


《数据结构与算法》课程组  
重庆大学计算机学院



# Data Structures & Algorithms





**MAXIMUM FLOW**

# Outlines

---

**18.1 Flow networks**

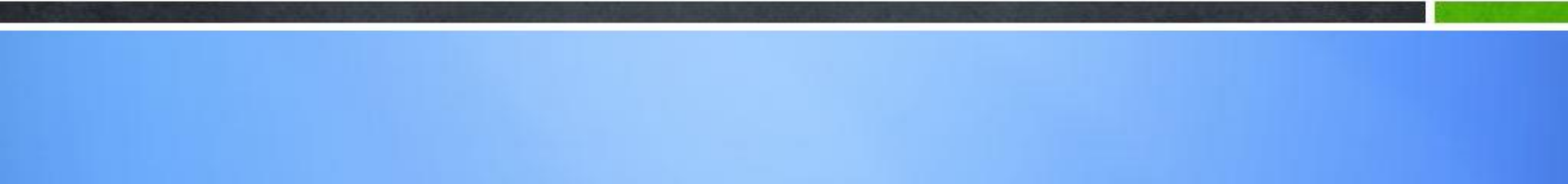
**18.2 Ford-Fulkerson method**

**18.3 Edmonds & Karp Algorithm**

**18.4 Applications**



# **18.1 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.”

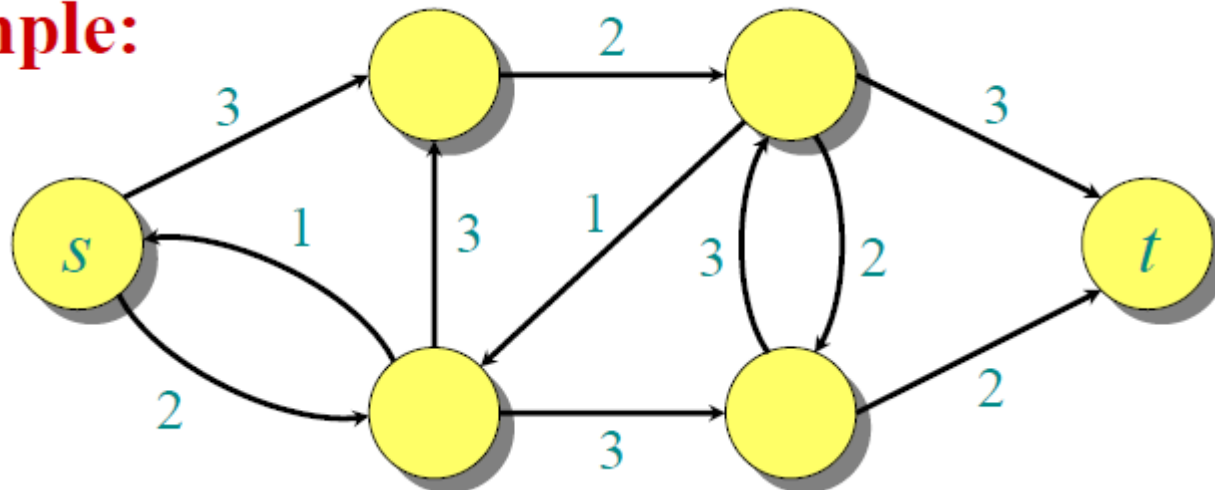
-Ford & Fulkerson, *Ford & Fulkerson Algorithm*

“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

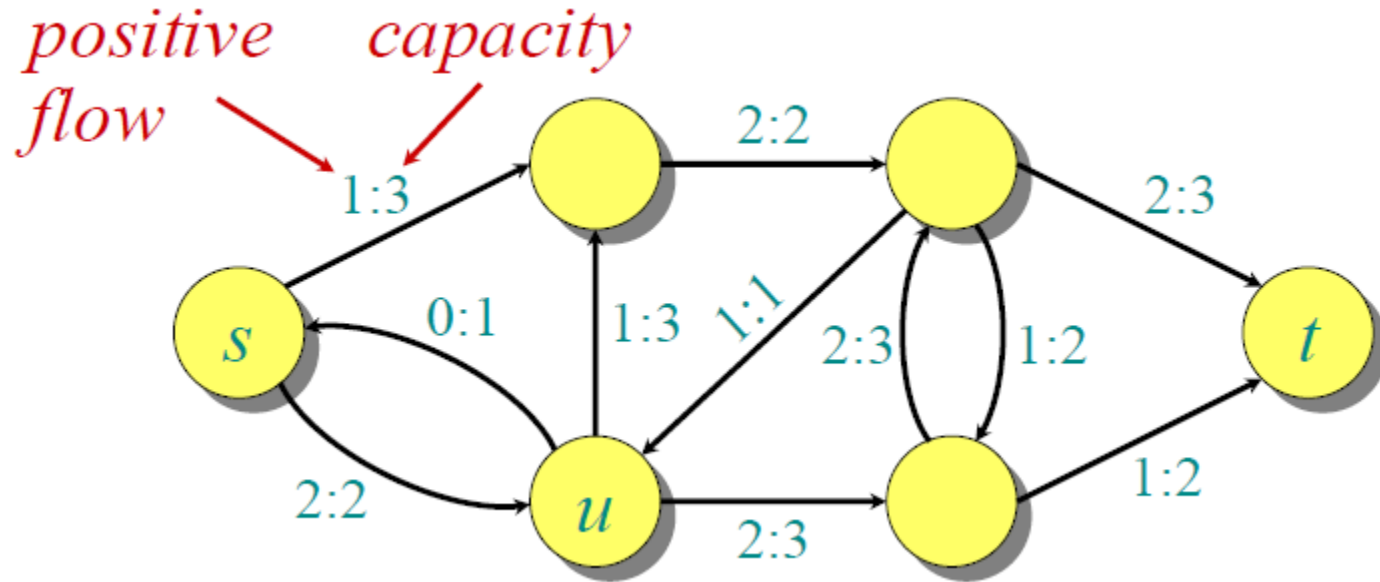
**Example:**



# 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:  $p : V \times V \rightarrow \mathbb{R}$ 
  - **Capacity constraint:** For all  $u, v \in V$ ,  
 $0 \leq p(u, v) \leq c(u, v)$ .
  - **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 Kirchhoff's current law):

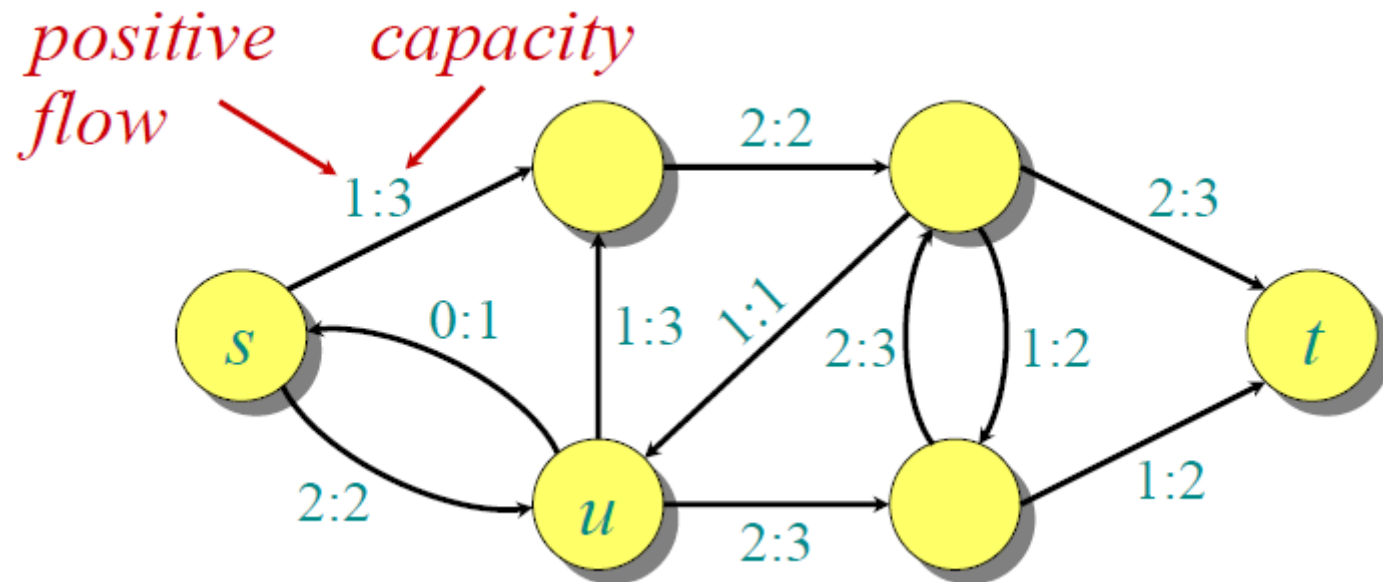
- Flow into  $u$  is  $2 + 1 = 3$ .
- Flow out of  $u$  is  $0 + 1 + 2 = 3$ .



# Flow Value

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

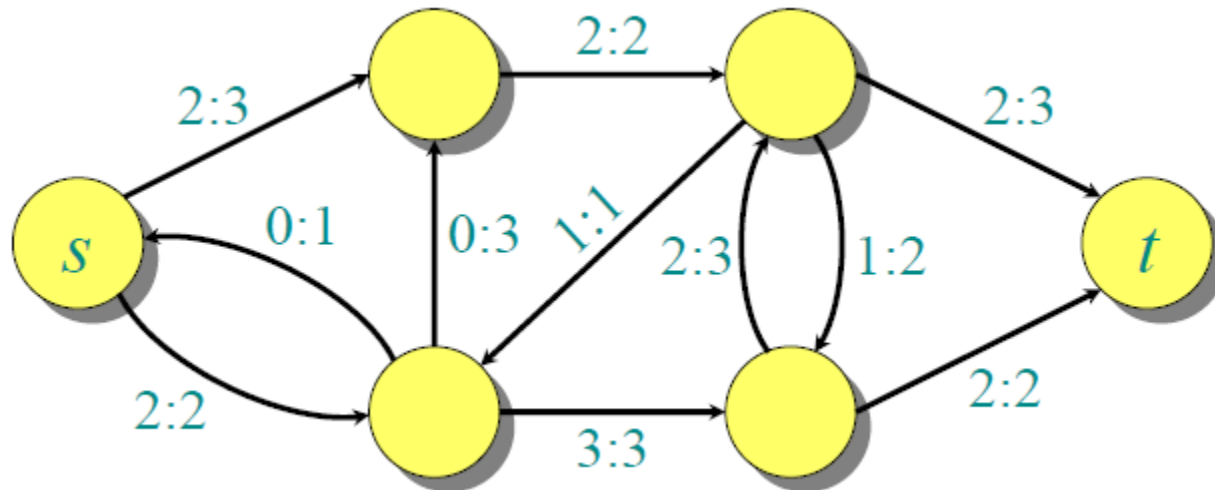
$$\sum_{v \in V} p(s, v) - \sum_{v \in V} p(v, s).$$



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

# The Maximum-Flow Problem

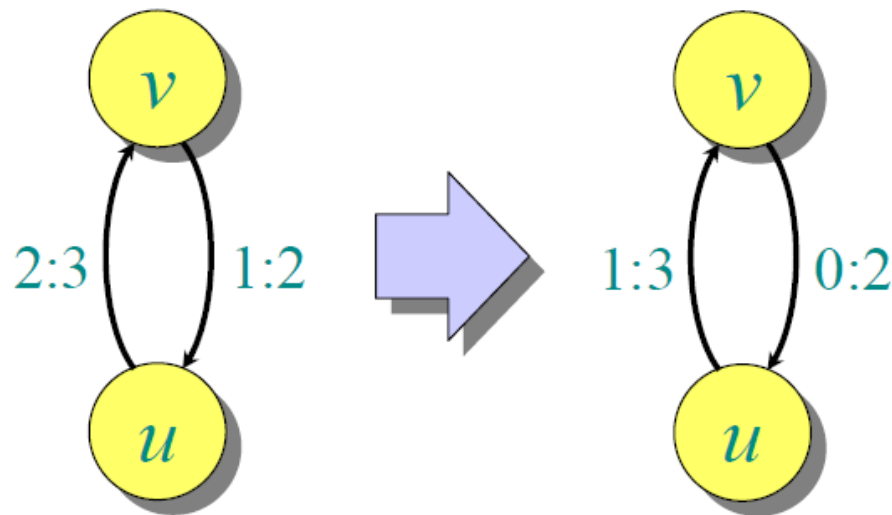
**Maximum-flow problem:** Given a flow network  $G$ , find a flow of maximum value on  $G$ .



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  $f : V \times V \rightarrow \mathbb{R}$  satisfying the following:

- **Capacity constraint:** For all  $u, v \in V$ ,  
 $f(u, v) \leq c(u, v)$ .
- **Skew symmetry:** For all  $u, v \in V$ ,  
 $f(u, v) = -f(v, u)$ .
- **Flow conservation:** For all  $u \in V - \{s, t\}$ ,  
$$\sum_{v \in V} f(u, v) = 0.$$
 ← One summation instead of two.

# Net Flow Value

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

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



**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).

$$\begin{aligned} f(X \cup Y, Z) &= \sum_{s \in X \cup Y} \sum_{t \in Z} f(s, t) \\ &= \left( \sum_{s \in X} + \sum_{s \in Y} \right) \sum_{t \in Z} f(s, t) \quad \because X \cap Y = \emptyset \\ &= \sum_{s \in X} \sum_{t \in Z} f(s, t) + \sum_{s \in Y} \sum_{t \in Z} f(s, t) \\ &= f(X, Z) + f(Y, Z) \end{aligned}$$

# Simple Properties of Net Flow

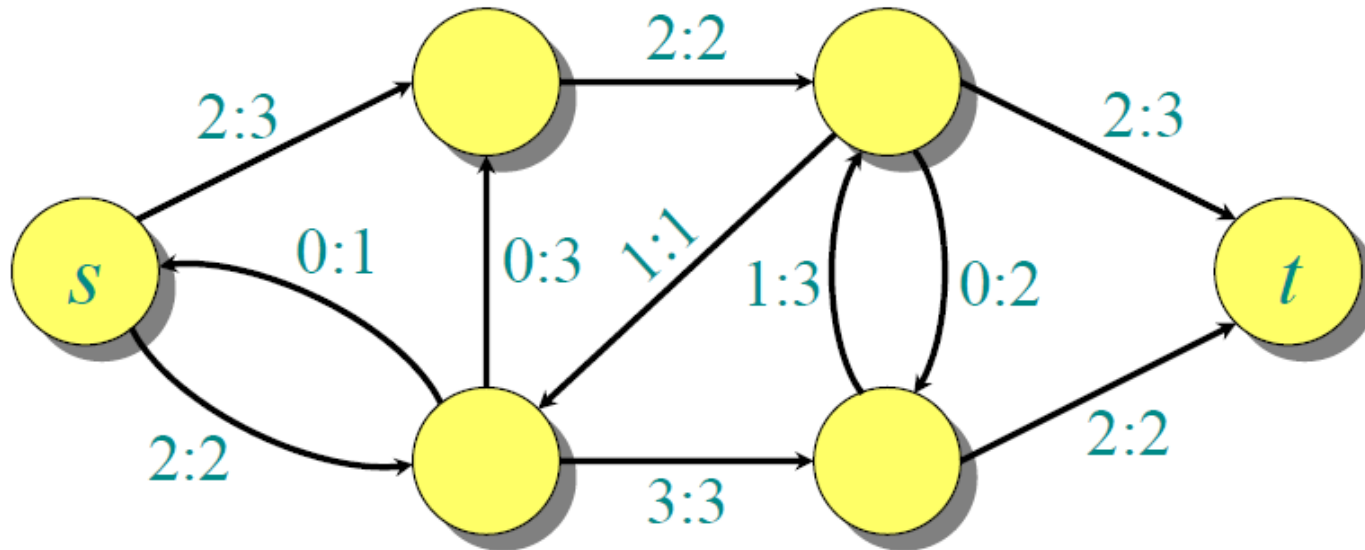
**Theorem.**  $|f| = f(V, t)$ .

*Proof.*

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



# Net Flow into Sink



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

$$f(V, t) = 4$$

# 思考题

可删除

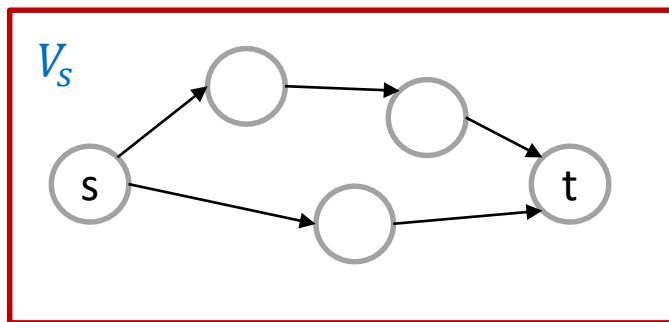
如果流量图中有节点 $v$ ，从源节点 $s$ 到达该节点的路径不存在，则有最大流，从节点 $v$ 流向其它节点的流为零，且从其它节点流向 $v$ 的流也为零。



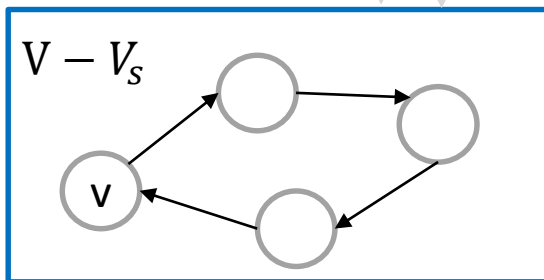
$$\forall u \in V: p(u, v) = 0 \wedge p(v, u) = 0$$

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

子图



子图



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



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



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

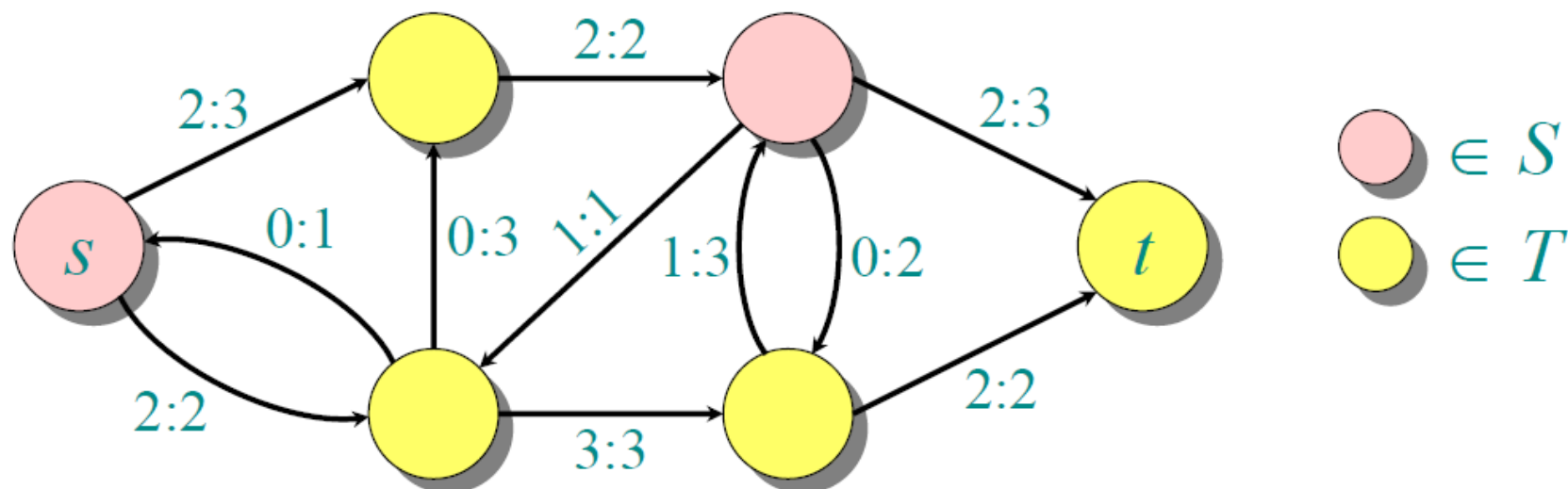


$$\therefore f(V - V_s, V - V_s) = 0$$

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

# 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*

$$\begin{aligned} f(S, T) &= (2 + 2) + (-2 + 1 - 1 + 2) \\ &= 4 \end{aligned}$$

# 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  $t \in T$ .

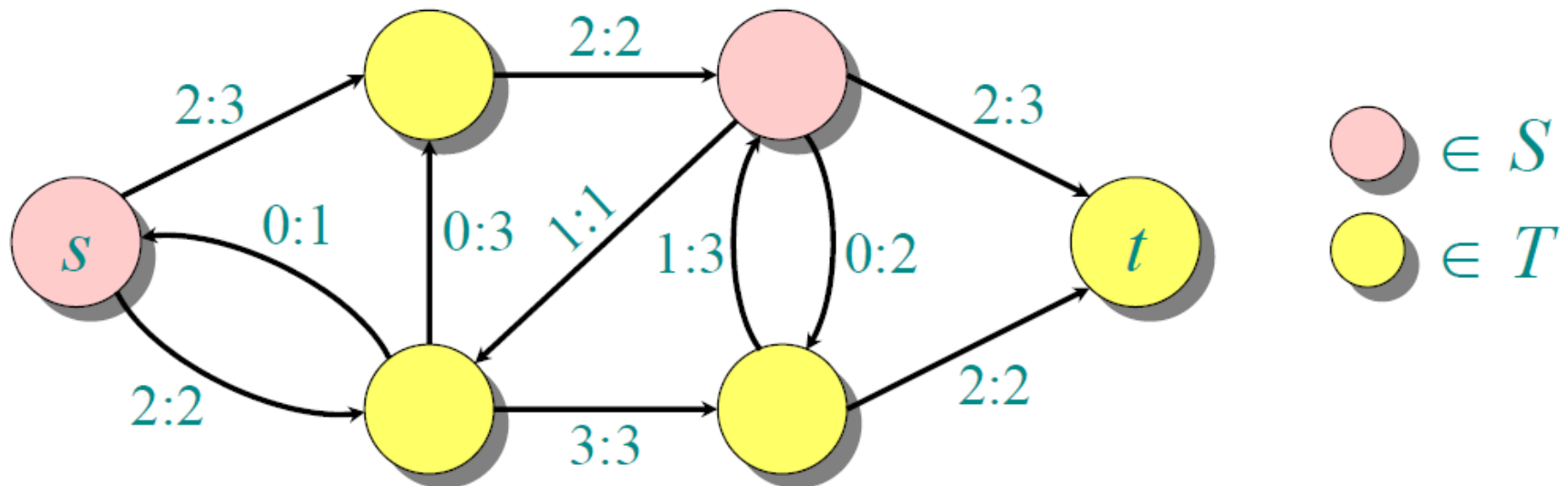
**Lemma.**  $|f| = f(S, T)$ .

*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  $c(S, T)$ .



$$\begin{aligned} c(S, T) &= (3 + 2) + (1 + 2 + 3) \\ &= 11 \end{aligned}$$

# Upper Bound on Flow Value

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

*Proof.*

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

# 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*

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

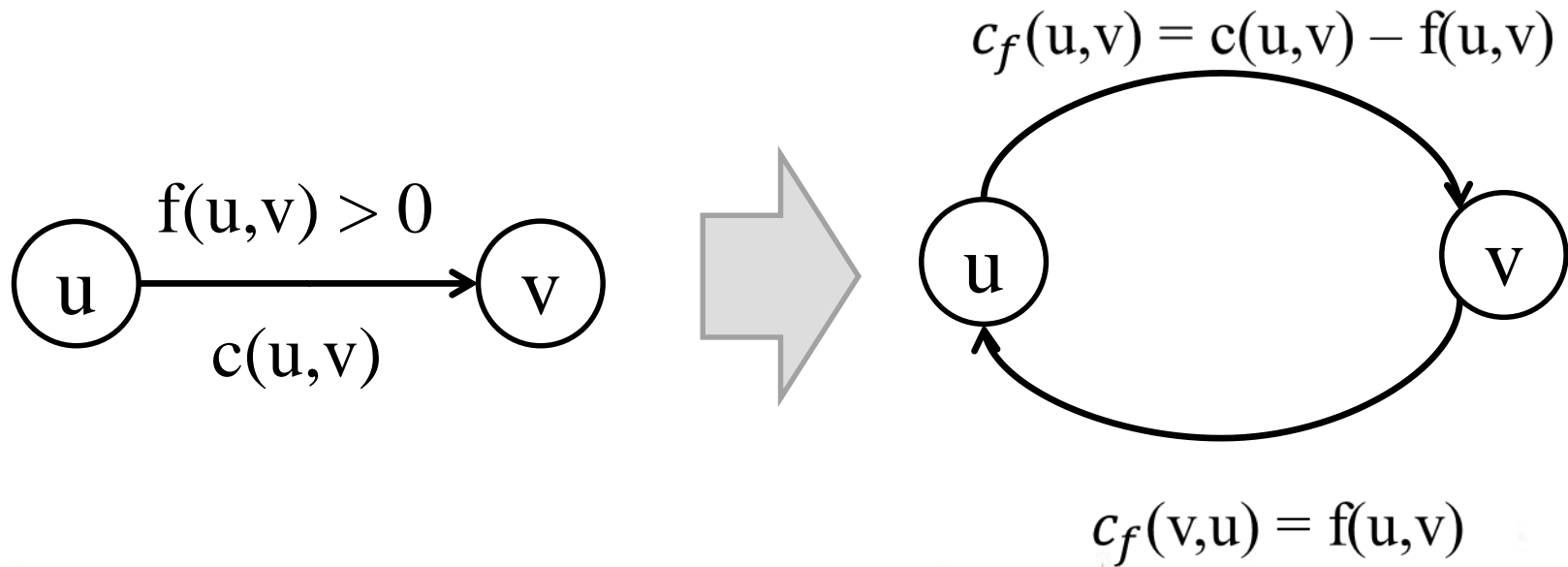
**Example:**



Edges in  $E_f$  admit more flow.

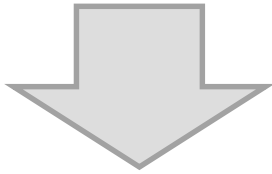
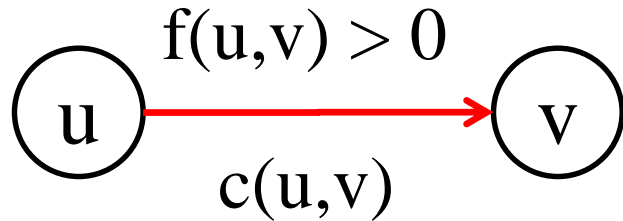
# Residual Network

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





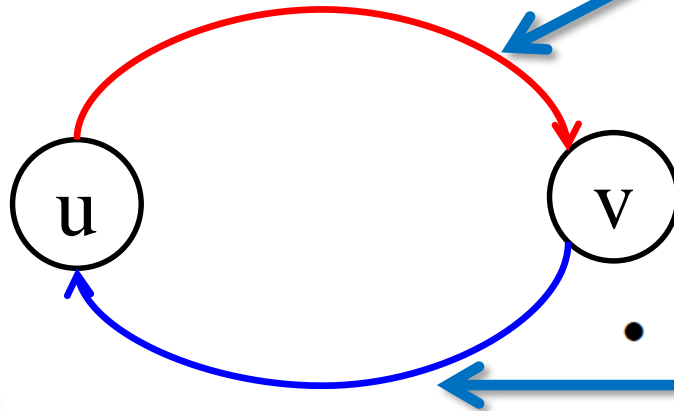
# Residual Network



- **Forward Edges**

$$c_f(u,v) = c(u,v) - f(u,v)$$

$\text{flow}(u,v) < \text{capacity}(u,v)$   
flow can be increased!

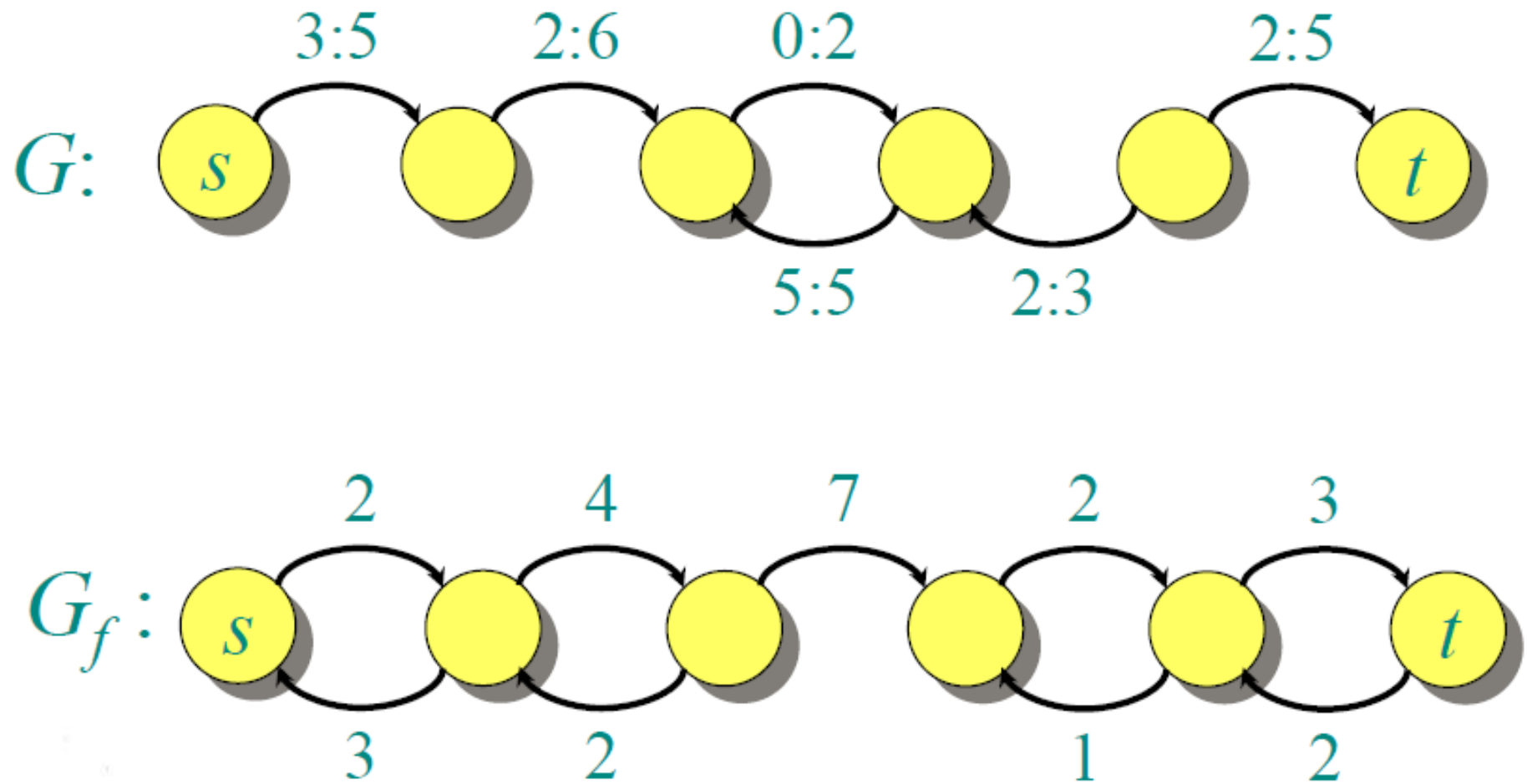


- **Backward Edges**

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

$\text{flow}(u,v) > 0$   
flow can be decreased!

# Residual Network Example



# Augmenting Path

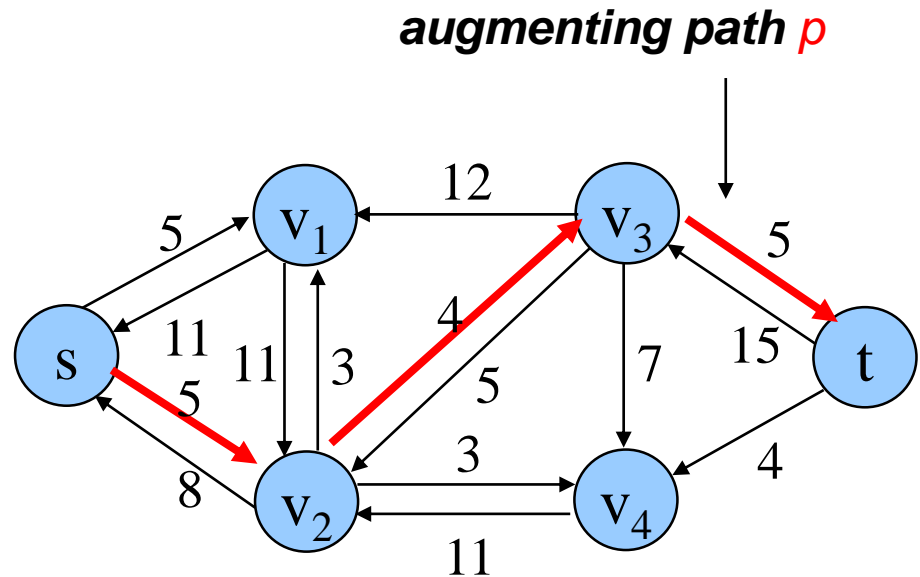
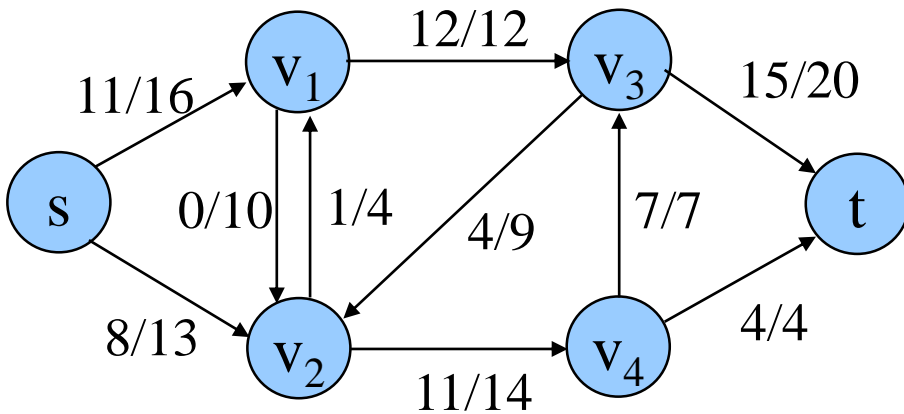
An **augmenting path**  $p$  is a simple path from  $s$  to  $t$  in the residual network  $G_f$  of a flow network  $G$ .

**residual capacity** of  $p$

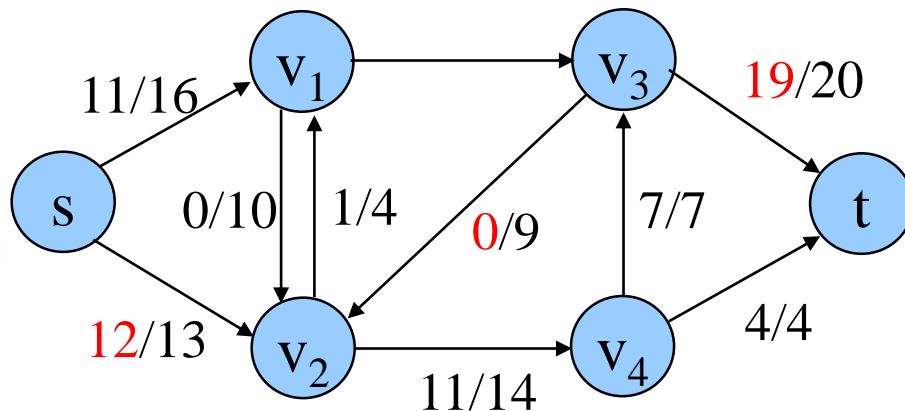
$$c_f(p) = \min_{(u,v) \in p} \{c_f(u,v)\}.$$

the maximum flow  $|f|$  can be 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  $|f| \leq c(S, T)$  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.

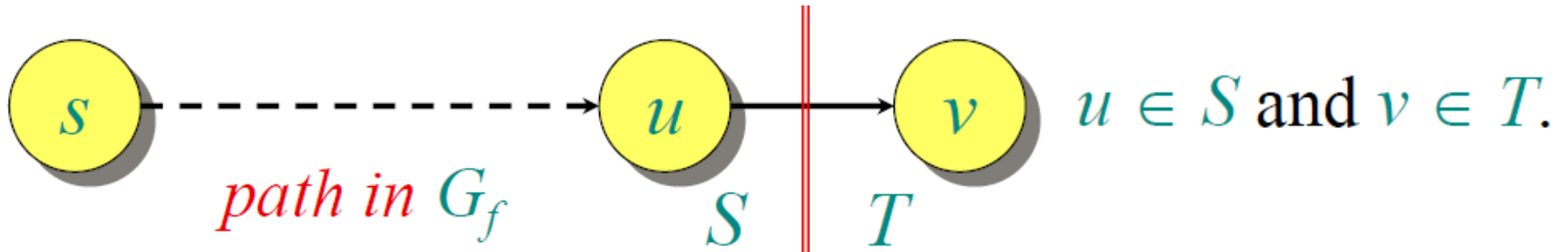
$S = \{v \in V : \text{there exists a path in } G_f \text{ from } s \text{ to } v\}$

$T = V - S$

$(S, T)$  is a cut! Why?

# Max-Flow, Min-Cut Theorem

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



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

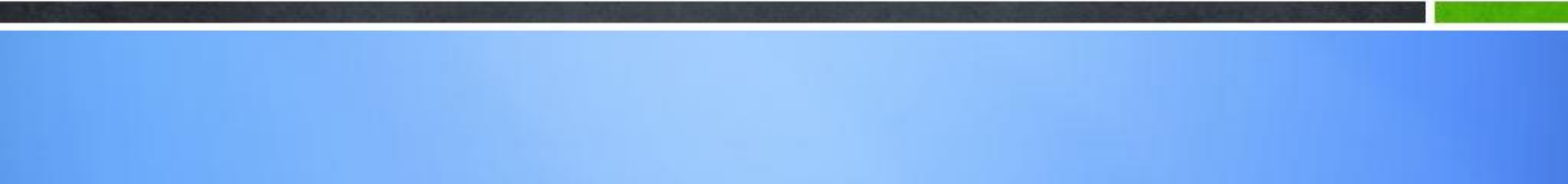
$$\Rightarrow f(u, v) = c(u, v) \quad \because c_f(u, v) = c(u, v) - f(u, v)$$

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

$$\Rightarrow f(S, T) = c(S, T) = |f| \quad \text{Maximum flow!}$$



## **18.2 Ford-Fulkerson Algorithm**



# A Funny (?) 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;

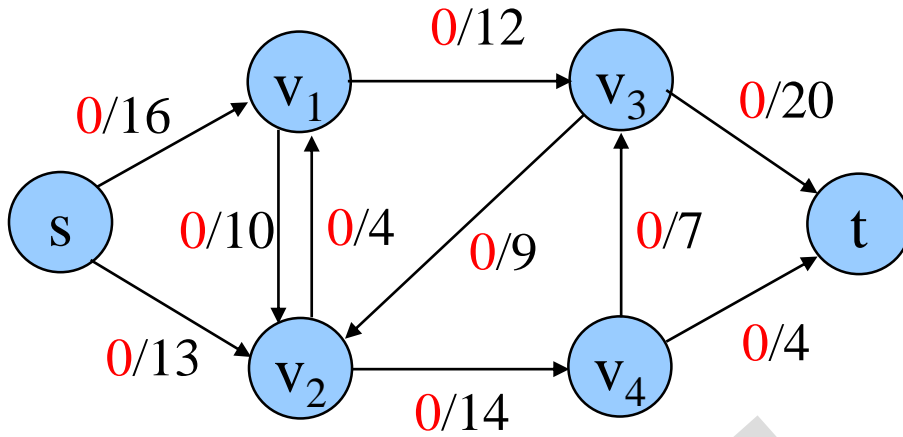
recursive call to FindFlow;

# Algorithm

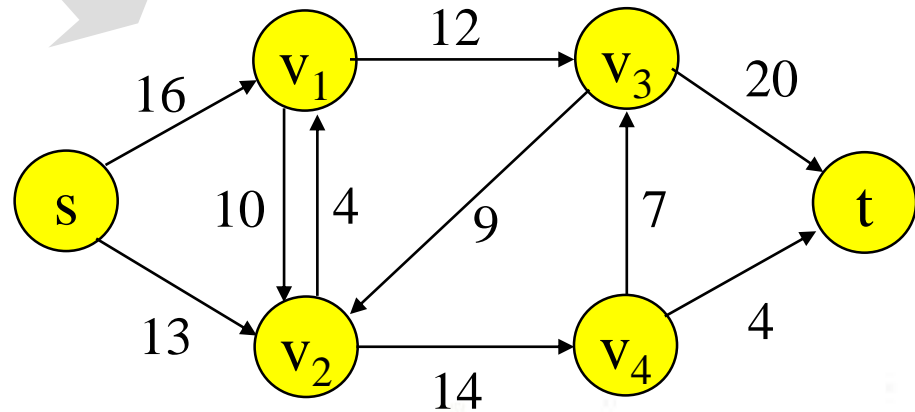
$f[u, v] \leftarrow 0$  for all  $u, v \in V$

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

# Example—Basic Implementation



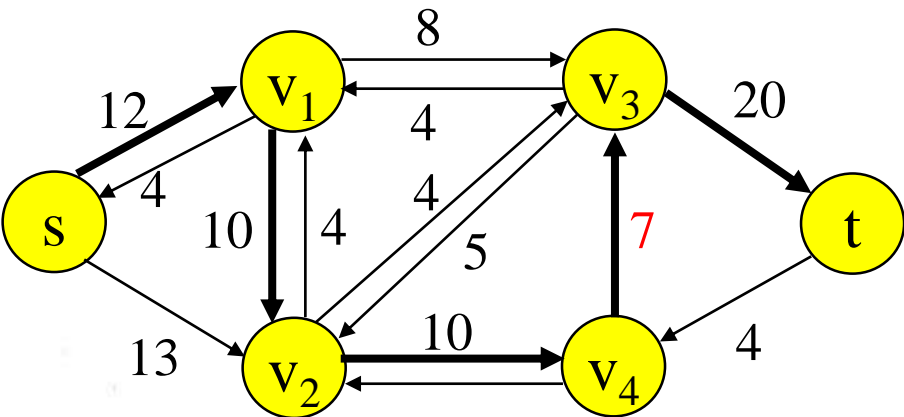
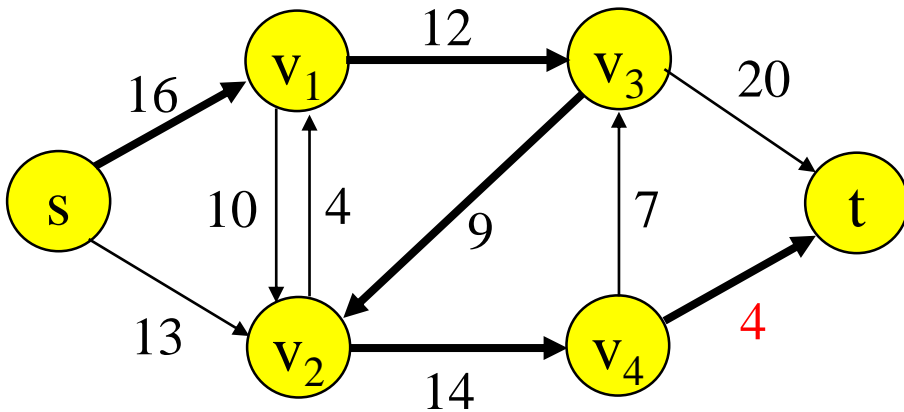
Flow initialization



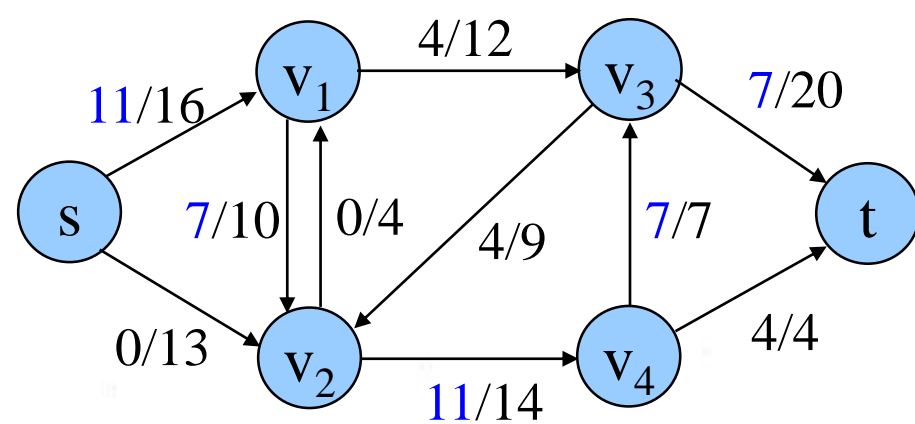
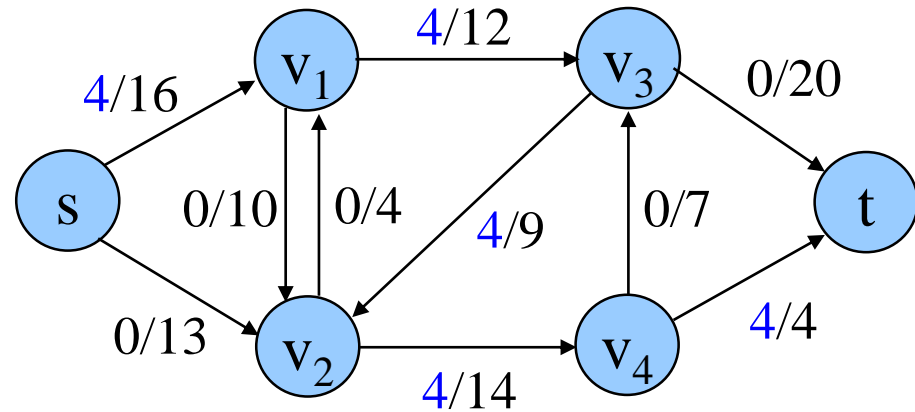
Residual network

# Example

## Residual Networks



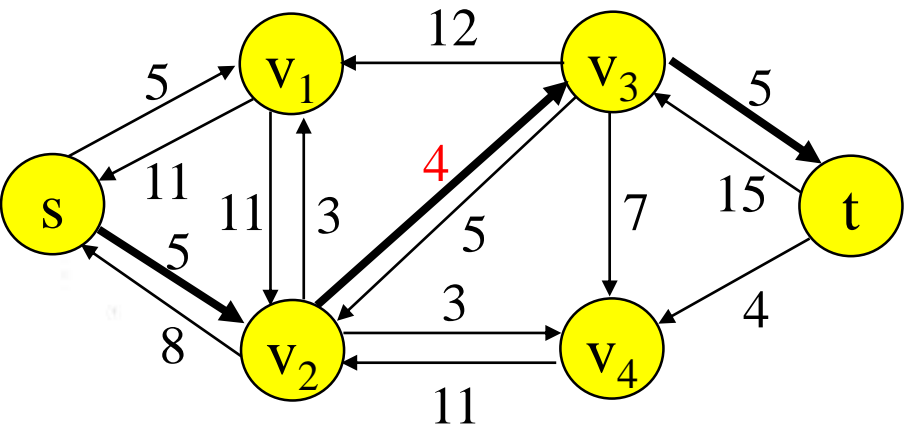
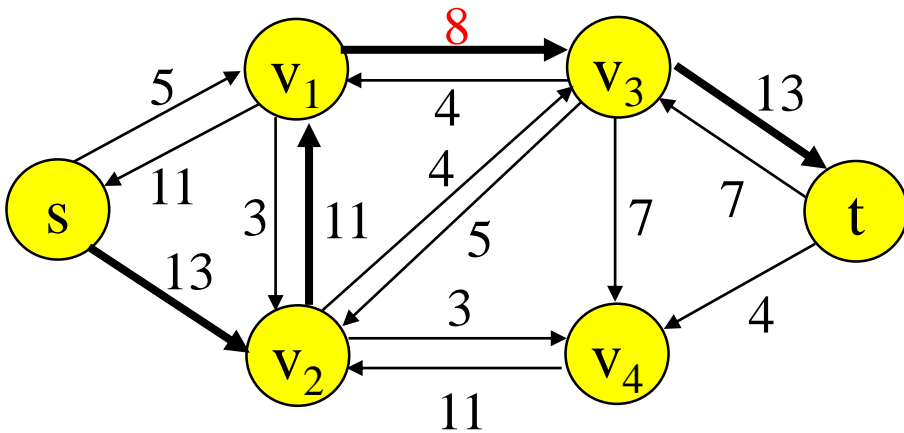
## Flows



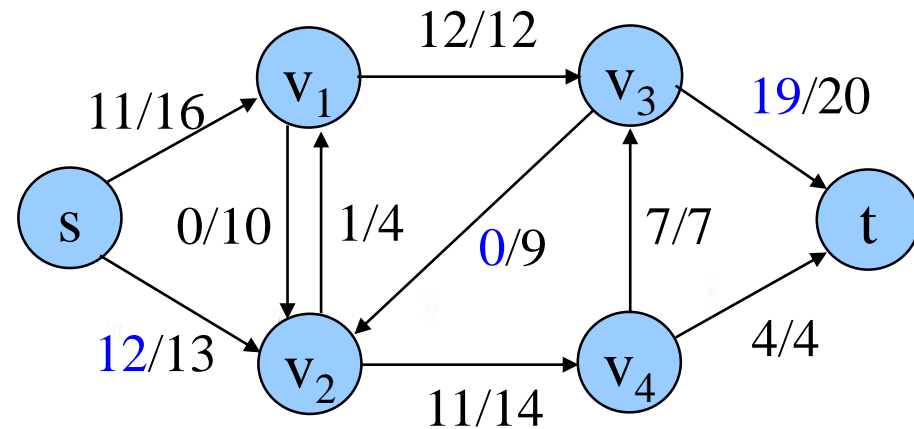
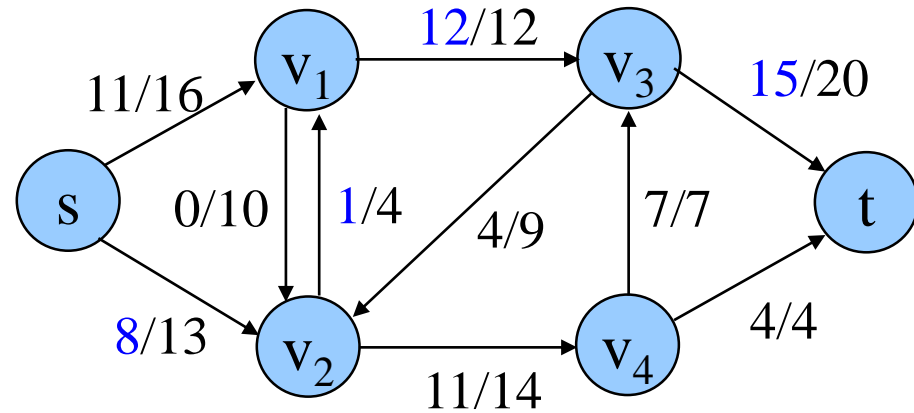


# Example

## Residual Networks

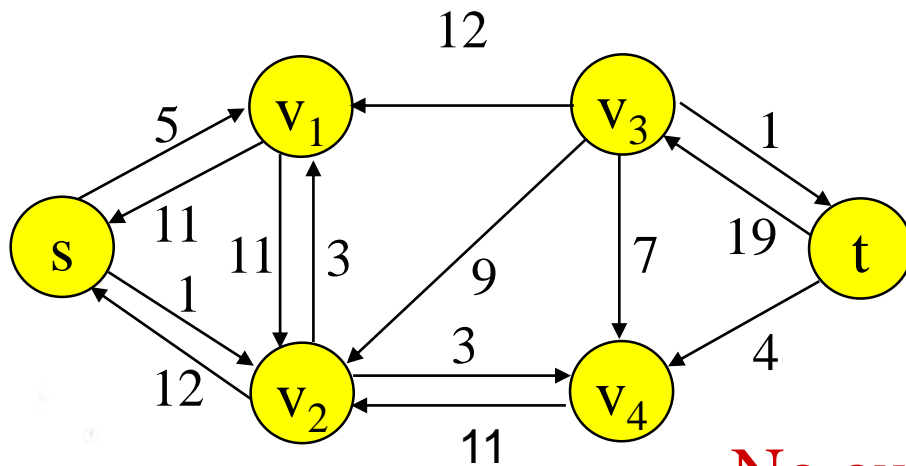


## Flows



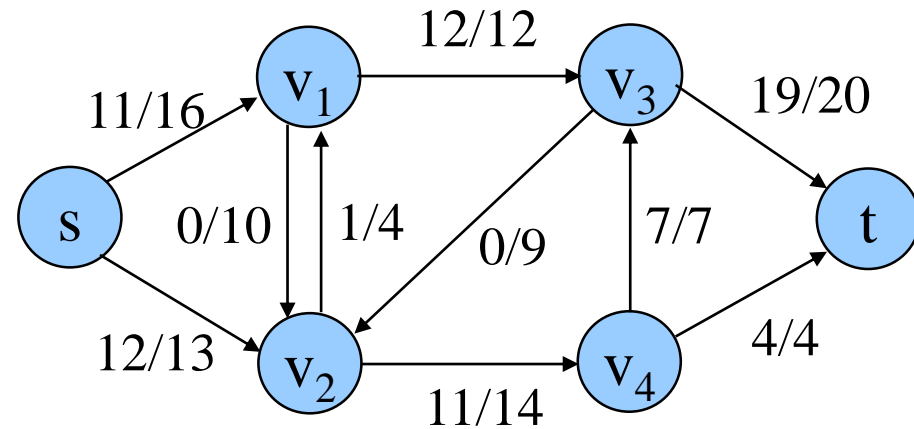
# Example

## Residual Networks



No augmenting path

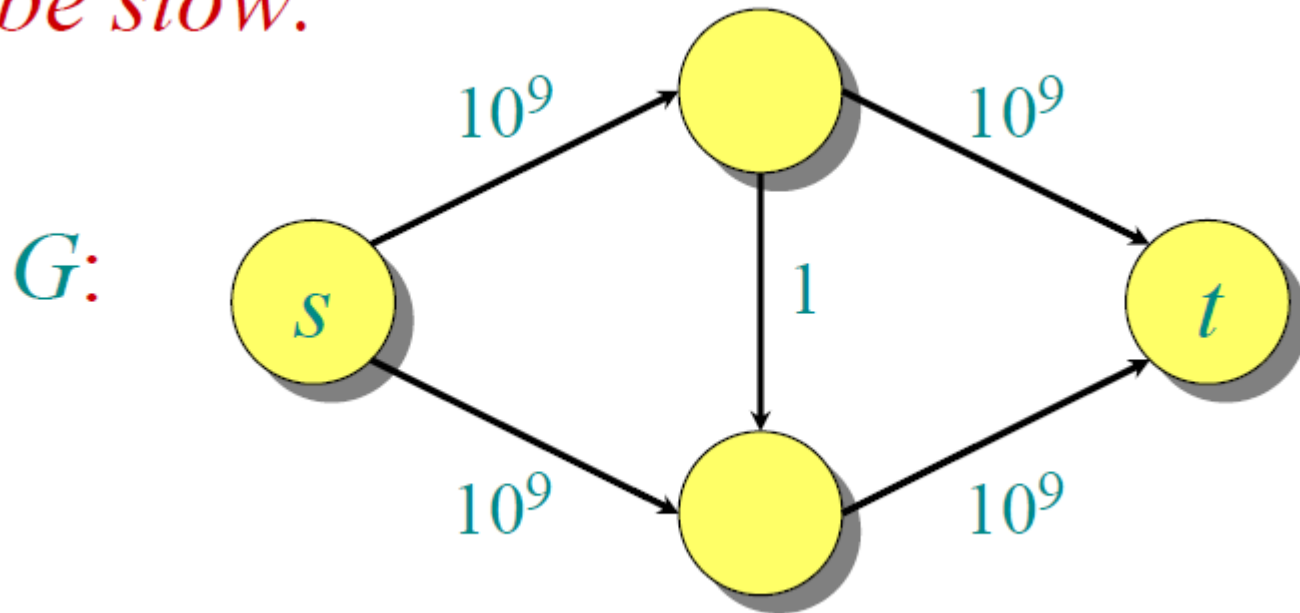
## Flows



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

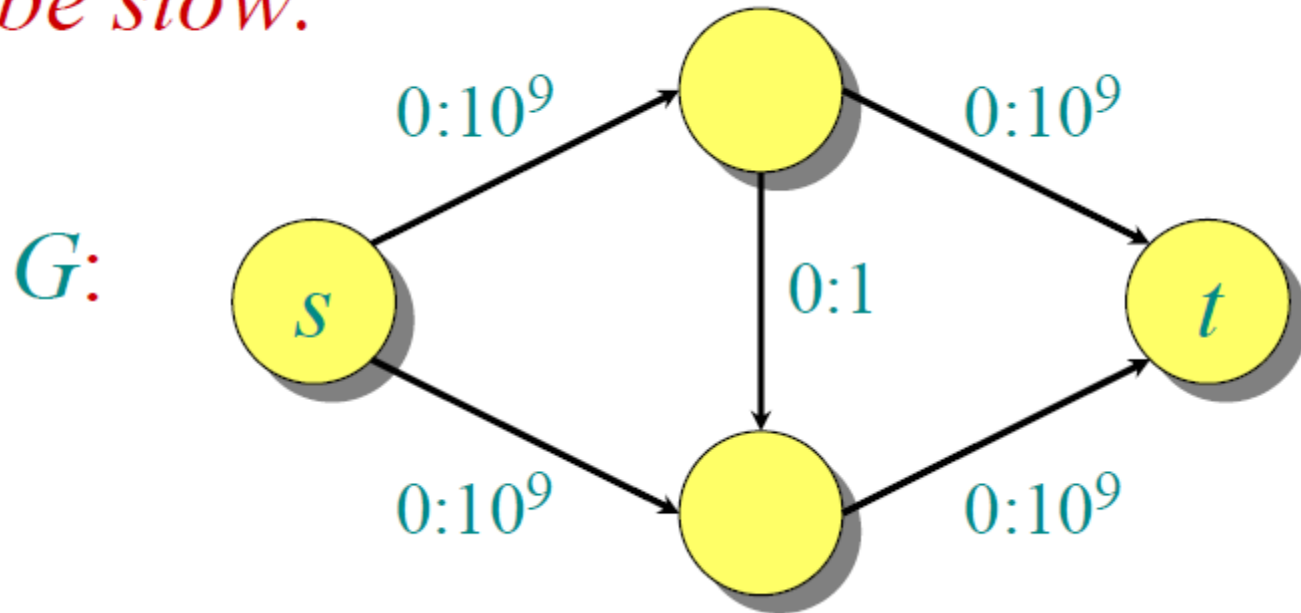
# Problem: Time Complexity

*Can be slow:*



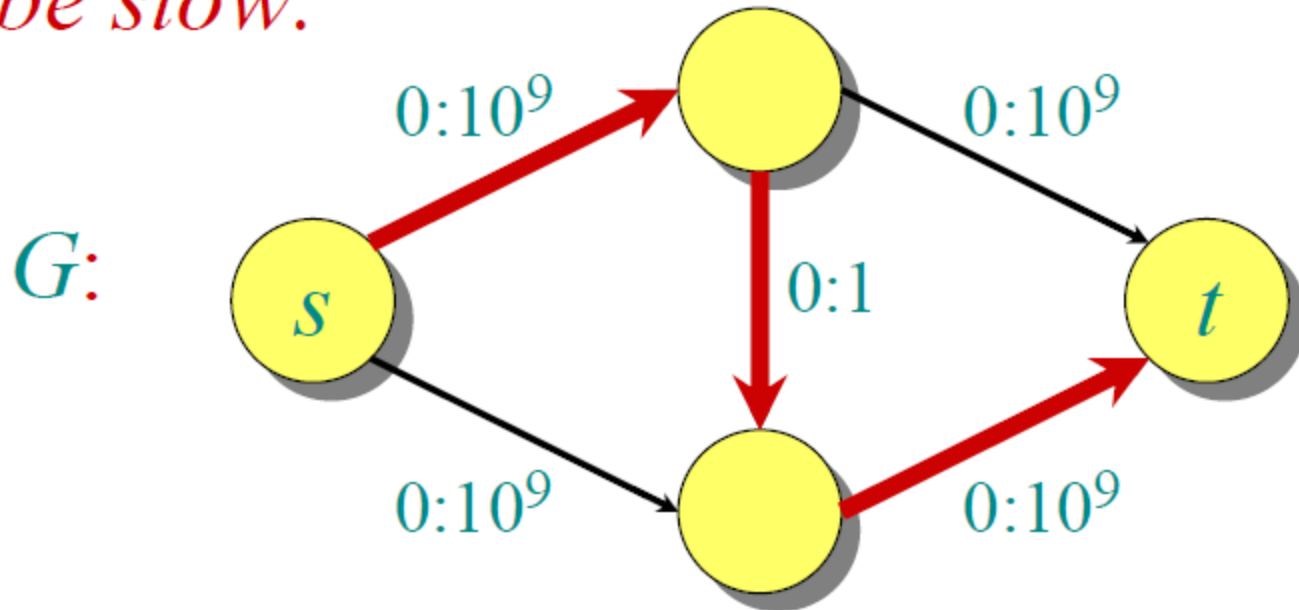
# Problem: Time Complexity

*Can be slow:*



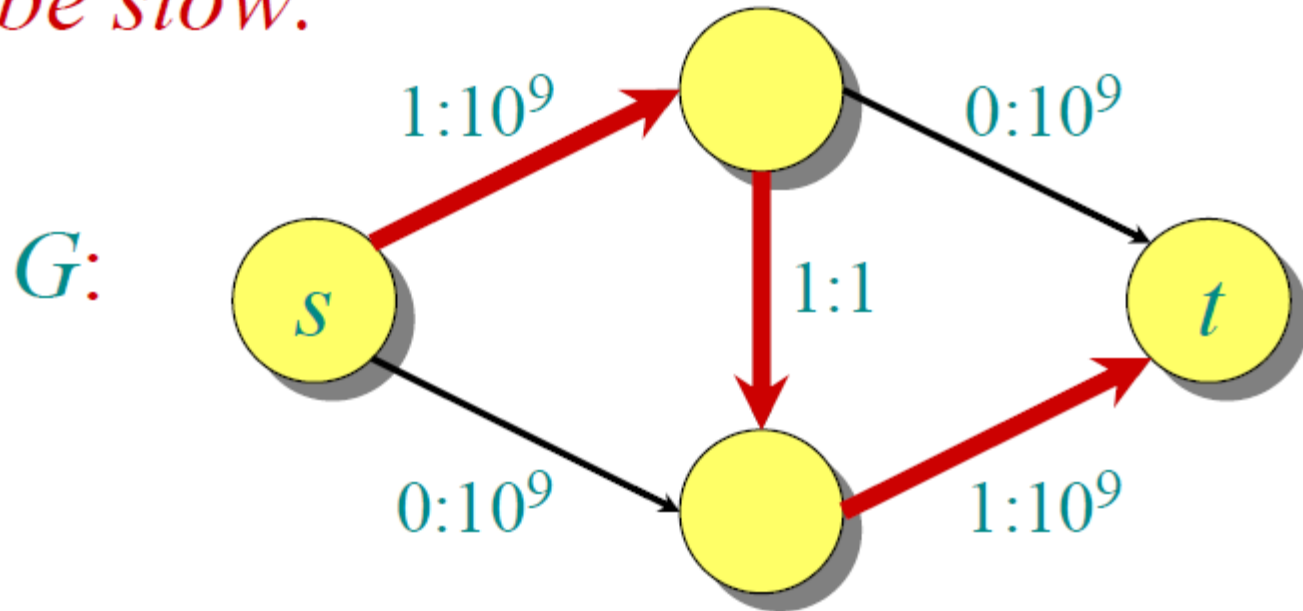
# Problem: Time Complexity

*Can be slow:*



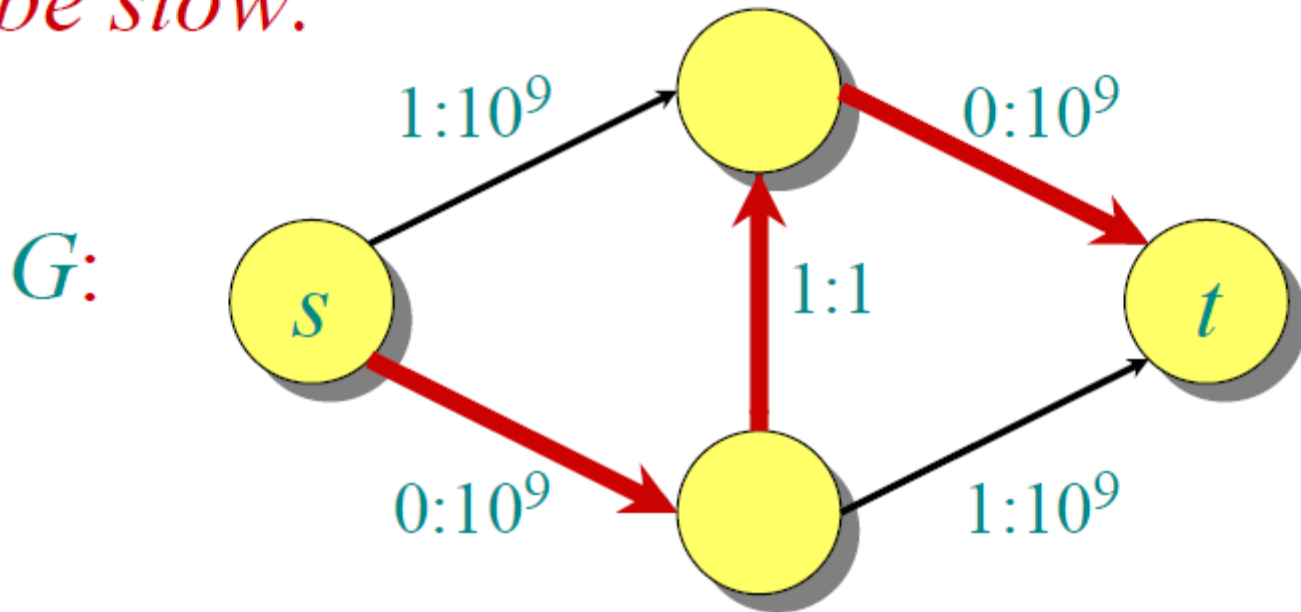
# Problem: Time Complexity

*Can be slow:*



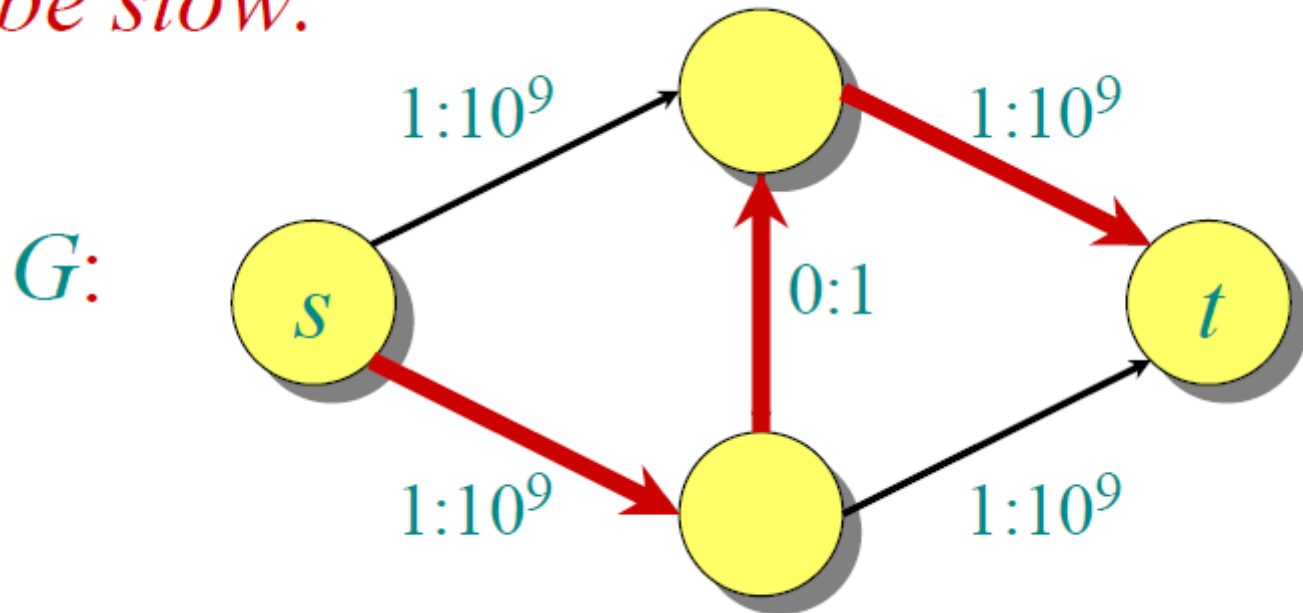
# Problem: Time Complexity

*Can be slow:*



# Problem: Time Complexity

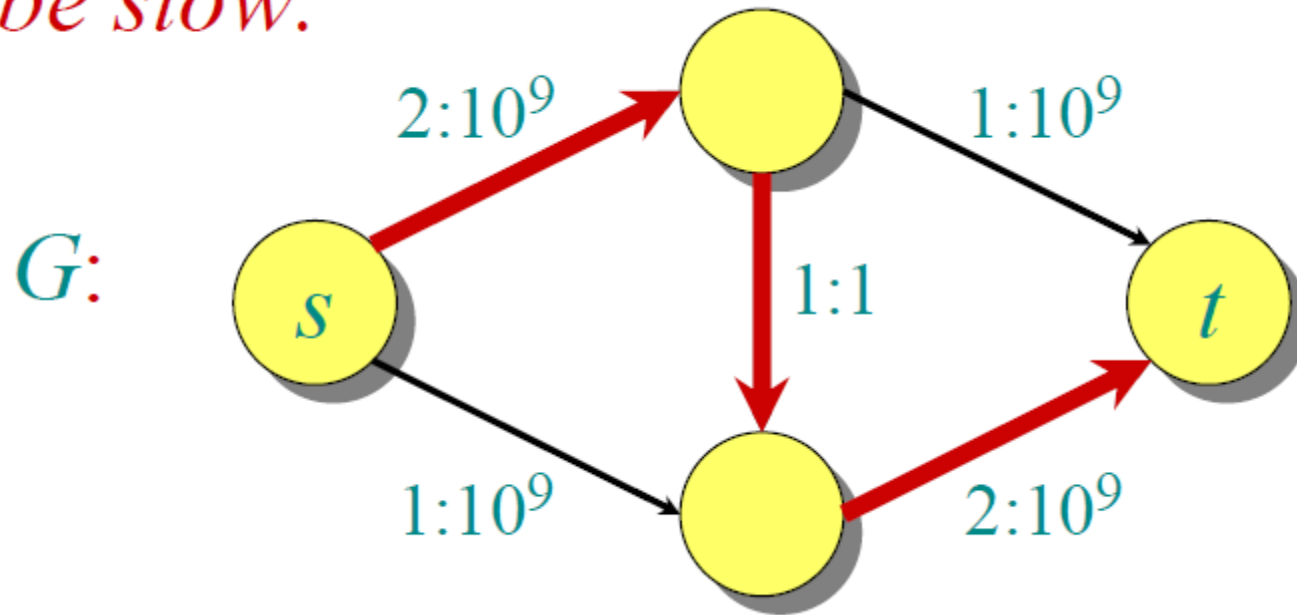
*Can be slow:*





# 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

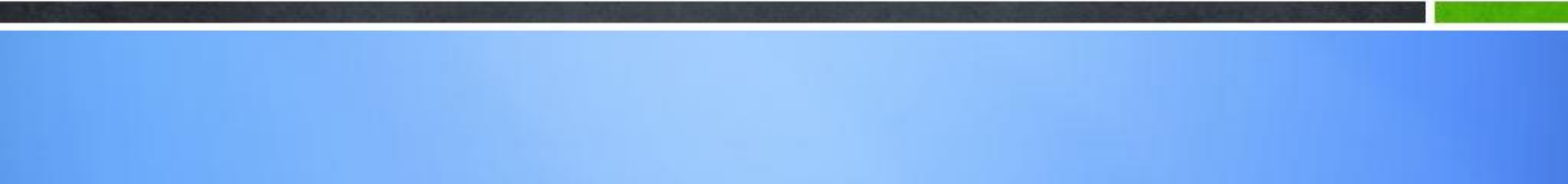
The problem with this algorithm, however, is that it is strongly dependent on the **maximum flow value  $F$** .

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

Then, along came Edmonds & Karp...



## **18.3 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.

## BFS(G,s)

```
1. for each vertex  $u$  in  $V[G] - \{s\}$ 
2     do  $color[u] \leftarrow \text{white}$ 
3          $d[u] \leftarrow \infty$ 
4          $\pi[u] \leftarrow \text{nil}$ 
5  $color[s] \leftarrow \text{gray}$ 
6  $d[s] \leftarrow 0$ 
7  $\pi[s] \leftarrow \text{nil}$ 
8  $Q \leftarrow \Phi$ 
9  $\text{enqueue}(Q,s)$ 
10 while  $Q \neq \Phi$ 
11     do  $u \leftarrow \text{dequeue}(Q)$ 
12         for each  $v$  in  $\text{Adj}[u]$ 
13             do if  $color[v] = \text{white}$ 
14                 then  $color[v] \leftarrow \text{gray}$ 
15                      $d[v] \leftarrow d[u] + 1$ 
16                      $\pi[v] \leftarrow u$ 
17                      $\text{enqueue}(Q,v)$ 
18          $color[u] \leftarrow \text{black}$ 
```

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  $v$

# Analysis of BFS

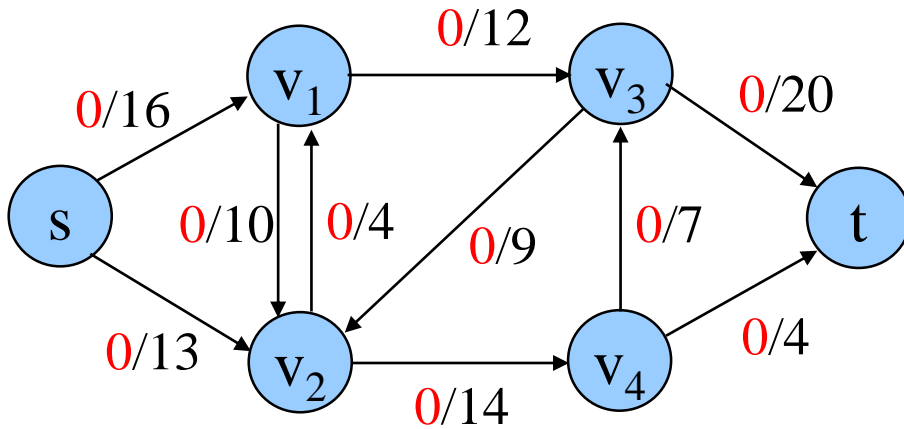
- Initialization takes  $O(|V|)$ .
- Traversal Loop
  - Each vertex is enqueued and dequeued at most once, so the total time for queuing is  $O(|V|)$ .
  - The adjacency list of each vertex is scanned at most once.
  - The sum of lengths of all adjacency lists is  $\Theta(|E|)$ .
- Total running time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

# 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!**

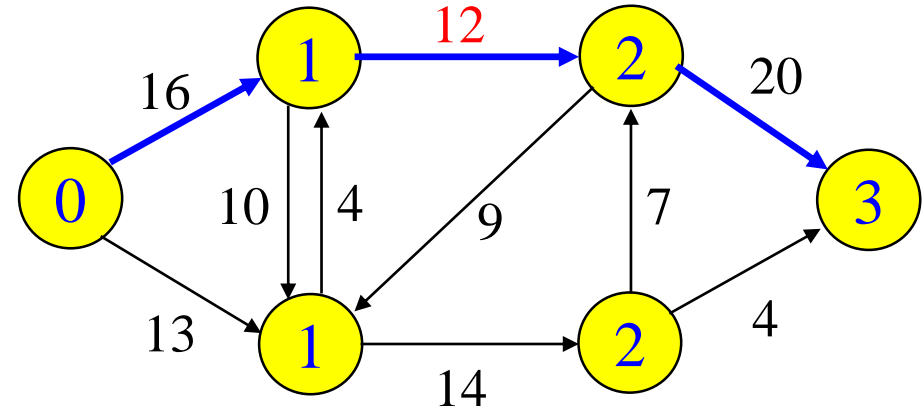
# Example

## Flows



## Residual Networks

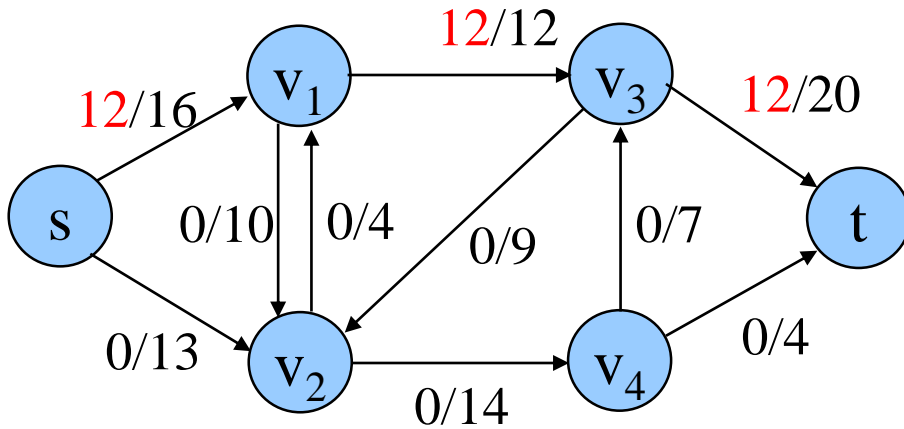
### BFS





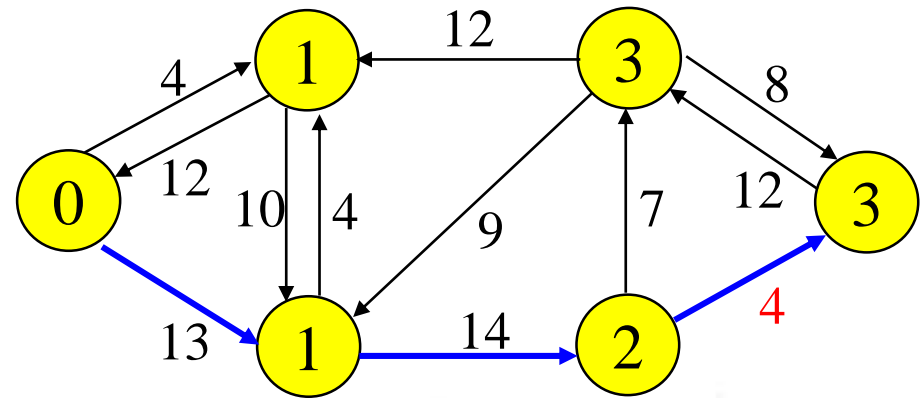
# Example

## Flows



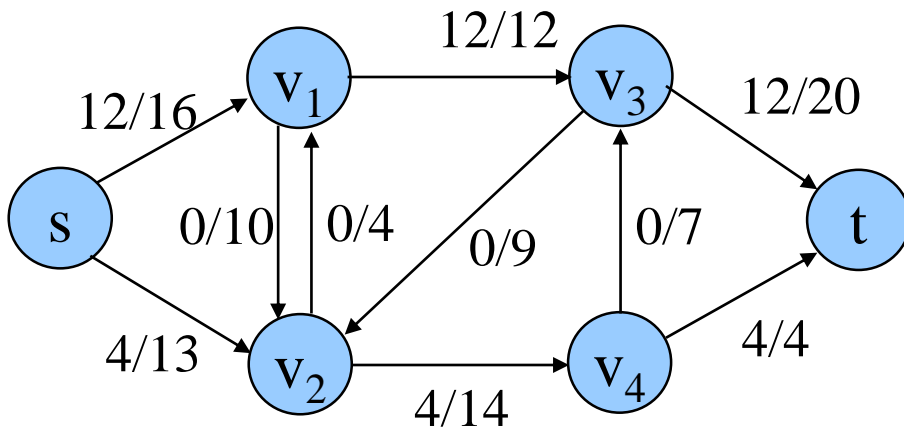
## Residual Networks

### BFS



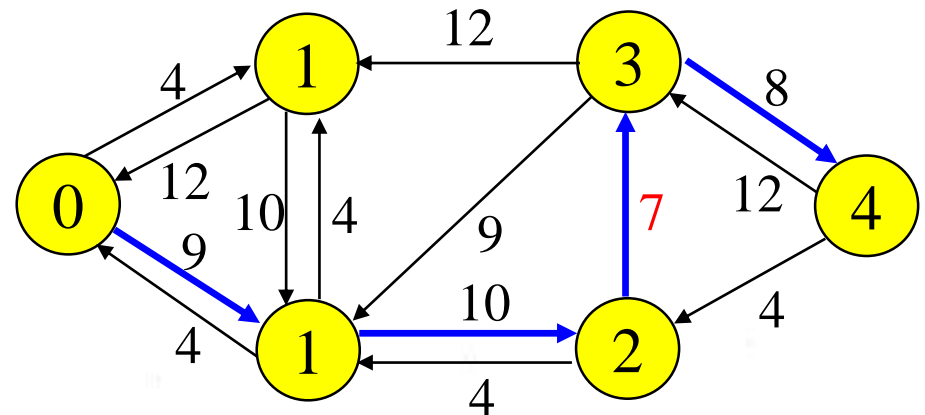
# Example

## Flows



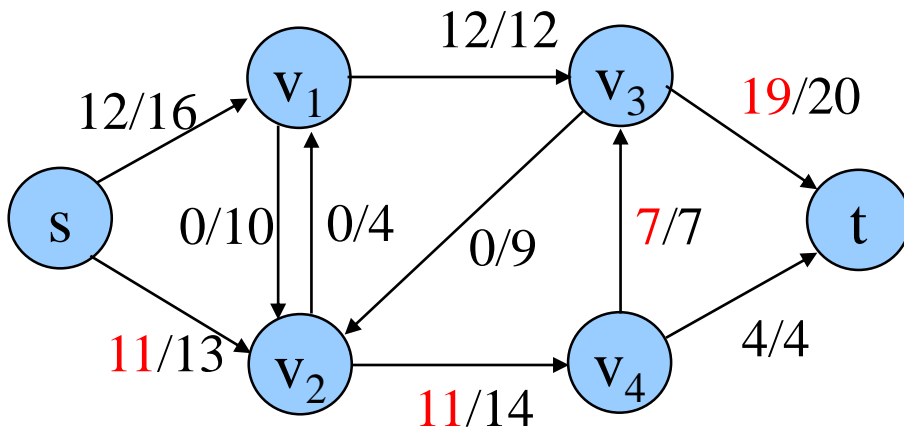
## Residual Networks

### BFS



# Example

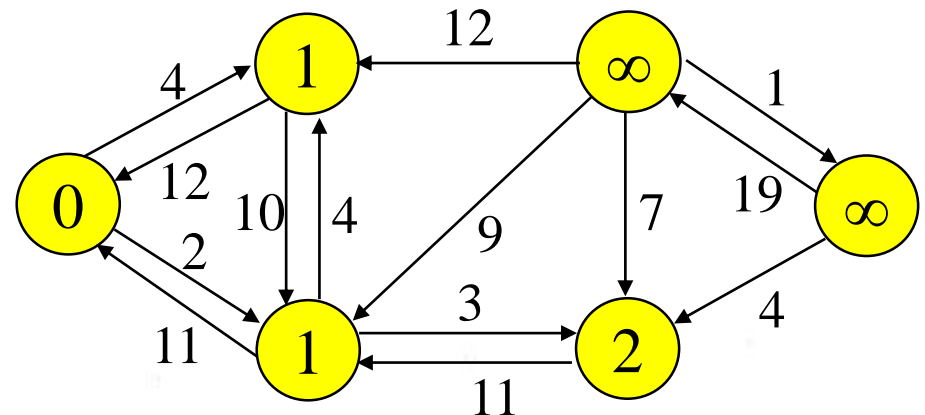
## Flows



Maximum!

## Residual Networks

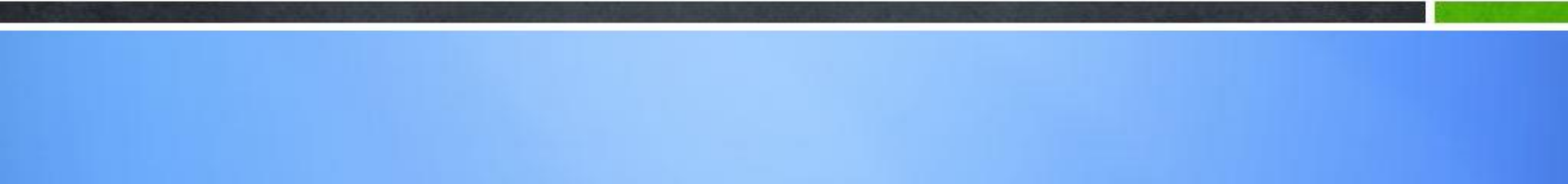
BFS



No path to sink



## **18.4 Applications**



# 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	--

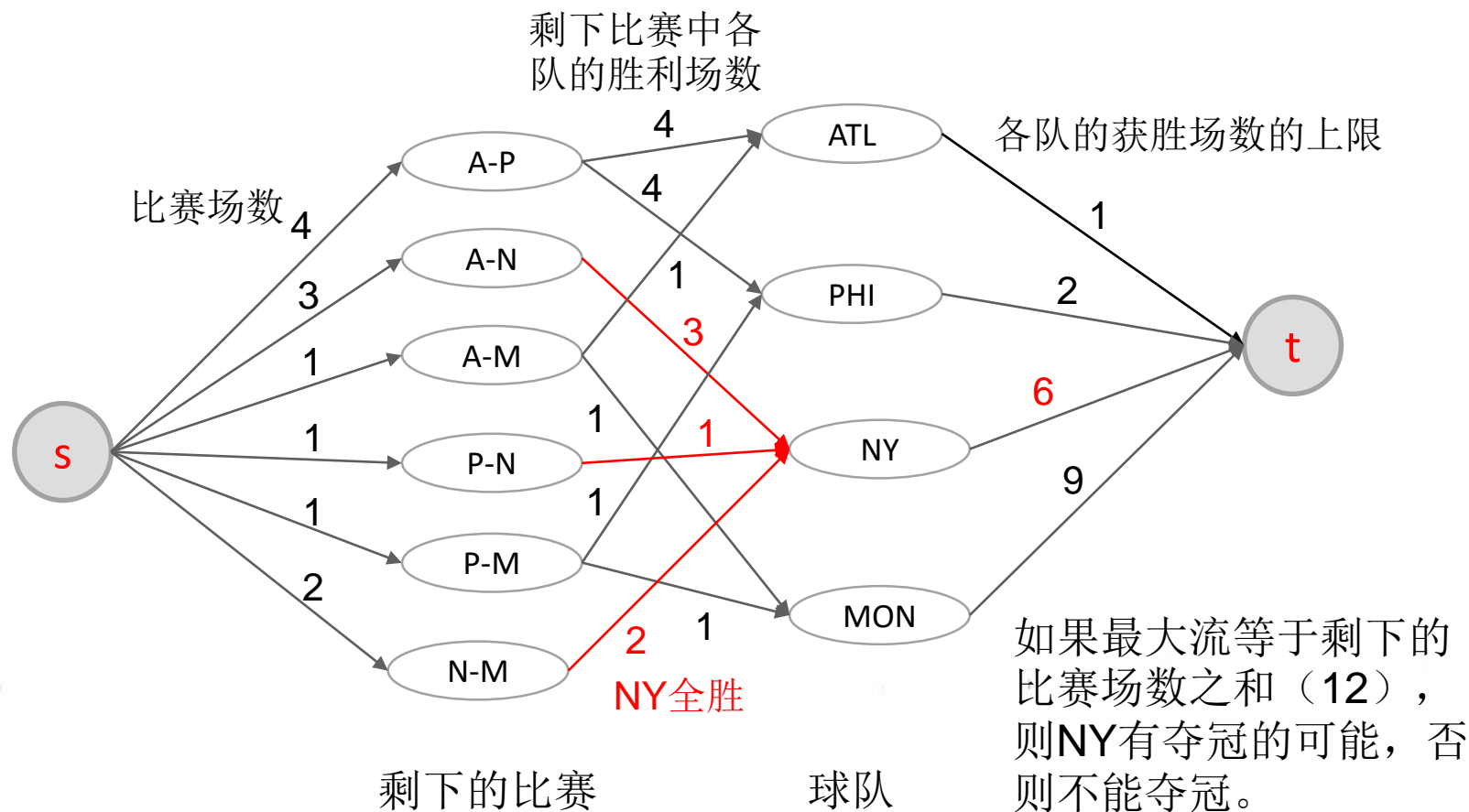
美国职业棒球的例行赛，每个球队都要打 **162** 场比赛，  
所胜场数最多者为该分区的冠军

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。



# Applications

## 二分图最大匹配问题

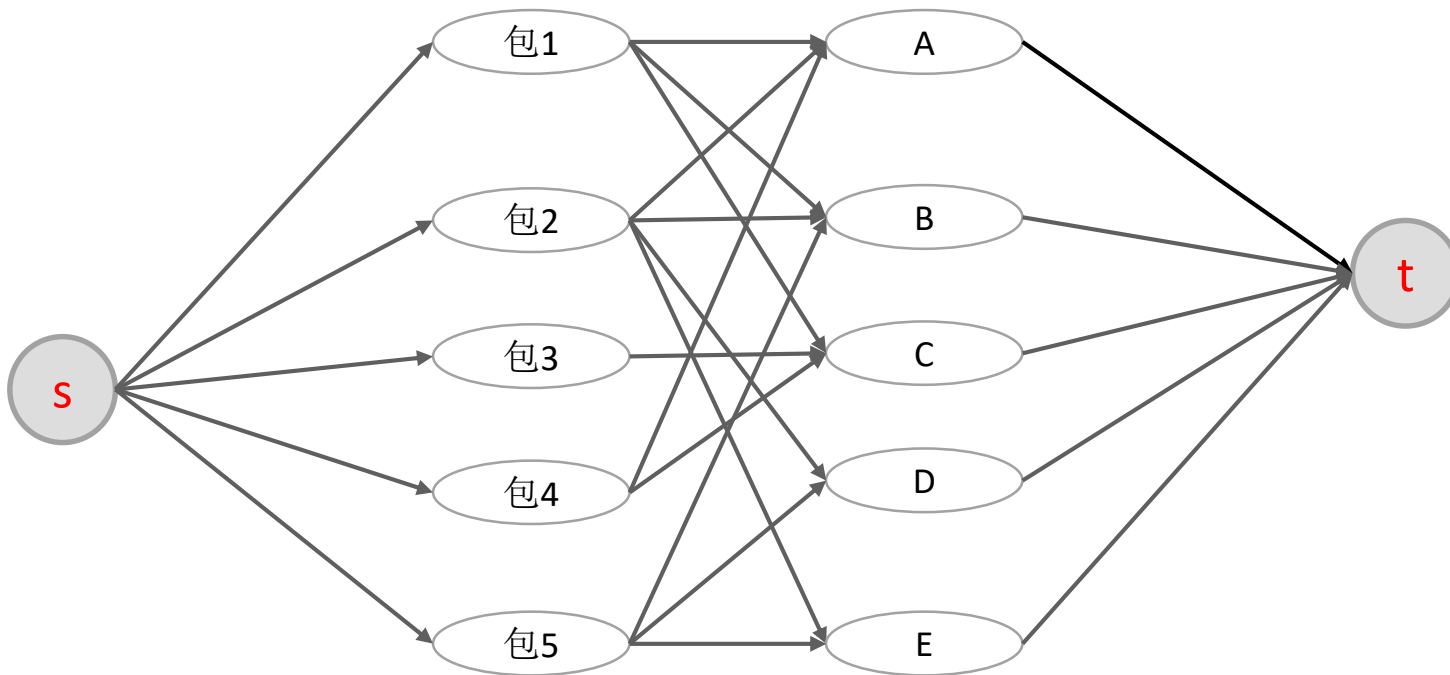
有5个背包和5件货品ABCDE，由于包的形状、大小、载重量的限制，每个包可装的货品不同：

- 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

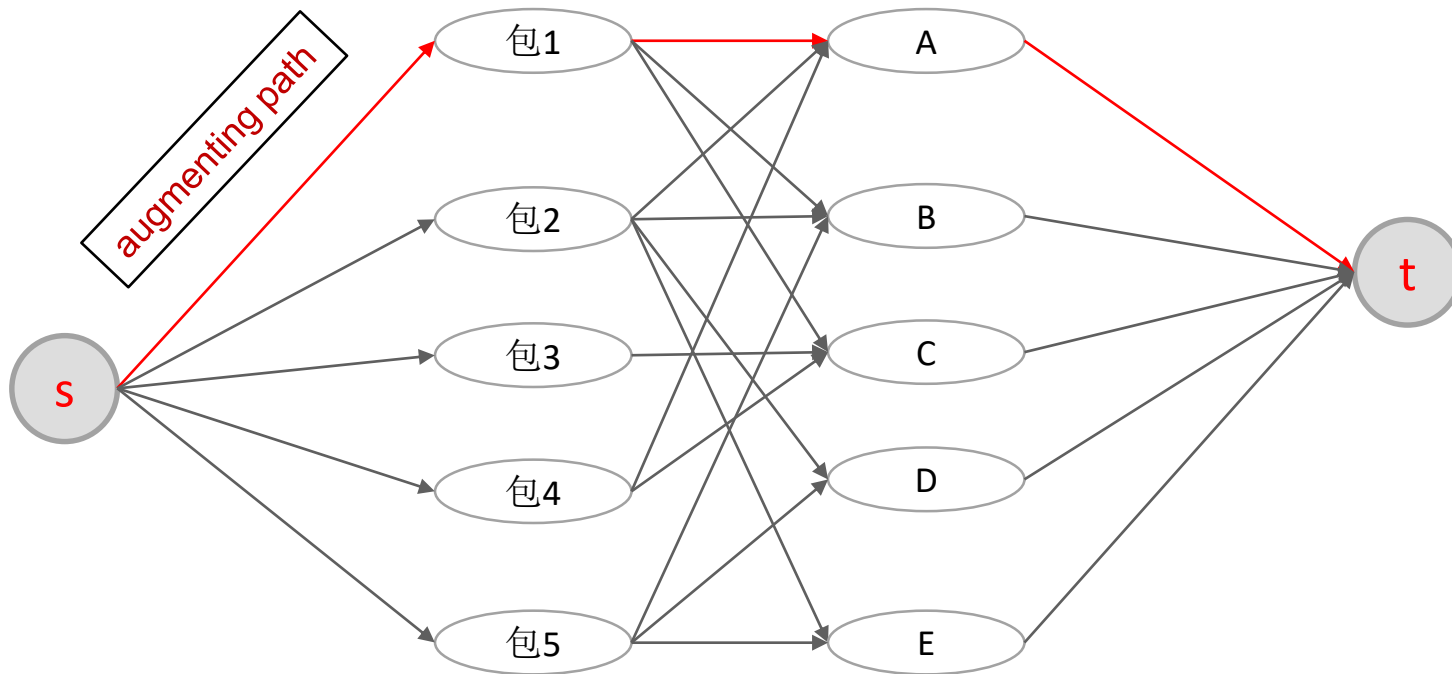
解题思路：（流量图中所有边的容量为1）





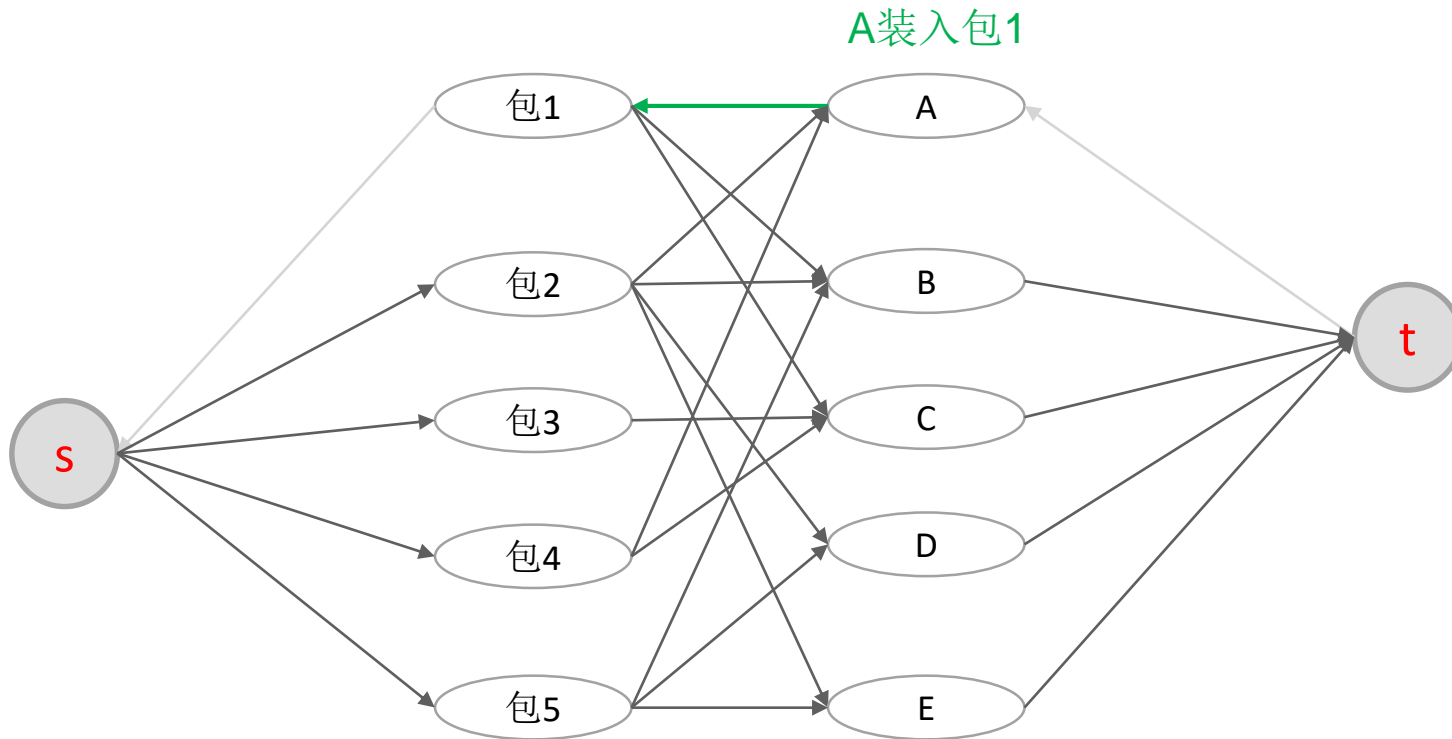
# Applications

解题思路：（流量图中所有边的容量为1）



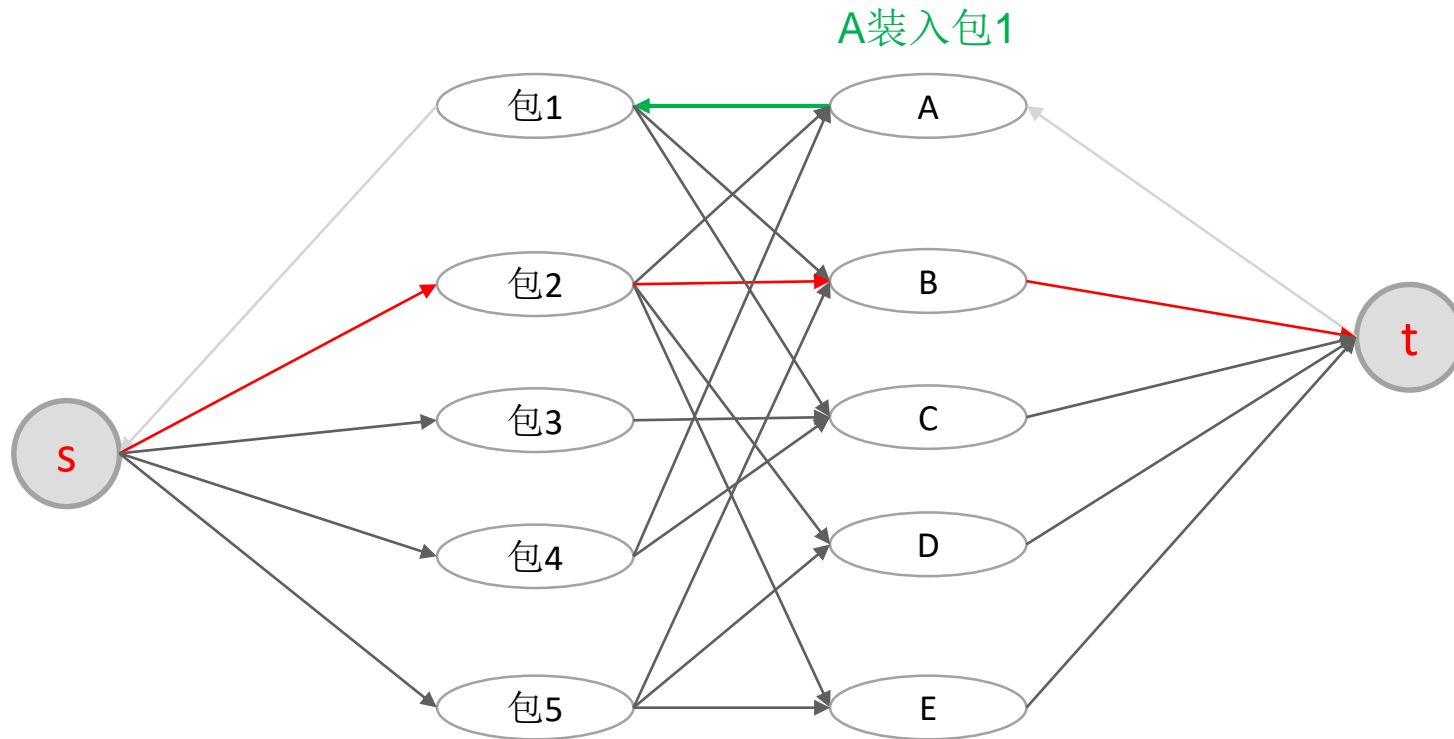
# Applications

解题思路：（流量图中所有边的容量为1）



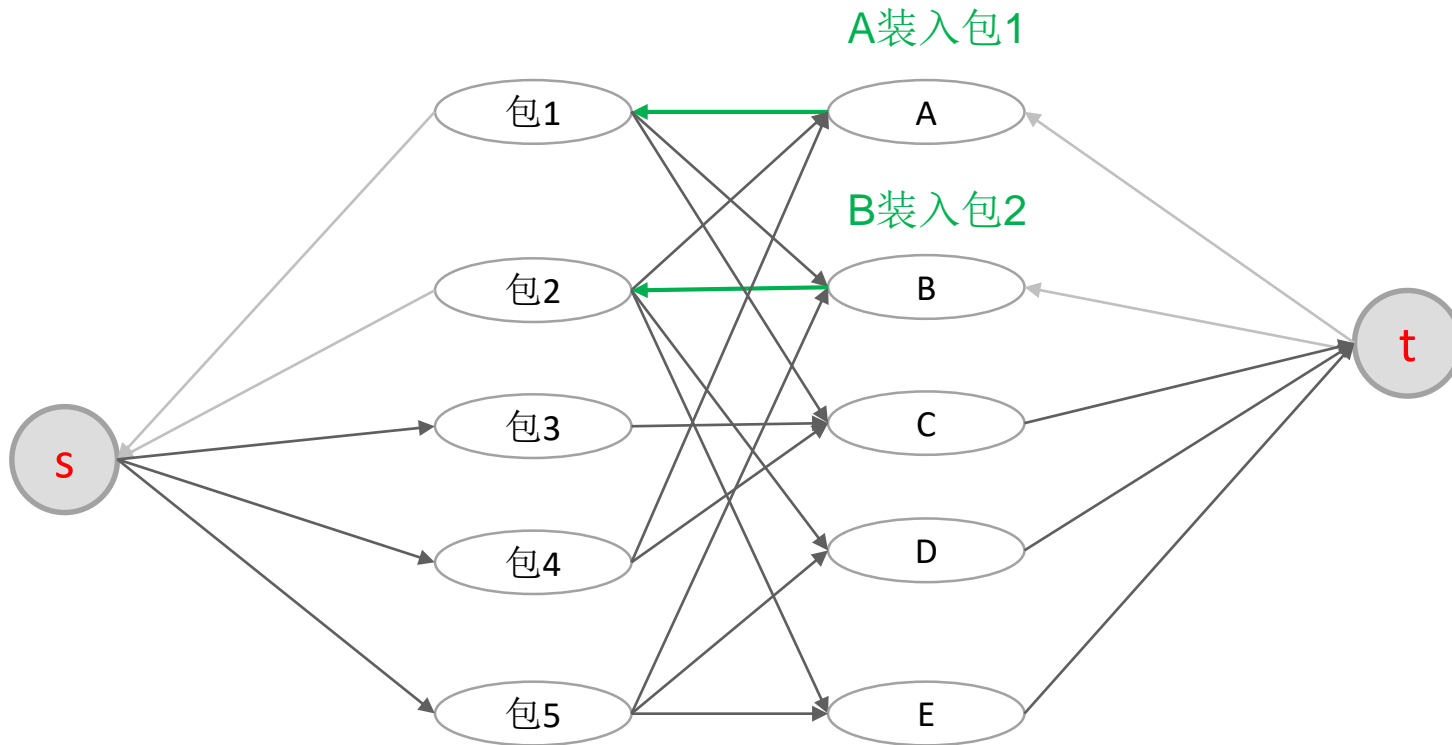
# Applications

解题思路：（流量图中所有边的容量为1）



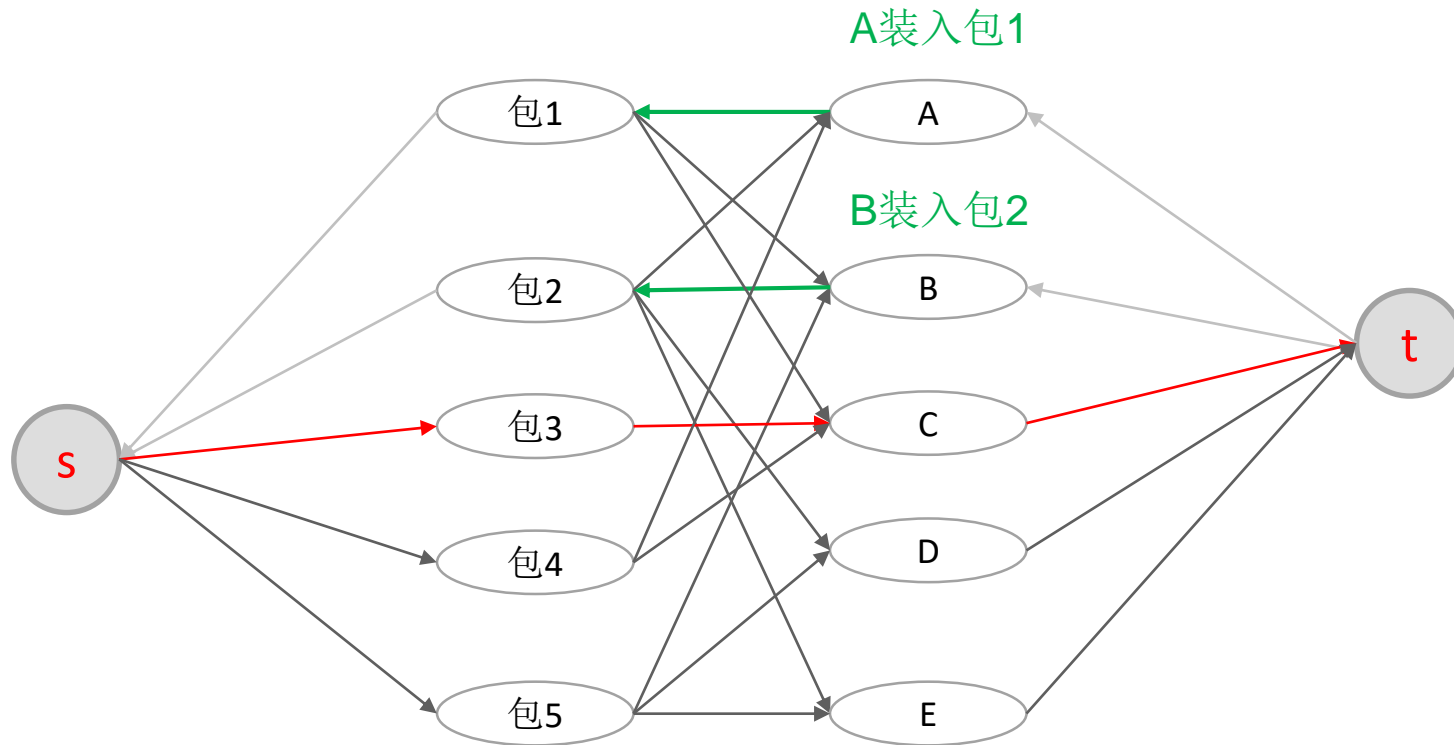
# Applications

解题思路：（流量图中所有边的容量为1）



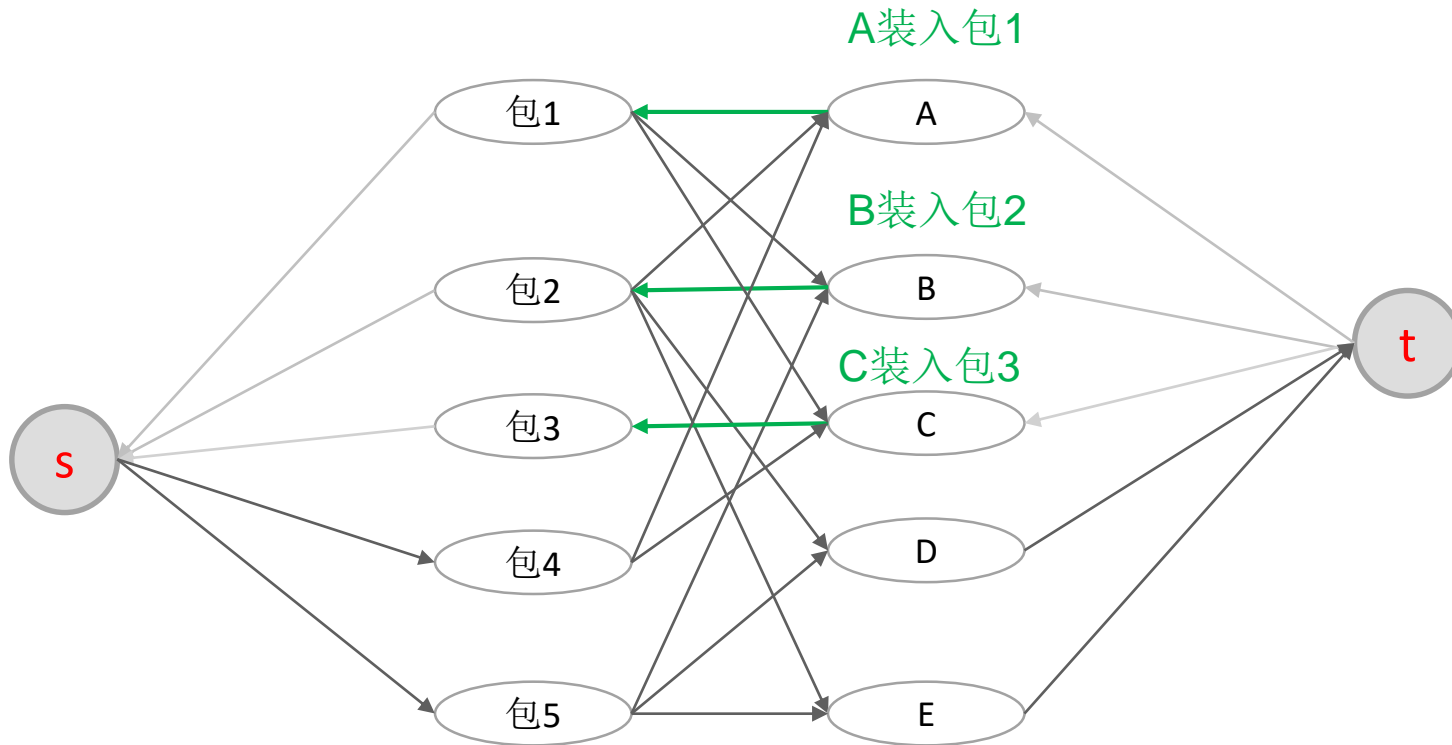
# Applications

解题思路：（流量图中所有边的容量为1）



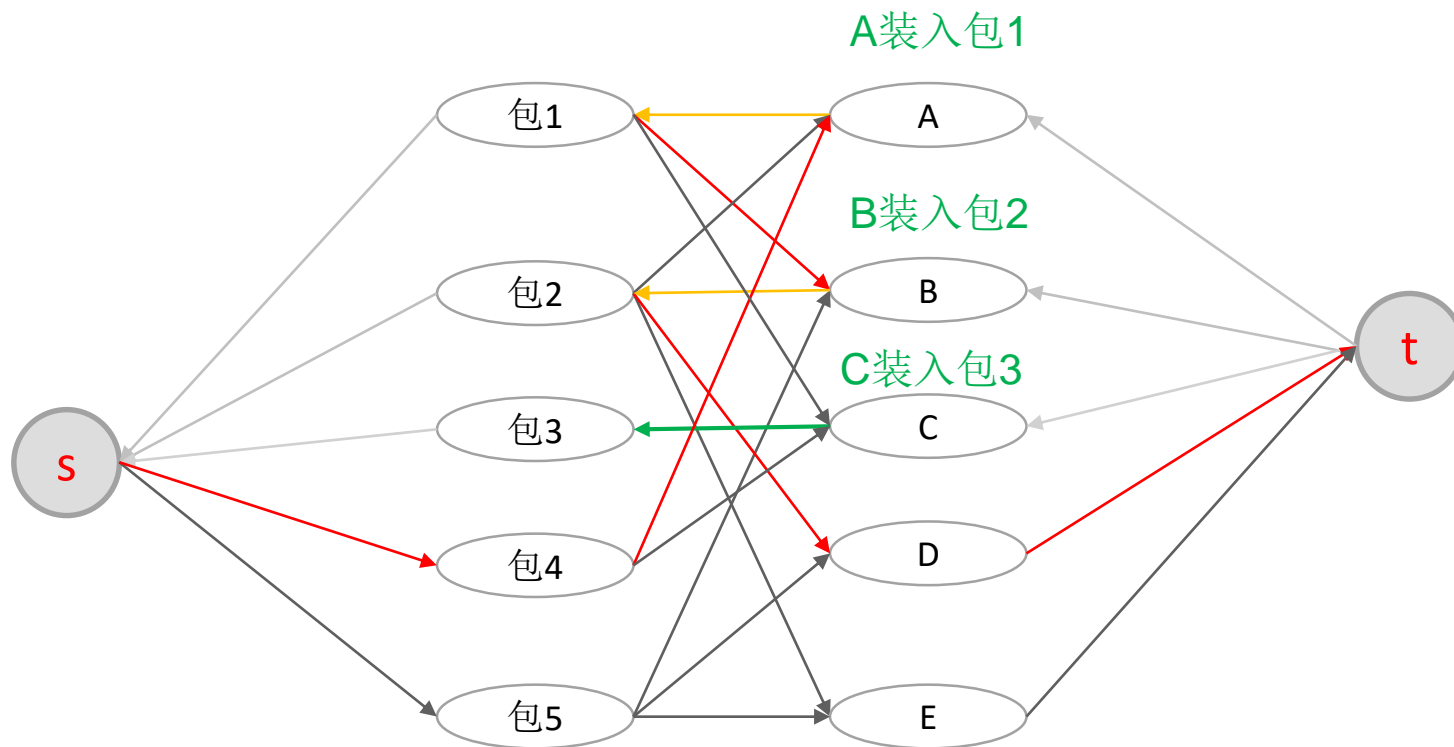
# Applications

解题思路：（流量图中所有边的容量为1）



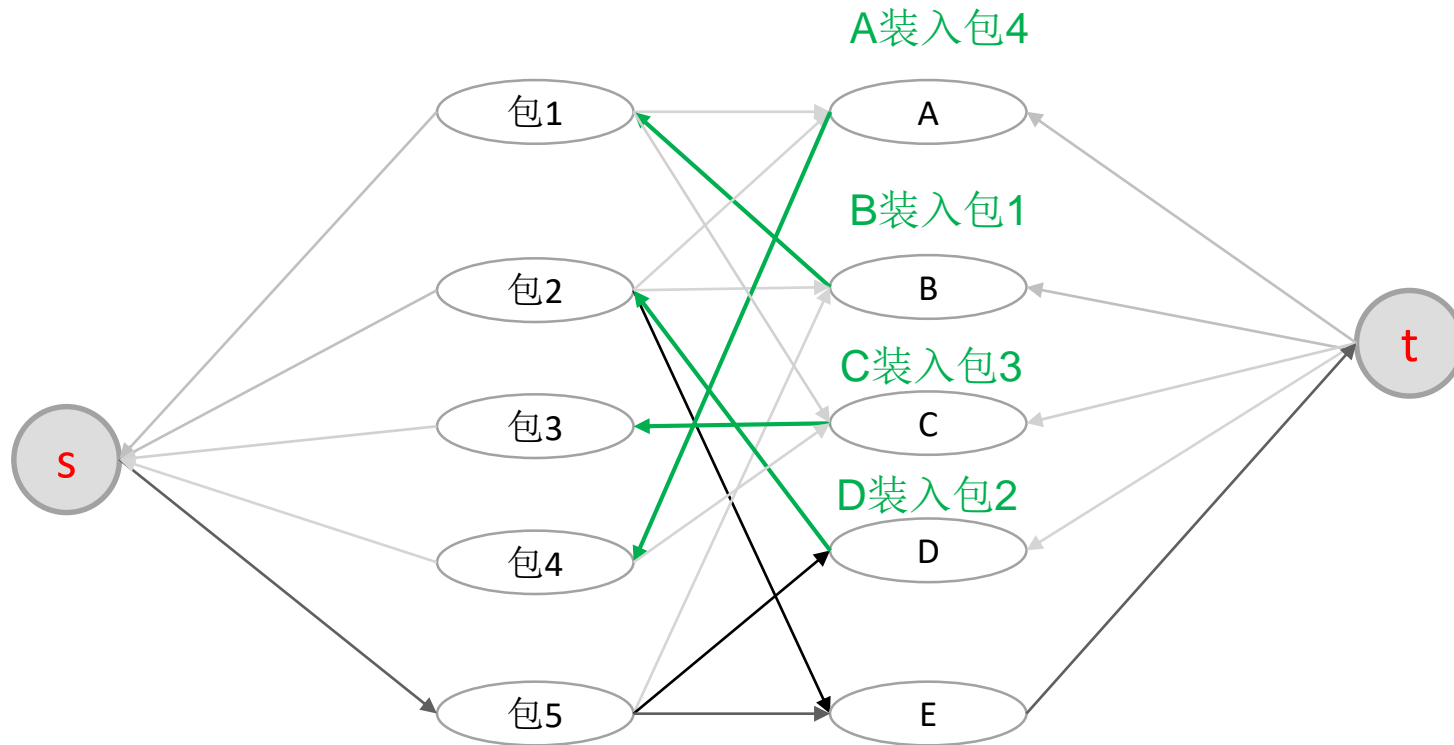
# Applications

解题思路：（流量图中所有边的容量为1）



# Applications

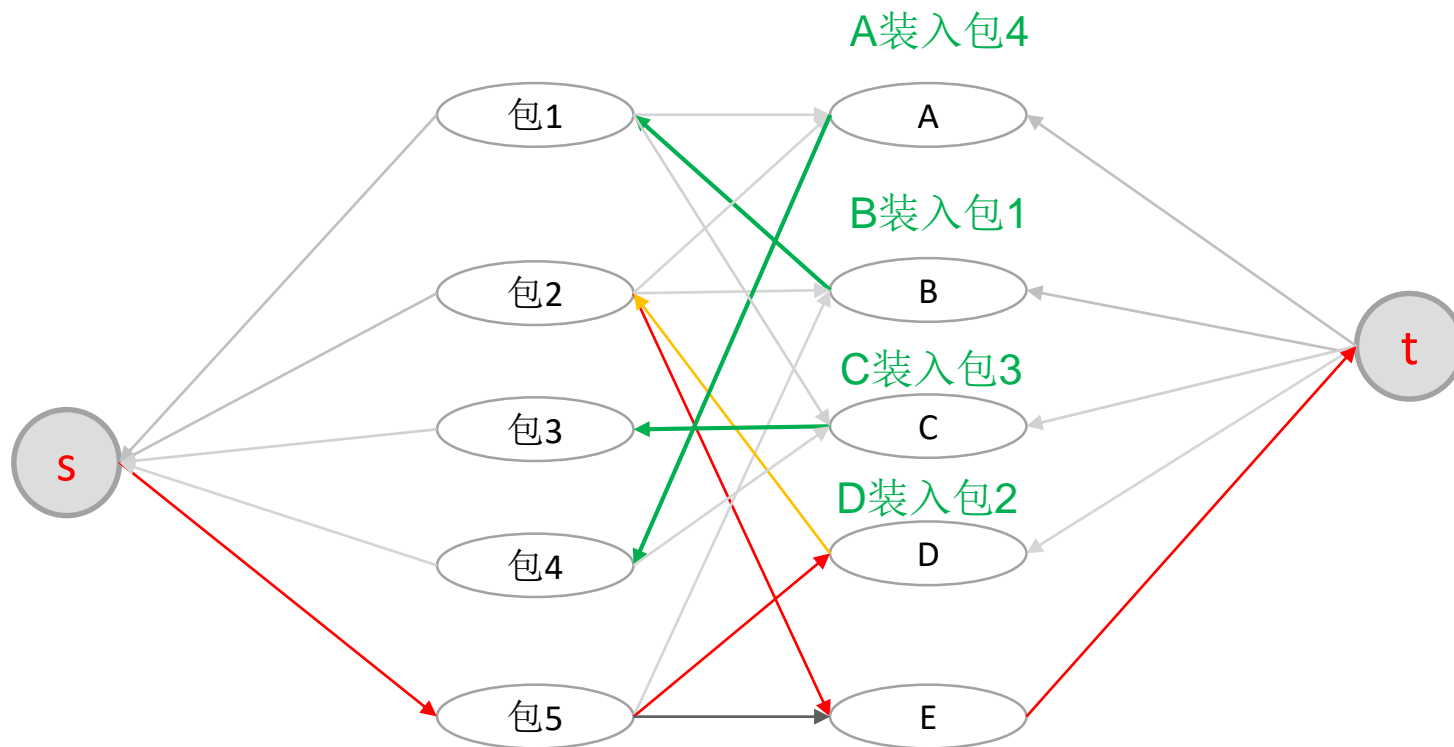
解题思路：（流量图中所有边的容量为1）





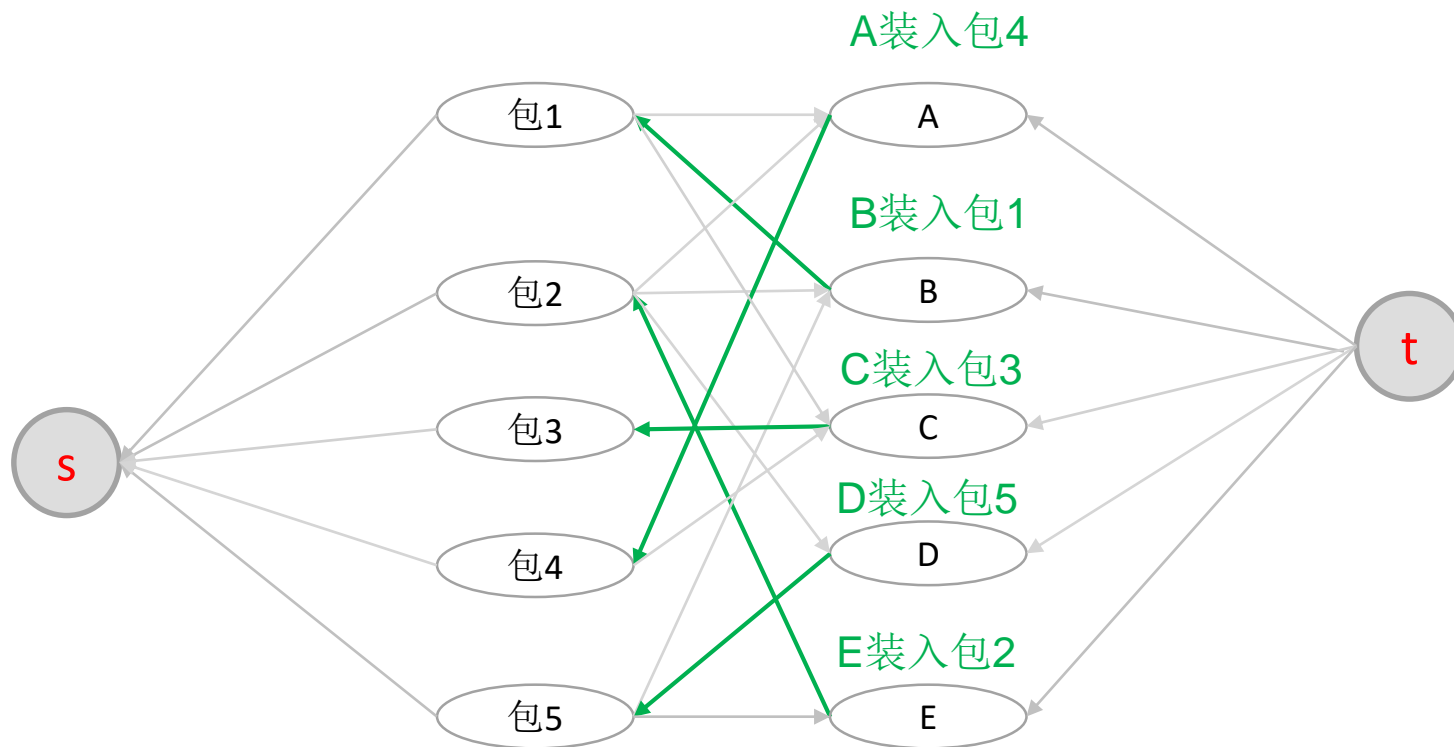
# Applications

解题思路：（流量图中所有边的容量为1）



# Applications

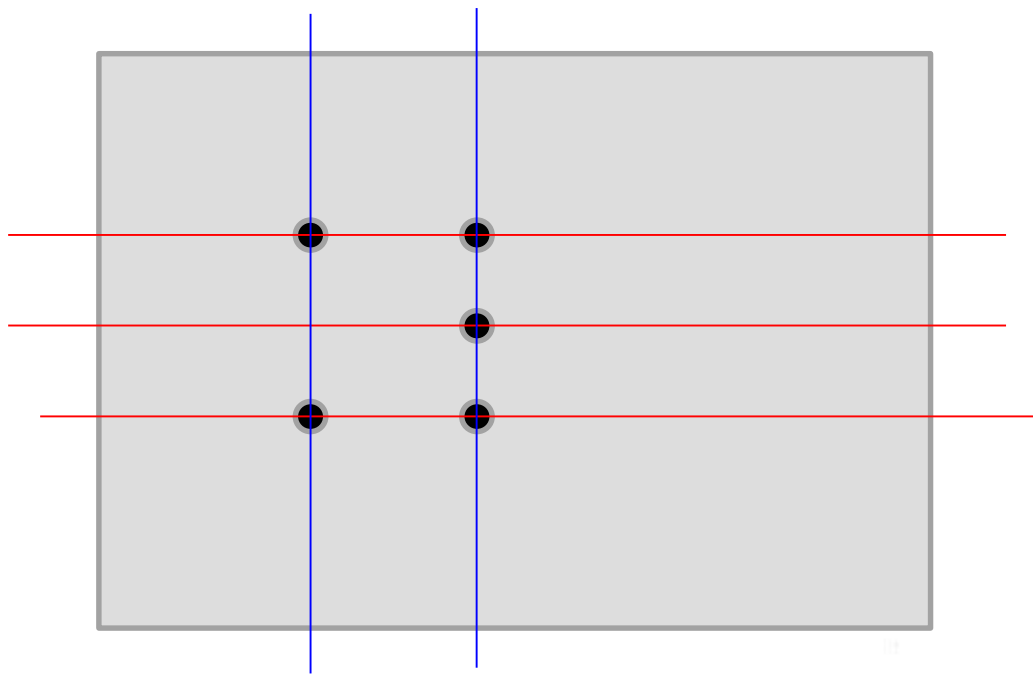
解题思路：（流量图中所有边的容量为1）



# Applications

## 线（点）覆盖问题

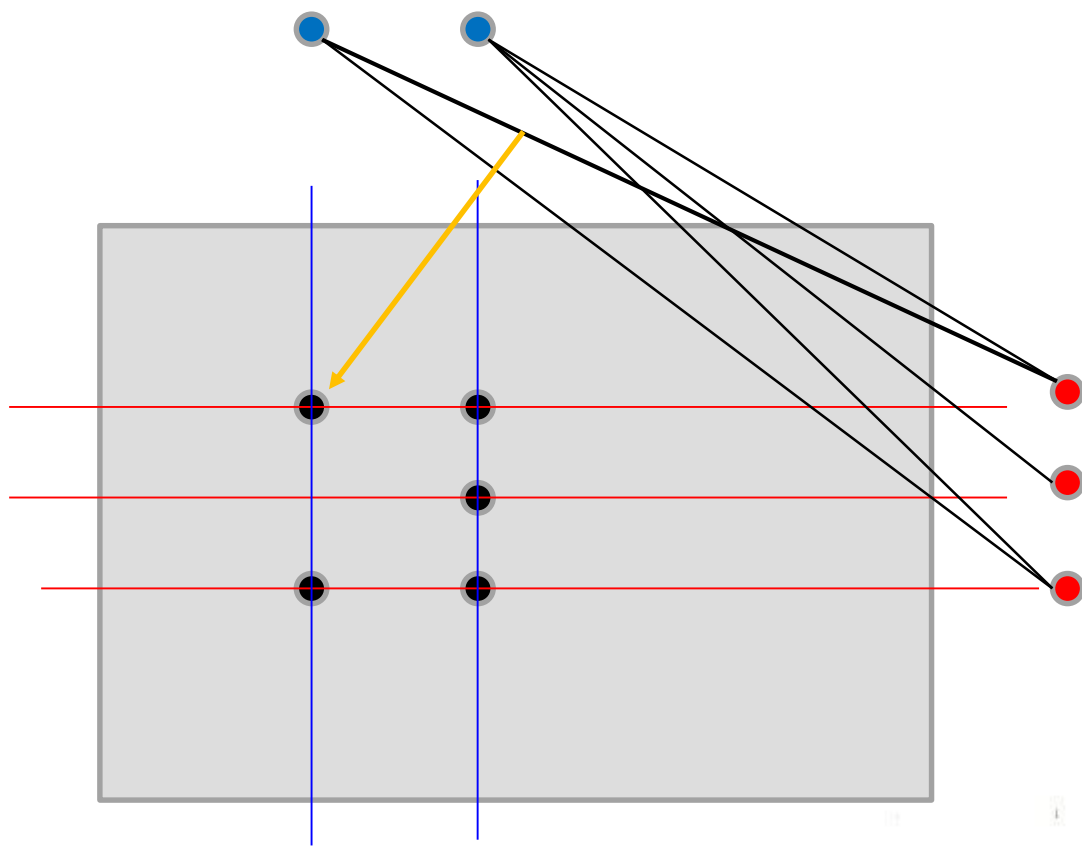
平面上有 $n$ 个点 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ 。通过在平面划横线或竖线把一个或多个点覆盖，如图。若要覆盖所有的点，最少可以划多少条线？



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。




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

最少划两条线



数据结构与算法课程组  
重庆大学计算机学院



# End of Section.

