# 最短通路问题

# 上节回顾

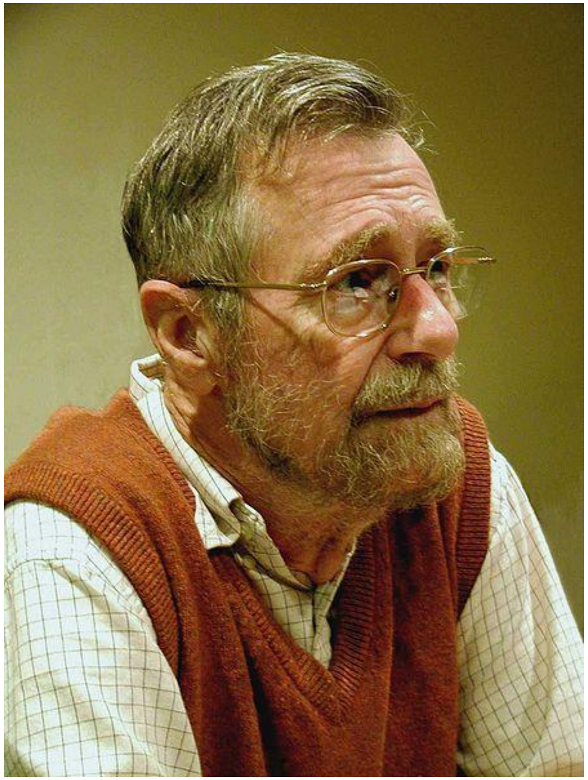
- 内容1：欧拉图
  - 什么是欧拉图：含有欧拉回路
  - 欧拉图的充要条件：所有顶点度数为偶数
  - 如何构造欧拉回路：Fleury算法
- 内容2：哈密尔顿图
  - 什么是汉密尔顿图：含有汉密尔顿回路
  - 哈密尔顿图的必要和充分条件：
    - 必要条件：$P(G-S) \leq |S|$，只能用来判断一个图不是汉密尔顿图
    - 充分条件：Ore定理，只能用来判断一个图是汉密尔顿图
  - 哈密尔顿图有哪些应用

# 本节提要

- 内容1：**Dijkstra**算法
- 内容2：**Floyd-Warshall**算法
- 内容3：旅行商问题（**TSP**）
- 内容4：最大流问题*

# Dijkstra算法





Named after its inventor Edsger Dijkstra (1930-2002)

Truly one of the "founders" of computer science

This is just one of his many contributions

# 带权图与最短通路问题

- 带权图：三元组 (*V*, *E*, *W*)，(*V*, *E*)是图，*W*是从E到非负实数集的一个函数。*W*(*e*)表示边*e*的权。
- 一条通路上所有边的权的和称为该通路的长度。
- 两点之间长度最小的通路称为两点之间的最短通路，不一定是唯一的。
- 单源点最短路问题

  给定带权图 *G*(*V*, *E*, *W*)并指定一个源点，确定该源点到图中其它任一顶点的最短路径。

# Dijkstra算法的思想

- 源点s到顶点v的最短路径若为s…uv，则s…u是s到u的最短路径
  - 反之，可由近及远的计算s到所有点的最短路径
  - (n-1)条最短路径按照由近及远（长度的非减次序）求得，设它们的相应端点分别为$u_1$, …$u_{n-1}$，最短路径长度记为$d(s, u_i)$， $i=1,…n-1$
- 每一步骤：选择最近的未知点并加入到已知点集合，更新s到其他未知点的距离
- 假设前i条最短路径已知，第(i+1)条最短路径长度：

$$d(s, u_{i+1})=\min\{d(s, u_j) +W(u_j, u_{i+1})\mid j=1,\ldots i\}$$

# Dijkstra算法的描述

- 输入：连通带权图G，$|V_G|=n$, 指定源顶点$s\in V_G$
- 输出：每个顶点$v$的标注$(L(v), u)$, 其中：
  - $L(v)$即从$s$到$v$的最短路径长度（目前可得的）
  - $u$是该路径上$v$前一个顶点。

1．初始化：$i=0$, $S=\{s\}$, $L(s)=0$, 对其它一切$v\in V_G$, 将$L(v)$ 置为$\infty$。若$n=1$，结束。

2．$\forall v\in S'=V_G-S$, 比较$L(v)$和$L(u_i)+W(u_i, v)$的值 $(u_i\in S)$

如果$L(u_i)+W(u_i, v)<L(v)$, 则将$v$的标注更新为$(L(u_i)+W(u_i, v), u_i)$，

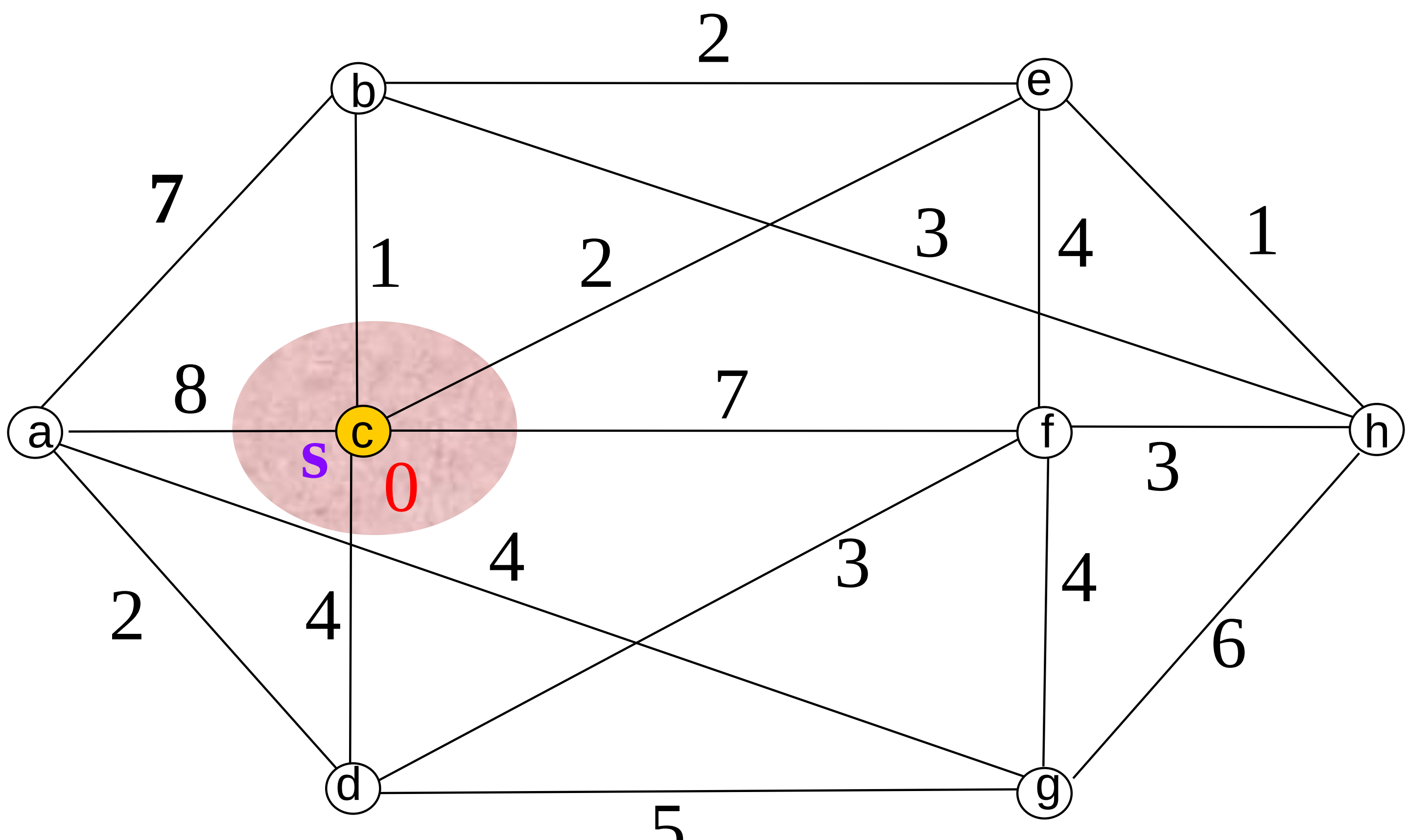
即：$L(v)=\min\{ L(v), \min_{u\in S_i}\{L(u)+W(u,v)\} \}$

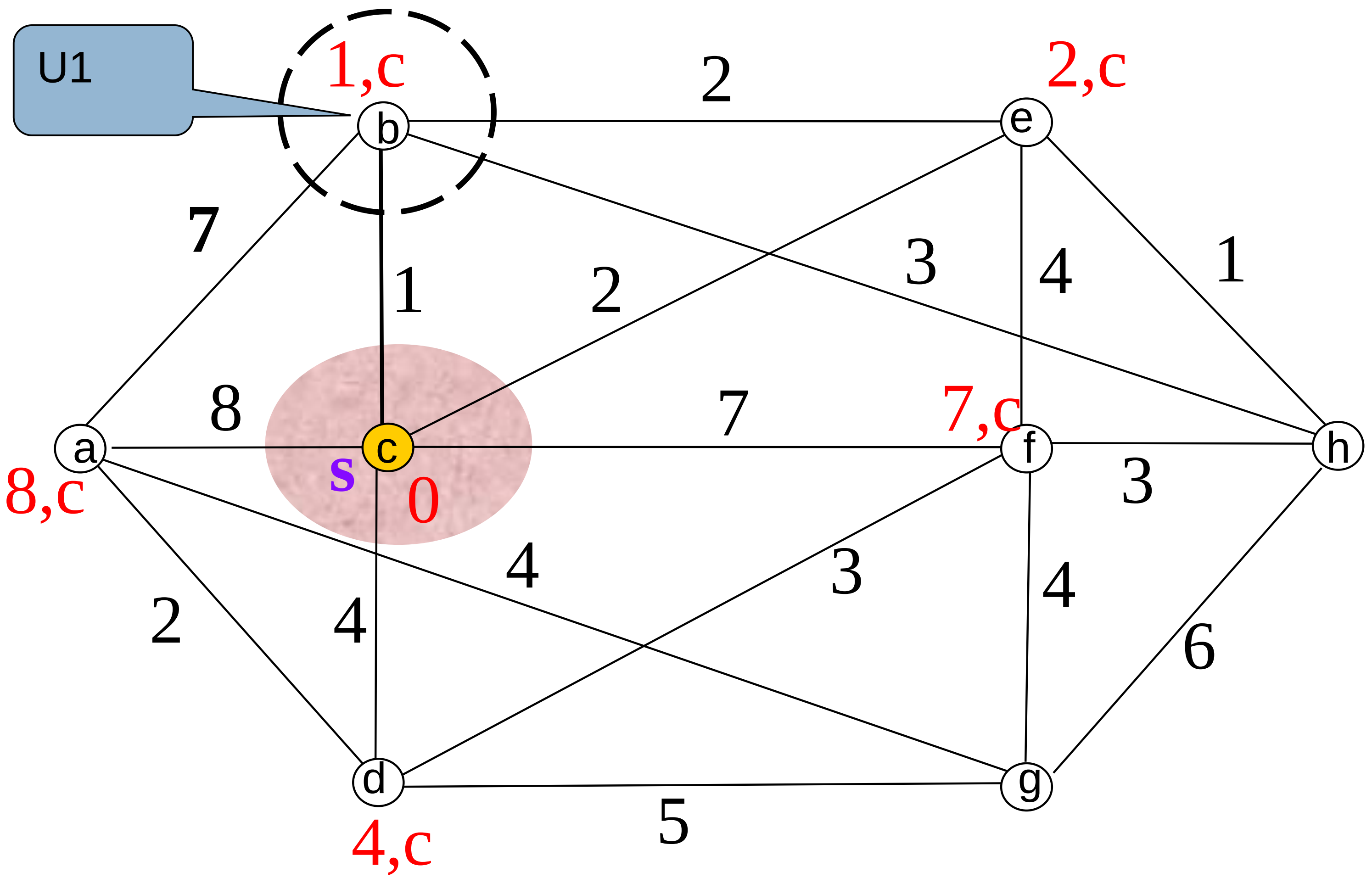
3. 对所有$S'$中的顶点，找出具有最小$L(v)$的顶点$v$, 作为$u_{i+1}$

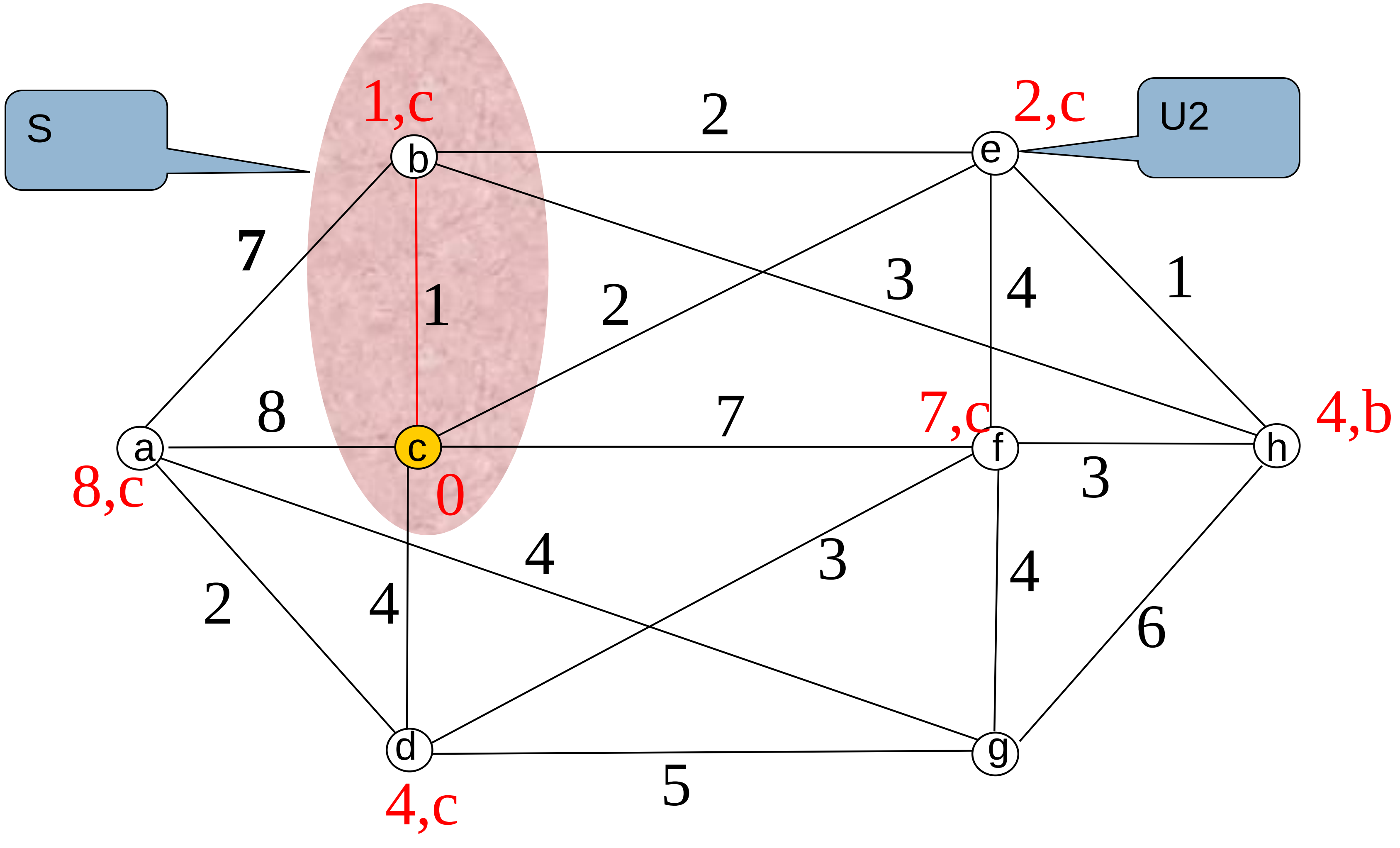
4．$S = S\cup\{u_{i+1}\}$

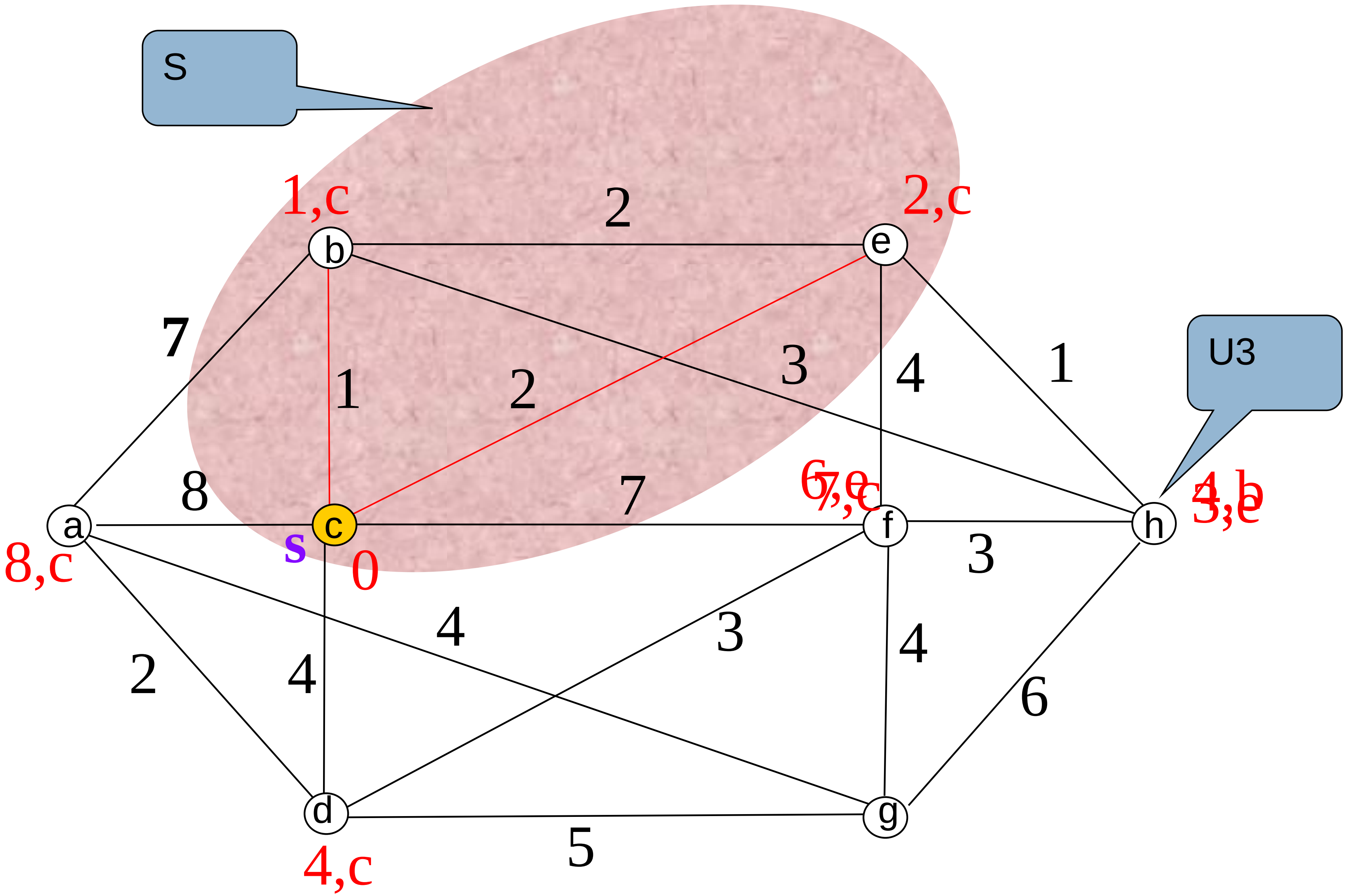
5. $i = i+1$; 若$i=n-1$, 终止。否则：转到第2步。

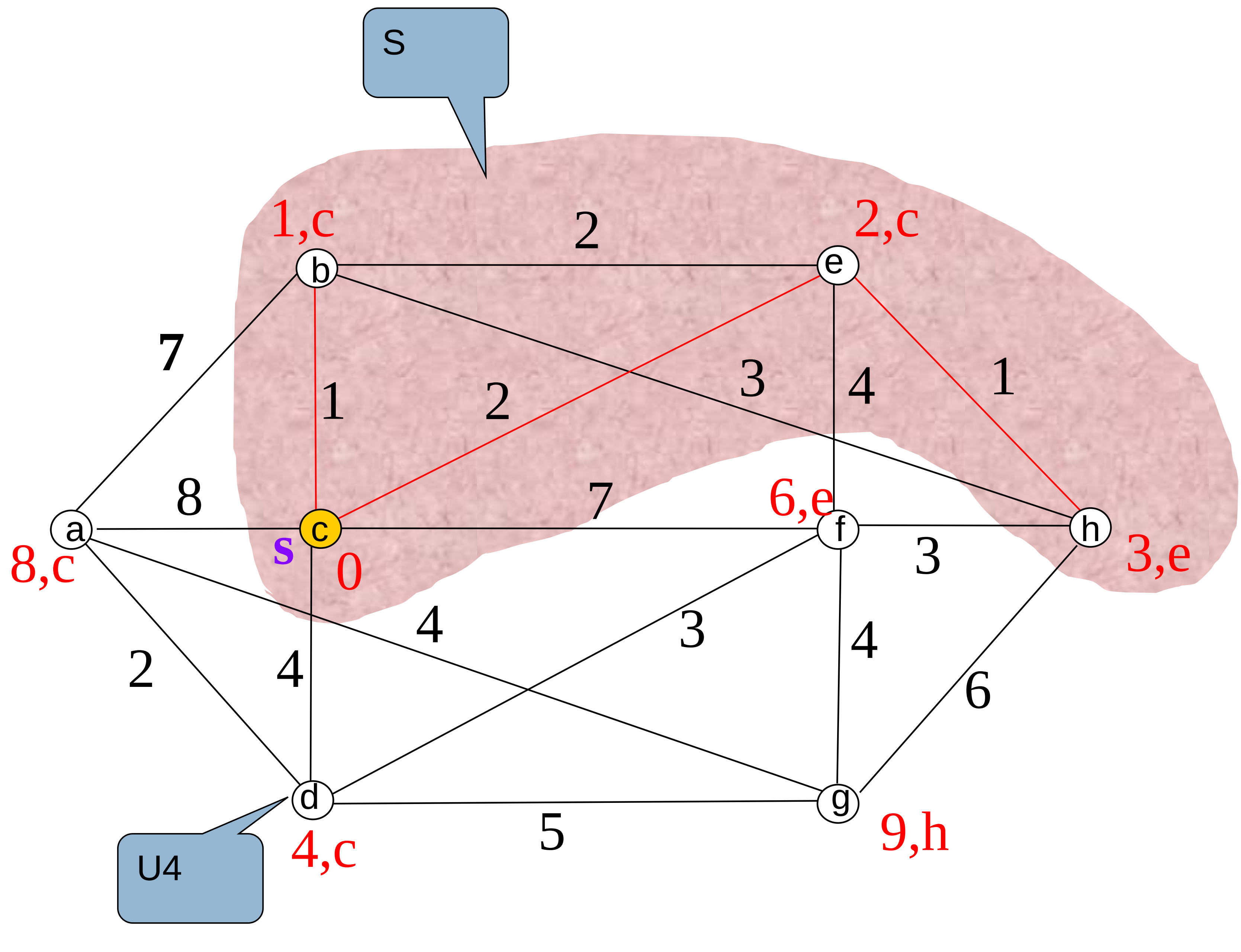
# 求最短路的一个例子

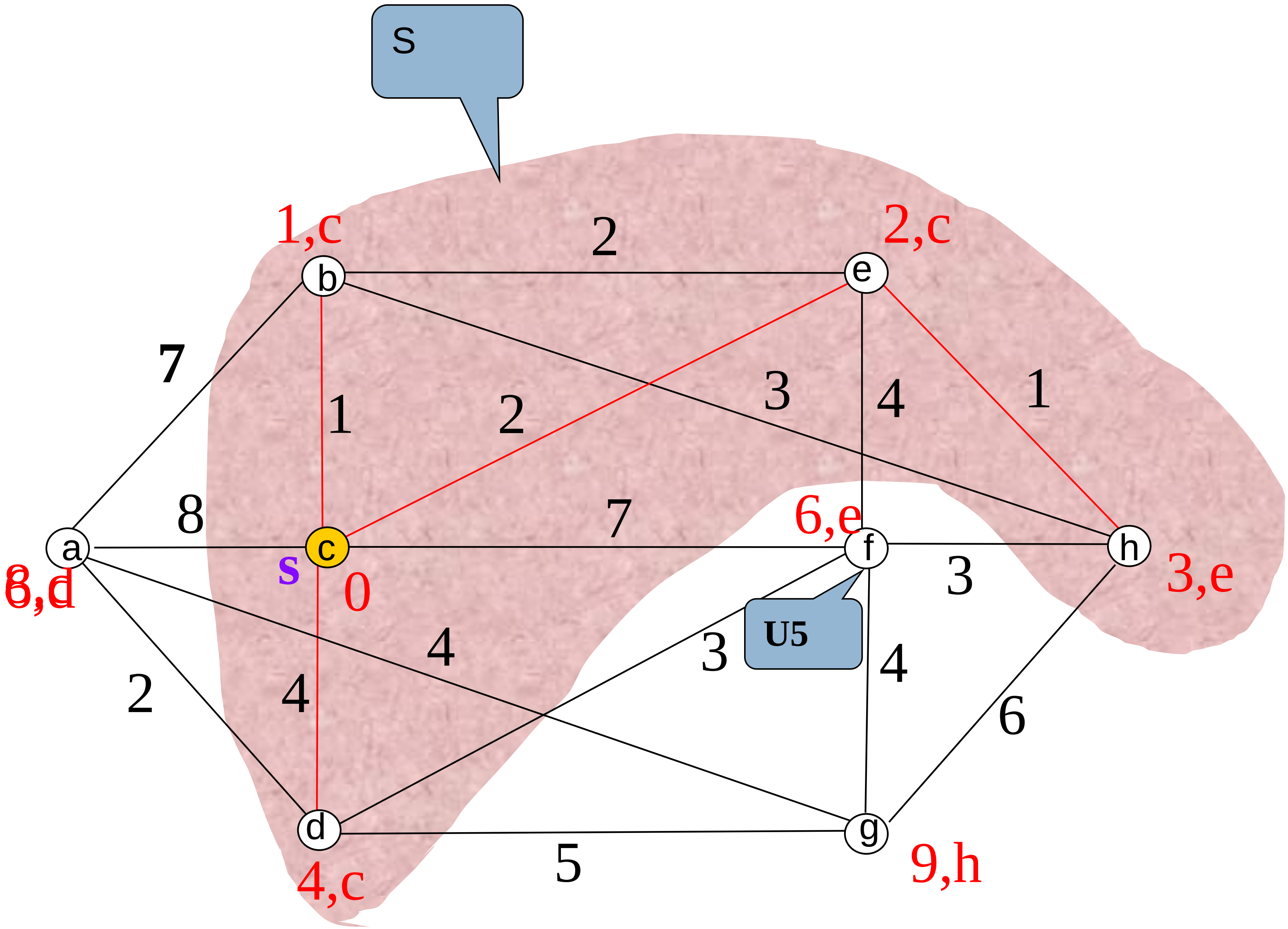


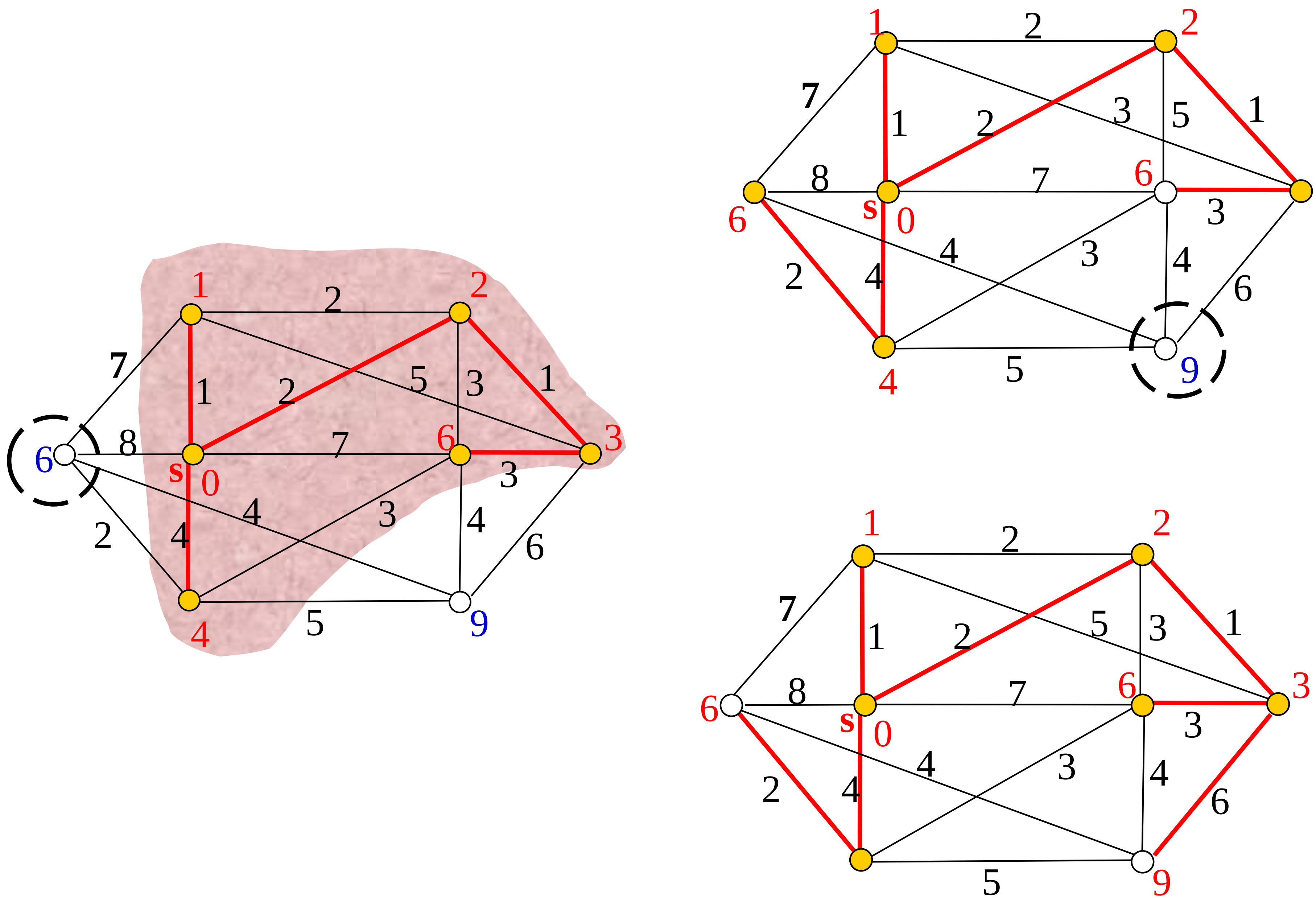

U1
1,c
2
2,c
b
e
7
1
2
3
4
1
8
7
7,c
a
c
f
h
s
0
8,c
3
4
3
4
2
4
6
d
g
5
4,c


S
U2
1,c
2,c
8,c
0
7,c
4,b
4,c
a
b
c
d
e
f
g
h
2
7
1
2
3
4
1
8
7
3
2
4
4
3
4
6
5


S
U3
1,c
2,c
2
7
1
2
3
4
1
8
7
6,e
7,c
4,b
3,e
a
b
c
d
e
f
g
h
s
0
8,c
3
4
3
4
2
4
6
5
4,c


S
1,c
2,c
2
b
e
7
1
2
3
4
1
8
7
6,e
a
c
f
h
s
0
8,c
3,e
3
4
3
4
2
4
6
d
g
5
U4
4,c
9,h


S
1,c
2,c
a
b
c
d
e
f
g
h
s
0
6,e
3,e
4,c
9,h
U5
2
7
1
2
3
4
1
8
7
3
4
3
4
2
4
6
5



1
2
2
7
1
2
5
3
1
6
8
7
6
3
s
0
3
4
3
4
2
4
6
5
4
9
1
2
2
7
1
2
3
5
1
8
7
6
s
0
3
6
2
4
4
3
4
6
5
4
9
1
2
2
7
1
2
5
3
1
6
8
7
6
3
s
0
3
4
3
4
2
4
6
5
9

# Dijkstra算法的分析

- 可终止性
  - 计数控制
- 正确性

  需证明当算法终止时

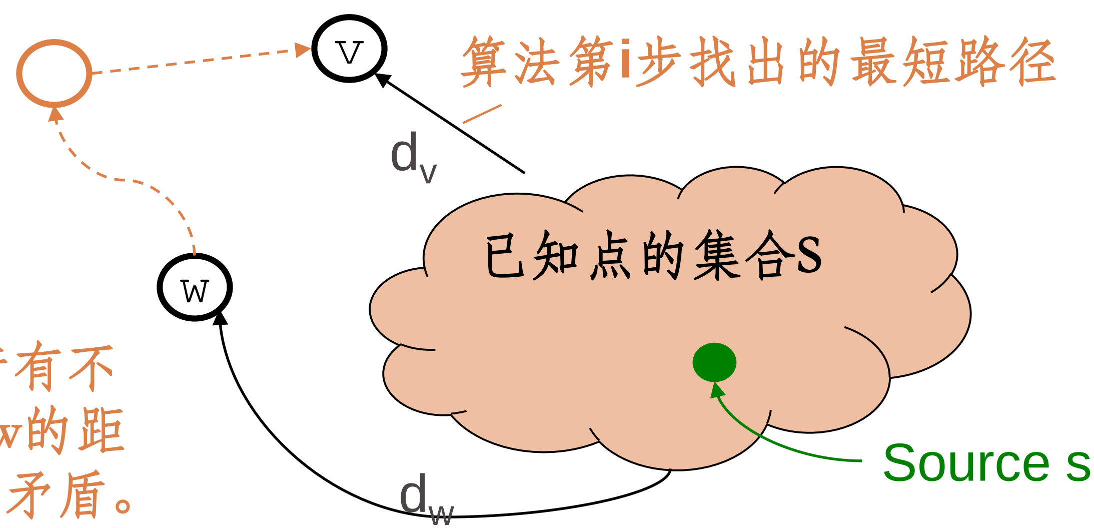
  - $L(v)=d(s, v)$对一切$v$成立。
  - 由标记中的诸$u_i$确定的路径是一条最短路径

    (这里$d(s,v)$是$s$到$v$的最短路径长度，即距离。)
- 复杂性
  - $O(n^2)$：每次迭代可能需要更新所有点（S’中的点）
  - O(|E| lg |V|)：通过binary heap

# Dijkstra算法的分析（正确性）

- 本质是一个**greedy**算法：
  - 每一步找当前的最优解，不会调整已经得到的结果
  - 但是一般情况下**local**最优并不一定是**global**最优
- 巧的是**Dijkstra**算法就是**global**最优
  - 只需证明第**i**步将$u_i$标记为已知时，一定是s到$u_i$的**global**最优解

能否存在s到v
更近的路径?




不能。因为存在就意味着有不在S中的点w。此时，s到w的距离一定小于s到v的距离，矛盾。

# 本节提要

- 内容1：**Dijkstra算法**
- 内容2：**Floyd-Warshall算法**
- 内容3：旅行商问题（**TSP**）
- 内容4：最大流问题*

# 求所有结点间的最短距离?

- Dijkstra算法
  - 不能处理负边
  - **O(|V| |E| lg |V|) with binary heap — O(|V|$^3$ lg |V|) if dense**
- Floyd-Warshall 算法
  - 可以处理负边，只要没有负的回路
  - **O(|V|$^3$)**，无需fancy的数据结构
  - 动态规划算法：存储并利用子问题的解

# 所有点队的最短路问题

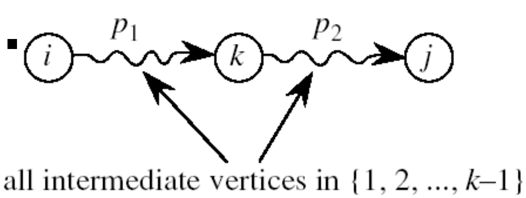
- 输入：给定带权有向图 $G(V, E, W)$ 。
  - W将边映射到一个实数
  - 假设没有负回路（**negative weight cycles** ）
- 输出: **n × n**矩阵包含了所有的最短距离**δ(u, v).**

# Floyd-Warshall算法

- 动态规划方法：**Use optimal substructure of shortest paths:** ***Any subpath of a shortest path is a shortest path.***
- 构建一个三维矩阵:
  - 定义$d_{ij}^{(k)}$：中间节点在{1,2, …, k}中的从节点i到节点j的最短距离
  - 最终目的是要计算$d_{ij}^{(n)}$

# 计算$d_{ij}^{(k)}$

- 当k=0: $d_{ij}^{(0)} = w_{ij}$.
- 当k>0:令$p=<v_i, . . . , v_j>$为从节点i到节点j的最短路径，其中间节点在{1,2, …, k}中.
  - 情况1：如果节点k不是中间节点，则所有中间节点在{1,2, …, k-1}中.
  - 情况2：如果节点k是中间节点，则p可以分解成两个最短路径，其中间节点均在{1,2, …, k-1}中.




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

# 算法描述

$\text{FLOYD-WARSHALL}(W, n)$

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

**for** $k \leftarrow 1$ **to** $n$

  **do for** $i \leftarrow 1$ **to** $n$

    **do for** $j \leftarrow 1$ **to** $n$

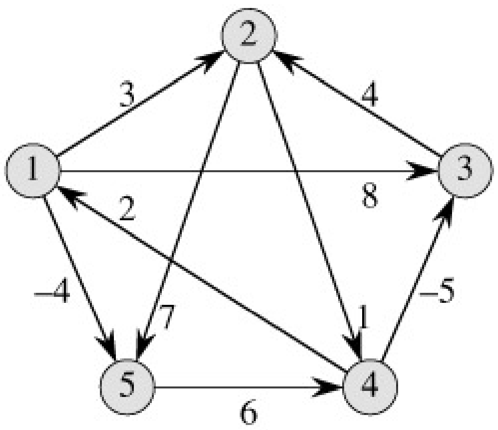
      **do** $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$

**return** $D^{(n)}$

- 时间复杂度：**O($n^3$).**
- 空间复杂度：**O($n^2$) (if we drop the superscripts).**

# 例

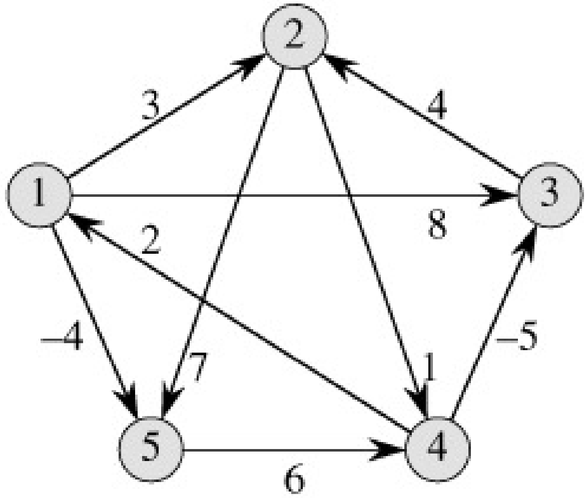
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0\,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \ge 1\,. \end{cases}$$




$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

# Step 1

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

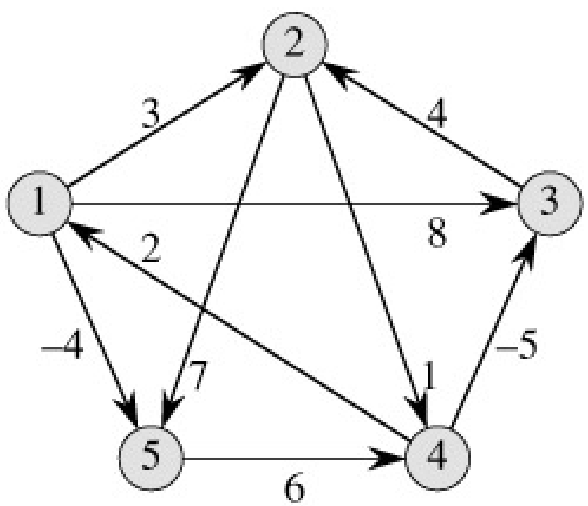



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

# Step 2

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

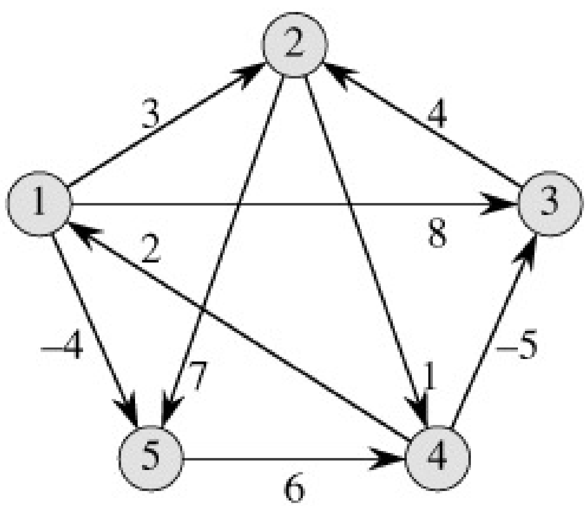



$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

# Step 3

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

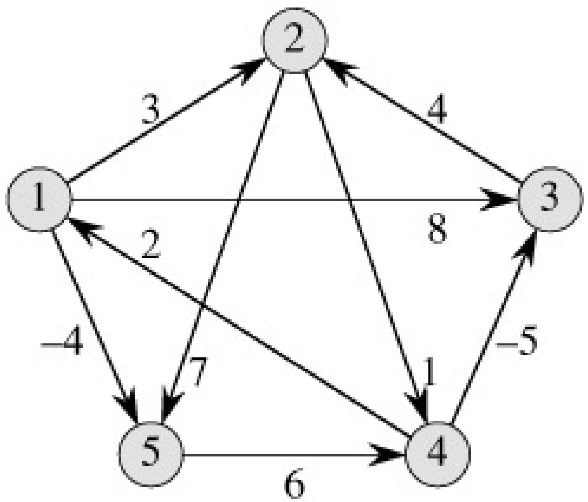



$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

# Step 4

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

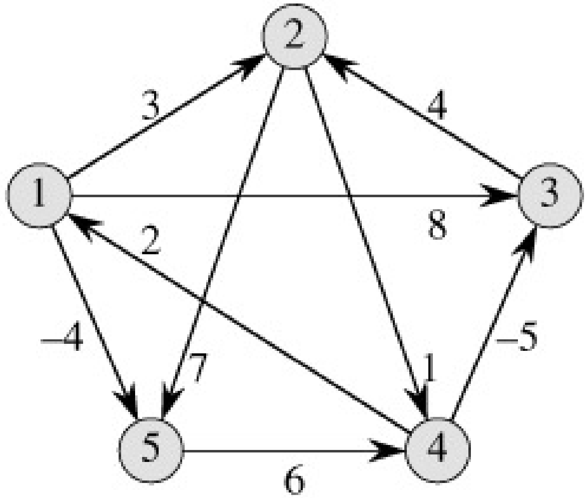



$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

# Step 5

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




$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

# 本节提要

- 内容1：**Dijkstra**算法
- 内容2：**Floyd-Warshall**算法
- 内容3：旅行商问题（**TSP**）
- 内容4：最大流问题*

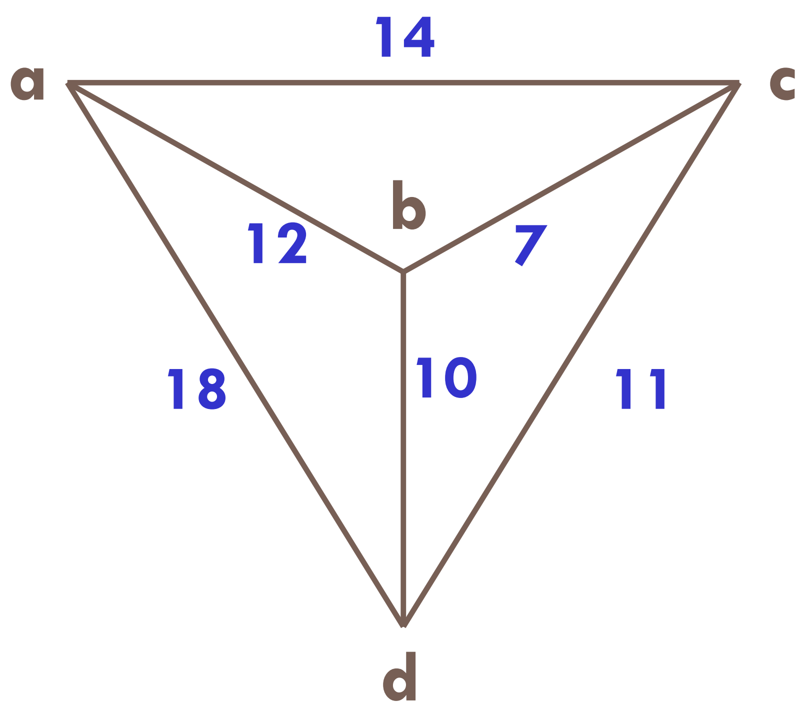
# 旅行商问题
# (Travelling Salesman Problem, TSP )

- **n**个城市间均有道路，但距离不等，旅行商从某地出发，走过其它**n-1**城市各一次，最后回到原地，如何选择最短路线?
- 数学模型:
  - 无向带权图**G**：顶点对应于城市，边对应于城市之间的道路，道路长度用相应边的权表示。
  - 问题的解：权最小的哈密尔顿回路。
  - **G**是带权完全图，总共有**(n-1)!/2**条哈密尔顿回路。因此，问题是如何从这**(n-1)!/2**条中找出最短的一条。

**(含25个顶点的完全图中不同的哈密尔顿回路有约3.1×$10^{23}$条，若机械地检查，每秒处理$10^9$条，需1千万年。)**

# 旅行商问题

- 一个货郎（销售员）生活在城市a，假定访问的城市是d, b, c，然后回到a，求完成这次访问的最短路径的距离.



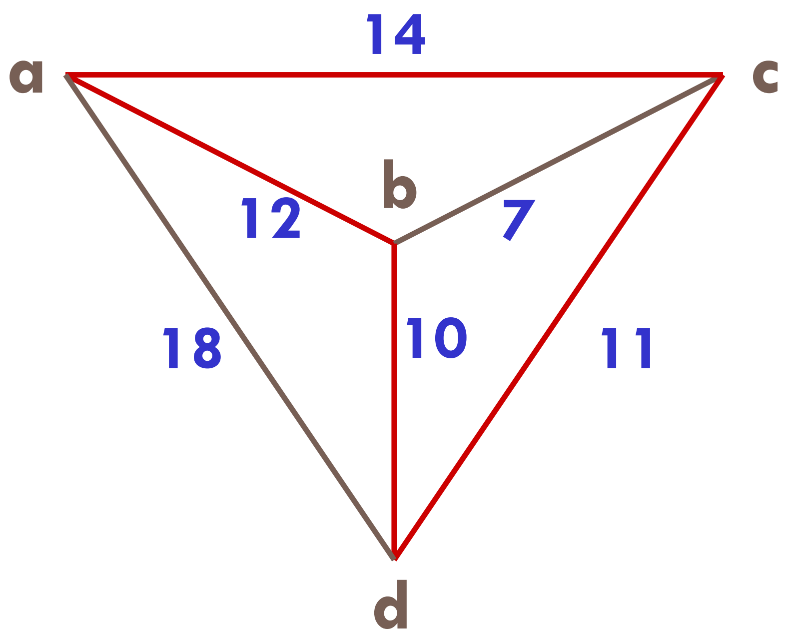
# 旅行商问题

- 解：列出哈密尔顿回路，并求其距离：

（1）（abcda）=（12+7+11+18）= 48

（2）（acbda）=（14+7+10+18）= 49

（3）（abdca）=（12+10+11+14）= 47



# 旅行商问题

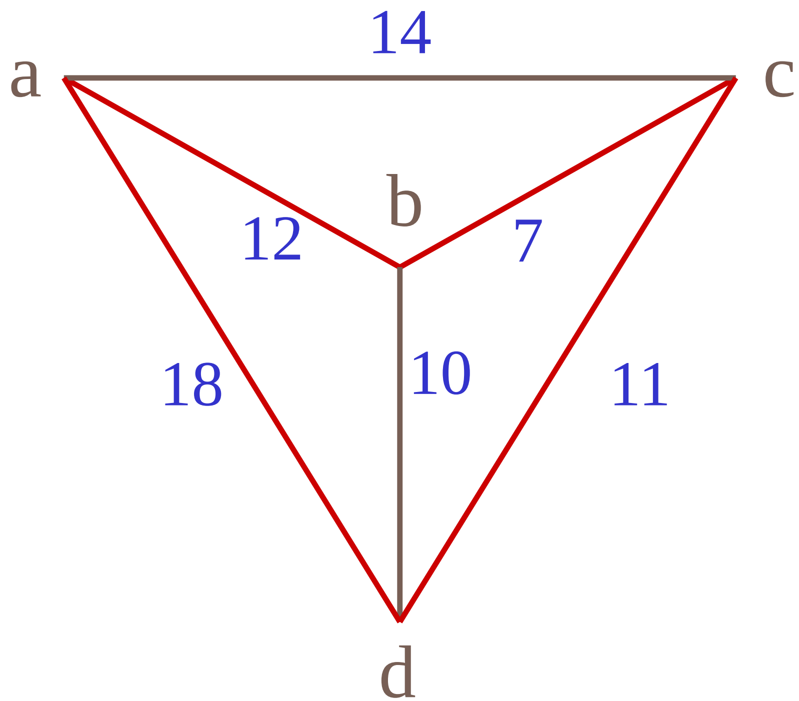
- 哈密尔顿回路（路径）的最短路径问题
- 下面介绍一种最邻近算法：

（1）选择任一顶点作为始点，找出离始点距离最小的顶点，形成一条边的初始路径；

（2）设u是最新加到这条路径上的顶点，从不在这条路径上的所有顶点中选择一个与u距离最小的顶点，把连接u与此结点的边加入路径中；重复执行直到G中的各顶点均含在这条路径中。

# 旅行商问题

（3）把始点到最后加入的顶点的边放入路径中得到一条哈密尔顿回路，并为近似最短的哈密尔顿回路.



# 旅行商问题(TSP)的研究进展

- （在最坏情况下）时间复杂性为多项式的算法?
- （在最坏情况下）时间复杂性为多项式的近似算法
  - 保证: $W \le W' \le cW$ $(c=3/2)$, 误差为50 %
- 实际应用中，已有好的算法能够在几分钟内处理**1000**个节点的规模，误差在**2%**

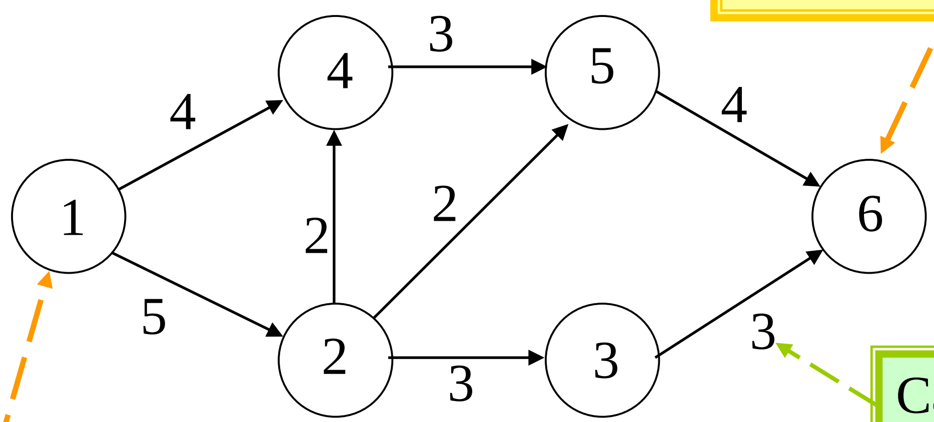
# 本节提要

- 内容1：**Dijkstra算法**
- 内容2：**Floyd-Warshall算法**
- 内容3：旅行商问题（**TSP**）
- 内容4：最大流问题*

# 传输网(Transport network)



The unique node with out-degree 0
The **sink**




Capacity of edge, $C_{i,j}$

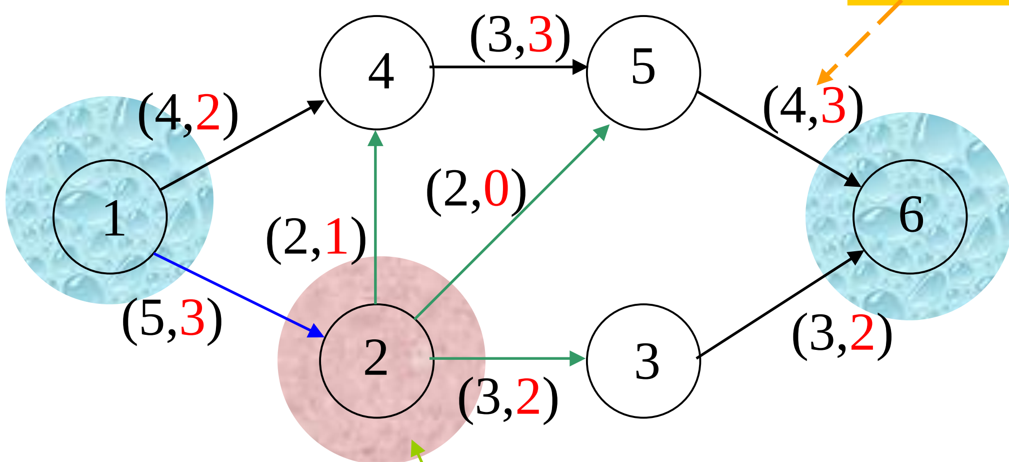
The unique node with in-degree 0
The **source**

It is assumed that all edges are in one direction.

# 流(Flow)



$(C_{i,j}, F_{i,j})$,
For any edge, $F_{i,j} \leq C_{i,j}$




**Value of the flow is 5**

Conservation of flow:
Here: 3=1+0+2

# 流(Flow)



- Let $(G,k)$ be a transport network with source $S$ and sink $D$. Assume the capacity function $k$ is defined on the edges of $G$. A flow in $G$ is a nonnegative real-valued function $F$ defined on the edges of $G$ such that:
  - [Capacity constraint] $0 \leq F(e) \leq k(e)$ for each edge $e \in E(G)$
  - [Conversation equation]

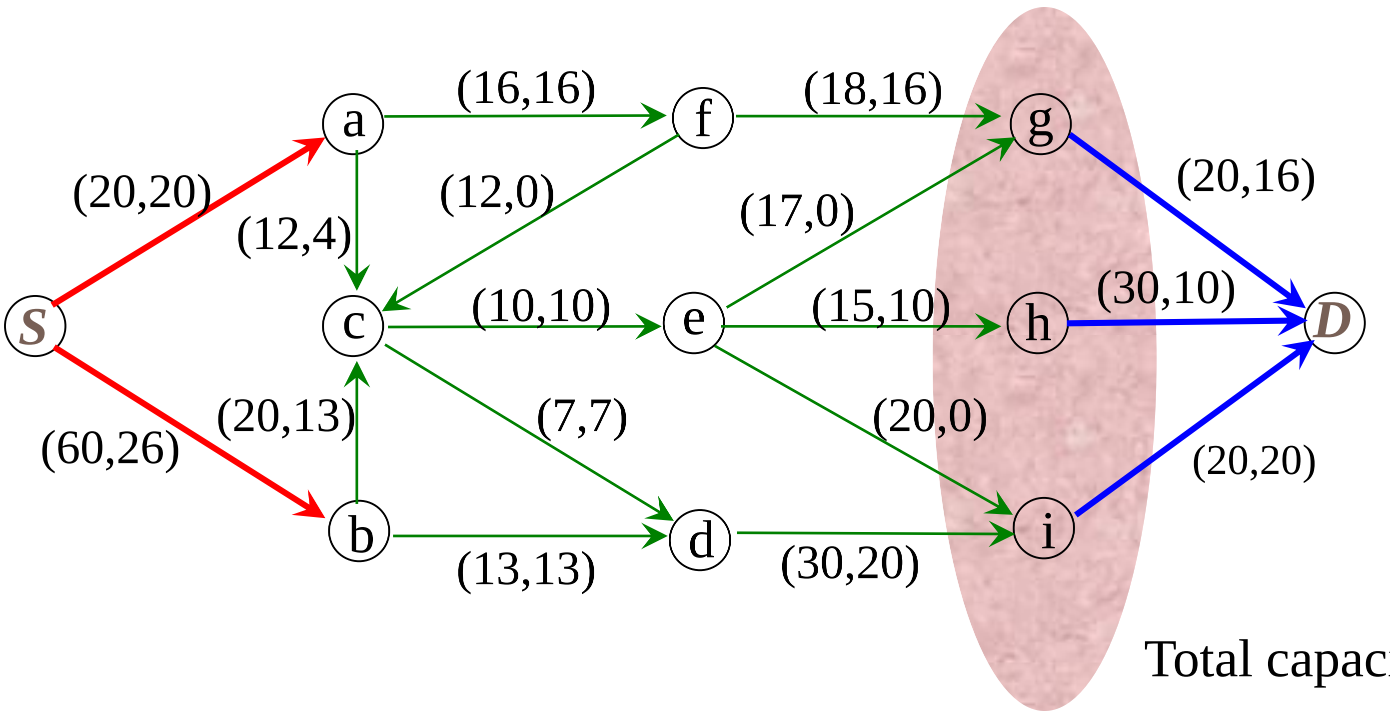
$$\sum_{y \in A(x)} F(xy) = \sum_{z \in B(x)} F(zx) \text{ for every } x \in V(G) - \{S, D\}$$

where $A(x) = \{y \mid xy \in E(G)\}$, $B(x) = \{z \mid zx \in E(G)\}$

- The value of a flow $F$ (denoted as $|F|$) is defined as the value of $F(S, V_G)$ (or, $F(V_G, D)$)

# 例






Total capacity: 70

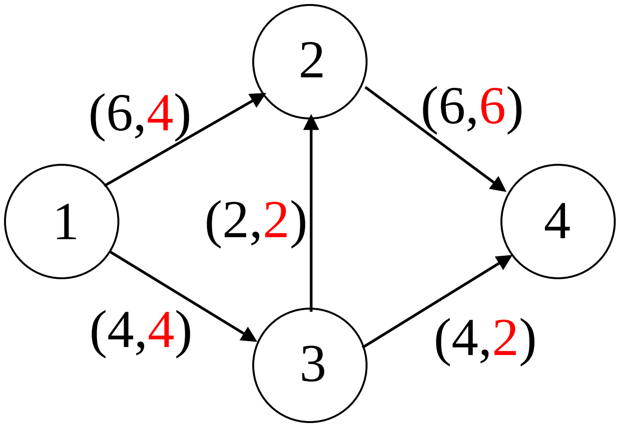
Actual receipt: 46

**The problem: Can we send more on the network?**

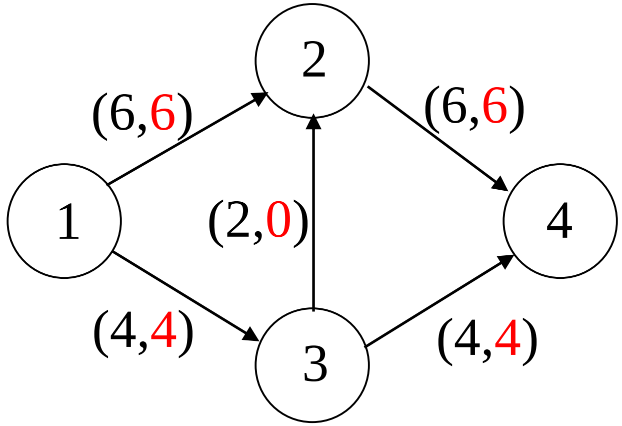
# 最大流



- A flow $F$ in a network $(G,k)$ is a maximum flow if $|F| \geq |F'|$ for every flow $F'$ in $(G,k)$




Value of flow: 8

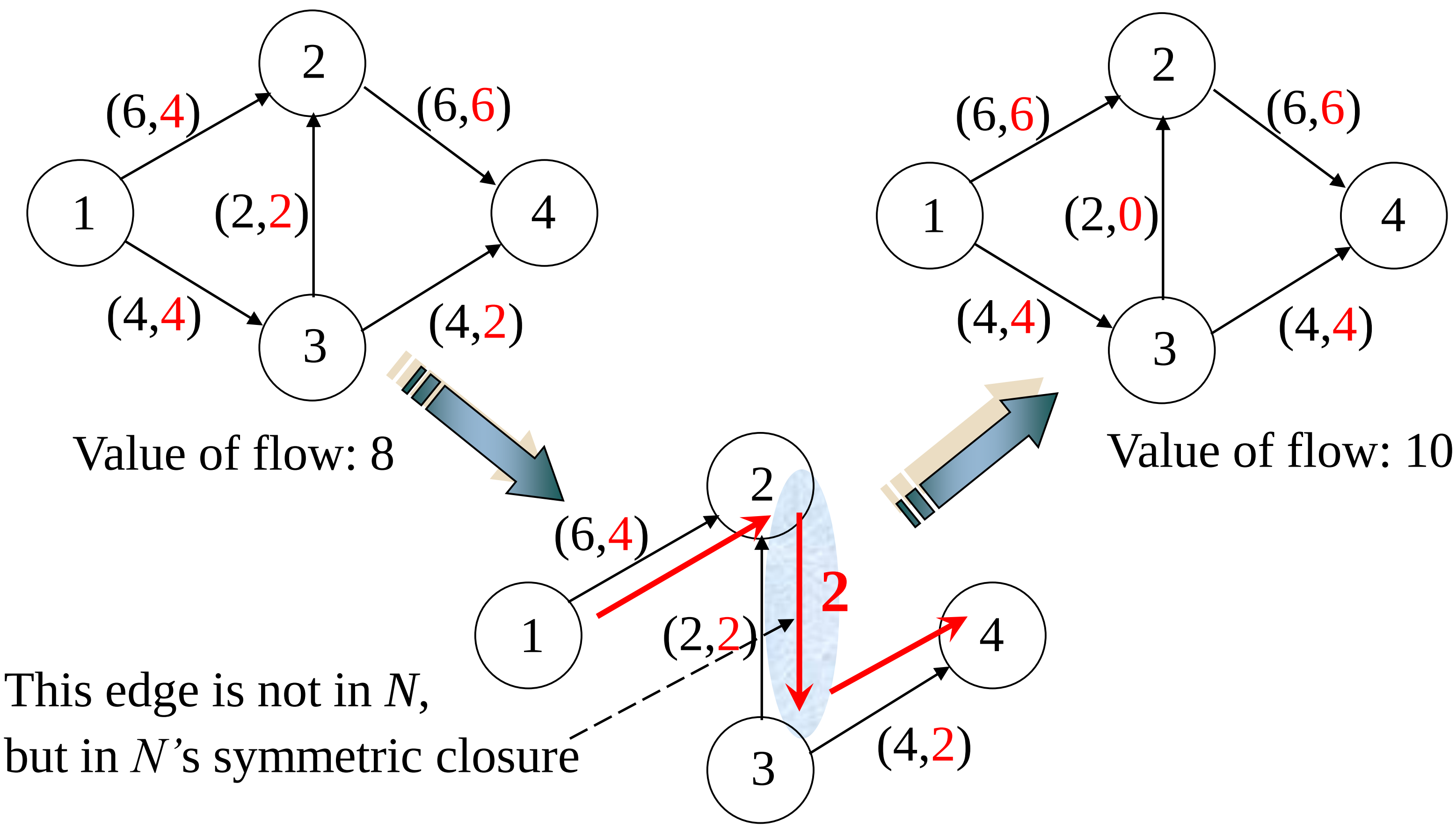



Value of flow: 10

Basic Problems: (1) Largest value of flow?
(2) A flow with the largest value?

# 最大流




2
(6,4)
(6,6)
1
(2,2)
4
(4,4)
3
(4,2)
Value of flow: 8
2
(6,6)
(6,6)
1
(2,0)
4
(4,4)
3
(4,4)
Value of flow: 10
2
(6,4)
2
1
(2,2)
4
3
(4,2)
This edge is not in *N*,
but in *N*'s symmetric closure

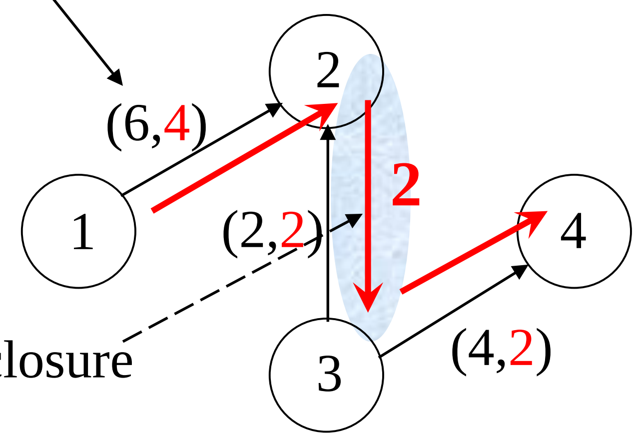
# 最大流：容量过剩



π:1 2 3 4 is not a path in *N*, but in *G*, the symmetric closure.

(1,2) is in *N*, this edge has an excess capacity *2* (=6-4)

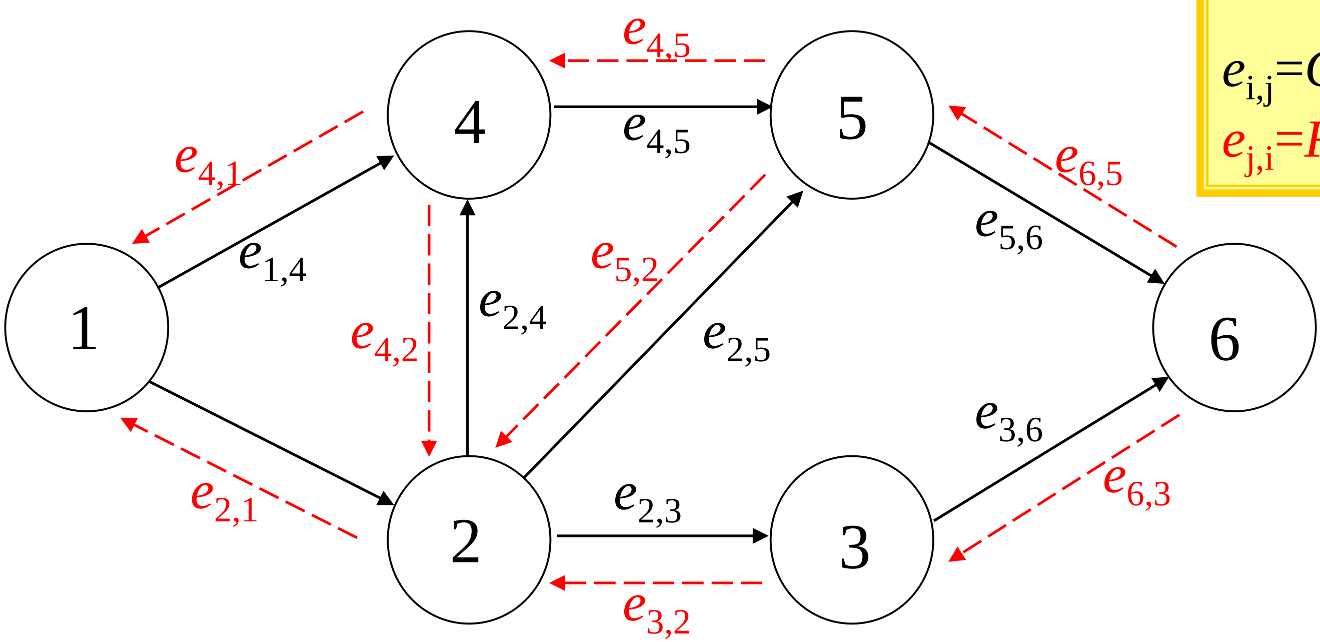
(2,3) is not in *N*, this edge has an excess capacity *2*




This edge is not in *N*,
but in *N*'s symmetric closure

# 最大流：容量过剩






Excess capacity:

$e_{i,j}=C_{i,j}-F_{i,j}$

$e_{j,i}=F_{i,j}$ if $F_{i,j}>0$

$C_{i,j}$ is the capacity of edge $(i,j)$

$F_{i,j}$ is the flow on edge $(i,j)$

edges in $N$

edges in s($N$), but not in $N$

# Labeling Algorithm (Ford & Fulkson)



Initialization: set all flow to 0

Step 1: (1) Identify $N_1$

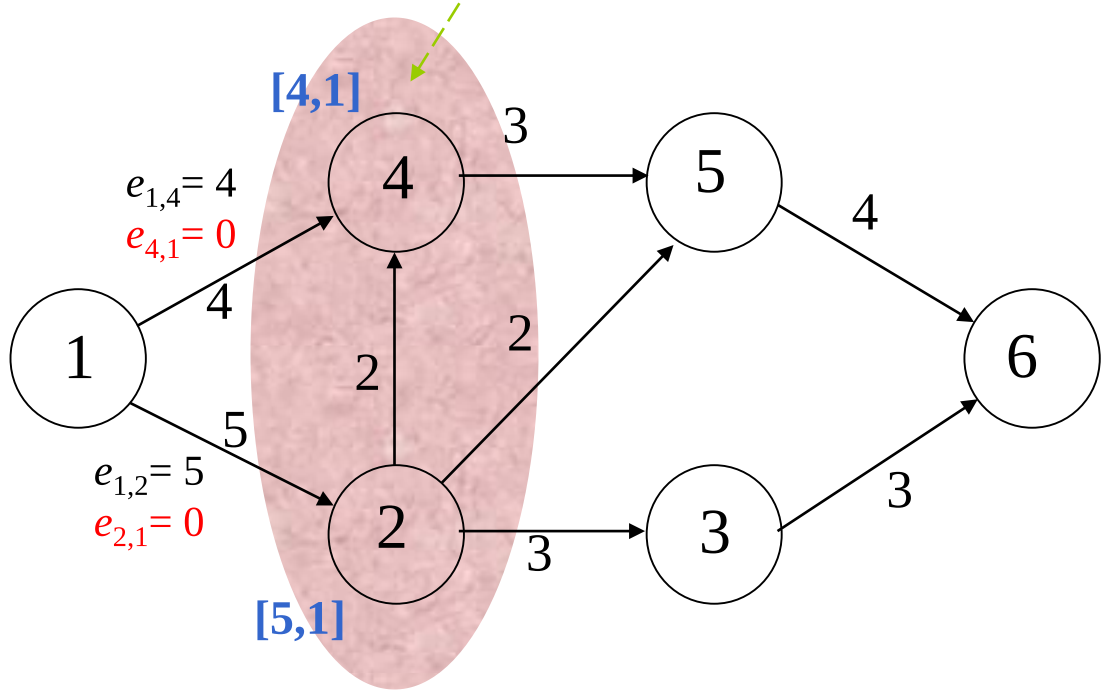
(2) Label nodes in $N_1$ as follows

$N_1$, all nodes connected to the source by an edge with positive excess capacity

Excess capacity of $(1,j)$

$[E_j, 1]$

Denoting node 1



# Labeling Algorithm (Ford & Fulkson)



Step 2: (1) Identify $N_2(j)$, based on the node $j$, with the smallest number, in $N_1$

(2) Label nodes in $N_2(j)$ as follows

min{$E_j$, Excess capacity of $(j,k)$}

$[E_k, j]$

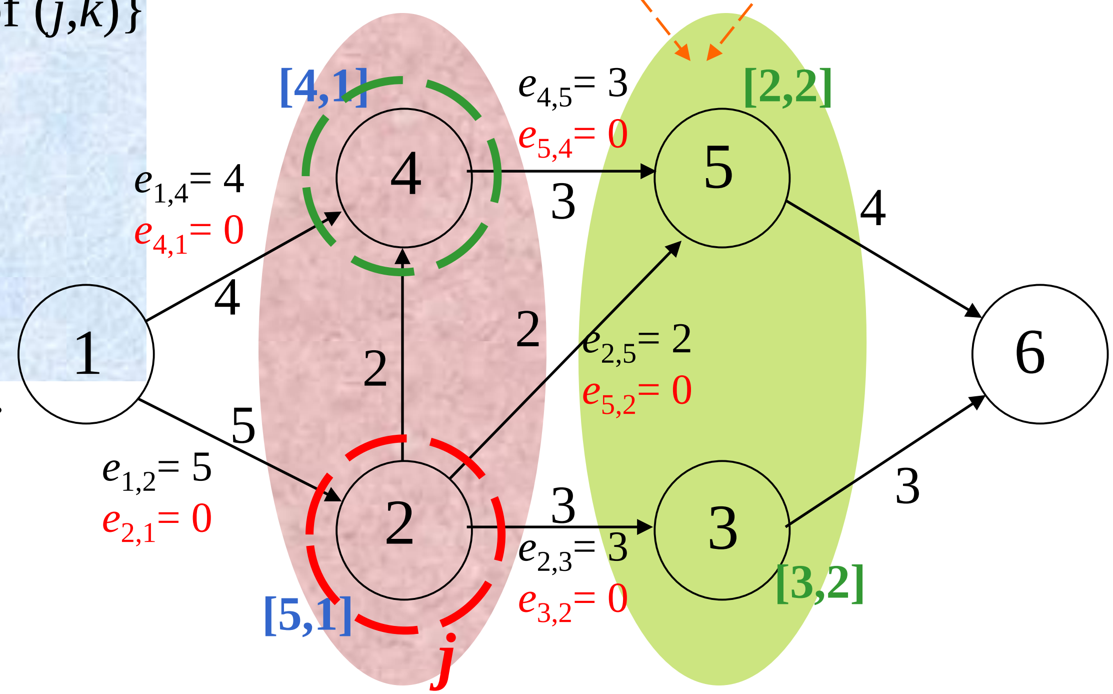
Denoting node $j$

(3) Do as above for all $j$ in $N_1$, and let

$$N_2 = \cup_{j \in N_1} N_2(j)$$

$N_2(j)$, all unlabelled nodes connected to node $j$ by an edge with positive excess capacity

Also $N_2$, in this case



# Labeling Algorithm (Ford & Fulkson)



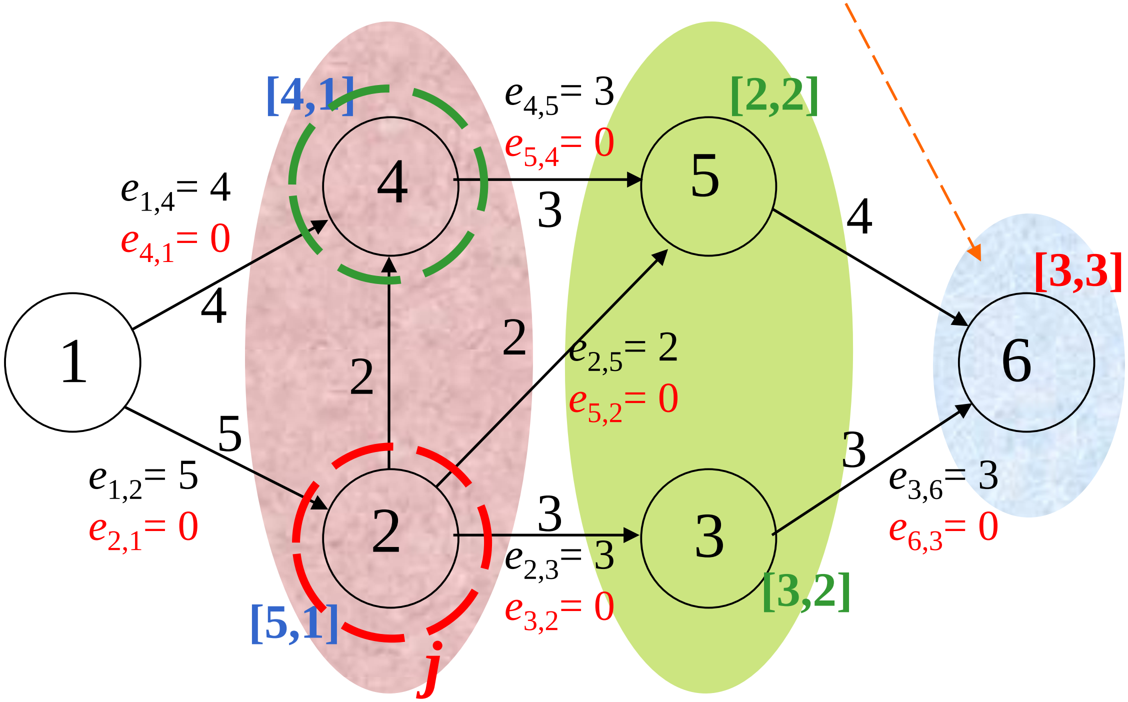
Step 3: Continue as in step 2, forming $N_3$, $N_4$, $N_5$, ..., until:

i. the sink has been labeled, (goto step 4)

ii. the sink has not been labeled, and no other nodes can be labeled according to the rules (algorithm ends, and the total flow is the maximum flow)

**(note: the source is not labeled)**

$N_3$, the sink has been labeled



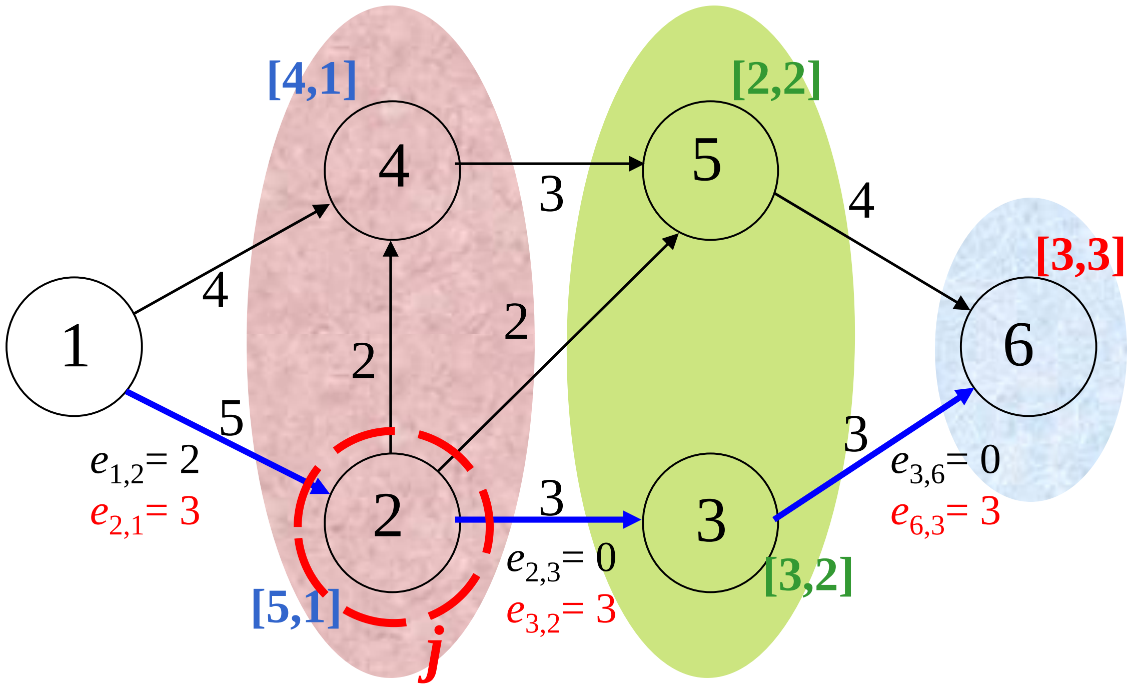
# Labeling Algorithm (Ford & Fulkson)



Step 4:

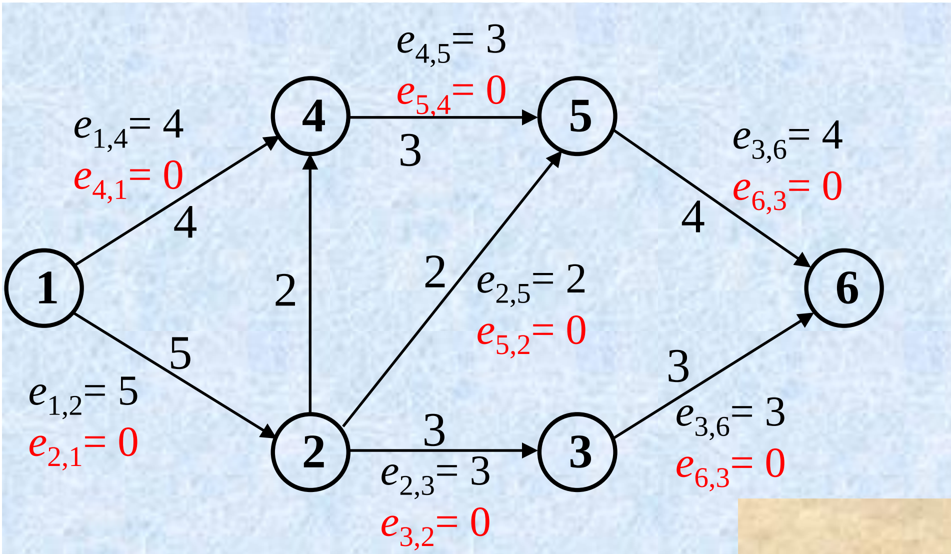
The label of sink is [$E_n$, $m$] (here, [3,3]), where $E_n$ is the amount of extra flow that can be made to reach the sink through a path π, and the path can be traced backward by node $m$

Update $e_{i,j}$, $e_{j,i}$ accordingly, **and then return to step 1**



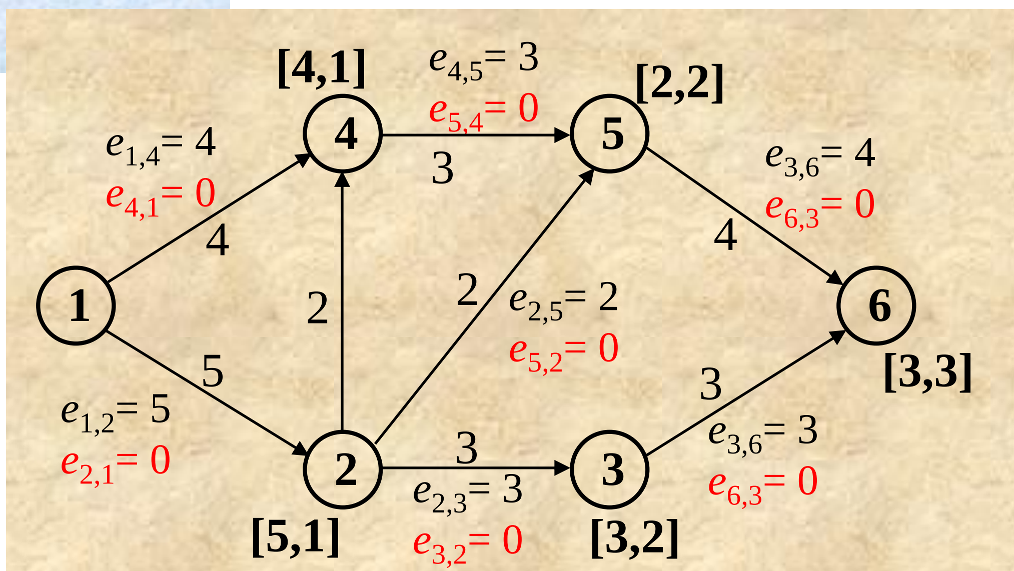
# 算法示例





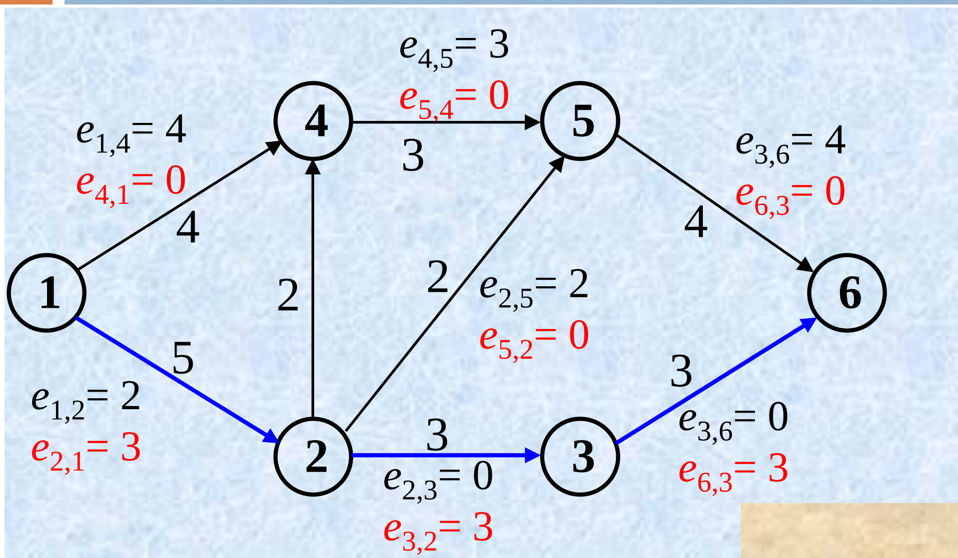

After the first cycle

At the beginning,
setting all flow to 0



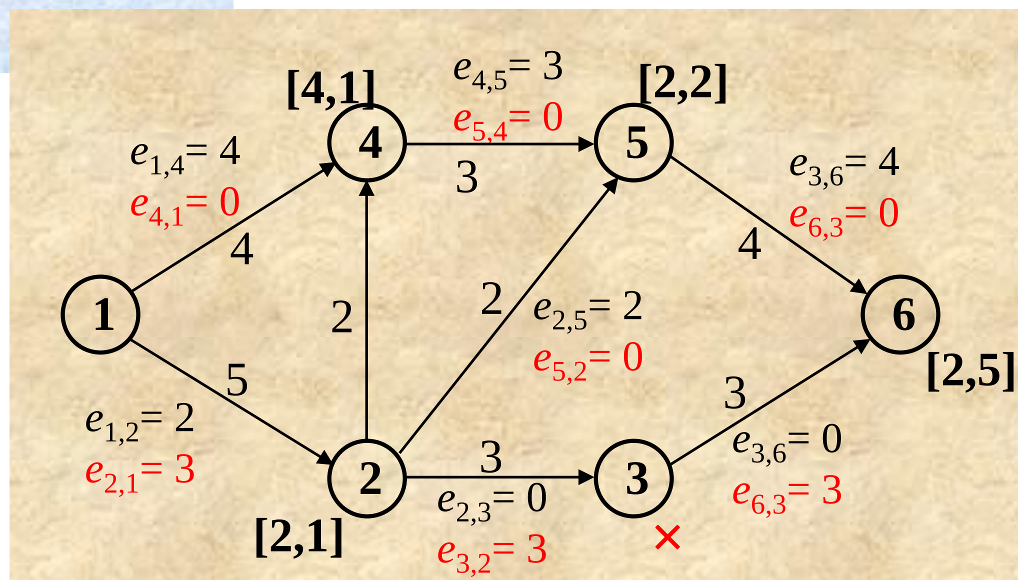
# 算法示例





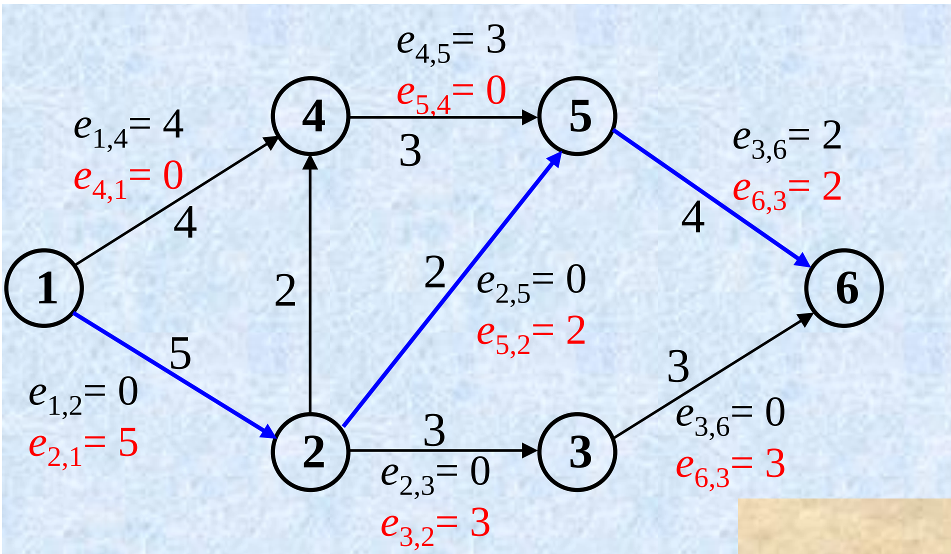

After the first cycle

After the second cycle



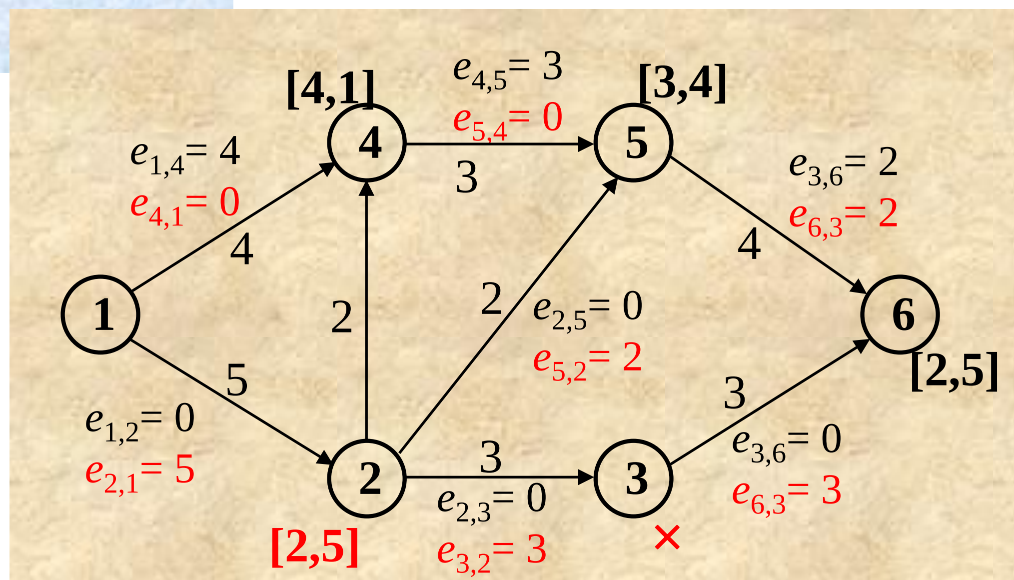
# 算法示例





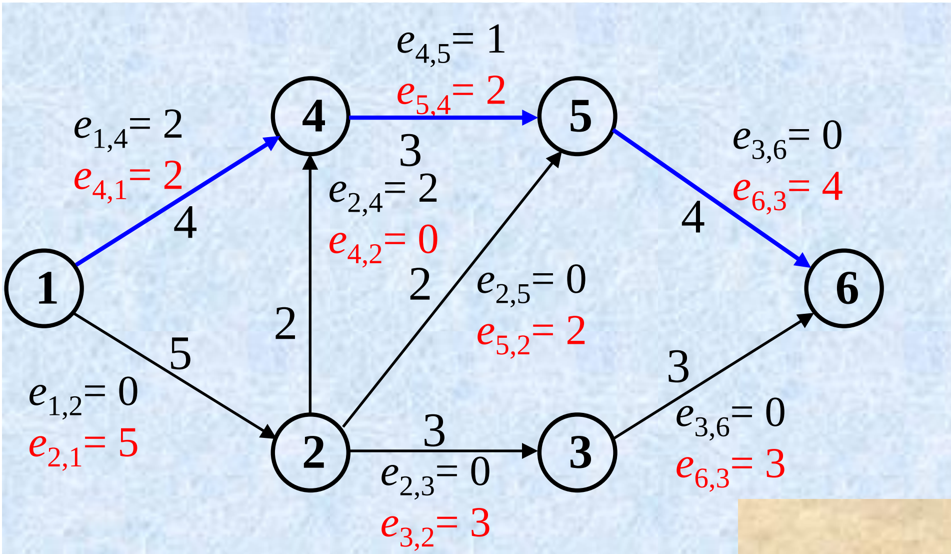

After the second cycle

After the third cycle



# 算法示例

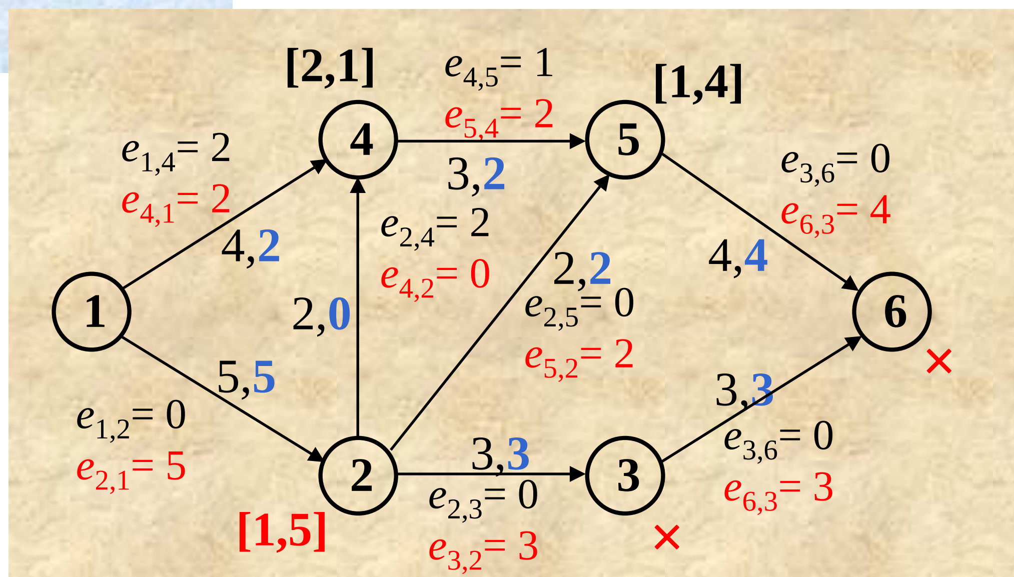





After the third cycle

After the fourth cycle

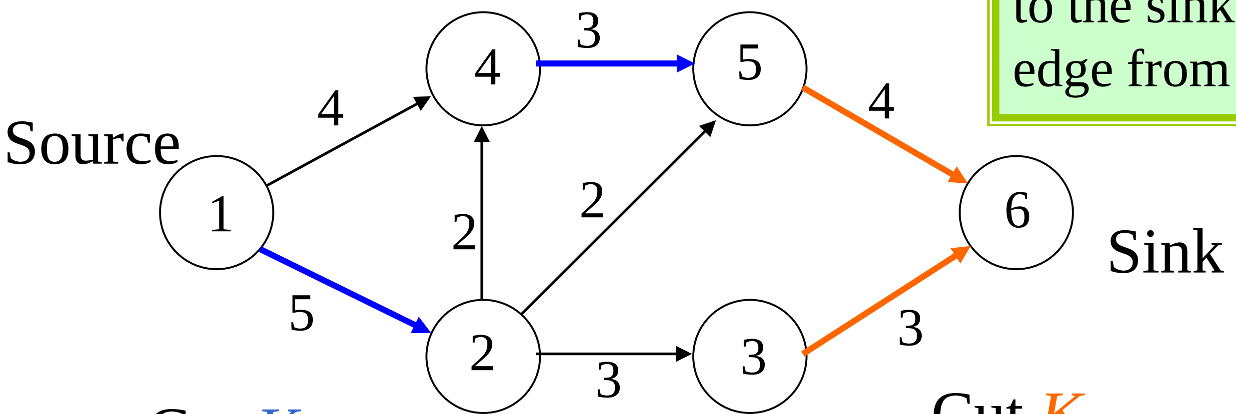
The sink has not been labeled, so the final result reached



# 流与割（Flow and Cut）



**Cut**: a set $K$ of edges in a network $N$, having the property that **every** path from the source to the sink contains **at least one** edge from $K$.




Cut $K_1$

$c(K_1)=8$

Cut $K_2$

$c(K_2)=7$

# Max Flow Min Cut Theorem

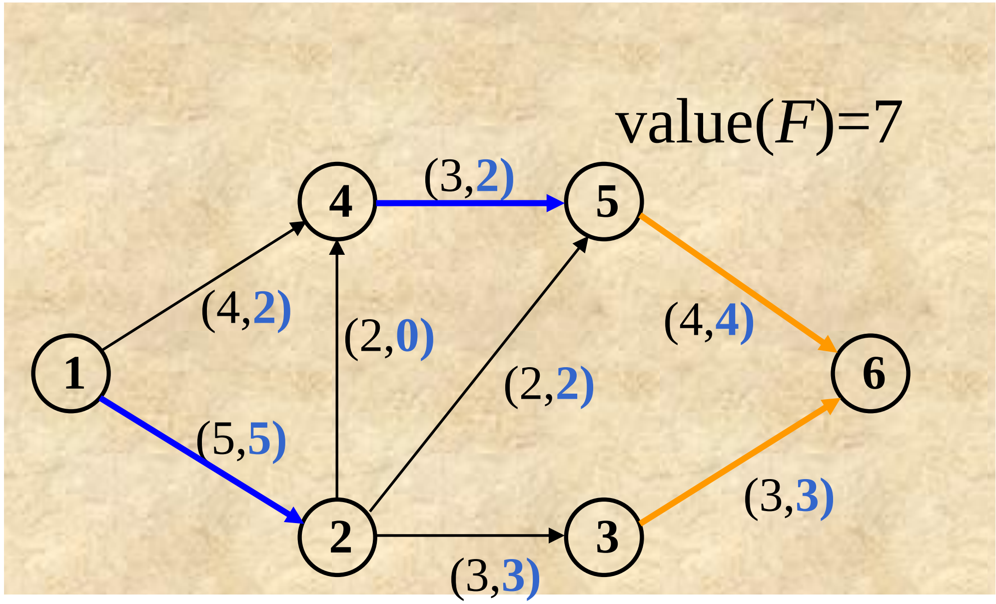


For any flow $F$, and any cut $K$, all parts of $F$ must pass through the edges of $K$. Since $c(K)$ is the maximum amount that can pass through the edges of $K$, so, value($F$)$\leq c(K)$.

If value($F$)$=c(K)$, then the flow uses the full capacity of all edges in $K$, $F$ must be a flow with maximum value, and, on the other hand, $K$ must be a cut with minimum capacity.

**Theorem**

A maximum flow $F$ in a network has value equal to the capacity of a minimum cut of the network



# 作业

- 见课程网站