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**
- **Edmonds & Karp Algorithm**
- **Applications**

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*

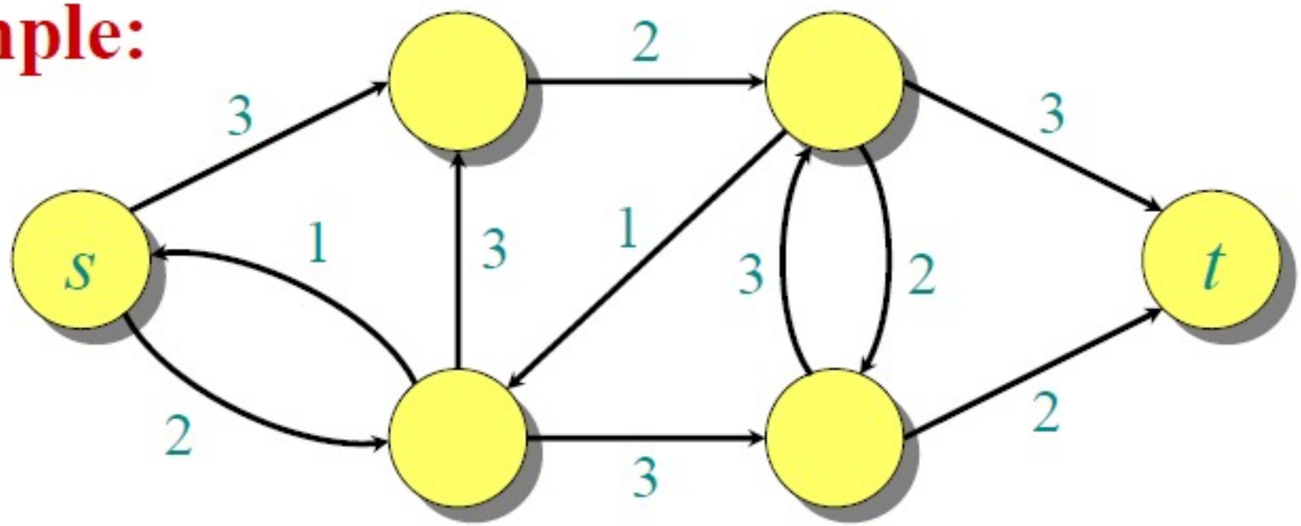
"Life is flow; flow is life."
-Ford & Fulkerson, *Ford & Fulkerson Algorithm*

"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: $p : V \times V \to \mathbb{R}$

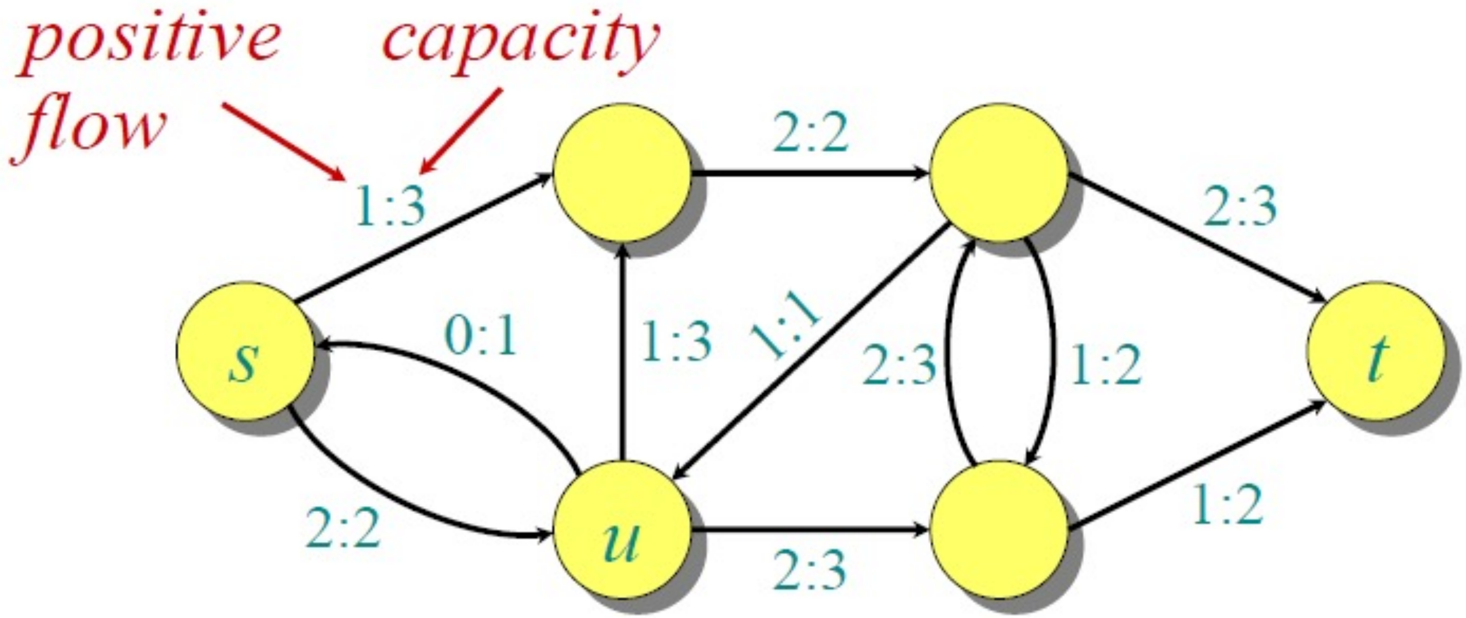
  - ***Capacity constraint:*** For all $u, v \in V$,

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

  - ***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



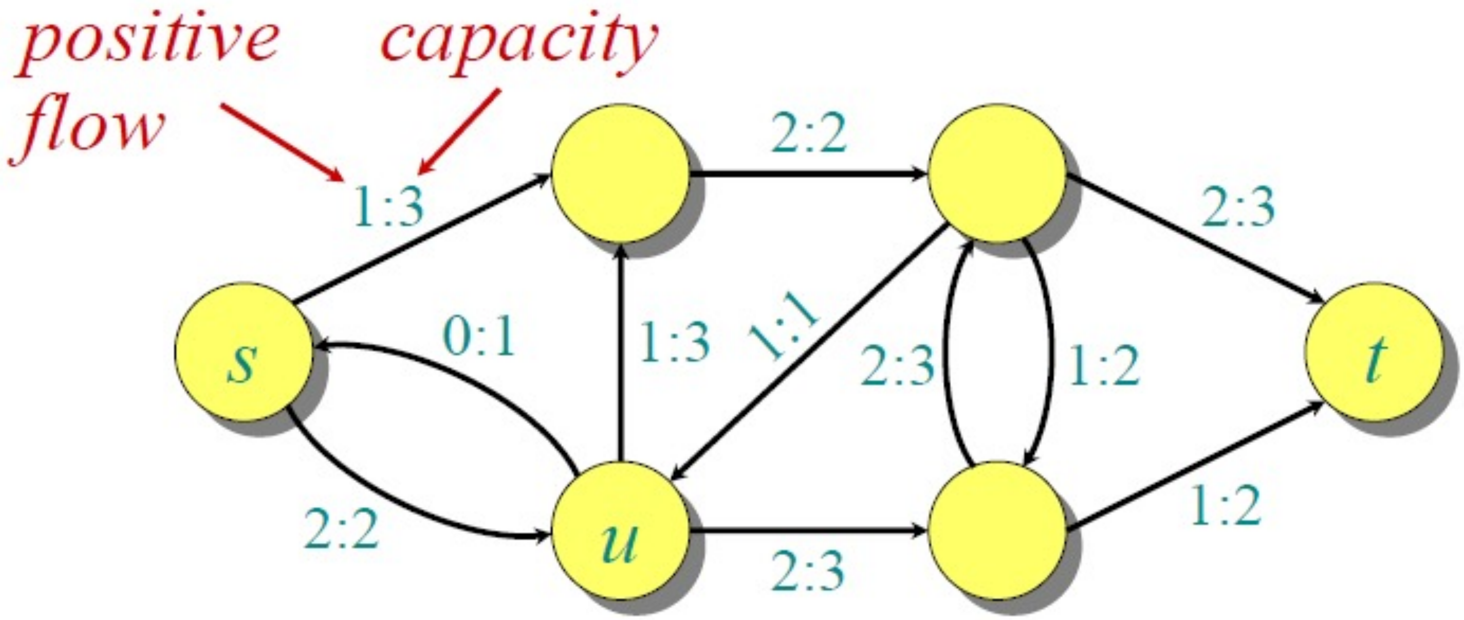
Flow conservation (like Kirchoff's current law):

- 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:

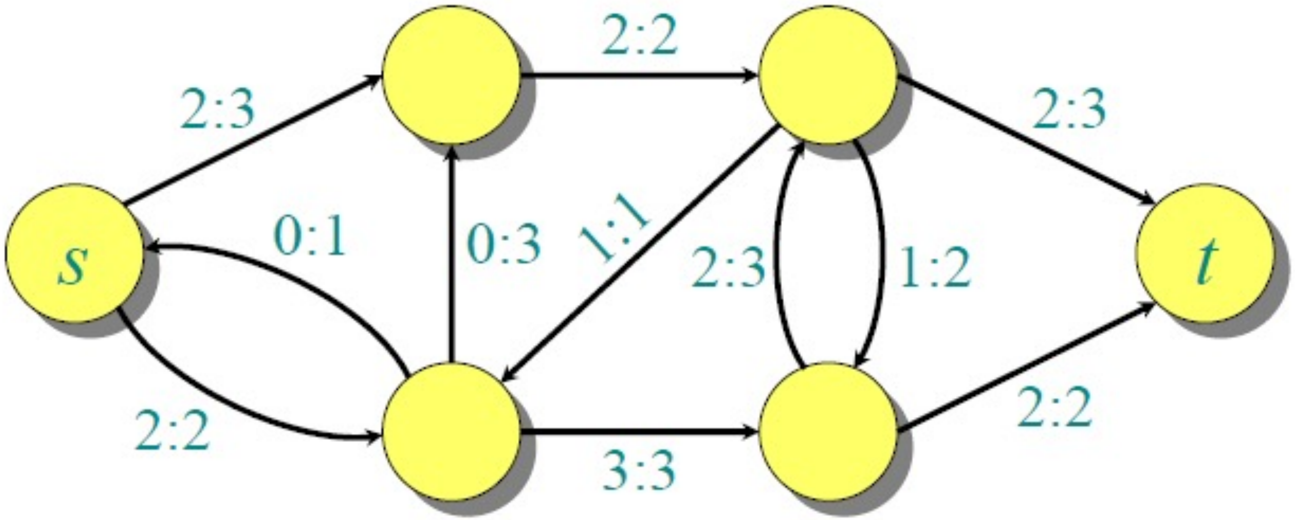
$$\sum_{v \in V} p(s,v) - \sum_{v \in V} p(v,s).$$



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

The Maximum-Flow Problem

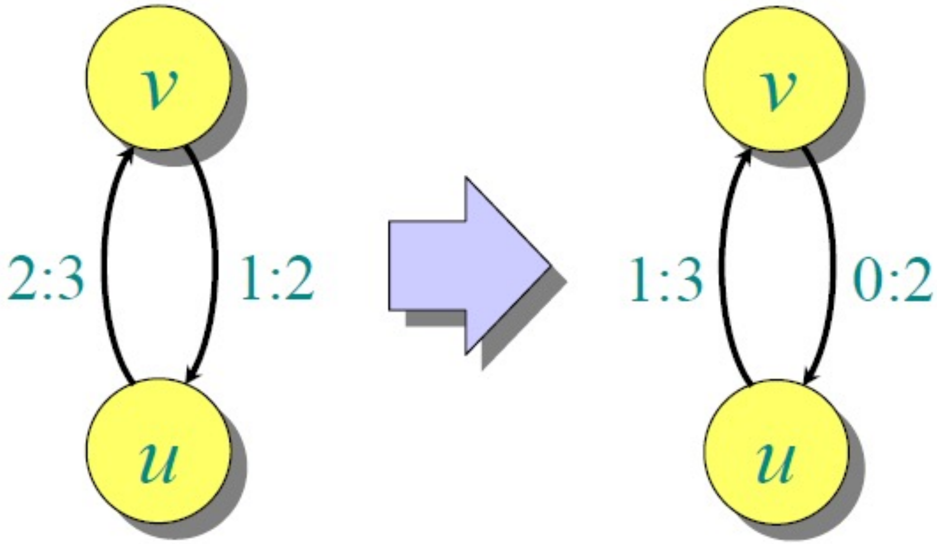
Maximum-flow problem: Given a flow network G, find a flow of maximum value on G.



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.

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

  (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.

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

(Proof).

$$f(X \cup Y, Z) = \sum_{s \in X \cup Y} \sum_{t \in Z} f(s,t)$$

$$= \left(\sum_{s \in X} + \sum_{s \in Y}\right) \sum_{t \in Z} f(s,t) \qquad \because X \cap Y = \emptyset$$

$$= \sum_{s \in X} \sum_{t \in Z} f(s,t) + \sum_{s \in Y} \sum_{t \in Z} f(s,t)$$

$$= f(X,Z) + f(Y,Z)$$

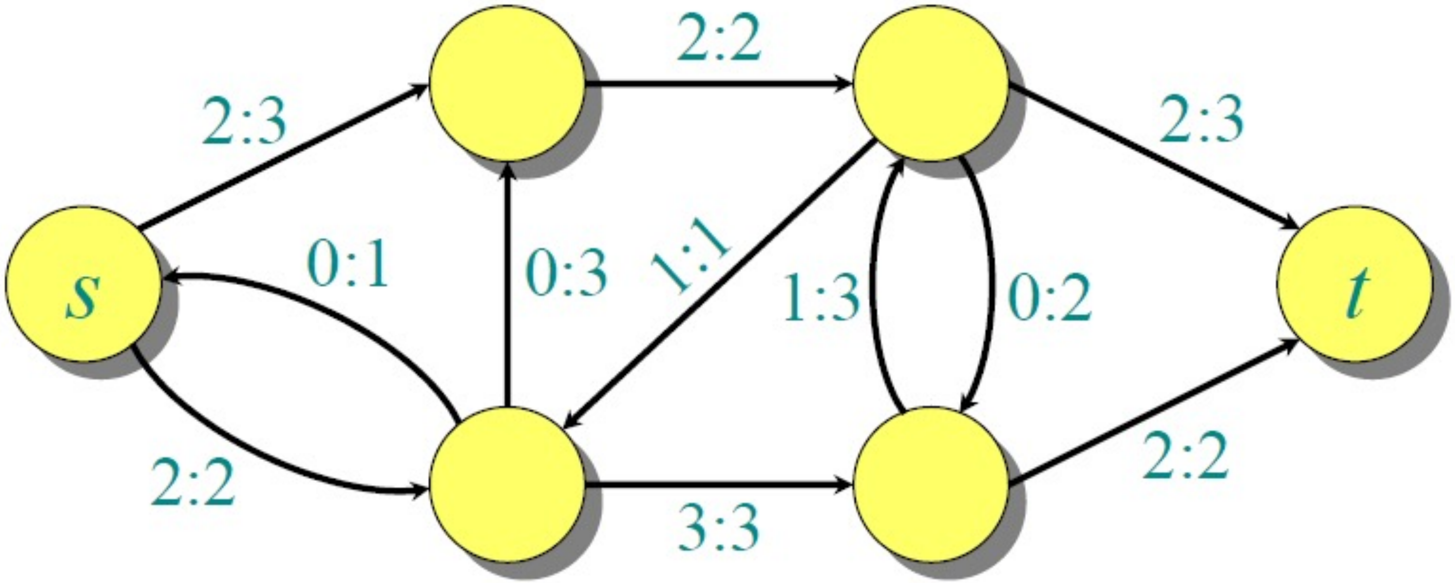
Simple Properties of Net Flow

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

Proof.

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

Net Flow into Sink

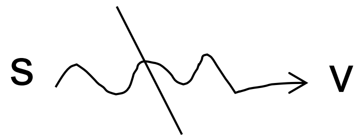


$$|f| = f(s, V) = 4 \qquad f(V, t) = 4$$

思考题

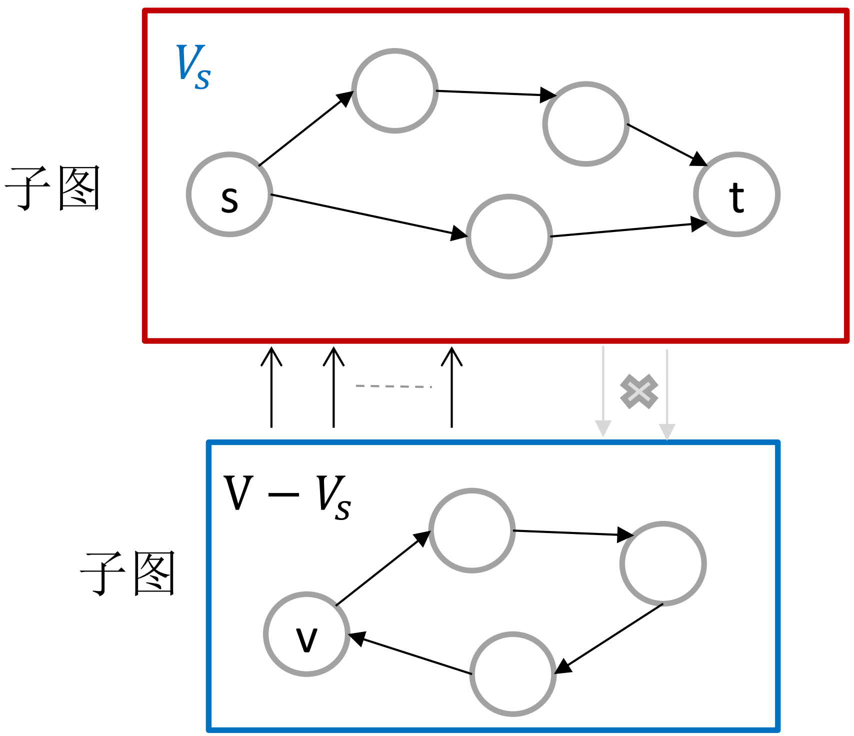
可删除

如果流量图中有节点v，从源节点s到达该节点的路径不存在，则有最大流，从节点v流向其它节点的流为零，且从其它节点流向v的流也为零。



$$\forall u \in V: \quad p(u,v) = 0 \wedge p(v,u) = 0$$

设s能够到达的节点集合为 V_s， 显然 $v \notin V_s$ *and* $\forall v' \in V - V_s \ : s \nrightarrow v'$



$$f(V - V_s, V) = 0$$

$$\Downarrow$$

$$f(V - V_s, V_s) = 0$$

$$\Downarrow$$

$$\forall x \in V - V_s \ \forall y \in V_s: \quad p(x,y) = 0$$

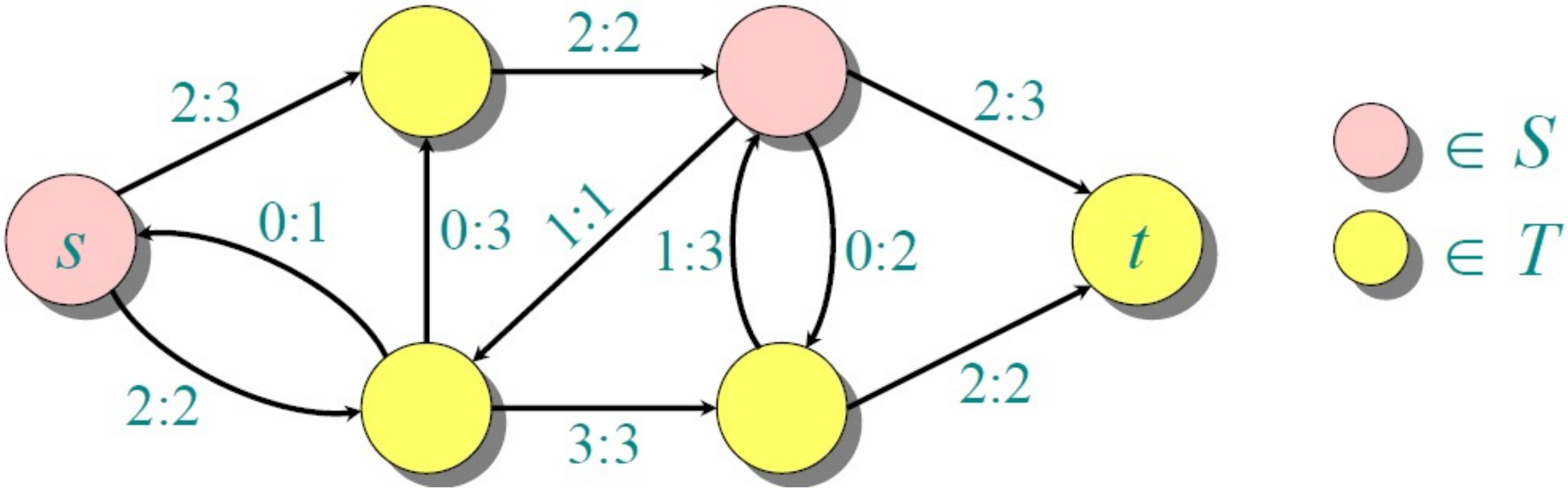
$$\Downarrow$$

$$\because f(V - V_s, V - V_s) = 0$$

$$\therefore \forall w,z \in V - V_s: \quad p(w,z) = 0 \ \textit{is valid}$$

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 $t \in T$.

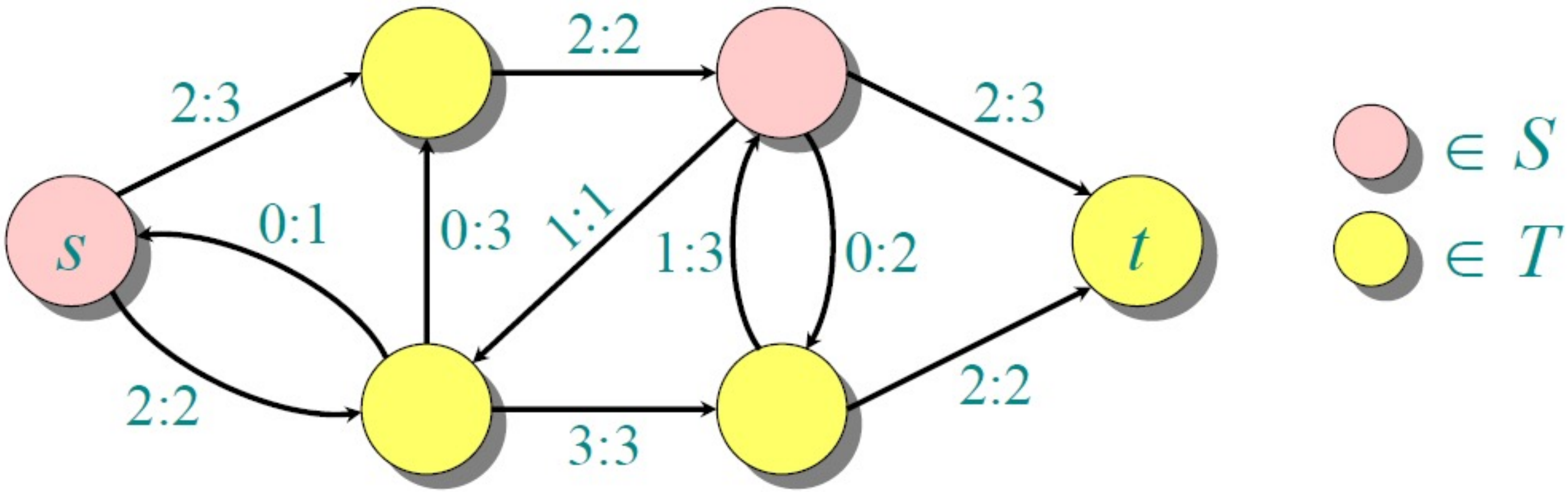
Lemma. $|f| = f(S, T)$.

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.

Proof.

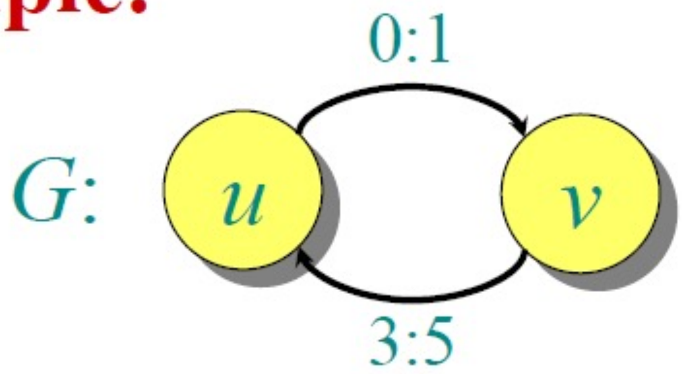
$$\begin{aligned}|f| &= f(S,T) \\ &= \sum_{u\in S}\sum_{v\in T} f(u,v) \\ &\le \sum_{u\in S}\sum_{v\in T} c(u,v) \\ &= c(S,T)\,.\end{aligned}$$

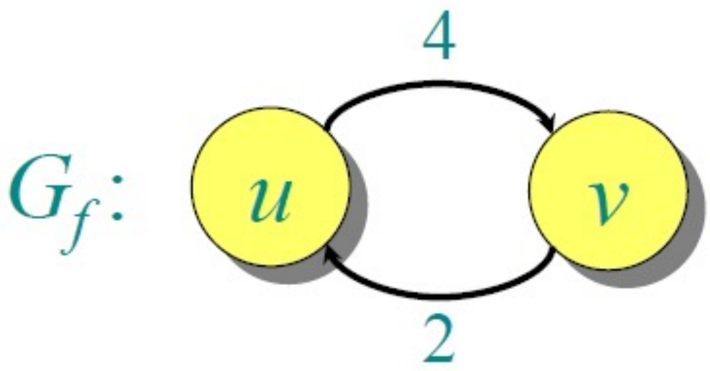
Residual Network

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:

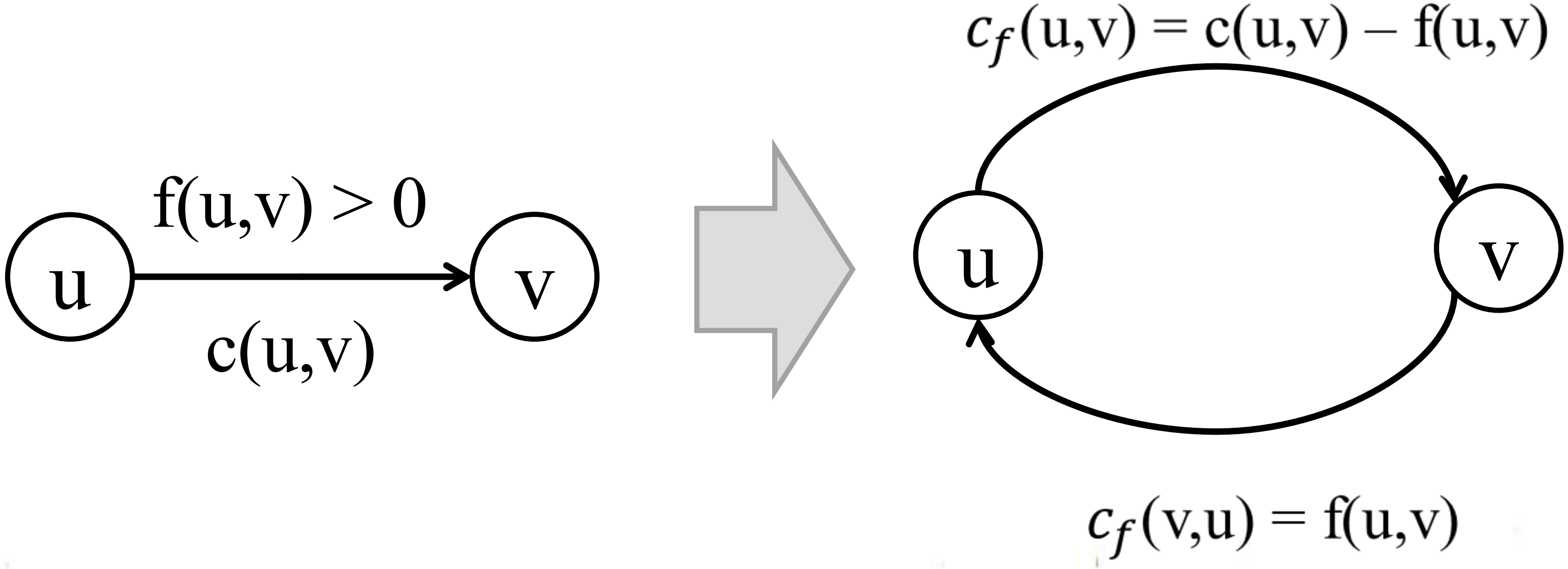




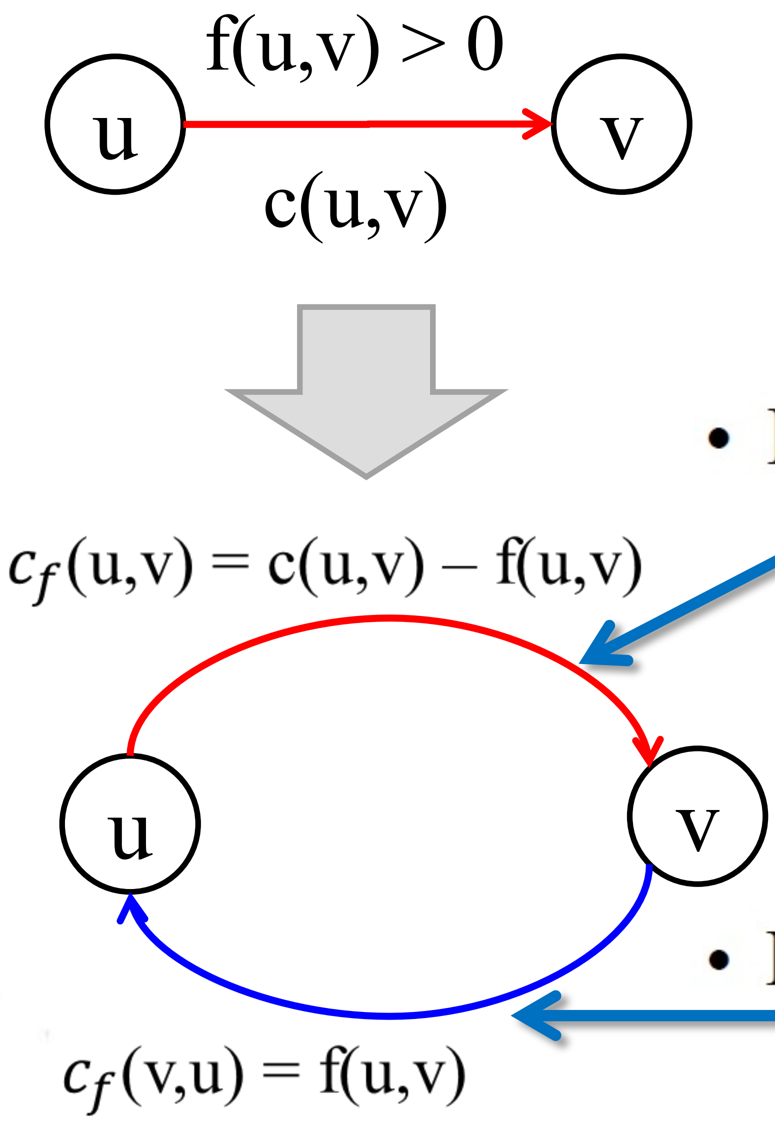
Edges in E_f admit more flow.

Residual Network

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

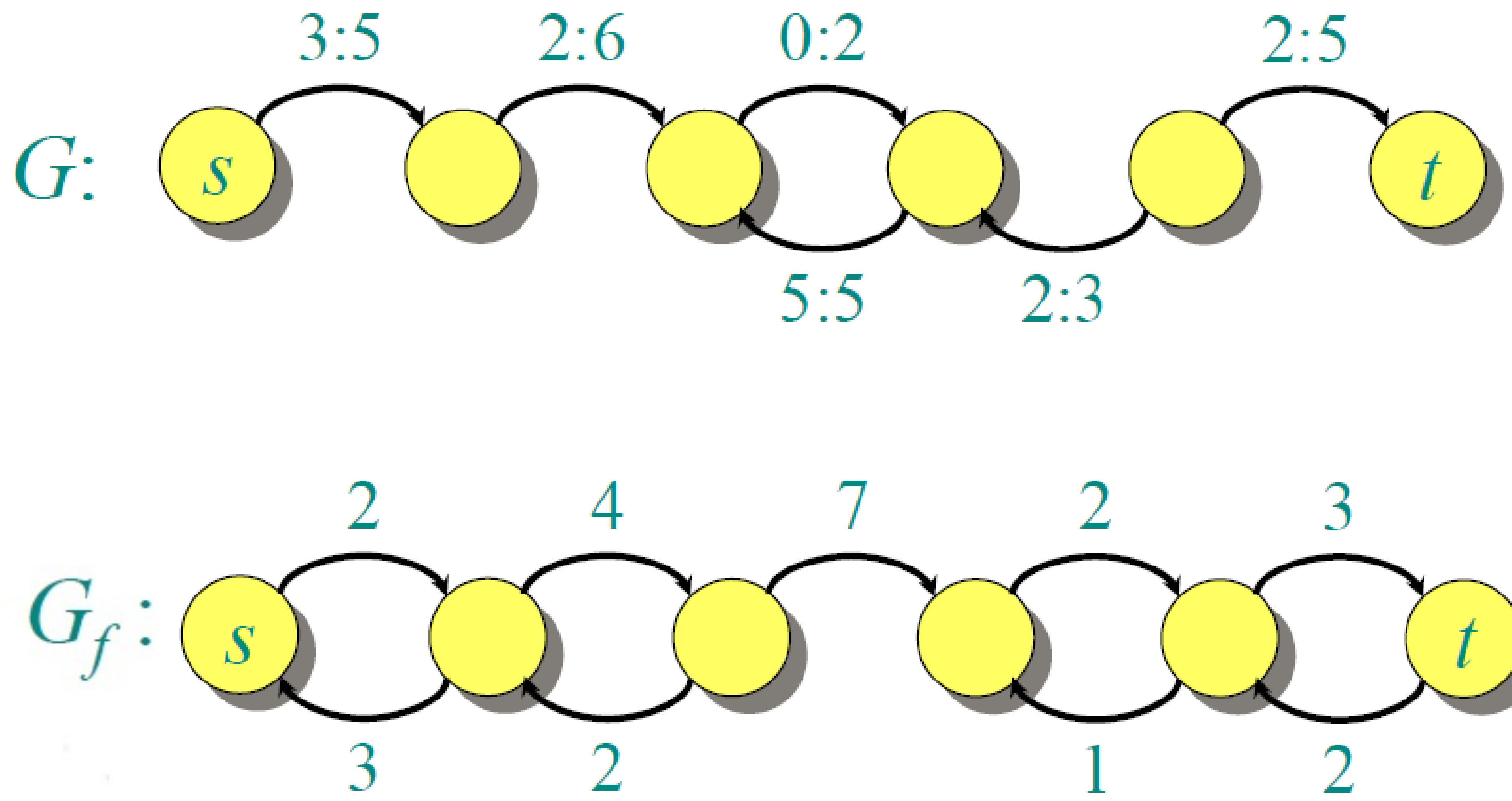


Residual Network



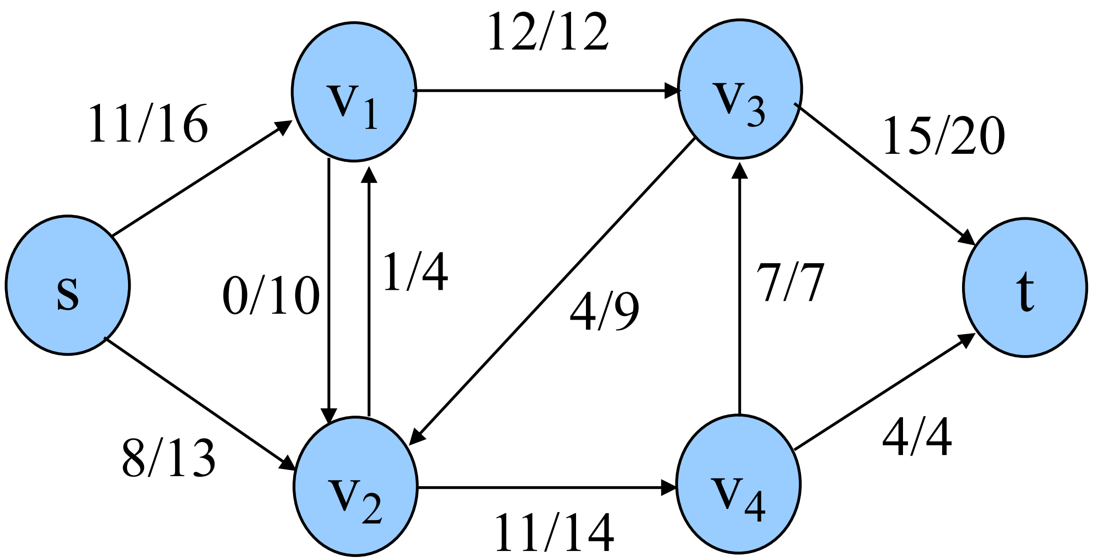
- **Forward Edges**
  flow(u,v) < **capacity**(u,v)
  flow can be increased!
- **Backward Edges**
  flow(u,v) > 0
  flow can be decreased!

Residual Network Example



Short Test in Class

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



Exercises

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

Augmenting Path

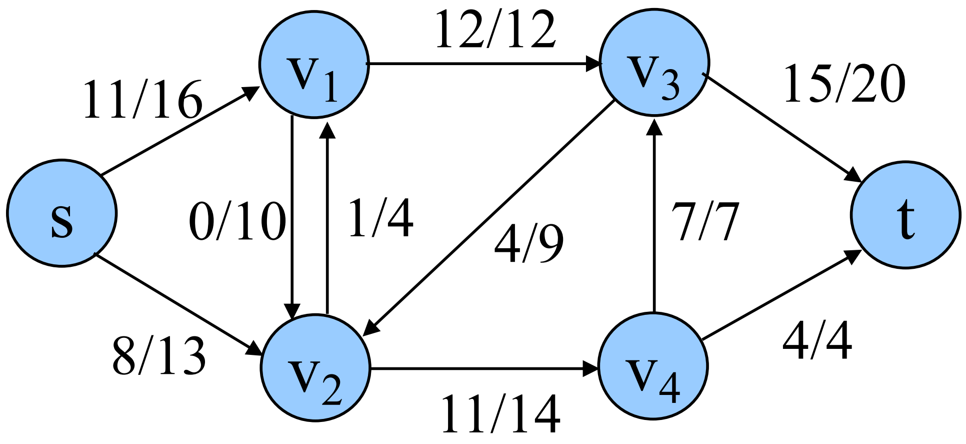
An ***augmenting path*** *p* is a simple path from s to t in the residual network G_f of a flow network *G*.

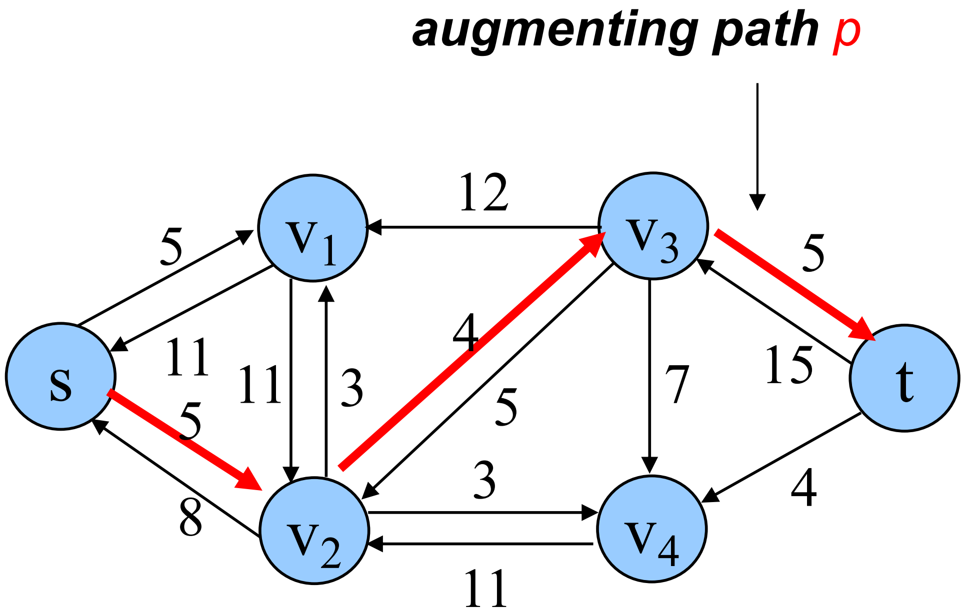
residual capacity of *p*

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

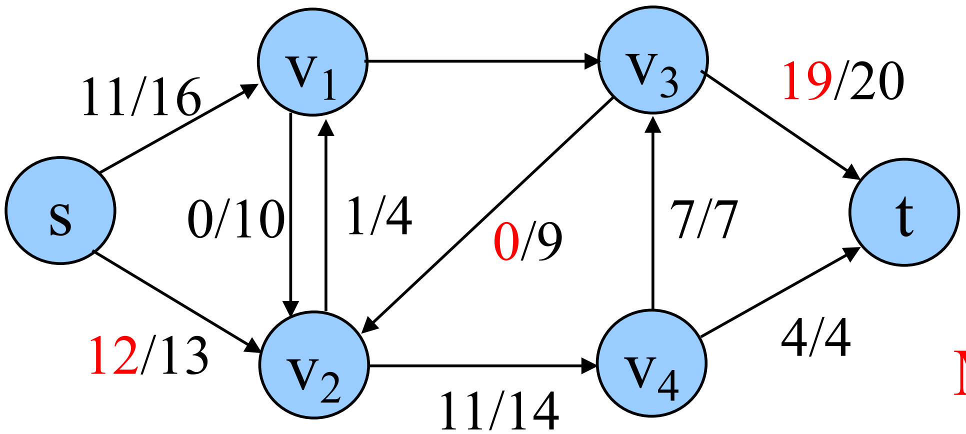
the maximum flow $|f|$ can increased by increasing the flow on each edge in *p*

Augmenting Path Example





$$c_f(p) = 4$$



Maximum flow increased by 4!

Maximum Flow Theorem

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

Flow is maximum $\Rightarrow$ No augmenting path

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

No augmenting path $\Rightarrow$ 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 $|f| \le c(S, T)$ for any cut (S, T)

$\Downarrow$

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

$\Downarrow$

$|f|$ 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.

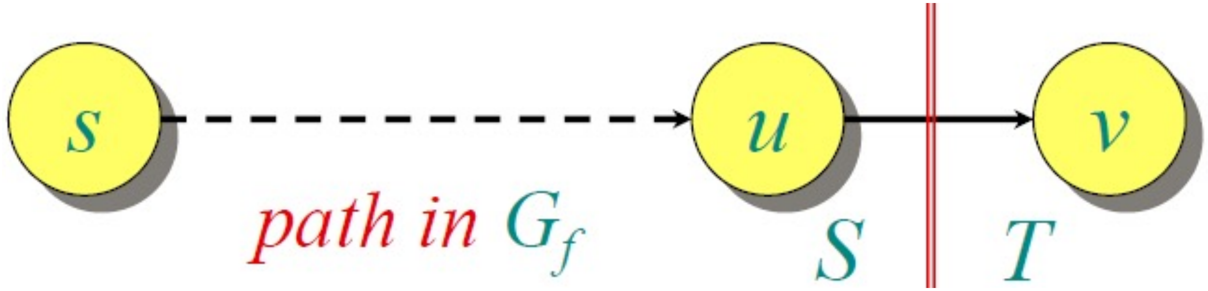
$S = \{v \in V :$ there exists a path in G_f from s to $v\}$

$T = V - S$

(S,T) is a cut! Why?

Max-Flow, Min-Cut Theorem

Proof. (3) $\Rightarrow$ (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;

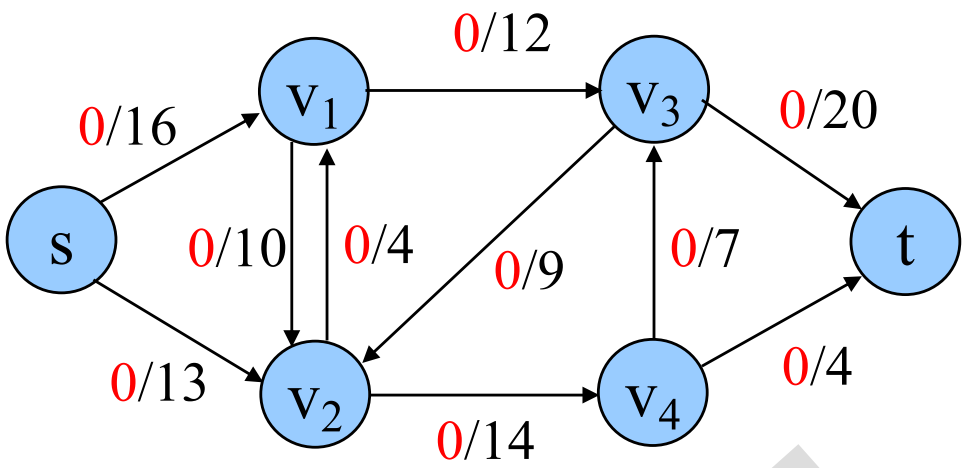
recursive call to FindFlow;

Algorithm

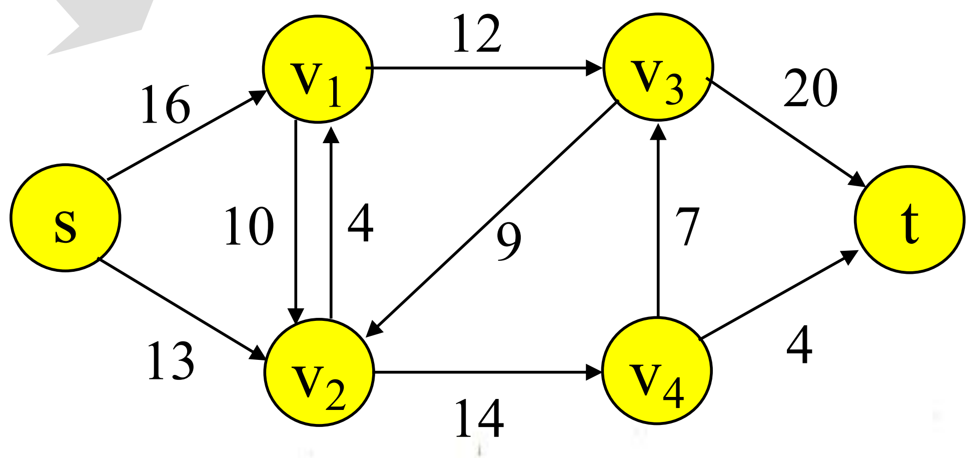
$f[u, v] \leftarrow 0$ for all $u, v \in V$

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

Example—Basic Implementation



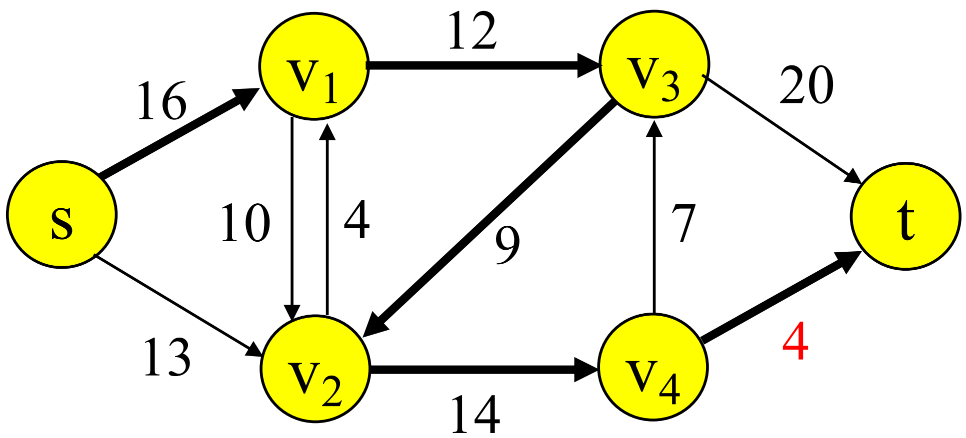
Flow initialization

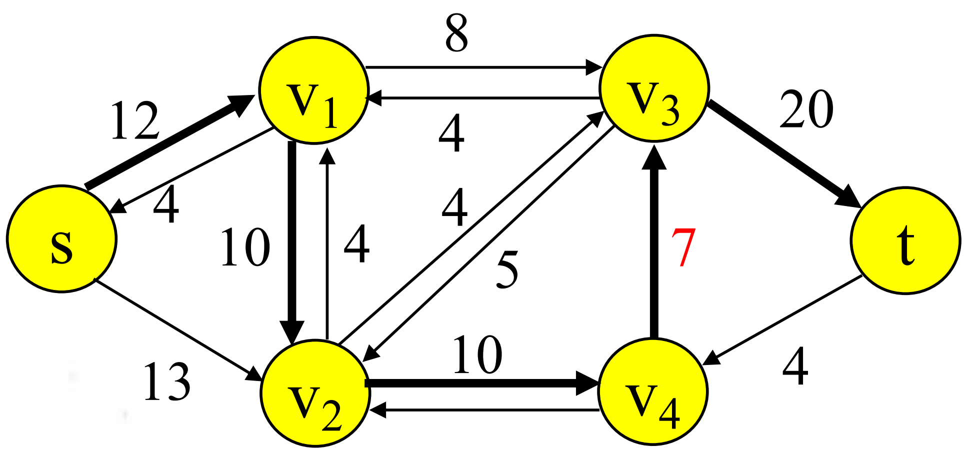


Residual network

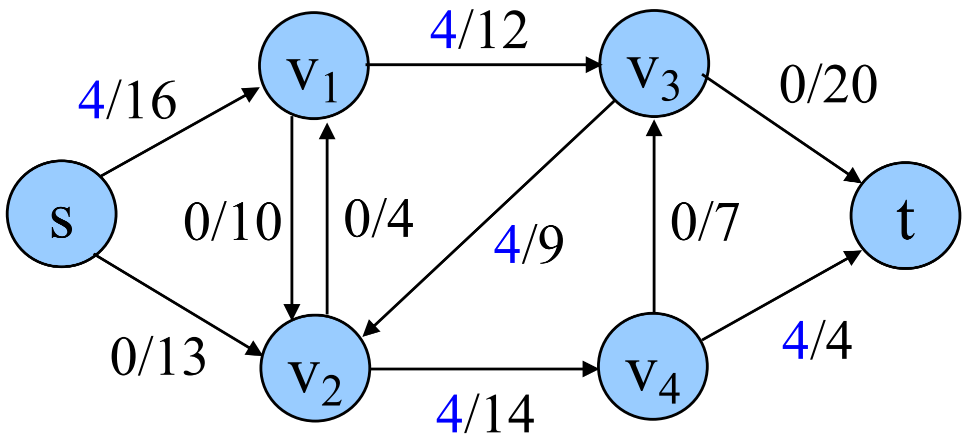
Example

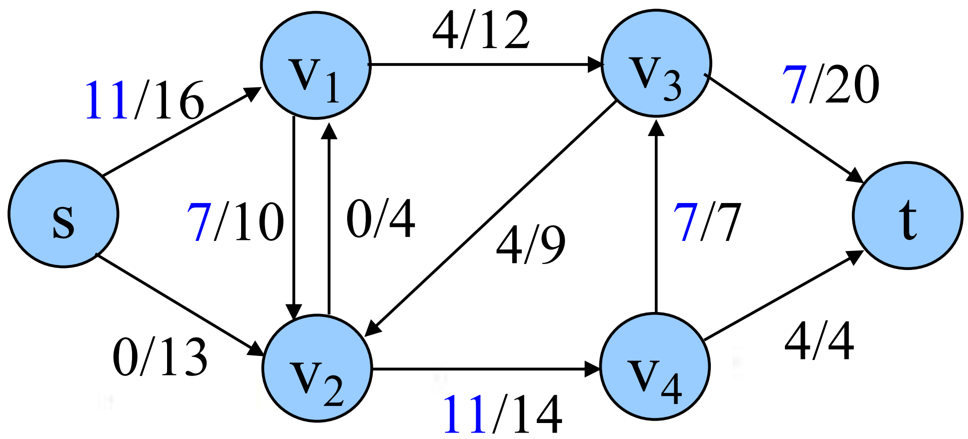
Residual Networks





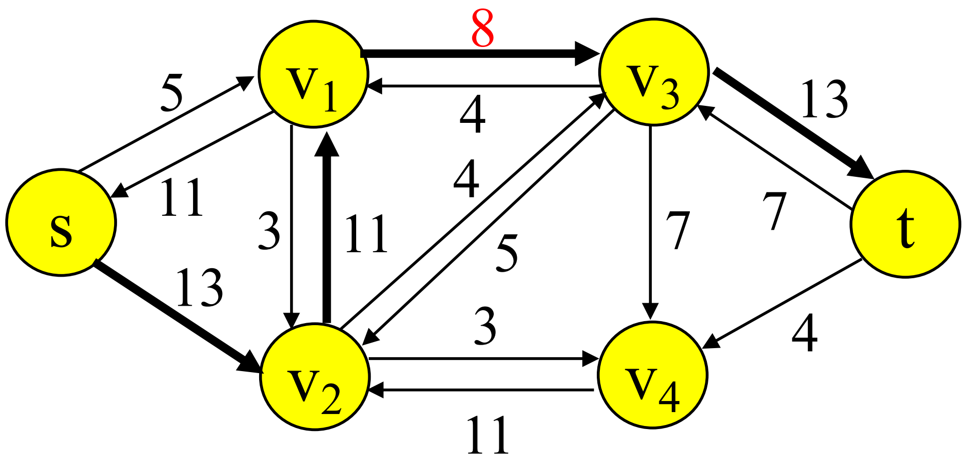
Flows

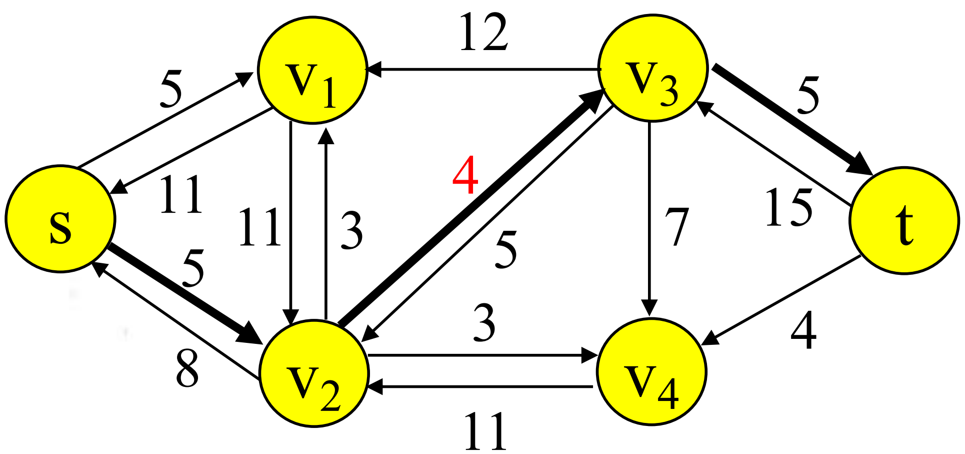




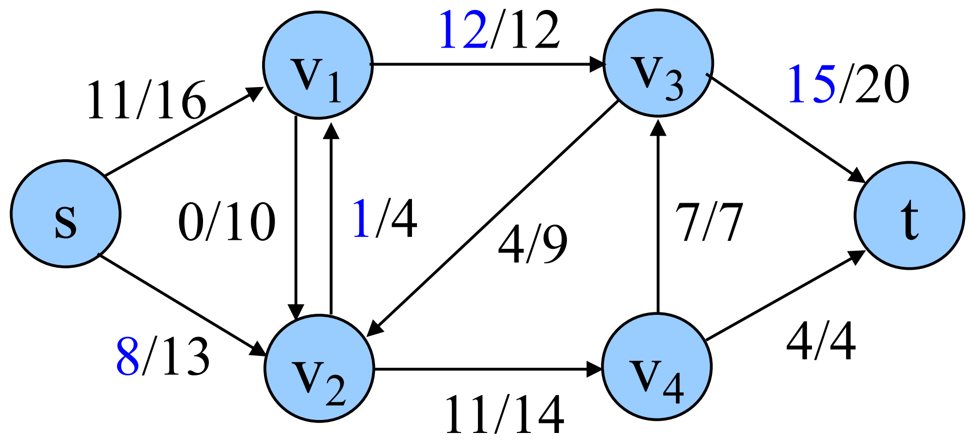
Example

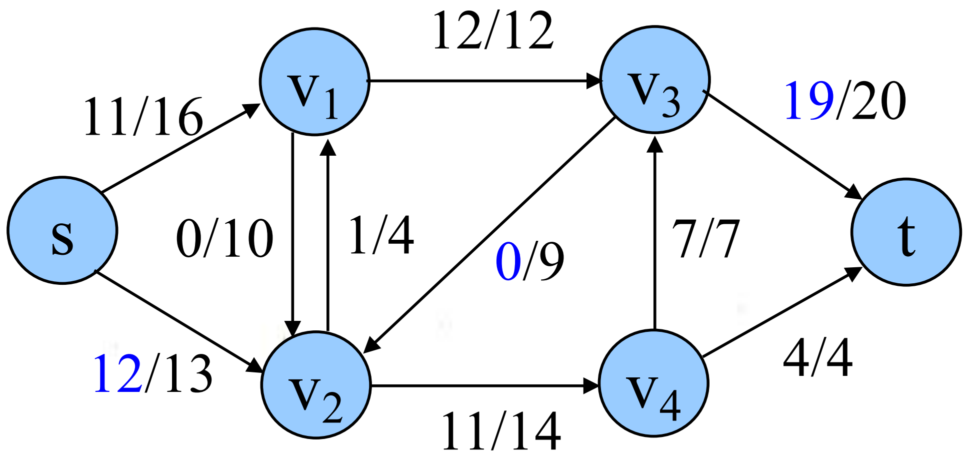
Residual Networks





Flows

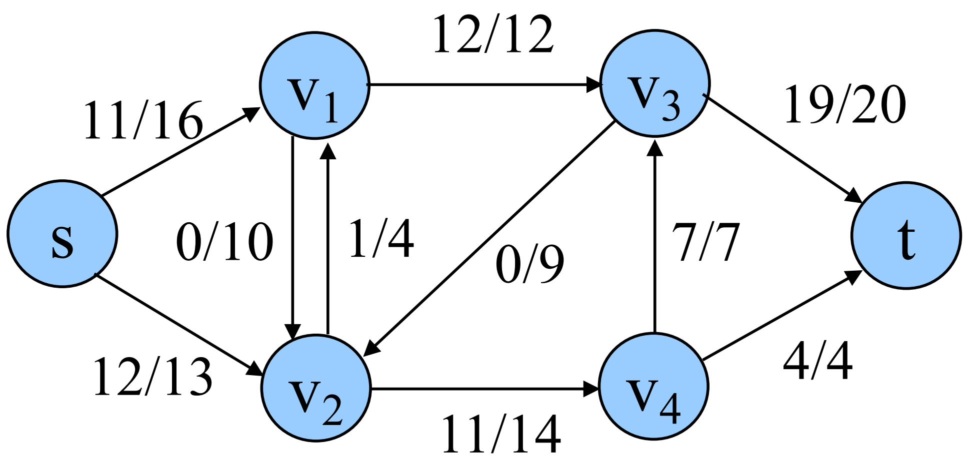




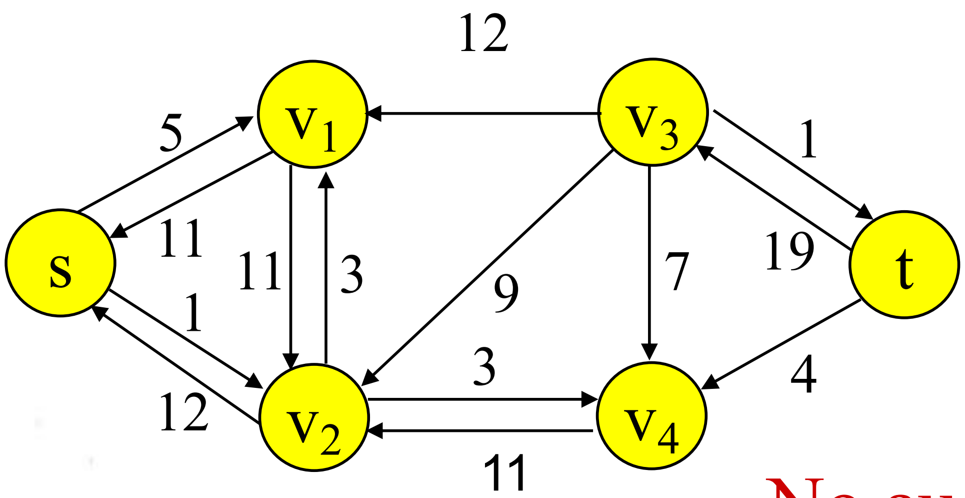
Example

Residual Networks

Flows



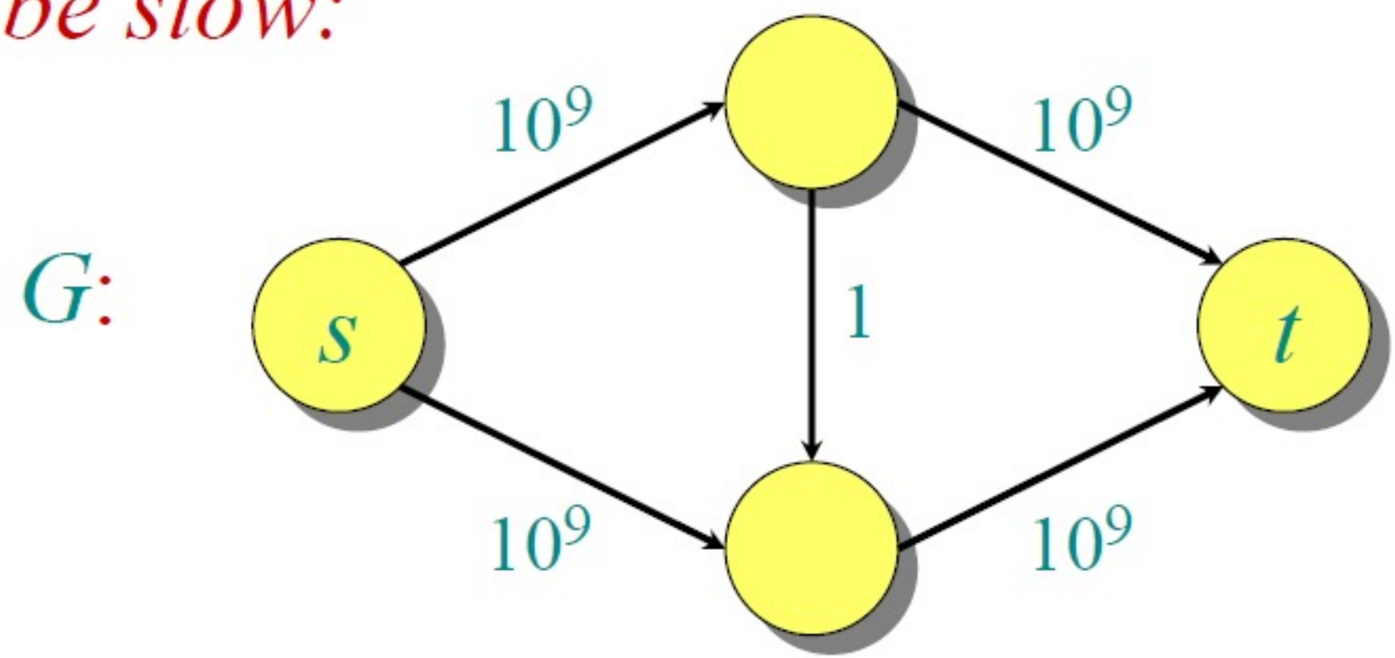
Maximum flow |*f*| = 11+12 = 23



No augmenting path

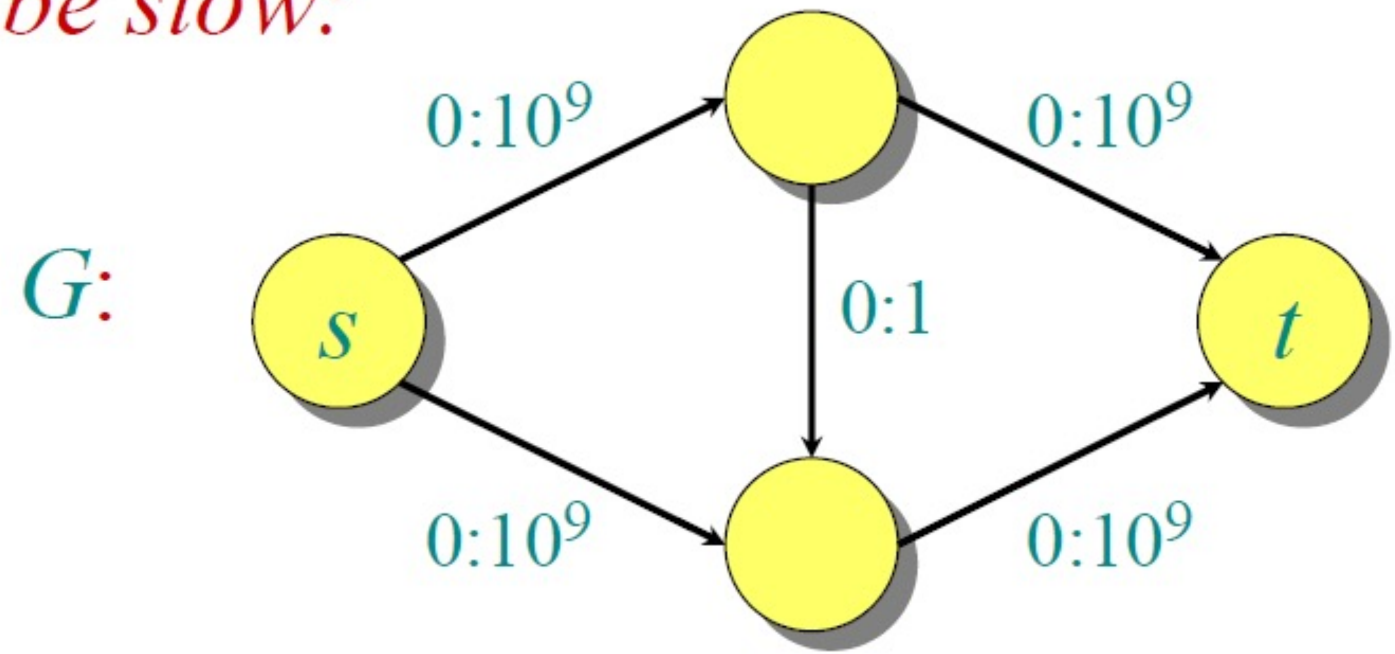
Problem： Time Complexity

Can be slow:



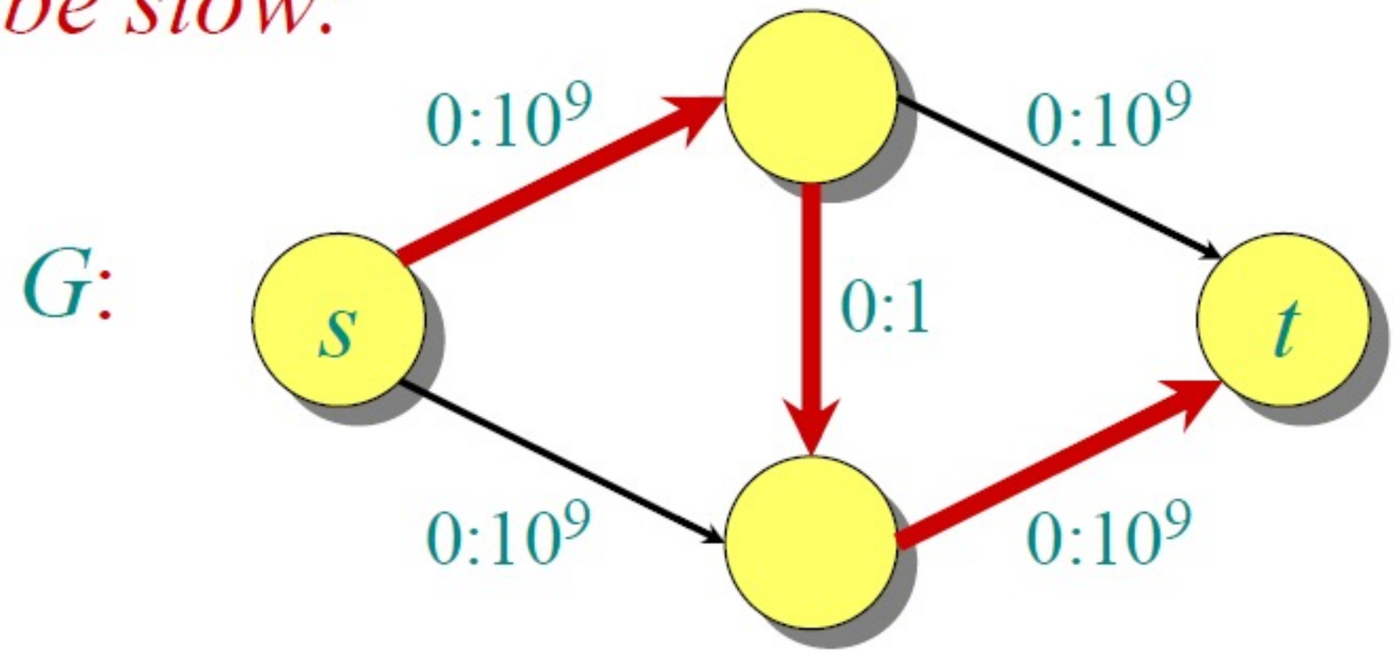
Problem： Time Complexity

Can be slow:



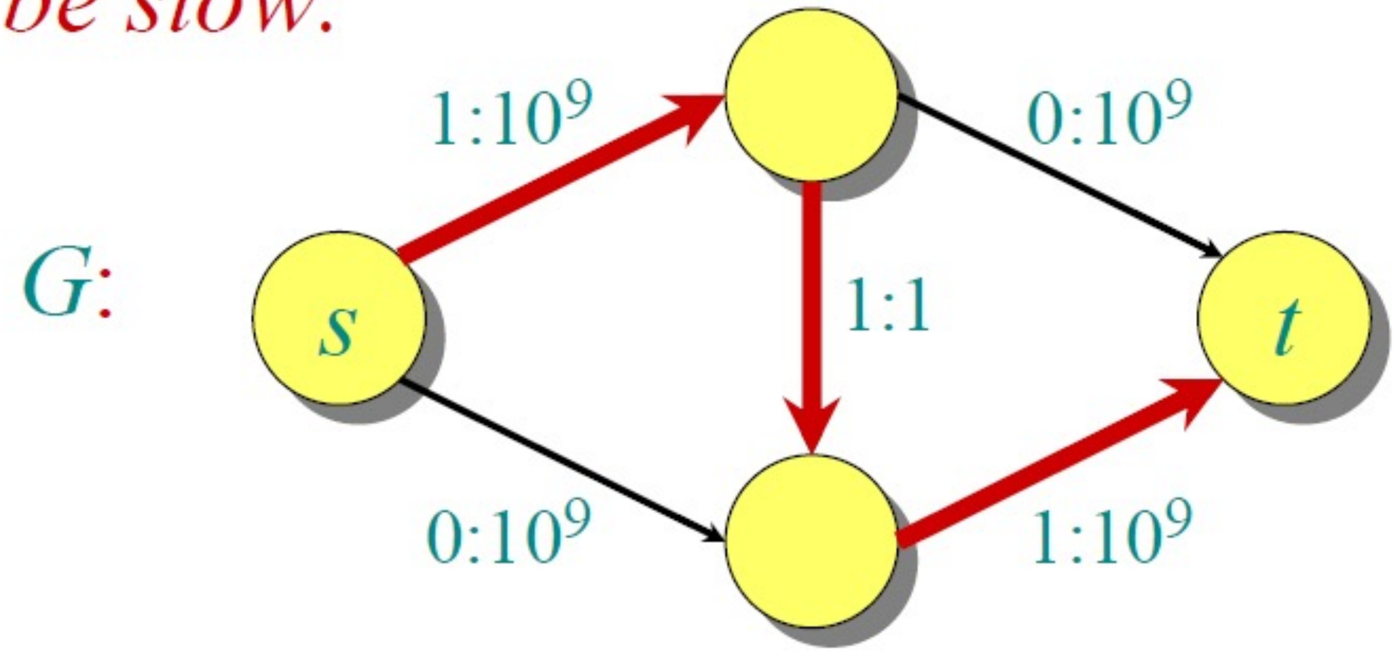
Problem： Time Complexity

Can be slow:



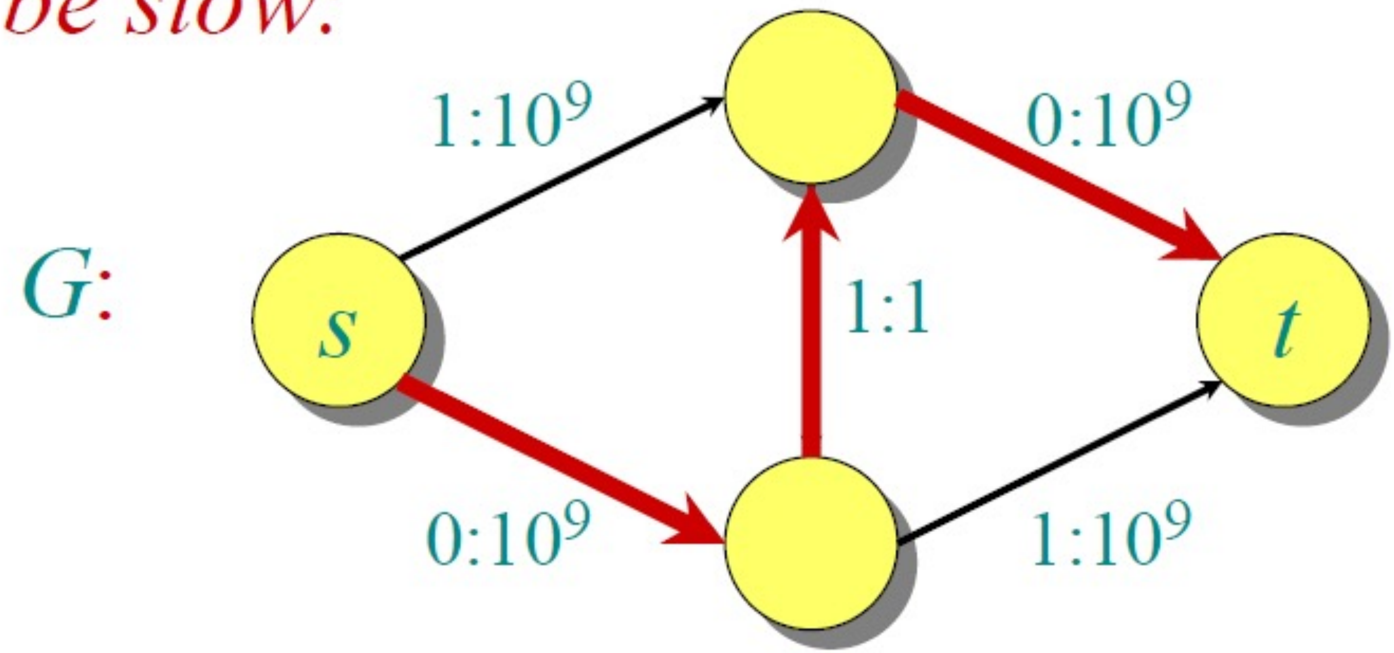
Problem： Time Complexity

Can be slow:

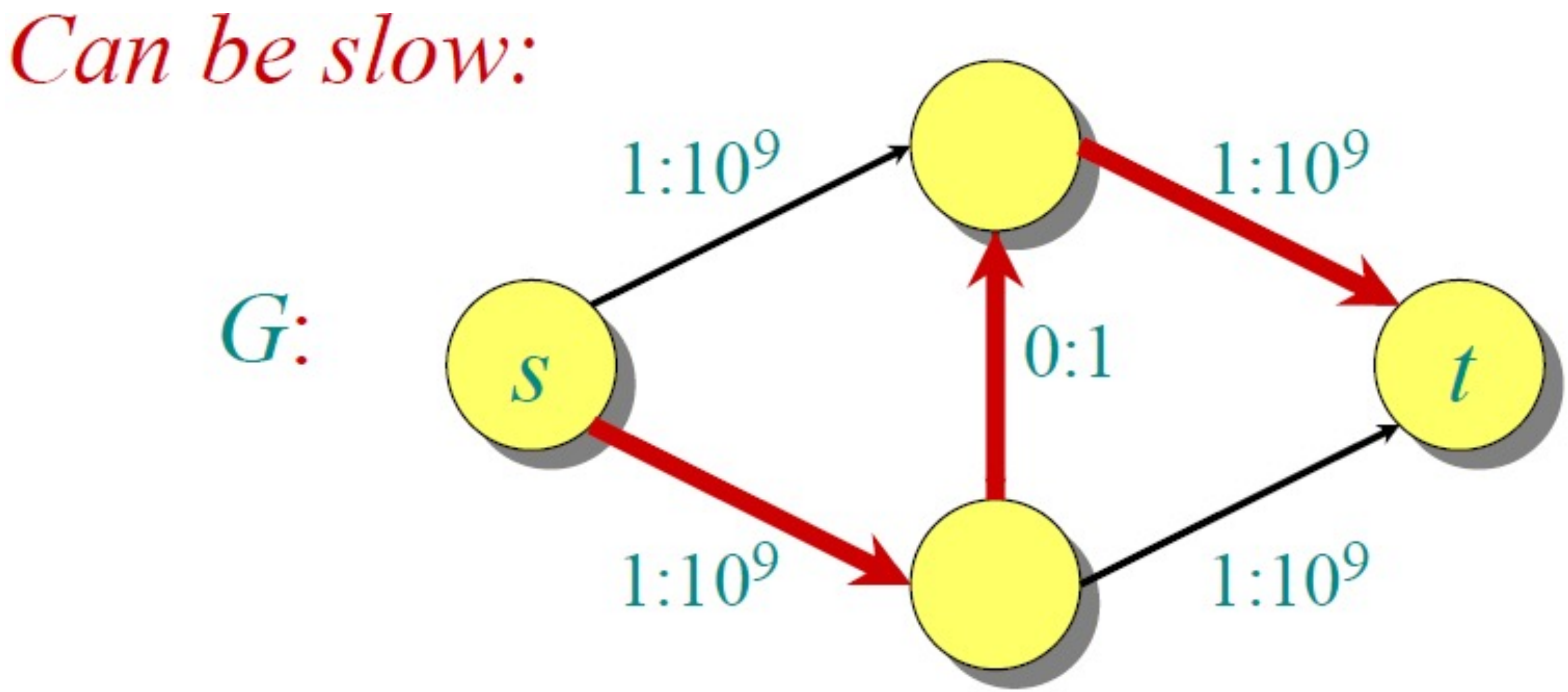


Problem： Time Complexity

Can be slow:

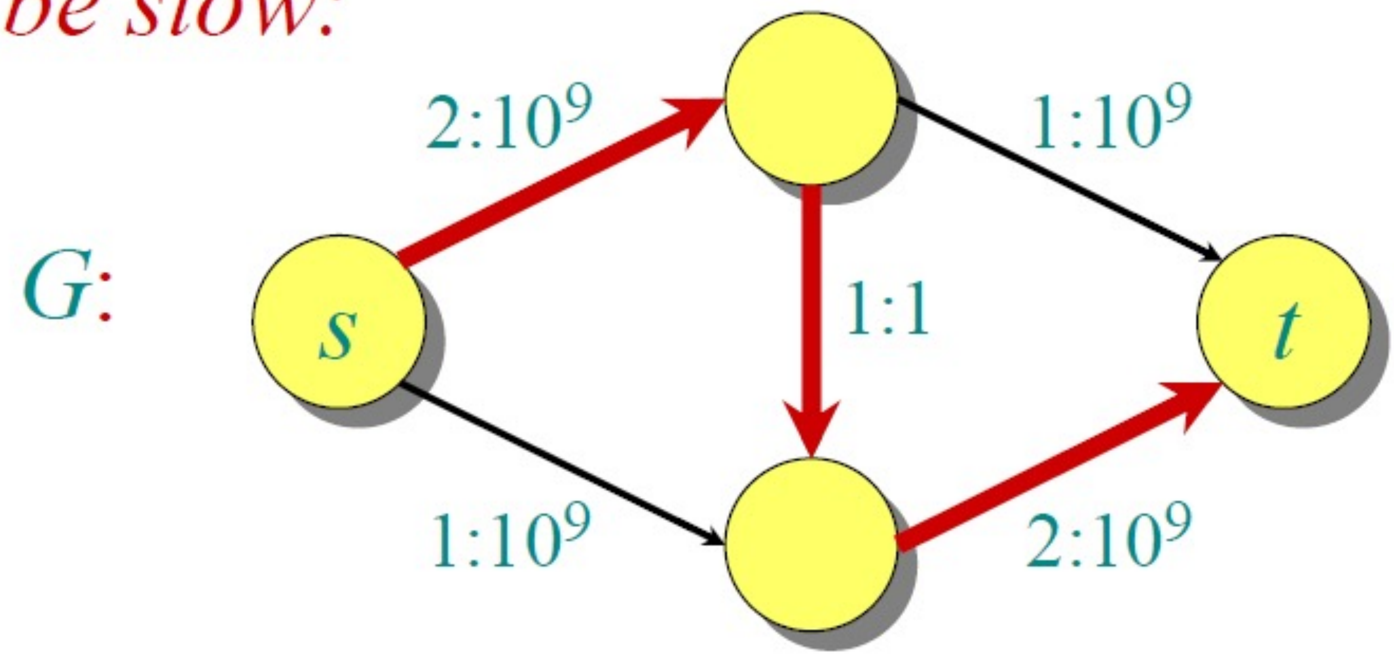


Problem： Time Complexity



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.

$$\text{if } F = 2^n$$

Then, along came Edmonds & Karp...

Edmonds & Karp Algorithm

Breadth-First Search

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

- **Output:** for all $v \in V$

  - $d[v]$ = 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 ($v_1, v_2, \ldots, v_k$) 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.

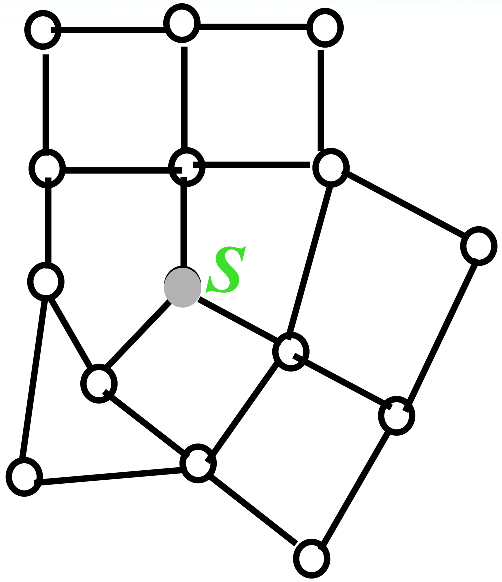
Principle of Breadth-First Search

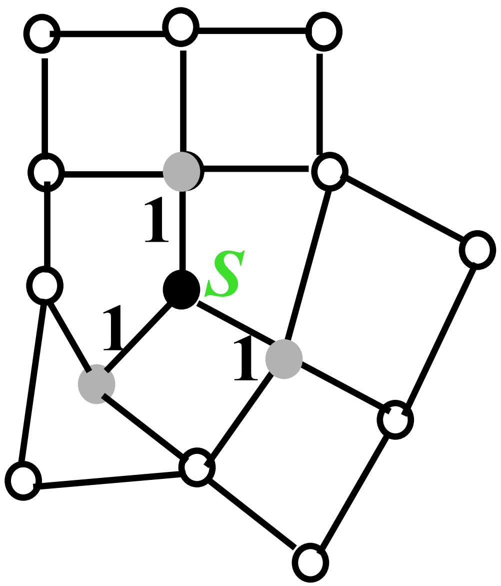
- **Expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier.**
  - **A vertex is "discovered" the first time it is encountered during the search.**
  - **A vertex is "finished" if all vertices adjacent to it have been discovered.**

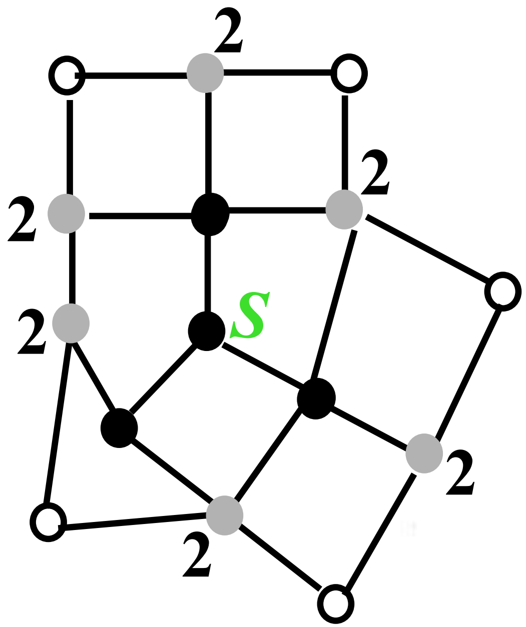
BFS for Shortest Paths

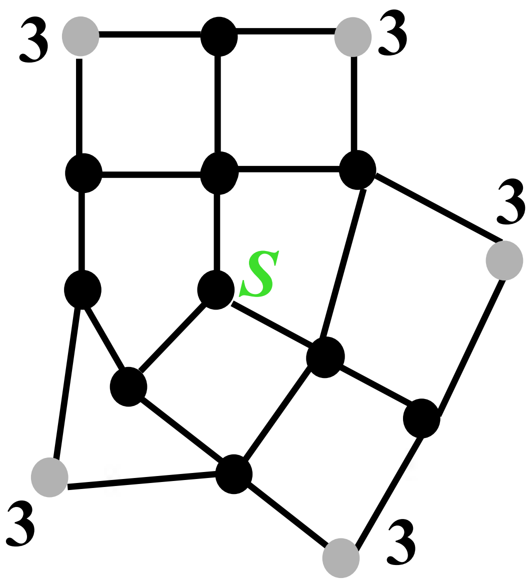
Colors the vertices to keep track of progress.

- **Undiscovered**
- **Discovered**
- **Finished**









BFS(G,s)

```
for each vertex u in V[G] – {s}
        do color[u] ← white
           d[u] ← ∝
           π[u] ← nil
color[s] ← gray
d[s] ← 0
π[s] ← nil
Q ← Φ
enqueue(Q,s)
while Q ≠ Φ
        do u ← dequeue(Q)
                for each v in Adj[u]
                        do if color[v] = white
                                then color[v] ← gray
                                     d[v] ← d[u] + 1
                                     π[v] ← u
                                     enqueue(Q,v)
                color[u] ← black
```

white: undiscovered
gray: discovered
black: finished

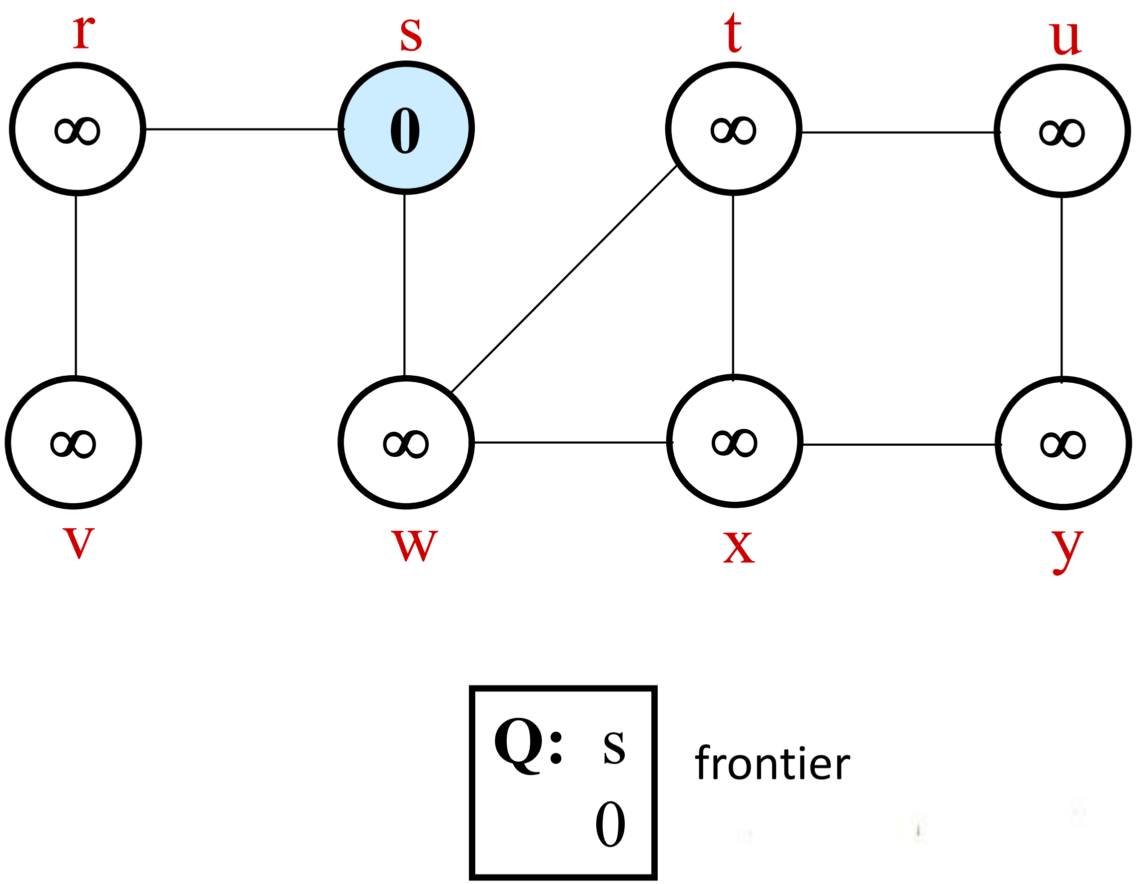
Q: a queue of discovered vertices

color[v]: color of v

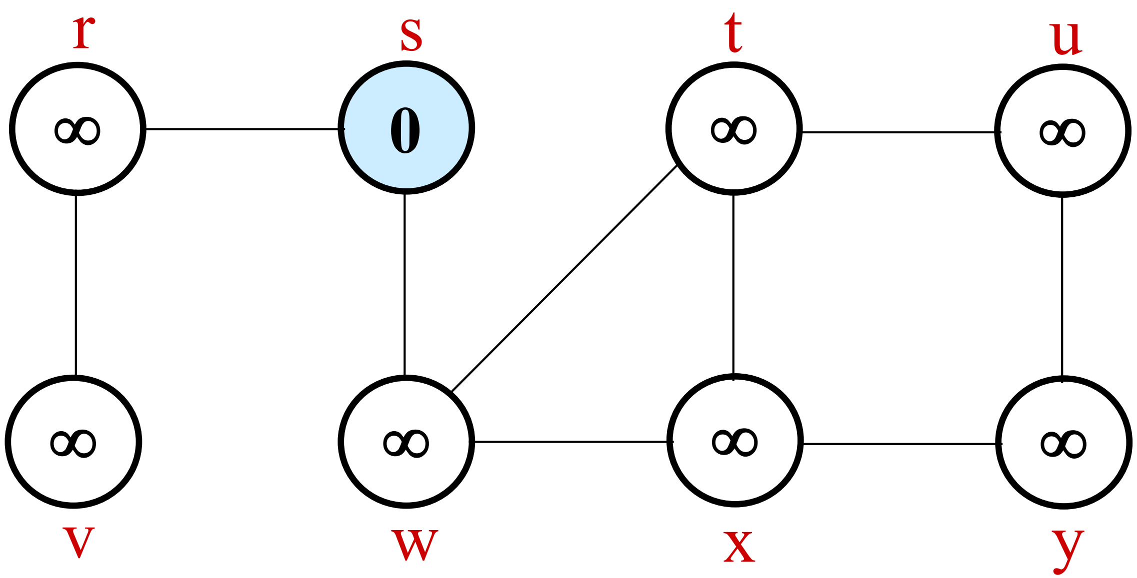
d[v]: distance from s to v

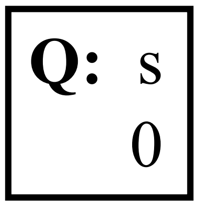
π[u]: predecessor of v

Example (BFS)



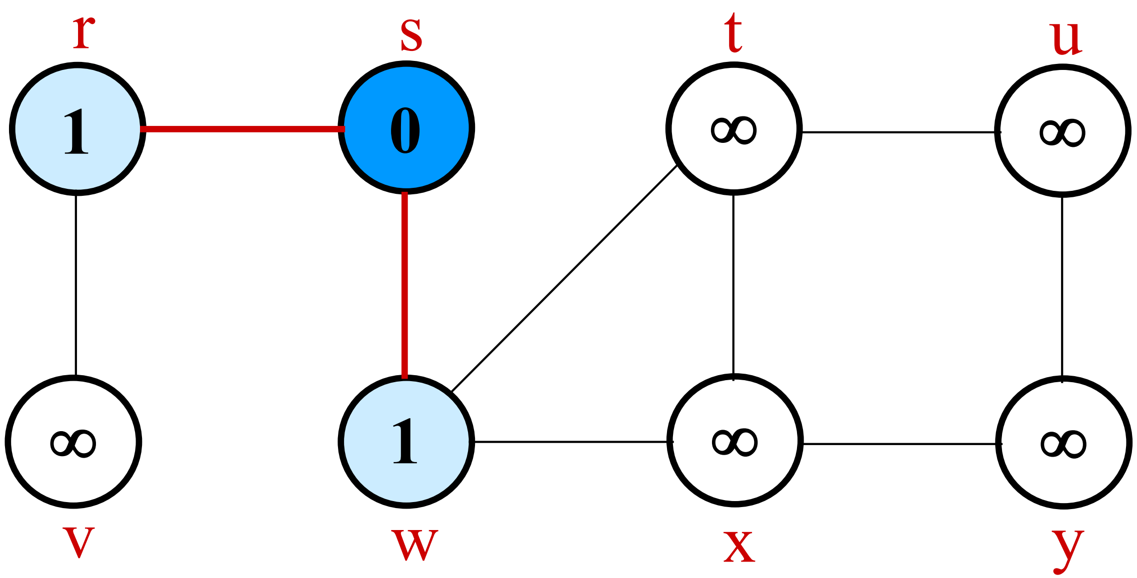
Example (BFS)





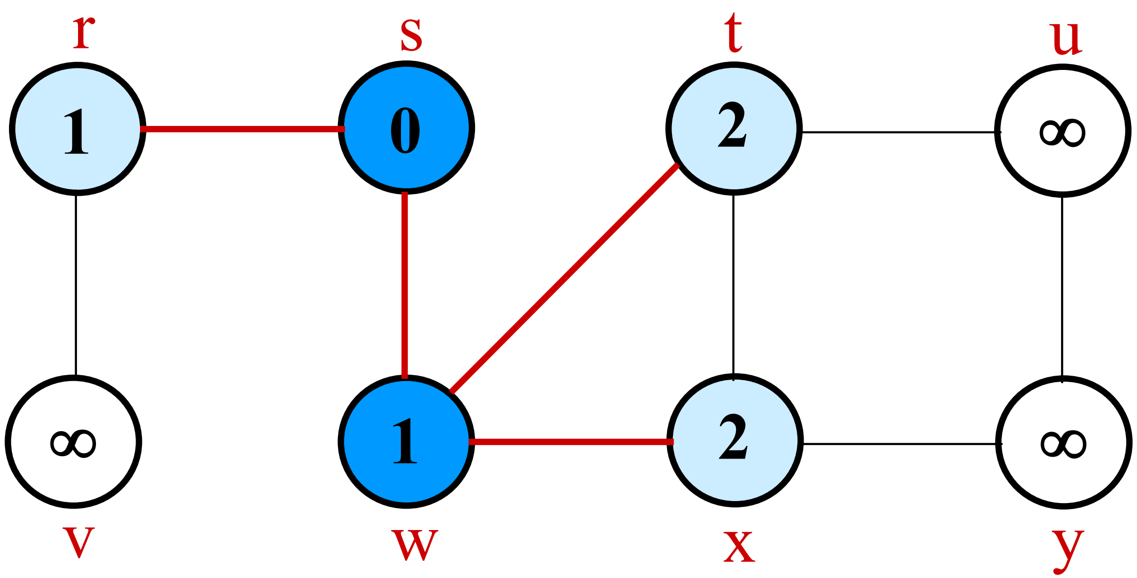
frontier

Example (BFS)



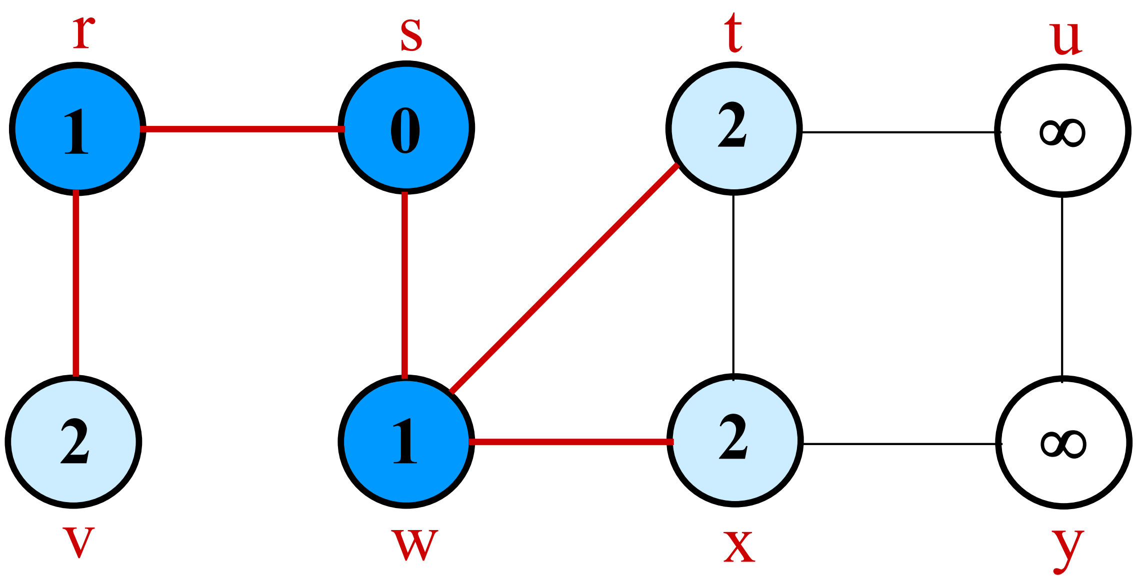
Q:	w	r
	1	1

Example (BFS)



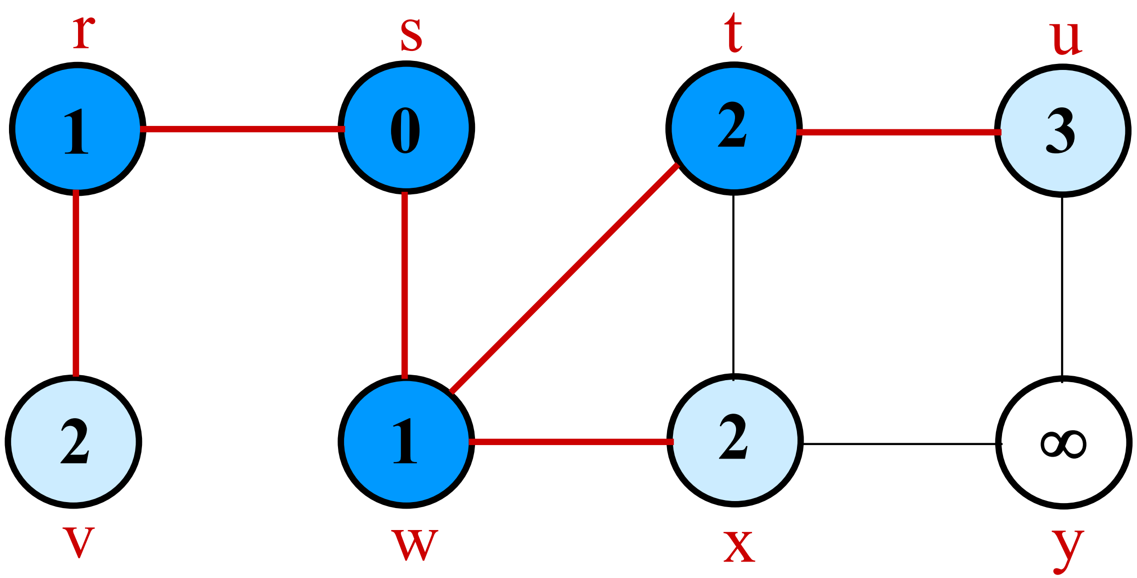
Q:	r	t	x
	1	2	2

Example (BFS)



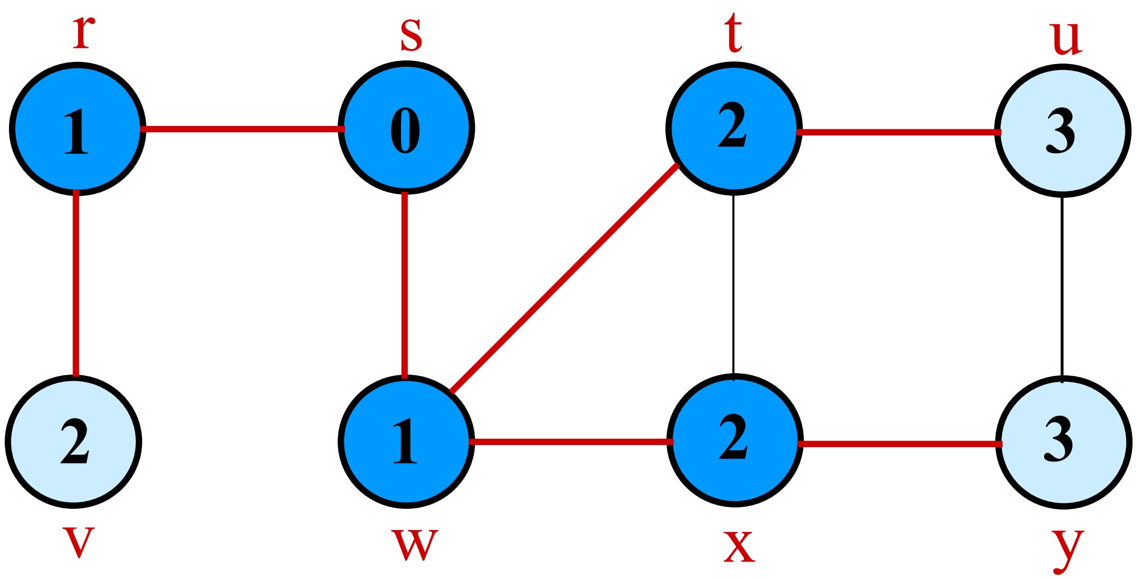
Q:	t	x	v
	2	2	2

Example (BFS)



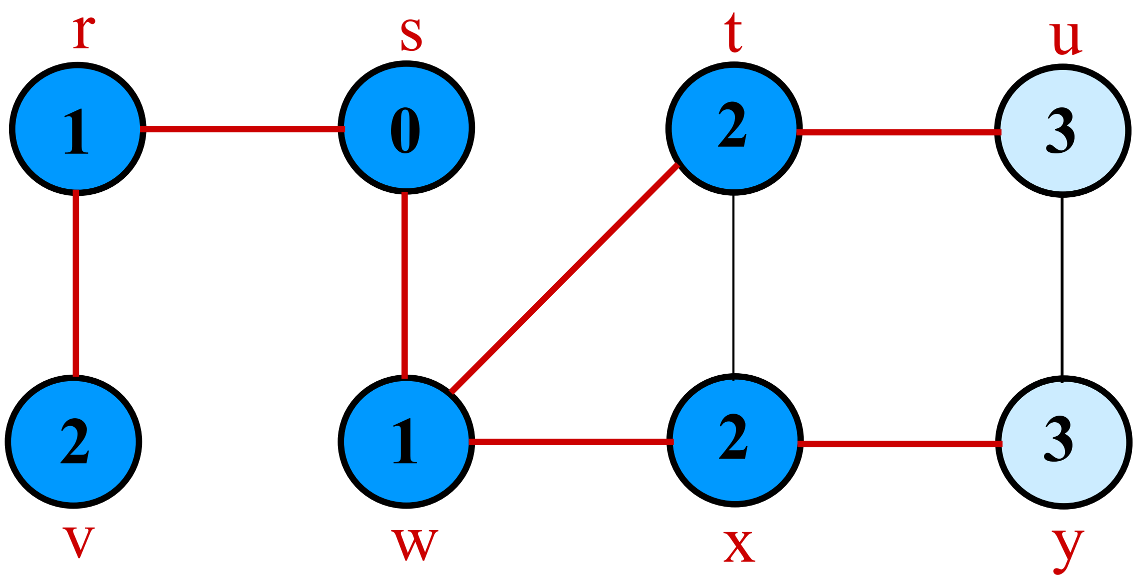
Q:	x	v	u
	2	2	3

Example (BFS)



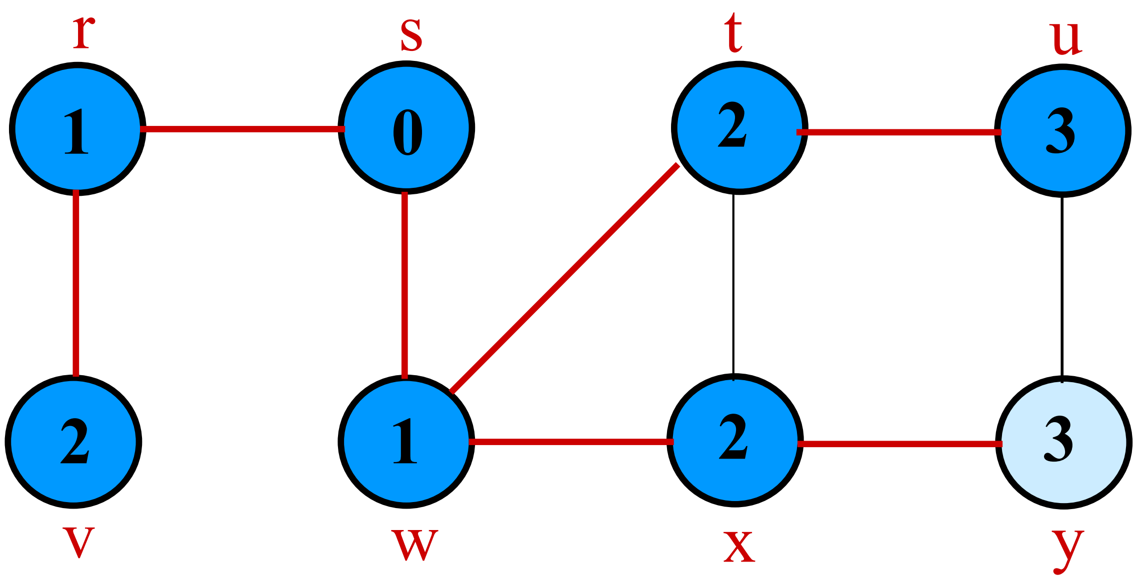
Q: v u y
2 3 3

Example (BFS)



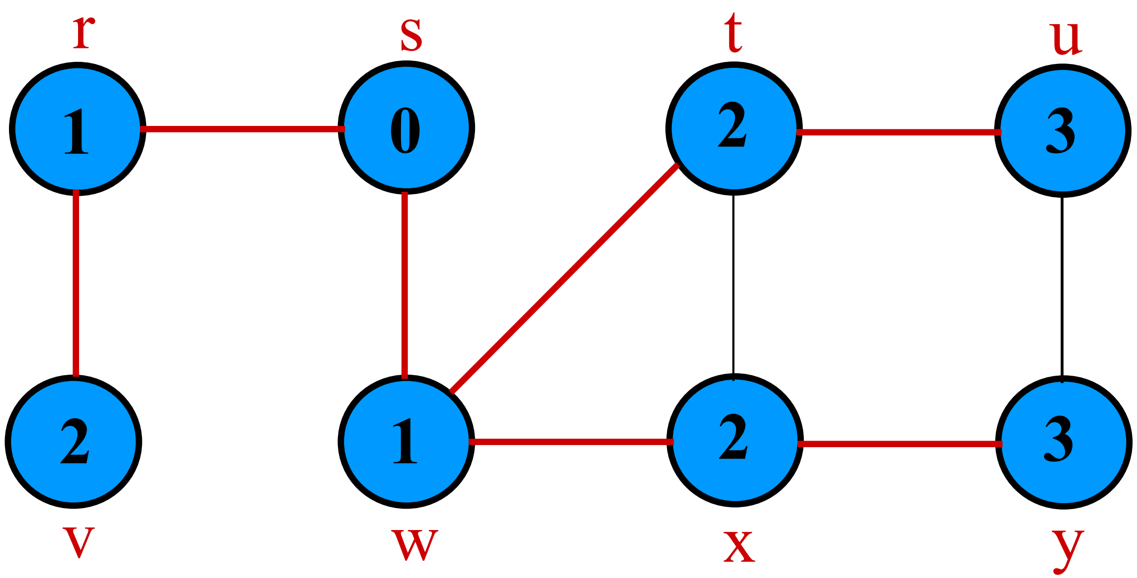
Q:	u y
	3 3

Example (BFS)



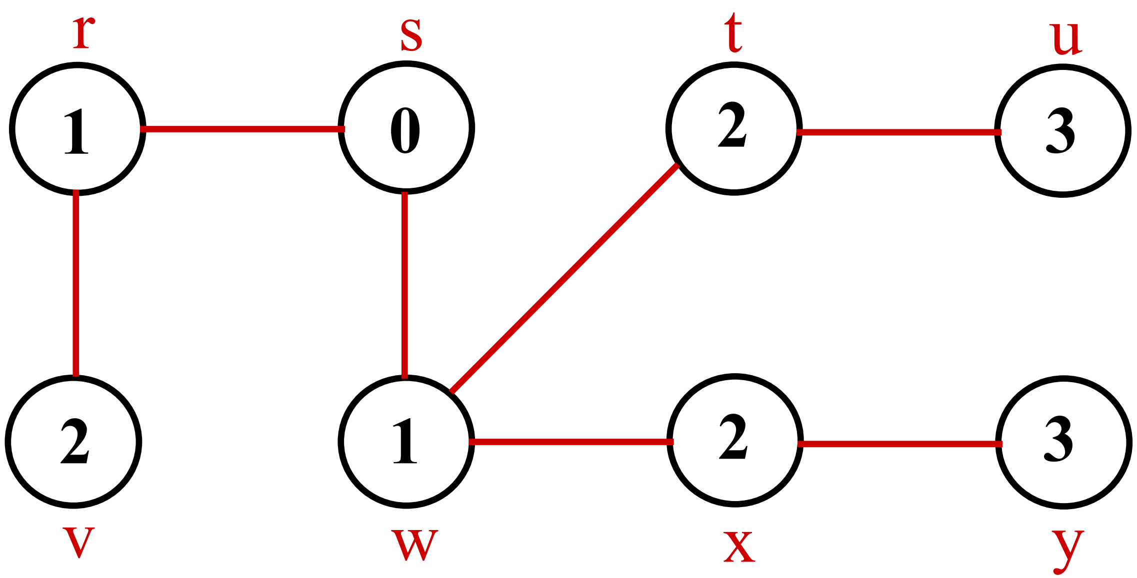
Q: y
3

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

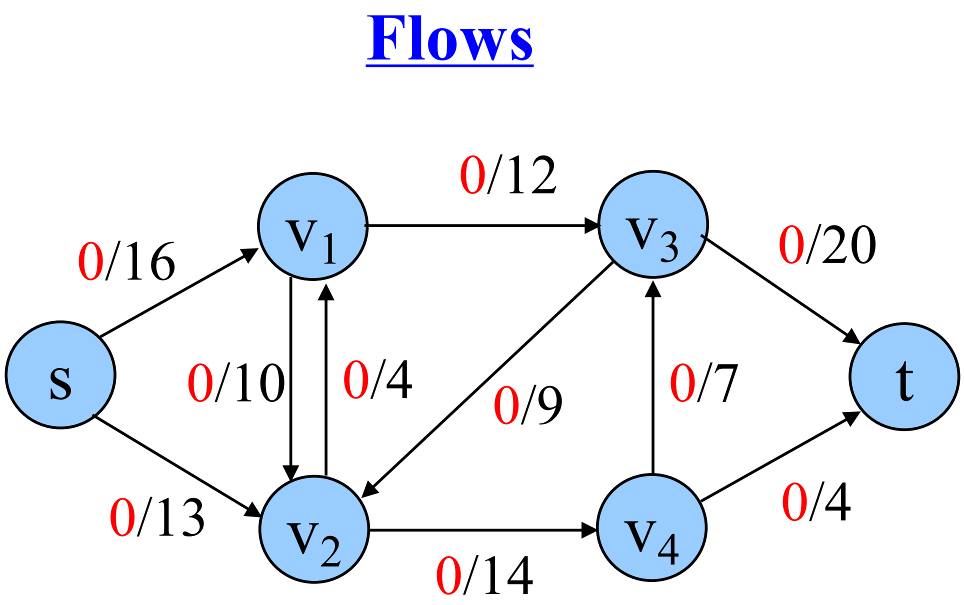
- Find the augmenting path using **breadth-first search**.

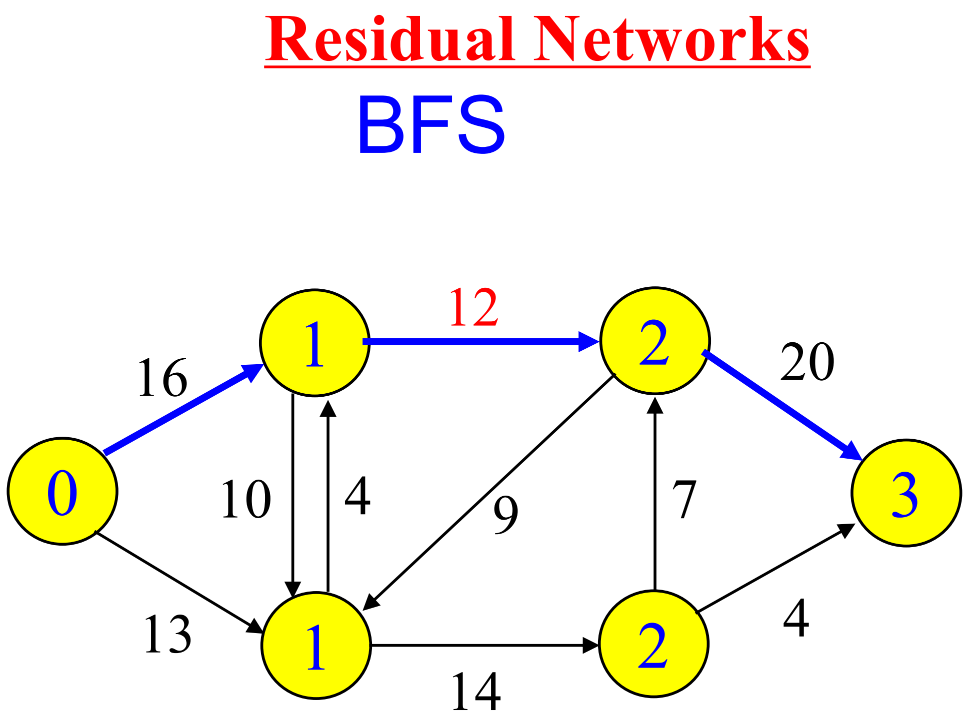
- Breadth-first search gives the shortest path for graphs **(Assuming the length of each edge is 1.)**

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

- The proof is very hard!.

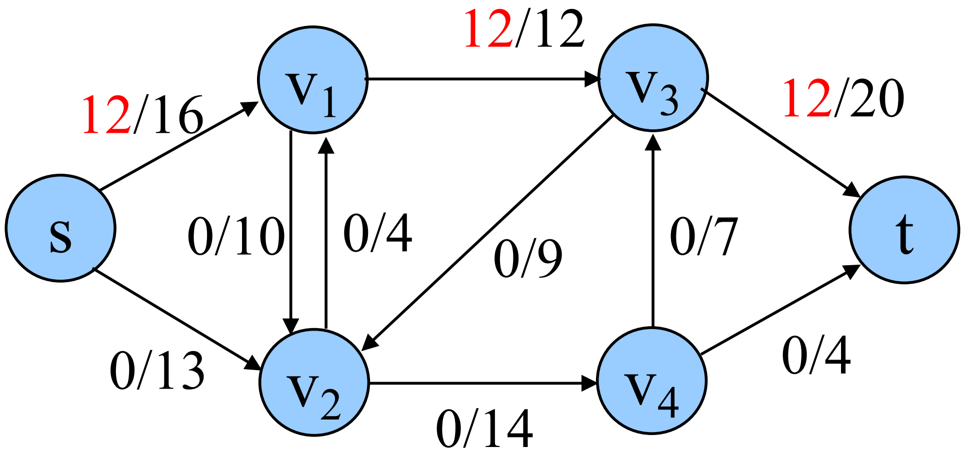
Example


Flows
s
v1
v2
v3
v4
t
0/16
0/13
0/10
0/4
0/12
0/9
0/14
0/7
0/20
0/4


Residual Networks
BFS
0
1
1
2
2
3
16
13
10
4
12
9
14
7
20
4

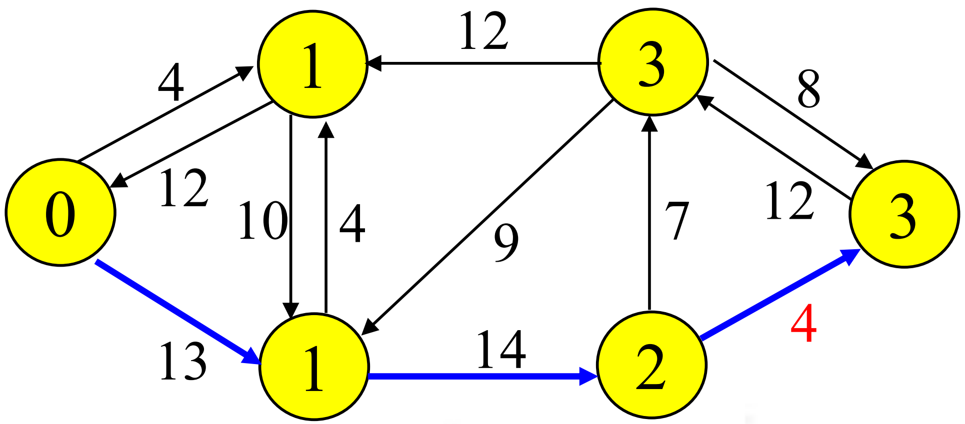
Example

Flows



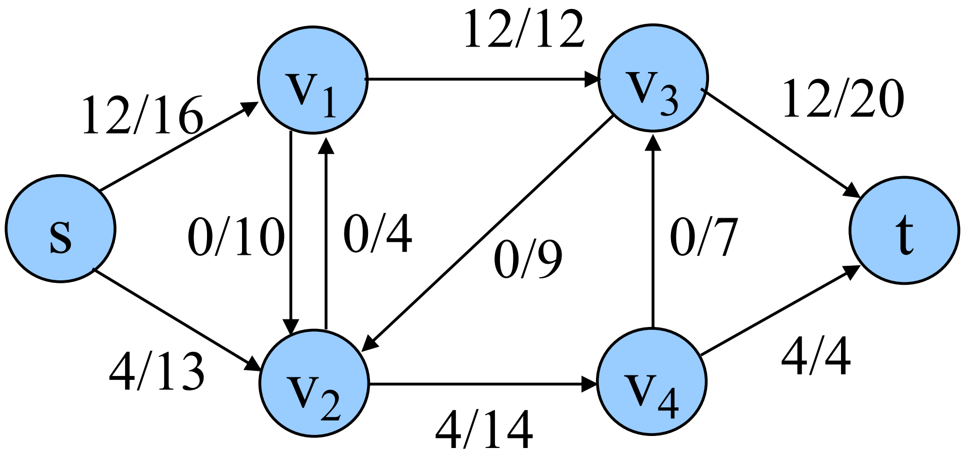
Residual Networks

BFS



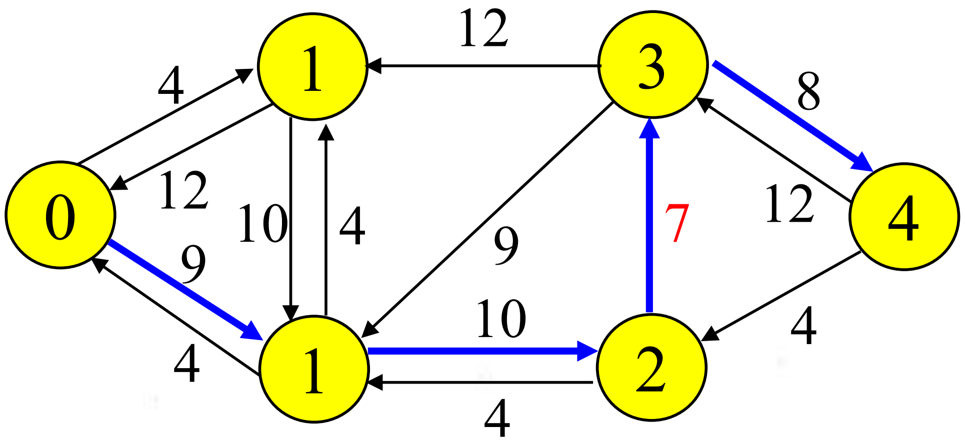
Example

Flows



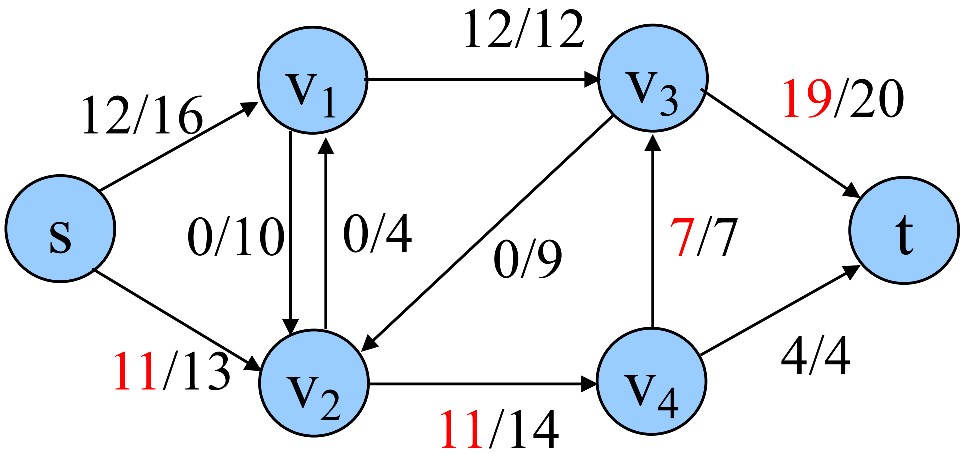
Residual Networks

BFS



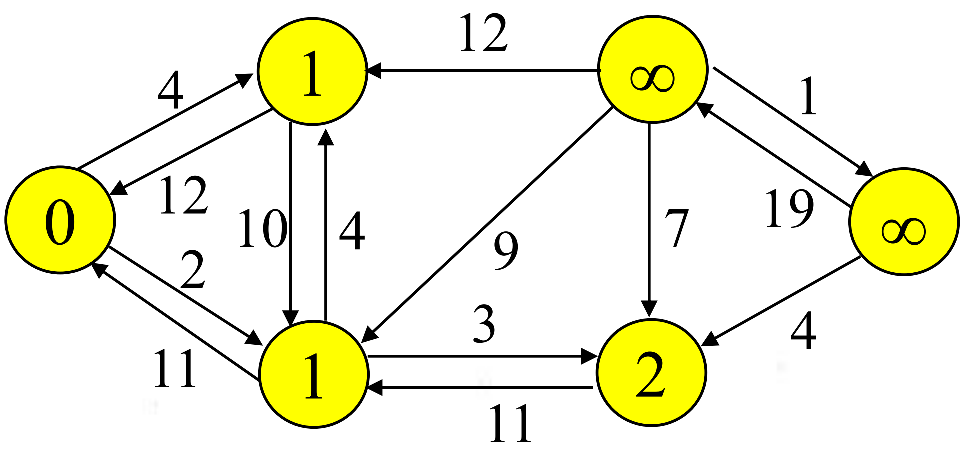
Example

Flows



Residual Networks

BFS



Maximum!

No path to sink

Edmonds & Karp Algorithm

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

命题1： 随着流量的递增，剩余流量图中任何顶点v的d[v]值（BFS)不会减少，只会增加或保持不变。

命题2：每次增加流量，增广路径中至少有一条边会从剩余流量图中删除（只剩反方向的边）。这条边在后序的增流中可能会再次出现，然后再被删除。。。但任何边被反复删除的次数不超过结点数量的一半。

Edmonds & Karp Algorithm

命题1： 随着流量的递增，剩余流量图中任何顶点v的d[v]值（BFS)不会减少，只会增加或保持不变。

证明　　设$G_f^0, G_f^1, \ldots G_f^j, \ldots.$为增加$j(\geq 1)$次流量后的剩余流量图

假设有顶点在增流过程中d值减少，则存在k>0, G_f^k中第一次出现d值减少的顶点

那么一定有顶点$y \in V$, $\mathrm{d}^k[\mathrm{y}] < \mathrm{d}^{k-1}[\mathrm{y}]$ 且 $d^k[x] \geq d^{k-1}[x] \wedge x = prev^k[y]$

$$d^{k-1}[x] \leq d^k[x] = d^k[y] - 1 < d^{k-1}[y] - 1$$

$$d^{k-1}[x] < d^{k-1}[y] - 1 \text{ 且 } G_f^{k-1}: (x, y) \notin E_f^{k-1}$$

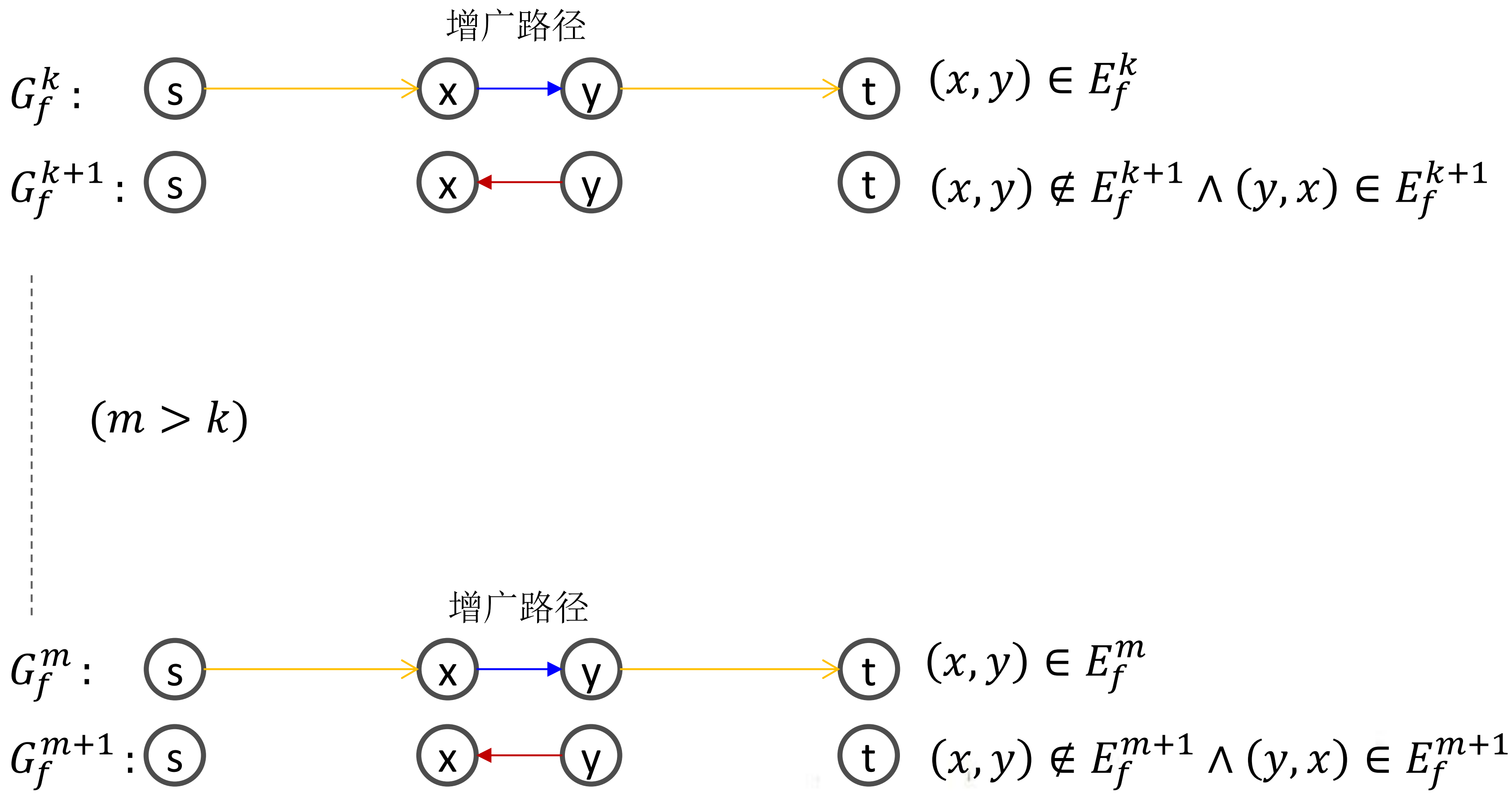
$$\because G_f^k: (x, y) \in E_f^k \quad \therefore G_f^{k-1}: (y, x) \in E_f^{k-1}\text{且在增广路径上}$$

$$d^{k-1}[x] = d^{k-1}[y] + 1 \text{ 与 假设的 } d^{k-1}[x] < d^{k-1}[y] - 1\text{矛盾}$$

命题成立

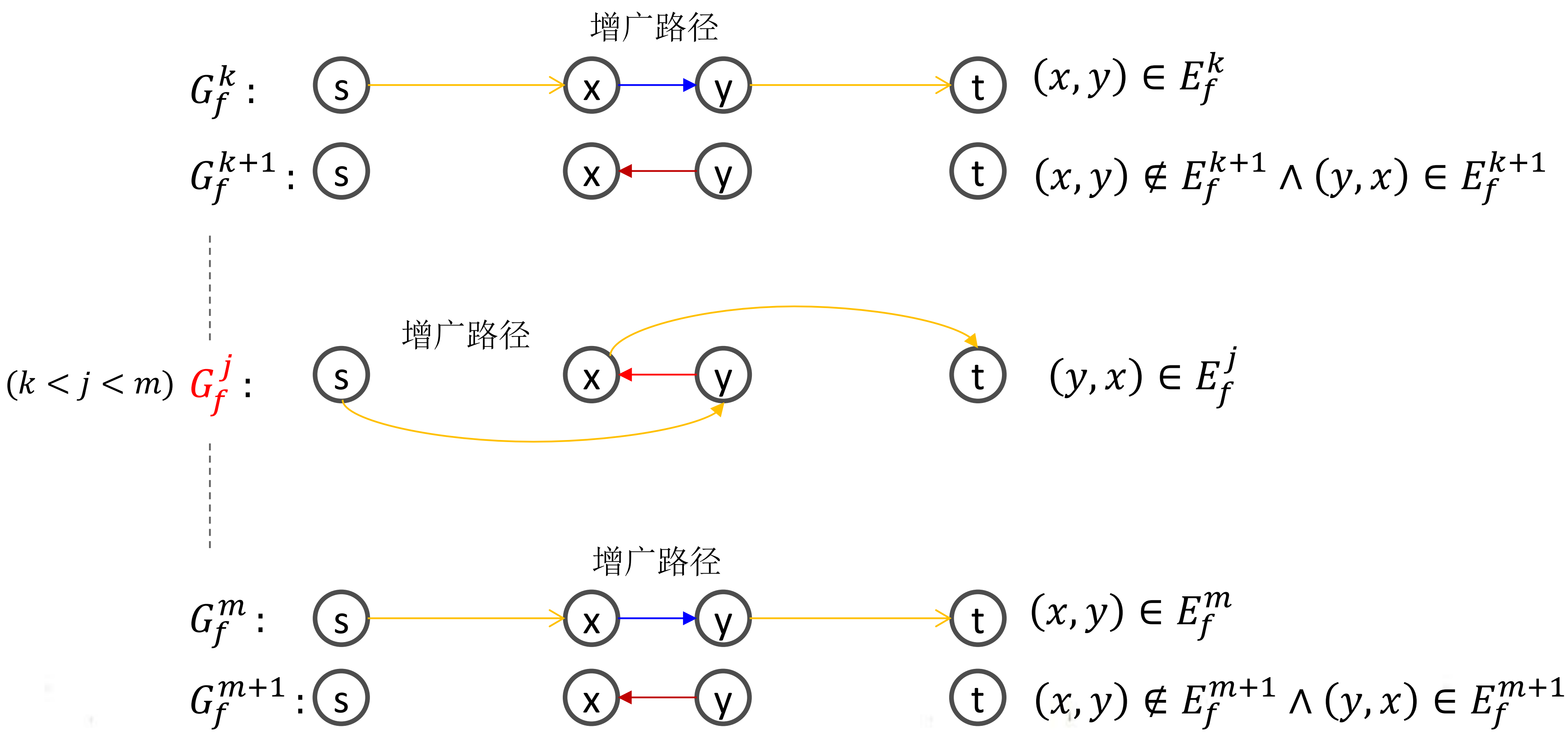
Edmonds & Karp Algorithm

命题2：每次增加流量，增广路径中至少有一条边会从剩余流量图中删除（只剩反方向的边）。这条边在后序的增流中可能会再次出现，然后再被删除。。。但任何边被反复删除的次数不超过结点数量的一半。



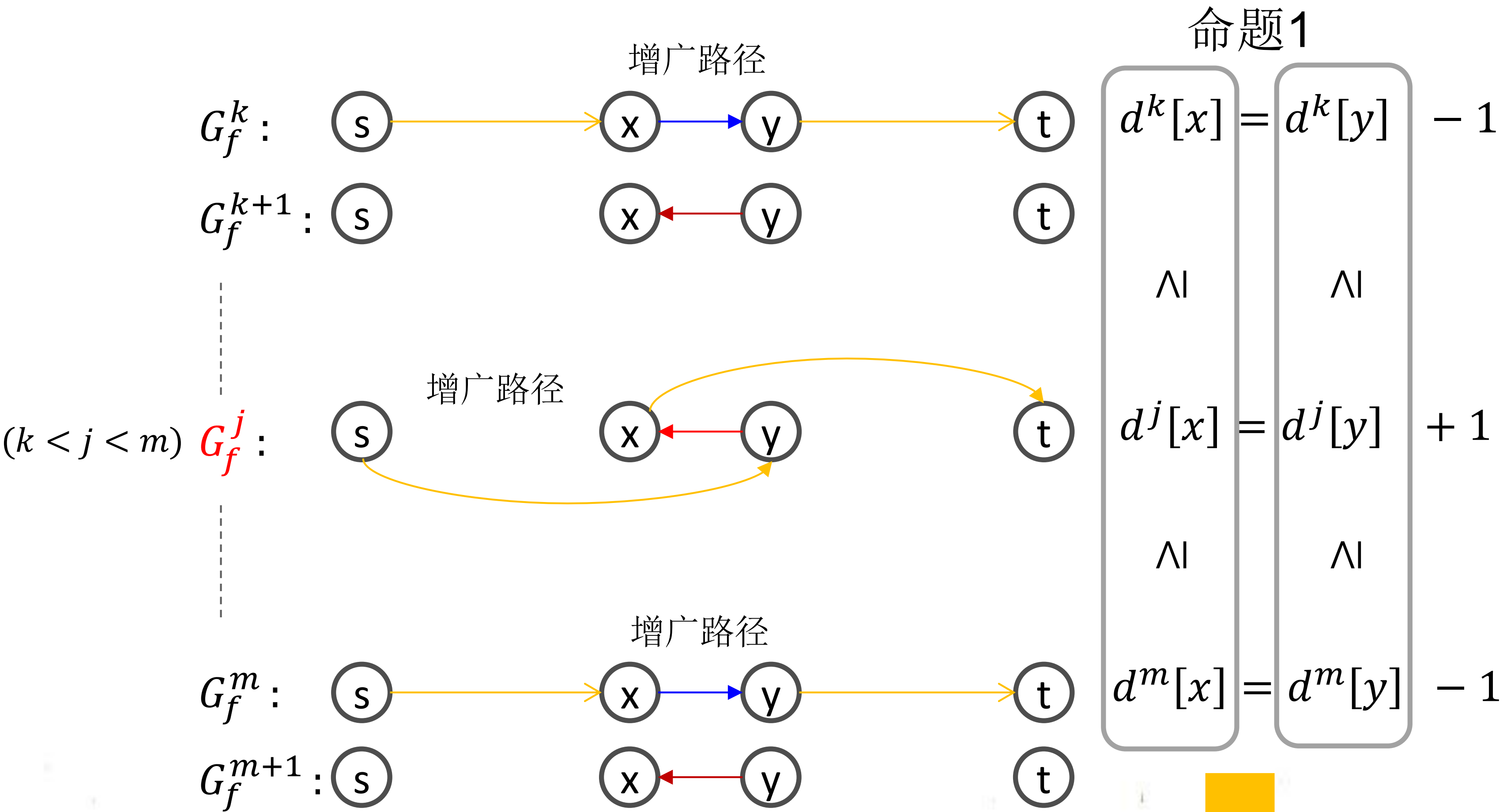
Edmonds & Karp Algorithm

命题2：每次增加流量，增广路径中至少有一条边会从剩余流量图中删除（只剩反方向的边）。这条边在后序的增流中可能会再次出现，然后再被删除。。。但任何边被反复删除的次数不超过结点数量的一半。



Edmonds & Karp Algorithm

命题2：每次增加流量，增广路径中至少有一条边会从剩余流量图中删除（只剩反方向的边）。这条边在后序的增流中可能会再次出现，然后再被删除。。。但任何边被反复删除的次数不超过结点数量的一半。



$$d^m[y] - d^k[y] \geq 2$$

Applications

比赛淘汰问题

Teams	Wins	Losses	To play	ATL	PHI	NY	MON
Atlanta	82	71	8	--	4	3	1
Philly	81	75	6	4	--	1	1
New York	78	78	6	3	1	--	2
Montreal	74	84	4	1	1	2	--

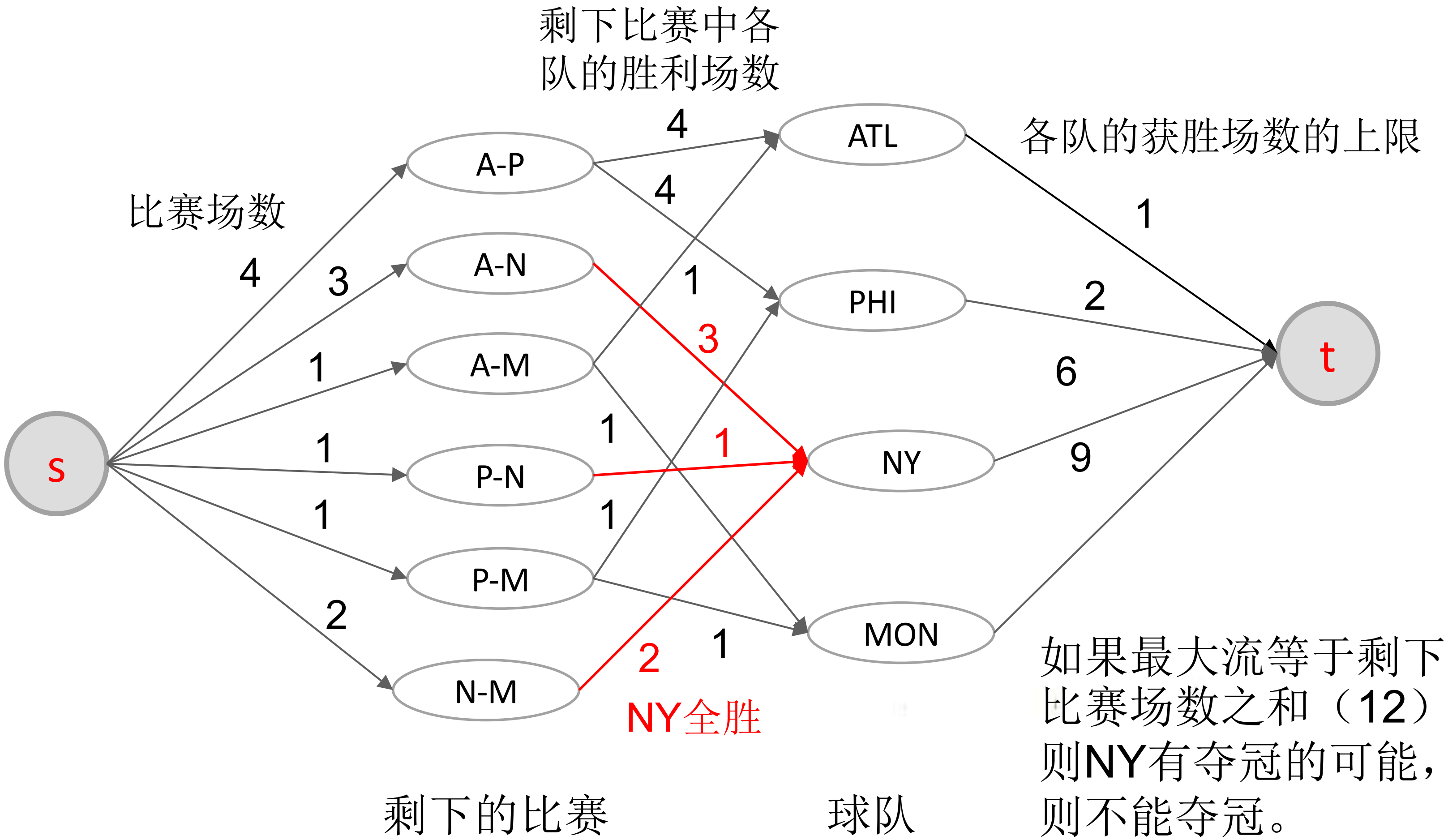
美国职业棒球的例行赛，每个球队都要打 162 场比赛，所胜场数最多者为该分区的冠军

根据目前各球队的得分情况和剩余的场次安排，判断New York队是否有夺冠的可能？

Applications

解题思路

New York队能够夺得冠军的最低条件，剩下的比赛都赢即赢6场胜利，且其他队伍最终赢的场数不超过84。



如果最大流等于剩下的比赛场数之和（12），则NY有夺冠的可能，否则不能夺冠。

Applications

最大匹配问题

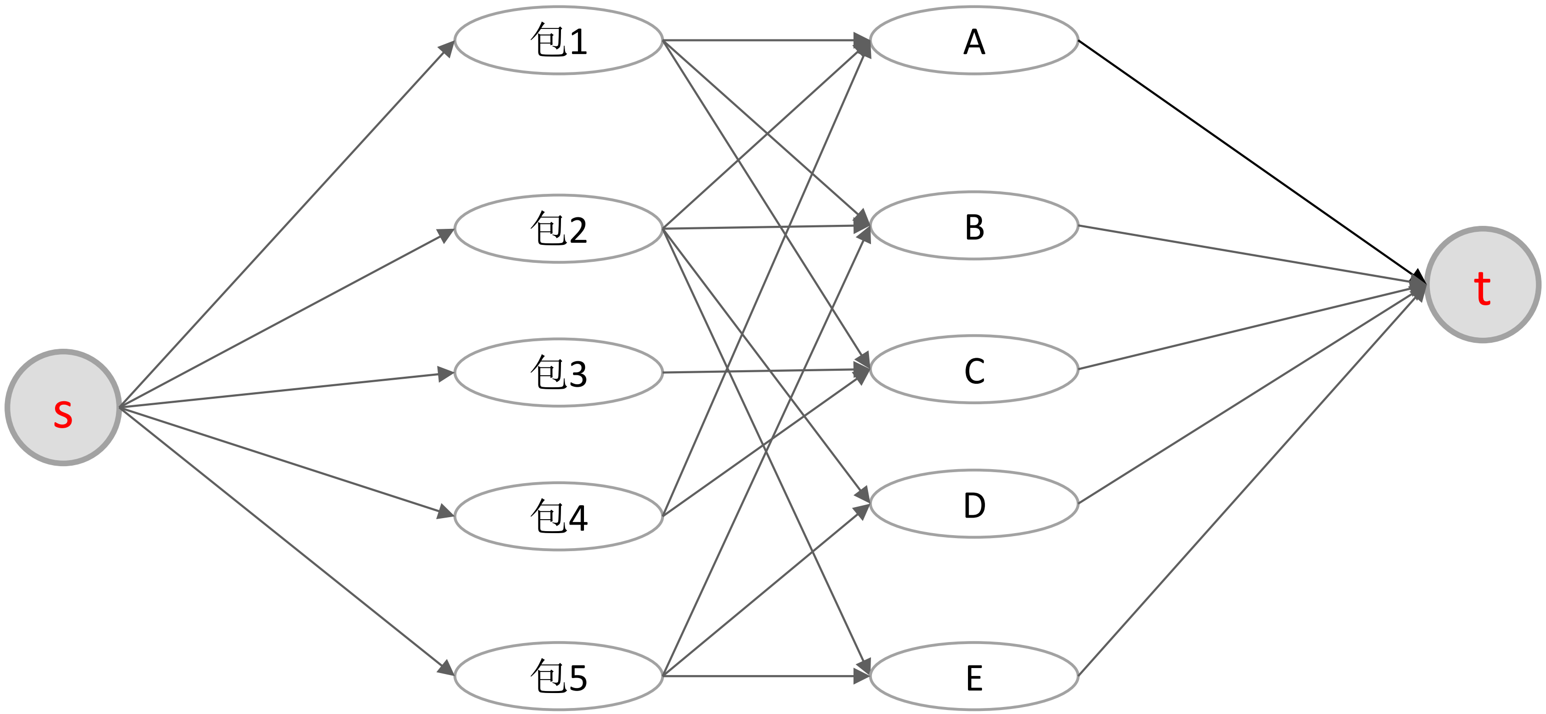
有5个背包和5件货品ABCDE，由于包的形状、大少、载重量的限制，每个包可装的货品不同：

- 第1个包可装A B C
- 第2个包可装A B D E
- 第3个包只能装C
- 第4个包可装A C
- 第5个包可装D E

如果每个包只能装一件货品，如何分配货品，使装上货品的包数量最多？

Applications

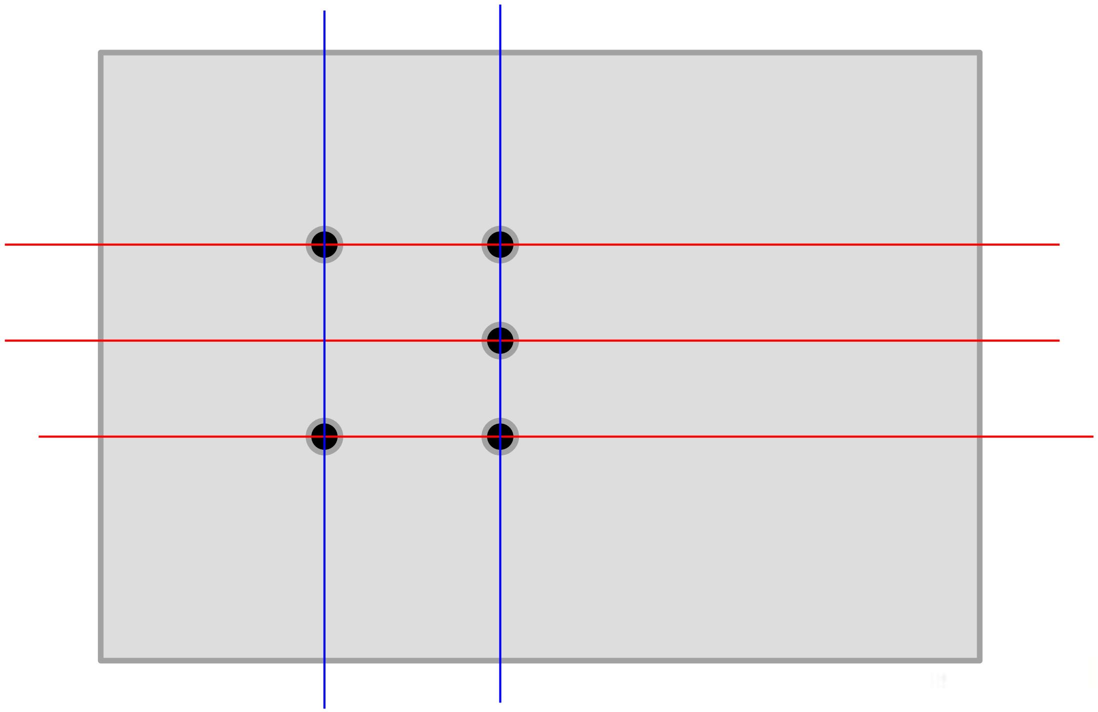
解题思路：　（流量图中所有边的容量为1）



Applications

线（点）覆盖问题

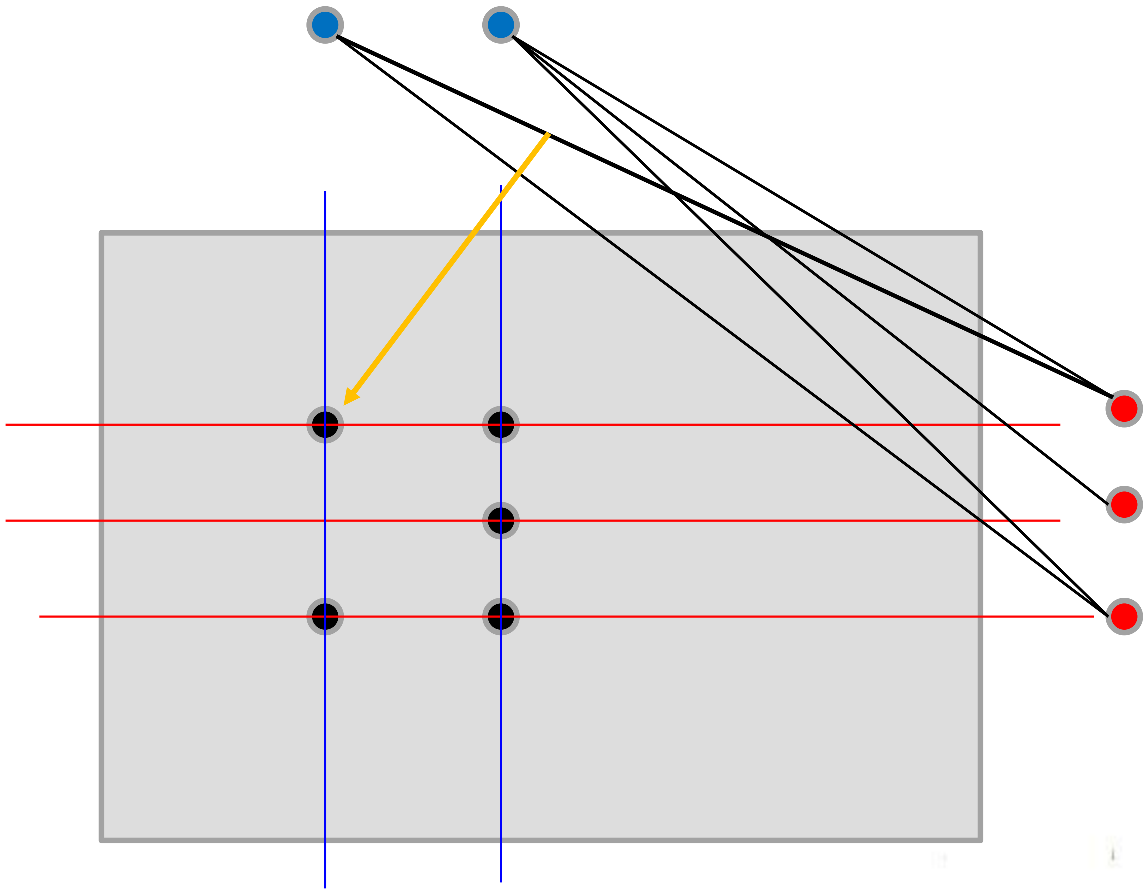
平面上有n个点(x_1,y_1), (x_2,y_2),……,(x_n,y_n)。通过在平面划横线或竖线把一个或多个点覆盖，如图。若要覆盖所有的点，最少可以划多少条线？



最少划两条线

Applications

解题思路：用蓝色点和红色点分别表示横线和竖线，用红蓝两点之间的边表示两条线可同时覆盖的平面上的点。



最少划两条线

求覆盖所有边（原来平面上的点）的最少点（原题中的横线或竖线）的问题，可转换为求蓝色的点集与红色点集之间的最大比配问题

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.