T&R Team of Algorithm Design

College of Computer Science and Engineering, CQU

Algorithm Analysis & Design

Introduction to Algorithm

Chapter 26:

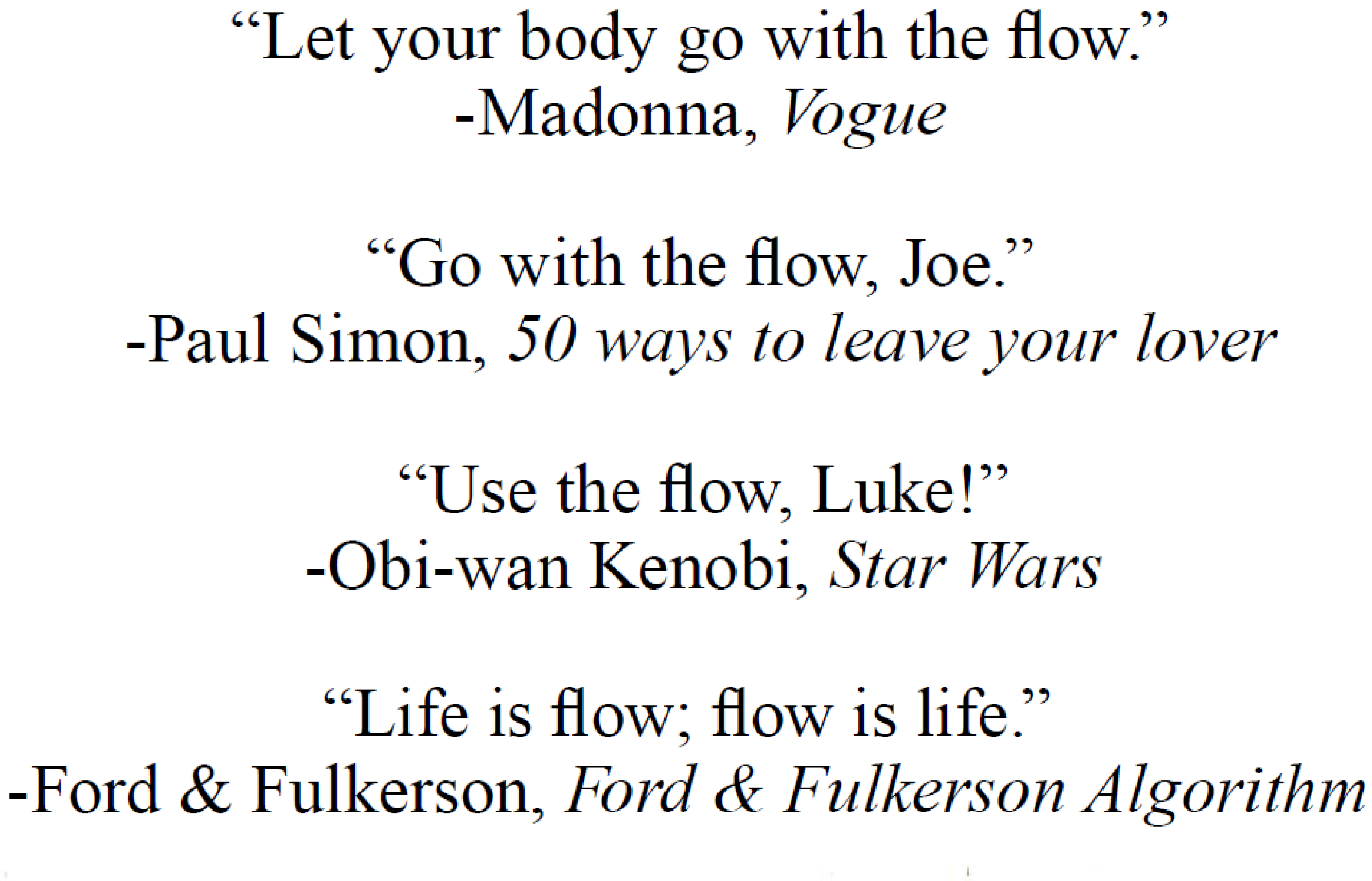
Maximum Flow

Outlines

- Flow networks
- Ford-Fulkerson method

Flow Networks

The Tao of Flow



“Let your body go with the flow.”
-Madonna, *Vogue*

“Go with the flow, Joe.”
-Paul Simon, *50 ways to leave your lover*

“Use the flow, Luke!”
-Obi-wan Kenobi, *Star Wars*

“Life is flow; flow is life.”
-Ford & Fulkerson, *Ford & Fulkerson Algorithm*

The Tao of Flow

"Let your body go with the flow."
-Madonna, *Vogue*

"Go with the flow, Joe."
-Paul Simon, *50 ways to leave your lover*

"Use the flow, Luke!"
-Obi-wan Kenobi, *Star Wars*

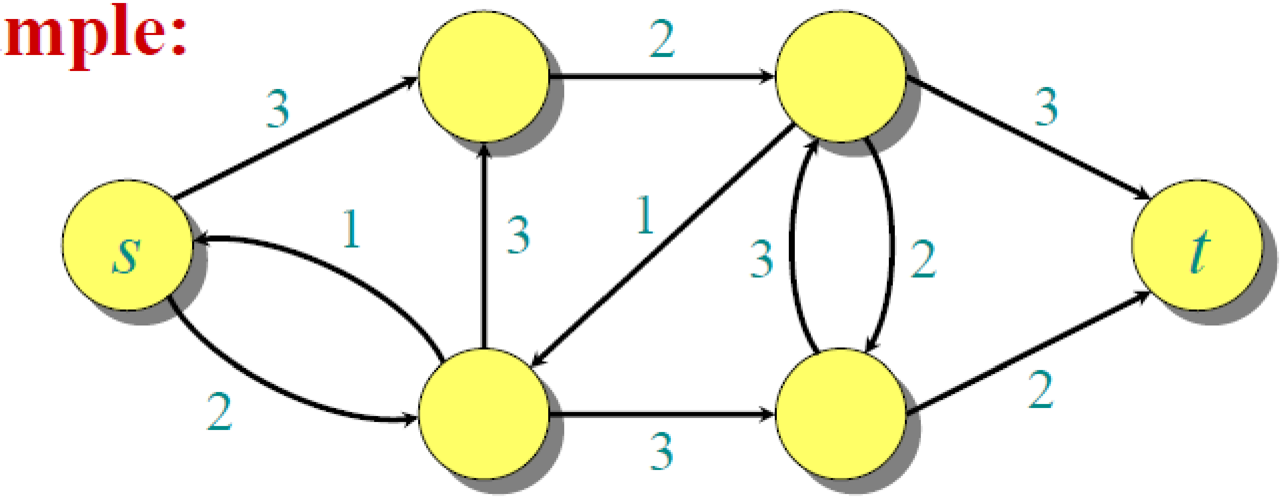
"Life is flow; flow is life."
-Ford & Fulkerson, *Ford & Fulkerson Algorithm*

"Learn flow, or flunk the course"

Flow Network

- digraph $G = (V, E)$
- weights, called capacities on edges $c(u, v)$
- two distinct vertices
    - Source, “s”:
    - Sink, “t”:
- each vertex on some path from source to sink

Example:



Capacity and Flow

- Edge Capacities: $c(u, v)$

  Nonnegative weights on network edges

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

- Flow:

  Function on network edges: $p : V \times V \to \mathbb{R}$

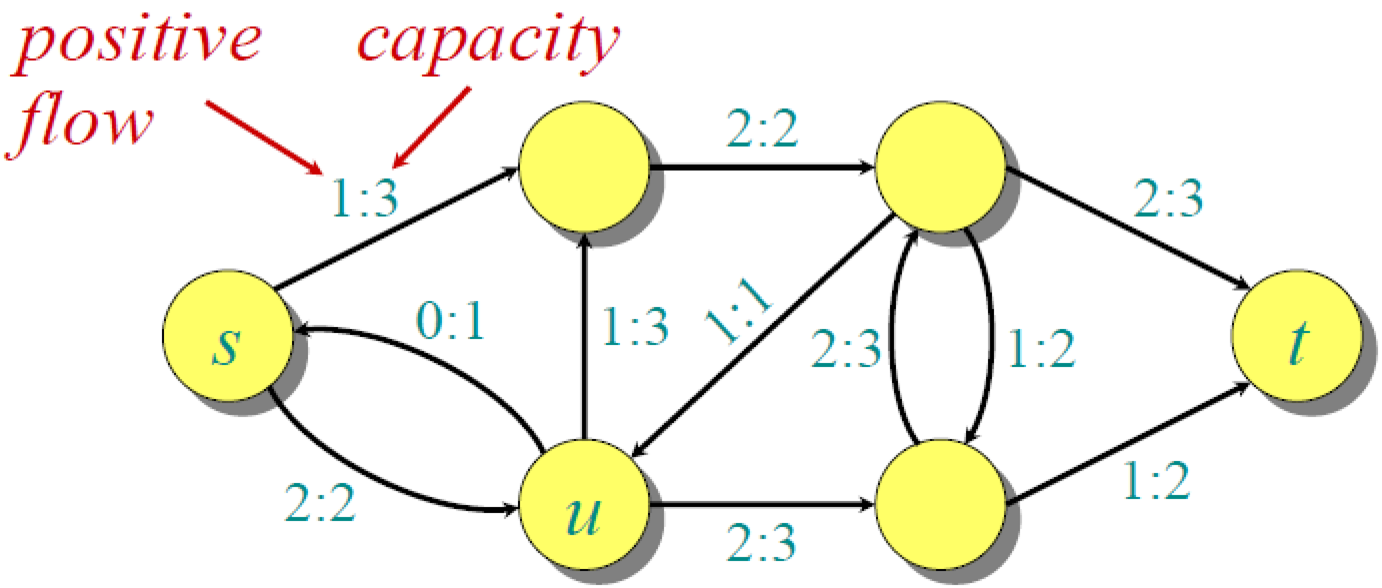
  - ***Capacity constraint:*** For all $u, v \in V$,

    $$0 \le p(u, v) \le c(u, v).$$

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

    $$\sum_{v \in V} p(u,v) - \sum_{v \in V} p(v,u) = 0\,.$$

Capacity and Flow



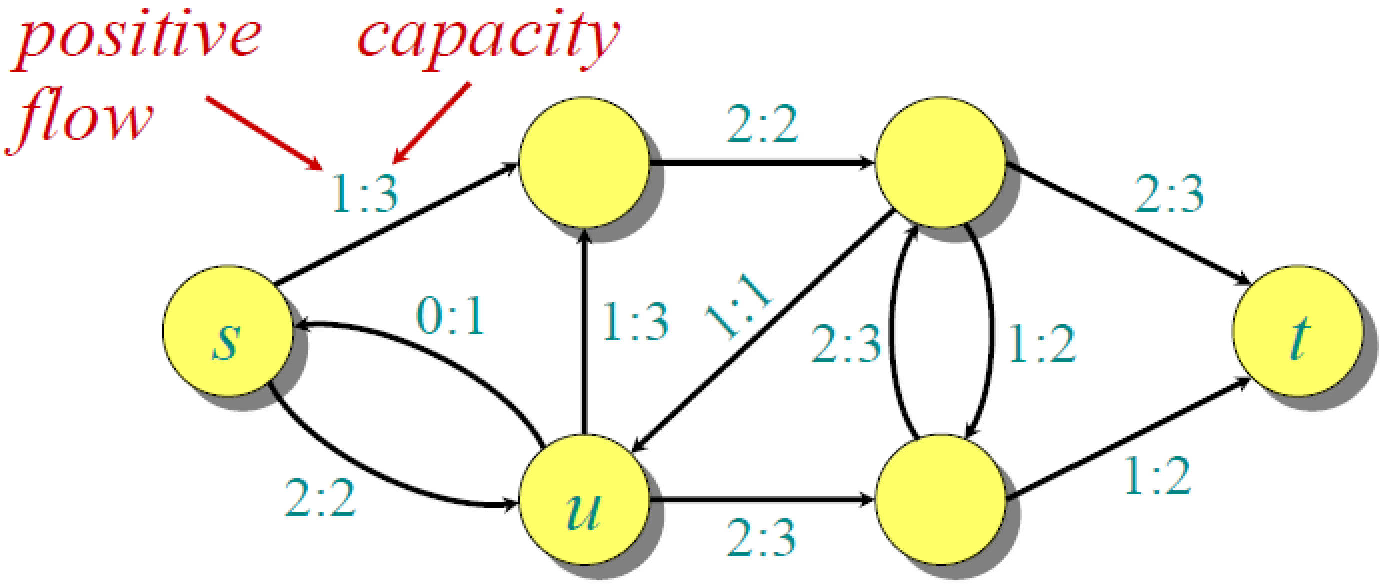
Flow conservation (like Kirchoff's current law):

- Flow into u is $2 + 1 = 3$.
- Flow out of u is $0 + 1 + 2 = 3$.

Flow Value

The ***value*** of a flow is the net flow out of the source:

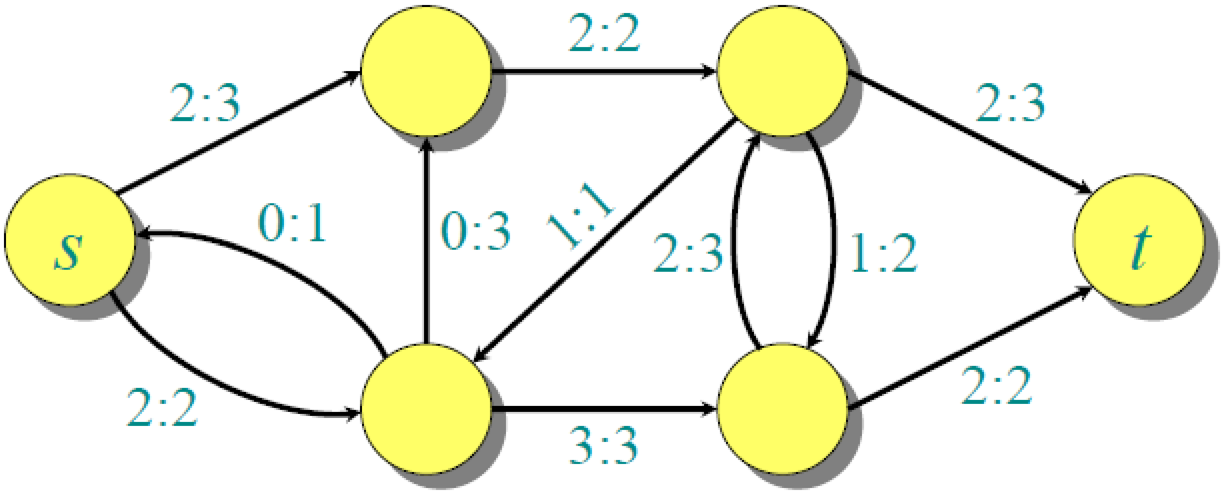
$$\sum_{v \in V} p(s,v) - \sum_{v \in V} p(v,s) .$$



The value of this flow is $1 - 0 + 2 = 3$.

The Maximum-Flow Problem

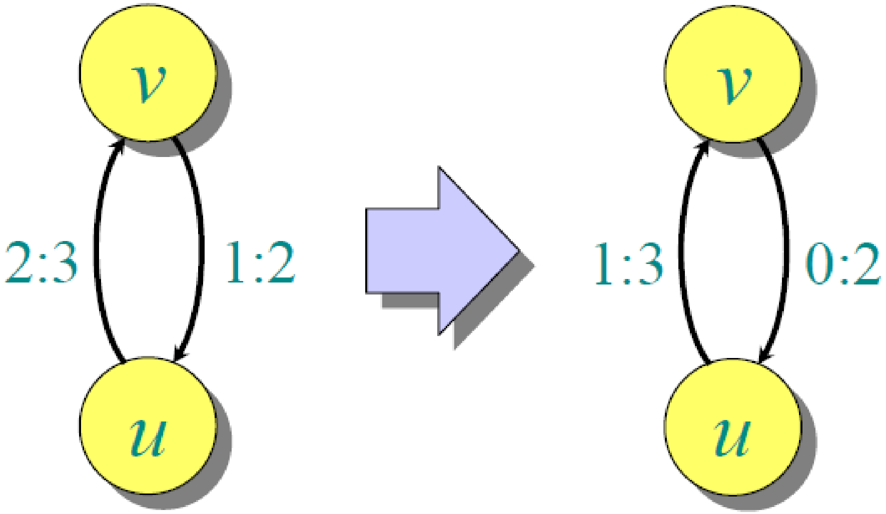
Maximum-flow problem: Given a flow network G, find a flow of maximum value on G.



The value of the maximum flow is 4.

Flow Cancellation

Without loss of generality, positive flow goes either from *u* to *v*, or from *v* to *u*, but not both.



Net flow from *u* to *v* in both cases is 1.

INTUITION: View flow as a *rate*, not a *quantity*.

Net Flow Definitions

IDEA: Work with the net flow between two vertices

Definition. A *(net) flow* on G is a function $f : V \times V \to \mathbb{R}$ satisfying the following:

- ***Capacity constraint:*** For all $u, v \in V$,

$$f(u, v) \le c(u, v).$$

- ***Skew symmetry:*** For all $u, v \in V$,

$$f(u, v) = -f(v, u).$$

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

$$\sum_{v \in V} f(u,v) = 0. \longleftarrow$$

One summation instead of two.

Net Flow Value

Definition. The ***value*** of a flow f, denoted by $|f|$, is given by

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

Implicit summation notation

- **Example** — flow conservation:
  $f(u, V) = 0$ for all $u \in V - \{s, t\}$.

Simple Properties of Net Flow

Lemma.

- $f(X, X) = 0,$
- $f(X, Y) = -f(Y, X),$
- $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ if $X \cap Y = \emptyset.$

Simple Properties of Net Flow

Lemma.

- $f(X, X) = 0$,

  (Proof). $\sum_{x\in X}\sum_{y\in X} f(x,y) + \sum_{y\in X}\sum_{x\in X} f(y,x) = 0$

- $f(X, Y) = -f(Y, X)$,

- $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ if $X \cap Y = \emptyset$.

Simple Properties of Net Flow

Lemma.

- $f(X, X) = 0$,

  (Proof). $\sum_{x \in X} \sum_{y \in X} f(x, y) + \sum_{y \in X} \sum_{x \in X} f(y, x) = 0$

- $f(X, Y) = -f(Y, X)$,

- $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ if $X \cap Y = \emptyset$.

Simple Properties of Net Flow

Lemma.

- $f(X, X) = 0,$

  (Proof). $\sum_{x\in X}\sum_{y\in X} f(x,y)+\sum_{y\in X}\sum_{x\in X} f(y,x) = 0$

- $f(X, Y) = -f(Y, X),$

  (Proof). $\sum_{x\in X}\sum_{y\in Y}(f(x,y) + f(y,x)) = 0$

- $f(X\cup Y, Z) = f(X, Z) + f(Y, Z)$ if $X\cap Y = \emptyset.$

Simple Properties of Net Flow

Lemma.

- $f(X, X) = 0$,

  (Proof). $\sum_{x \in X} \sum_{y \in X} f(x, y) + \sum_{y \in X} \sum_{x \in X} f(y, x) = 0$

- $f(X, Y) = -f(Y, X)$,

  (Proof). $\sum_{x \in X} \sum_{y \in Y} (f(x, y) + f(y, x)) = 0$

- $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ if $X \cap Y = \emptyset$.

Simple Properties of Net Flow

Lemma.

- $f(X, X) = 0$,

  (Proof). $\sum_{x\in X}\sum_{y\in X} f(x,y)+\sum_{y\in X}\sum_{x\in X} f(y,x) = 0$

- $f(X, Y) = -f(Y, X)$,

  (Proof). $\sum_{x\in X}\sum_{y\in Y}(f(x,y) + f(y,x)) = 0$

- $f(X\cup Y, Z) = f(X, Z) + f(Y, Z)$ if $X\cap Y = \varnothing$.

  (Proof). Exercise

Simple Properties of Net Flow

Theorem. $|f| = f(V, t)$.

Proof.

Simple Properties of Net Flow

Theorem. $|f| = f(V, t)$.

Proof.

$$|f| = f(s, V)$$

Simple Properties of Net Flow

Theorem. $|f| = f(V, t)$.

Proof.

$$|f| = f(s, V)$$

$$= f(V, V) - f(V - s, V)$$

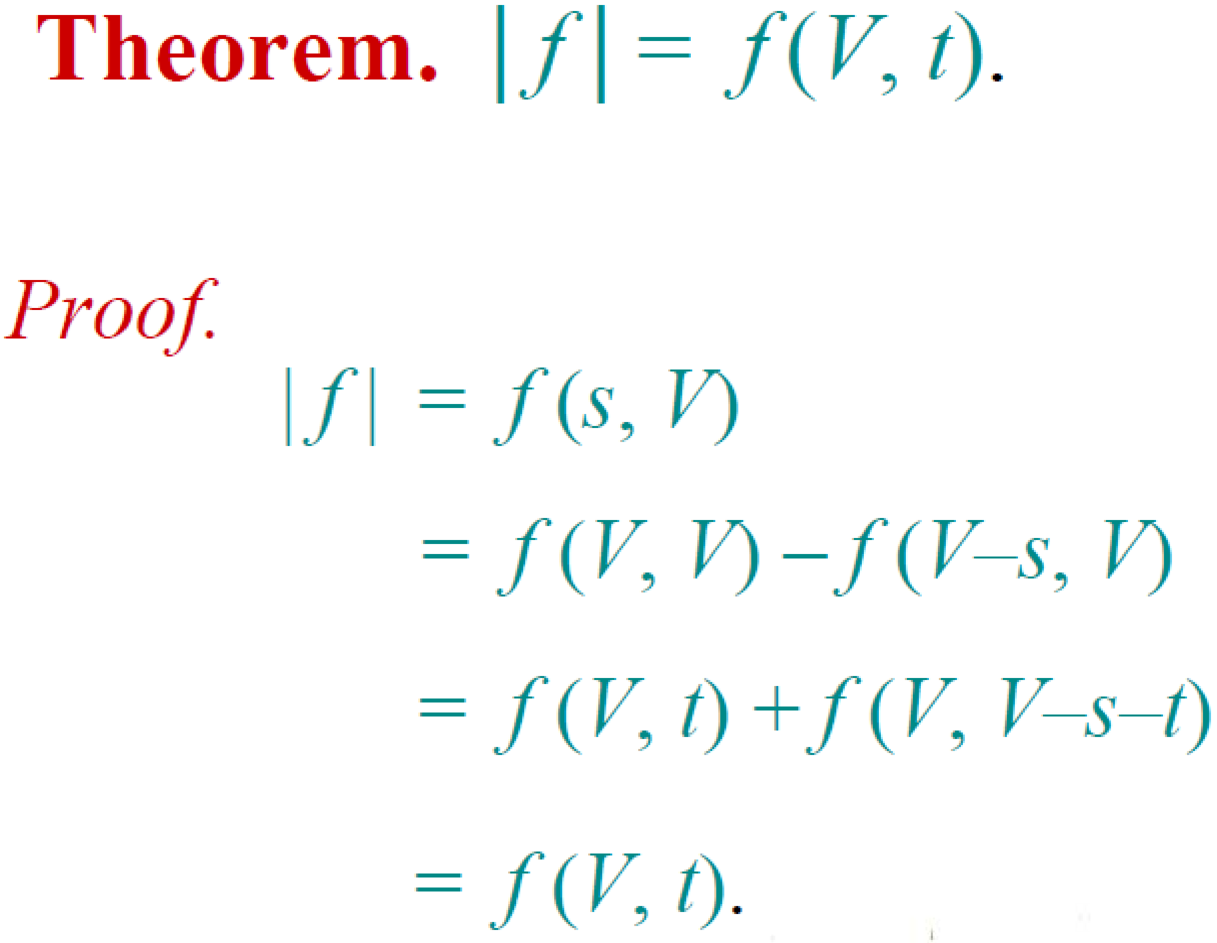
Simple Properties of Net Flow

Theorem. $|f| = f(V, t)$.

Proof.

$$
\begin{aligned}
|f| &= f(s, V) \\
&= f(V, V) - f(V-s, V) \\
&= f(V, t) + f(V, V-s-t)
\end{aligned}
$$

Simple Properties of Net Flow

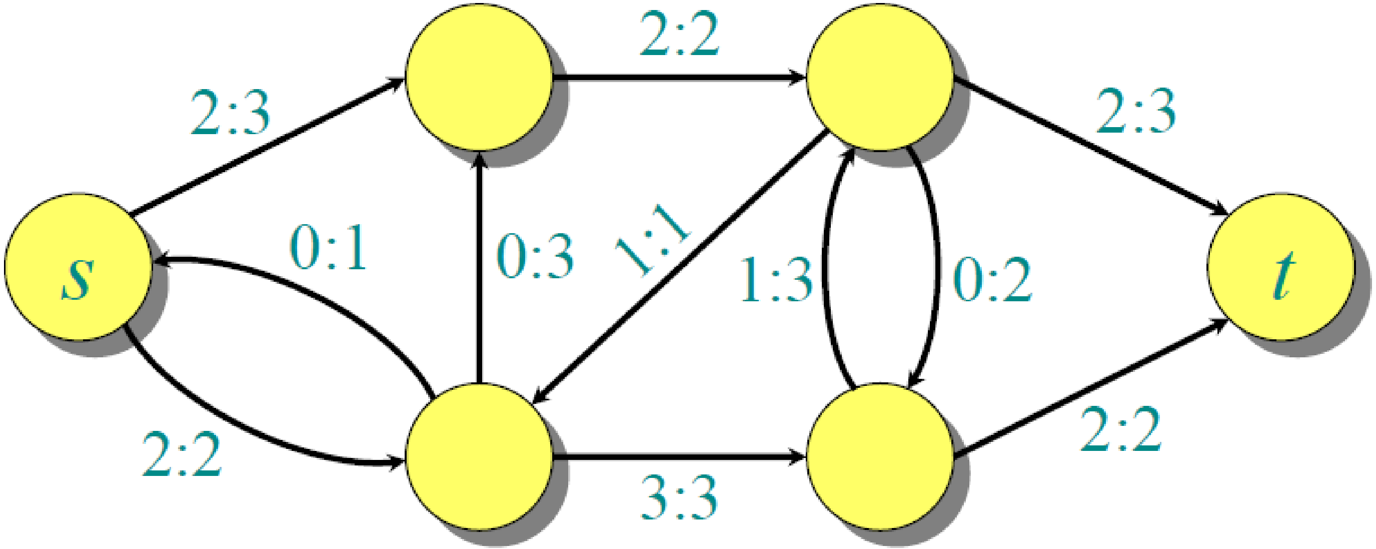


Theorem. $|f| = f(V, t)$.

Proof.

$$\begin{aligned} |f| &= f(s, V) \\ &= f(V, V) - f(V-s, V) \\ &= f(V, t) + f(V, V-s-t) \\ &= f(V, t). \end{aligned}$$

Net Flow into Sink

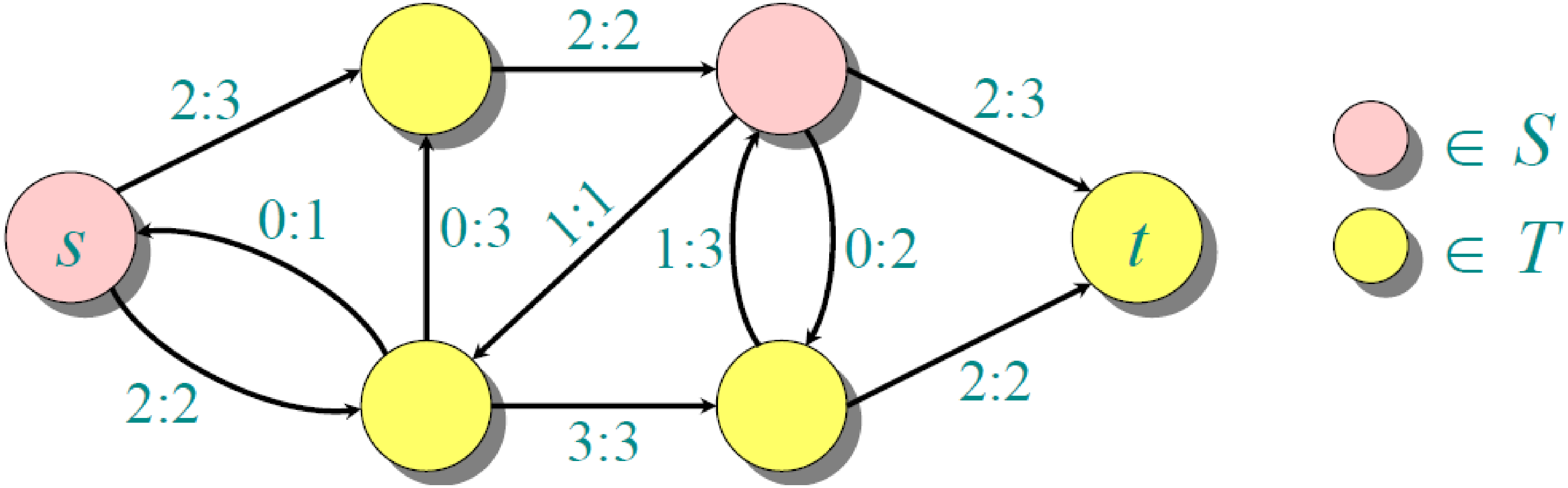


$|f| = f(s, V) = 4$

$f(V, t) = 4$

Cut

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



flow across the cut

$$f(S, T) = (2 + 2) + (-2 + 1 - 1 + 2)$$
$$= 4$$

Flow of A Cut

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

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

Flow of A Cut

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

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

Proof. $f(S, T) = f(S, V) - f(S, S)$

Flow of A Cut

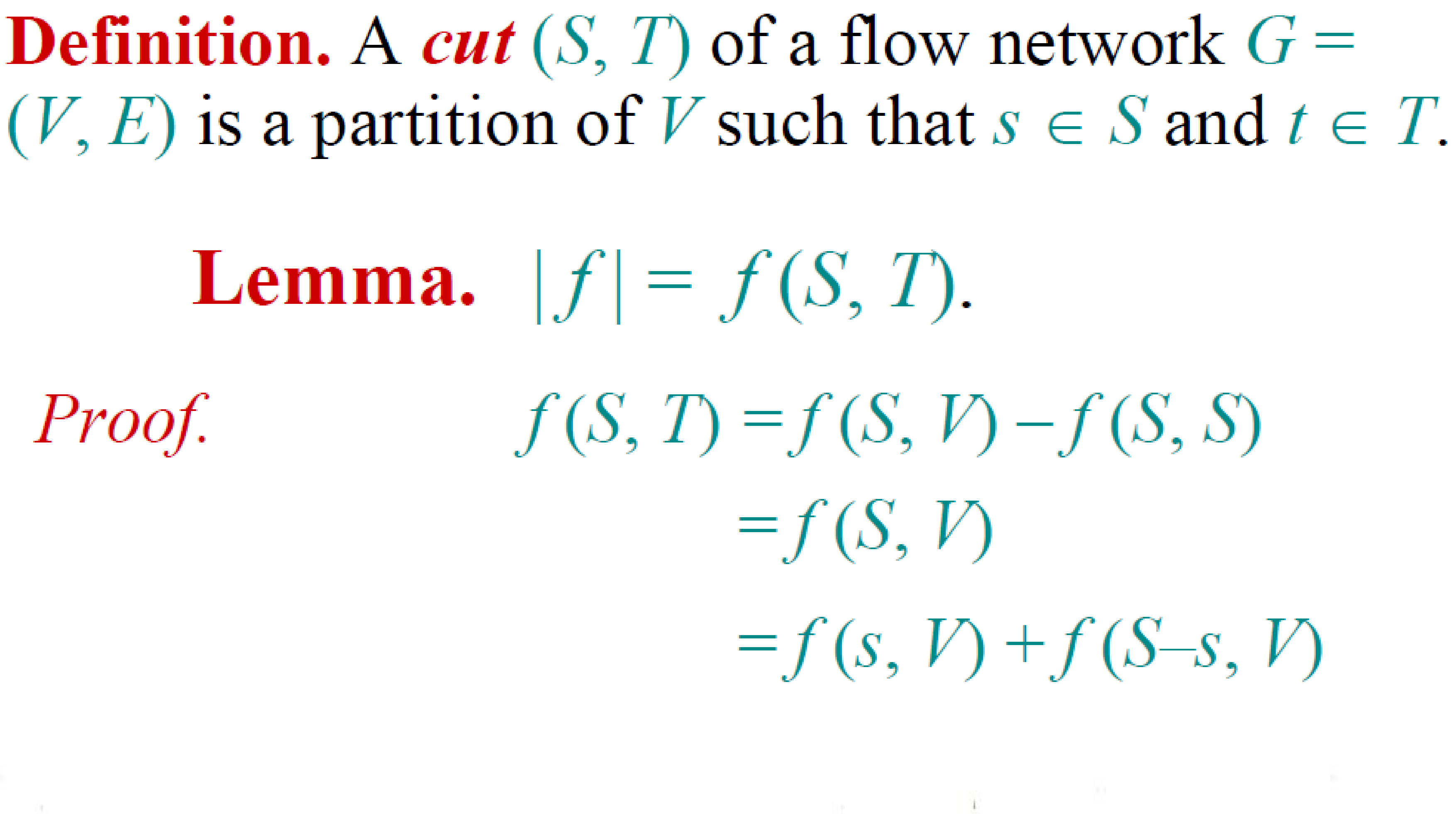
Definition. A ***cut*** (S, T) of a flow network $G = (V, E)$ is a partition of V such that $s \in S$ and $t \in T$.

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

Proof.

$$\begin{aligned} f(S, T) &= f(S, V) - f(S, S) \\ &= f(S, V) \end{aligned}$$

Flow of A Cut



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

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

Proof.

$$\begin{aligned} f(S, T) &= f(S, V) - f(S, S) \\ &= f(S, V) \\ &= f(s, V) + f(S-s, V) \end{aligned}$$

Flow of A Cut

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

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

Proof.

$$
\begin{aligned}
f(S, T) &= f(S, V) - f(S, S) \\
&= f(S, V) \\
&= f(s, V) + f(S-s, V) \\
&= f(s, V)
\end{aligned}
$$

Flow of A Cut

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

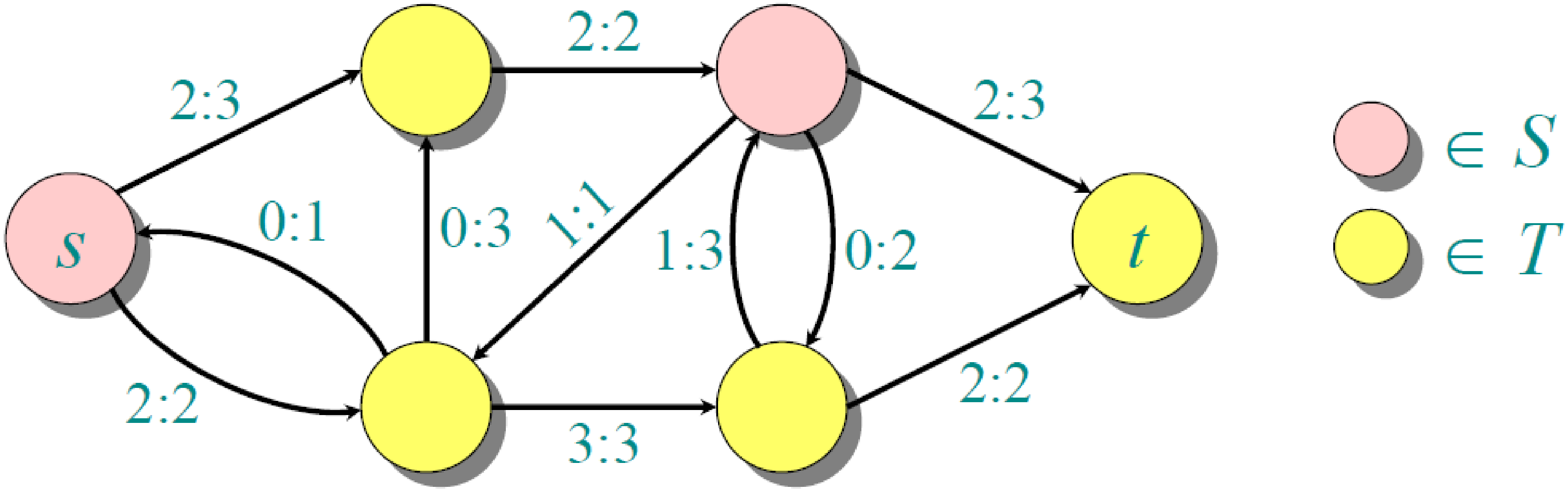
Lemma. $|f| = f(S, T)$.

Proof.

$$\begin{aligned} f(S, T) &= f(S, V) - f(S, S) \\ &= f(S, V) \\ &= f(s, V) + f(S-s, V) \\ &= f(s, V) \\ &= |f|. \end{aligned}$$

Capacity of A Cut

Definition. The ***capacity of a cut*** (S, T) is $c(S, T)$.



$$c(S, T) = (3 + 2) + (1 + 2 + 3)$$
$$= 11$$

Upper Bound on Flow Value

Theorem. The value of any flow is bounded by the capacity of any cut.

Upper Bound on Flow Value

Theorem. The value of any flow is bounded by the capacity of any cut.

Proof.

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

Upper Bound on Flow Value

Theorem. The value of any flow is bounded by the capacity of any cut.

Proof.

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

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

Upper Bound on Flow Value

Theorem. The value of any flow is bounded by the capacity of any cut.

Proof.

$$
\begin{aligned}
|f| &= f(S,T) \\
&= \sum_{u\in S}\sum_{v\in T} f(u,v) \\
&\le \sum_{u\in S}\sum_{v\in T} c(u,v)
\end{aligned}
$$

Upper Bound on Flow Value

Theorem. The value of any flow is bounded by the capacity of any cut.

Proof.

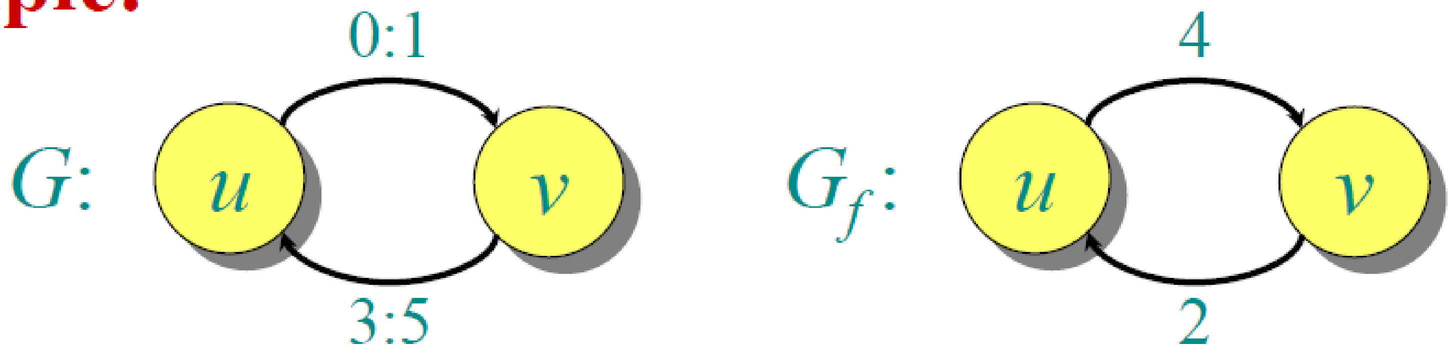
$$
\begin{aligned}
|f| &= f(S,T) \\
&= \sum_{u\in S}\sum_{v\in T} f(u,v) \\
&\le \sum_{u\in S}\sum_{v\in T} c(u,v) \\
&= c(S,T)\,.
\end{aligned}
$$

Residual Network

Definition. Let f be a flow on $G = (V, E)$. The ***residual network*** $G_f(V, E_f)$ is the graph with strictly positive ***residual capacities***

$$c_f(u, v) = c(u, v) - f(u, v) > 0.$$

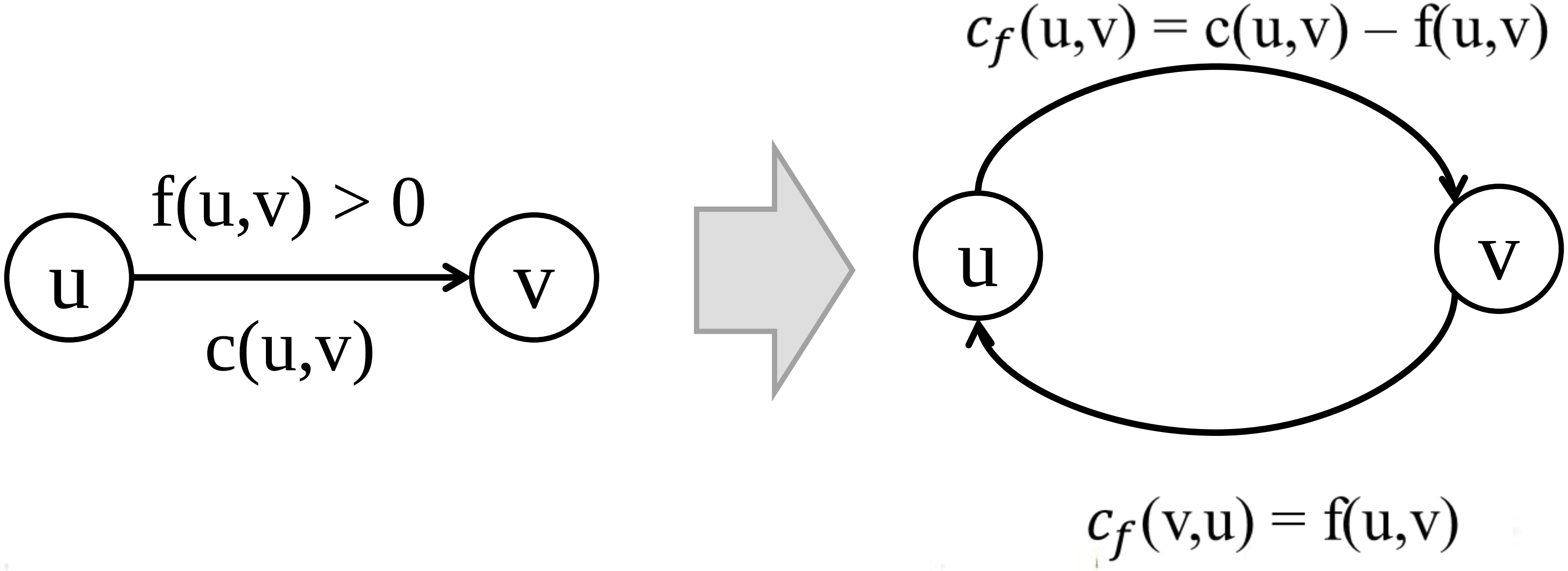
Example:



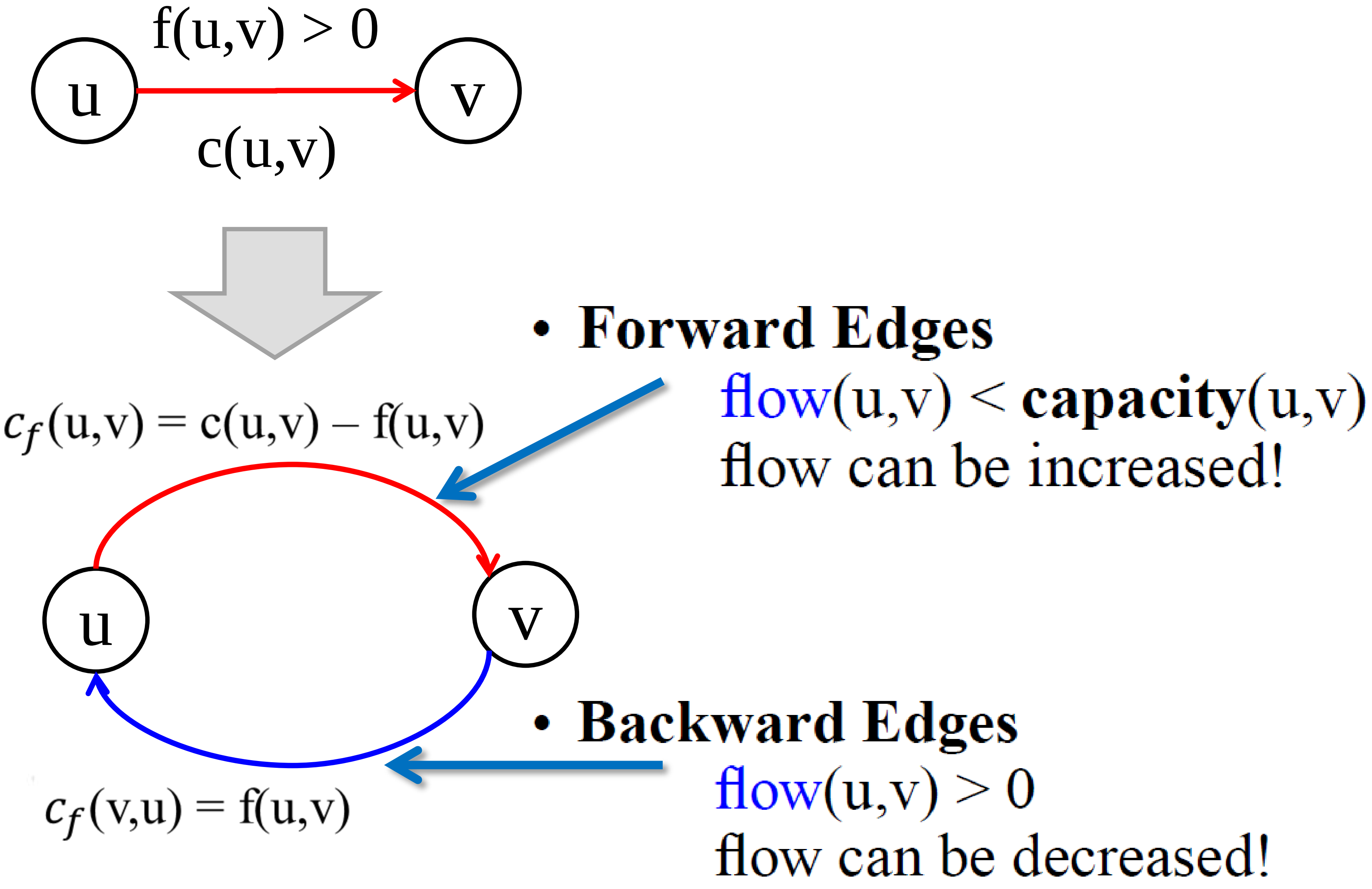
Edges in E_f admit more flow.

Residual Network

Lemma. $|E_f| \leq 2|E|$.



Residual Network



f(u,v) > 0

c(u,v)

c_f(u,v) = c(u,v) – f(u,v)

c_f(v,u) = f(u,v)

- **Forward Edges**

  flow(u,v) < **capacity**(u,v)

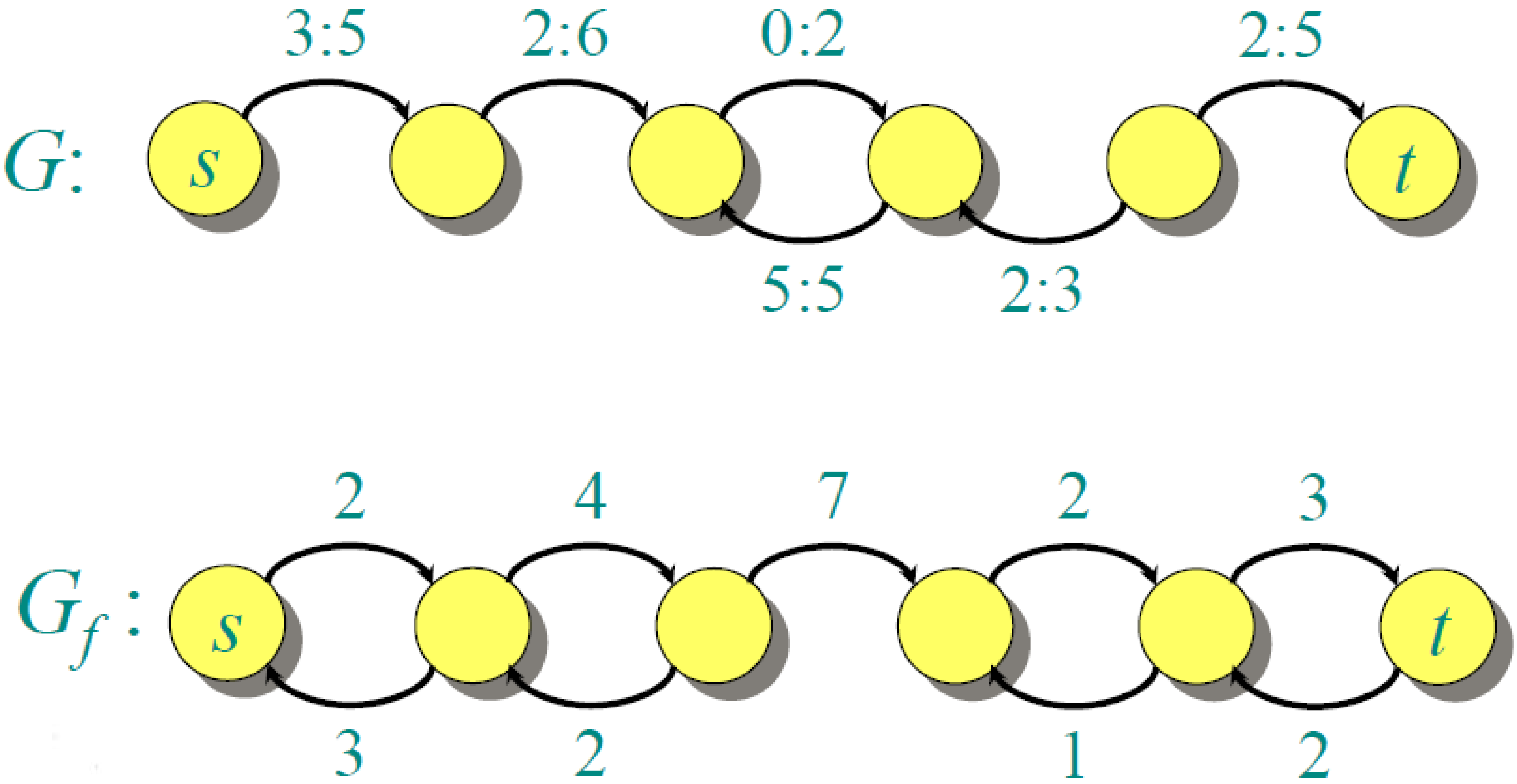
  flow can be increased!

- **Backward Edges**

  flow(u,v) > 0

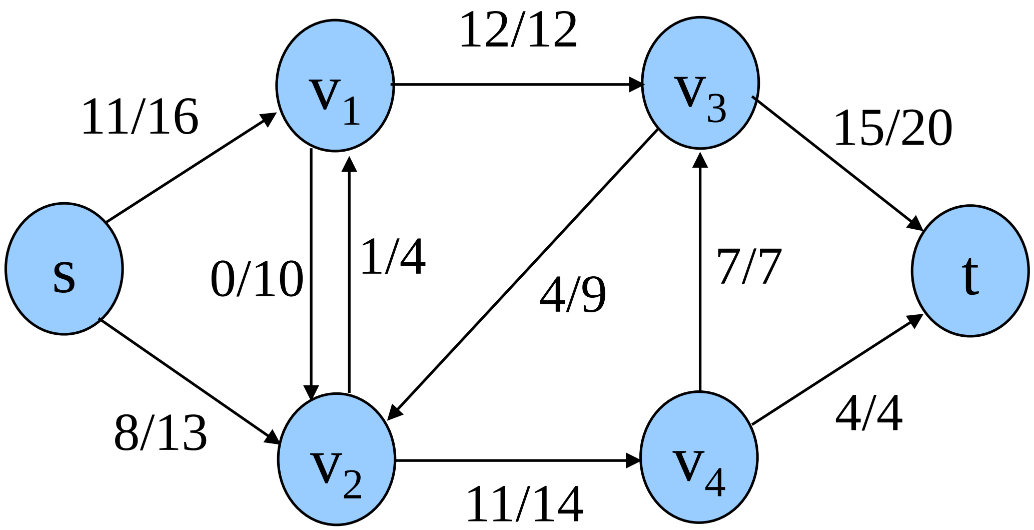
  flow can be decreased!

Residual Network Example



Short Test in Class

- **Give the residual network of the next graph**



Exercises

- **26.1-1**
- **26.1-3**

Augmenting Path

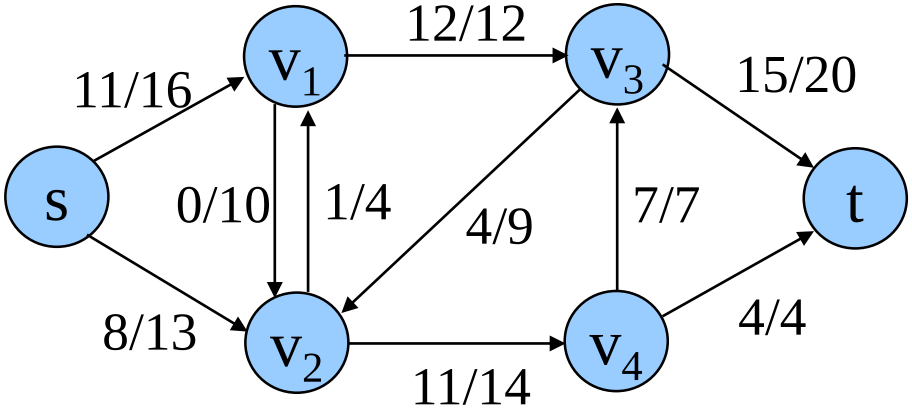
An ***augmenting path*** *p* is a simple path from s to t in the residual network G_f of a flow network G.

residual capacity of *p*

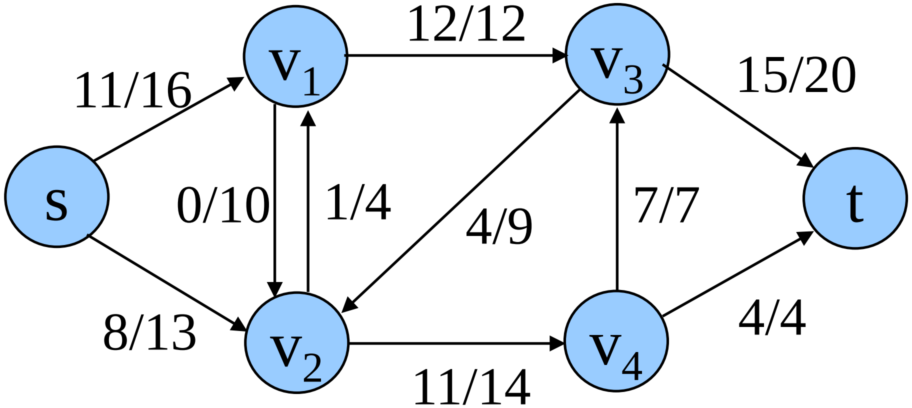
$$c_f(p) = \min_{(u,v)\in p} \{c_f(u,v)\} .$$

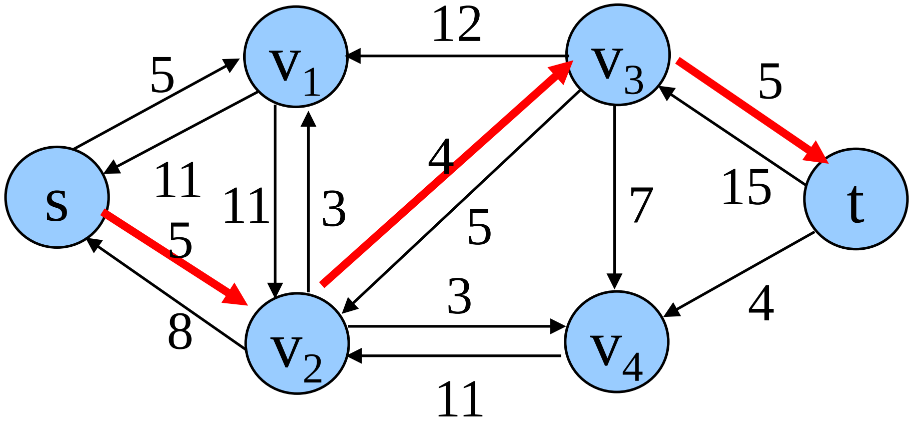
the maximum flow | *f* | can increased by increasing the flow on each edge in *p*

Augmenting Path Example

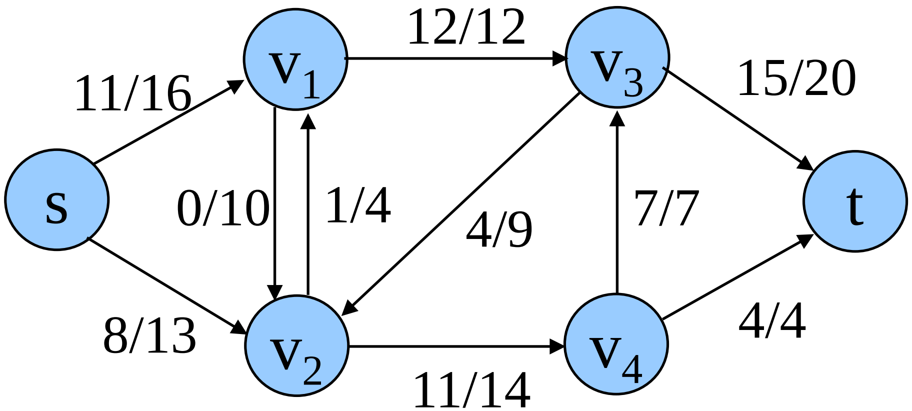


Augmenting Path Example

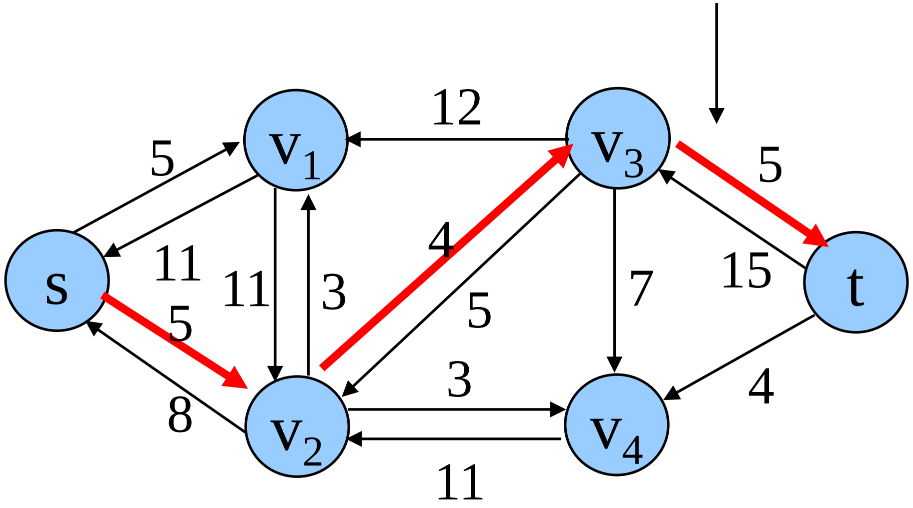




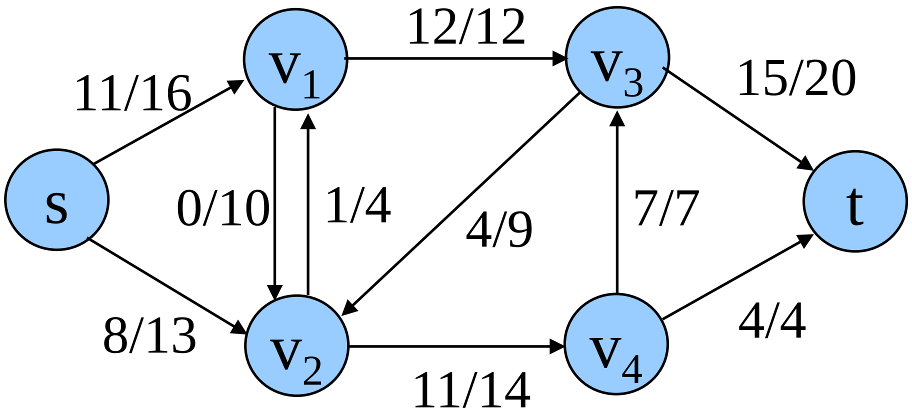
Augmenting Path Example



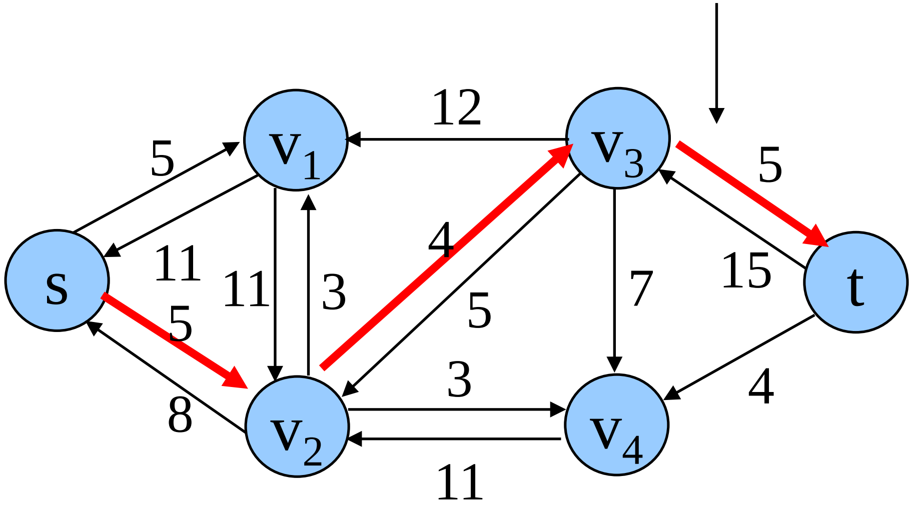
augmenting path *p*



Augmenting Path Example

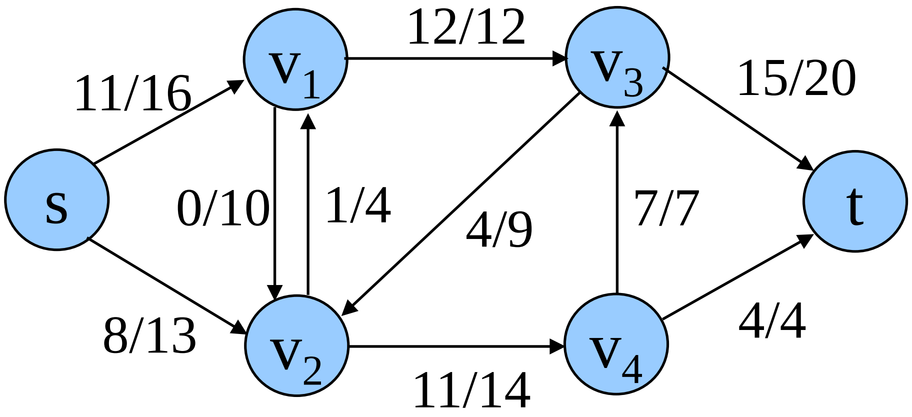


augmenting path *p*

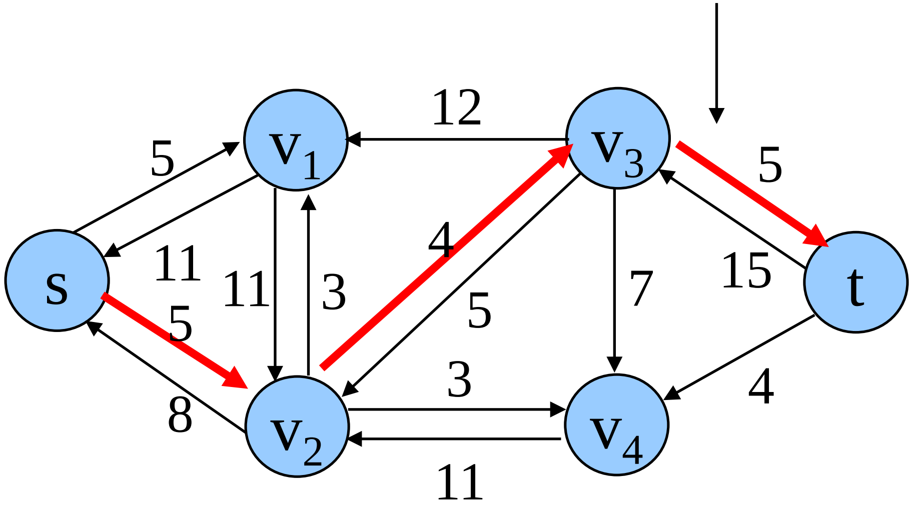


$$c_f(p) = 4$$

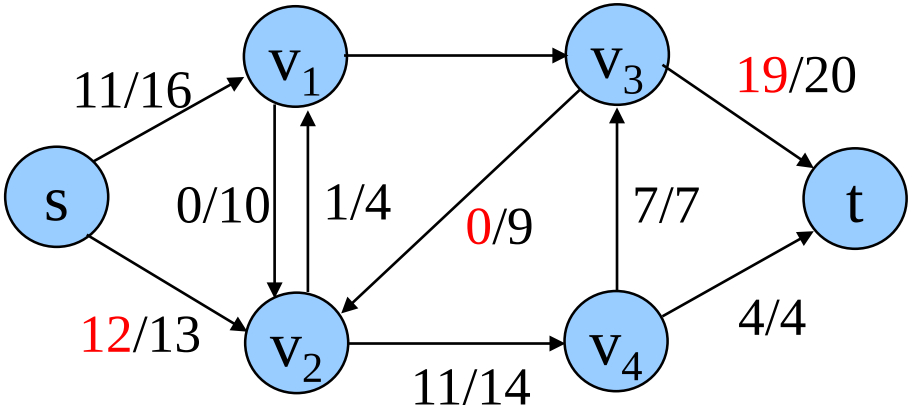
Augmenting Path Example



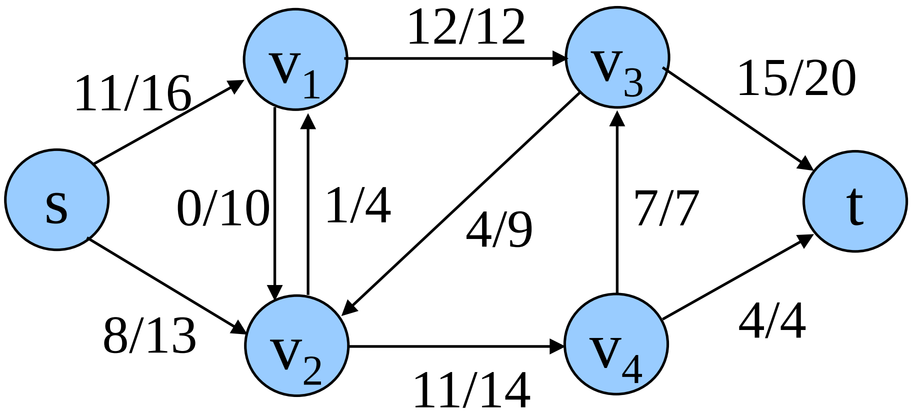
augmenting path *p*

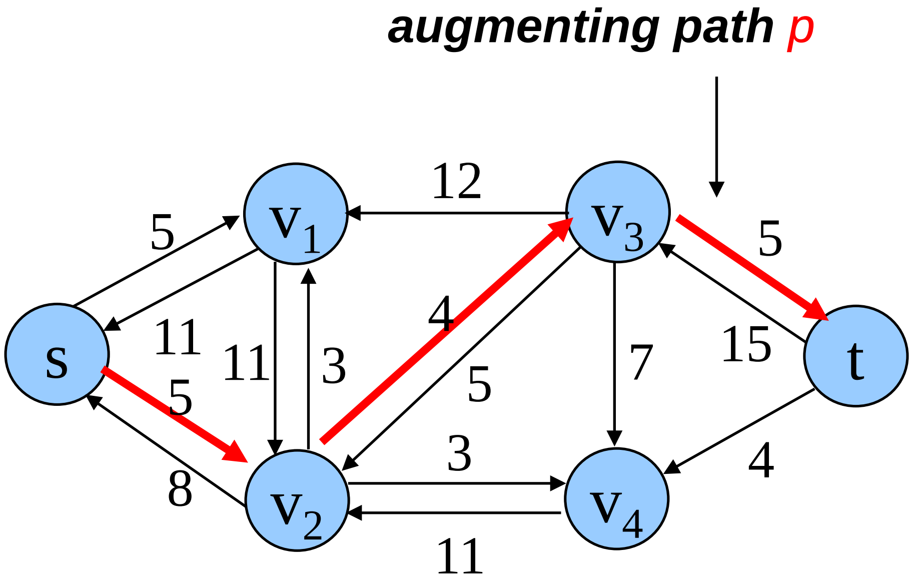


$$c_f(p) = 4$$

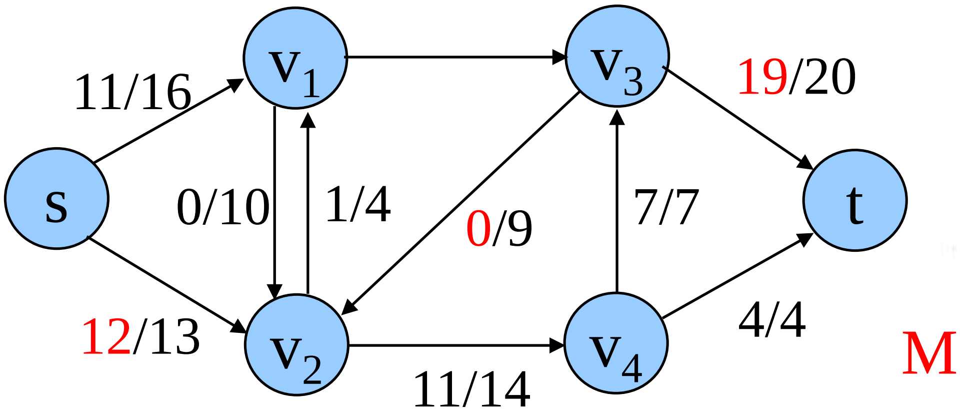


Augmenting Path Example



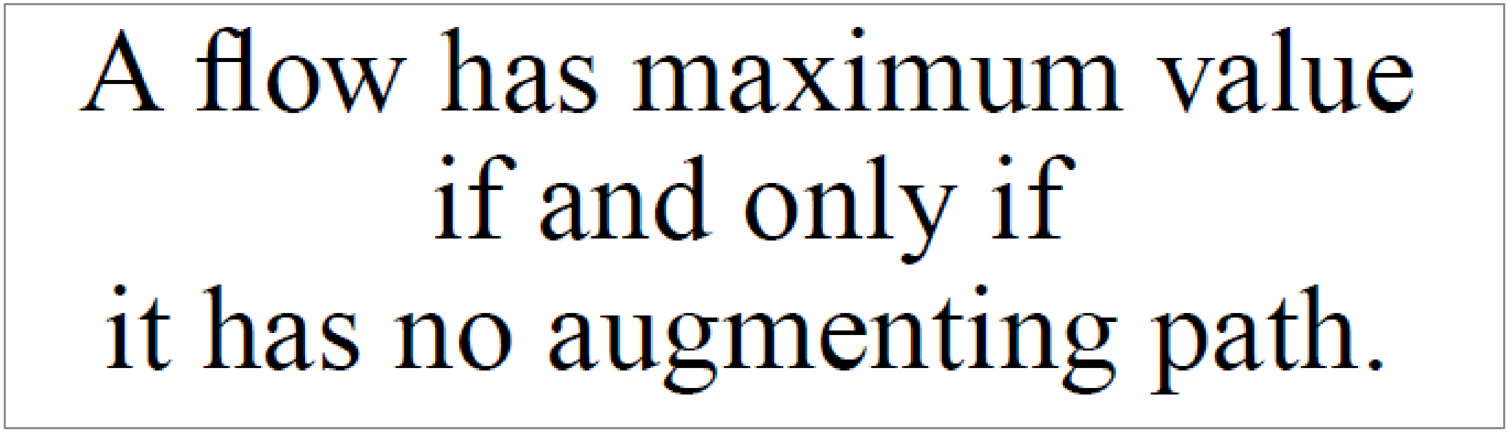


$$c_f(p) = 4$$



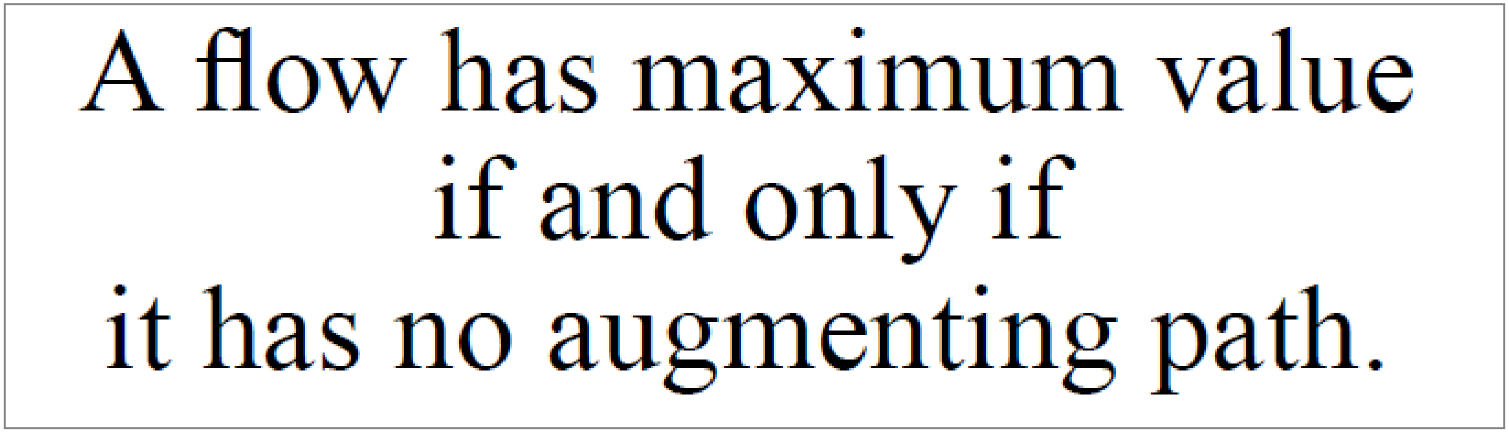
Maximum flow increased by 4!

Maximum Flow Theorem

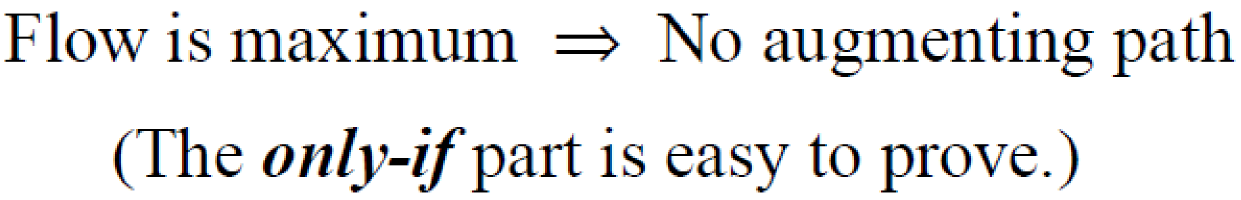


A flow has maximum value
if and only if
it has no augmenting path.

Maximum Flow Theorem



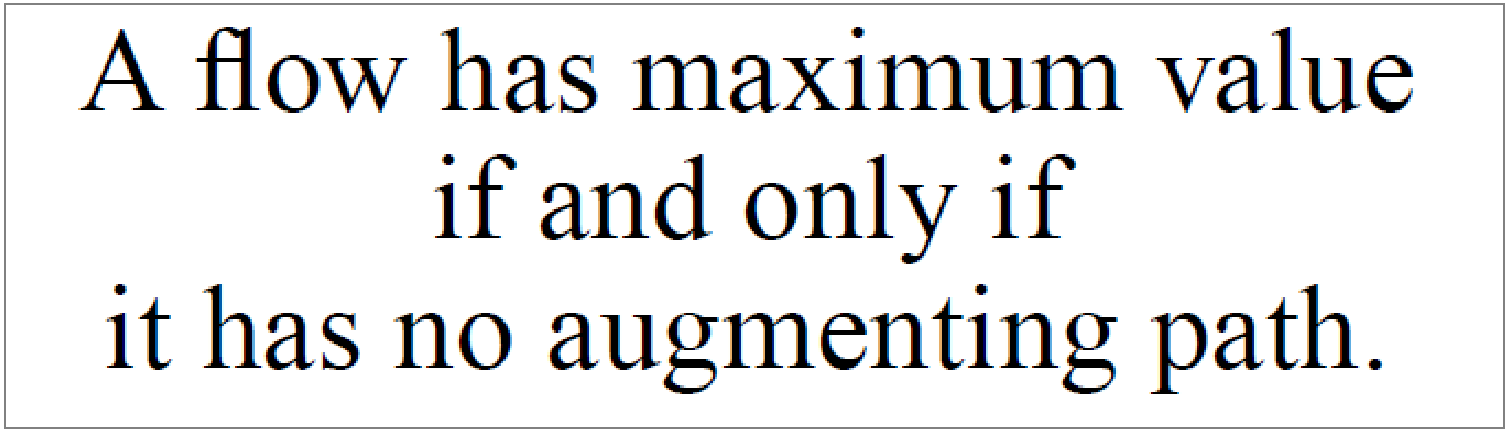
A flow has maximum value
if and only if
it has no augmenting path.



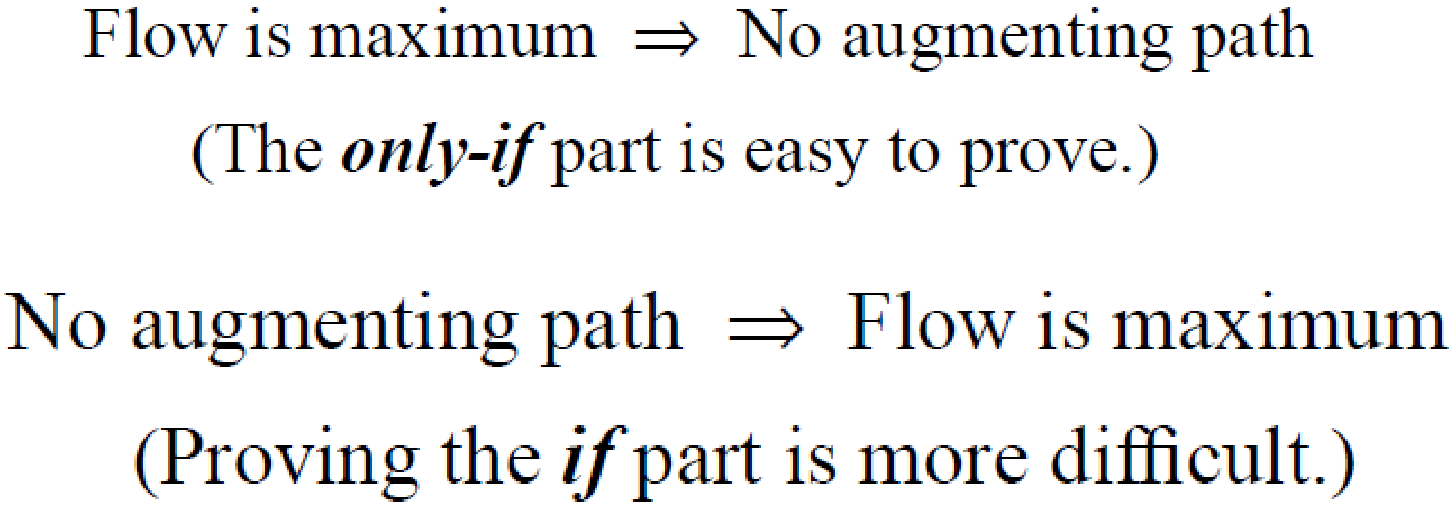
Flow is maximum $\Rightarrow$ No augmenting path

(The ***only-if*** part is easy to prove.)

Maximum Flow Theorem



A flow has maximum value
if and only if
it has no augmenting path.



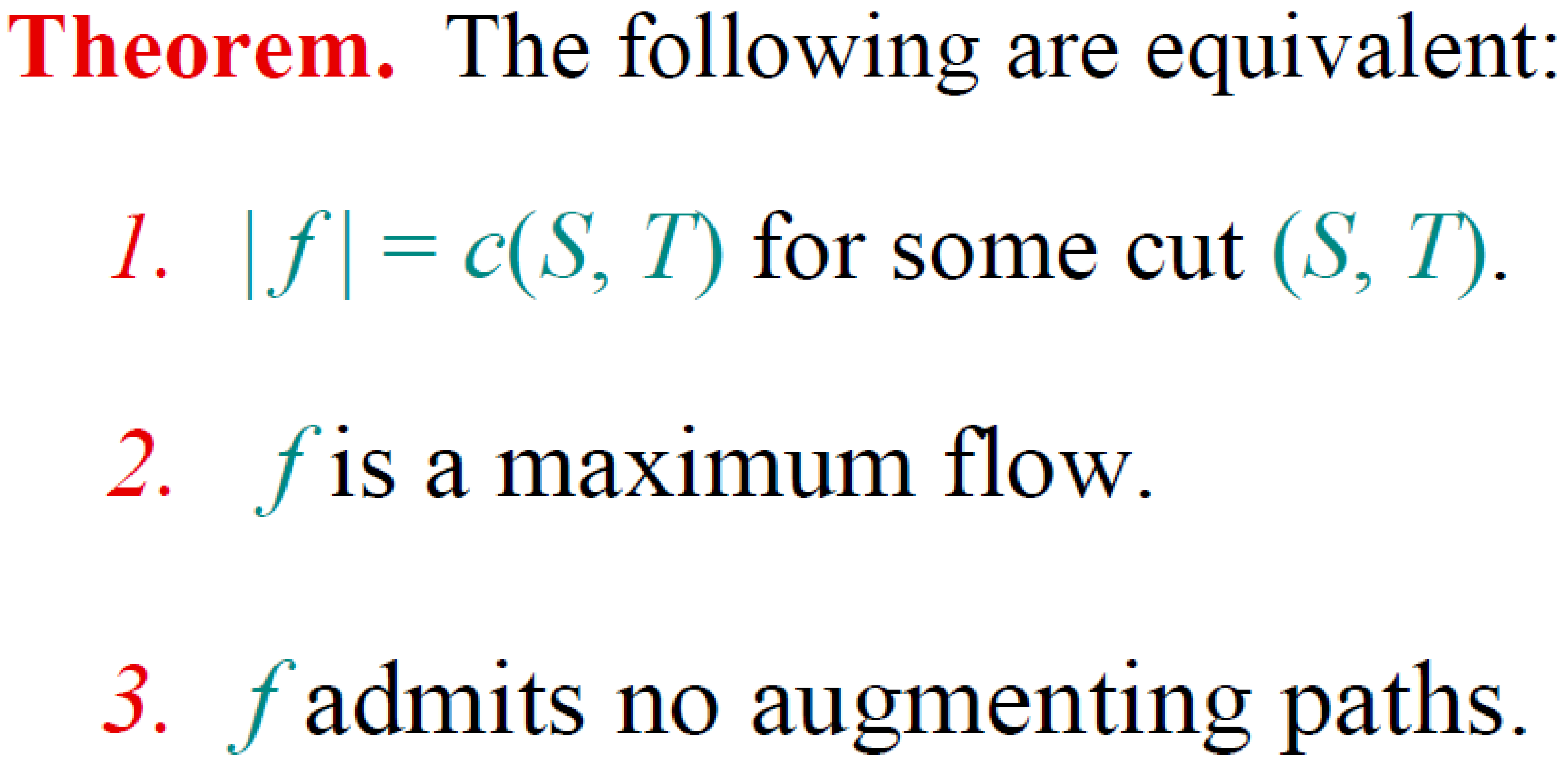
Flow is maximum ⇒ No augmenting path

(The ***only-if*** part is easy to prove.)

No augmenting path ⇒ Flow is maximum

(Proving the ***if*** part is more difficult.)

Max-Flow, Min-Cut Theorem



Theorem. The following are equivalent:

1. $|f| = c(S, T)$ for some cut (S, T).

2. f is a maximum flow.

3. f admits no augmenting paths.

Max-Flow, Min-Cut Theorem

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

Proof.

$(1) \Rightarrow (2)$: Since $|f| \leq c(S, T)$ for any cut (S, T)

$|f| = c(S, T)$ implies that f is a maximum flow.

Max-Flow, Min-Cut Theorem

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

Proof.

$(2) \Rightarrow (3)$: If there were an augmenting path,

⇓

$|f|$ flow value could be increased,

Max-Flow, Min-Cut Theorem

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

Proof.

$(3) \Rightarrow (1)$: f admits no augmenting paths.

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

$T = V - S$

Max-Flow, Min-Cut Theorem

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

Proof.

$(3) \Rightarrow (1)$: f admits no augmenting paths.

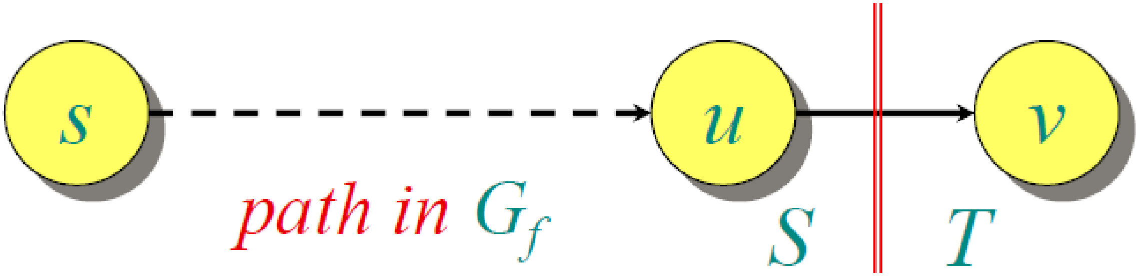
$S = \{v \in V :$ there exists a path in G_f from s to $v\}$

$T = V - S$

(S,T) is a cut! Why?

Max-Flow, Min-Cut Theorem

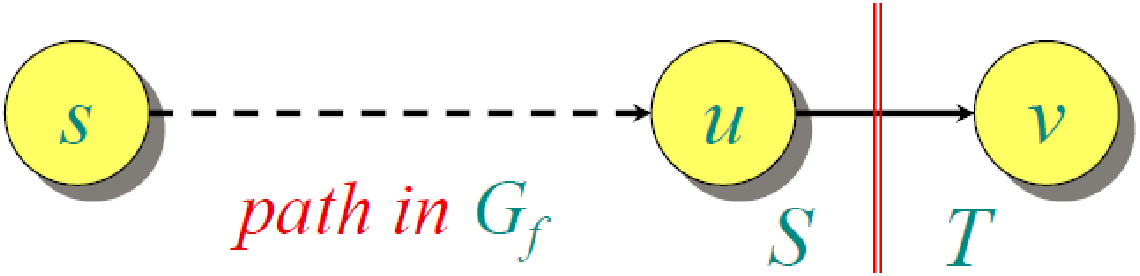
Proof. (3) ⇒ (1): f admits no augmenting paths.



$u \in S$ and $v \in T$.

Max-Flow, Min-Cut Theorem

Proof. (3) ⇒ (1): f admits no augmenting paths.

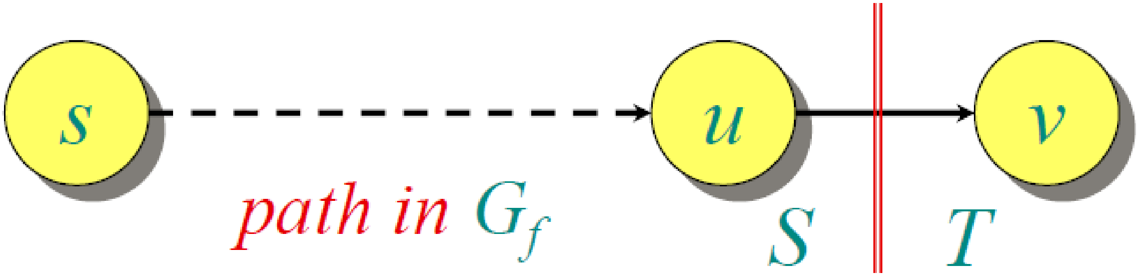


$u \in S$ and $v \in T$.

$$v \in T \Rightarrow (u, v) \notin E_f \Rightarrow c_f(u, v) = 0$$

Max-Flow, Min-Cut Theorem

Proof. (3) ⇒ (1): f admits no augmenting paths.



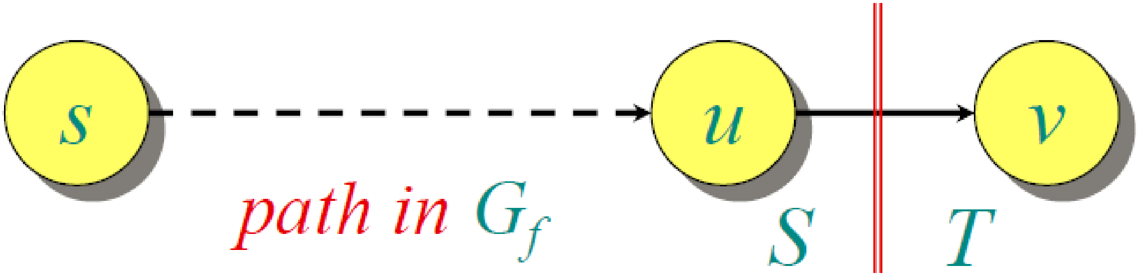
$u \in S$ and $v \in T$.

$$v \in T \Rightarrow (u, v) \notin E_f \Rightarrow c_f(u, v) = 0$$

$$\Rightarrow f(u,v) = c(u,v) \quad \because c_f(u,v) = c(u,v) - f(u,v)$$

Max-Flow, Min-Cut Theorem

Proof. (3) ⇒ (1): f admits no augmenting paths.



$u \in S$ and $v \in T$.

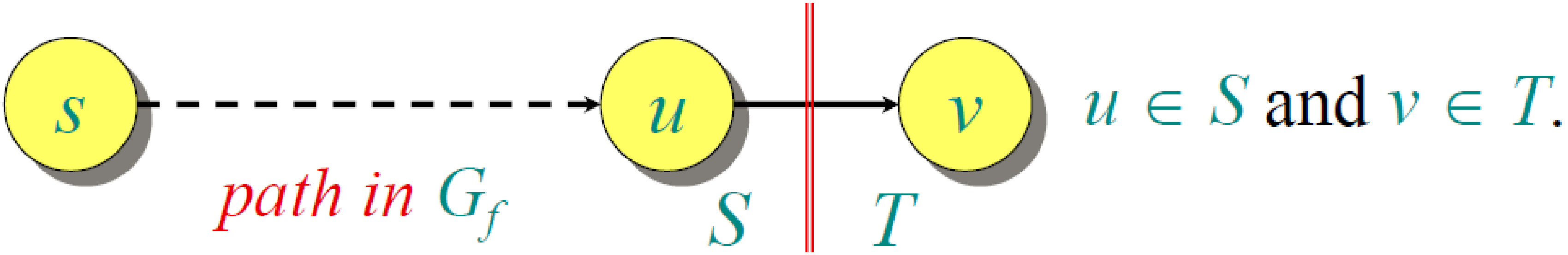
$$v \in T \Rightarrow (u, v) \notin E_f \Rightarrow c_f(u, v) = 0$$

$$\Rightarrow f(u,v) = c(u,v) \quad \because c_f(u,v) = c(u,v) - f(u,v)$$

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

Max-Flow, Min-Cut Theorem

Proof. (3) ⇒ (1): f admits no augmenting paths.



$$v \in T \Rightarrow (u, v) \notin E_f \Rightarrow c_f(u, v) = 0$$

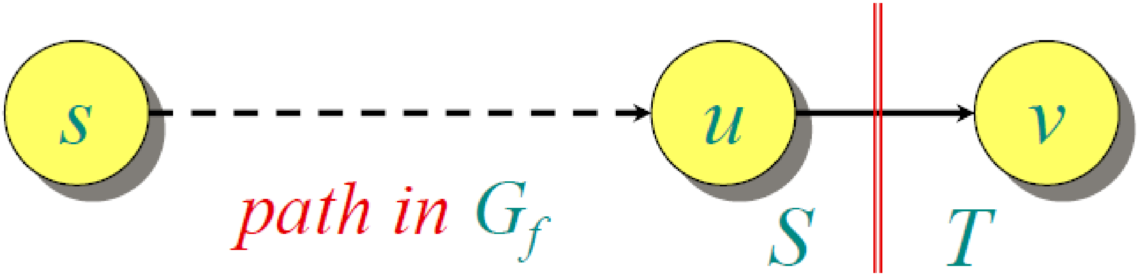
$$\Rightarrow f(u,v) = c(u,v) \quad \because c_f(u,v) = c(u,v) - f(u,v)$$

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

$$\Rightarrow f(S,T) = c(S,T) = |f|$$

Max-Flow, Min-Cut Theorem

Proof. (3) ⇒ (1): f admits no augmenting paths.



$u \in S$ and $v \in T$.

$$v \in T \Rightarrow (u, v) \notin E_f \Rightarrow c_f(u, v) = 0$$

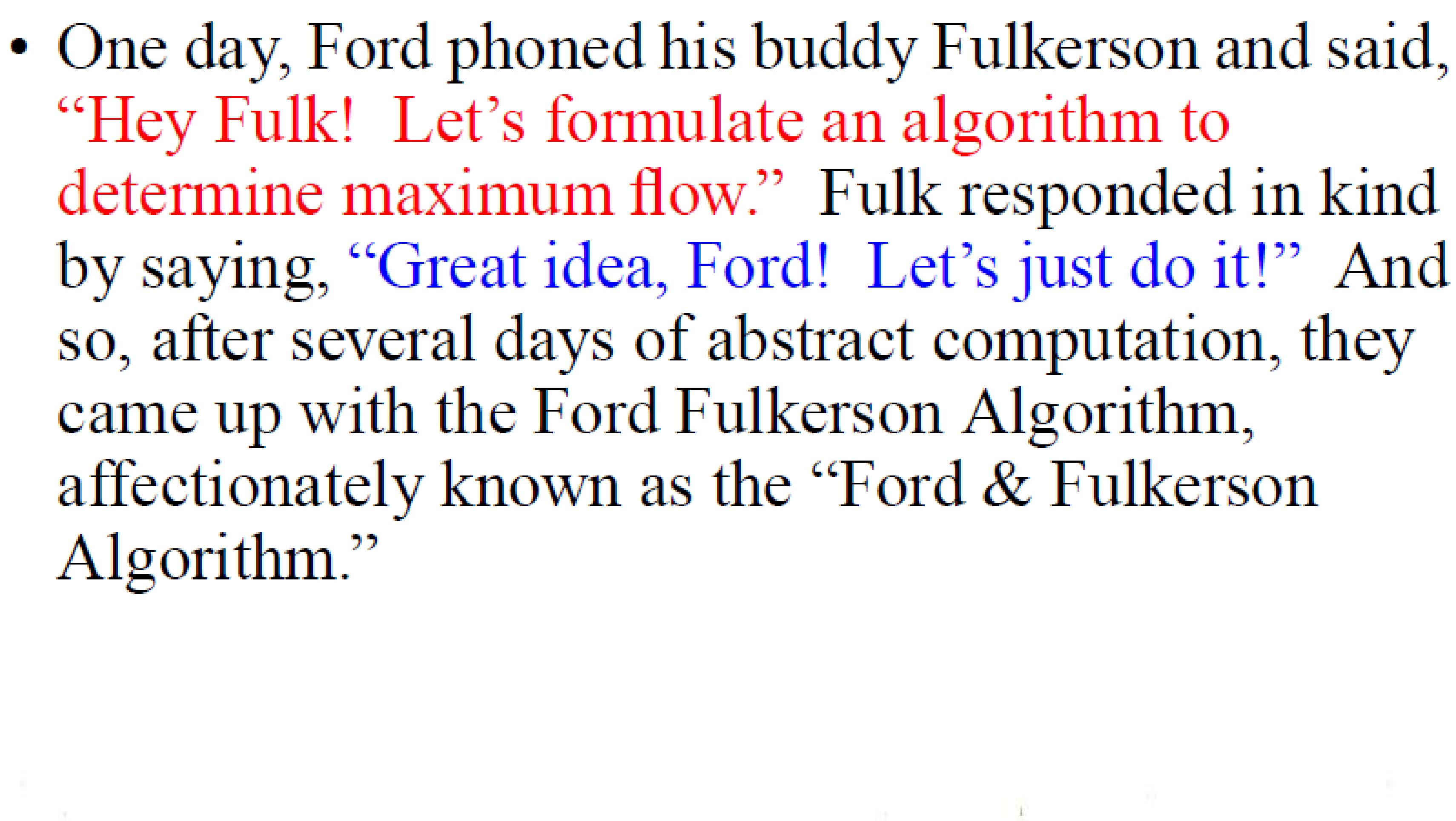
$$\Rightarrow f(u,v) = c(u,v) \quad \because c_f(u,v) = c(u,v) - f(u,v)$$

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

$$\Rightarrow f(S,T) = c(S,T) = |f|$$ Maximum flow!

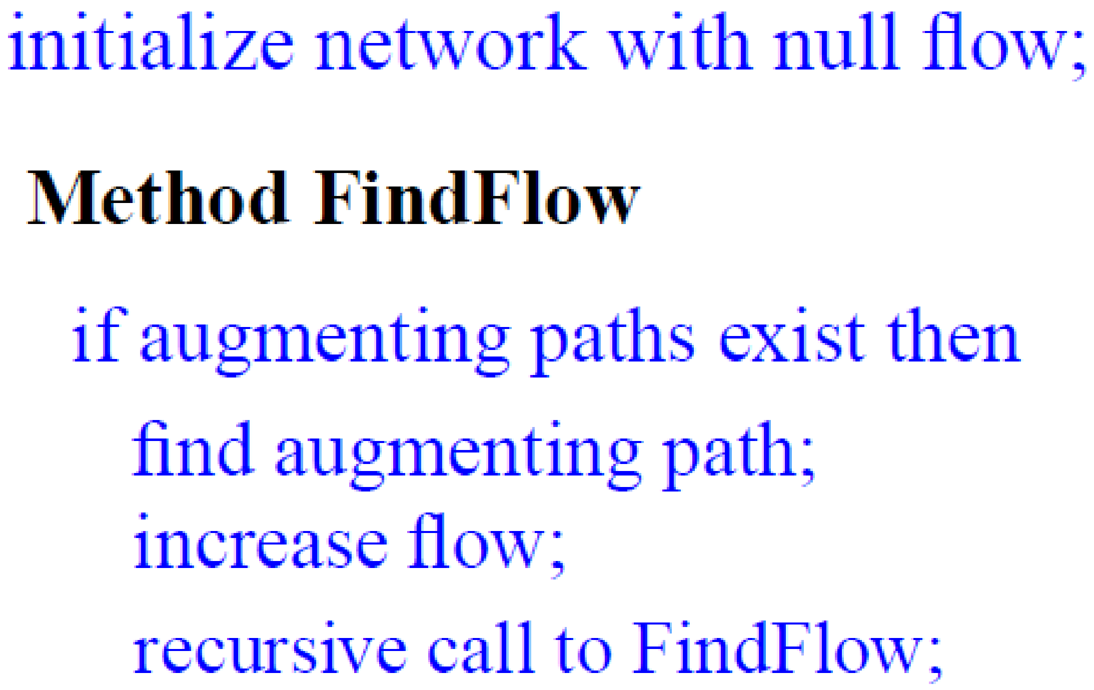
Ford-Fulkerson Algorithm

A Story



- One day, Ford phoned his buddy Fulkerson and said, “Hey Fulk! Let’s formulate an algorithm to determine maximum flow.” Fulk responded in kind by saying, “Great idea, Ford! Let’s just do it!” And so, after several days of abstract computation, they came up with the Ford Fulkerson Algorithm, affectionately known as the “Ford & Fulkerson Algorithm.”

Rough Idea

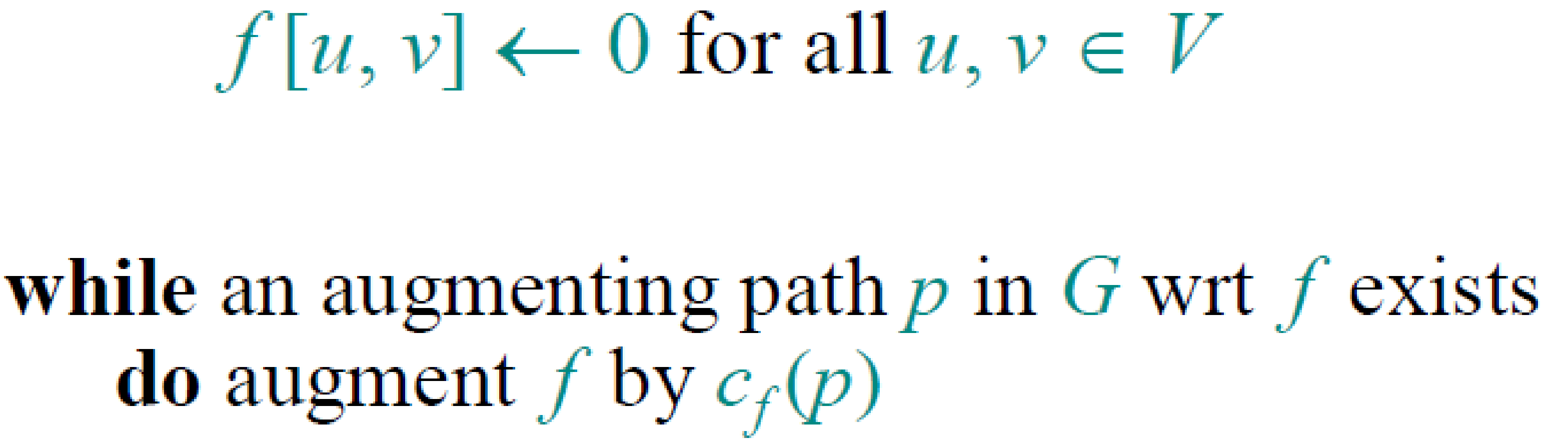


```
initialize network with null flow;

Method FindFlow

  if augmenting paths exist then
    find augmenting path;
    increase flow;
    recursive call to FindFlow;
```

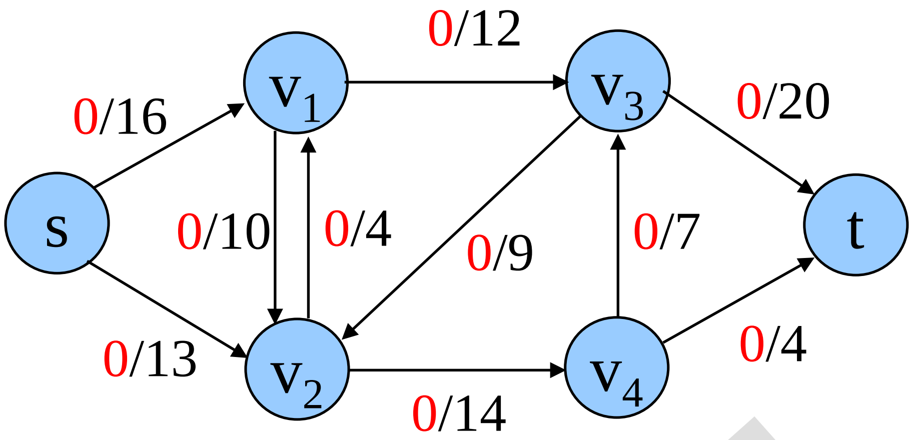
Algorithm



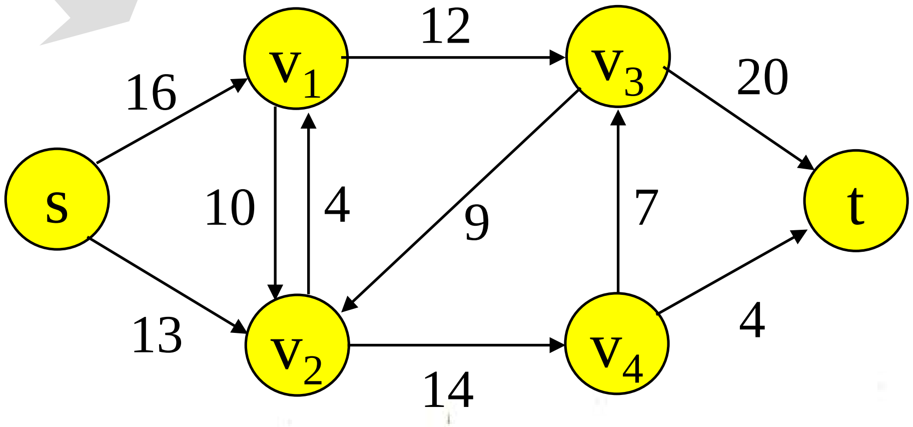
```
$f[u, v] \leftarrow 0$ for all $u, v \in V$

while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

Example—Basic Implementation



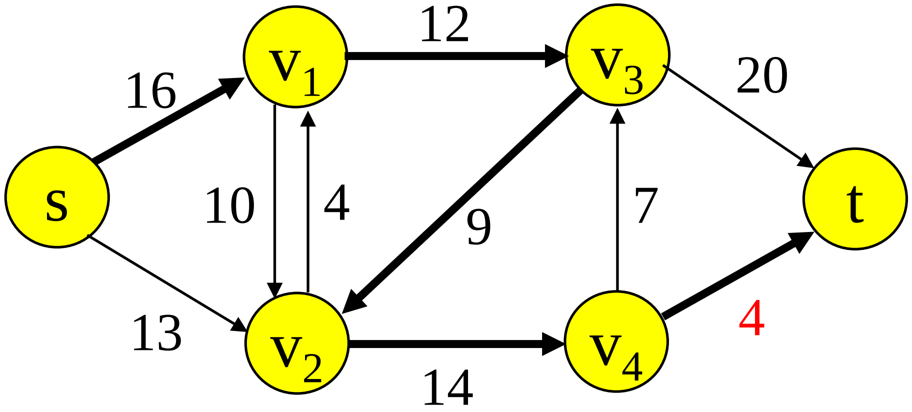
Flow initialization

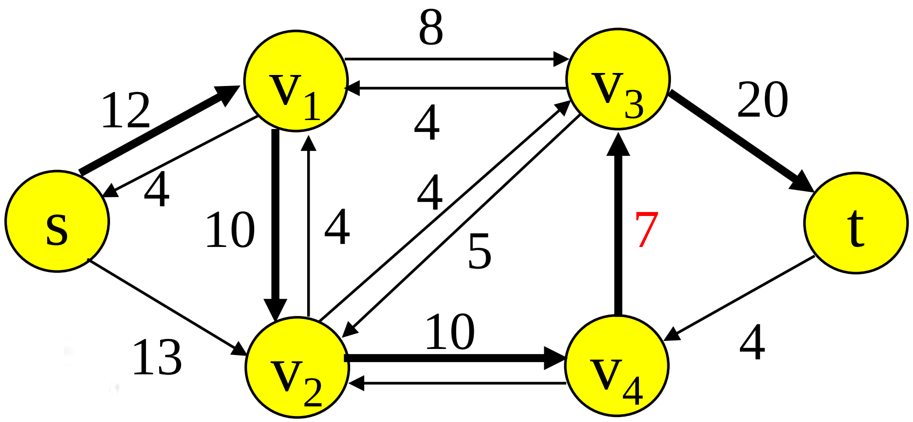


Residual network

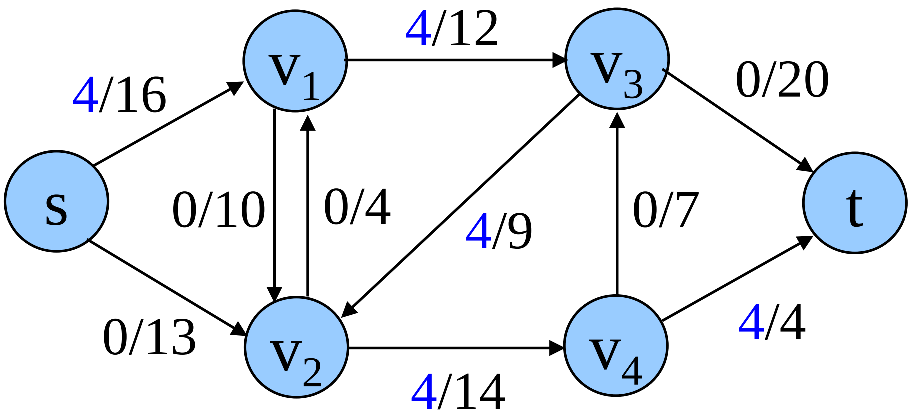
Example

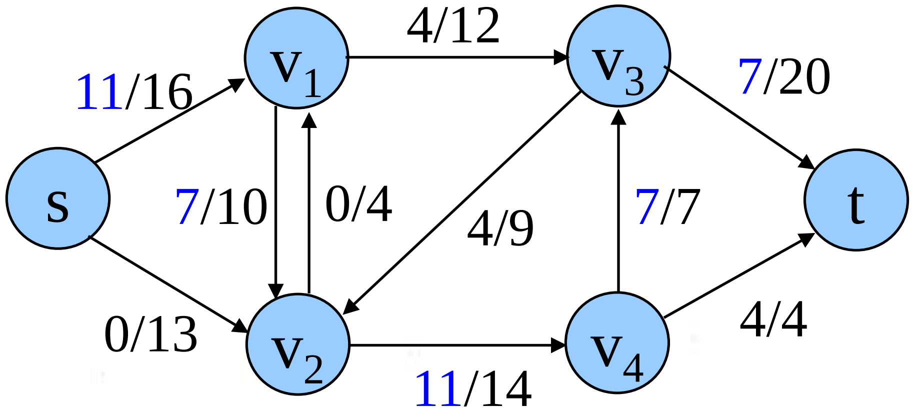
Residual Networks





Flows

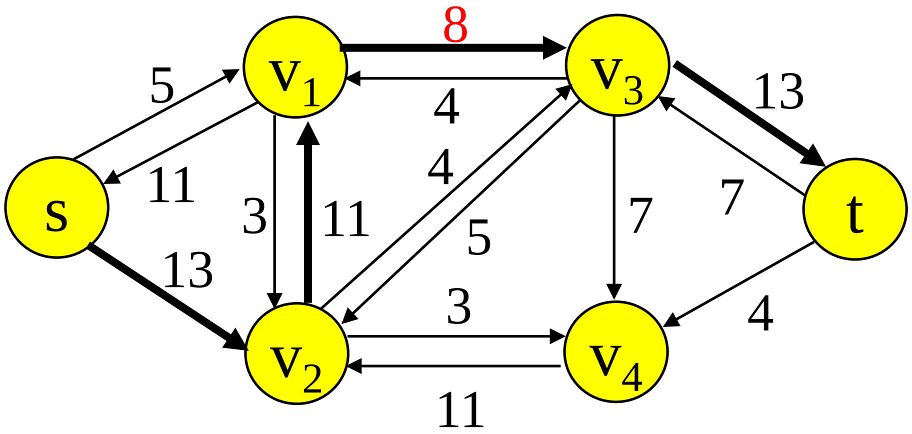


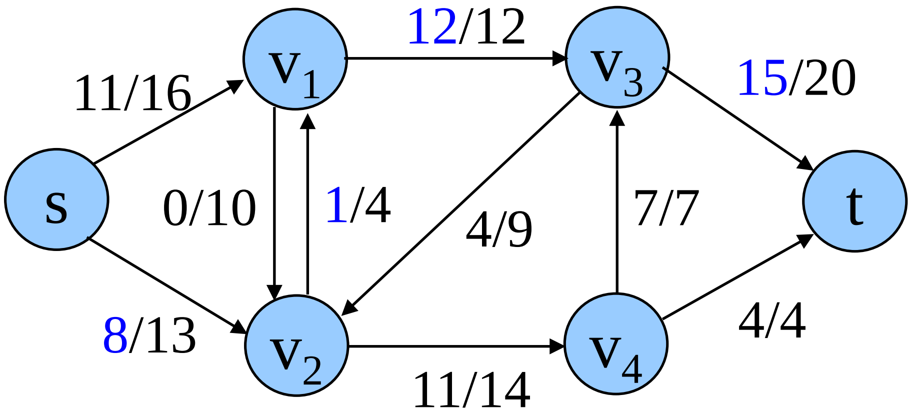


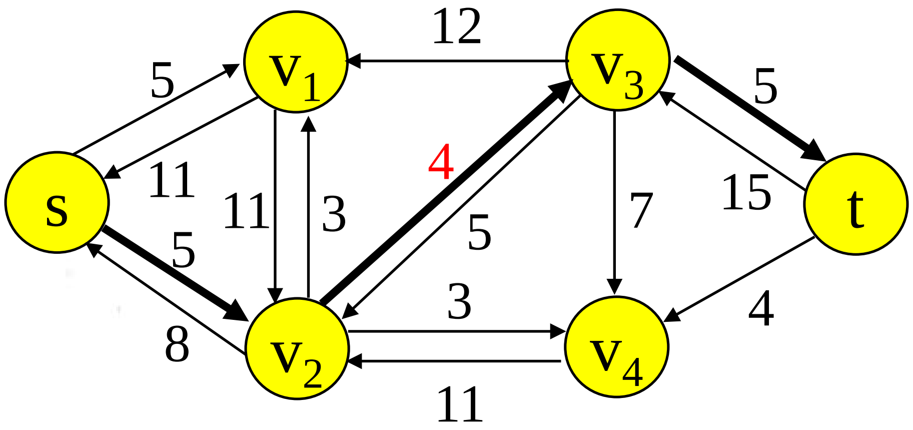
Example

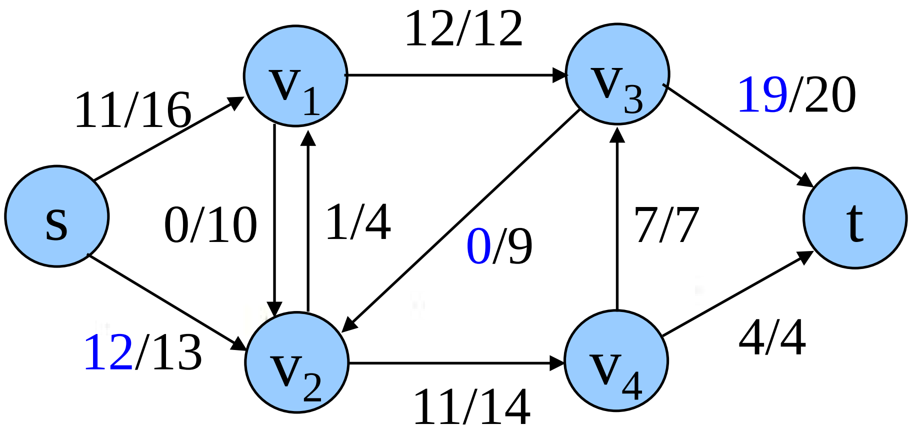
Residual Networks

Flows





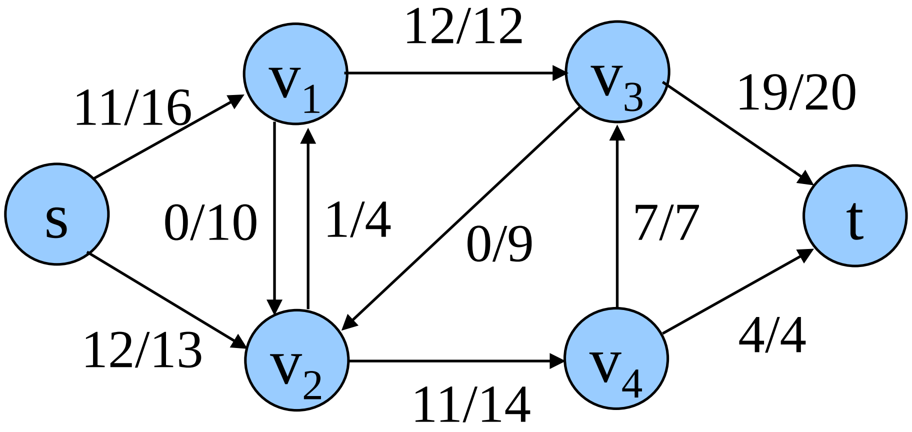


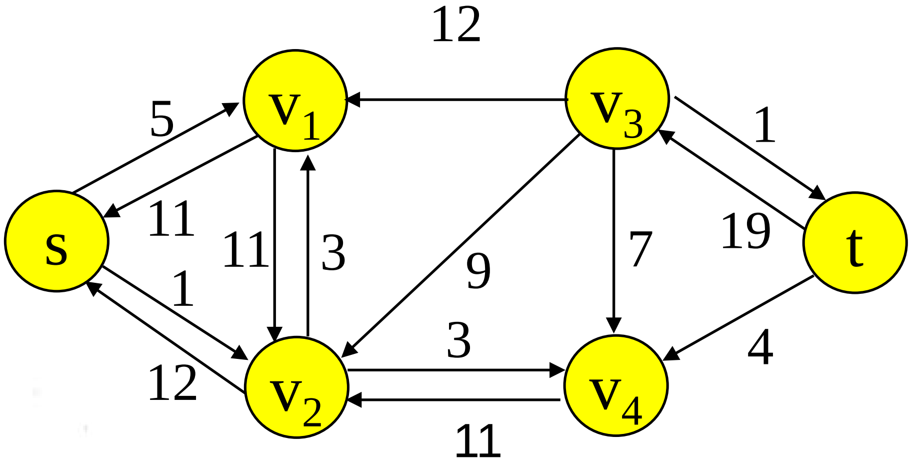


Example

Residual Networks

Flows

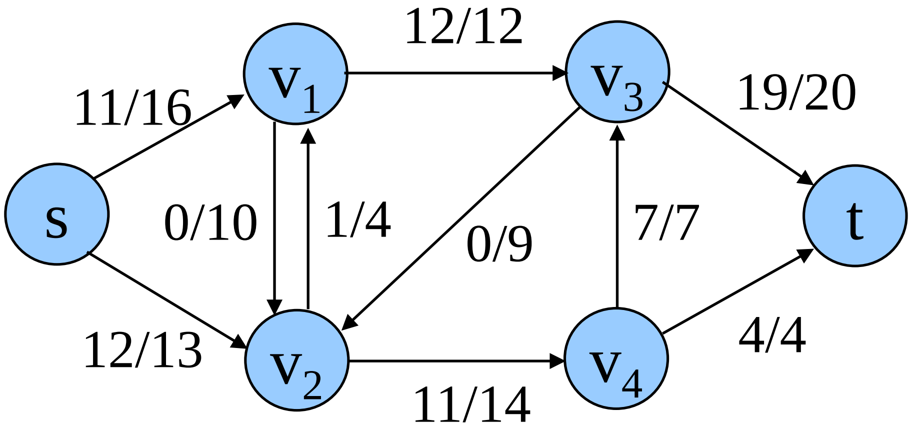


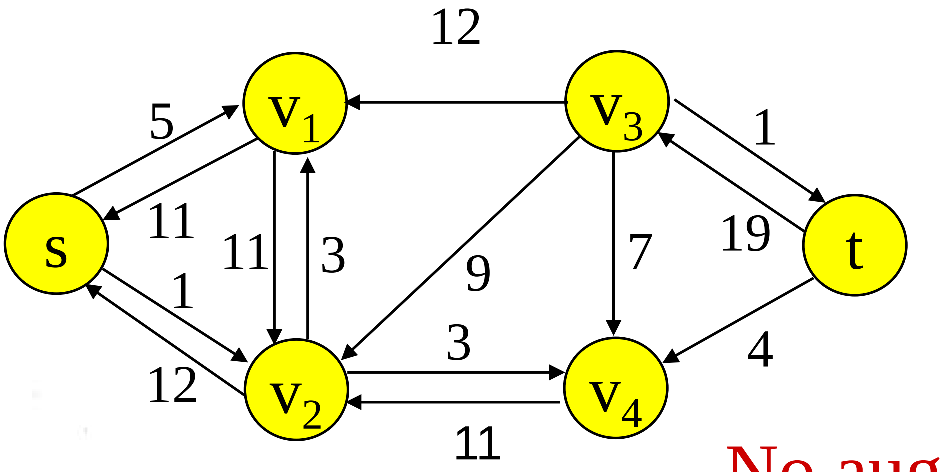


Example

Residual Networks

Flows



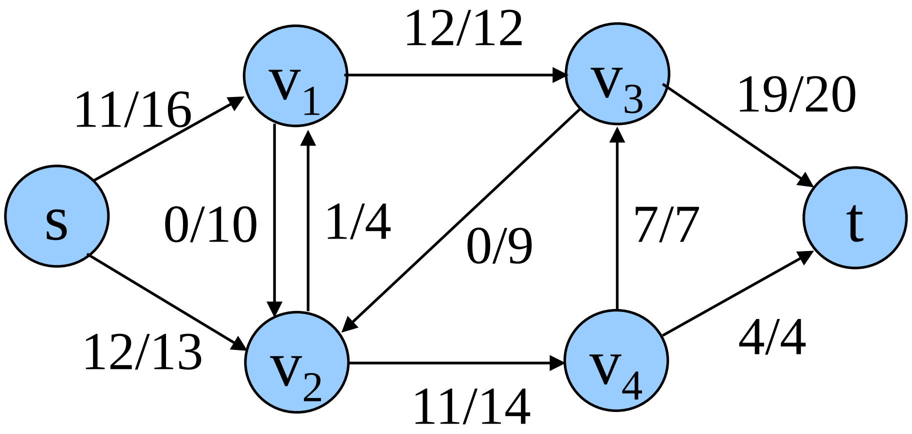


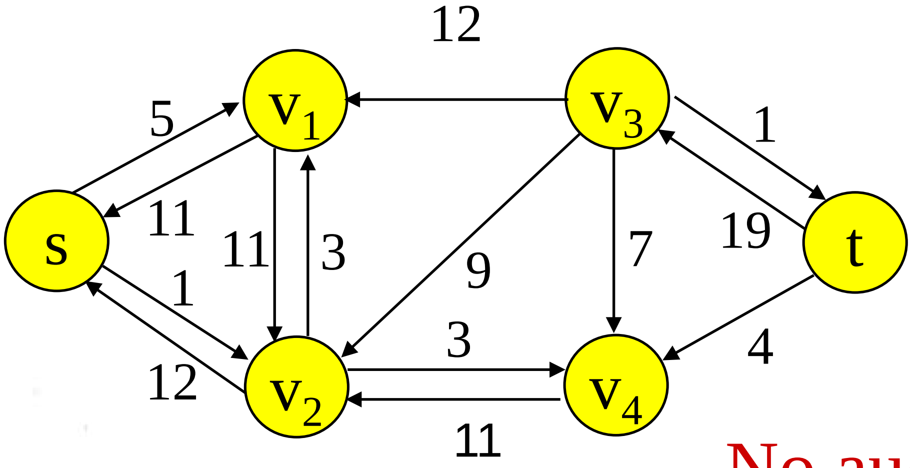
No augmenting path

Example

Residual Networks

Flows



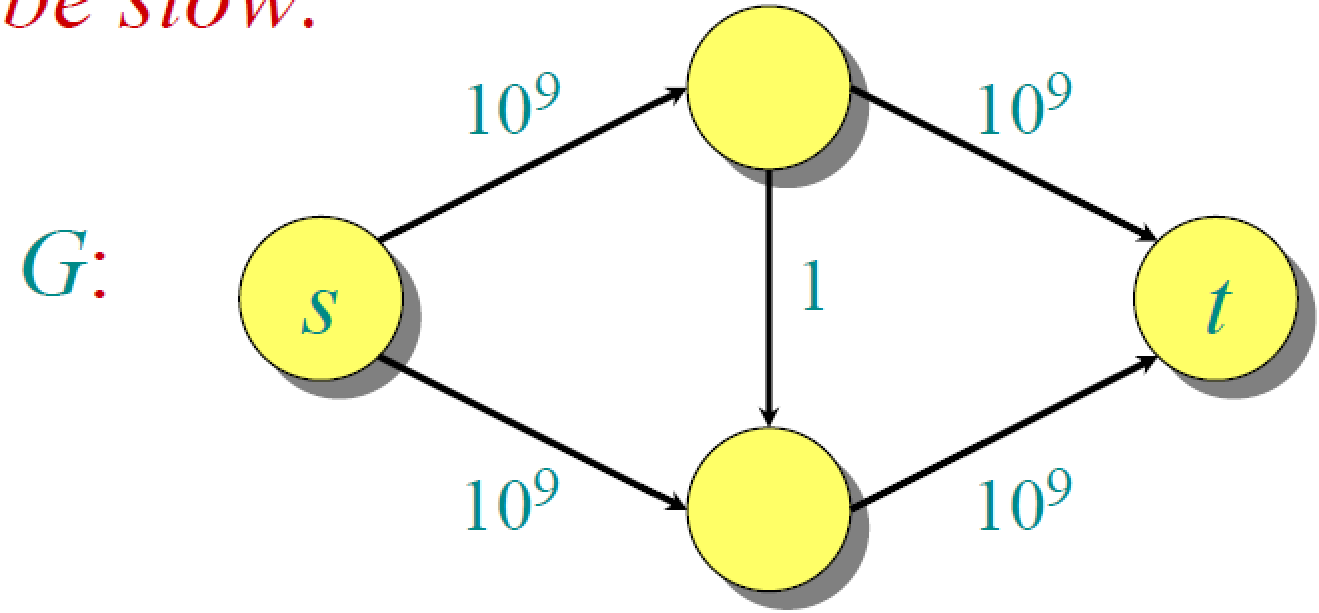


Maximum flow |*f*| = 11+12 = 23

No augmenting path

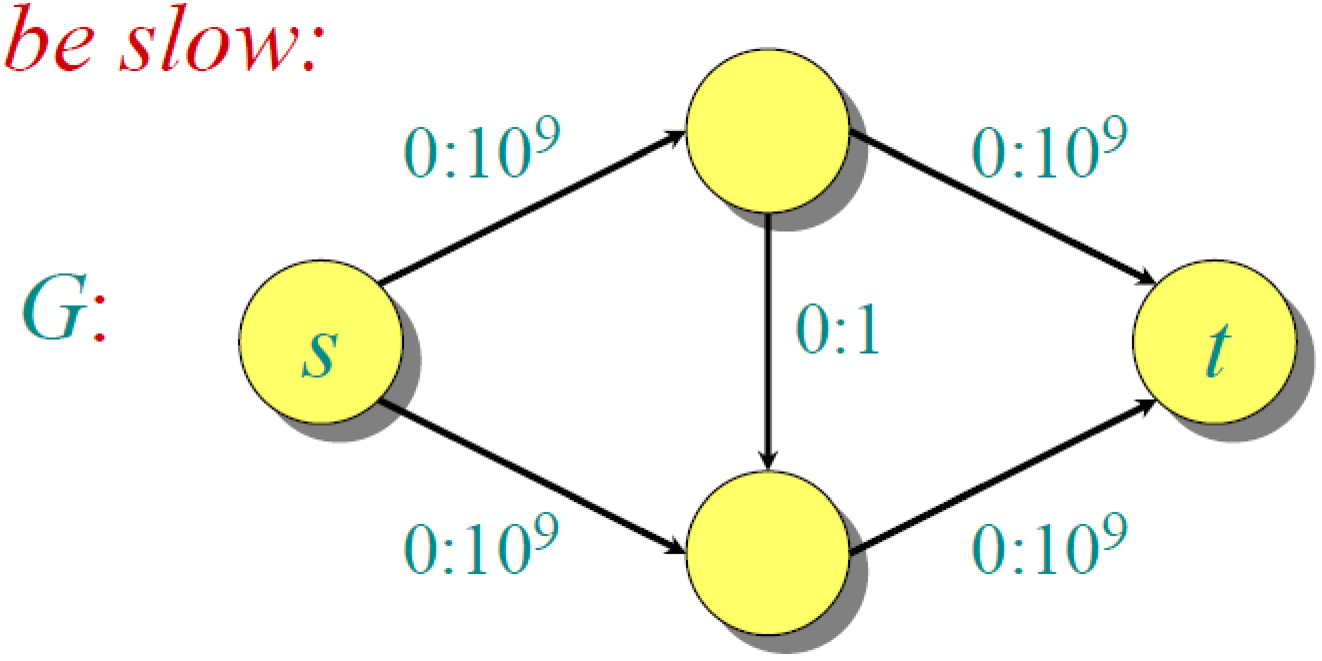
Problem：Time Complexity

Can be slow:



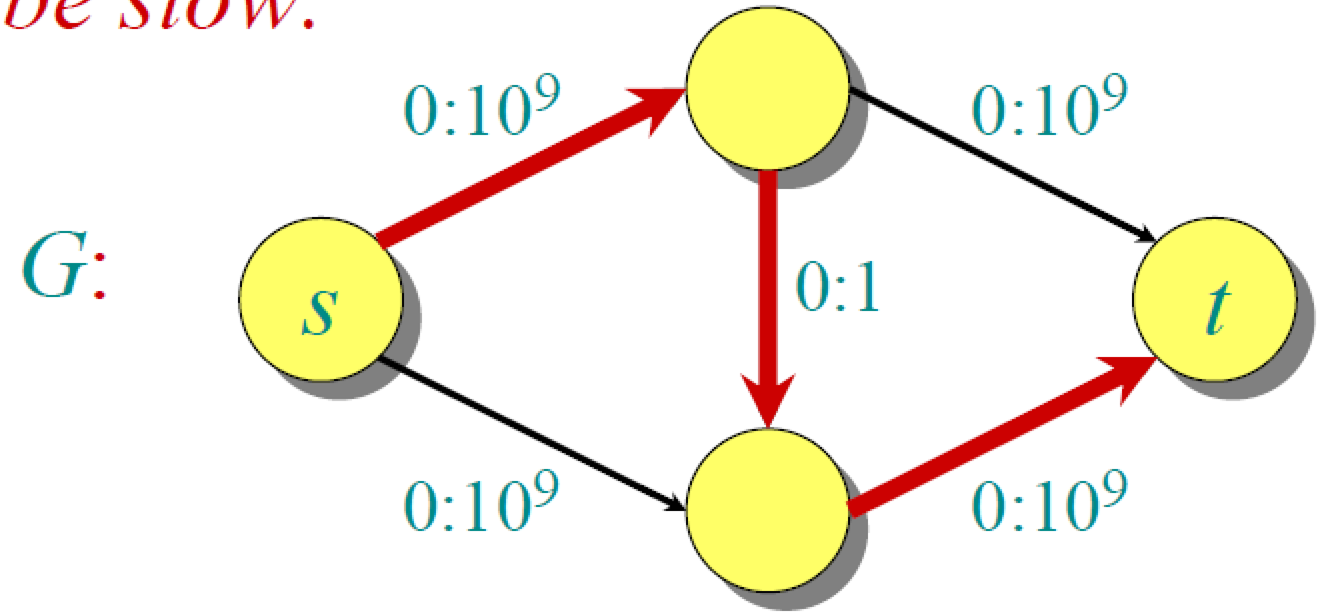
Problem： Time Complexity

Can be slow:



Problem： Time Complexity

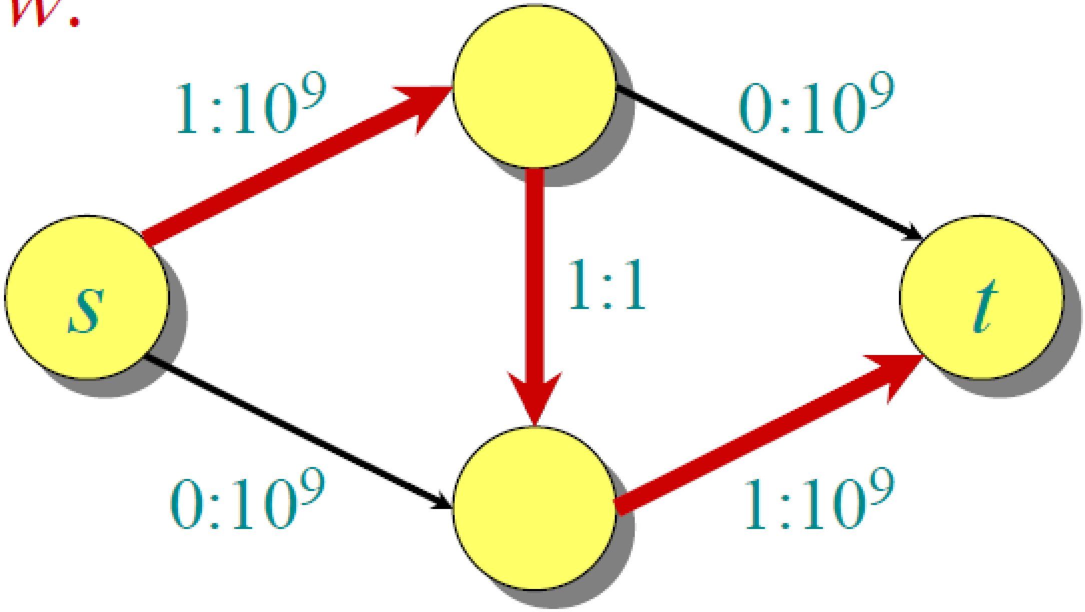
Can be slow:



Problem：Time Complexity

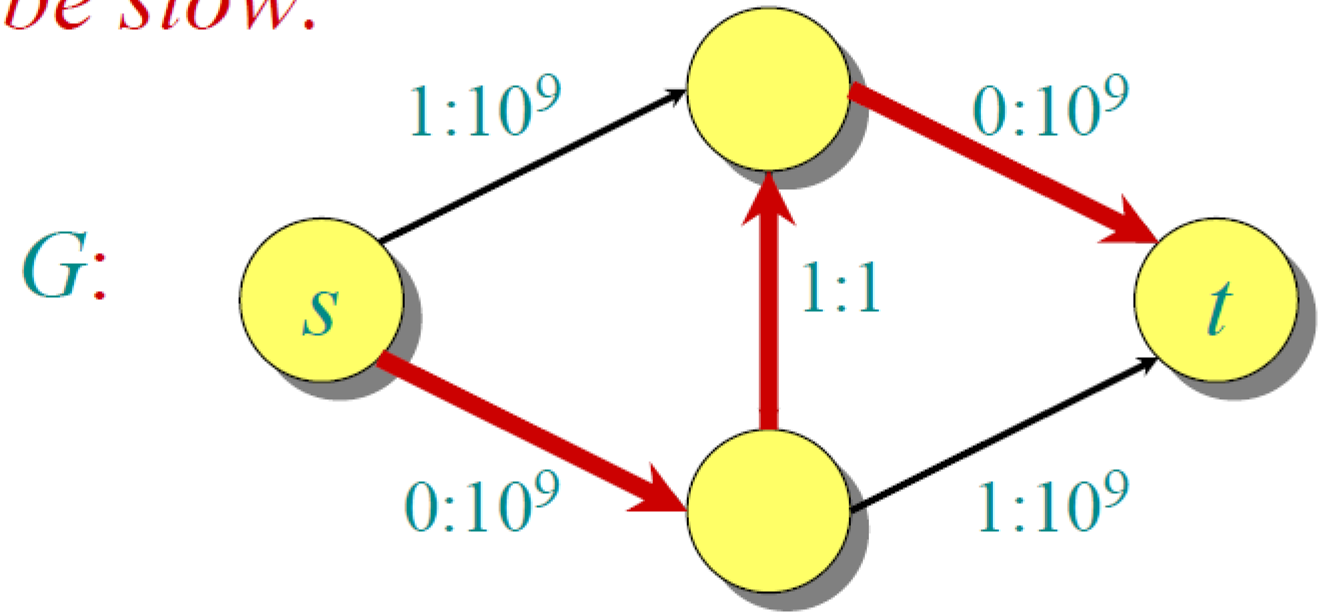
Can be slow:

G:



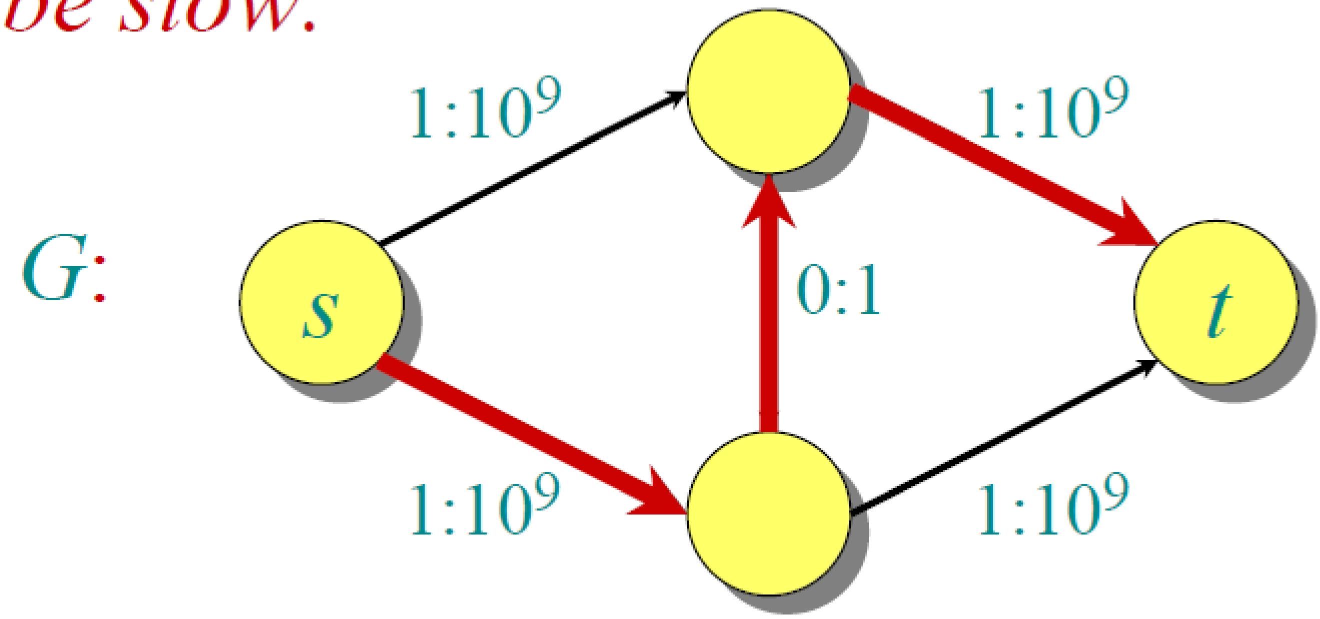
Problem： Time Complexity

Can be slow:



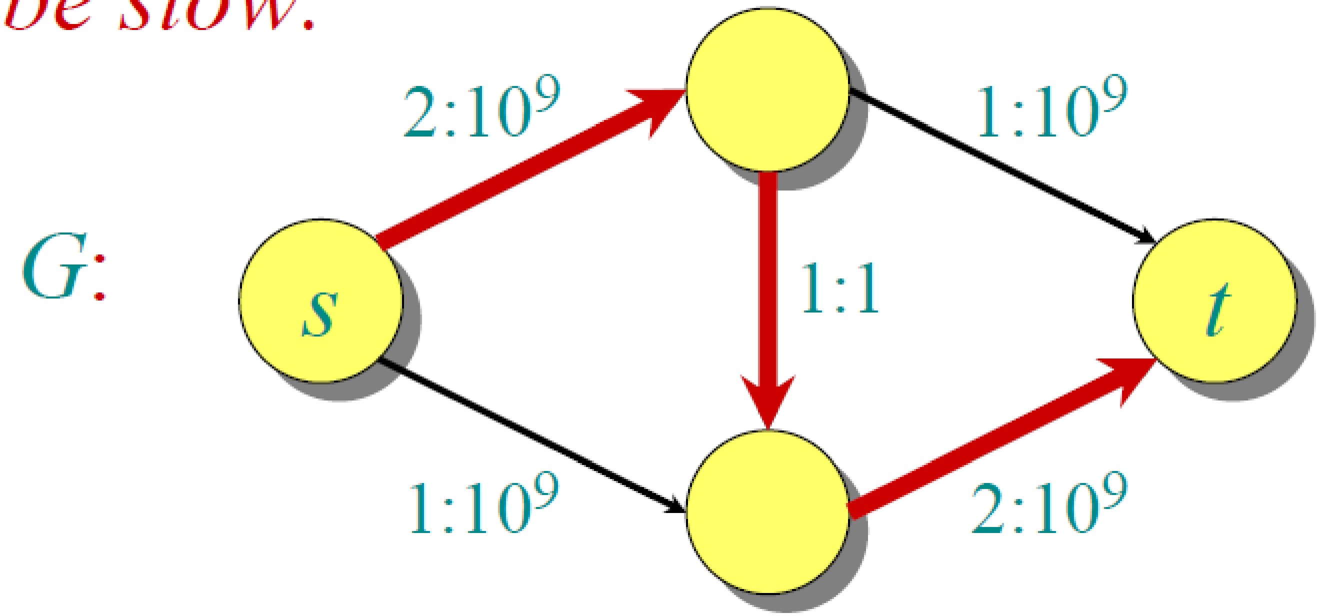
Problem： Time Complexity

Can be slow:



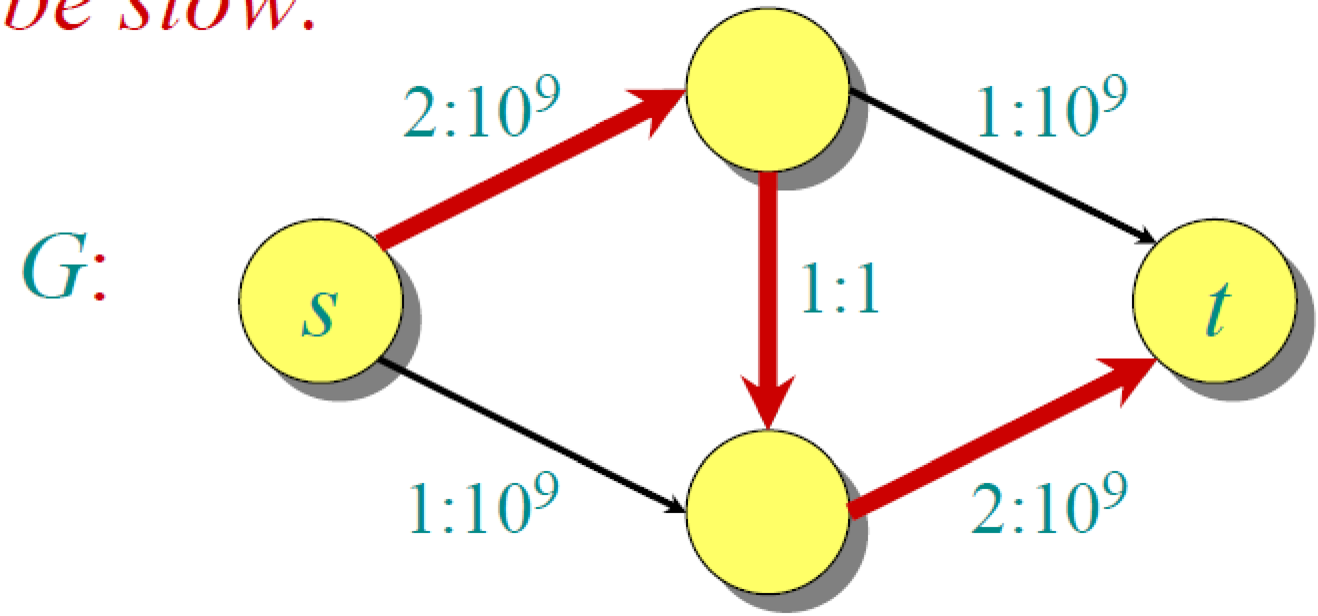
Problem：Time Complexity

Can be slow:



Problem：Time Complexity

Can be slow:



2 billion iterations on a graph with 4 vertices!

Time Complexity

$$O(\,F\,(n + m)\,)$$

where F is the maximum flow value, n is the number of vertices, and m is the number of edges

The problem with this algorithm, however, is that it is strongly dependent on the maximum flow value F.

if $F=2^n$ -----

Then, along came Edmonds & Karp...

Edmonds & Karp Algorithm

Breadth-First Search

- **Input:**
  - Graph *G* = (*V*, *E*), either directed or undirected,
  - *source vertex s* ∈ *V*.

- **Output:** **for all *v* ∈ *V***

  - **$d[v]$ = length of shortest path from *s* to *v***
    ($d[v] = \infty$ if *v* is not reachable from *s*).

  - **$\pi[v] = u$ if (*u*, *v*) is last edge on shortest path $s \rightsquigarrow v$.**
    - ***u* is *v*'s predecessor.**

  - **breadth-first tree = a tree with root *s* that contains all reachable vertices.**

Definitions on BSF

- **Path between vertices u and v:**

  vertices (v_1, v_2, …, v_k) such that

  $u=v_1$ and $v=v_k$,

  $(v_i,v_{i+1}) \in E$, for all $1\leq i \leq k-1$.

- **Length of the path: Number of edges in the path.**

- **Path is simple if no vertex is repeated.**

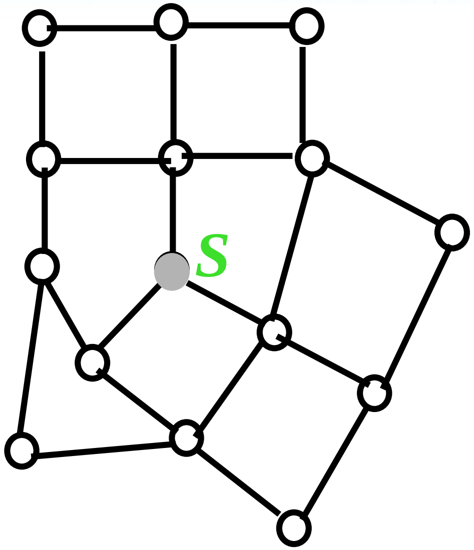
Principle of Breadth-First Search

- **Expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier.**
  - **A vertex is "discovered" the first time it is encountered during the search.**
  - **A vertex is "finished" if all vertices adjacent to it have been discovered.**

BFS for Shortest Paths

Colors the vertices to keep track of progress.

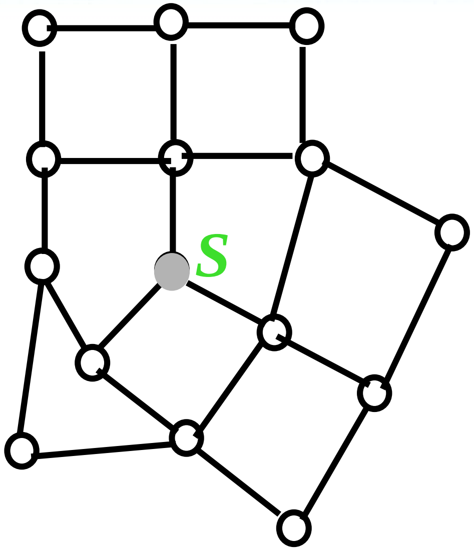
- ○ Undiscovered
- ● Discovered
- ● Finished

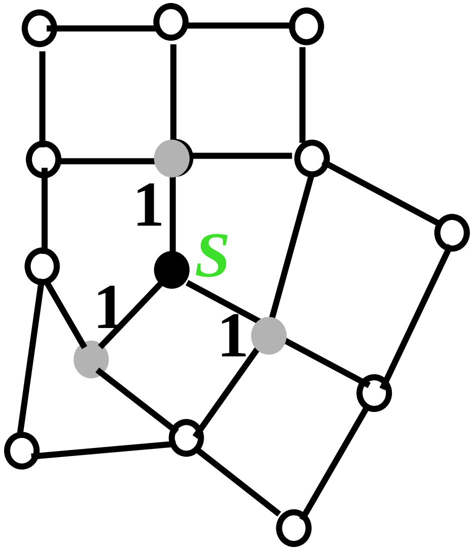


BFS for Shortest Paths

Colors the vertices to keep track of progress.

- ○ Undiscovered
- ● Discovered
- ● Finished

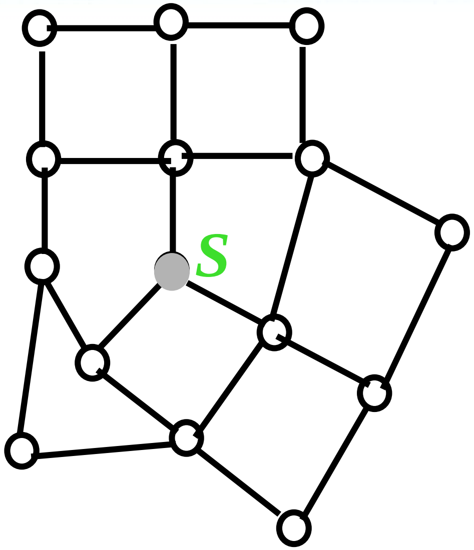


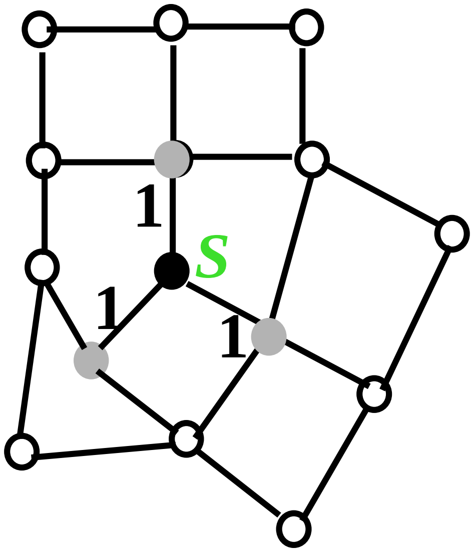


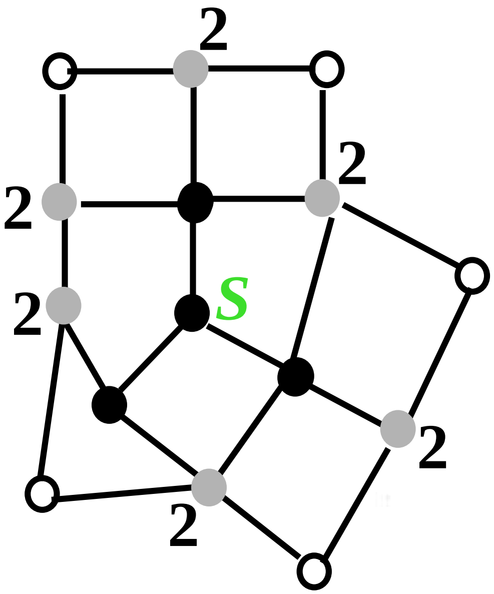
BFS for Shortest Paths

Colors the vertices to keep track of progress.

- ○ Undiscovered
- ● Discovered
- ● Finished







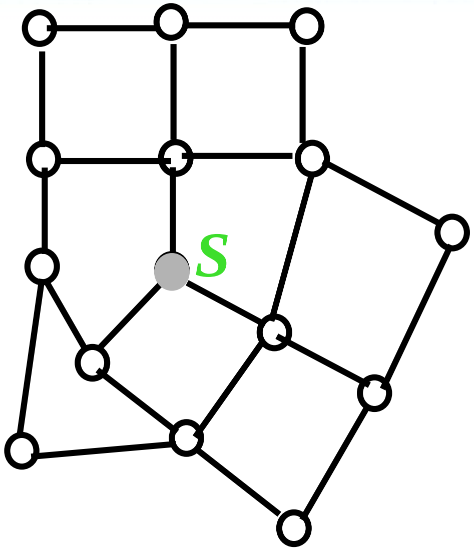
BFS for Shortest Paths

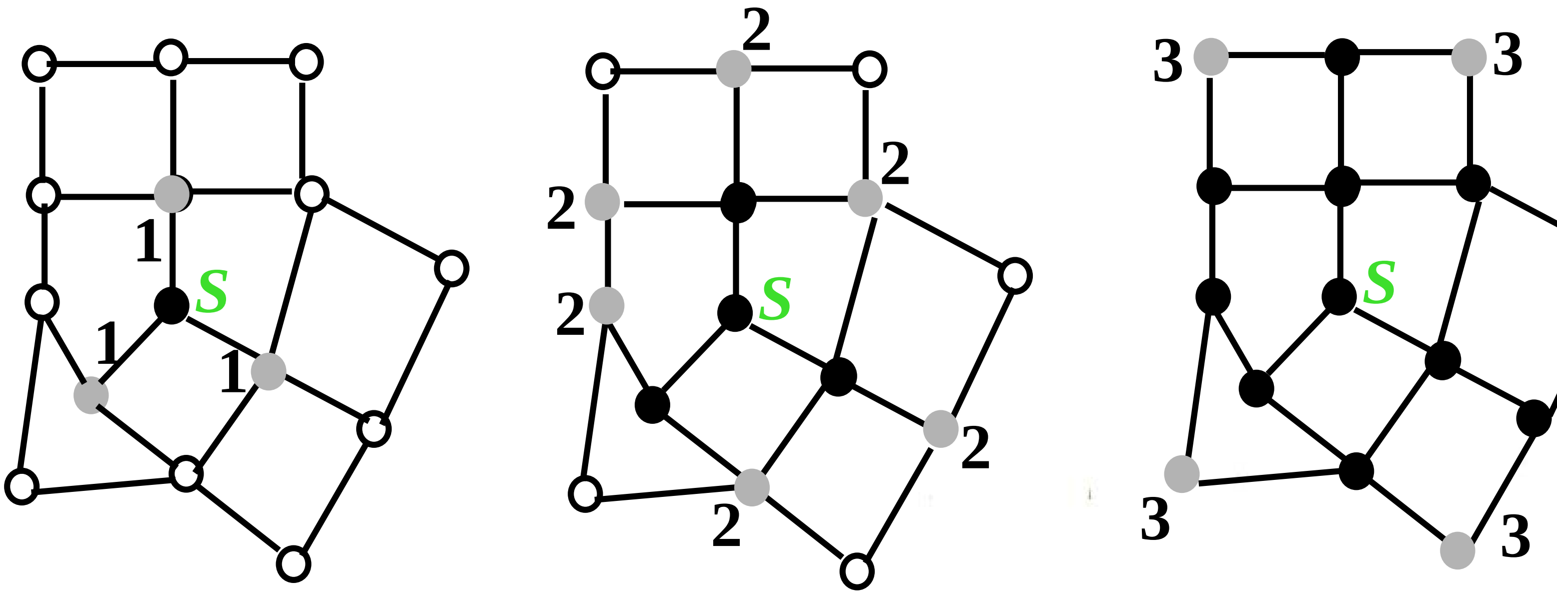
Colors the vertices to keep track of progress.

○ Undiscovered

● Discovered

● Finished





BFS(G,s)

```
for each vertex u in $V[G] - \{s\}$
    do $color[u] \leftarrow$ white
       $d[u] \leftarrow \propto$
       $\pi[u] \leftarrow$ nil
color[$s$] $\leftarrow$ gray
d[$s$] $\leftarrow 0$
$\pi[s] \leftarrow$ nil
$Q \leftarrow \Phi$
enqueue($Q$,s)
while $Q \neq \Phi$
    do u $\leftarrow$ dequeue(Q)
        for each $v$ in Adj[$u$]
            do if color[$v$] = white
                then color[$v$] $\leftarrow$ gray
                     $d[v] \leftarrow d[u] + 1$
                     $\pi[v] \leftarrow u$
                     enqueue($Q$,$v$)
        color[$u$] $\leftarrow$ black
```

BFS(G,s)

```
for each vertex u in V[G] – {s}
        do $color[u] \leftarrow$ white
           $d[u] \leftarrow \propto$
           $\pi[u] \leftarrow$ nil
color[$s$] ← gray
d[$s$] ← 0
$\pi[s] \leftarrow$ nil
$Q \leftarrow \Phi$
enqueue($Q$,s)
while $Q \neq \Phi$
        do u ← dequeue(Q)
                for each $v$ in Adj[$u$]
                        do if color[$v$] = white
                                then color[$v$] ← gray
                                     $d[v] \leftarrow d[u] + 1$
                                     $\pi[v] \leftarrow u$
                                     enqueue($Q$,$v$)
                color[$u$] ← black
```

white: undiscovered
gray: discovered
black: finished

BFS(G,s)

```
for each vertex u in V[G] – {s}
    do $color[u] \leftarrow$ white
       $d[u] \leftarrow \propto$
       $\pi[u] \leftarrow$ nil
color[$s$] $\leftarrow$ gray
d[$s$] $\leftarrow 0$
$\pi[s] \leftarrow$ nil
$Q \leftarrow \Phi$
enqueue($Q$,s)
while Q $\neq \Phi$
    do u $\leftarrow$ dequeue(Q)
        for each $v$ in Adj[$u$]
            do if color[$v$] = white
                then color[$v$] $\leftarrow$ gray
                     $d[v] \leftarrow d[u] + 1$
                     $\pi[v] \leftarrow u$
                     enqueue($Q$,$v$)
        color[$u$] $\leftarrow$ black
```

white: undiscovered
gray: discovered
black: finished

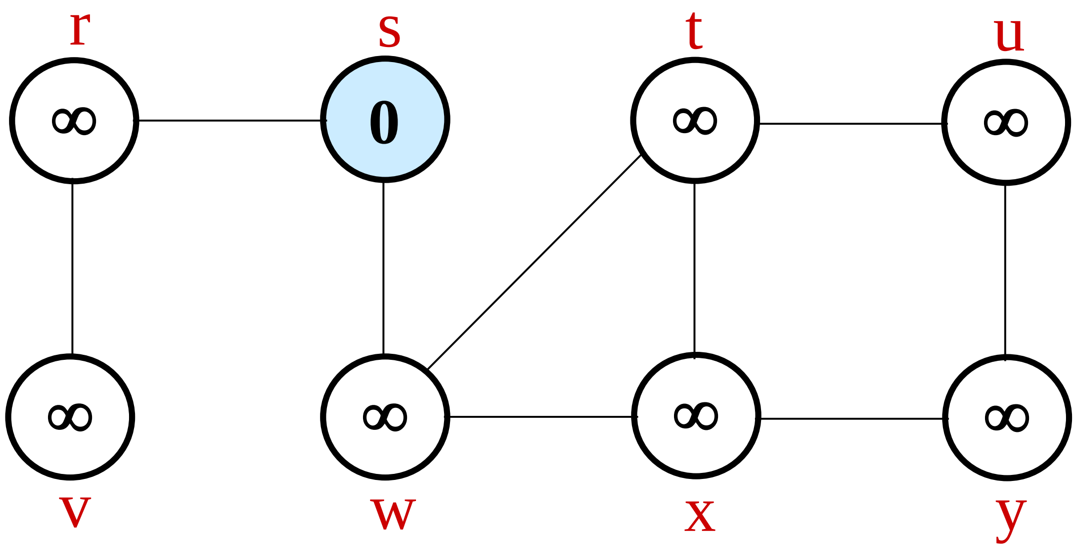
Q: a queue of discovered vertices

color[v]: color of v

d[v]: distance from s to v

$\pi[u]$: predecessor of v

Example (BFS)

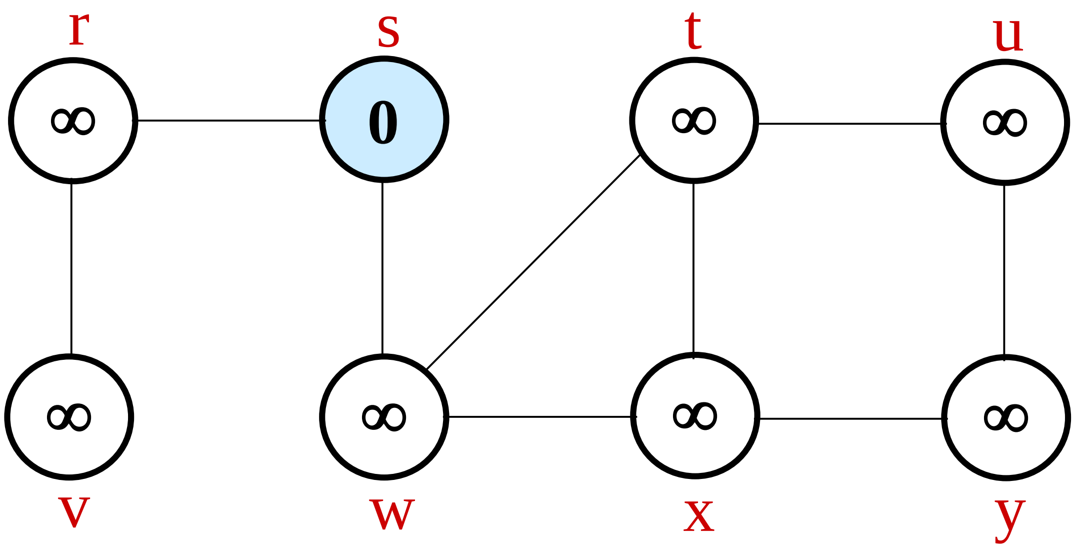


Q: s

0

frontier

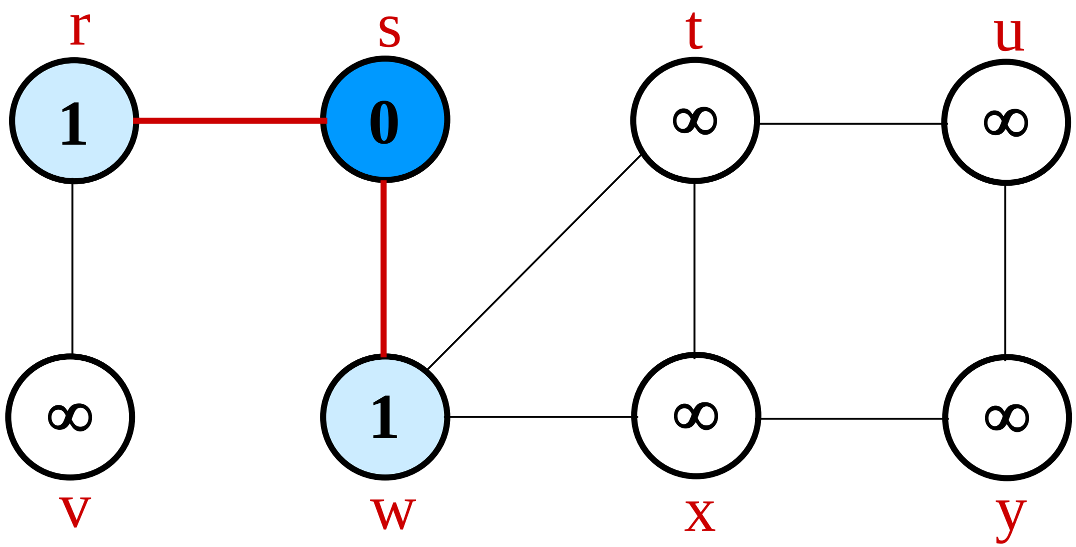
Example (BFS)



Q: s
0

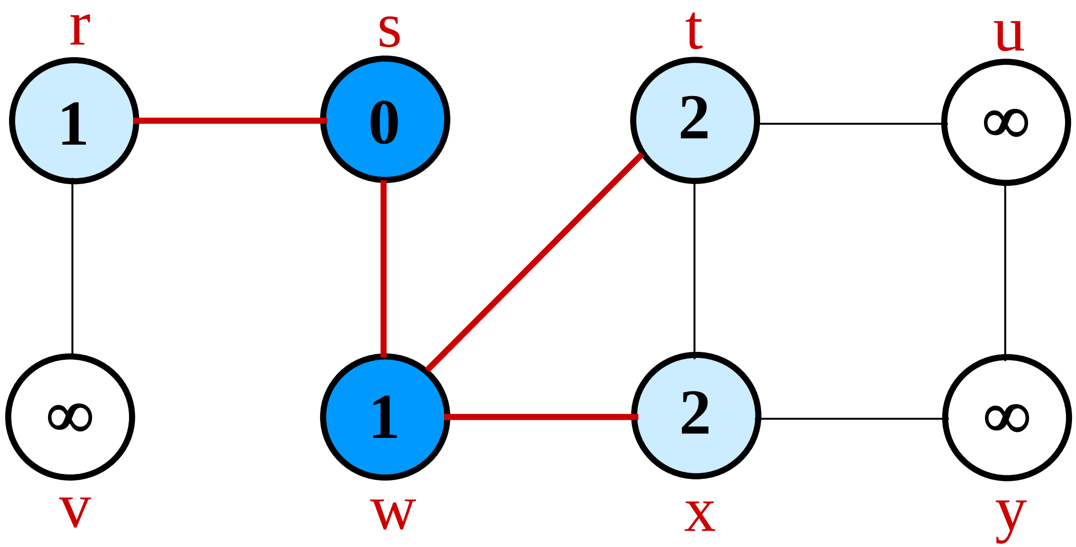
frontier

Example (BFS)



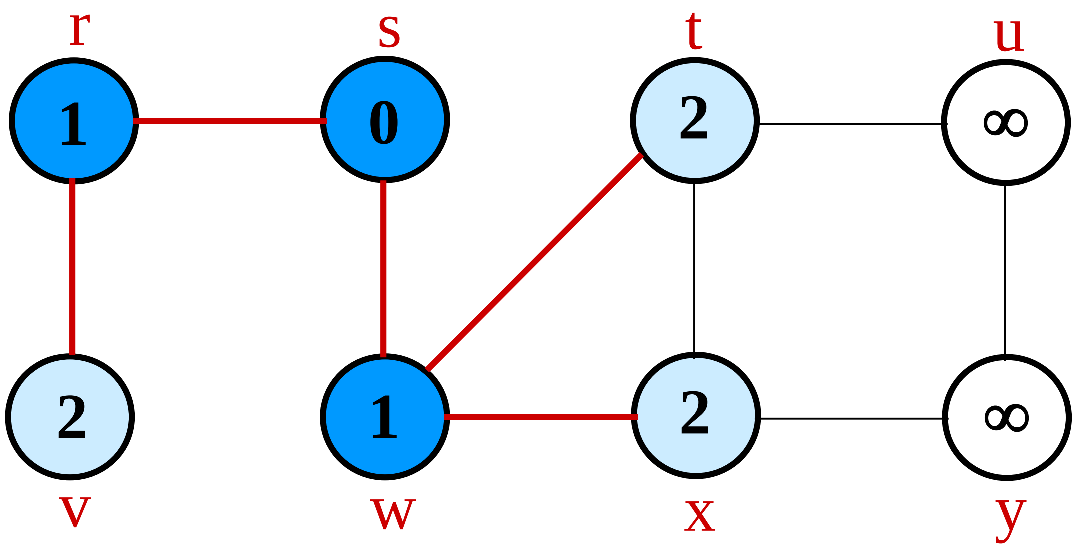
Q: w r
1 1

Example (BFS)



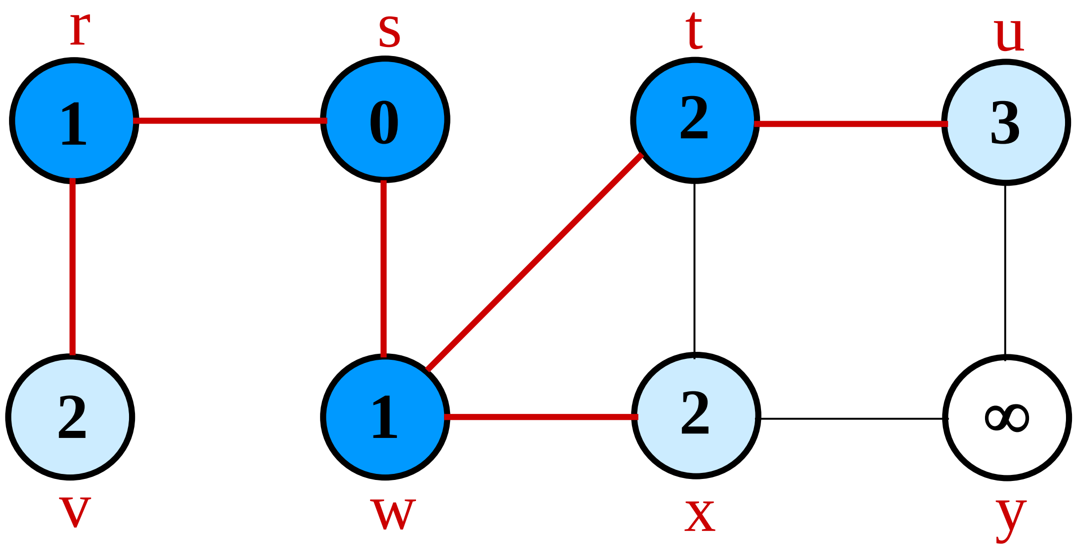
Q:	r	t	x
	1	2	2

Example (BFS)



Q: t x v
2 2 2

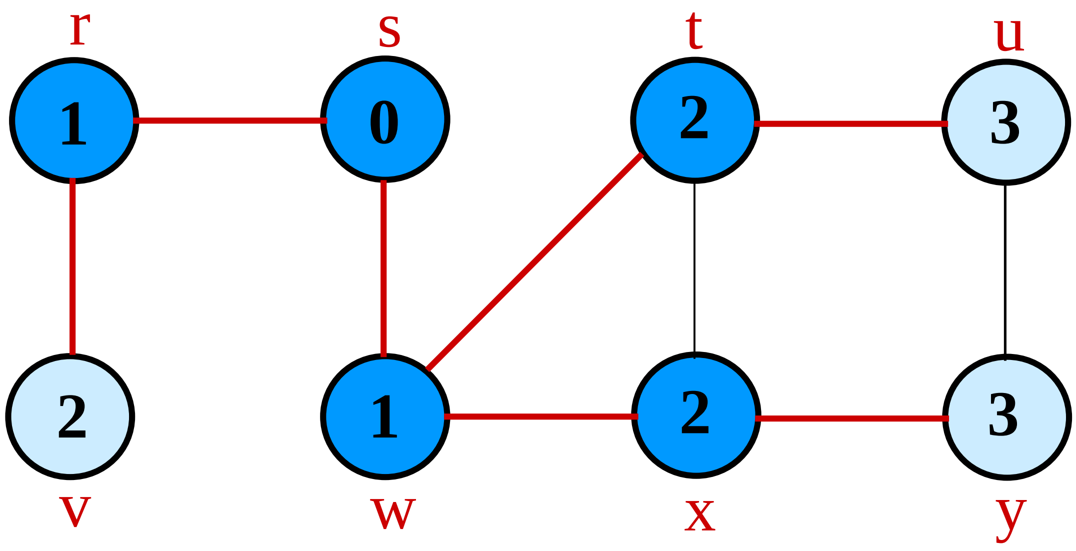
Example (BFS)



Q: x v u

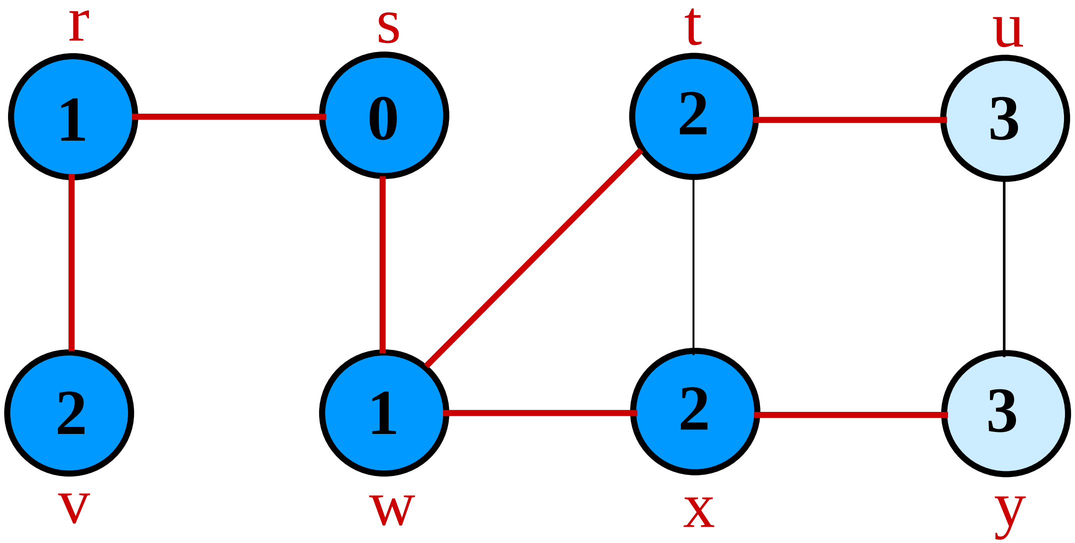
2 2 3

Example (BFS)



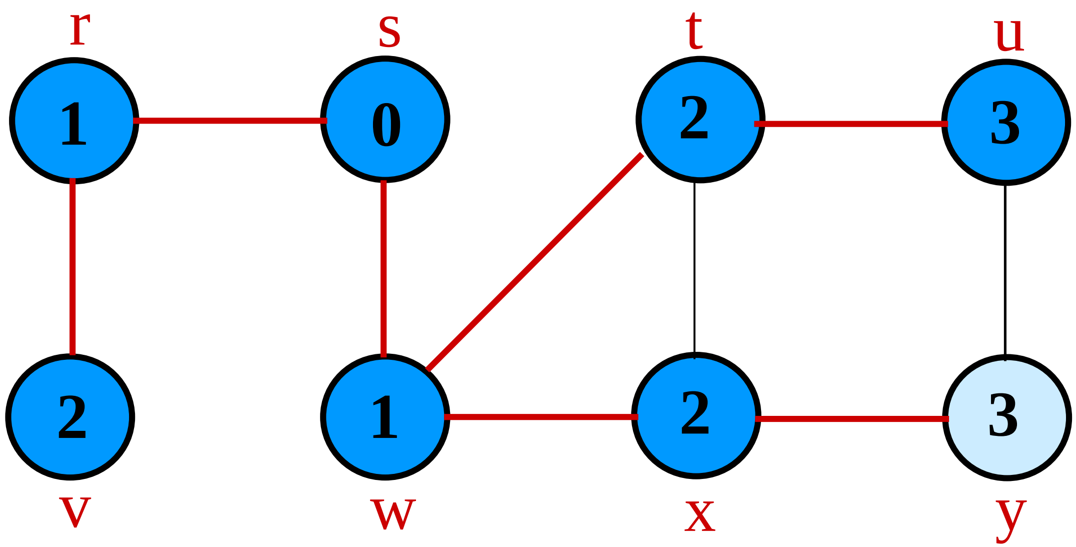
Q: v u y
2 3 3

Example (BFS)



Q:	u	y
	3	3

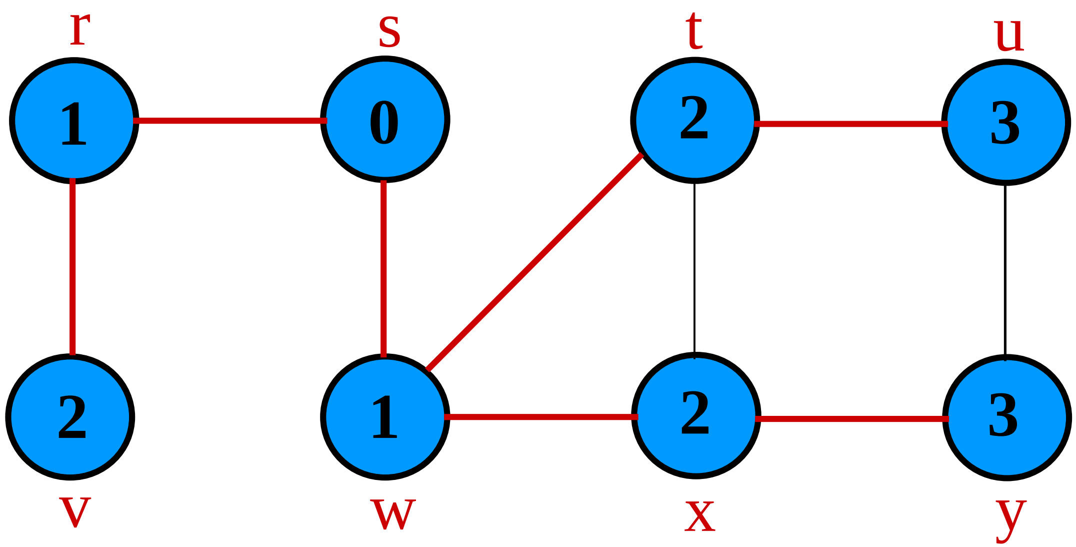
Example (BFS)



Q: y

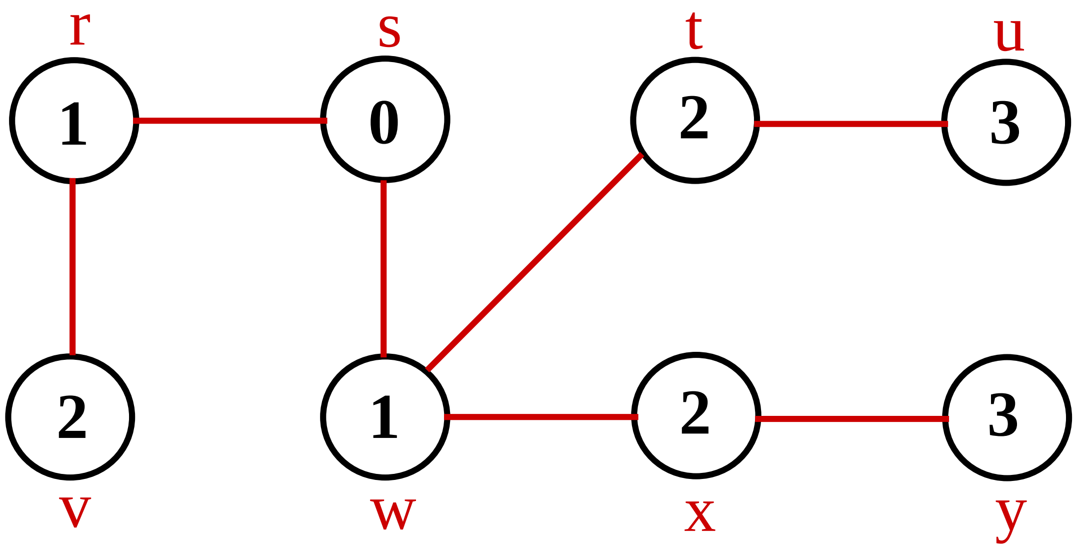
3

Example (BFS)



Q: $\emptyset$

Example (BFS)



BF Tree

Breadth-First Tree

- **Predecessor sub-graph of $G = (V, E)$ with source s is**
  $G_\pi = (V_\pi, E_\pi)$ where
  - $V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} + \{s\}$
  - $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$
- **G_π is a breadth-first tree if:**
  - **V_π consists of the vertices reachable from s**
  - **for all $v \in V_\pi$, there is a unique simple path from s to v in G_π**
  - **the path is also a shortest path from s to v in G.**
- **The edges in E_π are called tree edges.**
  $|E_\pi| = |V_\pi| - 1$.

Analysis of BFS

- **Initialization takes $O(|V|)$.**

- **Traversal Loop**
  - **Each vertex is enqueued and dequeued at most once, so the total time for queuing is $O(|V|)$.**
  - **The adjacency list of each vertex is scanned at most once.**
  - **The sum of lengths of all adjacency lists is $\Theta(|E|)$.**

- **Total running time of BFS is $O(|V|+|E|)$**
- **Correctness of BFS (see Dijkstra later)**

Edmonds & Karp Algorithm

- **Find the augmenting path using breadth-first search.**

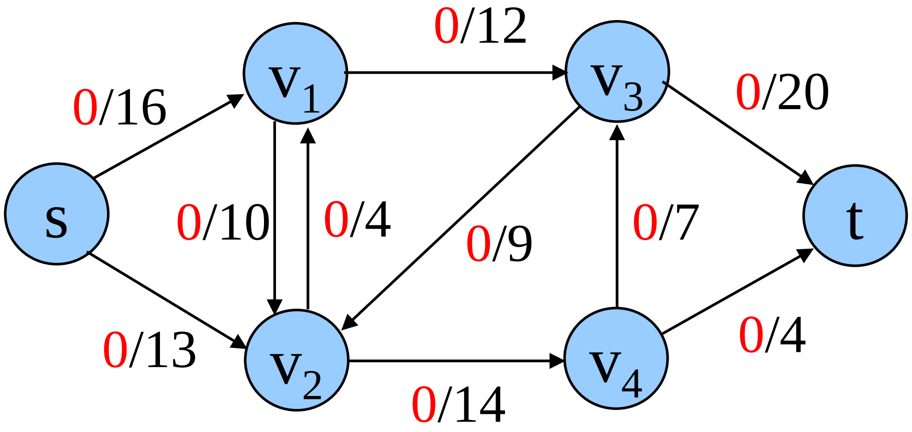
- **Breadth-first search gives the shortest path for graphs (Assuming the length of each edge is 1.)**

- **Time complexity of Edmonds-Karp algorithm is $O(|V||E|^2)$.**

- **The proof is very hard!.**

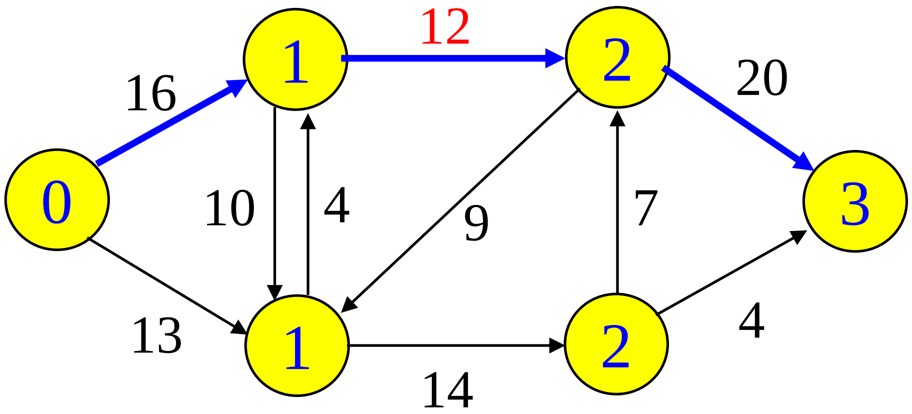
Example

Flows



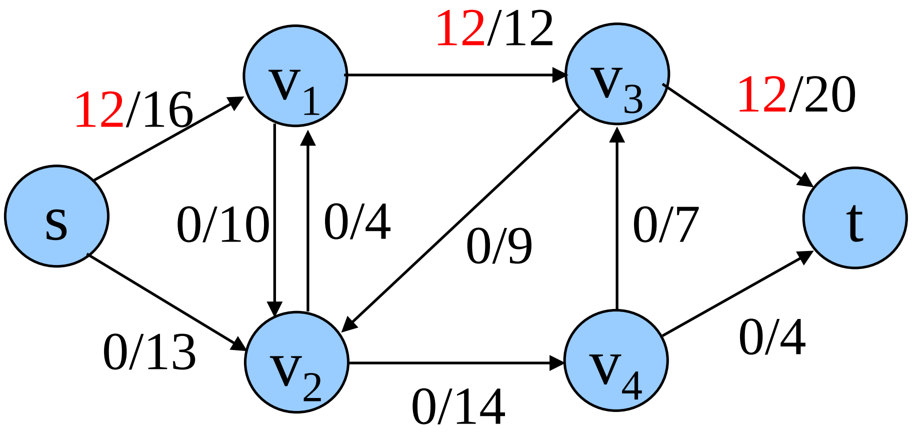
Residual Networks

BFS



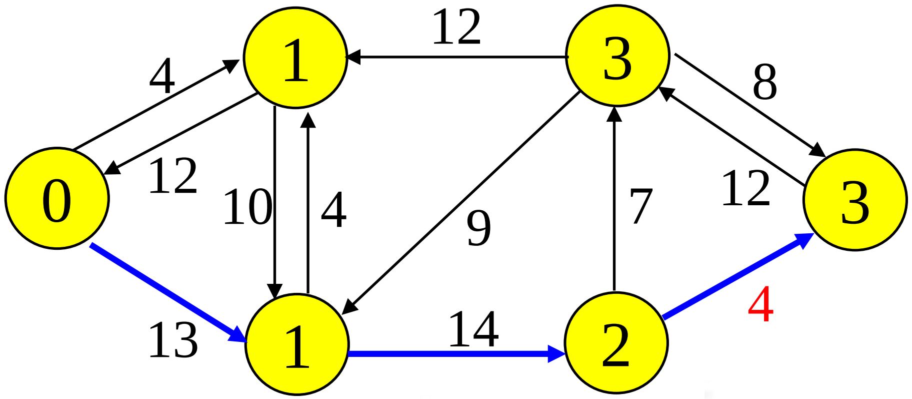
Example

Flows



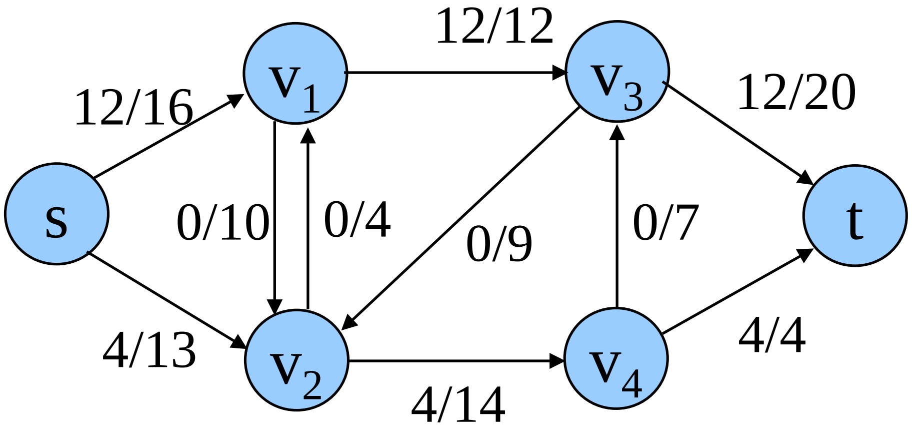
Residual Networks

BFS



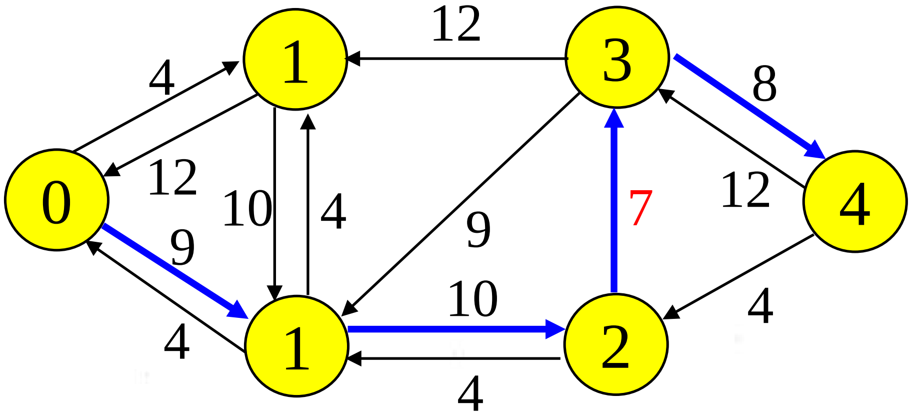
Example

Flows



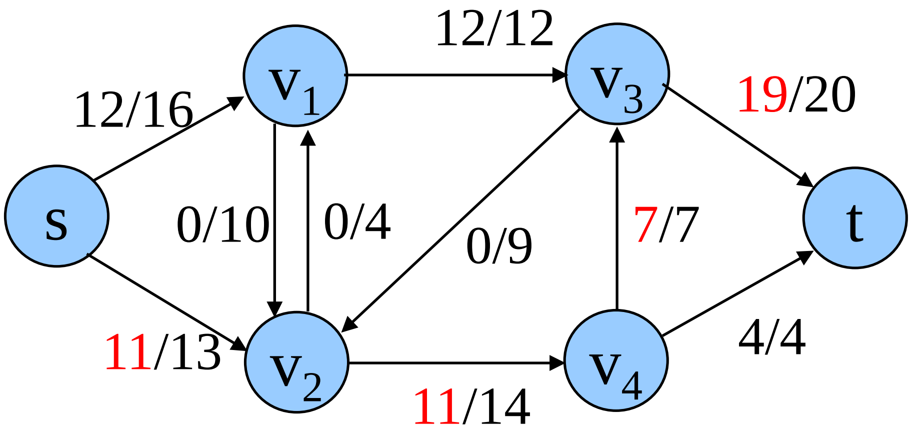
Residual Networks

BFS



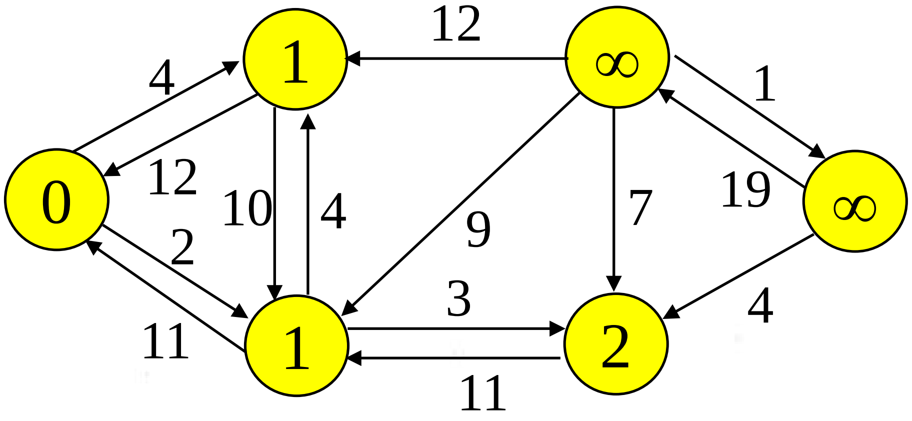
Example

Flows



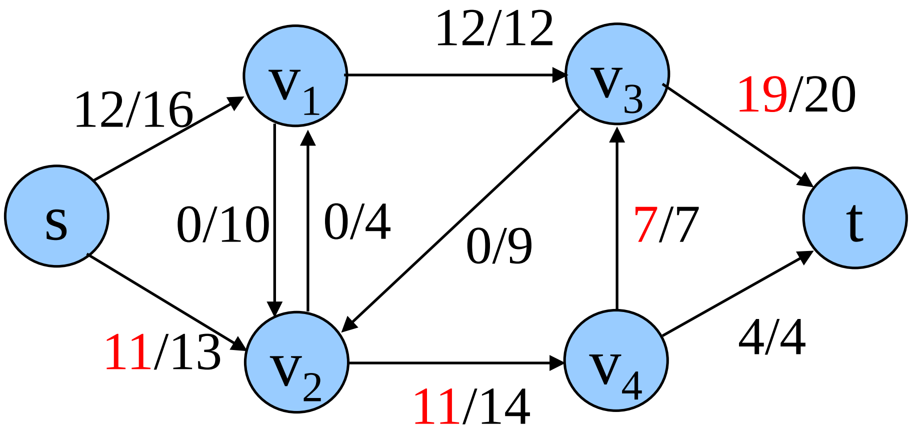
Residual Networks

BFS



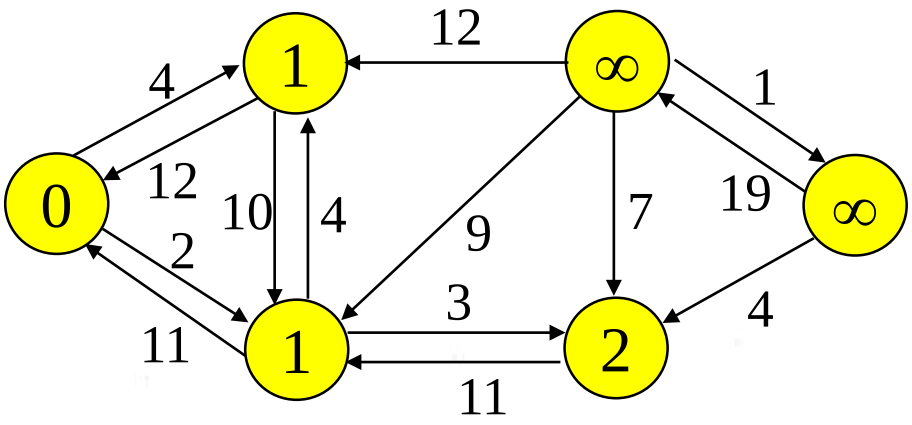
Example

Flows



Residual Networks

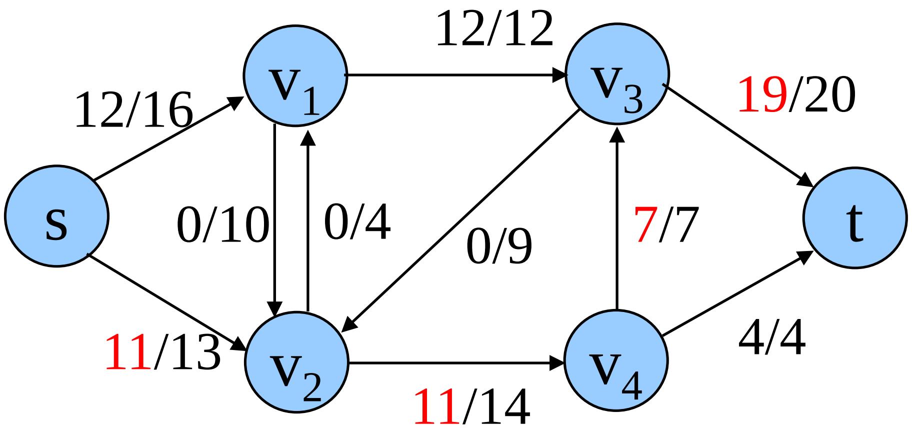
BFS



No path to sink

Example

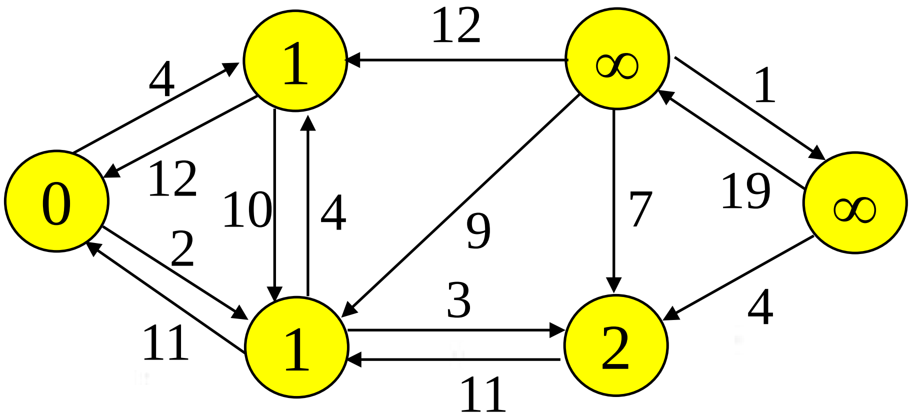
Flows



Maximum!

Residual Networks

BFS



No path to sink

Exercises

- **26.2-3**
- **26.2-8**

Exercises

- 26.2-3
- 26.2-8

No path to sink

算法分析课程组

重庆大学计算机学院

End of Section.