算法设计与分析

Computer Algorithm Design & Analysis

吕志鹏

zhipeng.lv@hust.edu.cn

班级名称：算法设计与分析2025

QQ：419123077

微助教： QE854

Chapter 25
All-Pairs Shortest Paths

所有结点对的最短路径问题

一个新的最短路径问题：

给定一个带权重的**有向图**G=(V,E)，其权重函数为ω:E→R。在图中，对所有的结点对 u,v∈V，找出从结点u到结点v的最短路径。

该问题的解以表格（二维数组）的形式给出：第u行第v列给出从结点u到结点v的最短路径权重。

- **一条路径的权重是组成该路径的所有边的权重之和。**

本章约定：

1）结点编号：不失一般性，结点编号为1，2，…，|V|。

2）成本邻接矩阵：图G用一个n×n的邻接矩阵$W=(w_{ij})$表示，其中，

$$w_{ij}=\begin{cases}0 & \text{若 } i=j\\ \text{有向边}(i,j)\text{的权重} & \text{若 } i\neq j \text{ 且 } (i,j)\in E\\ \infty & \text{若 } i\neq j \text{ 且 } (i,j)\notin E\end{cases}$$

3）允许存在权重为负值的边，但不能包含权重为负值的环路。

- 否则无解。

4）最短路径矩阵：算法的输出为一个n×n的**最短路径矩阵** $\mathbf{D=(d_{ij})}$，其中d_{ij}表示从结点i到结点j的一条最短路径的权重。算法结束时有$d_{ij}=\delta(i,j)$。

5）前驱结点矩阵：

前驱结点矩阵记为：$\Pi=(\pi_{ij})$，其中

$$\pi_{ij}=\begin{cases} NIL, & \text{若i = j或者从i到j不存在路径} \\ \text{从结点i到结点j的某条最短路径上结点j的前驱结点，否则} \end{cases}$$

- 利用前驱结点矩阵Π可以计算出每对结点间的最短路径。

前驱结点矩阵Π的第i行所诱导(induce)的子图是一棵根结点为i的最短路径树。

对于每个结点i∈V，定义图G对于结点i的**前驱子图**为$G_{\pi,i}=(V_{\pi,i},E_{\pi,i})$，其中

$$V_{\pi,i}=\{j\in V:\pi_{ij}\neq \text{NIL}\}\cup\{i\}$$

$$E_{\pi,i}=\{(\pi_{ij},j):j\in V_{\pi,i}-\{i\}\}\ .$$

6）PRINT-ALL-PAIRS-SHORTEST-PATH过程：

如果$G_{\pi,i}$是一棵最短路径树，则**PRINT-ALL-PAIRS-SHORTEST-PATH**过程输出从结点i到结点j的一条最短路径。

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
1  if i == j
2       print i
3  elseif π_ij == NIL
4       print "no path from" i "to" j "exists"
5  else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
6       print j
```

25.1 最短路径和矩阵乘法

本节给出有向图所有结点对最短路径问题的一种**动态规划算法**。时间复杂度$\Theta(V^4)$，然后改进到$\Theta(V^3 \lg V)$。

- **最短路径的最优子结构性质：每条路径都是最短路径**

考虑从结点i到结点j的一条最短路径p。假定p至多包含m条边（假定没有权重为负值的环路），且m为有限值。

- 如果i=j，则p中不包含任何边，所以p的权重等于0；
- 如果$i \neq j$，则将路径p分解为 $i \overset{p'}{\rightsquigarrow} k \rightarrow j$ ，其中
  - p′至多包含m-1条边，根据引理24.1，p′是从i到k的一条最短路径，
  - 且$\delta(i,j)=\delta(i,k)+w_{kj}$。 ■

■ 递归解

设 $l_{ij}^{(m)}$ 是从结点i到结点j的至多包含m条边的任意路径中的最小权重。

- 当m=0时，表示从结点i到结点j**中间没有边**的最短路径，所以有

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \\ \infty & \text{if } i \neq j. \end{cases}$$

- 当m≥1时，可以在 $l_{ij}^{(m-1)}$ 的基础上计算从i到j的最多由m条边组成的任意路径的最小权重 $l_{ij}^{(m)}$ 。
  - $l_{ij}^{(m-1)}$ 是从i到j最多由m-1条边组成的最短路径的权重。

- $l_{ij}^{(m)}$的计算：通过对j的所有可能的前驱k进行检查获得。

$l_{ij}^{(m)}$的递归定义：

$$l_{ij}^{(m)} = \min\left(l_{ij}^{(m-1)}, \min_{1\le k\le n}\{l_{ik}^{(m-1)} + w_{kj}\}\right)$$

$$= \min_{1\le k\le n}\{l_{ik}^{(m-1)} + w_{kj}\} .$$

含有$l_{ij}^{(m-1)}$

- 结点间的最短路径权重δ(i,j)

如果图G不包含权重为负值的环路，则对于每一对结点i和j，如果δ(i,j)<∞，则从i到j之间存在一条最短路径。并且，由于最短路径是简单路径，其中至多包含n-1条边，因此有：

$$\delta(i, j) = l_{ij}^{(n-1)}$$

且：

$$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \ldots$$

自底向上计算最短路径权重

（1）$l_{ij}^{(1)}$ 表示从结点i到结点j中间至多包含1条边的路径的最小权重，因此 $l_{ij}^{(1)} = w_{ij}$ 。

（2）自底向上计算方法：

根据输入矩阵W=(w_{ij})，计算L(1)=($l_{ij}^{(1)}$)，然后根据递归关系式计算L(2)、…、L(n-1)。其中，

$$L^1 = (w_{ij}) = W$$

$$L^m = l_{ij}^{(m)}, \quad m=1,2, \ldots, n-1。$$

（3）L(n-1)即是最后的最短路径权重结果矩阵，有

$$l_{ij}^{(n-1)} = \delta(i, j)$$

自底向上计算算法的伪代码表示

EXTEND-SHORTEST-PATHS算法在给定W和$L^{(m-1)}$的情况下计算$L^{(m)}$：

```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          l'_ij = ∞
6          for k = 1 to n
7              l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```

- 算法将最近计算出的最短路径扩展一条边：
  - 找一条从i到某结点k，再经过边(k, j)到达j的更短路径。
- 算法结束时返回结果矩阵$L' = L^{(m)}$。
- 时间复杂度$O(n^3)$。

最短路径和矩阵乘法

上述一次计算最短路径结果矩阵$L^{(m)}$的过程和矩阵乘法在整体框架上是一致的：(详见P401)

```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          l'_ij = ∞
6          for k = 1 to n
7              l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

$L \bullet W \equiv A \bullet B$

时间复杂度都是$O(n^3)$

一次“L与W的乘L•W”完成一次从$L^{(m-1)}$向$L^{(m)}$的计算。

所以从上述L的计算过程以及与矩阵乘法的对比可以看出：

设A•B表示由算法EXTEND-SHORTEST-PATHS(A,B)返回的矩阵“乘积”，可以得到以下从$L^{(1)}$到$L^{(n-1)}$的计算序列：

$$
\begin{aligned}
L^{(1)} &= L^{(0)} \cdot W &&= W, \\
L^{(2)} &= L^{(1)} \cdot W &&= W^2, \\
L^{(3)} &= L^{(2)} \cdot W &&= W^3, \\
&\vdots \\
L^{(n-1)} &= L^{(n-2)} \cdot W &&= W^{n-1}.
\end{aligned}
$$

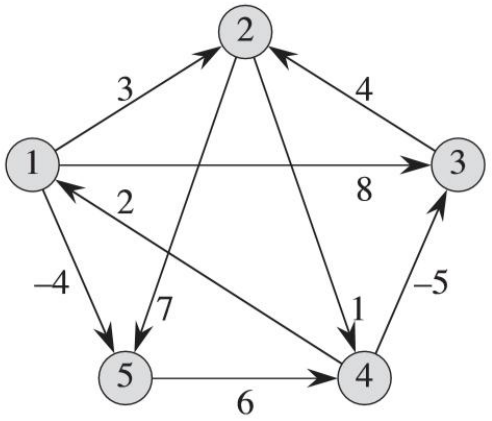
矩阵$L^{(n-1)}=W^{(n-1)}$将包含最短路径权重

计算矩阵序列的过程：

```
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)
1  n = W.rows
2  L^(1) = W
3  for m = 2 to n − 1
4       let L^(m) be a new n × n matrix
5       L^(m) = EXTEND-SHORTEST-PATHS(L^(m−1), W)
6  return L^(n−1)
```

时间复杂度：$O(n^4)$

例：已知有向图：




```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          l'_ij = ∞
6          for k = 1 to n
7              l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```

SLOW-ALL-PAIRS-SHORTEST-PATHS的计算过程：

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

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

矩阵序列：$L^{(1)} \rightarrow L^{(2)} \rightarrow L^{(3)} \rightarrow L^{(4)}$。而任何$m \geq 4$，$L^{(m)} = L^{(4)}$。

改进的算法：

我们的目标是算出矩阵$L^{(n-1)}$，而对所有$m \geq n-1$，$L^{(m)} = L^{(n-1)}$，因此可用以下$\lceil \lg(n-1) \rceil$个矩阵乘过程来计算矩阵$L^{(n-1)}$，而不是依次计算所有的$L^{(1)} \sim L^{(n-1)}$矩阵。

$$
\begin{aligned}
L^{(1)} &= W, \\
L^{(2)} &= W^2 &&= W \cdot W, \\
L^{(4)} &= W^4 &&= W^2 \cdot W^2 \\
L^{(8)} &= W^8 &&= W^4 \cdot W^4, \\
&\vdots \\
L^{(2^{\lceil \lg(n-1) \rceil})} &= W^{2^{\lceil \lg(n-1) \rceil}} &&= W^{2^{\lceil \lg(n-1) \rceil - 1}} \cdot W^{2^{\lceil \lg(n-1) \rceil - 1}}.
\end{aligned}
$$

注：由于 $2^{\lceil \lg(n-1) \rceil} \geq n-1$，所以最后的乘积 $L^{(2^{\lceil \lg(n-1) \rceil})}$ 等于$L^{(n-1)}$。

改进的计算过程如下：

下面的过程使用“重复平方技术”（repeated squaring）来计算上述矩阵序列：

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
1  n = W.rows
2  L^(1) = W
3  m = 1
4  while m < n − 1
5      let L^(2m) be a new n × n matrix
6      L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
7      m = 2m
8  return L^(m)
```

- 计算从m=1开始，每次迭代后对m加倍；
- 最后计算的$L^{(2m)}$即是$L^{(n-1)}$：$L^{(2m)}= L^{(n-1)}$，其中$n-1\le 2m\le 2n-2$，即第一次出现$n-1\le 2m$即终止。
- 算法的运行时间是：$O(n^3 \lg n)$。

25.2 Floyd-Warshall算法

本节讨论另一种动态规划策略来求解有向图的所有结点对最短路径问题——Floyd-Warshall算法。

- 算法的时间复杂度$\Theta(V^3)$。
- 算法允许图中存在负权重的边，但不能存在权重为负值的环路。

- **最短路径结构的重新描述：**

中间结点：一条简单路径$p=\langle v_1, v_2, \cdots, v_l\rangle$上的**中间结点**是指路径p上除$v_1$和$v_l$之外的其它任意结点。

假定图G的结点集为V={1,2,...,n}。考虑其中的一个子集{1,2,...,k}，这里k是小于n的某个整数，并是其中的最大编号。

对于任意一对结点i, j∈V，定义p是从i到j、且所有中间结点均取自于集合{1,2,...,k}的最短路径。

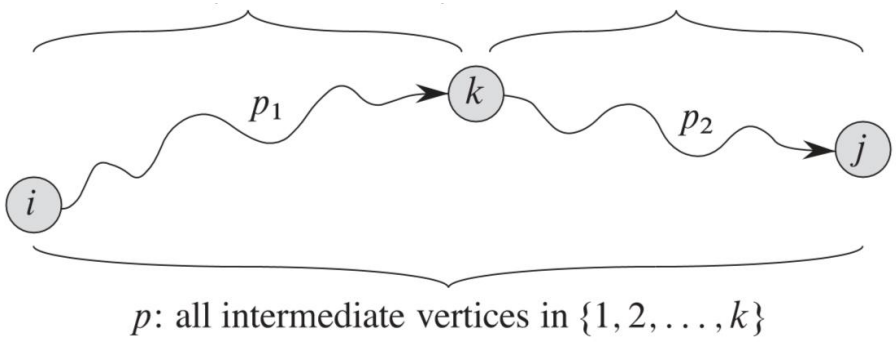
- p是简单路径，且p的中间结点都不大于k。
- p从i到j，仅经过集合{1, 2, …, k}中的结点，但，
  - 不一定经过其中的每一个结点；
  - 也可能不存在这样的路径，此时p的权重等于∞。

在从i到j之间中间结点均取自集合{1,2,...,k-1}的基础上，试图回答这样一个问题：结点k是否是路径p上的一个中间结点?

结点k是否是路径p上的一个中间结点?

- 情况1：如果**结点k不是路径p上的中间结点**，则p上的所有中间结点都属于集合$\{1, 2, \cdots, k-1\}$。
  - 此时，从结点i到结点j的中间结点取自集合{1,2,...,k-1}的一条最短路径也是从结点i到结点j的中间结点取自集合{1,2,...,k}的一条最短路径。

- 情况2：如果结点k是路径p上的中间结点，则k将路径p分解为两段：$i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$



  - 根据引理24.1（最优子结构性），p_1是从结点i到结点k的一条最短路径，且中间结点全部取自集合{1,2,...,k-1}。

    因为结点k不是路径p_1上的中间结点，所以路径p_1上的所有结点都属于集合$\{1, 2, \cdots, k-1\}$。

  - 同理，p_2是从结点k到结点j的一条最短路径，且中间结点全部取自集合{1,2,...,k-1}。

状态转移方程：

设 $d_{ij}^{(k)}$ 为从结点i到结点j的所有中间结点全部取自集合{1,2, ...,k}的一条最短路径的权重，则有：

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

其中，

- k=0时，代表从结点i到结点j的一条不包含编号大于0的中间结点的路径，这样的路径没有任何中间结点，最多只有一条边。所以 $d_{ij}^{(0)} = w_{ij}$ 。

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

- 而因为任何路径的中间结点都属于集合{1,2,…,n}，所以k=n时，$d_{ij}^{(n)}$ 给出所有可能的从结点i到结点j的中间结点均取自集合{1,2,...,n}的一条最短路径的权重，也就是从结点i到结点j的最短路径的权重。所以对所$i, j \in V$ 有：

$$d_{ij}^{(n)} = \delta(i, j)$$

- **矩阵** $D^{(n)} = \left(d_{ij}^{(n)}\right)$ **为结果矩阵。**

算法描述：

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

- $W_{n\times n}$：权重邻接矩阵；
- $D_{n\times n}$：最短路径权重矩阵
- FLOYD-WARSHALL自底向上地完成D矩阵的计算。
- D矩阵的计算可以在原址上完成。

构建最短路径

在计算矩阵$D^{(k)}$的同时，计算前驱矩阵Π序列：$\Pi^{(0)}$，$\Pi^{(1)}$，…，$\Pi^{(n)}=\Pi$。

其中，

- $\pi_{ij}^{(k)}$ 为从结点i到结点j的一条所有中间结点都取自集合{1, 2, …, k}的最短路径上**j的前驱结点。**
- k=0时：

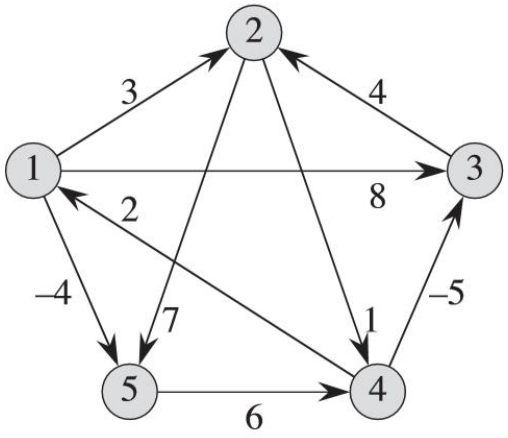
$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases}$$

  - k=0时，是一条从i到j的没有中间结点的最短路径，所以当路径存在时($w_{ij}<\infty$)，j的前驱就是i。

- k≥1时，从结点i到结点j的一条所有中间结点都取自集合{1,2, …, k} 的最短路径或者经过k，或者不经过k。
  - 若不经过k，则有 $d_{ij}^{(k-1)} \le d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$
    - 此时求从结点i到结点j的所有中间结点都取自集合{1, 2, …, k}的最短路径上的j的前驱等价于求从结点i到结点j的所有中间结点都取自集合{1, 2, …, k-1}的最短路径上的j的前驱。
  - 若经过k，则有 $d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$
    - 此时求从结点i到结点j的所有中间结点都取自集合{1, 2, …, k}的最短路径上的j的前驱等价于求从结点k到结点j的所有中间结点都取自集合{1, 2, …, k-1}的最短路径上的j的前驱。

所以有：
$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \le d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases},$$

例：对下例， FLOYD-WARSHALL算法的计算过程如下：



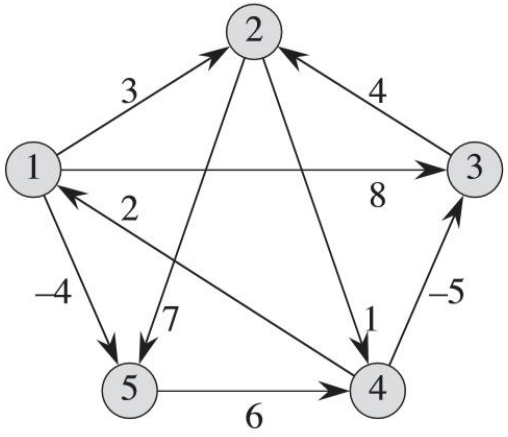

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
8  return D^(n)
```

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

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

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

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



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

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

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

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \le d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$

算法时间分析

- 3层嵌套的for循环，所以时间是：$\Theta(n^3)$。

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4       let D^(k) = (d_ij^(k)) be a new n × n matrix
5       for i = 1 to n
6            for j = 1 to n
7                 d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

思考：怎么在上述算法中加入计算Π的代码?

有向图的传递闭包

定有向图G=(V,E)，定义图G的传递闭包G*=(V,E*)，其中

E*={(i,j)：如果图G中包含一条从结点i到结点j的路径}。

求有向图的传递闭包：

方法一： 给E中每条边赋权重1，然后运行FLOYD-WARSHALL算法，可以在$\Theta(n^3)$求出权重路径矩阵D。在D中若$d_{ij}<n$，则表示存在一条从结点i到结点j的路径；否则$d_{ij}=\infty$。

方法二： 定义矩阵$T=\{t_{ij}\}$，若存在一条从结点i到结点j的路径，$t_{ij}=1$，否则$t_{ij}=0$。

计算T

对FLOYD-WARSHALL算法进行改造：用逻辑或操作（V）和逻辑与操作（Λ）替换算术操作min和+，得以下计算公式：

k=0时，
$$t_{ij}^{(0)} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i,j) \notin E\ , \\ 1 & \text{if } i = j \text{ or } (i,j) \in E\ , \end{cases}$$

k≥1时，
$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)\ .$$

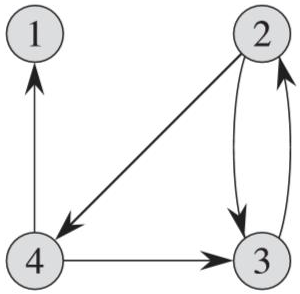
计算过程如下：

```
TRANSITIVE-CLOSURE(G)
 1  n = |G.V|
 2  let T^(0) = (t_ij^(0)) be a new n × n matrix
 3  for i = 1 to n
 4      for j = 1 to n
 5          if i == j or (i, j) ∈ G.E
 6              t_ij^(0) = 1
 7          else t_ij^(0) = 0
 8  for k = 1 to n
 9      let T^(k) = (t_ij^(k)) be a new n × n matrix
10      for i = 1 to n
11          for j = 1 to n
12              t_ij^(k) = t_ij^(k−1) ∨ (t_ik^(k−1) ∧ t_kj^(k−1))
13  return T^(n)
```

时间复杂度：$\Theta(n^3)$

注：逻辑运算比算术运算快；空间需求也较小

例，求如图所示的图G的传递闭包矩阵T。



$$T^{(0)}=\begin{pmatrix}1&0&0&0\\0&1&1&1\\0&1&1&0\\1&0&1&1\end{pmatrix}\quad T^{(1)}=\begin{pmatrix}1&0&0&0\\0&1&1&1\\0&1&1&0\\1&0&1&1\end{pmatrix}\quad T^{(2)}=\begin{pmatrix}1&0&0&0\\0&1&1&1\\0&1&1&1\\1&0&1&1\end{pmatrix}$$

$$T^{(3)}=\begin{pmatrix}1&0&0&0\\0&1&1&1\\0&1&1&1\\1&1&1&1\end{pmatrix}\quad T^{(4)}=\begin{pmatrix}1&0&0&0\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{pmatrix}$$

25.4 用于稀疏图的Johnson算法

对于有|V|个结点、|E|条边的有向图，求每对结点之间的最短路径：

- FLOYD-WARSHALL算法的时间复杂度：$\Theta(n^3)$
- 用单源最短路径算法，执行|V|次单源最短路径算法，每次使用一个不同的结点作为源点，求出每个结点到其他所有结点的最短路径。
- 如果所有的边的权重为非负值，用Dijkstra算法：
  - 用线性数组实现最小优先队列：$O(V^3+VE)=O(V^3)$；
  - 用二叉堆实现最小优先队列：$O(VE\lg V)$；（对稀疏图较好）
  - 用斐波那契堆实现最小优先队列：$O(V^2\lg V+VE)$；

- 用单源最短路径算法
  - 如果有权重为负值边，用Bellman-Ford算法：
    - 一般的运行时间：$O(V^2E)$；
    - 对稠密图，运行时间为$O(V^4)$。

Johnson算法：在稀疏图中求每对结点之间的最短路径权重。

- 对稀疏图，Johnson算法优于Floyd-Warshall算法，时间复杂度可达$O(V^2 \lg V + VE)$。
- Johnson算法使用Dijkstra算法和Bellman-Ford算法作为自己的子程序，可处理带有负权重的图。
- 如果图中包含所有结点对的最短路径，Johnson算法输出一个包含所有结点对的最短路径权重矩阵；否则报告图中包含权重为负值的环路。

重赋权重：Johnson算法使用**重新赋予权重**的技术求解。

工作原理：

- 如果图G=(V,E)中所有的边权重ω皆为非负值，则通过对每个结点运行一次Dijkstra算法来找到所有结点对之间的最短路径；
- 如果图G包含权重为负值的边，但没有权重为负值的环路，则通过**重赋权重**，构造出一组**新的非负权重值**，然后使用上面同样的方法求解(Dijkstra算法)。

  新赋予的权重函数记为：$\hat{w}$

  - $\hat{w}$ 必须满足以下两个重要性质：

新赋予的权重 $\hat{w}$ 必须满足以下两个重要性质：

1. **路径等价性**：对于所有结点对u, v∈V，一条路径p是在使用权重函数ω时的从结点u到结点v的一条最短路径，当且仅当p是在使用权重 $\hat{w}$ 时的从u到v的一条最短路径。
   - 即不管是使用原来的权重函数还是新的权重函数，所能求出来的最短路径应是一致的。
2. **非负性**：对于所有的边(u, v)，新权重 $\hat{w}(u,v)$ 为非负值。
   - 即，需要经过技术处理，把负权重的边的权重改造成非负值。

下面的引理，使得我们可以对边的权重重新赋值，以满足上面的两个条件：

这里，

- 用δ表示从权重函数ω所导出的最短路径权重；
- 用 $\hat{\delta}$ 表示从权重函数 $\hat{w}$ 所导出的最短路径权重。

引理25.1 （重新赋予权重并不改变最短路径） 给定带权重的有向图G=(V,E)，其权重函数为ω：E→R，设h:V→R为任意函数，该函数将结点映射到实数上。对于每条边(u,v)∈E，定义

$$\hat{w}(u,v) = w(u,v) + h(u) - h(v) .$$

设p=<$v_0, v_1, \cdots, v_k$>为从结点v_0到结点v_k的任意一条路径，那么，p是在使用权重函数ω时从结点v_0到结点v_k的一条最短路径，当且仅当p是在使用权重函数$\hat{w}$时从结点v_0到结点v_k的一条最短路径，即：ω(p)=δ(v_0,v_k)当且仅当$\hat{w}(p) = \hat{\delta}(v_0, v_k)$。

而且，图G在使用权重函数ω时不包含权重为负值的环路，当且仅当p在使用权重函数$\hat{w}$也不包含权重为负值的环路。

证明：

（1）首先证明：$\hat{w}(p) = w(p) + h(v_0) - h(v_k)$

根据定义，对于边有：$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$.

所以有：

$$\begin{aligned}\hat{w}(p) &= \sum_{i=1}^{k} \hat{w}(v_{i-1}, v_i) \\ &= \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\ &= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \\ &= w(p) + h(v_0) - h(v_k) .\end{aligned}$$

因为$h(v_0)$和$h(v_k)$不依赖于任何具体路径，因此，如果从结点v_0到结点v_k的一条路径在使用权重函数ω时比另一条路径短，则其在使用权重函数$\hat{w}$时也比另一条短。因此$\omega(p) = \delta(v_0, v_k)$当且仅当 $\hat{w}(p) = \hat{\delta}(v_0, v_k)$

然后证明：

若G在使用权重函数ω时包含一个权重为负值的环路当且仅当p在使用权重函数 $\hat{w}$ 也包含一个权重为负值的环路。

考虑任意环路 $c=\langle v_0, v_1, \cdots, v_k \rangle$，其中 $v_0=v_k$。

因为 $\hat{w}(p) = w(p) + h(v_0) - h(v_k)$

所以有
$$\begin{aligned} \hat{w}(c) &= w(c) + h(v_0) - h(v_k) \\ &= w(c) , \end{aligned}$$

因此，环路c在使用权重函数ω时为负当且仅当在使用权重函数 $\hat{w}$ 时也为负值。 证毕。

剩下的问题就是到哪里去找能使 $\hat{w}$ 为非负的h函数映射

通过重新赋值来生成非负权重

对于图G构造一幅新图G′=(V′, E′)，其中

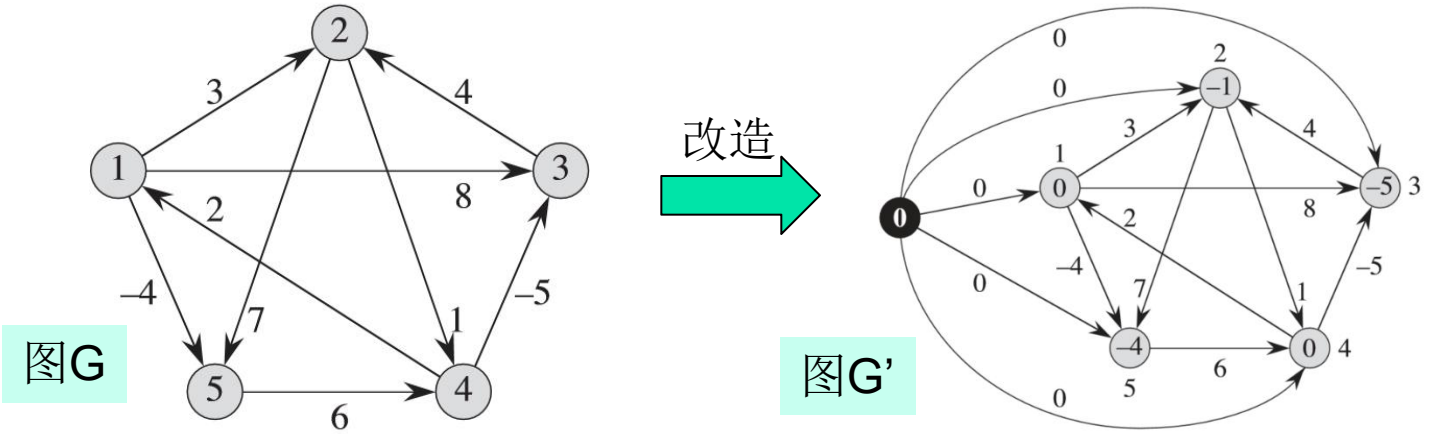
V′ =V∪{s}，s是一个新结点，$s \notin V$，

$E' = E \cup \{(s, v) : v \in V\}$ 。

并令，对于所有结点v∈V，有ω(s, v)=0。

注：由于结点s没有入边，所以除了以s为源点的最短路径外，图G′ 中没有其它包含s的最短路径。而且G'不包含权重为负值的环路当且仅当图G不包含权重为负值的环路。

如：



假定图G和图G’都不包含权重为负值的环路。

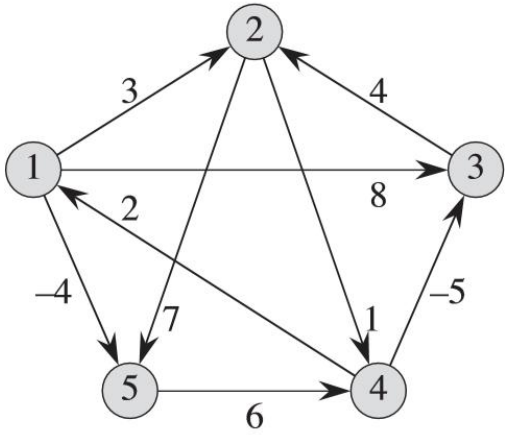
对于所有的结点v∈V’，定义：h(v)=δ(s,v)。

- 根据三角不等式，对于所有的边(u, v)∈E’，有

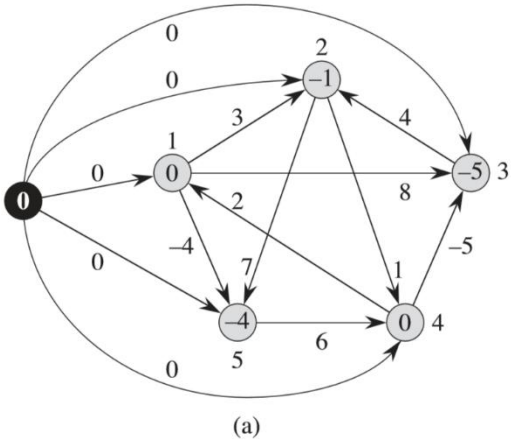
$$h(v) \le h(u) + \omega(u,v)。$$

定义新权重 $\hat{w}$，有 $\hat{w}(u,v) = w(u,v) + h(u) - h(v) \ge 0$

即满足性质二的要求（性质1已经证明成立）。

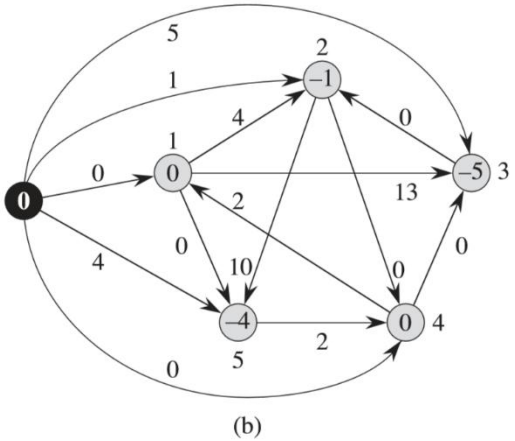


图G



(a)

结点里标记的是h(v)=δ(s,v)的值

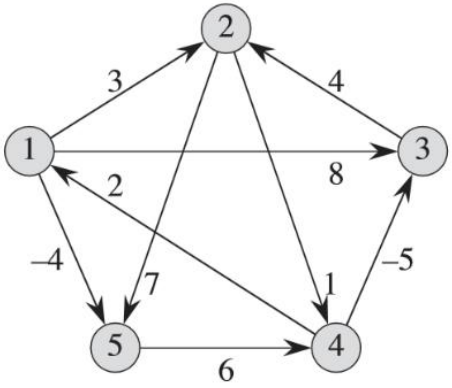


(b)

重新赋予权重的图G’

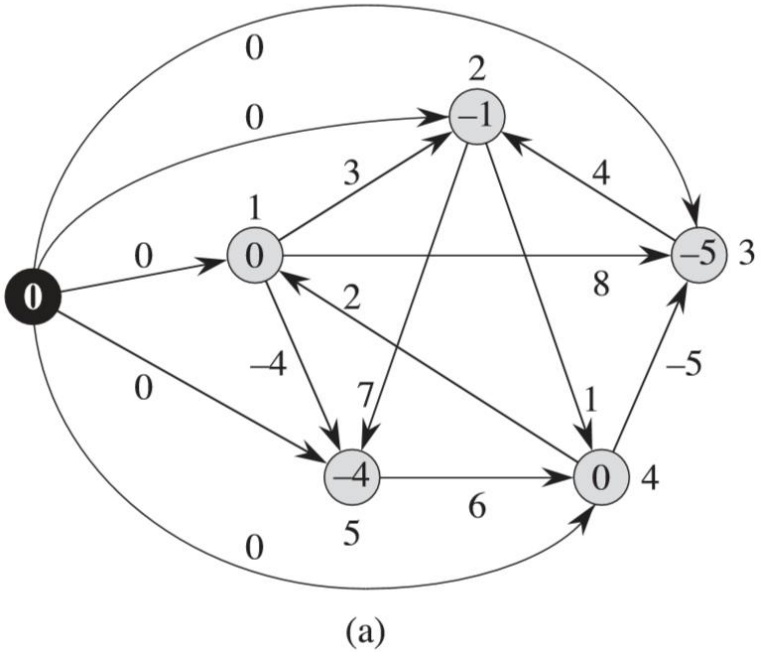
每条边的新权重函数为：

$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$

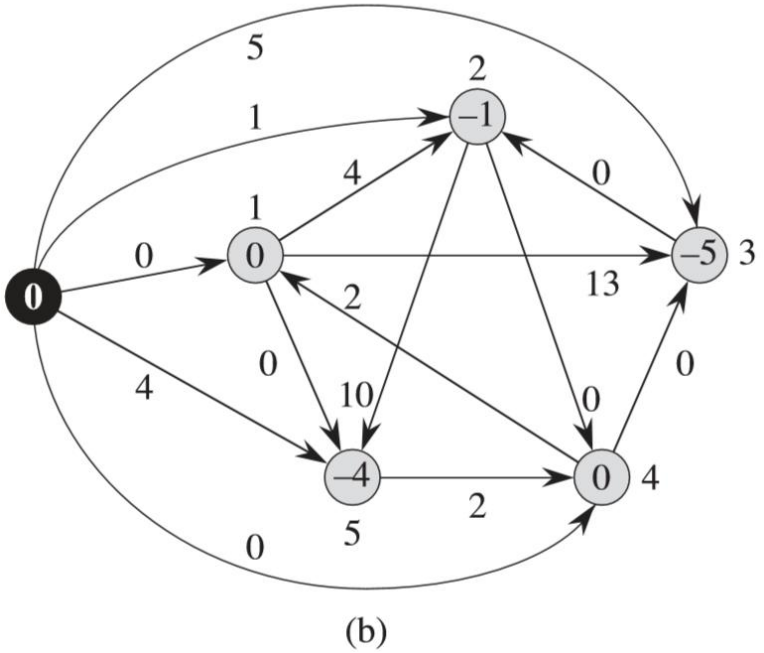


图G

$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$



(a)

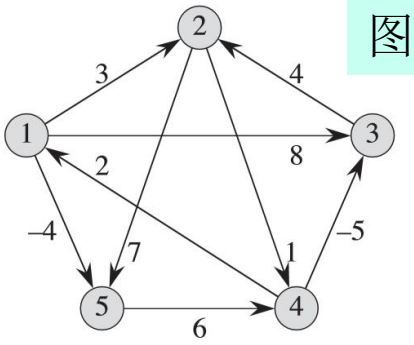


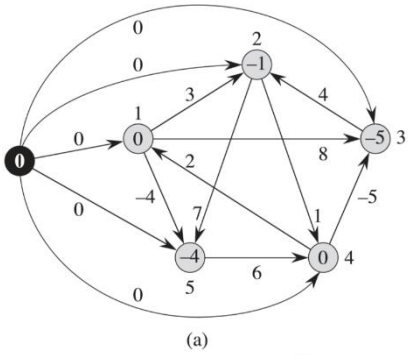
(b)

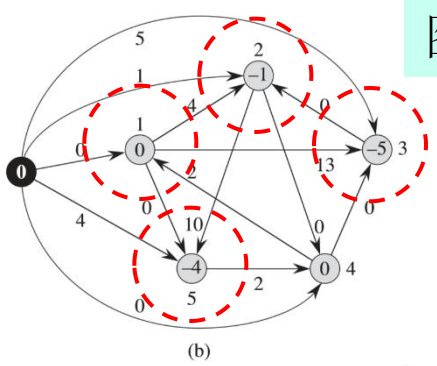
结点里标记的是h(v)=δ(s,v)的值

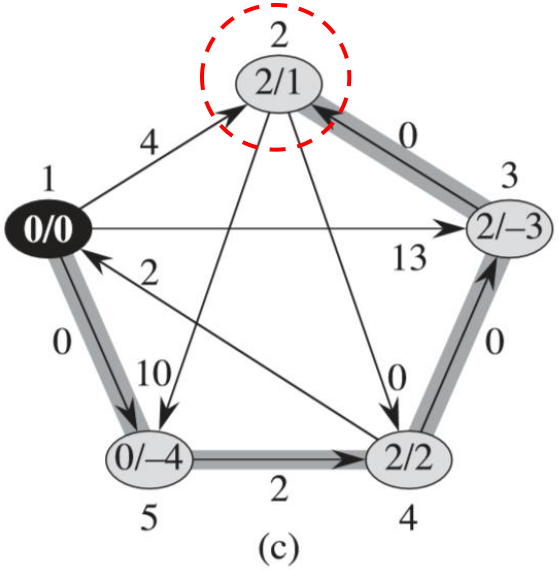
重新赋予权重的图G'
每条边的新权重函数为：

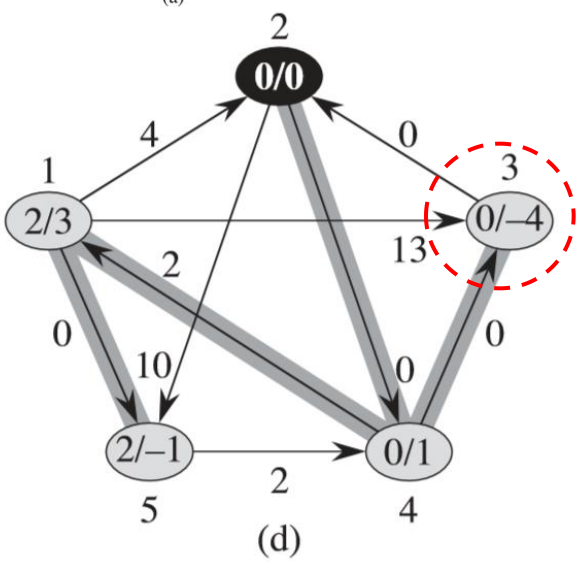
$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$

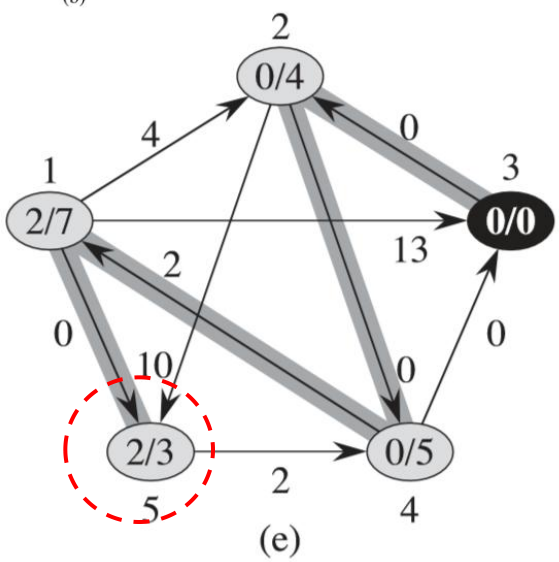












- (c)~(g)使用权重函数 $\hat{w}$ 在G的每个结点上运行Dijkstra算法。
- 黑色结点是当前源结点u。
- 加了阴影的边是由算法计算出来的属于最短路径树里面的边。
- 每个结点里面标记的是 $\hat{\delta}(u,v)$ / $\delta(u,v)$的值。还原：

$$d_{uv} = \delta(u,v) = \hat{\delta}(u,v) + h(v) - h(u)$$

如：

d_{12} =2+h(2)-h(1)
=2+(-1)-0 = 1

d_{23} =0+h(3)-h(2)
=0+(-5)-(-1) = -4

d_{35} =2+h(5)-h(3)
=2+(-4)-(-5)=3

Johnson算法计算所有结点对之间的最短路径

Johnson算法使用Bellman-Ford算法和Dijkstra算法作为子程序来计算所有结点之间的最短路径。

```
JOHNSON(G, w)
1  compute G', where G'.V = G.V ∪ {s},
       G'.E = G.E ∪ {(s, v) : v ∈ G.V}, and
       w(s, v) = 0 for all v ∈ G.V
2  if BELLMAN-FORD(G', w, s) == FALSE
3       print "the input graph contains a negative-weight cycle"
4  else for each vertex v ∈ G'.V
5            set h(v) to the value of δ(s, v)
                 computed by the Bellman-Ford algorithm
6       for each edge (u, v) ∈ G'.E
7            ŵ(u, v) = w(u, v) + h(u) − h(v)
8       let D = (d_uv) be a new n × n matrix
9       for each vertex u ∈ G.V
10           run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
11           for each vertex v ∈ G.V
12                d_uv = δ̂(u, v) + h(v) − h(u)
13      return D
```

先用Bellman-Ford算法计算s到各个结点的最短路径，并判断是否存在负权重的环路。

利用Bellman-Ford算法计算得到的δ(s,v)定义h(v)的值，并重新赋边的权重。

还原路径权重，得到原图的路径权重矩阵D。

```
JOHNSON(G, w)
1  compute G', where G'.V = G.V ∪ {s},
       G'.E = G.E ∪ {(s, v) : v ∈ G.V}, and
       w(s, v) = 0 for all v ∈ G.V
2  if BELLMAN-FORD(G', w, s) == FALSE
3       print "the input graph contains a negative-weight cycle"
4  else for each vertex v ∈ G'.V
5            set h(v) to the value of δ(s, v)
                 computed by the Bellman-Ford algorithm
6       for each edge (u, v) ∈ G'.E
7            ŵ(u, v) = w(u, v) + h(u) − h(v)
8       let D = (d_uv) be a new n × n matrix
9       for each vertex u ∈ G.V
10           run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
11           for each vertex v ∈ G.V
12                d_uv = δ̂(u, v) + h(v) − h(u)
13      return D
```

- Johnson算法假定所有的边都保存在邻接表里。返回一个|V|×|V|的矩阵D=d_{ij}，其中d_{ij}=δ(i,j)。
- 或者报告图G中包含权重为负值的环路。

时间分析：

算法的运行时间依赖于Dijkstra算法中最小优先队列的实现方式：

- 如果使用斐波那契堆实现，则Johnson算法的运行时间为$O(V^2 \lg V+VE)$。 $O(V\lg V+E)$
- 如果使用二叉最小堆实现，则Johnson算法的运行时间为$O(VE \lg V)$。 $O((V+E)\lg V)$
- 在稀疏图的情况下，该算法的时间比Floyd-Warshall算法的表现($O(V^3)$)要好。

22-25章作业

- 计算题：
  24.1-1
  24.4-1
  25.2-1

- 设计、证明题：
  24.1-3
  24-3
  25.2-7

- 思考题：25.2-6

课堂练习

1、（15分）带权有向图记为G=(V,E)，W是G的成本邻接矩阵。设计一个算法求G中所有有路径可达的结点之间相距最远的两个结点（两个结点之间的距离用这两个结点之间的最短路径长度进行度量）。

2、（15分）快递公司在城市的若干位置设有固定的业务点，由于物流的原因，每天的快件需要先运往总部，然后再从总部发往各业务点。现已知有n个业务点（用编号1~n表示，编号唯一、不重复）及各业务点在城里的位置和交通，并且对任意两个业务点之间直接可达（即不经过其他业务点）的交通时间进行了测量：如果从业务点i可以前往业务点j，tij表示从i到j的直接可达时间，$1 \le i,j \le n$。但由于交通和其他原因，tij未必等于tji，甚至可能从业务点i不能直接前往业务点j，反之亦然。

基于上述已知情况，快递公司现在考虑将总部设在哪个业务点才能够以最短的时间将快件全部送达至其它各业务点处（注：总部所在位置本身也是业务点之一，其送达时间可认为是0）。假设快递员人数足够，并且不计快递员在每个业务点的停留时间。请你设计一个算法帮助公司解决上述问题，即确定总部所在的位置。

要求：给出对该问题的一种分析，以便建立描述该问题的形式模型，然后给出相应的算法（用伪代码描述），并分析算法的时间复杂度。