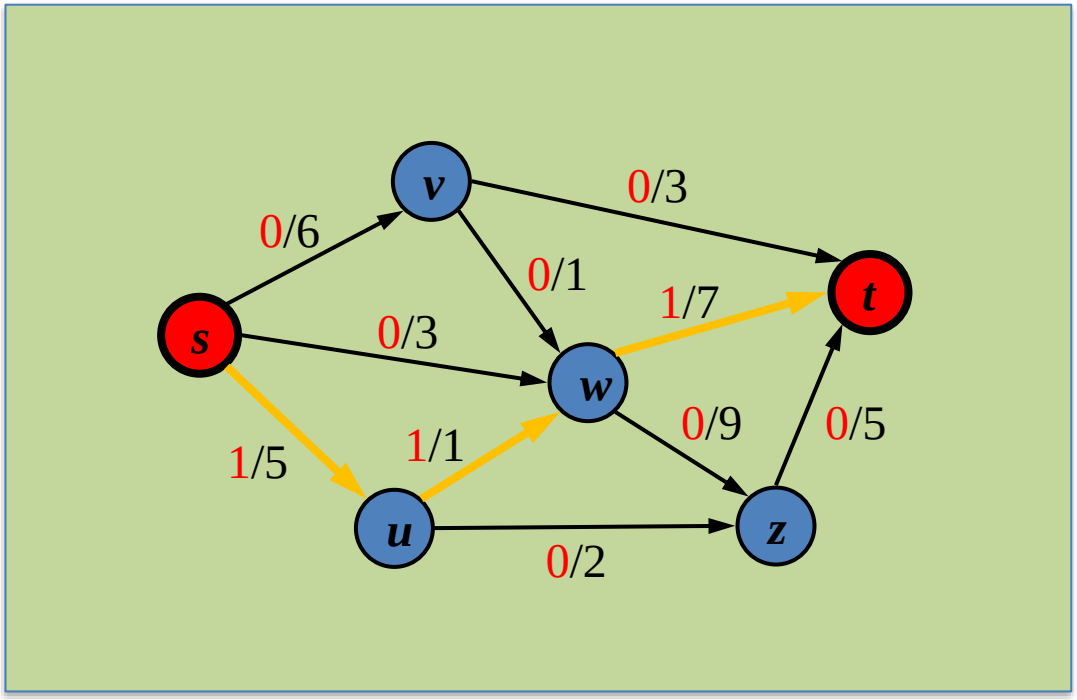
Example:



Total Flow $|f| = 0$

The Ford-Fulkerson Algorithm

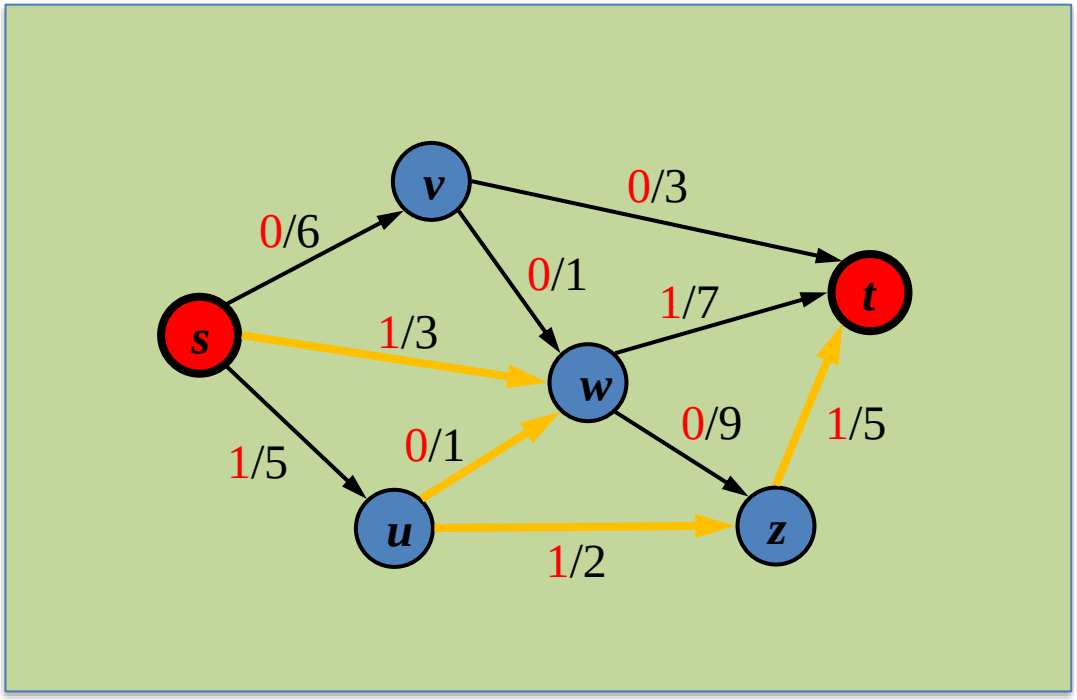
Example:



Total Flow $|f| = 1$

The Ford-Fulkerson Algorithm

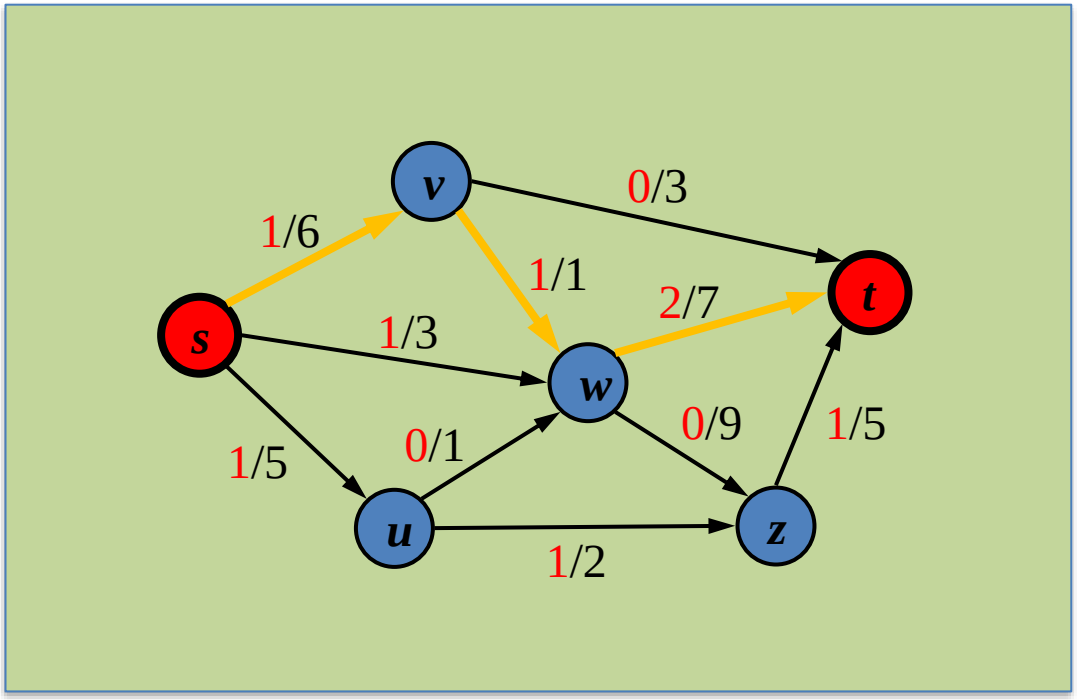
Example:



Total Flow $|f| = 2$

The Ford-Fulkerson Algorithm

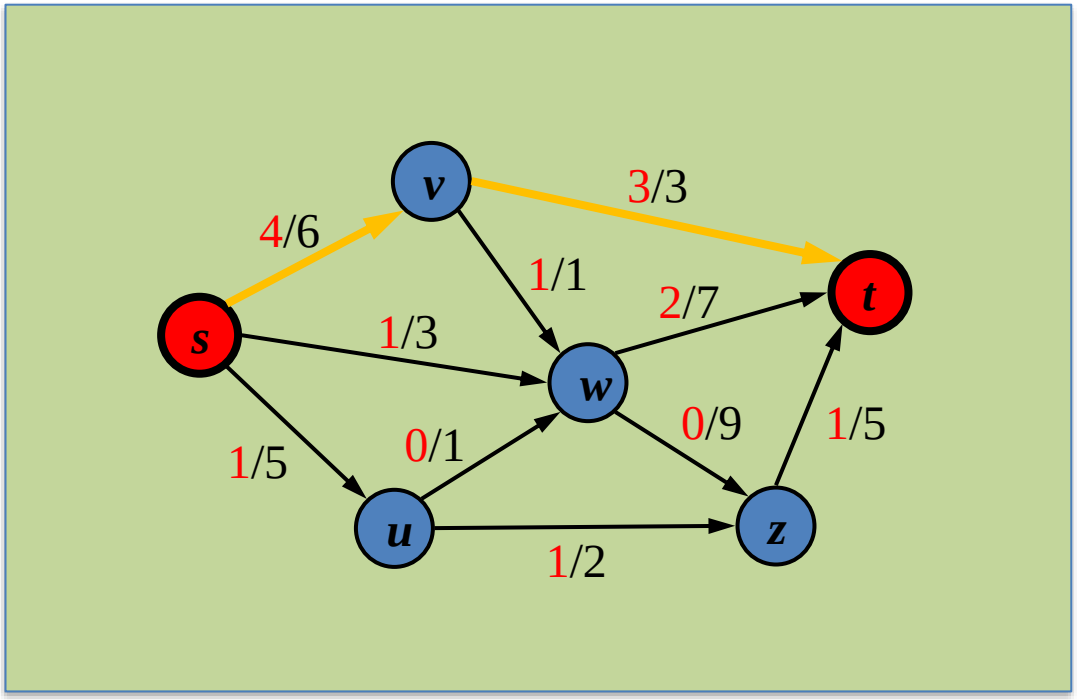
Example:



Total Flow $|f| = 3$

The Ford-Fulkerson Algorithm

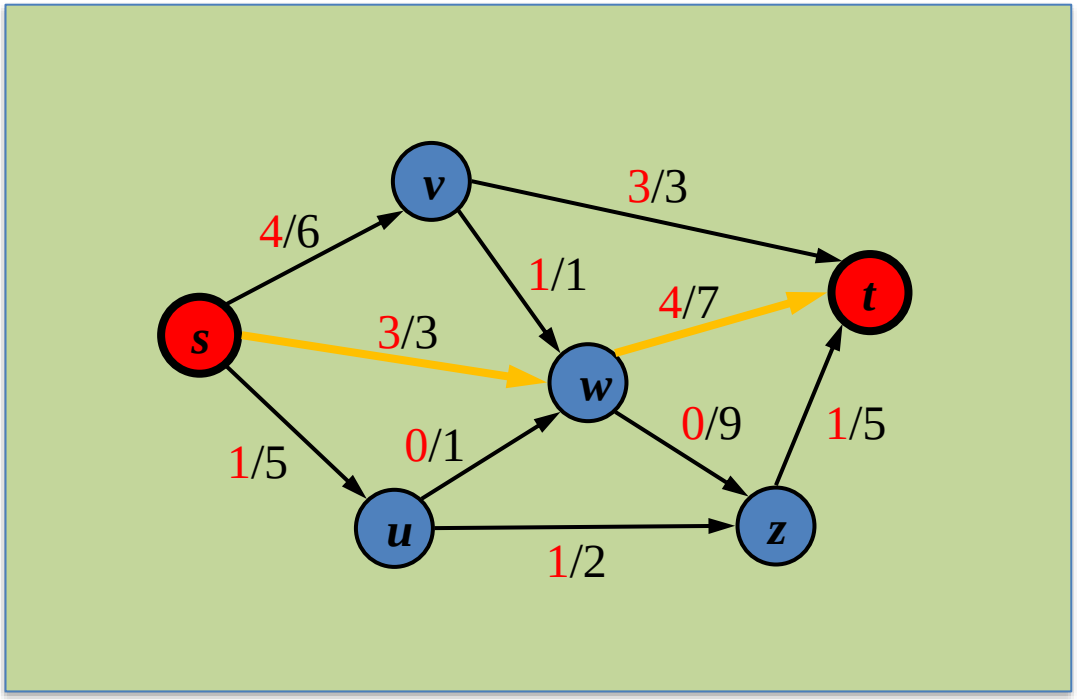
Example:



Total Flow $|f| = 6$

The Ford-Fulkerson Algorithm

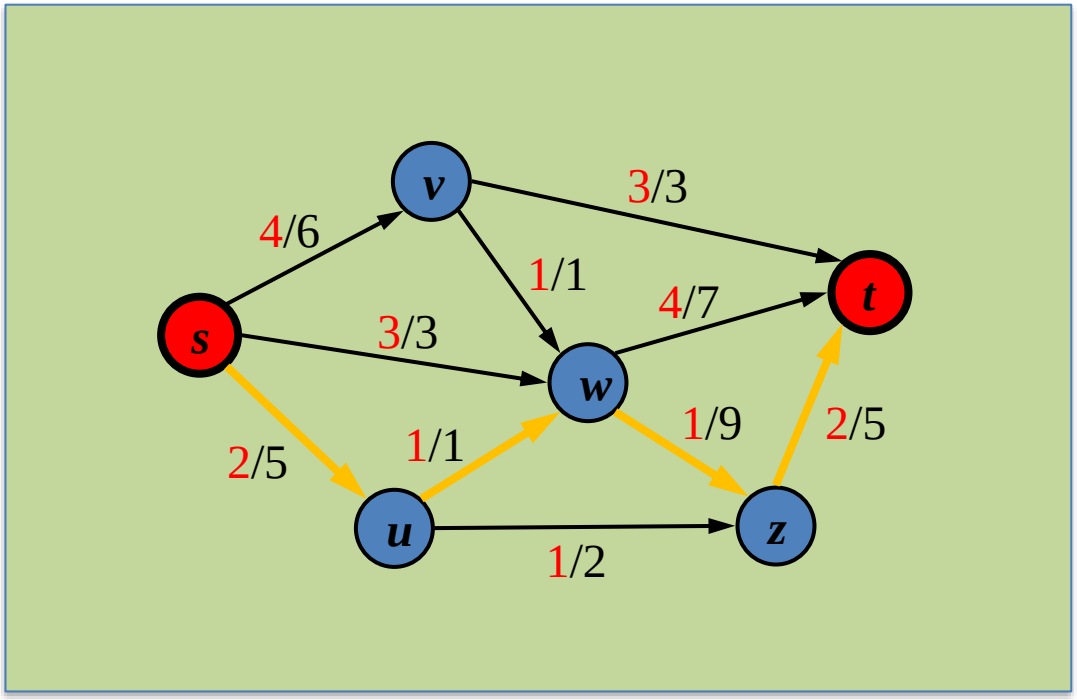
Example:



Total Flow $|f| = 8$

The Ford-Fulkerson Algorithm

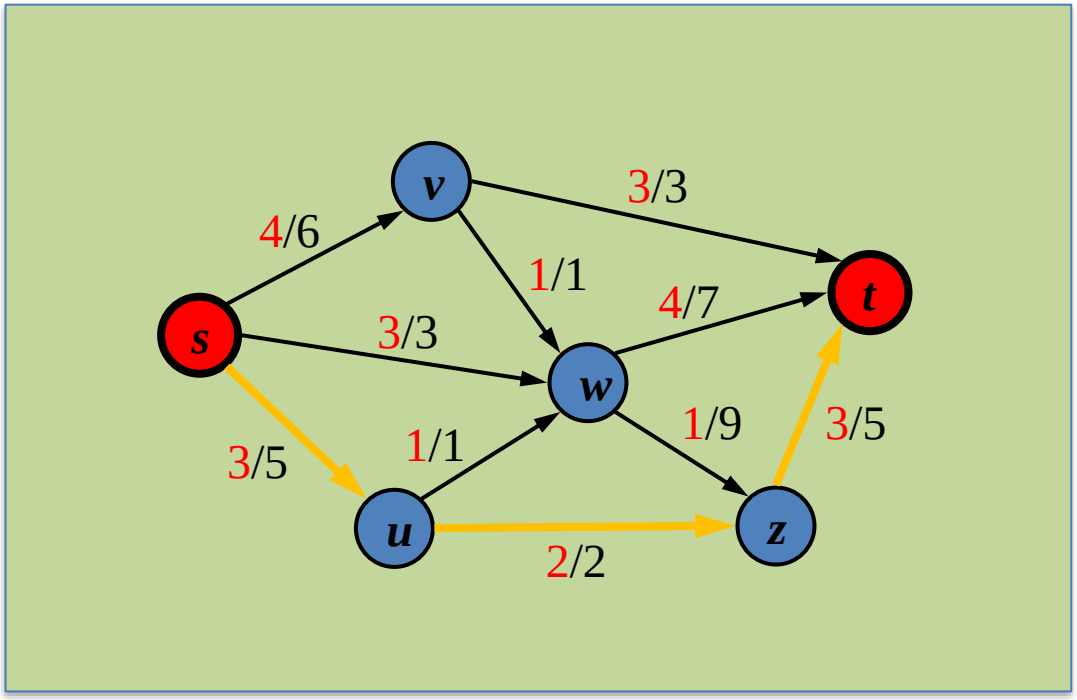
Example:



Total Flow $|f| = 9$

The Ford-Fulkerson Algorithm

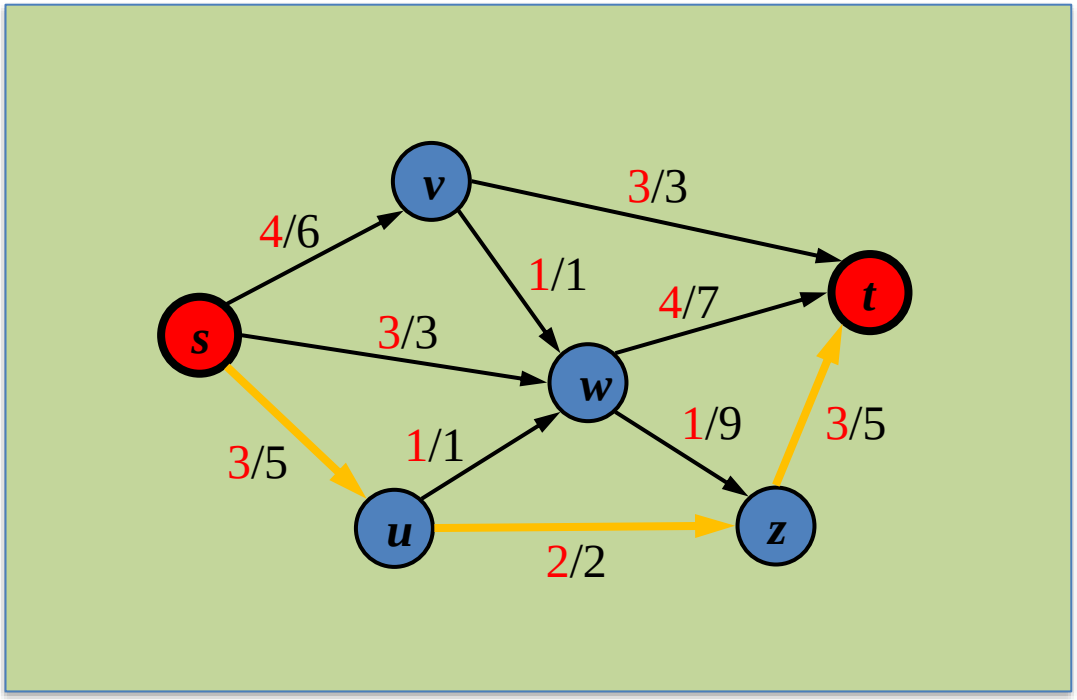
Example:



Total Flow $|f| = 10$

The Ford-Fulkerson Algorithm

Example:



Total Flow $|f| = 10$

No more augmenting paths!

The Ford-Fulkerson Algorithm

Pseudocode:

Algorithm MaxFlowFordFulkerson
Input: Flow network (G, c, s, t)
Output: A maximum flow f

```
for each edge e
    f(e) ← 0
stop ← false
repeat
    traverse G starting at s to find an augmenting path for f
    if an augmenting path π exists then
        // Compute the residual capacity Δ_f(π) of π
        Δ ← +∞
        for each edge e ∈ π do
            if Δ_f(e) < Δ then
                Δ ← Δ_f(e)
        for each edge e ∈ π do   // push Δ = Δ_f(π) units along π
            if e is a forward edge then
                f(e) ← f(e) + Δ
            else
                f(e) ← f(e) − Δ        // e is a backward edge
    else
        stop ← true        // f is a maximum flow
until stop
```

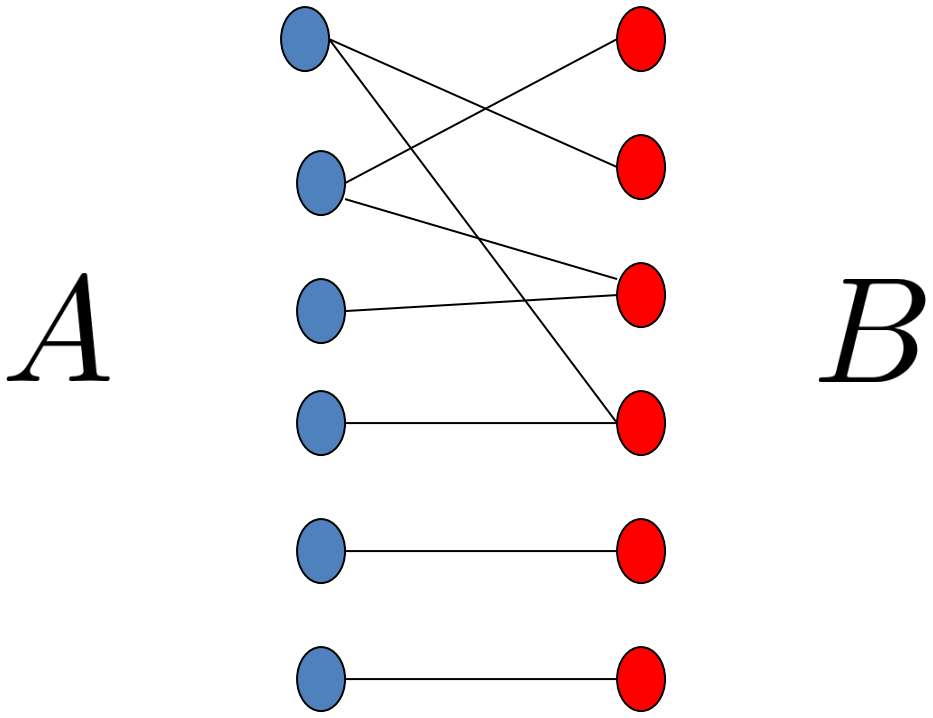
Initialization $f = 0$

Δ: min residual capacity on aug. path

Update flow on aug. path

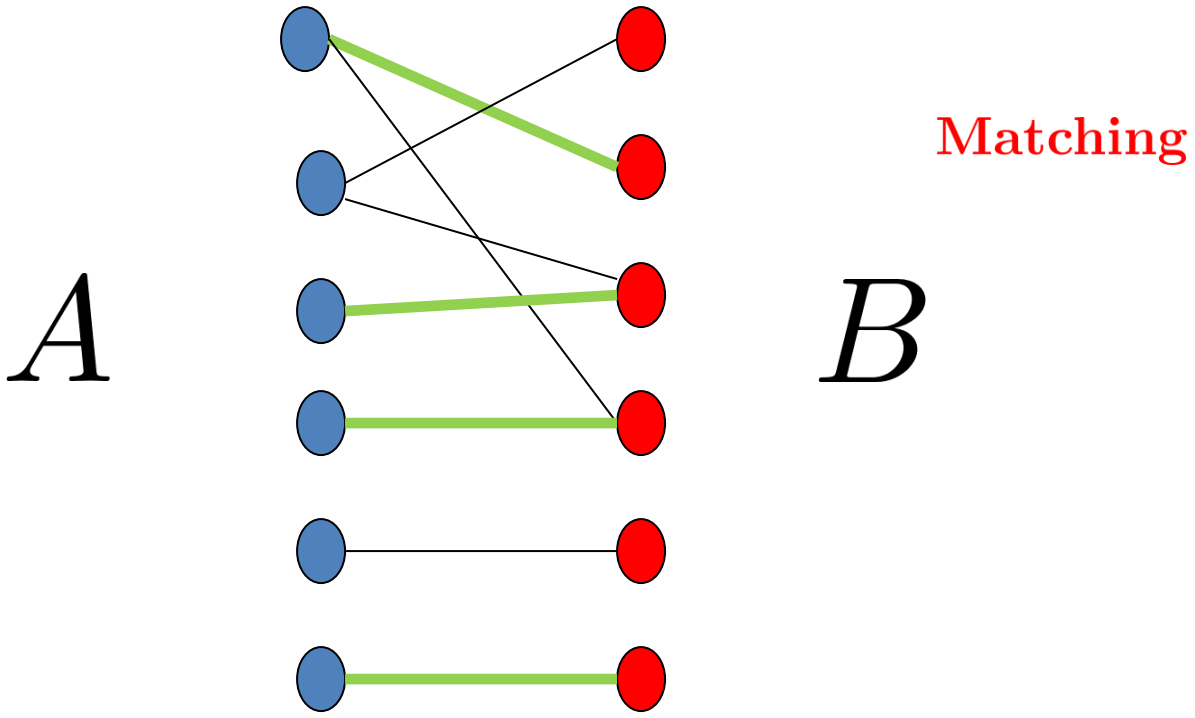
No more aug. paths

Application: Maximum Matching



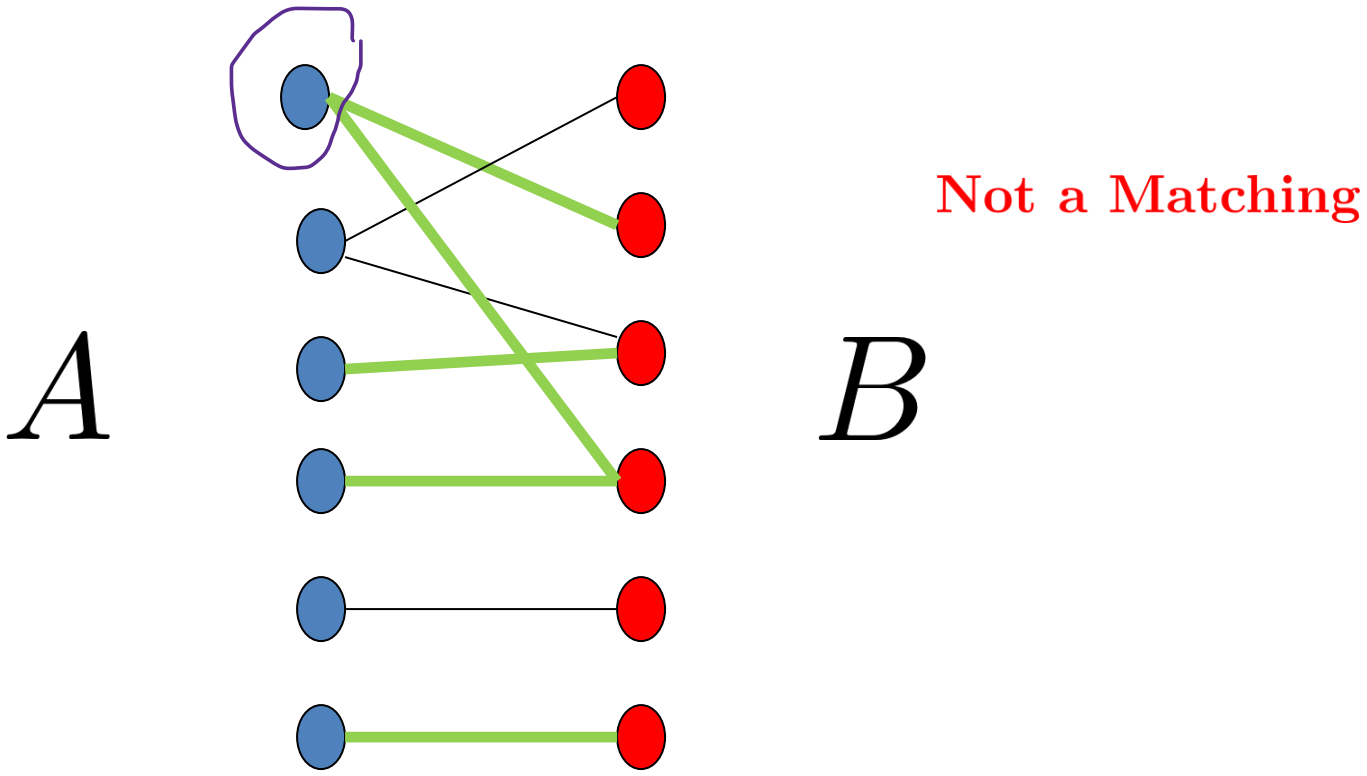
Definition: Given a **bipartite** graph, a **matching** is just a collection of edges that do not share a vertex.

Application: Maximum Matching



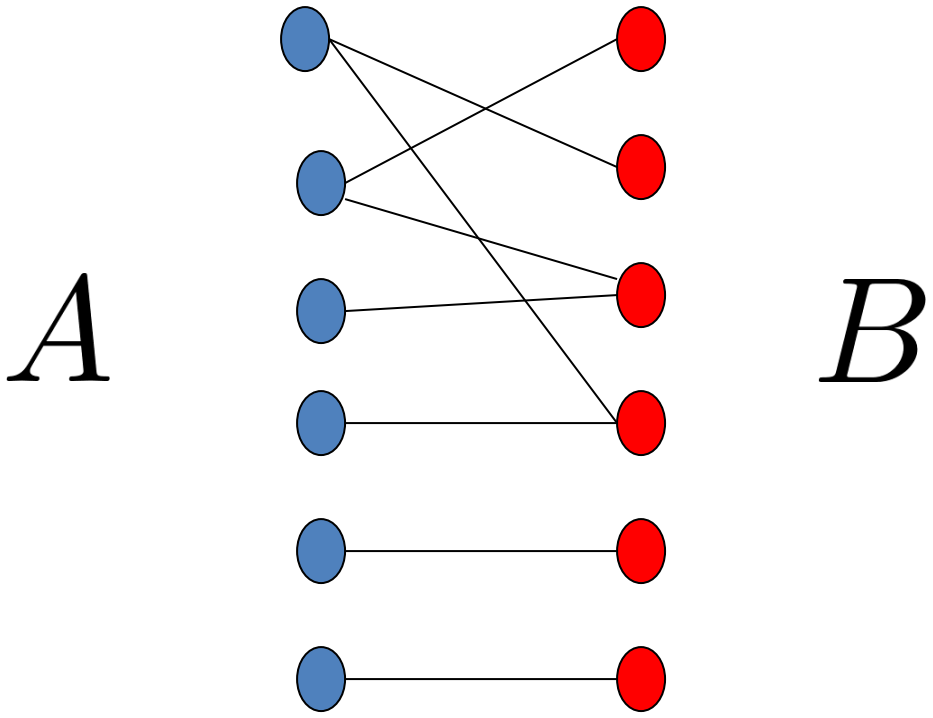
Definition: Given a **bipartite** graph, a **matching** is just a collection of edges that do not share a vertex.

Application: Maximum Matching



Definition: Given a **bipartite** graph, a **matching** is just a collection of edges that do not share a vertex.

Application: Maximum Matching

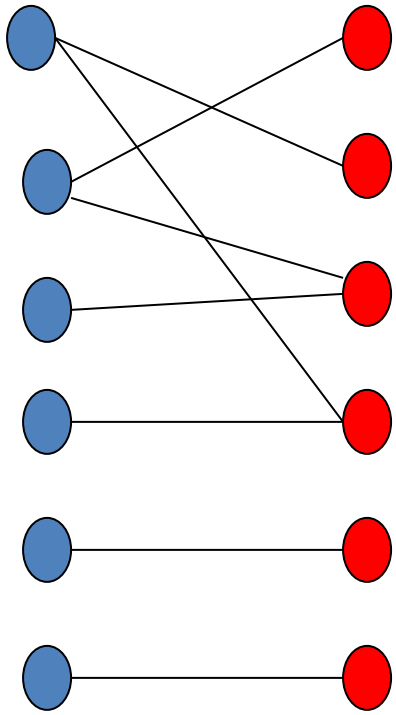


Problem: Given a **bipartite** graph, compute/find a maximum matching.

Application: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

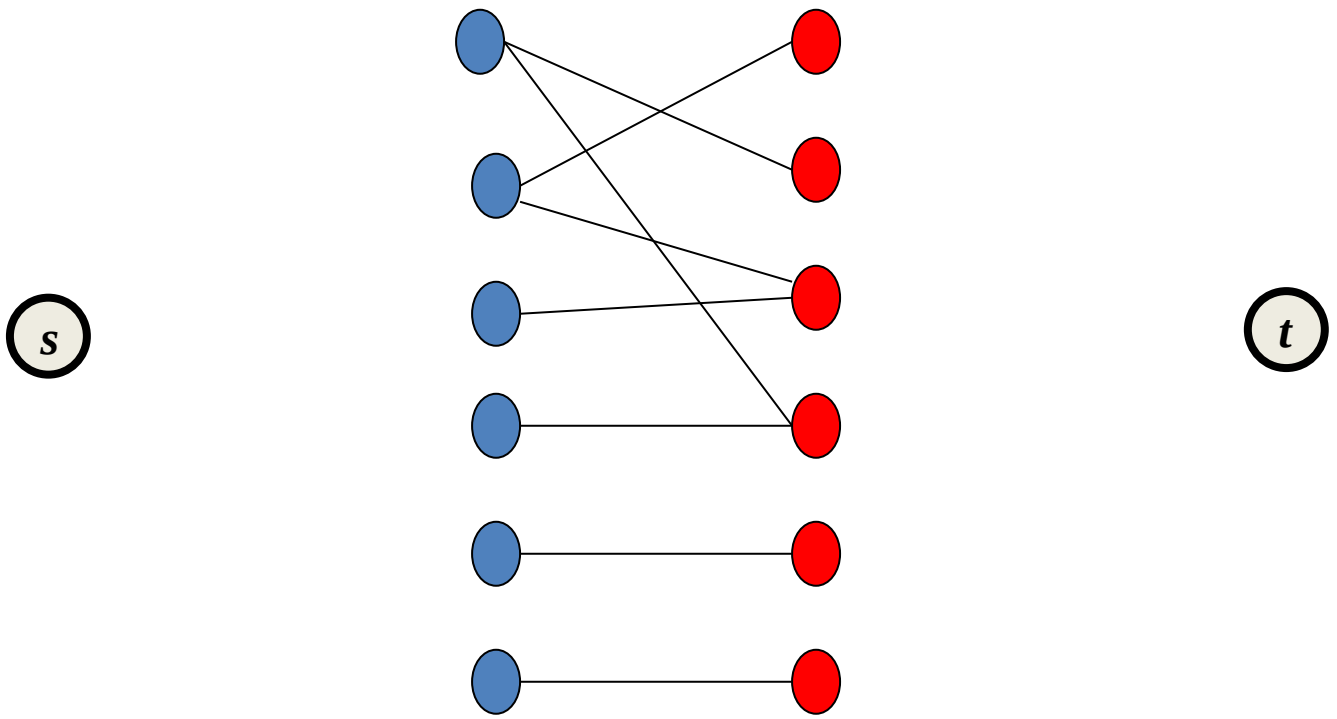
Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and s, t.



Application: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

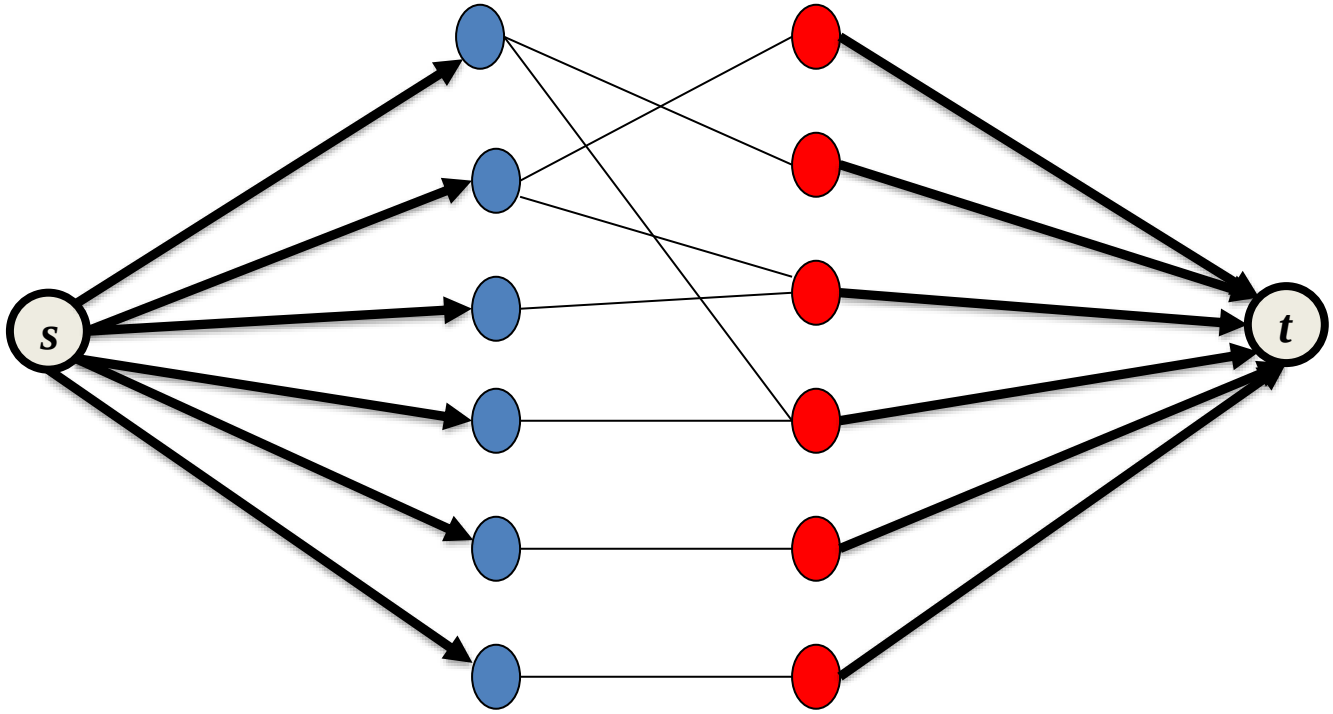
Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and s, t.



Application: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

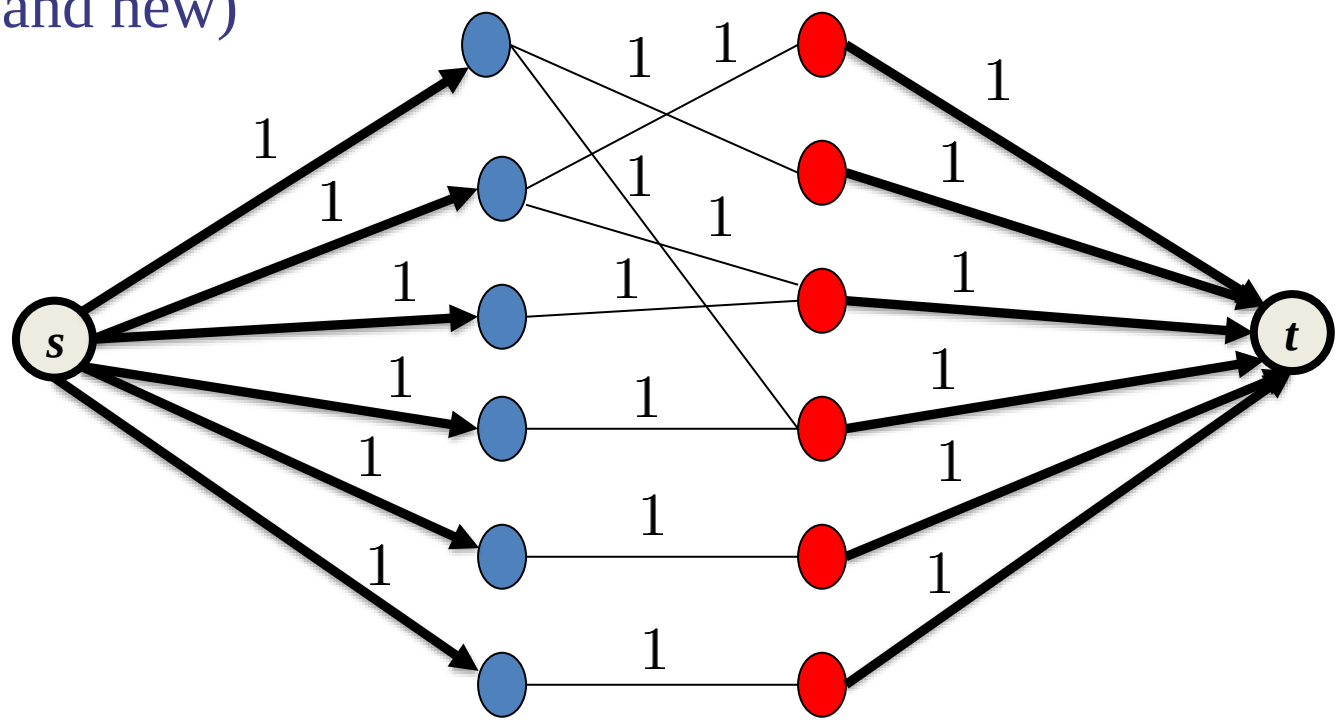
Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and s, t.



Application: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

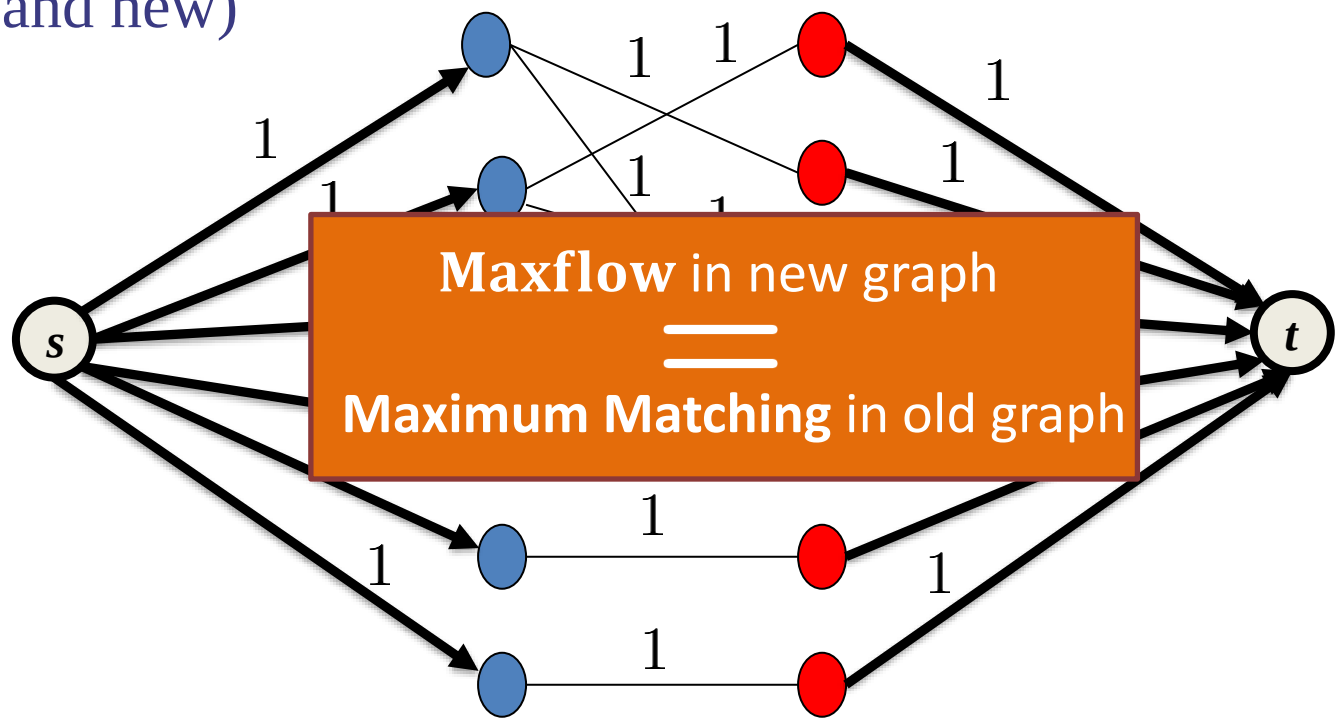
Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and s, t. Put capacity one for all edges (old and new)



Application: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and s, t. Put capacity one for all edges (old and new)



Application: Maximum Matching

Problem: Given a **bipartite** graph, compute/find a maximum matching.

Idea: **Reduce** it to Maxflow problem. To do that, we need a network flow and

Running time $O((V + E) \cdot V)$

ges (old and new)